ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ

Σχολή Χρηματοοικονομικής και Στατιστικής

Τμήμα Στατιστικής και Ασφαλιστικής Επιστήμης

ΜΕΤΑΠΤΥΧΙΑΚΟ ΠΡΟΓΡΑΜΜΑ ΣΠΟΥΔΩΝ
ΣΤΗΝ **ΕΦΑΡΜΟΣΜΕΝΗ ΣΤΑΤΙΣΤΙΚΗ**

ΑΝΑΠΤΥΞΗ ΒΙΟΔΕΙΚΤΩΝ ΓΙΑ ΤΗΝ ΕΞΑΤΟΜΙΚΕΥΣΗ ΘΕΡΑΠΕΙΩΝ ΜΕ ΧΡΗΣΗ ΤΗΣ ΑΝΑΛΥΤΙΚΗΣ ΤΩΝ ΔΕΔΟΜΕΝΩΝ ΚΑΙ ΜΗΧΑΝΙΚΗΣ ΜΑΘΗΣΗΣ

Ανέστης Κιοσσές

Διπλωματική Εργασία

που υποβλήθηκε στο Τμήμα Στατιστικής και Ασφαλιστικής Επιστήμης του Πανεπιστημίου Πειραιώς ως μέρος των απαιτήσεων για την απόκτηση του Μεταπτυχιακού Διπλώματος Ειδίκευσης στην *Εφαρμοσμένη Στατιστική*

Πειραιάς
Σεπτέμβριος 2022

ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ
Σχολή Χρηματοοικονομικής και Στατιστικής

Τμήμα Στατιστικής και Ασφαλιστικής Επιστήμης

ΜΕΤΑΠΤΥΧΙΑΚΟ ΠΡΟΓΡΑΜΜΑ ΣΠΟΥΔΩΝ
ΣΤΗΝ **ΕΦΑΡΜΟΣΜΕΝΗ ΣΤΑΤΙΣΤΙΚΗ**

ΑΝΑΠΤΥΞΗ ΒΙΟΔΕΙΚΤΩΝ ΓΙΑ ΤΗΝ ΕΞΑΤΟΜΙΚΕΥΣΗ ΘΕΡΑΠΕΙΩΝ ΜΕ ΧΡΗΣΗ ΤΗΣ ΑΝΑΛΥΤΙΚΗΣ ΤΩΝ ΔΕΔΟΜΕΝΩΝ ΚΑΙ ΜΗΧΑΝΙΚΗΣ ΜΑΘΗΣΗΣ

Ανέστης Κιοσσές

Διπλωματική Εργασία

που υποβλήθηκε στο Τμήμα Στατιστικής και Ασφαλιστικής Επιστήμης του Πανεπιστημίου Πειραιώς ως μέρος των απαιτήσεων για την απόκτηση του Μεταπτυχιακού Διπλώματος Ειδίκευσης στην *Εφαρμοσμένη Στατιστική*

Πειραιάς
Σεπτέμβριος 2022

UNIVERSITY OF PIRAEUS

School of Finance and Statistics

Department of Statistics and Insurance Science

POSTGRADUATE PROGRAM IN
APPLIED STATISTICS

BIOMARKERS DEVELOPMENT FOR PRECISION MEDICINE USING DATA ANALYTICS AND MACHINE LEARNING

By

Anestis Kiosses

MSc Dissertation

submitted to the Department of Statistics and Insurance Science of the University of Piraeus in partial fulfilment of the requirements for the degree of Master of Science in Applied Statistics

Piraeus, Greece
September 2022

Η παρούσα Διπλωματική Εργασία εγκρίθηκε ομόφωνα από την Τριμελή Εξεταστική Επιτροπή που ορίσθηκε από τη ΓΣΕΣ του Τμήματος Στατιστικής και Ασφαλιστικής Επιστήμης του Πανεπιστημίου Πειραιώς στην υπ' αριθμ. .... συνεδρίασή του σύμφωνα με τον Εσωτερικό Κανονισμό Λειτουργίας του Προγράμματος Μεταπτυχιακών Σπουδών στην Εφαρμοσμένη Στατιστική

Τα μέλη της Επιτροπής ήταν:
Σωτήριος Μπερσίμης, Αναπληρωτής Καθηγητής (Επιβλέπων)
Παντελής Μπάγκος, Καθηγητής
Κωσταντίνος Πολίτης, Αναπληρωτής Καθηγητής

Η έγκριση της Διπλωματική Εργασίας από το Τμήμα Στατιστικής και Ασφαλιστικής Επιστήμης του Πανεπιστημίου Πειραιώς δεν υποδηλώνει αποδοχή των γνωμών του συγγραφέα.

Έχω διαβάσει και κατανοήσει τους κανόνες του ΠΜΣ που περιέχονται στον Οδηγό Συγγραφής ΔΕ και ιδιαίτερα όσα συνιστούν λογοκλοπή. Δηλώνω ότι η παρούσα διπλωματική εργασία αποτελεί προϊόν αποκλειστικά δικής μου προσπάθειας, υπό την καθοδήγηση του επιβλέποντος καθηγητή, ενώ για όλες τις πηγές που χρησιμοποιήθηκαν περιλαμβάνονται οι αντίστοιχες αναφορές.

Περίληψη

Η εξατομίκευση των θεραπειών είναι καινοτόμα προσέγγιση που προσαρμόζει τη πρόληψη και τις θεραπείες λαμβάνοντας υπόψη διαφοροποιήσεις βιοδεικτών των ανθρώπων όπως επίσης και παράγοντες από το περιβάλλον και τον τρόπο ζωής. Οι βιοδείκτες είναι δείκτες που μπορούν να μετρηθούν, να αξιολογηθούν και να χρησιμοποιηθούν ώστε να προβλέψουν την εμφάνιση εξέλιξη μίας ασθένειας. Κλινικά επικυρωμένοι βιοδείκτες χρησιμοποιούνται για την εξατομίκευση των θεραπειών ώστε να διαχωρίσουν ασθενείς ως προς τον κίνδυνο ή την πρόγνωση να νοσήσουν ή την ανταπόκριση σε μία ασθένεια. Οι μέθοδοι μηχανικής μάθησης χρησιμοποιούνται ευρέως για την αναζήτηση βιοδεικτών και πολλοί από αυτούς όπως τα Δέντρα Αποφάσεων, τα Νευρωνικά Δίκτυα, τα Support Vector Machines και άλλα παρουσιάζονται παρακάτω. Επίσης, παρουσιάζονται αρκετές έρευνες στις οποίες αλγόριθμοι μηχανικής μάθησης έχουν εφαρμοστεί ώστε να αναζητήσουν πιθανούς βιοδείκτες για την εξατομίκευση θεραπειών. Τέλος πραγματοποιήθηκαν τρεις αναλύσεις σε τρία διαφορετικά σύνολα δεδομένων ώστε να εφαρμοστούν στην πράξη πολλές από τις μεθόδους που αναφέρονται και κάποιες ακόμα επιπρόσθετες διαδικασίες.

Abstract

Precision medicine is an innovative approach to tailoring disease prevention and treatment that takes into account differences in people's biomarkers such as genes, environments, and lifestyles. Biomarkers are indices which can be measured, evaluated and used for predictions about disease occurrence and progression. Clinically validated biomarkers are used for precision medicine to classify patients based on their disease risk, prognosis and/or response to treatment. Machine learning methods are popularly used for biomarker discovery and many of them such as Decision Trees, Support Vector Machines, Neural Networks and others are presented below. Moreover, presented many studies in which machine learning algorithms have been applied in order to discover potential biomarkers for precision medicine. Finally they were analyzed three data sets for building in practice some of the methods that are referred and others beyond them.

Contents

1 Introduction

1.1 Precision Medicine

Precision medicine is an innovative approach to tailoring disease prevention and treatment that takes into account differences in people's biomarkers such as genes, environments, and lifestyles. Patients with different biomarkers present with different risks of developing a disease, different disease prognoses or different responses to treatment. The goal of precision medicine is to target the right treatments to the right patients at the right time [1]. This approach stands in contrast to a decades-long effort in critical care to create broad definitions of syndromes.

Precision medicine does not literally mean the creation of drugs or medical devices that are unique to a patient. It is about the ability of using the available clinical, biological, and environmental large and complex data about patients to identify subgroups of patients that differ in their susceptibility to a particular disease, in the biology or prognosis of those diseases they may develop, or in their response to a specific treatment. Consequently, guide health care decisions toward the most effective treatment for a given patient, and thus, improve care quality, while reducing the need for unnecessary diagnostic testing and therapies [2].

The good use of large and complex data requires the development of advanced informatics tools including data mining, machine learning, and other aspects of artificial intelligence. Those tools identify and combine multiple predictors-biomarkers for early detection of a disease, prediction of the response to a treatment, estimate the probability of particular clinical outcomes or events either at the same time (diagnostic model) or in the future (prognostic model). Their application to health care may aid to the definition of dynamic patterns of health and disease as well as to create more efficient and sustainable models of care driven by data and technology [2].

A number of aspects have to be considered for model building, such as variable selection, modelling of continuous variables and interactions or restrictions by sample size. The best choice of model depends on the problem. Overall, no method is best, but all have different strengths and weaknesses. Once a stable and meaningful model has been derived, the result is an algorithm or a clinical prediction rule that forecasts a clinically relevant outcome such as disease onset or mortality and that is defined by the predicting variables and their respective weights in the algorithm. These models are always developed to fit the given data in an optimal way. Thus, their actual performance will be estimated too optimistically, and they need to be validated in independent samples.

Technological progress in precision medicine is expected to continue, and spearheaded. This innovation will likely change the way that healthcare services are organised and delivered: the creation of new molecular testing infrastructure and the development of 'learning' health information systems that analyse molecular and health record data to inform future prevention, detection or treatment strategies. Advances in precision medicine have already led to powerful new discoveries and FDA-approved treatments that are tailored to specific characteristics of individuals, such as a person's genetic makeup, or the genetic profile of an individual's tumor[1].

1.2 Machine Learning

An early definition of machine learning is: *Field of study that gives computers the ability to learn without being explicitly programmed* [3]. Machine learning algorithms build a model that can "learn" from available inputs, also called training data, to make predictions or decisions without being explicitly programmed to do so. The learning process is just an algorithm implementation repeatedly, making small adjustments until specific criteria are met [4].

In figure 1, presented 7 steps of machine learning algorithms development process. Data preparation is a large subject and is crucial for building a successful algorithm. It is the process of transforming the data to a form that can be applied machine learning algorithms. It includes data cleaning that is the removal or fixing of missing or incomplete data. Next, depending on the algorithm the data may be required transformations like standardization [5].

Model selection is also crucial task to avoid wasting time and processing cost. The choice of the appropriate algorithm is determined by the data, the problem, the complexity, the available time etc. Each machine learning algorithm has been designed to solve a specific problem. For example, for predicting continuous variables linear regression is a simple but probably efficient algorithm. For binary classification problems, Logistic Regression or Support Vector Machines are a good choice. For Multi-class classification, Random Forest or softmax regression probably would be a good choice. In general, linear algorithms are simpler and faster to train and they are a good choice if the linearity is met. Otherwise, a non-linear algorithm probably is more effective [6].

Also, the choice of model is based on complexity. In the case that high accuracy is required, then more features and parameters may be included in the model, and hence, longer training time. In this case a deep learning model probably is efficient option. If the data are insufficient, poor quality, unprocessed and there is not time for preprocessing before the training process then, unsupervised machine learning techniques are a better choice [6].

Each algorithm has one or more hyperparameters. Hyperparameters are tuned during the training process and have a significant impact in the final performance. There are many ways to tune the hyperparameters either trying by 'hand' one by one parameters or using automating methods like Bayesian optimization which is presented below.

The algorithm is evaluated based on a specific accuracy metric. It is important in this step, to use data that the algorithm has not seen in the training process, typically named as test set. Otherwise, there is a danger to overestimate its performance. A successful learner should have the ability to predict accurately unseen data. This is called generalization. The computational cost of making an algorithm should be low and the cost of using it even lower. This is important for machine learning, since querying the trained algorithm should be as fast as possible provided it happens more often and the result is often needed immediately. Machine learning is primarily of three types: supervised machine learning, unsupervised machine learning and reinforcement learning [4].

Supervised Machine Learning

In supervised machine learning the algorithm learns using training data that is already tagged with the correct label and must be able to predict reliably each possible case which is not given during training process [4]. Some tasks that supervised machine learning algorithms can

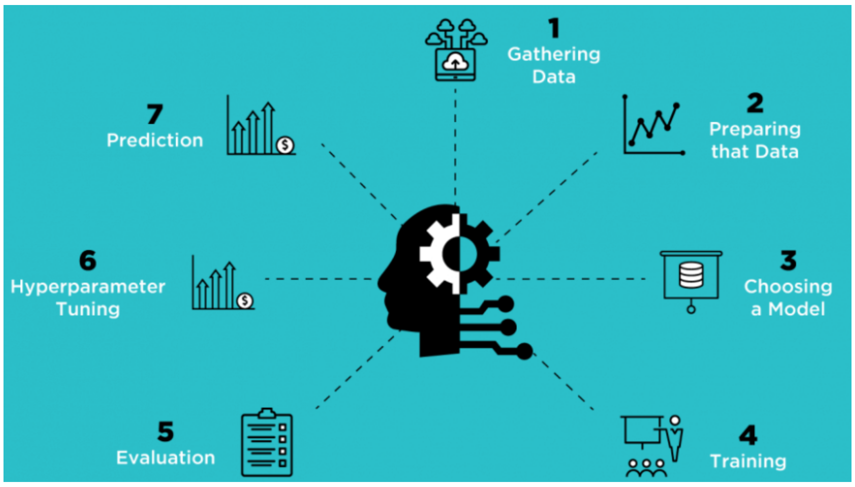


Figure 1: 7 steps of machine learning [7].

perform are the following.

Classification is used when the predicted data is categorical. Suppose that it is given a labeled data set having data vectors belong to N categories. The algorithm based on learning the training data set, tries to classify new inputs to one of N classes. The classification methods presented below vary in the ways that the learning process is done. However, most of them attempt to find decision boundaries that are used to classify data to one of the different classes. Many classification algorithms will be presented below [6].

Regression has to do with the case that the predicted data (represented by y) is a continuous variable. It is used to find the correlation between variables and make predictions for a dependent variable based on one or more independent variables [6].

Unsupervised Machine Learning

In unsupervised machine learning the training data is not labeled. The algorithm tries to identify hidden patterns or structures in data, so that inputs that are similar are classified in the same category. After the training process, the algorithm will be able to categorize a new data sample [4] . Some tasks that unsupervised machine learning algorithms can perform are the following.

Clustering is an unsupervised problem and is the task of grouping a collection of objects into groups or clusters such that objects within each cluster are more similar than objects assigned to different clusters [6].

Dimensionality Reduction is about processing and simplification of the data by reducing the number of features. The dimensionality reduction model reduces the features that are not crucial for the problem without significant loss of information and modifying the characteristics. Some positive outcomes of dimensionality reduction are the reduction of complexity, the training process and data visualization become faster and easier, less storage space is required, noise and not useful data are removed. Consequently, dimensionality reduction is a very useful method of machine learning [6].

1.3 Biomarkers

1.3.1 Biomarkers definition

Biomarkers are used in clinical practice for more than 130 years. Body temperature and blood pressure was first described as measures in animals in 1733 and introduced as traditional indicators of health by physicians [8]. The National Institute of Health Consortium in 2001 defined a biomarker as a “characteristic that is objectively measured and evaluated as an indicator of normal biological processes, pathogenic processes, or pharmacologic responses to a therapeutic intervention”. Biomarkers play an important role in the evaluation of disease as well as in the development of drug treatments [9].

1.3.2 Biomarker types

Biomarkers fall under two categories of measurement: quantitative or qualitative. Qualitative biomarkers determine the markers’ existance or not. While quantitative are involved in pathogenic process detection with a threshold effect [10]. For example, people with allergy issues tend to have increased T-helper type 2 (TH2) cells, however everyone has TH2 cells. So an idea of how many are present will help indicate the severity of issues with allergies or asthma [11].

There are many ways that biomarkers can be classified. More extensively they can be categorized as [12]:

- Susceptibility/risk biomarker: A biomarker that indicates the potential for developing a disease or medical condition or sensitivity to an exposure in an individual without clinically apparent disease or medical condition.
- Diagnostic biomarker: A biomarker used to identify individuals with the disease or condition of interest or to define a subset of the disease.
- Monitoring biomarker: A biomarker measured serially and used to detect a change in the degree or extent of disease. Monitoring biomarkers may also be used to indicate toxicity or assess safety, or to provide evidence of exposure, including exposures to medical products.
- Prognostic biomarker: A biomarker used to identify likelihood of a clinical event, disease recurrence or progression.
- Predictive biomarker: A biomarker used to identify individuals who are more likely than other patients, exhibiting similar clinical profiles but lacking the specific biomarker manifestation, to experience a favorable or unfavorable effect from a specific intervention or exposure.
- Pharmacodynamic/response biomarker: A biomarker used to show that a biological response has occurred in an individual who has received an intervention or exposure.
- Safety biomarker: A biomarker used to indicate the presence or extent of toxicity related to an intervention or exposure.

1.3.3 Biomarker characteristics

According to the US Food and Drug Administration (FDA), an ideal biomarker should be [13]:

- specific for a particular disease and able to differentiate between different physiological states.
- safe and easy to measure.
- rapid so as to enable faster diagnosis.
- cheap.
- able to give accurate results.
- consistent between different ethnic groups and genders.

A clinically useful biomarker should be able to provide meaningful information about prognosis and/or guide clinical decision-making and not simply duplicate information that is already available clinically. Derivation and validation to associate a biomarker to a disease process should also be carried out in different subsets of population. In general, biomarkers predicting disease risk perform much better in the derivation cohort compared to a validation cohort. Moreover, it is important to establish reference limits with the understanding that reference limits are influenced by the characteristic of an assay in the group analyzed to derive those limits. For instance, blood troponin assays made by several manufacturers are different and have varying reference limits for detection of clinically important vascular events [9]. Biomarkers can be measured in a variety of biological material (for example, blood, organ tissue, stool, saliva and urine) alone or in a group, often called a biomarker panel, to infer risk, diagnosis, prognosis and therapeutic response [14].

1.4 Omics

Rapid technical advances and opportunities for more cost-efficient large-scale data generation in biomedical research have created various new data sources for precision medicine, commonly referred to as "omics". A whole range of additional omics technologies has been developed, with 'omics' referring to the comprehensive study of the roles, relationships, and actions of various types of molecules in cells of an organism. This includes fields such as genomics, transcriptomics (the study of the expression of all genes in a cell or organism), proteomics (the analysis of all proteins), metabolomics (the comprehensive analysis of all small molecules) to name a few. It has also yielded a plethora of other 'omics' terms, such as epigenomics, lipidomics, metagenomics, glycomics, connectomics, cellomics, and even foodomics [15]. Therefore, the challenge is the development of methods for extracting useful information from these complex, multidimensional datasets to guide clinical practice.

1.4.1 Genomics

Genomic approaches, technologies, and data were the first widespread omics data available for application in precision medicine. Genomics is the study of all of a person's genes (the genome), including interactions of those genes with each other and with the person's environment. A gene traditionally refers to the unit of DNA that carries the instructions for making a specific protein or set of proteins [16]. Genome-wide association studies, next generation whole exome and whole genome sequencing data have provided a large set of DNA sequence variants that are associated with a plethora of diseases and traits in humans. In genomics, two approaches are explored: the search for unique genetic variants that are strong predictors of disease (e.g., single nucleotide variants associated with a Mendelian disorder or with inherited cancer risk, such as BRCA1 variants), or the composite predictor in the form of a genetic risk score that combines tens to hundreds of common genetic variants to provide population risk estimates without the need to identify individual causal sequence changes and their modes of action. Despite these rapidly expanding datasets, progress in identifying the specific disease causing sequence variants and the pathophysiological mechanisms by which they affect disease development or progression has been slow [17].

1.4.2 Trancriptomics

DNA is transcribed into RNA, which is then translated into proteins. The complete set of RNA transcripts including messenger RNAs (mRNAs), microRNAs (miRNAs), and different types of long noncoding RNAs (lncRNAs) produced by the genome, under a specific condition or in a specific cell, is called transcriptome. Transcriptomics is the study of the transcriptome through which researchers gain insight into gene activities and better understand how cells normally function or how changes in RNA activities can contribute to disease [18].

Transcriptomics has been characterized by technological innovations that continuously transform the field. Key contemporary transcriptomic technologies include DNA microarrays and NGS technologies called RNA-seq. Some important directions of transcriptomics studies are: identify biomarkers differently expressed between the diseased state and healthy state; distinguish disease stages or subtypes (e.g. cancer stages); establish the causative relationship between genetic variants and gene expression patterns to illuminate the etiology of diseases [19].

Normalization of transcriptomic data is an essential preprocessing step aimed at correcting unwanted biological effects and technical noises prior to any downstream analysis. Normalization methods shall be chosen according to the undertaken technology and can be platform-specific [20].

1.4.3 Proteomics

Proteomics technology is a promising tool for disease-associated biomarker detection in the biological fluids including urine, plasma, serum, etc. It is very important that body fluid samplings for proteomics research are less invasive and have low-cost advantages. Proteins are key players in cellular function and the proteomics technology is a powerful tool for the study of total expressed proteins in an organism or cell type at a particular time. As protein expression alters during disease condition in biological pathways, monitoring of these altered proteins in tissue, blood, urine, or other biological samples can provide indicators for the disease. The proteomics biomarker discovery is advanced in a variety of diseases such as cancer, cardiovascular diseases, acquired immune deficiency syndrome (AIDS), renal diseases, diabetes [21].

1.4.4 Metabolomics

Metabolomics, consists the analysis of small molecules present in a cell, tissue, or fluid. Metabolites are often viewed as the products of cellular processes, mediated by proteins; therefore, changes in metabolites are presumed to be reflective of changes in function of the mediating enzymes and proteins [15]. Metabolomics can be assessed in a targeted or unbiased manner, and mass spectrophotometry is used to identify chromatogram peaks as specific metabolites [22].

The vast majority of metabolomics analyses have focused on the analysis of plasma or serum samples from patients, often as an attempt to identify cancer or tumor-specific biomarkers that can be used in diagnosis without requiring an invasive tumor biopsy sample [15]. For example, Tissue, blood, and urinary metabolomics analyses have yielded putative biomarkers that classify patients into subtypes of lung cancer. However, these findings have yet to be sufficiently validated in more than one cohort [22].

1.4.5 Multi-omics

The Institute of Medicine's seminal report envisioned an Information Commons that contains multiple omics' approaches such that biomarker panels containing different molecule types, exposome data and/or demographic data could be developed to classify diseases into more precise subtypes. Integrating different types of data such as genes, proteins, RNAs and metabolites leads to more potential biomarker combinations and consequently, to the necessity for more functional studies and statistical tests. Two conceptual approaches to developing biomarker panels have been used. The first relies on adding new biomarkers to existing biomarkers or biomarker panels to improve the sensitivity or specificity of the panel because of either an interaction or an independent effect of the new biomarker. The second approach employs de novo analysis and integration of multiple sources of molecular data to identify the best combination of putative biomarkers [14].

2 Literature Review

2.1 Introduction

In this section presented many studies in which many machine learning algorithms have been applied to discover potential biomarkers for precision medicine. The studies are related to lung cancer, diabetes mellitus and cardiovascular diseases. Those diseases have been chosen due to their high mortality and prevalence.

2.2 Lung Cancer

Lung cancer (LC) is the leading cause of cancer death in both men and women and accounts for one in four cancer deaths leading to more than 1,000,000 deaths per year globally. The overall 5-year survival rate of LC is less than 20% mainly due to late diagnosis whereas patients with tumors diagnosed at early stages have 5-year survival rates of approximately 60% and 4% at the advanced stage. Despite advances in chemotherapy, radiation therapy and surgical management of lung cancer, the survival rate did not improve substantially. Therefore, early detection would be extremely important in decreasing the burden of lung cancer. Unfortunately, over 70% patients are diagnosed when their tumor are developed to the advanced stages due to the fact that LC symptoms occur late in the disease and the lack of reliable biomarkers at the early stage [23]. Therefore, it is important to find out new methods that would be less invasive and easier to conduct like powerful diagnostic biomarkers of lung cancer, particularly for the diagnosis of early lung tumor progression.

Lung cancer consists of two major histological types: Non-small-cell lung cancer (NSCLC) which is the most common type of LC and small-cell lung cancer (SCLC). Non–small cell lung cancer (NSCLC) which mainly consist of lung adenocarcinoma (LUAD, 44%),lung squamous cell carcinoma (LUSC, 26%) and large cell carcinoma (LCC) are the most common type of lung cancer accounting for 85% of all cases [24]. Small cell lung cancer (SCLC) remains one of the most lethal malignancies and a major health riddle having 5-year survival rate is less than 7% [25].

2.2.1 Lung Cancer Studies

The first study which is presented took place in China [23]. The researchers attempted to identify plasma metabolites that could act as promising biomarkers for distinguishing lung tumor patients with healthy individuals, disease stages as well as squamous carcinoma and adenocarcinoma patients with high sensitivity and specificity. Metabolomics and machine learning methods were combined, to detect early lung cancer diagnostic biomarkers. A total of 110 patients and 43 healthy individuals of the Hubei Taihe Hospital were included in this study. Patients were classified as stage I (n=54), stage II (n=31), stage III (n=25). Levels of 61 plasma metabolites were measured from targeted metabolomic study.

The Fast Correlation-Based Filter (FCBF) algorithm was applied to rank metabolites according to their ability to discern the classification label of an object [23]. FCBF (Fast Correlation Based Filter) is a multivariate feature selection method. It starts with full set of features and using a normalized information theoretic measure, called symmetrical uncertainty, computes dependences of variables and finds the best combination applying backward selection technique sequentially [26]. The top 8 metabolic biomarkers (table 1) that ranked using FCBF were used to develop the machine learning models. The AUC values changed with the number of variates are shown in table 2.

Six machine learning techniques of K-nearest neighbor (KNN), Naïve Bayes, AdaBoost, Support Vector Machine (SVM), Random Forest, and Neural Network with 10-cross fold technique were applied based on the metabolomic biomarkers features for the early lung tumor prediction based on the metabolomic biomarkers features. 80% of stage I lung tumor patients (43) and healthy individuals (35) were selected as the training set. The rest 20% samples were used for evaluation. As presented in table 3, in test set the specificity of Naïve Bayes, Neural Network, KNN, AdaBoost, and SVM was 1.000. The sensitivity of Naïve Bayes and Random Forest was 1.000, SVM and Neural Network was 0.909. Naïve Bayes concluded that was the best model with the highest level of sensitivity, specificity, and accuracy.

Moreover, through Mann–Whitney U test, 46 metabolic biomarkers showed statistically significant difference ($p - value < 0.05$) between lung cancer stage I patients and healthy individuals. Next these 46 influential metabolic biomarkers were applied to construct ROC curves. Based on the AUC (area under the ROC curve) value, sensitivity and specificity, top 10 metabolic biomarkers with higher diagnostic value ($AUC > 0.800$) are presented in table 4. The AUC of a combination of six variables (metabolites) based on logistic regression analysis was 0.989 (table 4). The metabolites used included proline, l-kynurenine, spermidine, aminohippuric acid, palmitoyl-L-carnitine and taurine with high AUC, sensitivity and specificity as shown in the table same table.

Additionally, they tried to discover metabolic biomarkers for discrimination of stage I, II, and III as well as for squamous carcinoma and adenocarcinoma patients. However, in both cases the biomarkers which found to be significant differentiated showed poor performanc [23].

Table 1: Ranked metabolomic biomarker features by FCBF [23].

Biomarker	**Score**
Taurine	0.655
Palmitoy-L-carnitine	0.482
proline	0.470
2-DG	0.446
PE(36:4)	0.353
Xanthine	0.303
PC(36:5)	0.290
Citric acid	0.280

Table 2: AUC value of machine learning models changing with the number of variates [23].

	1	2	3	4	5	6	7	8
KNN	0.947	0.959	0.964	0.971	1.000	0.988	0.988	1.000
SVM	0.925	0.927	0.961	0.995	1.000	1.000	1.000	1.000
Random Forest	0.945	0.952	0.960	0.968	0.996	1.000	0.998	0.997
Neural Network	0.895	0.917	0.944	0.944	0.999	0.991	0.991	0.998
Naïve Bayes	0.869	0.926	0.964	0.990	0.998	1.000	1.000	1.000
AdaBoost	0.833	0.859	0.859	0.887	0.887	0.899	0.899	0.913

Table 3: Machine learning models used for early lung tumor detection based on the metabolomic biomarker features [23].

		TP	FP	TN	FN	Accuracy	Sensitivity	Specificity	AUC	Precision
Training set	KNN	38	0	35	5	0.936	0.884	**1.000**	**1.000**	0.944
	SVM	43	2	33	0	0.974	**1.000**	0.943	**1.000**	0.975
	Random Forest	41	0	35	2	0.974	0.953	**1.000**	**1.000**	0.976
	Neural Network	43	0	35	0	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**
	Naïve Bayes	43	0	35	0	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**
	AdaBoost	38	4	31	5	0.885	0.884	0.886	0.885	0.885
Test set	KNN	9	0	8	2	0.895	0.818	**1.000**	**1.000**	0.916
	SVM	10	0	8	1	0.947	0.909	**1.000**	**1.000**	0.953
	Random Forest	11	2	6	0	0.895	**1.000**	0.750	**1.000**	0.911
	Neural Network	10	0	8	1	0.947	0.909	**1.000**	**1.000**	0.953
	Naïve Bayes	11	0	8	0	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**
	AdaBoost	4	0	8	7	0.632	0.364	**1.000**	0.682	0.804

In the next research, a novel computational approach was developed for recognizing important circulating miRNAs was that could be applied to early screening, diagnosis, and constant monitoring of lung cancer progression [27]. Expression profiles of 13 circulating miRNAs (table 5) in 48 patients with lung cancer and 984 control samples were downloaded from Gene Expression Omnibus. All the miRNAs were detected from serum samples from patients and controls.

Firstly, the minimum redundancy maximum relevance (mRMR) method was performed on the expression profiles of circulating miRNAs to evaluate their importance for lung cancer and ranked them as listed in Table 6.

MRMR (Minimum Redundancy Maximum Relevance Feature Selection) is a forward fea-

Table 4: ROC analysis of metabolomic biomarkers and combined variates [23].

	AUC	Std. error	Asymptotic 95% confidence interval		Optimal cut off	Sensitivity	Specificity	Youden index
			Lower bound	Upper bound				
ROC analysis of metabolomic biomarkers and combined variates in early lung tumor detection.								
L-Kynurenine	0.825	0.043	0.740	0.909	0.975	85.2%	72.1%	0.573
Proline	0.923	0.026	0.871	0.975	24.350	79.6%	93.0%	0.727
Spermidine	0.890	0.035	0.821	0.958	7.195	81.5%	90.7%	0.722
Amino-hippuric acid	0.811	0.045	0.722	0.900	4.035	68.5%	93.0%	0.615
Palmitoyl-l-carnitine	0.906	0.032	0.843	0.969	3.655	74.1%	100.0%	0.741
Taurine	0.920	0.032	0.856	0.983	71.300	88.9%	95.3%	0.842
TPhenylalanine	0.848	0.038	0.774	0.922	125.500	79.6%	76.7%	0.564
L-Valine	0.876	0.036	0.806	0.946	167.000	68.5%	95.3%	0.639
o-Tyr	0.822	0.043	0.738	0.906	24.650	83.3%	72.1%	0.554
Carnitine	0.848	0.040	0.769	0.926	4.680	72.2%	93.0%	0.652
Combination of two	0.933	0.028	0.878	0.978	0.337	85.2%	93.0%	0.782
Combination of three	0.968	0.019	0.931	1.000	−0.147	94.4%	97.7%	0.921
Combination of six	0.989	0.011	0.967	1.000	−0.102	98.1%	100.0%	0.981
ROC analysis of metabolomic biomarkers and combined variates of adenocarcinoma and squamous carcinoma patients.								
L-Kynurenine	0.423	0.060	0.306	0.540	1.050	77.8%	24.4%	0.022
Proline	0.580	0.057	0.469	0.692	35.150	54.0%	65.9%	0.198
Carnitine	0.536	0.058	0.422	0.650	6.835	38.1%	75.6%	0.137
Hypoxanthine	0.639	0.055	0.531	0.746	0.092	69.8%	56.1%	0.259
Hippuric acid	0.628	0.056	0.519	0.737	2.620	49.2%	77.5%	0.267
Combination of four	0.740	0.049	0.644	0.837	0.556	58.7%	78.0%	0.368

ture selection method which starts with zero variables and discovers the best combination of variables that is mutually and maximally dissimilar and can represent the target variable efficiently. The algorithm tries to find the set of variables that maximizes V_S which is the relevance of a set S and minimizes W_S which is the redundancy of a set S to the response variable. Redundancy and relevance are quantified using the mutual information of variables. V_S and W_S are calculated using equations 1 and 2. The mutual information between two variables say X and Z is calculated using equation 3 [28].

$$V_S = \frac{1}{|S|}\sum_{x\in S} I(x,y) \tag{1}$$

$$W_S = \frac{1}{|S|^2} \sum_{x,z \in S} I(x,z) \tag{2}$$

$$I(X,Z) = \sum_{i,j} P(X = x_i, Z = z_j) log \frac{P(X = x_i, Z = z_j)}{P(X = x_i)(Z = z_j)} \tag{3}$$

To evaluate those circulating miRNAs for distinguishing lung cancer samples from healthy samples, series of feature subsets were constructed from the above ranked feature list by mRMR: the first-ranked gene was used as the first feature subset; and then each following feature subset had one more feature; the next most important in the ranking. Then, for each feature subset, a Random Forest classifier was trained and evaluated the classification efficiency by 10-fold cross-validation. Synthetic Minority Oversampling Technique (SMOTE) was applied to decrease the influence of unbalanced data. The more features were using, the more the performance of RF classifiers was improving (figure 2). Using only the top 5 miRNAs there was a satisfactory Matthew Correlation Coefficient (MCC) value of 0.600 and AUC of 0.9865. When SMOTE was not performed, the performance of RF was lower due to the unbalanced data set.

Support vector machine (SVM) was also applied to be compared to RF doing the same procedures. The highest MCC was 0.636 when top 13 features were used, which was lower than that of the best RF classifier. Also, Sensitivity, Specificity, Accuracy, Precision, and F1-measure were used for evaluation of the model and they were 0.958, 0.943, 0.944, 0.451, and 0.613, respectively, which were all lower than those of the best RF classifier. These results suggest that RF was more proper than SVM to address the problem and that the top five miRNAs are potential biomarkers for lung cancer [27].

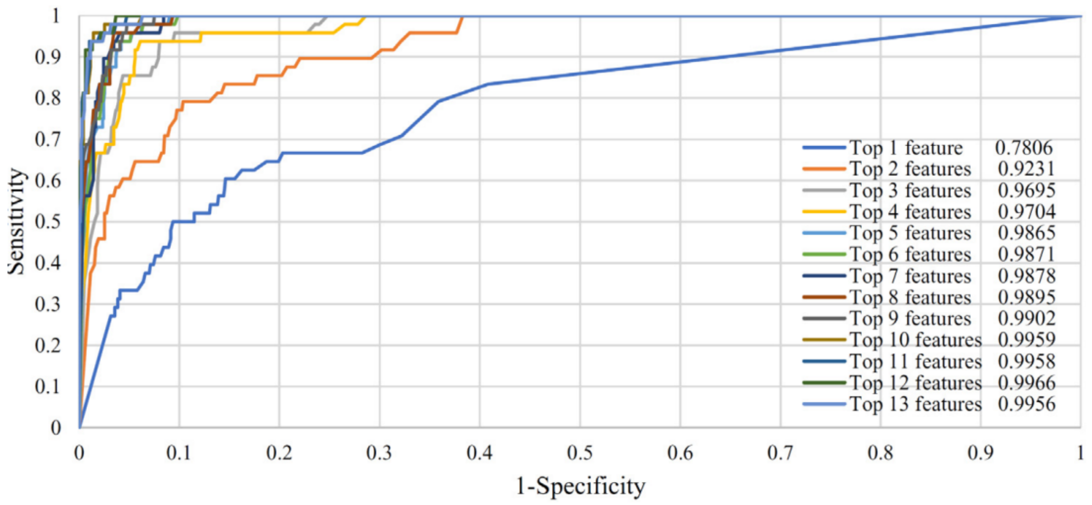


Figure 2: ROC curves of RF classifiers with different numbers of top features [27].

Table 5: High-ranked feature list according to mRMR [27].

Rank	Feature
1	hsa-mir-92a-000431
2	hsa-mir-140-5p-001187
3	hsa-mir-331-3p-000545
4	hsa-miR-374a-000563
5	hsa-mir-223-002295
6	hsa-mir-148a-000470
7	hsa-mir-484-001821
8	hsa-mir-328-000543
9	hsa-mir-191-002299
10	hsa-mir-30c-000419
11	hsa-mir-29a-002112
12	hsa-let-7d-002283
13	hsa-mir-30b-000602

In this work somatic mutation data and clinical data for 371 lung adenocarcinoma cases were used from The Cancer Genome Atlas in order to discover genetic markers that could be used for prognosis prediction and provide guidance for personal medicine [29].

The 3-year survival rate was close to 50%, hence the cases distributed in two different groups: good($> 3\ years$) and poor($< 3\ years$) prognosis. A newly developed genome-wide rate comparison method, EBT, was adopted to identify differentiated genes between the two groups. Only two genes, ADAMTS5 and PTPRC were found with significant mutation rate difference for $a = 5\%$, and 7,28,85 genes were found with significant mutation rate difference for $a = 10\%, 15\%, 20\%$ respectively.

Then, a Support Vector Machine with different kernels was applied to predict treatment outcomes taking as inputs the genetic variance features. The classification performance evaluated through 10−fold cross validation. The AUC of ROC curve of the SVM model with 85 genes ($EBT_0.20$) was significantly higher than that of 28 genes ($EBT_0.15$) while the latter model also outperformed the SVM with 7 genes ($EBT_0.10$) significantly (Table 7). The survival curves of cases within the predicted groups of the corresponding model were always differentiated significantly for prognosis, with a strikingly increase of the difference significance for $EBT_0.10$, $EBT_0.15$ to $EBT_0.20$ (Fig.3). The results together indicated an association between the prognosis of LUAD and somatic gene mutations, and the genetic variance could be useful for prognosis prediction [29].

The aim of this study was to discover the potential association between lung cancer and rou-

Table 6: Performance of RF for classifying lung cancer samples from healthy samples when using different numbers of features [27].

Top features	SN	SP	ACC	MCC	Precision	F1-measure
13	0.979	0.965	0.966	0.740	0.580	0.729
12	0.979	0.964	0.965	0.735	0.573	0.723
11	0.979	0.963	0.964	0.730	0.566	0.718
10	0.979	0.959	0.960	0.711	0.540	0.696
9	0.979	0.946	0.948	0.659	0.470	0.635
8	0.979	0.935	0.937	0.621	0.423	0.591
6	0.979	0.934	0.936	0.618	0.420	0.588
7	0.958	0.937	0.938	0.616	0.426	0.590
5	0.979	0.928	0.930	0.600	0.398	0.566
4	0.938	0.924	0.924	0.566	0.375	0.536
3	0.958	0.902	0.905	0.526	0.324	0.484
2	0.833	0.850	0.849	0.373	0.213	0.339
1	0.333	0.950	0.921	0.246	0.246	0.283

Table 7: Performance of different genetic models [29].

Model	**Features**	**ROC_AUC**	**Accuracy**	**Specificity**	**Sensitivity**
EBT_0.10	7	0.711±0.061	0.736±0.061	0.938±0.064	0.517±0.091
EBT_0.15	28	0.810±0.117	0.760 ± 0.098	0.907 ±0.084	0.600 ±0.160
EBT_0.20	**85**	**0.896± 0.055**	**0.800± 0.106**	**0.985± 0.034**	**0.600 ± 0.216**

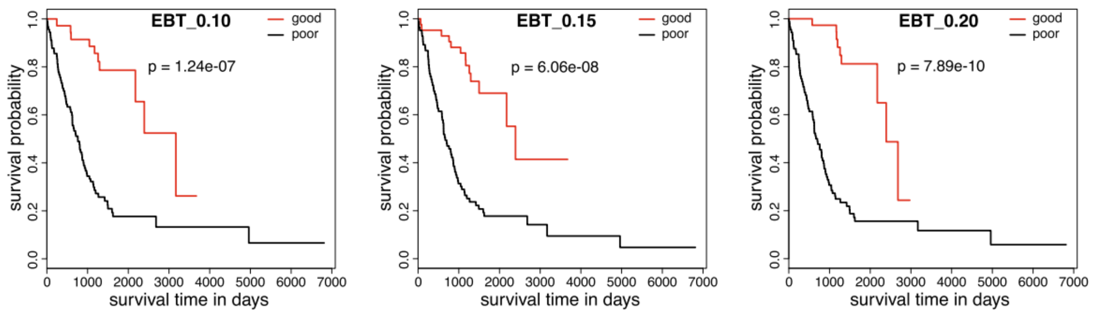


Figure 3: Survival curves of sub-groups of cases classified with different genetic models [29].

tine blood indices. Routine blood and biochemical test data were used which can be measured by common chemistry analyzers, with a cost of approximately 10 – 20 for each sample, to determine their correlation with lung cancer [30].

Data from routine blood tests were collected from the Second Hospital of Lanzhou University. A total of 277 patients with 49 types of routine blood indices were included, 183 of them was lung cancer patients (positive patients) and another 94 without lung cancer (negative patients). Among the 94 negative patients, 51 with tuberculosis were specifically included since there is a high false positive rate to distinguish lung cancer from tuberculosis. Then, the data were randomly split into a training set and a test set with a ratio of about 4 to 1. The training set included 149 lung cancer samples, 37 tuberculosis samples, and 36 other samples, and the remaining 55 samples were assigned to the test set.

The Random Forest method (RF) was applied at first using the entire set of indices on the basis of the 10−fold cross-validation. The 19 top-ranking indices, which are presented in table 8, with ntree and mtry values of 1300 and 9, respectively, were selected for the final model which had a similar prediction performance compared to the entire index space.

In the training set, Matthews correlation coefficient (MCC) of 91.36%, accuracy (ACC) of 95.7% and area under the curve (AUC) of 99.01% were attained (Fig 4.A). In the test set the sensitivity, specificity, and accuracy scores were all greater than 85% with values of 85.71%, 90%, 88.24%, respectively. The MCC value and AUC for the test set also got 75.71% and 90.16%, respectively (Fig 4.B). To validate the efficiency, reliability, and repeatability of the RBLC model, 34 serial blood samples from 15 additional patients were also included in the study. Overall, the sensitivity reached 92.31%, the specificity reached 80.95%, and the total accuracy reached 85.29%. This was the first time that a combination of routine blood biochemical indices was presented for its capability to well distinguish lung cancer, especially from tuberculosis [30].

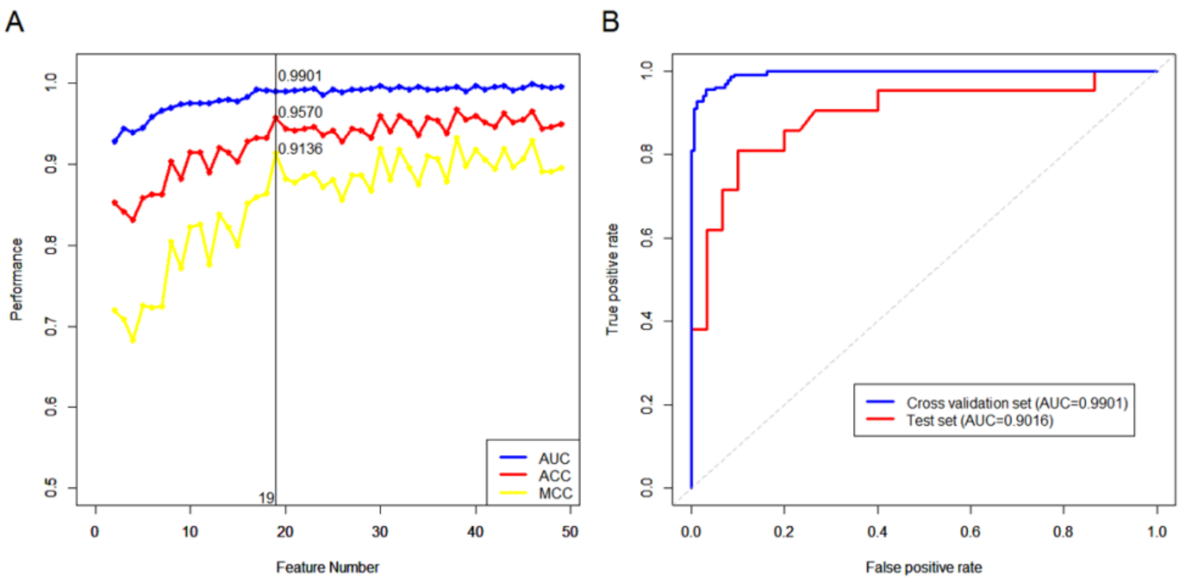


Figure 4: Classification performance of the RBLC model [30].

Table 8: Top-ranking blood indices for the identification of lung cancer [30].

Rank	Index	Reference range
1	Basophil ratio	0.00-0.01
2	Creatine kinase isoenzymes (U/L)	0.0-25.0
3	Platelet large cell ratio (%)	17.0-45.0
4	Albumin (g/L)	30.0-55.0
5	Platelet distribution width (fl)	9.0-17.0
6	Neutrophilic granulocytes (10^9/L)	2.00-7.00
7	White blood cell count (10^9/L)	4.00-10.00
8	Albumin/Globulin ratio	1.10-2.50
9	Monocytes (10^9/L)	0.12-1.20
10	Monocyte ratio	0.03-0.08
11	Lymphocyte ratio	0.20-0.40
12	Neutrophil granulocyte ratio	0.50-0.70
13	Lactate dehydrogenase (U/L)	0.0-240.0
14	Carbamide (mmol/L)	1.80-8.00
15	Eosinophil cells (10^9/L)	0.02-0.50
16	Mean corpuscular volume (fl)	80.0-100.0
17	Alkaline phosphatase (U/L)	0.0-120.0
18	Mean corpuscular hemoglobin (pg)	27.0-34.0
19	Creatine kinase (U/L)	0-195

In this work the researchers developed a novel approach to find highly sensitive and specific biomarkers by analyzing the miRNA profile of EVs isolated from the pleural fluids and lavages of 46 patients, including 25 control and 21 LC patients [31]. The workflow that was followed in the study is below in figure 5. The quality of the data included the removal of probes that had a Ct value of 40 in all samples, and the removal of samples in which more than 80% of the probes had a Ct value above 40. Finally, a total of 272 miRNA were kept for the differential expression analysis of 20 control and 14 LC patients.

The differential expression analysis between cancer and control cases yielded a list of 14 miRNAs that were significantly dysregulated. In order to evaluate whether differential expression translated into diagnostic power, a logistic model was performed based on the differentially expressed miRNAs. The best classifier was miRNA-1-3p, which showed an average accuracy of 0.941 ($95\%CI$: 0.803–0.993), sensitivity of 0.929, specificity of 0.950 and AUC value of 0.914 (Fig.6B). The next best classifiers, miRNA-144-5p and miRNA-150-5p, showed an average AUC values comparable with that of miRNA-1-30p with, however, significantly lower accuracy (0.882 and 0.912, respectively) and sensitivity (0.786 and 0.857) for the same specificity. Hence it was concluded that the use of EV-associated miRNA of pleural fluids and lavages are an untapped source of biomarkers, and specifically miRNA-1-3p, miRNA-144-5p and miRNA

150-5p are potential biomarkers for LC diagnosis. These biomarkers should be validated in an independent study including a larger cohort of patients and controls [31].

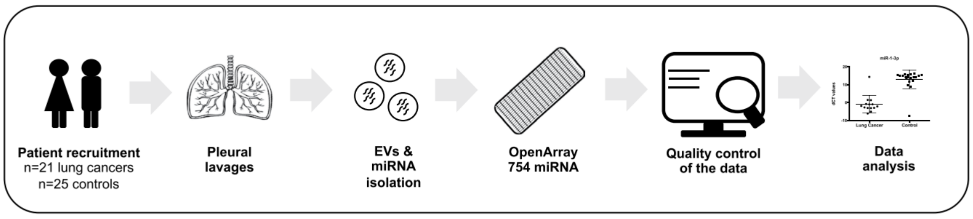


Figure 5: Workfow of the study design [31].

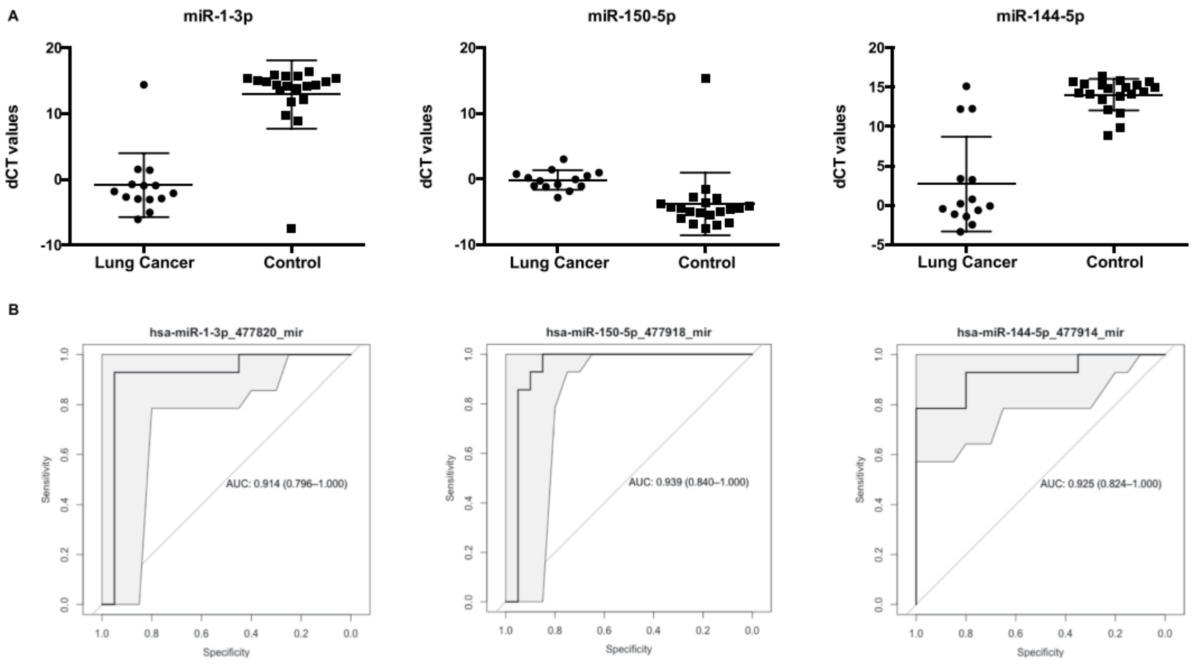


Figure 6: Diagnostic performance of the top diferentially expressed miRNAs. (A) Relative dCT values of top diferentially expressed miRNAs (miRNA-1-3p, miRNA-150-5p, and miRNA-144-5p) in patients with lung cancer (n=14) compared to control patients (n=20). **p¡0.05. (B) ROC-curves and AUC-scores for miRNA-1-3p, miRNA-150-5p, and miRNA-144-5p [31].

A predictive model was developed that can use noninvasive, reliable biomarkers for early lung cancer diagnosis [32]. Current clinically used tumor markers (alpha fetoprotein (AFP), carbohydrate antigen 19-9 (CA 19-9), carcinoma antigen 125 (CA 125), carcinoma antigen 15-3, (CA 15-3), and carcino-embryonic antigen (CEA)), lack sensitivity and specificity. Proteomic analyses of 231 human urine samples was performed. Among them there were: healthy individuals (CTL, $n = 33$), benign pulmonary diseases (with either pneumonia ($n = 23$) or COPD ($n = 17$), lung cancer ($n = 33$), bladder cancer ($n = 17$), cervical cancer ($n = 25$), colorectal cancer ($n = 22$), esophageal cancer ($n = 14$), and gastric cancer ($n = 47$) patients collected from multiple medical centers.

The training set that used to build the model consisted of healthy controls (CTL, $n = 23$) and lung cancer patients (LC, $n = 23$). Student's t-test resulted in 588 differentially expressed proteins between healthy controls and lung cancer patients at $p < 0.05$. Low abundant proteins were removed from the list and 68 candidates were chosen (Fig 7.), which were relatively abundant and detectable in < 70% of lung cancer urine specimens. Random forest with the 68 proteins was applied and the top 15 proteins ranked by variable importance presented an area under the ROC curve (AUC) of > 0.75 when they were combined.

Feature selection algorithm was implemented and 5 proteins were selected (FTL, MAPK1IP1L, FGB, RAB33B, and RAB15). With these 5 proteins, the predictive model could correctly separate most of lung cancer cases from the controls in the training set as sufficient as the random forest model using all 68 proteins. In test set, sensitivity and specificity were 90%. (Table 9)

Also, the predictive model was able to discriminate lung cancers from benign lung diseases as well as from other cancers with great sensitivity in all test sets (> 93%) and high specificity in three test sets (table below, CCA vs LC: 72%; EC vs LC: 85.71%; GC vs LC: 80.85%) (Table 9). In summary, the panel found in this study could be applied to clinical diagnostics of NSCLC for general purpose in the future after a validation trial with expanded sample numbers in a multi-center setting [32].

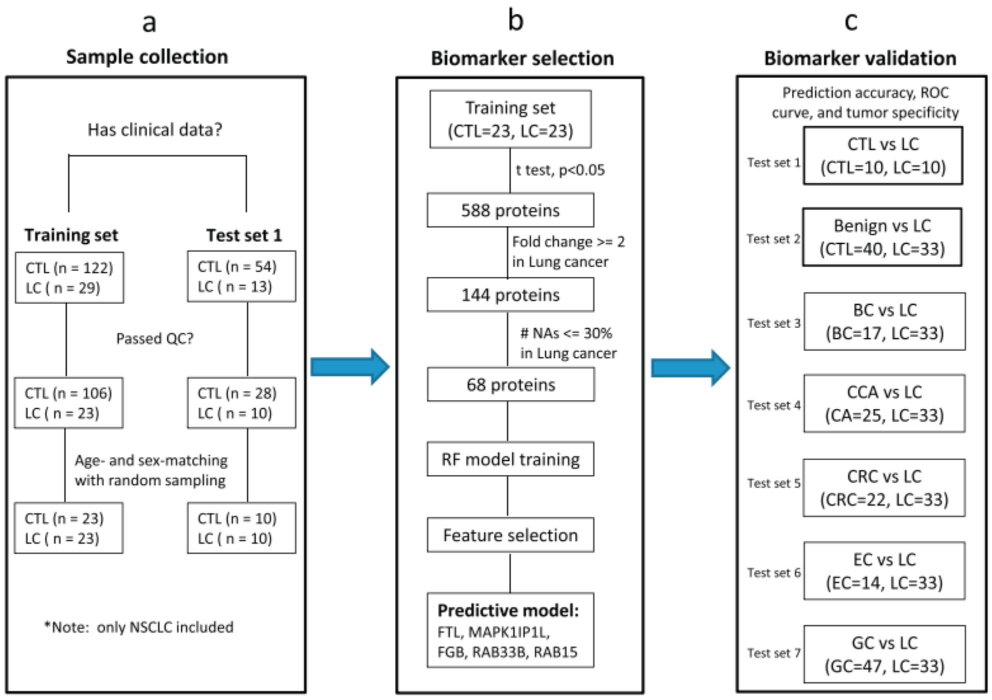


Figure 7: Flow diagram of lung cancer biomarker study [32].

A deep learning-based method for automated and robust detection and quantification of immune cell biomarkers in immunohistochemically stained tissue of human lung adenocarcinoma was created using as training data labeled images [33]. The training data was separated in two classes: positive class (T-cells) and negative class (other cells and artifacts).

Acquired data in total comprised 39 tissue slides. The dataset was separated in two parts: 27 slides for training and 12 slides for testing. 9 slides of each stain (CD3, CD8 and CD20) were used for training and 4 slides for testing the training progression.

The tissue areas, which contained stained immune cells, were manually annotated. From the positive annotations were extracted $1,500 \times 1,500$ px regions in which all the centers were manually annotated and from them, RGB patches of (46×46px) were extracted centered at the annotated point. These patches were used as a training data for the positive class (Fig 8.). They were also manually annotated tissue regions without positively stained cells. From these regions they directly randomly sampled 46×46 pixel RGB patches. These patches were used as examples for the negative class (Fig 8.).

For training 1,224,000 patches were from the 27 training slides (as an input for the convolutional network model) and from 12 testing slides 408,000 patches were used as a static validation set to monitor training progression. In total each class contained about 800 thousand patches.

The best performing neural network was comprised of six convolutional, two pooling layers and two fully connected layers (Fig.9). The network was trained using stochastic gradient descent with a learning rate of 0.01. The network training was stopped after one pass over all training patches as subsequent passes did not improve validation set results.

After subsequent training, the model obtained a validation set accuracy of 98.6% on the

Table 9: Random forest model in predicting lung cancer against controls and other cancers [32].

CTL vs LC				Benign vs LC			BC vs LC		
		Predicted			Predicted			Predicted	
	Group	CTL	LC	Group	Benign	LC	Group	BC	LC
Actual	CTL	9	1	Benign	28	12	BC	10	7
	LC	1	9	LC	1	32	LC	2	31
	Error	0.1		Error	0.178		Error	0.18	
	Sensitivity	90%		Sensitivity	96.97%		Sensitivity	93.94%	
	Specificity	90%		Specificity	70%		Specificity	58.82%	
CCA vs LC				CRC vs LC			EC vs LC		
		Predicted			Predicted			Predicted	
	Group	CCA	LC	Group	CRC	LC	Group	EC	LC
Actual	CCA	18	7	CRC	12	10	EC	12	2
	LC	1	32	LC	1	32	LC	1	32
	Error	0.138		Error	0.2		Error	0.064	
	Sensitivity	96.97%		Sensitivity	96.97%		Sensitivity	96.97%	
	Specificity	72%		Specificity	54.55%		Specificity	85.71%	
GC vs LC									
		Predicted							
	Group	GC	LC						
Actual	GC	38	9						
	LC	1	32						
	Error	0.125							
	Sensitivity	96.97%							
	Specificity	80.85%							

augmented patch level. The sensitivity in the discrimination of T-cells on the patch level was 98.8%, whereas specificity 98.7% (Fig 10.).

This was the first time that deep learning technique was applied specifically to immune cell counting in histological whole slide images [33].

The aim of the study was to find metabolomic biomarkers of sputum in order to distinguish lung cancer and healthy individuals as well as non-small cell and small-cell lung cancer patients [34]. Sputum or phlegm is the mucous substance secreted by cells in the lower airways (bronchi and bronchioles) of the respiratory tract and has been suggested as a potential biofluid source of biomarkers in lung cancer. Spontaneous sputum was collected and processed from 34 patients suspected of having LC, and 33 healthy controls. Of the 34 patients, 23 were subsequently

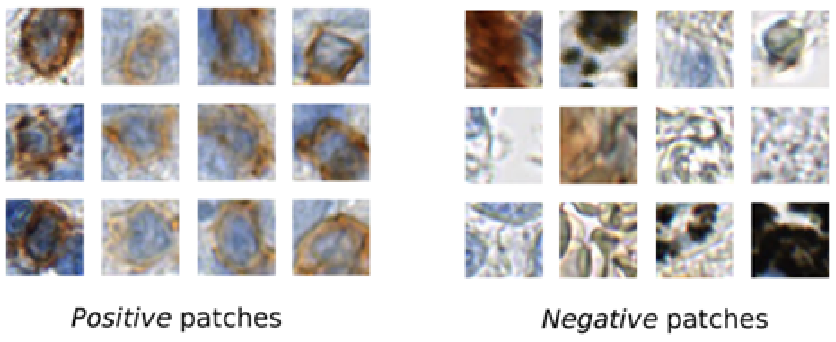


Figure 8: Example of the positive (A) and negative (B) patches of a training image set [33].

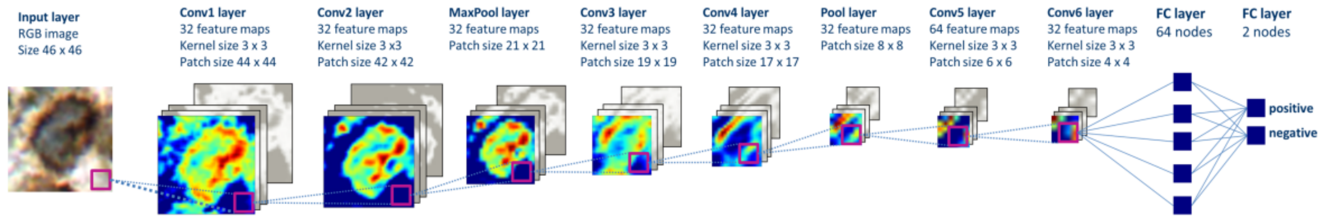


Figure 9: The structure of the deep convolutional neural network, which was applied to image classification [33].

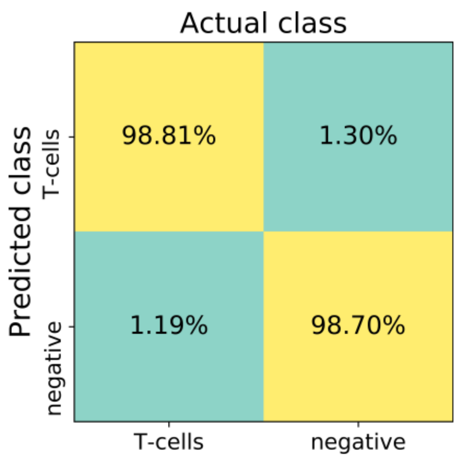


Figure 10: Confusion matrix [33].

diagnosed with LC (9LC+) at various stages of disease progression and 11 had symptoms (LC-). 2,582 m/z signals were identified using Flow Infusion Electrospray Ion Mass spectrometry (FIE-MS). Welch t-tests were performed and found 445 significantly differentiated m/z values between LC+ and healthy controls and 90 m/z values between LC+ and LC-.

Two ANN models were applied to discriminate lung cancer from healthy individuals as well as lung cancer from LC-. The activation function was set to hyperbolic tangent for both hidden and output layer; and the number of hidden layers was set to two for all problems. Weight decay regularization technique were used to control the complexity of the model parameters to avoid overfitting the models to the training data. Leave-one-out cross validation carried out to evaluate the model efficiency. The models presented high predictive efficiency through sensitivity, specificity and ROC curves of the two models which are presented in table 10.

Also, six m/z values were significantly differentiated between SCLC and NSCLC with p-values less than 0.05 from Welch t-tests. Thus, one more ANN model was applied to distinguish non-small cell lung cancer (NSCLC) and small-cell lung cancer (SCLC) patients taking as inputs the six potential biomarkers. Again leave-one-out cross validation carried out. The model achieved a sensitivity of 80% and a specificity of 100% for predicting SCLC (table 10). The 6 candidate metabolites markers were L-fucose, phenylacetic, caprylic, acetic, propionic acid, and glycine. Some of them, have already been linked to lung cancer. However, it was the first time that phenylacetate associated with cancer. Therefore, it is concluded that metabolomics based on the afore-mentioned methods can aid to discriminate lung cancer patient to healthy individuals and NSCLC to SCLC patients [34].

Table 10: Results of cross-validation prediction performance of our ANN models [34].

Classification	Mean no. of inputs	No. of hidden neurons	Sensitivity	Specificity	Positive Predictive Value	Negative Predictive Value	AUC
LC+ (vs Control)	1730.2	100, 50	96%	94%	92%	97%	0.99
LC+ (vs LC-)	71.9	40, 10	100%	91%	96%	100%	1.00
SCLC (vs NSCLC)	77.8	50, 20	80%	100%	100%	94%	1.00

2.3 Diabetes Mellitus

2.3.1 Type 2 Diabetes Mellitus

Type 2 diabetes mellitus (T2DM) has been linked to increased risk of cardiovascular and renal disease, dementia, and cognitive decline. Additionally,it is anticipated to rise to 366 million people by 2030 and its complications are prevalent and costly. T2DM progression can be reduced by early detection and appropriate treatment. Currently a clinical diagnosis of diabetes is based on a set of guidelines that specify levels of impaired fasting glucose (IFG) and impaired glucose tolerance (IGT); however diagnosing T2DM with these factors alone is problematic. The development of accurate methods for prediction of incident diabetes could facilitate the identification of individuals at high risk of T2DM and the design of prevention strategies. Machine learning methods are drawing increasing attention in the area of diabetes detection and risk assessment. They operate in a different manner than traditional statistical approaches due to their capabilities to deal successfully with large numbers of variables while producing powerful predictive models [35].

Studies

The following study created to identify possible biomarkers for predicting diabetes for diabetes development [35]. The data set which used belongs to a valuable clinical research database collected by the Jackson Heart Study, in an African American population known to be more unprotected to diabetes. It is comprised from 3,363 participants who were at-risk for developing diabetes and among them 584 developed incident diabetes during the 9-year follow-up period. In total, 93 features were collected like demographics, anthropometrics, blood biomarkers, medical history, echocardiograms, lifestyle behaviors and socio-economic status.

Five models were applied and compared. A RF (RF^{93}) and a logistic regression (LR^{93})

model based on 93 features, and a LR model previously published by the ARIC study (LR^{ARIC}). Moreover, a $RF^{15}$and LR^{15} were applied, based on the top 15 features (table 12) which were ranked from the RF^{93} using Gini index. Among the top-ranked variables, RF identified five well-known predictors of T2DM: hemoglobin A1c, fasting plasma glucose levels, waist circumference, triglycerides concentration, and age and potential biomarkers not involved in the ARIC model: adiponectin, C-reactive protein, and leptin. Also, both models identified six common variables: age, hemoglobin A1c, fasting glucose, waist circumference, HDL cholesterol and triglycerides.

To evaluate the efficiency of the five models, the dataset was separated 100 times into training and testing datasets to achieve robust performance estimates. For each iteration, the training dataset contained 500 diabetes patients and 500 healthy individuals. The rest data was used as testing dataset and the performance was measured using accuracy, sensitivity, specificity, and area under the curve (AUC).

RF^{93} produced mean values of $74\%, 75\%, 74\%$ and 0.82 of classification accuracy, sensitivity, specificity, and AUC, respectively (Table 11). LR^{ARIC} analyses produced mean values of $74\%, 74\%, 75\%$ and 0.82 of the same 4 metrics while LR^{93} produced $71\%, 70\%, 71\%$ and 0.78. The RF^{15} and LR^{15} generated little or no gains in performance.

The logistic regression model used in ARIC had inputs based on feedback from experts. On the other hand, RF was able to handle a large number of features and detect the most important for them, replacing the experts input during the model building process. The study concludes that RF methods have utility in the health care setting, where large datasets with thousands of well-characterized phenotypes and large numbers of participants are common [35].

Table 11: Prediction performance of the five models when using sample size 1000 (500 participants per group).

Method	Accuracy (%)	Sensitivity (%)	Specificity (%)	AUC
RF^{93}	74 (0.02)	75 (0.05)	74 (0.02)	0.82 (0.02)
LR^{ARC}	74 (0.01)	74 (0.05)	75 (0.01)	0.82 (0.02)
LR^{93}	71 (0.01)	70 (0.05)	71 (0.01)	0.78 (0.03)
RF^{15}	75 (0.02)	74 (0.05)	75 (0.01)	0.82 (0.02)
LR^{15}	74 (0.01)	74 (0.04)	74 (0.01)	0.82 (0.02)

Table 12: Top 15 Variables Found in Random Forest Analyses, according to the Gini Index (N = 1000)

Variable	Gini Index	Diabetes[a]	No Diabetes	p-value*
Hemoglobin A1c (%)	57.4	5.9(0.4)	5.4 (0.4)	< .0001
Fasting plasma glucose (mg/dL)	39.9	97.1 (10.7)	88.8 (7.8)	< .0001
Waist circumference (cm)	19.4	105.0 < (14.1)	97.3 (15.6)	< .0001
Adiponectin (ng/mL)	19.0	4091.9 (2750.3)	5566.3 (4032.8)	< .0001
Body mass index (kg/m^2)	17.6	33.56 (7.0)	30.7 (6.9)	< .0001
High sensitivity C-reactive protein (mg/dL)	15.4	0.6 (0.9)	0.4(0.7)	< .0001
Triglycerides (mg/dL)	14.9	113.88 (59.0)	94.8 (54.7)	< .0001
Age (years)	13.5	55.2 (11.1)	53.0 (12.8)	0.0001
Leptin (ng/mL)	13.2	32.1(27.2)	26.0 (21.9)	< .0001
Body Surface Area (m^2)	12.6	2.1 (0.2)	2.0 (0.2)	< .0001
eGFR (mL/min/1.73 m^2)	12.0	85.8(17.8)	87.2 (16.1)	0.02
2D calculated left ventricular mass (grams)	11.6	157.1 (89.3)	141.8 (39.3)	< .0001
Fasting HDL Cholesterol Level (mg/dL)	11.5	49.3 (12.9)	52.9 (14.8)	< .0001
Fasting LDL Cholesterol Level (mg/dL)	11.2	129.2 (37.9)	127.1 (35.9)	0.15
Aldosterone (ng/mL)	11.0	6.43 (6.48)	5.28 (4.05)	< .0001

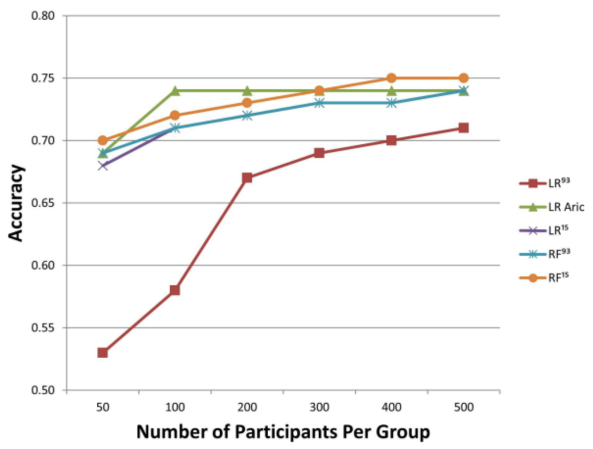


Figure 11: The dependence of classification accuracy on sample size is presented [35].

2.3.2 Gestational Diabetes

Gestational diabetes mellitus (GDM) is a pregnancy complication with considerable short and long term risks for both the mother and the offspring and can affect 1–20% of pregnancies depending on the diagnostic criteria used and on the population. Currently, risk factors alone are insufficient to accurately predict risk for GDM development. Typically, GDM is screened and diagnosed in the late second or early third trimester of pregnancy (24–28 weeks of gestation) due to the fact that there are no reliable tools for earlier detection. However, according to the recent guidance of the American Congress of Obstetricians and Gynecologists

(ACOG), early screening in the first trimester is recommended in women with risk factors (e.g., BMI above 25, hypertension, known impaired glucose metabolism, and family history of diabetes). Moreover, current studies show that Effective early identification, in the first trimester, of the development of GDM might reduce disease onset and associated maternal and perinatal complications. Hence, the discovery of a new method for early detection is necessary and the identification of early pregnancy biomarkers, may complement existing clinical risk factors in detecting women at high risk of developing GDM.

Studies

The aim of this study was to find microRNAs that can serve as possible biomarkers for Gestational diabetes mellitus (GDM) diagnosis in the first three months of pregnancy [36]. miRNAs were profiled from the maternal plasma of pregnant women.

There are many known clinical risk factors for GDM such as increased maternal age, history of GDM, obesity, family history of type 2 diabetes, a previous adverse pregnancy outcome and belonging to a high-risk ethnic group. However, they fail to precisely detect risk for GDM progress. GDM is diagnosed more accurately between sixth and seventh month of pregnancy with the current tools. GDM management after diagnosis at 24–28 weeks of pregnancy can cause long-term complications which include a greater likelihood of undergoing a caesarian section, the development of preeclampsia, and the post-pregnancy development of type 2 diabetes and cardiovascular disease in adulthood. These risks can be reduced through effective early detection in the first trimester. This is a necessity especially for women risk factors (e.g., BMI above 25, hypertension, known impaired glucose metabolism, and family history of diabetes) according to the recent guidance of the American Congress of Obstetricians and Gynecologists.

Two diagnostic strategies are the 75g oral glucose tolerance test (OGTT) and the 50g screening test followed by a 100g OGTT for those with a positive test. These tests are time-consuming, labor-intensive, and poorly tolerated by pregnant women. Therefore, new methods for easily tolerated screening and noninvasive diagnosis of pregnancy complications are necessary to be found. McroRNAs have been associated with several mechanisms related to diabetes pathogenesis. Also, they are stable and detectible in maternal blood and therefore they can serve as possible non-invasive biomarkers.

The sample used for the study were gathered in hospitals in Italy and Spain. It was consisted of 43 pregnant women who had not diabetes in the past: 23 women with GDM and 20 healthy women without complications during pregnancies. Cases and controls were matched about country of residence, pre-pregnancy maternal body mass index (BMI), maternal obesity, fetal gender, gestational age at delivery, and birth weight.

DESeq2 was applied to detect miRNAs for differential expression in GDM and healthy samples. Two miRNAs, miR-223 and miR-23a had significantly higher values in GDM patients (table 13). Three logistic regression models were applied based on these two miRNAs to test their discriminative efficiency. One model consisted of miR-223, one of miR-23a and the third model included both. miR-223 ($AUC = 0.94$ and $accuracy = 0.90$) was a little more accurate classifier than miR-223+miR-23a ($AUC = 0.91$ and $accuracy = 0.90$) or miR-23a ($AUC = 0.89$ and $accuracy = 0.90$). (Table 14)

Moreover, Logistic regression, random forest, and AdaBoost were applied combining several miRNAs which were not statistical significantly differentiated from DESeq2 test and all models had *accuracy* > 0.81 and $AUC > 0.74$.

Consequently, is recommended that circulating miRNAs in maternal plasma can act as predictive biomarkers for GDM. Those results should be validated on larger cohorts with samples from various origins [36].

Table 13: Differentially expressed miRNAs in GDM vs control groups.

Dataset	Country	MiRNA	Mean counts	log2 fold change	Adj. P value
1	Spain and Italy	miR-223-3p	185.22	5.46	1.42×10^{-7}
		miR-23a-3p	111.98	3.04	1.92×10^{-2}
2	Italy	miR-223-3p	65.52	5.61	2.56×10^{-4}
		miR-23a−3p	39.76	4.40	6.61×10^{-3}
3	Spain	miR-223−3p	463.90	4.88	1.42×10^{-2}

Table 14: Classification results. Classification results obtained via logistic regression (LR), random forest (RF) and AdaBoost models.

Model	AUC	Acc	Sens	Spec	F1 Score	MCC	PLR	NLR
RF	0.81**	0.81**	0.94	0.40	0.88**	0.41**	1.56**	0.16**
AdaBoost	0.77***	0.86***	0.94*	0.60*	0.91***	0.58***	2.34***	0.10***
LR	0.74**	0.76*	0.88**	0.40	0.85**	0.30	1.46	0.31*
LR: miR-223	0.94***	0.90***	0.94*	0.80***	0.94***	0.74***	4.69***	0.08***
LR: miR-23a	0.89***	0.90***	1.00***	0.60*	0.94***	0.73***	2.50***	0.00***
LR: miR-223 + mir-23a	0.91***	0.90***	0.94*	0.80***	0.94***	0.74***	4.69***	0.08***

This is a case-control study of pregnant women presenting to the West China Second University Hospital between December 2016 and December 2018 [37]. The aim of the study was to create an early detection model of (19 weeks) gestational diabetes mellitus (GDM) combining various potential predictors utilizing hepatic and renal and coagulation function measure. The data sample consisted of 215 with GDM and 275 healthy pregnant women. Blood samples were collected at 10–19 weeks of pregnancy. Forty-four hematologic and biochemical indices from routine blood tests, hepatic and renal function, and coagulation function examinations were collected. Bivariate tests of unadjusted associations between each predictor and GDM status were performed with Student's unpaired t-test or the Mann–Whitney U-test. The test was considered significant with a p value less than 0.001.

The parameters which differentiated significantly between the patients and control samples were further examined via support vector machines (SVMs) and generalized boosted modeling. The dataset was split 70% into a training set so that to create a predictive model for GDM, and 30% into a testing set. The optimal predictors were Prothrombin time (PAT-PT) and reference activated partial thromboplastin time (REF-APTT) having AUC 0.998 in light GBM and 0.997 in SVM. Also, direct bilirubin (DBIL) and fasting plasma glucose (FPG) value of hepatic and renal function examination had had in light GBM (AUC of 87.81%) and SVM (AUC of 84.79%).

Hence, it was concluded that PAT-PT and REF-APTT can be used risk assessment and earlier detection of GDM as well as a conjunction with FPG. This was first study to suggest coagulation function indexes during first 19 weeks of pregnancy as potential biomarkers of GDM progress.

2.4 Cardiovascular Diseases

2.4.1 Coronary Heart Disease

Coronary heart disease (CHD) is responsible for 13.3% deaths globally, and is also the most common cause of disability. Atherosclerosis is the most prevalent cause of CHD. Atheroma narrows the coronary artery and reduces the blood supply to the myocardium. T The gold standard method for determination of coronary arteries stenosis is coronary angiography (CA). It provides a detailed view of coronary anatomy, but is expensive and is associated with a significant morbidity and mortality. Hence the use of novel biomarkers with high sensitivity and specificity and their application to new algorithms to predict CHD remains an important approach to risk stratification.

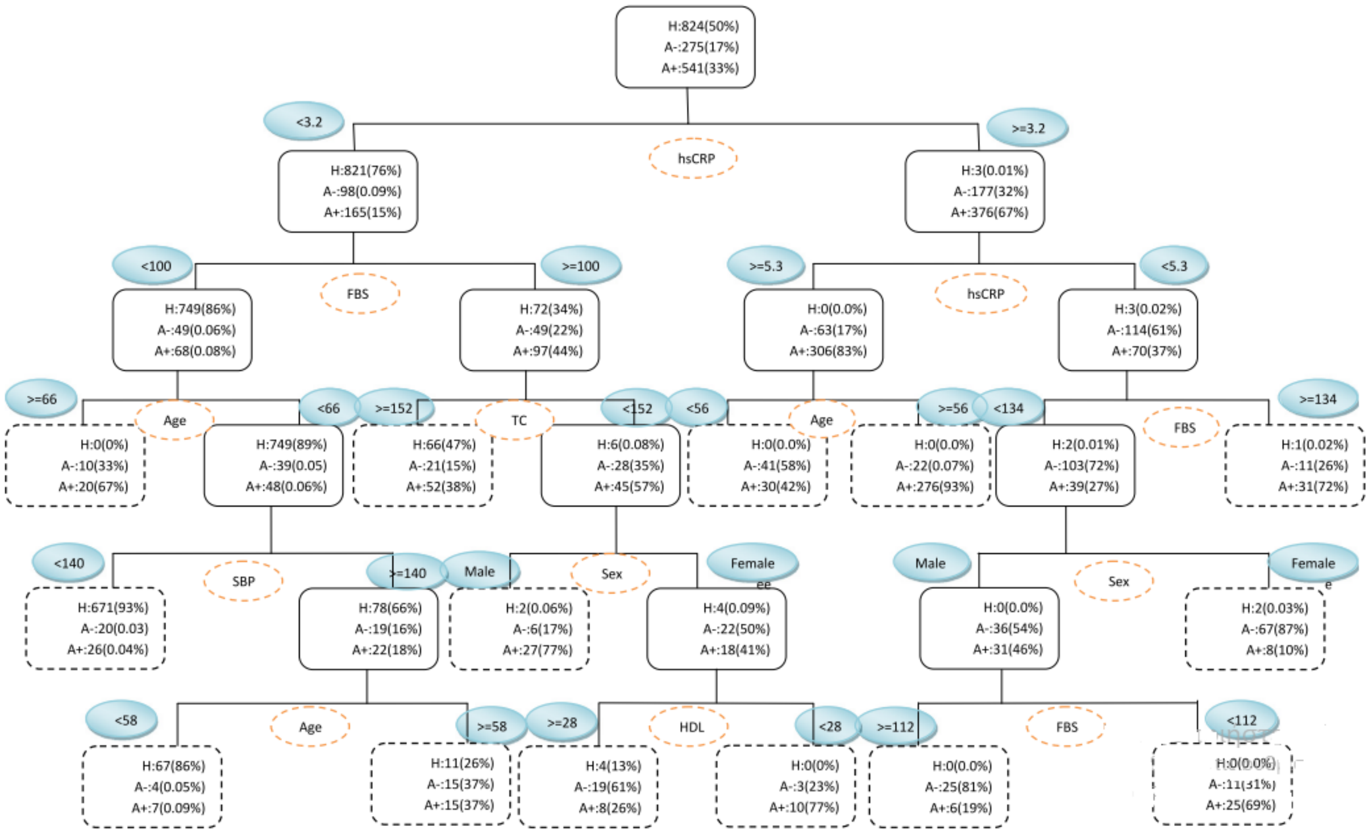
This study took place in Iran in 2016 to create a model for coronary heart disease prediction using a decision tree algorithm [38]. The study was based on a case-control study design of patients referred to Ghaem Hospital, Mashhad–Iran for coronary angiography, between September 2011 and May 2013. In 20 months of data collection, 1187 patients were enrolled. Using angiography, these individuals were divided into two groups: those (782) with significant angiographically defined CHD [Angiography (+)] (the case group) who had $\geq$ 50% occlusion in at least one coronary artery and those (405 individuals) with a normal angiogram ($<$ 50% obstruction in coronary arteries) [Angiography (–)]. Also, 1159 healthy controls were selected among people who made routine medical examinations.

The variables which were the well differentiated variables between the 3 groups of positive, negative angiography and healthy participants were low-density lipoprotein (LDL), high-density lipoprotein (HDL), systolic blood pressure (SBP), diastolic blood pressure (DBP) , total cholesterol (TC), Triglyceride (TG), Fasting blood glucose (FBG), high-sensitivity C-reactive protein (hs-CRP) age, sex. A decision tree was applied with 10 input variables and one output variable. The algorithm used the Gini index for selecting the variables and CART algorithm was used for pruning leaf nodes. As training set was used 70% (1640 cases) of the data and as testing dataset, 30% (706 cases). A confusion matrix was used to assess the efficiency of the model (table 15). Also, to compare the results the accuracy, sensitivity, specificity, and the receiver operating characteristics (ROC) curve were calculated and were equal to 95.3%, 97.8%, 92.9%, respectively. The area under the ROC curve was 0.95.

The variables remained in the final decision tree (Fig.12) were hs-CRP, FBG, age, TC, SBP,HDL and sex which had size 25, 14 leaves and 9 layers. An important result was that hs-CRP was at the top of the tree which divided the population with the highest information.

To validate the key role of hs-CRP in their model they ran the algorithm without hs-CRP. The results were a sensitivity, specificity, accuracy as 83.7%, 88.7%80.9%, respectively.

This was one of the most accurate decision trees model until that moment and can be used in clinical practice to distinguish healthy and CHD patients. They signify that hs-CRP as a new biomarker is strongly related with CHD even more than common biomarkers such as FBG and LDL [38].



H:824(50%) A-:275(17%) A+:541(33%)
<3.2
hsCRP
>=3.2
H:821(76%) A-:98(0.09%) A+:165(15%)
H:3(0.01%) A-:177(32%) A+:376(67%)
<100
FBS
>=100
>=5.3
hsCRP
<5.3
H:749(86%) A-:49(0.06%) A+:68(0.08%)
H:72(34%) A-:49(22%) A+:97(44%)
H:0(0.0%) A-:63(17%) A+:306(83%)
H:3(0.02%) A-:114(61%) A+:70(37%)
>=66
Age
<66
>=152
TC
<152
<56
Age
>=56
<134
FBS
>=134
H:0(0%) A-:10(33%) A+:20(67%)
H:749(89%) A-:39(0.05) A+:48(0.06%)
H:66(47%) A-:21(15%) A+:52(38%)
H:6(0.08%) A-:28(35%) A+:45(57%)
H:0(0.0%) A-:41(58%) A+:30(42%)
H:0(0.0%) A-:22(0.07%) A+:276(93%)
H:2(0.01%) A-:103(72%) A+:39(27%)
H:1(0.02%) A-:11(26%) A+:31(72%)
<140
SBP
>=140
Male
Sex
Female
Male
Sex
Female
H:671(93%) A-:20(0.03) A+:26(0.04%)
H:78(66%) A-:19(16%) A+:22(18%)
H:2(0.06%) A-:6(17%) A+:27(77%)
H:4(0.09%) A-:22(50%) A+:18(41%)
H:0(0.0%) A-:36(54%) A+:31(46%)
H:2(0.03%) A-:67(87%) A+:8(10%)
<58
Age
>=58
>=28
HDL
<28
>=112
FBS
<112
H:67(86%) A-:4(0.05%) A+:7(0.09%)
H:11(26%) A-:15(37%) A+:15(37%)
H:4(13%) A-:19(61%) A+:8(26%)
H:0(0%) A-:3(23%) A+:10(77%)
H:0(0.0%) A-:25(81%) A+:6(19%)
H:0(0.0%) A-:11(31%) A+:25(69%)

Figure 12: Decision tree with training dataset [38].

Table 15: Confusion matrix of testing dataset.

Actual outcome	Predicted outcome		
	Healthy	Angio−	Angio+
Healthy	328	4	3
Angiography−	11	56	64
Angiography+	15	32	193

2.4.2 Stroke

Stroke is a leading cause of permanent disability all over the world and it has become the main cause of death in China. About 80%–85% of strokes are ischemic with a short window period of 4.5 hours and thus the rapid and accurate diagnosis of strokes is of great importance for effective treatment. Traditional brain imaging technologies (e.g., the computed tomography (CT), magnetic resonance imaging (MRI), positron emission tomography (PET), and functional magnetic resonance imaging (FMRI)) are only suitable for detecting abnormal brain tissues with relatively large areas after a long onset time, and they are unable to discriminate different kinds of strokes. Alternatively, molecular biomarkers can serve as sensitive indicators for predicting stroke risk, and, consequently, the identification of novel stroke biomarkers may greatly facilitate the study of pathophysiological mechanisms of stroke and early diagnosis and symptomatic treatment of stroke even before an injury occurs.

The aim of the study was to establish a model to distinguish ischemic stroke patients from healthy persons [39]. It was the first time that fatty acid metabolites were used as biomarkers for stroke diagnosis. The whole data set was separated into two parts, a training and validation set, where 92 healthy persons and 149 patients included in the training and 30 healthy persons, and 48 patients included in the validation set.

158 fatty acids and their metabolites were tested in the training set and the measures among 17 of them between healthy and stroke patients were significantly differentiated (Figure 13.A). In addition to fatty acid metabolites, the measures of four clinical biochemical parameters were differentiated (Figure 13.B) including total cholesterol (TC), triglyceride (TG), high-density lipoprotein (HDL) and blood glucose were noticed.

Then, three models were applied using the above candidate biomarkers to distinguish ischemic stroke patients from the healthy group using stepwise logistic regression. The first one contained six fatty acid metabolites, AA, DHA, 13-HODE, 14,15- DHET and iso-PG (8-iso-15-keto-PGF2a and 8-iso-PGF3a). The second (model B) contained three clinical biochemical parameters, blood glucose, HDL and TG. The third model (C) included both fatty acid metabolites and clinical biochemical parameters, AA, DHA, 13-HODE, 8-iso-15-keto-PGF2a, and HDL. The training set was used in the training process of each algorithm. The algorithms aimed to distinguish ischemic stroke patients from the healthy group in the training set.

A random forest algorithm was applied for each model. The results are presented in figure 14. Model C had an AUC value of 0.9899, which was much greater than the AUC values of model A (0.9857) and model B (0.8527). Moreover, model C was superior to models A and B in accuracy (0.9577 vs. 0.9155 and 0.8169), sensitivity (100 vs. 96.30 and 77.78), and specificity (93.18 vs. 88.64 and 84.09). Therefore, model C is the most effective classifier to discriminate ischemic stroke patients from the healthy persons and using a combination of fatty acid metabolite and clinical biochemical parameters biomarkers can be more effective than the conventional approach of using a single biomarker as the clinical index with a risk of wrong diagnosis [39].

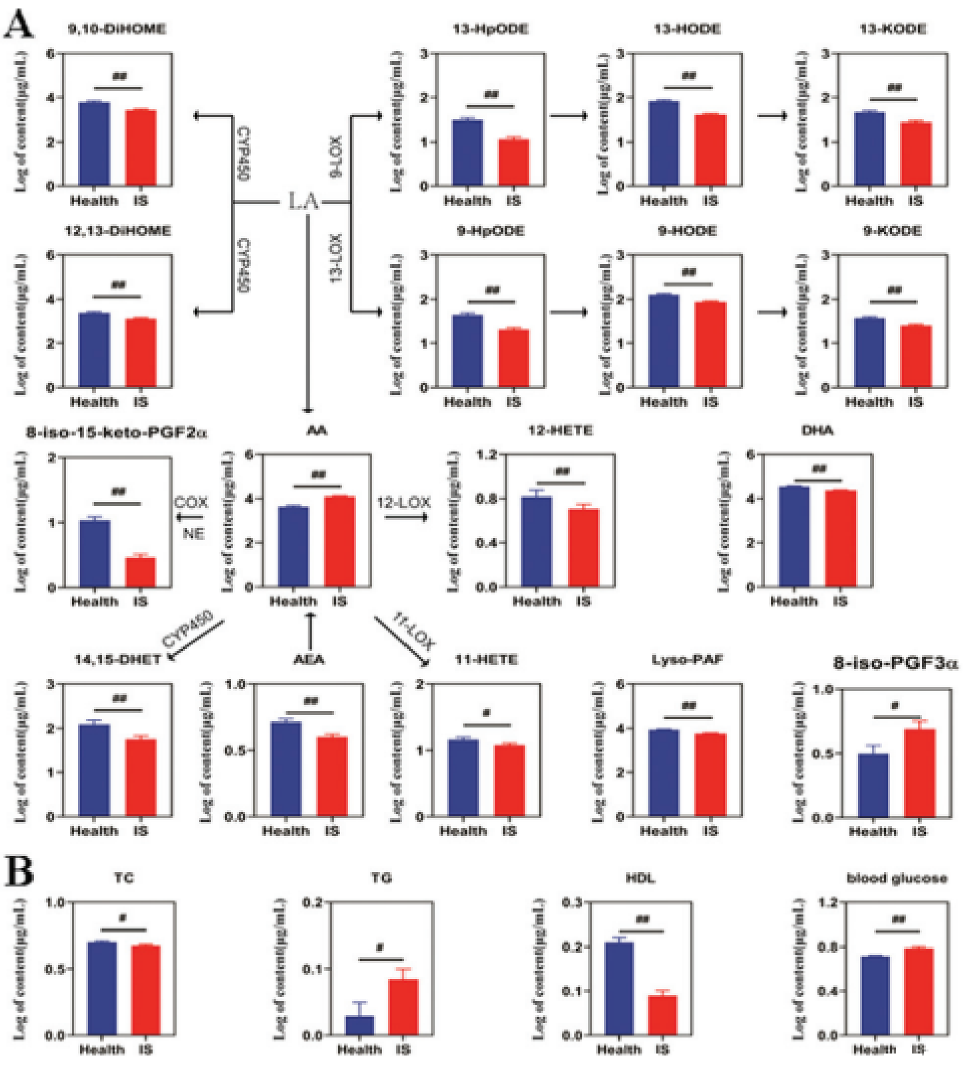
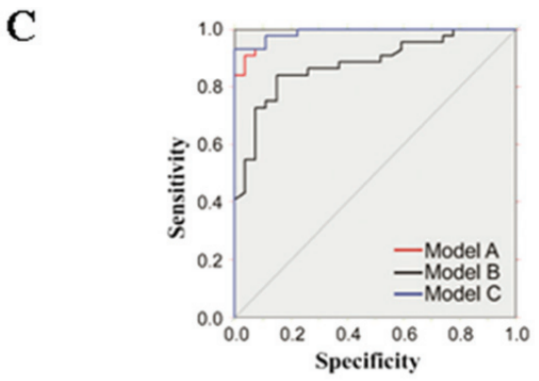
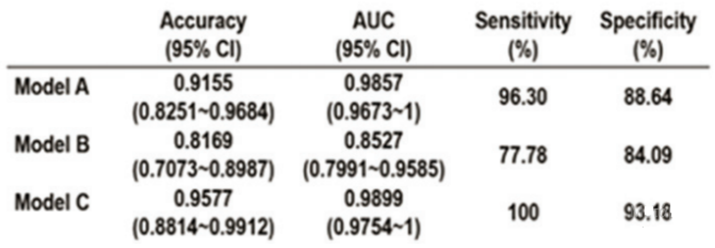


Figure 13: Comparison of fatty acid metabolite levels (A) and clinical biochemical parameters (B) between healthy volunteers (Health) and ischemic stroke patients (IS) in a training set [39].





	Accuracy (95% CI)	AUC (95% CI)	Sensitivity (%)	Specificity (%)
Model A	0.9155 (0.8251~0.9684)	0.9857 (0.9673~1)	96.30	88.64
Model B	0.8169 (0.7073~0.8987)	0.8527 (0.7991~0.9585)	77.78	84.09
Model C	0.9577 (0.8814~0.9912)	0.9899 (0.9754~1)	100	93.18

Figure 14: Comparison of the performance of different models using the receiver operating characteristic curves [39].

3 Unsupervised Learning

3.1 Principal Component Analysis

Principal Component Analysis, or PCA, is an unsupervised learning algorithm and is a primary tool for dimensionality reduction/feature extraction by transforming a large set of variables into a smaller one that still preserves most of the information in the original data [40]. The new variables are called principal components, their number is equal to the original variables and they are created as linear combinations of them. Principal components are constructed in a way that they are ordered by their importance which means that the maximum possible information is in the first component, the second maximum in the second component and so on. Some of the last principal components will be less importance and hence they can be ignored without losing significant information [40]. Moreover, principal components are uncorrelated which is a plus because there are models like the linear regression model that is required from the predictive variables to meet this assumption [41].

Suppose that are given n data points such that $x_i \in R^p$, $i = 1, ..., n$ i.e. $x_i = [x_{i1}, x_{i2}, \dots, x_{ip}]$ then can be created a $p \times n$ matrix:

$$X = \begin{bmatrix} \mathrm{x}_{11} & \cdots & \mathrm{x}_{1p} \\ \vdots & \cdots & \vdots \\ \mathrm{x}_{n1} & \cdots & \mathrm{x}_{np} \end{bmatrix}$$

The matrix of means M can be defined like

$M = [\overline{\mathrm{x}}, ..., \overline{\mathrm{x}}]$ where $\overline{\mathrm{x}} = \frac{1}{n} \times \sum_{i=1}^{n} x_i$

Then x can be centralized subtracting it from the matrix of means M like:

$$B = X - M \tag{4}$$

In practice, the idea is to create particular sets of coordinate axes and thus, to find some of the dimensions that can be ignored without losing significant information. A relative example illustrated in figure 15, which shows two versions of the same data. In the left diagram the data is presented having the original axes, while in the second new coordinate axes have been found. The y' dimension does not show much variability and so it can be probably ignored using only the x' axis values and still maintain most of the information of the original data. It is noteworthy that by doing this process, the results can be improved since some of the noise in the data is often removed [42].

The difficult task is to find those axes. The new coordinate axes are called principal com-

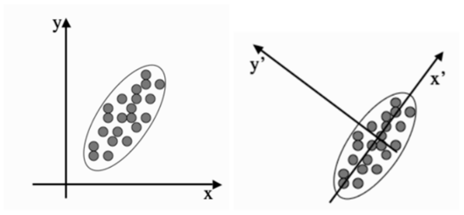


Figure 15: Two different sets of coordinate axes. The second consists of a rotation and translation of the first and was found using Principal Components Analysis [42].

ponents and can be specified by p orthogonal vectors $u_1, u_2, ..., u_p$. The principal components are orthogonal, linear transformations of the original data points. A principal component is considered as a direction in which the projection of the data has the maximum variation. [43]. The algorithm process begins by centering the data subtracting off the mean. Then the direction with the highest variation is chosen placing an axis in that direction which is called first principal component and corresponds to vector u_1. Subsequently, the direction which is orthogonal to the first and has the second highest variation is called second principal component and corresponds to vector u_2. The process is repeated until there are no others possible directions. Some of the last axes have very little variation, and so they can be ignored without significant variability loss [42].

To project data points to the direction of the first principal component, centralized matrix **B** should be multiplied by u_1^T. Vector u_1^T is $1 \times p$,**B** is $p \times n$ matrix so after multiplying them the result will be $1 \times n$. As mentioned above, the first principal component is the direction with the highest variation meaning that u_1 should be a vector that maximizes the $var(u_1^T \mathbf{B})$ [44].

$$\max_{u_1} var(u_1^T \mathbf{B}) \tag{5}$$

The sample covariance matrix of matrix B is $n \times n$ matrix and can be denoted using S. Since matrix B is multiplied by a linear transformation u_1^T then:

$$var(u_1^T \mathbf{B}) = u_1^T \; var(\mathbf{B}) \; u_1 = u_1^T \; S \; u_1$$

So, the maximization problem (5) is modified to:

$$\max_{u_1}(u_1^T S u_1) \tag{6}$$

The amount in equation 6 cannot be maximized because it is quadrative and there is no upper bound. Therefore, to address this issue the constrain $u_1^T u_1 = c$ is added to make it well defined. Typically, is chosen $c = 1$ which will be used to solve the problem but can be any other value. This makes sense because what needs to be calculated is the direction of the projection which is independent of the size of u. Thus, the maximization problem is modified to [44]

$$\max_{u_1}(u_1^T S u_1) \tag{7}$$

subject to

$$u_1^T u_1 = 1$$

The problem (7) is a constrained optimization problem. The gradient vector for the $u_1^T S u_1$ which needs to be maximized is parallel to the gradient vector of constrain which in this case is $u_1^T u_1 - 1$. Moreover, an unknown constant multiplier λ is necessary because the two gradients may have not equal magnitudes λ is just a multiplier then this means that [45]

$$\triangledown\,(u_1^T S u_1) = \lambda \triangledown\,(u_1^T u_1 - 1) \iff \frac{\partial(u_1^T S u_1)}{\partial(u_1)} = \lambda\frac{\partial(u_1^T u_1 - 1)}{\partial(u_1)} \iff S u_1 = \lambda u_1(8)$$

Since S is matrix and λ is scaler from the (8) arises that u_1 and λ are eigenvector and eigenvalue of S respectively. S is a $p \times p$ matrix so it can have p eigenvalues and p eigenvectors at most.

Replacing (8) in the $u_1^T S u_1$ arises that:

$u_1^T S u_1 = u_1^T \lambda u_1 = \lambda u_1^T u_1 = \lambda$

because of the constrained $u_1^T u_1 = 1$

So, (7) will be equal to:

$$\max_{u_1} \lambda \tag{9}$$

Hence, the value of $var(u_1^T \boldsymbol{B})$ will be the corresponding eigen value of eigen vector u_1 of covariance matrix S. Consequently, the eigen vector corresponding to maximum eigen value maximizes the equation (5). Thus, if the ordered eigen values are $\lambda_1 > \lambda_2 > \ldots > \lambda_p$ and the corresponding eigen vectors of S are $u_1, u_2, \ldots, u_p$, the u_1 eigen vector of covariance matrix S corresponding to the largest eigen value λ_1 and will be the first principal component. Since the second principal component is the direction of the second larger variation of the data after projection, the eigen vector corresponding to the second largest eigen value will be the second principal component and so on for the rest components [44].

Eigendecomposition problem of covariance matrix can be more easily solved using Singular Value Decomposition (SVD). The centralized **B** matrix is $p \times n$ and can be decomposed as following:

$$\mathbf{B} = \mathbf{U\Sigma V^T} \tag{10}$$

Where U contains the eigenvectors of $\mathbf{BB}^T$, $\mathbf{V}$ the eigen vectors of $\mathbf{B}^T\mathbf{B}$ and Σ is a diagonal matrix such that diagonal values are eigen values of $\mathbf{BB}^T$ or $\mathbf{B}^T\mathbf{B}$ [46]. At the same time, $\mathbf{BB}^T$ is the covariance matrix of $\mathbf{B}$, so $\mathbf{U}$ contains the eigenvectors of covariance matrix which has already been shown that are principal components. Thus, columns of $\mathbf{U}$ will be the principal components and it will be a $p \times p$ matrix since u_1 is $p \times 1$, u_2 is $p \times 1$ and so on [47]. The principal components in U are ordered automatically because in singular value decomposition the eigenvalues of $\mathbf{BB}^T$ which are the values of matrix Σ are ordered, so its first elements is λ_1, the second is λ_2 and so on. Thus, the first column of $\mathbf{U}$ will be the first principal component, the second column will be the second principal component etc [48].

Project data points to principal components

To project a point x, in i^{th} principal component must be computed $u_i^T b$, where b is the centralized value of x. To project a data point x to all the principal components must be computed $\mathbf{U}^T b$. So, $y = \mathbf{U}^T b$ will be a $p \times 1$ encoding of x. To project all the data points in principal components must be computed $\mathbf{U}^T\mathbf{B}$. So, $\mathbf{Y} = \mathbf{U}^T\mathbf{B}$ will be a $p \times 1$ matrix, containing the encodings of all data points x_i , $i = 1, \ldots, n$. Moreover, to project all the data points to the first p principal components is computed $\mathbf{Y} = \mathbf{U}_p^T\mathbf{B}$ where $\mathbf{Y}$ is $p \times n$ and obviously, $\mathbf{U}_p$ is the matrix having as columns the first p principal components which are the first p columns of matrix $\mathbf{U}$ [44].

Project test data point

Moreover, an out of sample data point x can be projected to principal components by computing $y = \mathbf{U}_b^T$ which will be a $p \times 1$ encoding of test data point x. In this case the data point x is also centered subtracting the average of all data points like was done before and this data point is also included to the average [44].

Reconstructing the data

After encoding the data and going to a lower dimensional space they also can be reconstructed meaning that they can get back to the original centered data. Columns of $\mathbf{U}$ are orthogonal to each other so $\mathbf{U}^T = \mathbf{U}^{-1}$. Therefore, reconstructing is done multiplying the projected data with the inverse of $\mathbf{U}$ transformation [44].

$$\hat{\mathbf{X}} = \mathbf{UY} = \mathbf{UU}^T\mathbf{B} \tag{11}$$

And now $\hat{\mathbf{X}}$ has dimensions $p \times n$ which is the same with the original matrix $\mathbf{B}$ of centered data. With the same way is reconstructed also a test data x:

$$\hat{x} = \mathbf{UU}^T b$$

Previously, was shown that the variance of data points which are projected in the i^{th} principal component is equal to the eigen value λ_i which has been shown that is the eigen value corresponding to eigen vector u_i. Thus, after chosen the first p principal components the percentage of the original variation that has been captured can be calculated using the following amount.

$$\frac{\sum_{i=1}^{d} \lambda_i}{\sum_{i=1}^{p} \lambda_i}$$

Obviously, $\sum_{i=1}^{d} \lambda_i$ is the variation that has been captured using the first d principal components and $\sum_{i=1}^{p} \lambda_i$ is the whole variance of the original data [40].

Standardization

It is critical to perform standardization prior to PCA, is that the latter is quite sensitive regarding the variances of the initial variables. That is, if there are large differences between the ranges of initial variables, those variables with larger ranges will dominate over those with small ranges.Mathematically, this can be done by subtracting the mean and dividing by the standard deviation for each value of each variable.

3.1.1 PCA Numerical Example

Assuming that is given a matrix data

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \\ -1 & -1 \end{bmatrix}$$

where the row represents the data samples, and the columns represent the features and the two features are named as $C1 = [1, 0, -1]$, $C2 = [0, 1, -1]$. Then, the covariance matrix must be constructed, where the diagonal entries are the variance, and off-diagonal entries are covariance.

$$Var_{C1} = 1$$
$$Var_{C2} = 1$$
$$Cov_{C1,C2} = Cov_{C2,C1} = \frac{(-1)(-1)}{3-1} = 0.5$$

Therefore the covariance matrix will be:

$$\begin{bmatrix} 1 & 0.5 \\ 0.5 & 1 \end{bmatrix}$$

Eigenvectors and eigenvalues can be calculated, simply by plugging λ_1, λ_2 into the following equation:

$$\begin{bmatrix} 1-\lambda & 0.5 \\ 0.5 & 1-\lambda \end{bmatrix} \cdot \vec{v} = \vec{0} \iff det \begin{bmatrix} 1-\lambda & 0.5 \\ 0.5 & 1-\lambda \end{bmatrix} = 0 \iff (1-\lambda)^2 - 0.5^2 = 0$$

and they will be

$$\lambda_1 = 1.5, \vec{v_1} = [0.707, 0.707]$$

$$\lambda_2 = 0.5, \vec{v_2} = [-0.707, 0.707]$$

To project the data points to the principal components, as shown above, the data matrix must be multiplied with the eigenvectors of the covariance matrix

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \\ -1 & -1 \end{bmatrix} \begin{bmatrix} 0.707 & -0.707 \\ 0.707 & 0.707 \end{bmatrix} = \begin{bmatrix} 0.707 & -0.707 \\ 0.707 & 0.707 \\ -1.414 & 0 \end{bmatrix}$$

The final matrix contains the transformed data. As shown above, the variance of data points that are projected in the i^{th} principal component is equal to the eigen value $_i$. As a result,

$$Var_{pc1} = 1.5 = \lambda_1$$

$$Var_{pc2} = 0.5 = \lambda_2$$

For this dataset, λ_1 explains 75% of the variance since $\frac{1.5}{1.5+0.5} = 0.75$; Likewise, λ_2 explains 25% of the variance since $\frac{0.5}{1.5+0.5} = 0.25$ [49].

3.1.2 Dual PCA

As mentioned above, the centralized matrix **B** can be decomposed using (10). There are many cases in which the size of dimensionality is higher than the number of data points. When that happens, it is computationally hard to compute the eigenvectors $\mathbf{B}\mathbf{B}^T$ as before. In contrast, is easier to compute the eigenvectors of $\mathbf{B}^T\mathbf{B}$ and the problem is set in a way for doing this. This leads to another algorithm is called Dual principal components analysis [47]. Nothing changes compared to direct PCA, meaning that mapping data to lower dimensional space, reconstructing and projecting out of samples data point are applied in a similar way. U is computed in terms of $\mathbf{B}$,Σ,$\mathbf{V}^T$, where Σ and V are calculated by decomposing the matrix $\mathbf{B}^T\mathbf{B}$ $\mathbf{V}$ is orthonormal matrix so $\mathbf{V}^T = \mathbf{V}^{-1}$ and $\mathbf{V}\mathbf{V}^T = 1$. Therefore, if equation (10) multiplied with $\mathbf{V}$ from right arises that [50]:

$$\mathbf{B}\mathbf{V} = \mathbf{U}\Sigma \tag{12}$$

Σ is diagonal matrix and contains the eigen values of $\mathbf{B}\mathbf{B}^T$ or $\mathbf{B}^T\mathbf{B}$ which are non-zero so Σ^{-1} can be computed [50].

$$\begin{bmatrix} \frac{1}{\lambda_1} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \frac{1}{\lambda_p} \end{bmatrix}$$

Thus (12) can be multiplied from right by Σ^{-1}.

$$\mathbf{B}\mathbf{V}\Sigma^{-1} = \mathbf{U} \tag{13}$$

Equation 13 is the key to derive dual PCA providing the U value for the steps of the algorithm. The equation 10 can be multiplied with $\mathbf{U}$ from the left and arises the following expression which is used in dual PCA to project the data points in principal components

$$\mathbf{Y} = \mathbf{U}^T\mathbf{B} = \Sigma\mathbf{V}^T \tag{14}$$

The projection of an out of sample point x is doing using:

$$\mathbf{Y} = \mathbf{U}^T b = \Sigma^{-1}\mathbf{V}^T\mathbf{B}^T b \tag{15}$$

The reconstruction of training data is doing using:

$$\hat{\mathbf{X}} = \mathbf{U}\mathbf{Y} = \mathbf{U}\mathbf{U}^T\mathbf{B} = \mathbf{B}\mathbf{V}\Sigma^{-1}\Sigma\mathbf{V}^T = \mathbf{B}\mathbf{V}\mathbf{V}^T \tag{16}$$

And in a similar way the reconstruction of an out of sample data point [50]

$$\hat{x} = \mathbf{U}y = \mathbf{U}\mathbf{U}^T b = \mathbf{B}\mathbf{V}\Sigma^{-1}\Sigma^{-1}\mathbf{V}^T = \mathbf{B}\mathbf{V}\Sigma^{-2}\mathbf{V}^T\mathbf{B}^T b \tag{17}$$

3.1.3 Kernel PCA

So far, the data was supposed to be linear. When this is not true, kernel PCA is applied. So far, the PCA has been described as a linear algorithm and this will not change [51]. However, what can change is the data in a way that the linear algorithm will be still efficient [52].

Kernel methods are not applied just in PCA, but also in many other machine learning algorithms. Going to a higher dimensional space, is usually computationally harder because when the dimensions increased, the number of parameters need to be learned exponentially grow. However, kernel functions can provide an effective solution to the issue of non-linearity. This is the idea behind kernel methods that when a nonlinear case is difficult to be handled, instead of changing the algorithm, change the data by going to higher dimensional space and then apply the linear technique there [52].

It is recalled that **B** is the matrix which contains the centralized original data and b a centralized data point. A data point b is mapped from the original space into a higher dimensional space H through a Φ such that $\Phi : b \to H$. However, $\Phi(b)$ is not computed explicitly [50]. Instead, some functions called kernel are applied to map the data in the feature space [53]. Those functions have the property that if they are applied to x,y are equal to $\Phi(x)^T\Phi(y)$ i.e. [50]

$$K(x, y) = \Phi(x)^T\Phi(y) \tag{18}$$

Each kernel function has a corresponding Φ which as mentioned before is not computed explicitly.On the contrary, the algorithm is modified in a way that it depends only on x^Ty instead of a point x itself and whenever there is in the algorithm the amount x^Ty to be replaced with the kernel function (5.16). The algorithm for Kernel PCA is similar to the one for Dual PCA except that in the case of Kernel PCA neither the training data nor the test data can be reconstructed [50].

Projection-Reconstruction in Kernel PCA

The data are projected to principal components applying:

$$\mathbf{Y} = \mathbf{U}^T\Phi(\mathbf{B}) = \Sigma\mathbf{V}^T \tag{19}$$

The **V** and Σ are computed from the decomposition of $\Phi(\mathbf{B})^T\Phi(\mathbf{B})$ so $\Sigma\mathbf{V}^T$ can be computed.

The reconstruction of data points would be using:

$$\hat{\mathbf{X}} = \mathbf{UY} = \mathbf{U}\Sigma\mathbf{V}^T = \Phi(\mathbf{B})\mathbf{V}\Sigma^{-1}\Sigma\mathbf{V}^T = \Phi(\mathbf{B})\mathbf{V}\mathbf{V}^T \tag{20}$$

$\mathbf{V}\mathbf{V}^T$ can be computed because **V** is the matrix with eigenvectors of $\Phi(\mathbf{B})^T\Phi(\mathbf{B})$. However, $\Phi(\mathbf{B})$ is unknown. So, reconstruction of an out of sample data point cannot be done.

To project out of sample point in kernel PCA is applied

$$\mathbf{y} = \mathbf{U}^T\Phi(\mathbf{B})^T = \Sigma^{-1}\mathbf{V}^T\Phi(\mathbf{B})^T\Phi(b) \tag{21}$$

This formula can be computed since Σ and V are already known and

$$\Phi(\mathbf{B})^T\Phi(b) = K(\mathbf{B}, b) \tag{22}$$

The reconstruction of test data point would be

$$\mathbf{U}y = \mathbf{U}\mathbf{U}^T\Phi(b) = \Phi(B)V\Sigma^{-2}V^T\Phi(B)^T\Phi(b) \tag{23}$$

and again the reconstruction cannot be done because $\Phi(B)$ is unknown [50].

3.2 Clustering

Clustering is an unsupervised problem and is the task of creating groups or clusters of objects such that objects that belongs to the same cluster are more similar than the objects are placed to different clusters [54]. To this end, clustering algorithms use similarity or dissimilarity (distance) measures and is defined a measure called variability as

$$variability = \sum_{e\,\in\,c} distance(mean(c), e)^2 \tag{24}$$

where c is a single cluster, mean(c) is the mean of the cluster and e is each point of the cluster [55]. There are various distances which can be used such as Minkowski which is defined by $d_{min} = (\sum_{i=1}^{n} |x_i - y_i|^m)^{\frac{1}{m}}$, Manhattan (Minkowski for $m = 1$), Euclidean (Minkowski for $m = 2$), average distance which is defined by $d_{ave} = (\frac{1}{n}\sum_{i=1}^{n} |x_i - y_i|^2)^{\frac{1}{2}}$, Mahalanobis distance which is defined by $d_{mah} = \sqrt{(x-y)S^{-1}(x-y)^T}$ where S is the covariance matrix of the data, and others. There is no standard choice of distance, each one can be used depending on the data [56]. Variability is very close to variance, but it is not the same. Variance is normalized, this means that the penalty for big cluster with a lot of variances in it is no higher than a penalty of a tiny little cluster with a lot of variances in it. By not normalizing, the big highly diverse clusters are penalized more than small highly diverse clusters. Assuming that C is a group of clusters then can be defined the sum of all variabilities to clusters in C as [55]

$$dissimilarity(C) = \sum_{c\,\in C} variability(c) \tag{25}$$

The optimization problem that must be solved in clustering is just to find a group of clusters C in which dissimilarity is minimized subject to a constraint for the number of clusters. The constraint is added to exclude the case in which each cluster consists of just a single data point. In this case the variability of each cluster would be zero and consequently the same for dissimilarity would be zero too, but obviously this is not an effective solution [55].

3.2.1 Hierarchical Clustering

Hierarchical Clustering can be applied either bottom up or top down. The bottom-up case will be presented below [57].

Let the distance between clusters i and j be represented as d_{ij} and n_i the number of objects in cluster is.

-Hierarchical Clustering is a deterministic algorithm and starts by assigning each item to a cluster, so that if are given N items, N clusters are created, each containing just one item.

-Find the pair of clusters i,j with the smallest d_{ij} and merge them into one cluster k, so that now there is one less cluster.

-Calculate a new set of distances d_{km} using the following formula

$$d_{km} = \alpha_i d_{im} + \alpha_j d_{jm} + \beta d_{ij} + \gamma |d_{im} - d_{jm}| \tag{26}$$

Where m represents any other cluster than k. The new distances replace d_{im} and d_{jm}.

-The process is repeated until all items are clustered into a single cluster of size N.

To calculate the distance between two clusters which are single observations various distances can be used depending on the application of the clustering. The Euclidian distance is the most widely used when the variables are continuous.

Linkage metrics are used to find the distance between clusters and combine them when they are not an individual observation. Each linkage metric corresponds to a choice for $\alpha_i, \alpha_j, \beta$ and γ.

The most common linkage metrics are the following.

Single-linkage. Also known as nearest neighbor clustering. The distance between two clusters is the shortest distance from any member of one cluster to any member of the other cluster. The coefficients of the distance equation are

$$\alpha_i = \alpha_j = 0.5, \beta = 0, \gamma = -0.5$$

Complete-linkage. Also known as furthest neighbor clustering. The distance between two clusters is the greatest distance from any member of one cluster to any member of the other cluster. The coefficients of the distance equation are

$$\alpha_i = \alpha_j = 0.5, \beta = 0, \gamma = 0.5$$

Average-linkage. In this case the distance between two clusters is the average distance from any member of one cluster to any member of the other cluster. The coefficients of the distance equation are [57]

$$\alpha_i = \frac{n_i}{n_k}, \alpha_j = \frac{n_j}{n_k}, \beta = 0, \gamma = 0.5$$

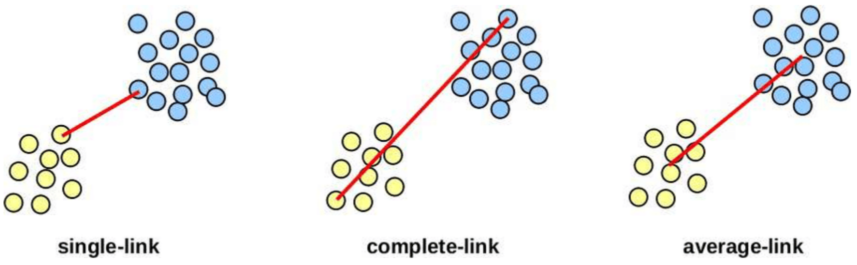


Figure 16: Linkage metrics [58].

A graphical represantation of the three methods is the figure 16. All linkage metrics are used in practice depending on the application of the clustering. Choosing different linkage criteria, the results are different too.

Stopping criteria

-In the case that there is a priori knowledge that the data naturally falls into k classes, the procedure can stop when k clusters have been created.

-The results of hierarchical clustering are usually presented in a **dendrogram**. A dendrogram example is represented below which shows the hierarchical clustering of 22 observations. Dendrograms begin with each observation in a separate cluster. The vertical axis represents the objects and clusters. At each step, the two clusters that are most similar are joined into a single new cluster. The horizontal axis reflects the distance between clusters [59]. Each fusion of two clusters is represented on the graph by joining two horizontal lines into one horizontal line. Once fused, objects are never separated. In the dendrogram in figure 17 the big difference between clusters is between cluster of 7,8,9,10,11,12 versus the cluster 14,15,16,17,18,19,20,21,22 and the cluster 1,2,3,4,5 [57]. Thus, the dendrogram can be used for choosing the optimal number of clusters.

Limitations

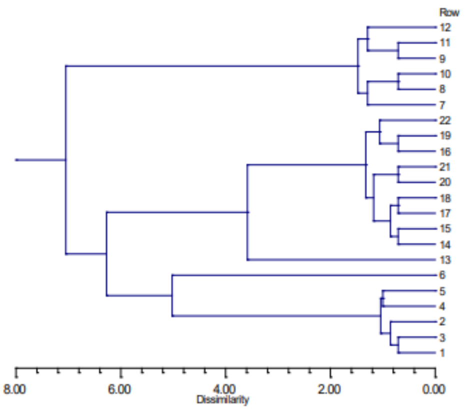


Figure 17: Dendrogram [57].

Hierarchical Clustering requires high amount of memory used by the algorithm to exetute and produce the result (space complexity). Moreover, is a slow algorithm, and it is not proposed especially in applications with big data [60]. However, it is used in K means algorithm which is presented next to find an appropriate number of clusters. The way is used is analyzed below.

3.2.2 K-means

$K-$means is a greedy and iterative algorithm and the most commonly used in clustering [61]. Suppose that are given N data points. The way that K-means works is [62]:

- Decide how many clusters are needed.

- Given that are needed K clusters, K random points are picked as initialized cluster centers.

- The data points are assigned to the closest cluster center

-Compute K new centroids by averaging examples in each cluster.

- Iterate until the points do not move between clusters and centroids are stabilized

The complexity of one iteration is $k * N * d$, where N is number of points and d time required to compute the distance between a pair of points. Usually, the algorithm converges quickly, and the number of iterations is small [61].

In K-means algorithm there are two issues. The first one is related to the number of clusters. Choosing wrong value for K can lead to strange and not accurate results [62].

Optimal K

One way to choose K is by using a priori knowledge about the application. This is something that often happens.

Elbow Method

Elbow method is used to find a proper K value. For choosing the appropriate number of clusters the average Within-Cluster-Sum of Squared Errors (WSS) for different values of k can be computed. If the total distance is high, it means that the points are far from each other and might be less similar to each other.The average within-cluster sum of squares measures how cohesive the clusters are and is defined as

$$WSS = \frac{1}{N}\sum_{i=1}^{K}\sum_{j=1}^{n_i}(x_{ij} - \overline{\mathrm{x}}_i)^2 \tag{27}$$

Where n_i is the number of points in cluster k for $k = 1, ..., K$ [63]. Starting from $K = 1$ and increasing K each time, the higher the value K will be, the lower the within distance. Obviously if $K = N$ the total distance will be zero because each cluster will contain only one data point, but this is not a solution. The aim is to find a value K that the amount of the decrease in within distance will be no significant any more [64]. For the data illustrated in figure 18, after $K = 3$, the reduction in total distance is not important. Therefore, going beyond this K value, will not contribute much to clustering algorithm and will only make clusters more complicated [63].

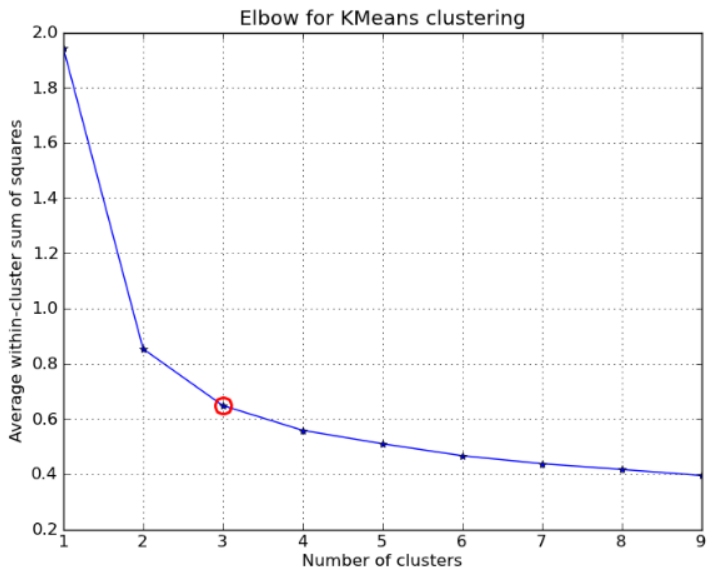


Figure 18: Elbow K Means [63].

Another approach is to run Hierarchical Clustering on a small subset of the data and use the number of clusters in K-means [55].

Initial centroids

The second problem in K-means is the choice of proper initial centroids. Different initial centroids can lead to different number of iterations and most important different results [62].

Random data points

In random data points approach, *k* random data points are selected from the dataset and used as the initial centroids. This approach is unstable and probably the selected centroids are not well positioned throughout the entire data space [65].

Sharding centroid initialization

The sharding centroid initialization algorithm primarily depends on the composite summation value of all the attributes for a particular instance or row in a dataset. The idea is to calculate the composite value and then use it to sort the instances of the data. Once the data set is sorted, it is then divided horizontally into k shards.Finally, all the attributes from each shard will be summed and their mean will be calculated. The shard attributes mean value collection will be identified as the set of centroids that can be used for initialization [64].

K-Means++

K-means++ is a smart centroid initialization method for the K-means algorithm. The goal is to spread out the initial centroid by assigning the first centroid randomly then selecting the rest of the centroids based on the maximum squared distance. The idea is to push the centroids as far as possible from one another [64].

Dimensionality Reduction in K-Means

K-Means uses Euclidean distance to calculate the distance between points. In very high dimensional space, Euclidean distances may not work properly. So, if a large number of features are given, the use of dimensionality reduction algorithm like PCA before K-means can solve this problem and speed up the process [66].

Mini Batch K-Means

For large datasets, K-Means can take a lot of time to converge. The Mini Batch K-Means is a variant of the K-Means algorithm which uses mini-batches to reduce the computation time, while still attempt to optimise the same objective function. Mini-batches are subsets of the input data, randomly sampled in each training iteration. These mini-batches manage to reduce significantly the computations required to converge to a local solution. In contrast to other algorithms that speed up the computation time of k-means, mini-batch k-means produces results that are only slightly worse than the standard algorithm [66].

3.2.3 K-means Numerical Example

An example for building a K-means Algorithm ”by hand” is presented below for the data set given in the following table. The data contain only 6 samples and two variables named x and y.

Table 16: Training data.

point	x	y
1	7	3
2	4	5
3	2	4
4	0	1
5	9	7
6	6	8

The initial centroids can randomly be chosen to be for Group 1: point 5 with center (9, 7) and for Group 2: point 6 with center (6, 8). The distances between the centroids and the data points can be calculated using Euclidean distance. The distances are presented in table 17. Based on the distance matrix, each point is assigned to its closest group. The new clusters and the new centroids are presented in table 18.

$$\sqrt{(x_a - x_b)^2 + (y_a - y_b)^2)}$$

Table 17: Distances

	1	2	3	4	5
2	3.61				
3	5.10	2.24			
4	7.28	5.66	3.61		
5	4.47	5.39	7.62	10.82	
6	5.10	3.61	5.66	9.22	3.16

The above process is repeated calculating the distance between each point to the center of each cluster again and again until the points do not move between clusters and centroids are stabilized. The new centroids were computed by taking the mean of the coordinates x and y of the points in one cluster. More specific, Group 1 includes the points 5 and 1 with (8, 5) as center

$8 = \frac{9+7}{2}$ and $5 = \frac{7+3}{2}$

and Group 2 includes the points 6, 2, 3 and 4 with (3, 4.5) as center

Table 18

	points	center
cluster1	5,1	(8,5)
cluster2	6,2,3,4	(3,4.5)

$3 = \frac{6+4+2+0}{4}$ and $4.5 = \frac{8+5+4+1}{4}$

The distances between the centroids and the data points were calculated again as before. Based on the distance matrix, each point is assigned to its closest group. The new clusters and the new centroids are calculated again and presented in table 20. After one more repetition, it is concluded from table 21 that all points are correctly placed to its nearest cluster, so the algorithm stops [67].

Table 19: Distances

points	distance to cluster 1	distance to cluster 1
1	2.24	4.27
2	4	1.12
3	6.08	1.12
4	8.94	4.61
5	2.24	6.5
6	3.61	4.61

Table 20: Distances

	points	center
cluster1	1,5,6	(7.33,6)
cluster2	2,3,4	(2,3.33)

Table 21: Distances

points	distance to cluster 1	distance to cluster 1
1	3.02	5.01
2	3.48	2.61
3	5.69	0.67
4	8.87	3.07
5	1.95	7.9
6	2.4	5.08

3.2.4 Clustering Performance Evaluation

To evaluate the performance of a clustering algorithm are used different metrics just like in classification algorithms.

Silhouette score

Silhouette score is used to evaluate the quality of clusters created using clustering algorithms in terms of how well samples are clustered with other samples that are similar to each other and thus, choosing the optimal number of clusters. Silhouette Score, S_i, for each observation i is calculated using the following formula [68]

$$S_i = \frac{b_i - a_i}{Max(a_i, b_i)} \tag{28}$$

where α_i also called average intra-cluster distance, is the average distance between the observation and all other data points in the same cluster.

b_i is also called average nearest-cluster distance is the average distance between the observation and all other data points of the next nearest cluster.

Silhouette Score ranges from -1 to +1. Higher value of Silhouette Score indicates observations are well clustered. Silhouette Score = 1 indicates that the observation (i) is well matched in the cluster assignment. A value near 0 represents overlapping clusters with samples very close to the decision boundary of the neighboring clusters. A negative score represent samples probably assigned to wrong cluster. [68].

The silhouette plots can be used to select the most optimal value of K (no. of clusters) in K-means clustering. The aspects to look out for in Silhouette plots are cluster scores below the average silhouette score, wide fluctuations in the size of the clusters, and the thickness of the silhouette plot. For example, in figure 19 is represented a Silhouette plot for K-Means clusters with 2,3,4,5 clusters. The optimal number of clusters is 3 because in this case the silhouette score for each cluster is above the average silhouette scores, the fluctuation in size is similar and the thickness is more uniform [69].

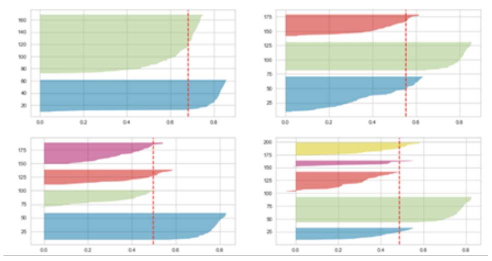


Figure 19: Silhouette Analysis for 2, 3, 4, 5 Clusters [69].

Dunn index

The Dunn index (DI) is a metric for evaluating clustering algorithms. The aim of this index is to identify sets of clusters that are compact, with a small variance between members of the cluster, and well separated, where the means of different clusters are sufficiently far apart, as compared to the within cluster variance. Dunn index can also be used for choosing the number of clusters k. The number of clusters that maximizes Dunn index is taken as the optimal one. The Dunn index for c number of clusters is defined as :

$$DI = \min_{1 \leq i \leq c} \left\{ \min_{1 \leq j \leq c, j \neq i} \left\{ \frac{\delta(X_i, X_j)}{\max\limits_{1 \leq i \leq k} \{\Delta(X_k)\}} \right\} \right\} \tag{29}$$

where $\delta(X_i, X_j)$ is the distance between cluster X_i and cluster X_j and $\Delta(X_k)$ is the distance within the cluster X_k. Dunn index is a ratio between 0 and 1 and the higher it is, the better is the clustering. It also has some drawbacks. As the number of clusters and dimensionality of the data increase, the computational cost also increases [70].

4 Logistic Regression

4.1 Introduction

Logistic regression is a classifier pretty popular especially in biostatistics. The idea in logistic regression is to make an assumption directly for the posterior probabilities of the K classes via linear functions in x. The model has the form

$$log\frac{(P\,(Y=k|X=x_i))}{(P\,(Y=K|X=x_i))} = w_{k0} + w_{k1}x_1 + w_{k2}x_2 + ... + w_{kp}x_p = w_{k0} + w_k^T x, \ \ k = 1, ..., K-1 \quad (30)$$

Where w_k are the parameter vectors corresponded to class k and x is the vector containing the p independent variables. The decision boundary that logistic regression generates is linear, which can be used for classification purposes [71].

4.2 Binary case

The two class case ($K = 2$) is called binary case and is widely used in many applications such that biostatistical applications where binary responses (two classes) occur quite frequently, in quality control to check if a product is effective or not etc. It can be assumed that $y_i \in [0, 1]$ which can be affected by one or more fixed covariates $x_1, x_2, ..., x_p$. Each covariate is corresponded to a parameter $w_r, r = 1, ..., p$. In binary case, logistic regression models posterior through sigmoid function [71].

$$if\ y = 1 : p_1(x; w) = P(Y = 1|X = x) = \frac{e^{w^T x_i}}{1 + e^{w^T x_i}} \quad (31)$$

$$if\ y = 0 : p_2(x; w) = \ P(Y = 0|X = x) = 1 - p_1(x; w) = \frac{1}{1 + e^{w^T x_i}} \quad (32)$$

where $w = \left[w_0, w_1, ..., w_p\right]$ is the parameters vector and $x_i = \left(x_{i1}, x_{i2}, ..., x_{ip}\right), i = 1, ..., N$ is the vector for the covariates for i^{th} observation. The parameter vector w is unknown and is computed minimizing the loss function. The loss function for N observations which is used is called cross entropy and is presented in (33). Loss function measures how close is the prediction of the model to the original value. The minimum of $J(W)$ cannot be calculated directly, and thus an iterative optimization algorithm like gradient descent can be used. Gradient descent requires the computation of derivatives with respect to the parameters w which are given in equations (34),(35). Subsequently, the optimized parameters w for class k can be computed using the gradient update rule (36) [72]. In this rule, η is called learning rate.

$$J(w) = -\frac{1}{N}\sum_{i=1}^{N}\left[y^{(i)}log\frac{e^{w^T x_i}}{1+e^{w^T x_i}} + (1-y^{(i)})log\frac{e^{w^T x_i}}{1+e^{w^T x_i}}\right] \; for\, k = 1,\ldots,K-1 \tag{33}$$

$$\nabla J(W) = \left[\frac{\partial J}{\partial w_1}, \quad \frac{\partial J}{\partial w_2}, \quad \ldots, \quad \frac{\partial J}{\partial w_p}\right] \tag{34}$$

$$\frac{\partial J(w)}{\partial w_j} = \frac{1}{N}\sum_{i=1}^{N}\left(\frac{e^{w^T x_i}}{1+e^{w^T x_i}} - y^{(i)}\right)x_{ij}, \quad j = 1,...,p \tag{35}$$

$$w_j^{'} = w_j + \eta\frac{1}{N}\sum_{i=1}^{N}\left(\frac{e^{w^T x_i}}{1+e^{w^T x_i}} - y^{(i)}\right)x_{ij}, \; j = 1,...,p \tag{36}$$

Logistic regression has a linear decision boundary (a line in $2-$dimensional space) which is given by

$$log\frac{p_1(x;w)}{p_2(x;w)} = log\frac{(P\,(Y=1|X=x))}{(P\,(Y=0|X=x))} = w_0 + w^T x_i = w_0 + w_1 x_{i1} + ... + w_{p-1}x_{i(p-1)} = 0 \tag{37}$$

and corresponds to $P(Y = 1|x) = P(Y = 0|x) = 0.5$ and separates the predictions of $1's$ from $0's$ [73].

A new data point is classified in class 1 if $log\frac{p_1(x;w)}{p_2(x;w)} > 0 \iff p_1(x;w) > 0.5$, otherwise is classified in class 2 [74].

4.2.1 Logistic Regression-Numerical Example

An example for building a Logistic Regression Algorithm ”by hand” is presented below for the data set given in the following table. The data contain only 4 samples, 2 independent variables x_1 and x_2 and a binary target variable y.

Three parameters can be assumed (2 weights and 1 bias) and the can be initialized as:

$$w_1 = w_2 = b = 0$$

$$\eta = 0.1$$

Table 22: Training data.

x1	x2	y
0	0	0
0	1	1
1	0	1
1	1	1

The gradient vector in this case has 3 dimensions:

$$\nabla J(W) = \begin{bmatrix} \frac{\partial L(w)}{\partial w_1} \\ \frac{\partial L(w)}{\partial w_2} \\ \frac{\partial L(w)}{\partial b} \end{bmatrix} = \begin{bmatrix} \frac{1}{4}\sum_{i=1}^{4}(\sigma(wx_i + b) - y_i)x_{i1} \\ \frac{1}{4}\sum_{i=1}^{4}(\sigma(wx_i + b) - y_i)x_{i2} \\ \frac{1}{4}\sum_{i=1}^{4}\sigma(wx_i + b) - y_i \end{bmatrix} = \begin{bmatrix} \frac{1}{4}\sum_{i=1}^{4}(\sigma(0) - 1)x_{i1} \\ \frac{1}{4}\sum_{i=1}^{4}(\sigma(0) - 1)x_{i2} \\ \frac{1}{4}\sum_{i=1}^{4}\sigma(0) - 1 \end{bmatrix} = \begin{bmatrix} \frac{-0.5-0.5}{4} \\ \frac{-0.5-0.5}{4} \\ \frac{0.5-0.5-0.5-0.5}{4} \end{bmatrix} =$$

$$\begin{bmatrix} -0.25 \\ -0.25 \\ -0.25 \end{bmatrix}$$

The gradient can be used to compute the new parameter vector as shown in (36) which will be as follows. The process can be repeated with the same way until the algorithm converges and the optimized parameters be found. Then, the model can be used for classification of new data points.

$$w' = \begin{bmatrix} w_1 \\ w_2 \\ b \end{bmatrix} - \eta \begin{bmatrix} -0.25 \\ -0.25 \\ -0.25 \end{bmatrix} = \begin{bmatrix} -0.025 \\ -0.025 \\ -0.025 \end{bmatrix}$$

4.3 Multinomial Logistic Regression

In the general case in which are given K classes and $K > 2$, it can be considered that $y_i \in 1, 2, \dots, K$ and an input is classified into one of K classes. Two variants of multinomial or multi-class logistic regression will be presented, **one vs rest classification** and **softmax regression**.

4.3.1 One vs rest classification

The idea is to train K different binary classifiers. The first classifier will predict whether an input belongs to class 1 or not, the second will predict whether an input belongs to class 2 or not, and so on until K classifier. Each one of the models will have its own parameter vector and they are trained on the same data as has already presented in the binary case but the labels of

$y_i = 0$ or $y_i = 1$ will be different in these K cases.

So at test time given a new input x the K binary models are used to calculate the posterior probabilities $P(Y = k|X = x), k = 1, \ldots, K$ and the data point is classified to the class k in which the probability is maximum.

This approach can be applied not just in logistic regression but also in any binary machine learning model and convert that into a multi- class classification model [75].

4.3.2 Softmax Regression

In softmax regression the output is a vector of probabilities corresponding to each class which is structured in such a way that the first element in the vector will be the posterior probability of being in the first class, the second element will be the probability of being in the second class and so on up to K^{th} element in the output which is the probability of being in the K^{th} class. The way that softmax regression models the posterior probability is through softmax function, ensuring that they sum to one and remain in [0, 1] [76].

$$f(x; W) = \begin{Bmatrix} \begin{pmatrix} P(y = 1|x; w_1) \\ P(y = 2|x; w_2) \\ \vdots \\ P(y = K|x; w_K) \end{pmatrix} \end{Bmatrix}$$

where

$$P(Y = k|X = x) = p_k(x; w_k) = \frac{e^{w_k^T x_i}}{\sum_{i=1}^{K} e^{w_l^T x_i}} \; for\, k = 1, \ldots, K-1 \tag{38}$$

and

$$P(Y = K|X = x) = 1 - \sum_{k=1}^{K-1} P(Y = k|X = x) = \frac{1}{\sum_{i=1}^{K} e^{w_l^T x_i}} \tag{39}$$

The choice of the last class as the denominator is arbitrary, it could be any other without affecting the model. In 2−class case there was just a single parameter vector but in SoftMax regression there is a parameter vector w_k corresponding to each class. The parameters vectors can be stacked in a parameter matrix

$$W = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_K \end{bmatrix} = \begin{bmatrix} w_{10} & w_{11} & \dots & w_{1p} \\ w_{20} & w_{21} & \dots & w_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ w_{K0} & w_{K1} & \dots & w_{Kp} \end{bmatrix}$$

Then the model output can be written as $SoftMax\ (Wx)$ .The parameters w_{ij} are unknown [77] . The optimized parameters w are calculated minimizing the loss function. The loss function which is used is called cross entropy. The formula for one data point's cross entropy is presented in (40). The total cross entropy, or loss, will be the sum of all the cross entropies and presented in (41). Just as in binary case, the minimum of $J(W)$ cannot be calculated directly, and gradient descent is used. Therefore, is required the computation of derivatives with respect to the parameters w which are given in equations (42),(43). Subsequently, the optimized parameters w for class k can be computed using the gradient update rule (44). In this rule, η is called learning rate and $I\{y^{(i)} = k\}$ will be 1 if x_i belongs to class k, and 0 if x_i does not belong to class k [78]. A new data point is classified in the class k in which the posterior probability $p_k(x; w_k) = P(Y = k|x; w_k)$ is maximum [73].

$$Entropy_{x_i} = -\sum_{k=1}^{K} I\{y^{(i)} = k\} log(\frac{e^{w_k^T\ x_i}}{\sum_{j=1}^{K} e^{w_j^T\ x_i}}) \tag{40}$$

$$J(w) = -\sum_{i=1}^{N}\sum_{k=1}^{K} I\{y^{(i)} = k\} log(\frac{e^{w_k^T\ x_i}}{\sum_{j=1}^{K} e^{w_j^T\ x_i}}) \tag{41}$$

$$\nabla J(W) = \begin{bmatrix} \frac{\partial J}{\partial w_1} \\ \frac{\partial J}{\partial w_2} \\ \vdots \\ \frac{\partial J}{\partial w_K} \end{bmatrix} = \begin{bmatrix} \frac{\partial J}{\partial w_{10}} & \frac{\partial J}{\partial w_{11}} & \dots & \frac{\partial J}{\partial w_{1p}} \\ \frac{\partial J}{\partial w_{20}} & \frac{\partial J}{\partial w_{21}} & \dots & \frac{\partial J}{\partial w_{2p}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial J}{\partial w_{K0}} & \frac{\partial J}{\partial w_{K1}} & \dots & \frac{\partial J}{\partial w_{Kp}} \end{bmatrix} \tag{42}$$

where

$$\frac{\partial J}{\partial w_k} = \sum_{i=1}^{N} (I\{y^{(i)} = k\} - \frac{e^{w_k^T x_i}}{\sum_{j=1}^{K} e^{w_j^T x_i}})x_i \tag{43}$$

$$w_k^{'} = w_k + \eta \frac{1}{m} \sum_{i=1}^{N} (I\{y^{(i)} = k\} \; - \; \frac{e^{w_k^T x_i}}{\sum_{j=1}^{K} e^{w_j^T x_i}}) x_i \tag{44}$$

5 Non-parametric methods for regression and classification

5.1 Introduction

In this chapter are presented extensively non-parametric methods for regression and classification. More specific are presented Decesion trees and K-neighbors since these methods are used in the field of biomarkers discovery from presicion medicine as mentioned in the literature review.

5.2 Decision Trees

Decision Trees are a non-parametric supervised learning method, capable of finding complex nonlinear relationships in the data. They can perform both classification and regression tasks. The computational cost of making a tree is low and the cost of using it is even lower. Classification using decision trees has grown in popularity over recent years. The idea of a decision tree is that is divides the input space into a set of choices about each feature in turn, starting at the root (base) of the tree and progressing down to the leaves or regions, where we receive the classification decision. The trees can even be turned into a set of if-then rules [79].

characteristics of a tree

In figure 20 is illustrated a classification tree trained on the IRIS dataset (flower species). Root (brown) and decision (blue) nodes contain questions which split into sub nodes. The root node is just the topmost decision node. The leaf nodes (green), also called terminal nodes, are nodes that do not split into more nodes. Classes assigned on the leaf nodes by majority vote i.e. the majority class of the instances in a leaf node. To use a classification tree, start at the root node (brown), and traverse the tree until a leaf (terminal) node is reached [80].

Pure node

A node is considered pure when most of the data points are correctly classified. The data is repeatedly split according to predictor variables so that nodes will be as pure as possible. Obviously, there is no need to split further on pure node. On the contrary, when the heterogeneity at a particular node is high, it is considered impure and such a node needs further splitting [79].

Tree depth and overfitting

Tree depth is a measure of how many splits a tree can make before coming to a prediction. In the figure 20, the tree depth is 2 and in figure 21 are presented different trees in depth [80]. Classification tree is a greedy algorithm which means that is introduced only one split at a time, without having the full tree 'in mind'. This process could be continued further with more splitting until the tree is as pure as possible. The problem of having many repetitions with this process is that they can lead to a very deep classification tree with many nodes. This often leads to overfitting on the training dataset. Decision trees are the most susceptible out of all the machine learning algorithms to overfitting. A large tree risks to overfit in the training data resulting in poor generalization ability to new samples. A small tree might not capture important structural information about the sample space. There are several ways to handle overfitting and select the appropriate size of the tree. Some of them will be analyzed below [79].

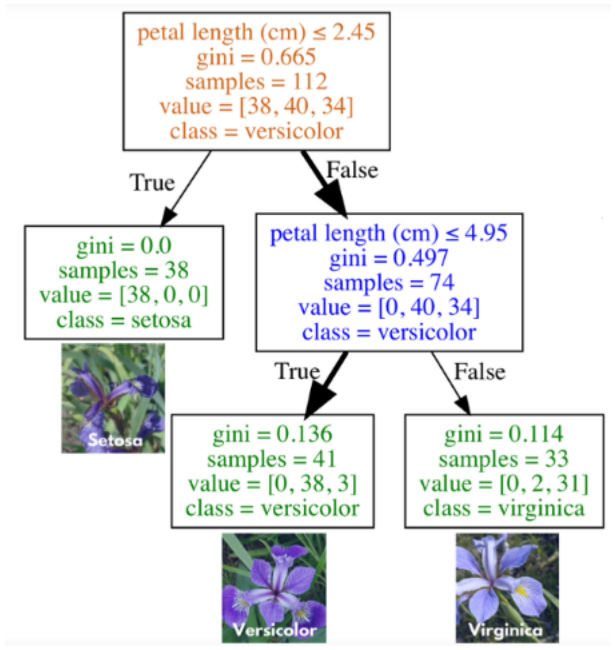


Figure 20: Classification tree to classification one of three flower species (IRIS Dataset) [80].

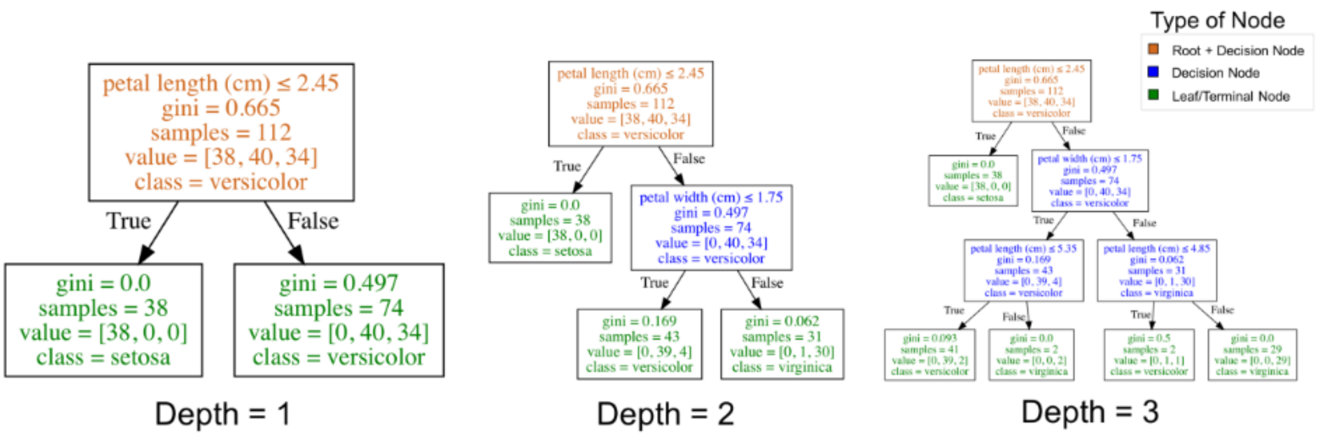


Figure 21: Classification trees of different depths fit on the IRIS dataset [80].

5.2.1 Entropy

The base of entropy comes from physics, where it is defined as the measurement of disorder, randomness, unpredictability, or impurity in the system. Similarly, in Machine Learning entropy (45) is the metric that measures the randomness,unpredictability or impurity in the data set. Every piece of information has a specific value and can be used to draw conclusions from it. The easier it is to draw useful results from a piece of information, the lower the entropy will be. An example for better understanding can be: flipping a coin. When a coin is flipped there can be two outcomes. It is difficult to predict the exact outcome while flipping a coin because there is a 50% probability of both outcomes. Is a case like this, entropy would be high .Given a data set having N classes, the mathematical formula of entropy is equation 4 where p_i is probability of randomly selecting an example in class i [81]. Figure 22 represents the change in entropy as the proportion of the number of instances belonging to a particular class change [79].

$$Entropy(p) = -\sum_{i=1}^{C} p_i log_2 p_i \tag{45}$$

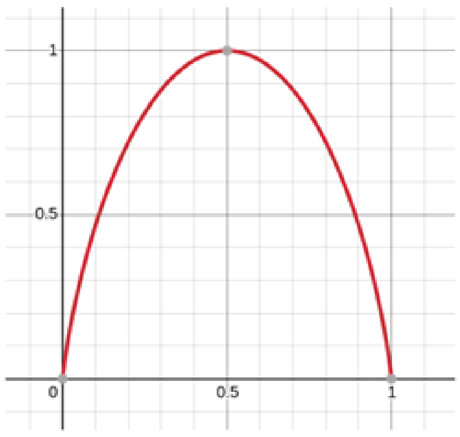


Figure 22: Change in entropy according to the proportion of the number of instances belonging to a particular class [79].

5.2.2 Iterative Dichotomiser 3 Algorithm

The most common algorithm for classification is Quinlan's Iterative Dichotomiser 3 (ID3). When building a decision tree using ID3, the best feature to pick is the one that gives the most information. This is quantified using entropy. The idea is to work out how much the entropy of the whole training set would be decreased if each feature is chosen for the next classification step. This is known as the information gain and it is defined as the entropy of the whole set minus the entropy when a particular feature is chosen.

$$Gain(S,F) = Entropy(S) - \sum_{f \in values(F)} \frac{|S_f|}{|S|} Entropy(S_f) \quad (46)$$

where S is the set of examples, F is a possible feature out of the set of all possible ones, and $|S_f|$ is a count of the number of members of S that have value f for feature F.

The ID3 algorithm computes the information gain for each feature and chooses the one that yields the highest value. This feature yields maximum reduction in entropy. Splits are not necessarily binary in this case. At each stage the best feature is selected and then is removed from the dataset and the algorithm is recursively called on the rest. The recursion stops when either there is only one class remaining in the data (in which case a leaf is added with that class as its label), or there are no features left, when the most common label in the remaining data is used.

The algorithm can deal with noise in the dataset because the labels are assigned to the most common value of the target attribute. Another benefit of decision trees is that they can deal with missing data [79].

5.2.3 C4.5 Algorithm

ID3 algorithm uses all the features that are available, even if some of them are not necessary, increasing the risk of overfitting. Hence, Quinlan constructed also an improved algorithm called C4.5. In this algorithm the tree is created using ID3 algorithm. However, to avoid overfitting a method called post-pruning is used [79]. In this process the whole tree is constructed first and then the non- significant rules are removed [82].

Additionally C4.5 can handle continuous variables by discretizing them [83]. To this end, the input variables must be split and thus, they are separated. More often, only one split is made in the continuous variable, but this is not obligatory. Decision trees split on the feature and corresponding split point that results in the largest information gain (IG) for a given criterion. Therefore, the algorithm calculates the information gain of many points within each variable. A good value for a split point does a good job of separating one class from the others [79].

5.2.4 C4.5 Algorithm-Numerical Example

An example for building a decision tree ”by hand” is presented below for the training set T given in the following table. T has 14 data points, 3 features and a target binary variable. Attributes x_1 and x_3 have nominal values, while attribute x_2 has numeric values.

Starting at the root level, given this training set T, it must be determined the first of the 3

Table 23: Top-ranking blood indices for the identification of lung cancer.

Sample	x_1	x_2	x_3	Class
1	A	70	true	C_1
2	A	90	true	C_2
3	A	85	false	C_2
4	A	95	false	C_2
5	A	70	false	C_1
6	B	90	true	C_1
7	B	78	false	C_1
8	B	65	true	C_1
9	B	75	false	C_1
10	C	80	true	C_2
11	C	70	true	C_2
12	C	80	false	C_1
13	C	80	false	C_1
14	C	96	false	C_1

attributes that will be used to form the node at the root. Since 9 samples belong to class C_1 and the remaining 5 samples to C_2, the entropy of the whole set is

$$info(T) = -\frac{9}{14}log_2(\frac{9}{14}) - \frac{5}{14}log_2(\frac{5}{14}) = 0.94$$

First, is tested the feature x_1 to split T into 3 subsets say T_1, T_2, and T_3, containing samples with x_1 equal to A, B, and C, respectively. T_1 has 5 samples, 2 are in C_1 and 3 in C_2, and so its entropy is

$$info(T_1) = -\frac{2}{5}log_2(\frac{2}{5}) - \frac{3}{5}log_2(\frac{3}{5}) = 0.971$$

T_2 has 4 samples, all are in C_1 and none inC_2, and so its entropy is

$$info(T_2) = -\frac{4}{4}log_2(\frac{4}{4}) - \frac{0}{4}log_2(\frac{0}{4}) = 0$$

T_3 has 5 samples, 3 are in C_1 and 2 in C_2, and so its entropy is

$$info(T_3) = -\frac{3}{5}log_2(\frac{3}{5}) - \frac{2}{5}log_2(\frac{2}{5}) = 0.971$$

Thus after this potential split, the resulting entropy is

$$info_{x_1}(T) = \frac{5}{14}info(T_1) + \frac{4}{14}info(T_2) + \frac{5}{14}info(T_3) = 0.694$$

The information gain (or loss in entropy) if the set is split using attribute x_1 is

$$Gain(x_1) = 0.940 - 0.694 = 0.246$$

With the same way, the information gain (or loss in entropy) if the set is split using attribute x_2 can be calculated and will be $Gain(x_2) = 0.940 - 0.838 = 0.102$ and using attribute x_3 will be $Gain(x_3) = 0.940 - 0.892 = 0.048$.

The highest gain is computed using x_1, and therefore this attribute is selected for the first splitting in the construction of a decision tree.

Next, each of the three subnodes are considered separately. Splitting T_1 having 2 C_1 samples and 3 C_2 samples the entropy is

$$info(T_1) = -\frac{2}{5}log_2(\frac{2}{5}) - \frac{3}{5}log_2(\frac{3}{5}) = 0.971$$

Choosing the test attribute x_1, it turns out the optimal threshold value is z = 70 and this optimal test can be denoted as x_4. This choice of z splits T_1 into 2 subsets. The first subset, consisting of 2 samples with $x_2 \leqslant 70$, has all the 2 samples in C_1 and none in C_2. The second subset, consisting of 3 samples with $x_2 > 70$, has all the 3 samples in C_2 and none in C_1. The resulting information is

$$info_{x_4}(T_1) = \frac{2}{5}[-\frac{2}{2}log_2(\frac{2}{2}) - \frac{0}{2}log_2(\frac{0}{2})] + \frac{3}{5}[-\frac{0}{3}log_2(\frac{0}{3}) - \frac{3}{3}log_2(\frac{3}{3})] = 0$$

The information gained by this test is

$$Gain(x_4) = 0.971 - 0 = 0.971$$

The 2 branches created by this split will be the final leaf nodes since the subsets of samples in each of the branches all belong to their separate classes.

Next, T_2 must be splitted. However since all 4 samples in T_2 belong to C_1, thus this node will be a leaf node, and no additional tests are necessary for this branch.

The last subset T_3, whose entropy is

$$info(T_3) = -\frac{3}{5}log_2(\frac{3}{5}) - \frac{2}{5}log_2(\frac{2}{5}) = 0.971$$

Testing the attribute x_3, T_2 is separated into 2 subsets. The first subset, consisting of 2 samples with x_3 = True, has no sample in C_1 and all 2 samples in C_2. The second subset, consisting of 3 samples with x_3 = False, has all the 3 samples in C_1 and none in C_2. The resulting information is

$$info_{x_3}(T_3) = \frac{2}{5}[-\frac{2}{2}log_2(\frac{0}{2}) - \frac{2}{2}log_2(\frac{2}{2})] + \frac{3}{5}[-\frac{3}{3}log_2(\frac{3}{3}) - \frac{0}{3}log_2(\frac{0}{3})] = 0$$

where this test have been denoted as x_5. The information gained by x_3

$$Gain(x_3) = 0.971 - 0 = 0.971$$

is the best. This test results in 2 uniform subsets of samples of the 2 separate classes, and therefore yields 2 final leaf nodes for this branch. The final decision tree for T is now determined. It can then be used to classify any new unseen sample [84].

5.2.5 CART algorithm

Trees can also be used for regression. If the target attribute is continuous, the resulting tree is called a regression tree. In general, data consists of p features and a response variable, for each of N data samples: that is, (x_i, y_i) for $i = 1, 2, ..., N$ where $x_i = (x_{i1}, x_{i2}, ..., x_{ip})$. A popular method for tree-based regression and classification is called CART. The decision tree in the left graph in figure 23 results in the 5 partitions in the right diagram of the same figure. Assuming that is given a partition into M regions $R_1, R_2, ..., R_M$, and model the response as a constant $\hat{y}_m$ in each region .Instead of the previous splitting criteria it could be used the mean squared error minimization which in each one of m regions will be [85]:

$$Q_m(T) = \frac{1}{n_m} \sum_{x_i \in R_m} (y_i - \hat{y}_m)^2 \tag{47}$$

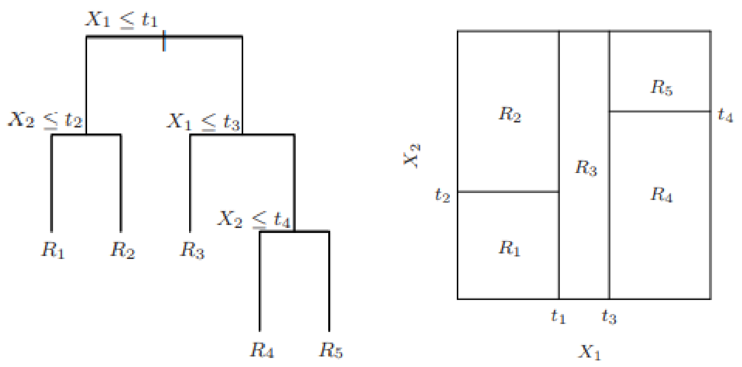


Figure 23: Partitions and Cart [85].

Since the criterion will be the minimization of the sum of square $\sum (y_i - \hat{y}_m)^2$, then the best prediction $\hat{y}_m$ is the average of y_i in region R_m:

$$\hat{y}_m = \bar{y}_m = \frac{1}{n_m} \sum_{x_i \in R_m} y_i \tag{48}$$

Finding the best tree T that minimizes (47) is a combinatorial problem and hence computationally infeasible. Hence, a greedy algorithm is used again. Starting using all data, a splitting variable j and split point s are considered creating two regions

$$R_1(j, s) = \{X|X_j \leq s\} \textit{ and } \{R_2(j, s) = X|X_j > s\}$$

Then are seeked the splitting variable j and split point s that solve

$$\min_{j,s}\{\min_{c_1} \sum_{x_i \in R_1(j,s)} (y_i - \hat{y}_1)^2 + \min_{c_2} \sum_{x_i \in R_2(j,s)} (y_i - \hat{y}_2)^2\} \tag{49}$$

Like before, $\hat{y}_1$ is the average of responses y_i in region R_1 and $\hat{y}_2$ is the average of responses y_i in region R_2: [85]

$$\hat{y}_1 = \frac{1}{n_1} \sum_{x_i \in R_1(j,s)} y_i, \quad \hat{y}_2 = \frac{1}{n_2} \sum_{x_i \in R_2(j,s)} y_i \tag{50}$$

In summary, what the algorithm does is to search for each variable the split point for which the sum of the sum of squares of the two regions is minimized and not the point for which both sum of squares are minimized. Among all variables, the one that minimizes the sum of the sum of squares is chosen to separate the data into to regions. Having found the best split point, the data is separated into the two resulting regions (binary splitting) and the splitting process is repeated to each of the two regions. Then this process is repeated to all the resulting regions.

If the target is a classification outcome taking values $1, 2, ..., K$, the only changes needed in the tree algorithm pertain to the criteria for splitting nodes. In a node m, representing a region R_m with N_m observations can be denoted the proportion of class k observations in node m like following [85].

$$\hat{p}_{mk} = \frac{1}{N_m} \sum_{x_i \in R_m} (y_i = k) \tag{51}$$

The observations are classified in node m to class $k(m) = argmax_k \hat{p}_{mk}$ which is the majority class in node m.

Tuning tree size

Tree size is determined during the training process. Firstly, a large tree T_0 is created. Next, T_0 is pruned using cost-complexity pruning. A subtree $T \subset T_0$ can be any tree created by pruning T_0, that is, deleting some of its internal (non-terminal) nodes. The idea is to find for each α, the subtree $T_\alpha \subset T_0$ to minimize $C_\alpha(T)$ which is called cost complexity criterion.

$$C_\alpha(T) = \sum_{m=1}^{|T|} N_m Q_m(T) + \alpha|T| \tag{52}$$

Large values of α result in smaller trees T_α, and conversely for smaller values of α. For each α there is a unique smallest subtree T_α that minimizes $C_\alpha(T)$. The $T_{\hat{\alpha}}$ that minimizes the sum of squares using five- or tenfold cross-validation is the final pruned tree.

Different measures $Q_m(T)$ of node impurity include the following [85]

- Misclassification error: $\frac{1}{N_m} \sum_{i \in R_m} I(y_i \neq k(m))$
- Gini index: $\sum_{k \neq k'} \hat{p}_{mk} \hat{p}_{mk'}$
- Entropy:$- \sum_{k=1}^{K} \hat{p}_{mk} log_2 \hat{p}_{mk'}$

Cross-entropy and the Gini index are more sensitive to changes in the node probabilities than the misclassification rate. To guide cost-complexity pruning, any of the three measures can be used, but typically it is the misclassification rate [85].

5.3 K-Nearest Neighbor

K-Nearest Neighbor (KNN) is a very simple non-parametric supervised classification algorithm which can be used for both regression and classification tasks. KNN uses all the available data classifies an unknown data point based on how its neighbors of a given data set are classified. To find the nearest neighbors a similarity measure is used. A typical measure is Euclidian distance. The Euclidian distance of unknown data sample from all the points in the training set must be calculated to identify the K nearest points to the unknown data. For classification tasks the test point is classified to the category that gets the most votes. For regression tasks the output is the mean of the k nearest points [86].

K in KNN is a parameter that refers to the number of nearest neighbors that are included in the majority voting process. Choosing the right value for K is important for better accuracy. The K must be an odd parameter to avoid as much as possible ties in the votes between two or more categories [87]. For the same reason the K must not be a multiple of the number of classes. If it is still got a tie, it is up to the user to classify randomly into one of the tied categories or decide not to assign the test point into a category. Low values for K ($K = 1, K = 2$) can be noisy and subject to the effects of outliers [88] . Also, the K value should not be very large because data points that belong to other groups can be considered and misclassification will start occurring [89].

There is no standard way to determine the optimal value for K. One common is to choose the value for k as $\sqrt{N}$ where N is the total number of data points. One other effective way is K-fold cross validation. The value K that leads to the best results can be used for the application [86].

KNN works very well in small data sets. In the case of big data set the main drawback of KNN is the complexity in searching the nearest neighbors for each sample. This is because there are lots of elements and the distance between each element of the training set and the new element that must be computed [89].

5.3.1 KNN Numerical Example

An example for building a K-Nearest Neighbor ”by hand” is presented below for the training set given in the following table. BMI and Age are the independent variable and the target variable is Sugar having two possible outcomes (Diabetic or Non Diabetic).

Table 24: Training data.

BMI	Age	Sugar
33.6	50	1
26.6	31	0
23.3	32	1
28.1	21	0
43.1	33	1
25.6	30	0
31	26	1
35.3	29	0
36.5	53	1
37.6	30	0

The choice for k was $k = 3$ since are given little data. The in the below table was chosen for testing.

Table 25: Test data.

BMI	Age	Sugar
43.3	33	0
34.6	32	1
39.8	41	1
22.2	57	0

What is the prediction for a person having $BMI = 43.3$ and $age = 33$? The original answer is not as it seems in the dataset.

First the distance between point (43.3,33) and 33.6 and 50 is calculated using the Euclidean distance. The distance between point(43.3,33) to each point in data-set must be calculated and for the other points the same process is repeated.

$$Distance = (43.3–33.6)^2 − (33–50)^2 = 383.09$$

In distance formula, square root is unnecessary for this case. The output of all calculated distances from point (43.3,33) is presented in table 18 and the minimum must be chosen. It is 9.04, 32.49, 65. Their values in the target variable are 2 non-diabetic and 1 is diabetic respectively. So, the new patient is classified as non diabetic. The original output is also Non-diabetic. Therefore the prediction was correct [90].

Table 26: Table with distances.

BMI	Age	Sugar	Distance
33.6	50	1	383.09
26.6	31	0	279.89
23.3	32	1	404
28.1	21	0	312.04
43.1	33	1	9.04
25.6	30	0	313.29
31	26	1	167.29
35.3	29	0	65
36.5	53	1	575.24
37.6	30	0	32.49

6 Generative Learning Algorithms

6.1 Introduction

Generative approaches is a way of telling a story about the origin of the data. A model can be considered as a “profile” of a class. Generative algorithms make a strong assumption on the data, assuming that the data is distributed as multi-variate Gaussian. Naive Bayes (another generative algorithm) assumes that each feature is uncorrelated with other features in the data. On the contrary, discriminative algorithms make weak assumptions on the data [91].

6.2 Bayes’ Rule

The conditional probability of a class C given that a feature X has value x $P(C|x)$ according to Baye’s rule will be [92]

$$P(C|x) = \frac{P(x|C)P(C)}{P(x)} \tag{53}$$

Bayes Rule is a way of going from the probability $P(x|C)$, known from the training dataset and much easier to compute, to find $P(C|x)$ [93]. It is one of the most important equations in machine learning. The denominator (the term on the bottom of the fraction) acts to normalize everything, so that all the probabilities sum to 1 [92]. Baye’s rule is a source of inspiration for many other methods which have wide applications in real world data.

In case that x vector of feature values contains more than one features then according to Baye’s rule the conditional probability equals:

$$P(C|x) = P(C|x_1, x_2, ..., x_p) = \frac{P(x_1, x_2, ..., x_p|C)P(C)}{P(x_1, x_2, ..., x_p)} \tag{54}$$

and can be used to assign a new observation to a class C_i for which

$$P(C_i|x) > P(C_j|x),\ \forall i \neq j \tag{55}$$

where $x = (x_1, \ldots, x_p)$ is a vector of p feature values. This is known as the maximum a posteriori or MAP hypothesis and provides a way to choose a class as the output. [92]

6.2.1 Naïve Bayes' Classifier

Naive bayes' comes from the assumption that the elements of a feature vector $x = (x_1, \ldots, x_p)$ are conditionally independent of each other. Therefore, the string of feature values $P(x_1 = a_1, x_2 = a_2, ..., x_p = a_p|C_i)$ is just equal to the product of multiplying together all of the individual probabilities [92]:

$$P(x_1 = a_1, x_2 = a_2, ..., x_p = a_p|C_i) = P(x_1 = a_1|C_i) \times P(x_2 = a_2|C_i) \times ... \times P(x_p = a_p|C_i) \quad (56)$$

This simplification of Bayes Theorem is common and widely used for classification predictive modeling problems and is generally referred to as *Naive Bayes*. The classifier rule is to select the class C_i for which the following computation is the maximum [93]

$$P(C_i|x_1 = a_1, x_2 = a_2, ..., x_p = a_p) = \frac{P(C_i) \prod_k P(x_p = a_p|C_i)}{Z} \quad (57)$$

where $Z = P(x_1) \times P(x_2) \times ... \times P(x_p)$

Probability Z depended only on $x_1, \ldots, x_p$ features and is the same for all classes C_i, thus this probability can be dropped [94]. Finally, the classifier rule is to select the class C_i for which it is maximized the: [92]

$$P(C_i|x_1 = a_1, x_2 = a_2, ..., x_p = a_p) = P(C_i) \prod_p P(x_k = a_k|C_i) \quad (58)$$

Naive Bayes is a probabilistic machine learning algorithm. Although it seems to be a simple yet is a powerful algorithm. The algorithm can be coded up easily and the predictions are made quickly [93]. When the simplification is true, so that the features are conditionally independent of each other, the naïve Bayes' classifier produces exactly the MAP classification [92].

6.2.2 Naive Bayes Numerical Example

An example for building a Naive Bayes Algorithm "by hand" is presented below for the training set given in the following table. Long, Sweet and Yellow are the independent variables and the target variable is the fruit (Orange, Banana, Other). So the objective of the classifier is to predict if a given fruit is a 'Banana' or 'Orange' or 'Other' using the 3 features.

The algorithm will be used for the prediction of a fruit that is: Long, Sweet and Yellow. The training data can be aggregated to a table like following.

To begin with, the 'Prior' probabilities for each of the class of fruits must be computed. That is, the proportion of each fruit class out of all the fruits from the population. Out of 1000 records in training data, are given 500 Bananas, 300 Oranges and 200 Others. Therefore, priors are 0.5,

Table 27: Training data.

Fruit	Long (x1)	Sweet (x2)	Yellow (x3)
Orange	0	1	0
Banana	1	0	1
Banana	1	1	1
Other	1	1	0
..	..	..	..

Table 28: Training data.

Type	Long	Not Long	Sweet	Not Sweet	Yello	Not Yellow	Total
Banana	400	100	350	150	450	50	500
Orange	0	300	150	150	300	0	300
Other	100	100	150	50	50	150	200
Total	500	500	650	350	800	200	1000

0.3 and 0.2 respectively. The probability that goes in the denominator are $P(x_1 = Long) = \frac{500}{1000} = 0.50$, $P(x_2 = Sweet) = \frac{650}{1000} = 0.65$, $P(x_3 = Yellow) = \frac{800}{1000} = 0.8$. However, their computation is not necessary since they are the same for all the classes and so will not affect the probabilities. Subsequently, the conditional probabilities that contained in the numerator of equation (57) must be computed .

Probability of Likelihood for Banana will be $P(x_1 = Long|Y = Banana) = \frac{400}{500} = 0.80, P(x_2 = Sweet|Y = Banana) = \frac{350}{500} = 0.70, P(x_3 = Yellow|Y = Banana) = \frac{450}{500} = 0.90$. So, the overall probability of Likelihood for Banana = 0.8 * 0.7 * 0.9 = 0.504. With the same way, are calculated the probabilities for Orange and Other. Substituting the probabilities into the Naive Bayes formula, is concluded that the data point is classified as Banana since it gets the highest probability [95].

$P(Banana|Long, Sweet\ and\ Yellow) = \frac{P(Long|Banana)P(Sweet|Banana)P(Yellow|Banana)P(Banana)}{P(Long)P(Sweet)P(Yellow)} = \frac{0.8\ 0.7\ 0.9\ 0.5}{P(Long)P(Sweet)P(Yellow)}$

$P(Orange|Long, Sweet\ and\ Yellow) = \frac{P(Long|Orange)P(Sweet|Orange)P(Yellow|Orange)P(Orange)}{P(Long)P(Sweet)P(Yellow)} = 0$ because $P(Long|Orange) = 0$.

$P(Other\ Fruit|Long, Sweet\ and\ Yellow) = \frac{P(Long|Other\ Fruit)P(Sweet|Other\ Fruit)P(Yellow|Other\ Fruit)P(Other\ Fruit)}{P(Long)P(Sweet)P(Yellow)} = \frac{0.01875}{P(Long)P(Sweet)P(Yellow)}$

6.3 Linear Discriminant Analysis

Linear discriminant analysis (LDA) is particularly popular classification method and is based on Bayes' rule. An equivalent expression of Bayes' Rule is the following:

$$P(y = k|X = x) = \frac{f_k(x)\pi_k}{\sum_r f_r(x_0)\pi_r} \tag{59}$$

where $f_k(x_0)$ is the density of X conditioned on k and π_k is the prior probability of class with $\sum_{k=1}^{K} \pi_k = 1$.The denominator is identical for all classes.Then, as described before, according to the maximum-a-posterior (MAP) a new observation is assigned to class k if

$$G(x) = argmax_k P(y = k|X = x) = argmax_k f_k(x)\pi_k \tag{60}$$

In Linear discriminant analysis (LDA) it is assumed that the class conditional distribution is multivariate gaussian and the classes have a common covariance matrix $\Sigma^k = \Sigma$ ∀k.

This means that

$$f_k(x) = \frac{1}{(2\pi)^{\frac{d}{2}}|\Sigma|^{\frac{1}{2}}} e^{\frac{-1}{2}(\boldsymbol{x}-\mu_k)^T\Sigma^{-1}(\boldsymbol{x}-\mu_k)} \tag{61}$$

Consequently,the decision rule in LDA will be

$$G(\boldsymbol{x}) = argmax_k \delta_k(\boldsymbol{x}) \tag{62}$$

where

$$\delta_k(\boldsymbol{x}) = \boldsymbol{x}^T\Sigma^{-1}\mu_k - \frac{1}{2}\mu_k^T\Sigma^{-1}\mu_k + log\pi_k$$

where $\delta_k(\boldsymbol{x})$ are called linear discriminant functions for class k. Therefore, new points are classified by computing the discriminant function $\delta_k(\boldsymbol{x})$ and returning the class k with the maximum $\delta_k(\boldsymbol{x})$.

For a random pair of two classes, say k,l, the decision boundary is a set of points such that probability of being on k class is the same as probability of being on l class. So, decision boundary based on the two simplification assumptions is a set of points x such that [96]:

$$D.B = \{\boldsymbol{x}|P(y = k|X = x) = P(y = l|X = \boldsymbol{x})\} \iff$$

$$\boldsymbol{x}^T(\Sigma^{-1}\mu_k - \Sigma^{-1}\mu_l) + \frac{1}{2}(\mu_l^T\Sigma^{-1}\mu_l - \mu_k^T\Sigma^{-1}\mu_k) + log(\frac{\pi_k}{\pi_l}) = 0 \tag{63}$$

Geometrically, in 2-dimensional space decision boundary is a line and in p-dimensional space is a p-dimensional hyperplane. The parameters are not known, thus they must be estimated using the training data as following [97]

$$\hat{\pi}_k = \frac{N_k}{N} \tag{64}$$

$$\hat{\mu}_k = \sum_{g_i=k} \frac{x_i}{N_k} \tag{65}$$

$$\hat{\Sigma} = \sum_{k=1}^{g} \frac{n_k \Sigma_k}{N} \tag{66}$$

where Σ_k is the covariance matrix of class k, n_k is the number of samples belong to class k and N the total number of data.

6.3.1 Quadratic Discriminant Analysis

Quadratic Discriminant Analysis (QDA) is a variant of LDA in which an individual covariance matrix is estimated for every class of observations rather than assuming that classes have common covariance matrix .The assumption that class conditional distribution is multivariate gaussian does not change.

Thus, is required to estimate Σ_k for each class $k \in 1, \ldots, K$ .The discriminant function of QDA is quadratic in x:

$$\delta_k(\boldsymbol{x}) = -\frac{1}{2} log|\Sigma_k| - \frac{1}{2}(\boldsymbol{x} - \mu_k)^T \Sigma_k^{-1} (\boldsymbol{x} - \mu_k) + log\pi_k \tag{67}$$

Similar to LDA, an observation can be classified based on largest quadratic classification function score. The decision boundary between each pair of classes k and l is the set of points for which

$$\{x : \delta_k(x) = \delta_l(x)\} \iff x^T(\Sigma_0^{-1}\mu_0 - \Sigma_1^{-1}\mu_1) - \frac{1}{2}x^T(\Sigma_0^{-1} + \Sigma_1^{-1})x =$$
$$log\pi_1 - log\pi_0 - \frac{1}{2}log|\Sigma_1| + \frac{1}{2}log|\Sigma_0| - \frac{1}{2}(\mu_1^T \Sigma^{-1}\mu_1 - \mu_0^T \Sigma^{-1}\mu_0) \tag{68}$$

This is a quadratic form,thus the decision boundary for QDA is always quadratic [96].

The estimates for QDA are similar to those for LDA, except that separate covariance matrices must be estimated for each class using the following expression:

$$\hat{\Sigma}_k = \frac{1}{N_k - 1} \sum_{i:y_i=k} (\boldsymbol{x_i} - \mu_k)(\boldsymbol{x_i} - \mu_k)^T \tag{69}$$

where n_k is the number of class k observations [98].

6.3.2 Computations for LDA and QDA

The computations required for LDA and QDA can be significantly simplified through a linear transformation. The quantity δ_k in equation (67) contain the terms $\frac{1}{2}log\pi_k$ which can be considered as a constant and the terms $-\frac{1}{2}log|\Sigma_k|$ and $(x-\mu_k)^T\Sigma_k^{-1}(x-\mu_k)$ which is also known as Mahalanobis distance [99].

Suppose that $\Sigma_k = I$ for each k. In this case the data are spherical and the term $(x-\mu_k)^T I(x-\mu_k)$ will be the square Euclidian distance between point x and μ_k. Also, $log\Sigma_k = 0$. So, in this case a point x is classified to class k simply by compared the distances of x from the mean of each class and classified to the one which is closer.

Suppose now that for some classes $k : \Sigma_k \neq I$ which means that the data in these classes are not spherical. The covariance matrix Σ_k of each class k can be decomposed using singular value decomposition as

$$\Sigma_\mathrm{k} = U_k S_k V_k^T \tag{70}$$

Since Σ_k is covariance matrix, is symmetric and orthonormal. Therefore, $\Sigma_k\Sigma_k^T = \Sigma_k^T\Sigma_k$ and $\Sigma_k^{-1} = \Sigma_k^T$, so $U_k = V_k$ so Σ_k can be decomposed as

$$\Sigma_k = U_k V_k U_k^T \tag{71}$$

and

$$\Sigma_k^{-1} = (U_k S_k U_k^T)^{-1} = (U_k^T)^{-1} S_k^{-1} U_k^T = U_k S_k^{-1} U_k^T \tag{72}$$

Substituting this in term $(x-\mu_k)^T\Sigma_k^{-1}(x-\mu_k)^T$ arises that

$$\begin{gathered}(x-\mu_k)^T U_k S_k^{-1} U_k^T (x-\mu_k) = (U_k^T x - U_k^T \mu_k^T) S_k^{-1} (U_k^T x - U_k^T \mu_k) = \\ (S_k^{-1/2} U_k^T x - S_k^{-1/2} U_k^T \mu_k^T)(S_k^{-1/2} U_k^T x - S_k^{-1/2} U_k^T \mu_k)\end{gathered} \tag{73}$$

which is a transformed Euclidian distance between transformed point and transformed mean in class k. Hence, in the case that the data in k class are not spherical ($\Sigma_k \neq I$) for k=1,..,K the linear transformation $X \rightarrow S_k^{1/2} U_k^T x$ can be applied in the data points of class k and make them spherical. The mean of the transformed data in class k will be linearly changed and their covariance matrix will be identity, no matter what the shape of the original data in class k is. However, as discussed earlier in the case that the data are spherical is quite easy to classify a new data point just by measuring the distance between the transformed data point and the mean of each class. In this case, to classify a new data point, all the transformations for k classes are applied one by one to this point and each time is measured the distance between the transformed data point and the mean of the class that is related to the transformation.The data point is classified to the class with the minimum of these distances.

The same process is done in LDA but since the classes have common covariance matrices the linear transformation that is applied to the entire data is the same. The transformation is $X \rightarrow S^{-1/2}U^{T}x$ and make the data spherical.To classify a new data point, the linear transformation is applied to this point and the distance between the transformed data point and the mean of each class is measured. The data point is classified to the class with the minimum distance [99].

6.3.3 LDA Numerical Example

An example for building a Linear Discriminant Algorithm "by hand" is presented below for the training set given in the following table. The data contain only 7 samples, two independent variables named x_1 and x_2 and one binary target variable [100].

Table 29: Training data.

x_1	x_2	Target
2.95	6.63	Passed
2.53	7.79	Passed
3.57	5.65	Passed
3.16	5.47	Passed
2.58	4.46	Not Passed
2.16	6.22	Not Passed
3.27	3.52	Not Passed

As mentioned in theory, a data point is classified based on linear discriminant functions in (62). To calculate them, a data matrix $\mathbf{X}$ is constructed containing the two independent features

$$\mathbf{X} = \begin{bmatrix} 2.95 & 6.63 \\ 2.53 & 7.79 \\ 3.57 & 5.65 \\ 3.16 & 5.47 \\ 2.58 & 4.46 \\ 2.16 & 6.22 \\ 3.27 & 3.52 \end{bmatrix}$$

where x_k represents the data of k row. Also, $\mu_i,\ i = 1, 2$ will be the mean of features belong to class i and easily calculated that

$\mu_1 = \begin{bmatrix} 3.05 & 6.38 \end{bmatrix}$ and $\mu_2 = \begin{bmatrix} 2.67 & 4.73 \end{bmatrix}$

and the global mean vector will be

$\mu = \begin{bmatrix} 2.88 & 5.676 \end{bmatrix}$

The centralized matrix is created subtracting from data matrix X the global mean vector

$$\mathbf{B} = \begin{bmatrix} 0.06 & 0.951 \\ -0.357 & 2.109 \\ 0.679 & -0.025 \\ 0.269 & -0.209 \\ -0.305 & -1.218 \\ -0.732 & 0.547 \\ 0.386 & -2.155 \end{bmatrix}$$

To calculate the pooled covariance matrix, the covariance matrix of each group is required at first. It is computed using only the data of the group and not the whole data. Therefore, the centralized data matrix is separated into to matrices, one for each group.

$$\mathbf{B}_1 = \begin{bmatrix} 0.06 & 0.951 \\ -0.357 & 2.109 \\ 0.679 & -0.025 \\ 0.269 & -0.209 \end{bmatrix}$$

$$\mathbf{B}_2 = \begin{bmatrix} -0.305 & -1.218 \\ -0.732 & 0.547 \\ 0.386 & -2.155 \end{bmatrix}$$

The covariance matrix for group i will be $\Sigma_i = \frac{(B_i)^T B_i}{n_i}$

$$\Sigma_1 = \begin{bmatrix} 0.166 & -0.192 \\ -0.192 & 1.349 \end{bmatrix}$$

$$\Sigma_2 = \begin{bmatrix} 0.259 & -0.286 \\ -0.286 & 2.142 \end{bmatrix}$$

The pooled covariance matrix is calculated by $\hat{\Sigma} = \sum_{k=1}^{g} \frac{n_k \Sigma_k}{N}$ as shown in equation (66). In this example will be

$$\Sigma = \begin{bmatrix} \frac{4}{7}0.166 + \frac{3}{7}0.259 & \frac{4}{7}(-0.192) + \frac{3}{7}(-0.286) \\ \frac{4}{7}(-0.192) + \frac{3}{7}(-0.286) & \frac{4}{7}1.349 + \frac{3}{7}2.142 \end{bmatrix} = \begin{bmatrix} 0.206 & -0.233 \\ -0.233 & 1.689 \end{bmatrix}$$

$$\Sigma^{-1} = \begin{bmatrix} 5.745 & 0.791 \\ 0.791 & 0.701 \end{bmatrix}$$

And the priors for each group are calculated as $p = \begin{bmatrix} p_1 \\ p_2 \end{bmatrix} = \begin{bmatrix} \frac{n_1}{N} \\ \frac{n_2}{N} \end{bmatrix} = \begin{bmatrix} \frac{4}{7} \\ \frac{3}{7} \end{bmatrix} = \begin{bmatrix} 0.571 \\ 0.429 \end{bmatrix}$

Since all the terms for the discriminant function (62) have been calculated, it can be used to classify a new data point. Given a new point with $x_1 = 2.81$ and $x_2 = 5.46$ the prediction for the point is “not passed” since the discriminant function for that class is higher as presented below.

$$\delta_1(\boldsymbol{x}) = \boldsymbol{x}^T \Sigma^{-1} \mu_1 - \frac{1}{2} \mu_1^T \Sigma^{-1} \mu_1 + log\pi_1 =$$

$$\begin{bmatrix} 2.81 & 5.46 \end{bmatrix} \begin{bmatrix} 5.745 & 0.791 \\ 0.791 & 0.701 \end{bmatrix} \begin{bmatrix} 3.05 \\ 6.38 \end{bmatrix} - \frac{1}{2} \begin{bmatrix} 3.05 & 6.38 \end{bmatrix} \begin{bmatrix} 5.745 & 0.791 \\ 0.791 & 0.701 \end{bmatrix} \begin{bmatrix} 3.05 \\ 6.38 \end{bmatrix} + log\pi_1 =$$

$$101.0101 - \frac{1}{2} 112.7608 + log(\frac{4}{7}) = 44.05$$

$$\delta_2(\boldsymbol{x}) = \boldsymbol{x}^T \Sigma^{-1} \mu_2 - \frac{1}{2} \mu_2^T \Sigma^{-1} \mu_2 + log\pi_2 =$$

$$\begin{bmatrix} 2.81 & 5.46 \end{bmatrix} \begin{bmatrix} 5.745 & 0.791 \\ 0.791 & 0.701 \end{bmatrix} \begin{bmatrix} 2.67 \\ 4.73 \end{bmatrix} - \frac{1}{2} \begin{bmatrix} 2.67 & 4.73 \end{bmatrix} \begin{bmatrix} 5.745 & 0.791 \\ 0.791 & 0.701 \end{bmatrix} \begin{bmatrix} 2.67 \\ 4.73 \end{bmatrix} + log\pi_2 =$$

$$83.26 - \frac{1}{2} 76.61 + log(\frac{3}{7}) = 44.1$$

7 Support Vector Machines

7.1 Introduction

Support Vector Machines (SVM) is one of the most popular algorithms in modern machine learning. It often provides very impressive classification performance on reasonably sized datasets. Assuming that is given data having two classes. In figure 24 illustrated such a case and three different possible classification lines. Each one of them separates correctly all the training data. But in the first and third classification lines there are data points very close to the line. This means that using test data there is a high chance for misclassification. On the contrary, it does not happen in the middle line which has a higher distance from two classes, so it is a better solution. This is the idea beyond the support vector machines to find a line that has maximum distance to each of the classes, as it seems in figure 25. The distance between the hyperplane and the closest point is called margin, labelled M. This region is symmetric about the line, so that it forms a cylinder about the line in 3D, and a hyper-cylinder in higher dimensions. A data point inside the margin is declared to be too close to the line to be accurately classified. Thus, support vector machines look for the decision boundary that has maximum margin. The datapoints in each class that lie closest to the classification line are called support vectors [101].

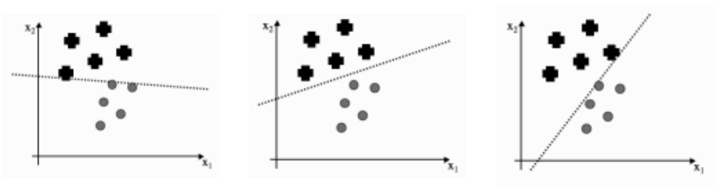


Figure 24: Three different classification lines [101].

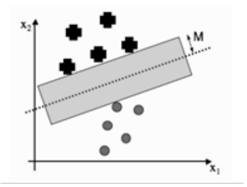


Figure 25: The margin [101].

7.2 Hard margin SVM

The case in which data are linearly separable is called hard margin SVM. Suppose that are given n data points $\mathbf{x} \in R^d$, having two classes, labeled $\{-1, 1\}$. Firstly, it will be computed the margin and then the hyperplane that has the maximum margin to those classes.

A hyperplane is defined from the equation $\boldsymbol{\beta}^T \mathbf{x}+\boldsymbol{\beta}_0$ where $\boldsymbol{\beta}$ and $\boldsymbol{\beta}_0$ are vectors. The direction of vector $\boldsymbol{\beta}$ is perpendicular to hyperplane. Also, for any data point $\mathbf{x}$ that lies on the hyperplane [101]:

$$\beta^T \mathbf{x} + \boldsymbol{\beta}_0 = 0 \iff \boldsymbol{\beta}_0 = -\boldsymbol{\beta}^T \mathbf{x} \tag{74}$$

Moreover, to compute the distance D between a data point x_i and the hyperplane, a data point x_0 can be chosen which lies on the hyperplane, compute $x_i - x_0$ and then projected to the direction of β

$$d_i = \frac{\boldsymbol{\beta}^T (x_i - x_0)}{||\boldsymbol{\beta}||} = \frac{\boldsymbol{\beta}^T x_i - \boldsymbol{\beta}^T x_0}{||\boldsymbol{\beta}||} = \frac{\boldsymbol{\beta}^T x_i - \boldsymbol{\beta}_0}{||\boldsymbol{\beta}||}$$

The distance for point x_i would be

$$D = d_i - y_i \tag{75}$$

As mentioned above, margin is the distance of the hyperplane to the closest point. So, margin is computed using the following expression:

The distance for point x_i would be

$$M = \min d_i y_i = \min \frac{\boldsymbol{\beta}^T x_i - \boldsymbol{\beta}_0}{||\boldsymbol{\beta}||} \tag{76}$$

For any point x_i that is not on the hyperplane $y_i(\boldsymbol{\beta}^T x_i + \boldsymbol{\beta}_0) > 0$. Thus, it means that for every such a point there will be a positive constant c such that $y_i(\boldsymbol{\beta}^T x_i + \boldsymbol{\beta}_0) > c$. So, it is far from hyperplane by c. Both sides can be divided by c:

$$y_i(\frac{\boldsymbol{\beta}^T}{c} x_i + \frac{\boldsymbol{\beta}_0}{c}) \geq 1 \iff y_i(\boldsymbol{\beta}^* x_i + \boldsymbol{\beta}_0^*) \geq 1 \tag{77}$$

From (77) is concluded that for any hyperplane, vectors $\boldsymbol{\beta}$ and $\boldsymbol{\beta}_0$ can be rescaled in a way that the distance of all points from hyperplane is at least 1. When the vector $\boldsymbol{\beta}$ is divided by a constant c its direction does not change, what does change is only its size which is not a problem because vector $\boldsymbol{\beta}$ is computed only to define the decision boundary through its direction. So, the smallest value of (77) is 1. Combining this with the fact that (76) must be minimized, the definition of margin can be modified to [101]:

$$M = \min d_i y_i = \frac{1}{||\boldsymbol{\beta}||} \tag{78}$$

Consequently, the margin depends only on vector $\boldsymbol{\beta}$. As already said, margin should be as large as possible. As a result, the hyperplane which maximizes the margin will be the optimal decision boundary. Therefore, the optimization problem can be formed such that:

$$\min_{\boldsymbol{\beta},\boldsymbol{\beta}_0} ||\boldsymbol{\beta}|| \tag{79}$$

Subject to

$$y_i(\boldsymbol{\beta}^T x_i + \boldsymbol{\beta}_0) \geq 1$$

The constrain $y_i(\boldsymbol{\beta}^T x_i + \boldsymbol{\beta}_0) \geq 1$ is used to make sure that among all hyperplanes that satisfy this constrain, the one that has the minimum $||\boldsymbol{\beta}||$ is the optimal choice. For easier computations the optimization problem can be modified as following without change anything [101]:

$$\min_{\boldsymbol{\beta},\boldsymbol{\beta}_0} \frac{1}{2}||\boldsymbol{\beta}^2|| \tag{80}$$

Subject to

$$y_i(\boldsymbol{\beta}^T x_i + \boldsymbol{\beta}_0) \geq 1$$

Hence, $\boldsymbol{\beta}, \boldsymbol{\beta}_0$ will be chosen to maximize the margin. This is a convex optimization problem (quadratic objective function subject to linear constraints). So, it can be computed the global minimum which is the unique solution of this problem. The lagrangian of this objective function will be:

$$L(\boldsymbol{\beta}, \boldsymbol{\beta}_0, \alpha_i) = \frac{1}{2}||\boldsymbol{\beta}||^2 - \sum_{i=1}^{n} \alpha_i [y_i(\boldsymbol{\beta}^T x_i + \boldsymbol{\beta}_0) - 1] \tag{81}$$

where $\alpha_i \geq 0$ for each i

Instead of solving this primal problem is preferred to solve the dual problem (presented below) because it would be easier with nice properties [102]. The lagrangian has 3 parameters $\boldsymbol{\beta}, \boldsymbol{\beta}_0, \alpha_i$. The derivatives with respect to each one can be calculated and set equal to 0. The derivative with respect to α_i give the constrain back [101].

$$\frac{\partial L}{\partial \boldsymbol{\beta}} = 0 \iff \boldsymbol{\beta} - \sum_{i=1}^{n} \alpha_i y_i x_i = 0 \iff \boldsymbol{\beta} = \sum_{i=1}^{n} \alpha_i y_i x_i \tag{82}$$

$$\frac{\partial L}{\partial \boldsymbol{\beta}_0} = 0 \iff \sum_{i=1}^{n} \alpha_i y_i = 0 \tag{83}$$

Equation 82 shows the relation between the dual variable α and the primal $\boldsymbol{\beta}$ and provides a solution for vector β. Substituting (82), (83) to (81) it is obtained the following equation, known as dual problem, where everything are written in terms of α.

$$L_D = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j x_i^T x_j \tag{84}$$

Subject to $\alpha_i \geq 0, \sum_i^n \alpha_i y_i = 0$

The solution is obtained by maximizing L_D with respect to α_i. This is a simpler convex optimization problem having a global minimum, which can be found using standard software. This will be the final and unique solution for this problem [102].

In addition, the solution must satisfy the Karush–Kuhn–Tucker conditions, which include (82), (83) , $\alpha_i \geq 0$ and

$$\alpha_i[y_i(\boldsymbol{\beta}^T x_i - \boldsymbol{\beta}_0) - 1] = 0, \quad \forall i = 1, ..., n \tag{85}$$

If $\alpha_i > 0 : y_i(\boldsymbol{\beta}^T x_i - \boldsymbol{\beta}_0) = 1$ where $y_i(\boldsymbol{\beta}^T x_i - \boldsymbol{\beta}_0)$ is the distance of point x_i from the hyperplane. The minimum distance is 1 as happens now. Therefore, this point is on the margin because its distance from the hyperplane is the minimum. These points are called support vectors [102]. If $y_i(\boldsymbol{\beta}^T x_i - \boldsymbol{\beta}_0) > 1$ then for each i: $\alpha_i = 0$. These points does not affect β which is defined from (82). It is vector in terms of a linear combination of the support points x_i. Consequently, all the points that are not in the margin have no effect in the optimization. This is good for the algorithm because it makes model more stable and robust in terms of outliers.

$\boldsymbol{\beta}_0$ is obtained by solving (85) for any of the support vectors.

$$\hat{\boldsymbol{\beta}}_0 = \frac{1}{N_s} \sum_{supportvectors j} (y_j - \sum_{i=1}^{n} \alpha_i y_i x_i^T x_j) \tag{86}$$

where N_s is the whole set of support vectors [101]. The optimal separating hyperplane produces a function $\hat{f}(x) = x^T \hat{\boldsymbol{\beta}} + \hat{\boldsymbol{\beta}}_0 =$ for classifying new observations:[102]

$$\hat{G}(x) = sign \hat{f}(x)$$

7.2.1 SVM Numerical Example

An example for building a Support Vector Machine Algorithm ”by hand” is presented below for the data set given in the following table [103]. The data contain only 4 samples and two variables named x_1 and x_2 .

The separability constraints from equation (80) are summarized below. The inequality on a particular row is the separability constraint for the corresponding data point in that row

Table 30: Training data.

x$_1$	x$_2$	y
0	0	-1
2	2	-1
2	0	1
3	0	1

$$-b \geq 1 \; (i)$$
$$-2w_1 + 2w_2 + b \geq 1 \; (ii)$$
$$2w_1 + b \geq 1 \; (iii)$$
$$3w_1 + b \geq 1 \; (iv)$$

Combining (i) and (iii) gives

$$w_1 \geq 1$$

and combining (ii) and (iv) gives

$$w_2 \leq -1$$

Therefore the equation

$$\frac{1}{2}(w_1^2 + w_2^2) \geq 1$$

is true when $w_1 = 1$ and $w_2 = -1$. Moreover, it is easy verify that

$$b^* = -1, w_1^* = 1, w_2^* = -1$$

satisfies all four constraints, minimizes

$$\frac{1}{2}(w_1^2 + w_2^2) \geq 1$$

and therefore gives the optimal hyperplane which will be [103]

$$g(x) = sing(x_1 - x_2 - 1)$$

and the margin will be

$$\frac{1}{||w^*||} = \frac{1}{\sqrt{2}} \approx 0.707$$

7.3 Soft Margin SVM

Previously, the space was rescaled such that margin was 1 and the distance of each point from the decision boundary was always greater than equal one. Nothing was inside the margin. However, there are cases that it is not possible to have a perfect decision boundary because there are always points that are on the wrong side. Therefore, the constraint in the optimization problem (80) is not satisfied. So, a new optimization problem is defined in this case:

$$\min_{\boldsymbol{\beta},\boldsymbol{\beta}_0} \frac{1}{2}||\boldsymbol{\beta}^2|| + \gamma \sum_i \xi_i \tag{87}$$

Subject to

$$y_i(\boldsymbol{\beta}^T x_i + \beta_0) \geq 1 - \xi_i$$

Where $\xi_i \geq 0$. If $\xi_i > 0$, the points are on the wrong side or inside the margin. The constrain in (87) does not mean anything anymore because points are allowed to be less than one. However, a factor has been added to the objective function penalizing those points that are violating the original constrain. If $\xi_i = 0$, this is a situation as in hard margin SVM that the points are over the margin, and nothing added in the objective function. Otherwise, when $\xi_i > 0$ the sum of all ξ_i is multiplied by a coefficient γ and their product is added in the objective function. Since the objective function needs to be minimized, this must happen as less as possible [104].

Now, the primal variables are $\boldsymbol{\beta}, \boldsymbol{\beta_0}$ and a new primal variable ξ_i. So, the Lagrangian will be:

$$L(\boldsymbol{\beta}, \boldsymbol{\beta}_0, \xi_i, \alpha_i, \lambda_i) = \tfrac{1}{2}||\boldsymbol{\beta}||^2 + \gamma \textstyle\sum_{i=1}^n \xi_i - \sum_{i=1}^n \alpha_i[y_i(\boldsymbol{\beta}^T x_i + \boldsymbol{\beta}_0) - (1 - \xi_i)] - \sum_{i=1}^n \lambda_i \xi_i =$$

$$\frac{1}{2}||\boldsymbol{\beta}||^2 + \gamma \sum_{i=1}^n \xi_i - \sum_{i=1}^n \alpha_i y_i \boldsymbol{\beta}^T x_i - \sum_{i=1}^n \alpha_i y_i \boldsymbol{\beta}_0 \tag{88}$$

where α, λ are dual variables.

The derivatives with respect to the primal variables must be calculated and be set equal to 0.

$$\frac{\partial L}{\partial \boldsymbol{\beta}} = 0 \iff \boldsymbol{\beta} - \sum_{i=1}^n \alpha_i y_i x_i = 0 \iff \boldsymbol{\beta} = \sum_{i=1}^n \alpha_i y_i x_i \tag{89}$$

Which is the same as in the hard margin SVM.

$$\frac{\partial L}{\partial \boldsymbol{\beta}_0} = 0 \iff \sum_{i=1}^n \alpha_i y_i = 0 \tag{90}$$

$$\frac{\partial L}{\partial \xi_i} = 0 \iff \frac{\partial L}{\partial \xi_i} = \gamma - \alpha_i - \lambda_i = 0 \tag{91}$$

Substituting (89),(90),(91) to (88) it is obtained the following equation, known as dual problem, where everything are written in terms of dual variables.

$$\sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j x_i^T x_j \tag{92}$$

Subject to

$$\alpha_i \geq 0, \lambda_i \geq 0 \;\; and \;\; \sum_{i=1}^{n} \alpha_i y_i = 0$$

So, the objective function will be exactly the same as in the hard margin case. There is not λ_i in objective function. However, λ_i has to do with because $\gamma - \alpha_i - \lambda_i = 0 => \gamma - \alpha_i = \lambda_i$. So, if $\lambda_i \geq 0 => \gamma - \alpha_i \geq 0 => \gamma \geq \alpha_i$. Hence, $\lambda_i \geq 0$ implies that $\gamma \geq \alpha_i$ [104].

So, the dual problem in this case finally would be

$$\sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j x_i^T x_j \tag{93}$$

subject to

$$0 \leq \alpha_i \leq \gamma \;\; and \;\; \sum_{i=1}^{n} \alpha_i y_i = 0$$

The only difference between hard and soft margin is the constraint. In hard margin always $0 \leq \alpha_i$ and now has been added the constraint that $\alpha_i \geq \gamma$, and γ is the constant of the penalty function. Value γ chosen by the user. The larger the γ, the larger the penalty. The objective function is convex quadratic with linear constraints. Again, the optimization problem is quadratic programming. Thus, as in the hard margin case there is a global minimum, and it can be solved efficiently using standard software [104].

Additionally, the solution must satisfy the Karush–Kuhn–Tucker conditions, which include (89),(90),(91) and the constraints

$$\alpha_i[y_i(\boldsymbol{\beta}^T x_i + \boldsymbol{\beta}_0) - (1 - \xi_i)] = 0, \;\; \forall i = 1, ..., n \tag{94}$$

$$\lambda_i \xi_i = 0 \tag{95}$$

$$y_i(\beta^T \; x_i + \beta_0) - (1 - \; \xi_i) \geq 0 \tag{96}$$

Also, from (89) arises the solution for β with non-zero coefficients α_i. If $\alpha_i > 0$ then $[y_i(\beta^T\ x_i + \beta_0) - 1 + \xi_i] = 0$ so, there are two possibilities. Either $\xi_i = 0$ or $\xi_i > 0$.

When $\xi_i > 0$ from (95) $\lambda_i = 0$. If that happens then from (91) implies that $\alpha_i = \gamma$.

If $\xi_i = 0$ Always implies that $\alpha_i < \gamma$.

λ_i is not necessary 0 because in this case the condition (95) is satisfied by ξ_i. If $\lambda_i > 0$ then again from (91) implies that $\alpha_i < \gamma$.

So, those points that have $\alpha_i > 0$ and lie on the margin ($\xi_i = 0$) can be $\alpha_i < \gamma$ or $\alpha_i = \gamma$. But as shown above, for the points that violate the margin ($\xi_i > 0$) implies always that $\alpha_i = \gamma$. Hence, $\alpha_i = \gamma$ cannot be true for both cases. Therefore, in this case always $\alpha_i < \gamma$.

Therefore, the points that have non-zero coefficients α_i are the support vectors. Among them, some violate the margin and some other are over the margin. All these points make decision on the decision boundary. All the other points have no role.

Also, using the support vectors, the β_0 can be computed from the (94), and typically is used the average of all solutions as the final vector β_0 like in hard margin case. The optimal separating hyperplane produces a function $\hat{f}(x) = x^T\widehat{\beta}+\widehat{\beta_0}$ and for classifying new observations is used the $\hat{G}(x) = sign\hat{f}(x)$ [104].

7.4 Kernel SVM

So far, the data was linearly separable and the algorithm was presented was based on linearly separable data. In the case that the data are not linearly separable a decision boundary which is not linear must be computed. Instead of changing the algorithm in SVMs, is preferable to change the data making them linear and then apply the linear algorithm. By mapping the data in high dimensional space, they can be linearly separable, and a linear discrimination function can be found. To do this, kernel method is used as described previously [101]. A data point can be transformed in a higher dimensional space through $x \mapsto \Phi(x)$. As was applied in PCA the model must be written in a way that it does not depend on coordinates itself. So, a data point x must never appears itself but only the inner product $x^T x$ to put the value of K function in its place. In this way everything mapped to higher dimensional space automatically [104].

There are many kernel functions. Three common kernels in SVMs are [104]:

1)d^{th}-Degree polynomial: $K(x, x')\ = (1 + \langle x, x' \rangle)^d$

2) Radial basis: $K(x, x')\ = e^{-\gamma \|x-x'\|^2}$

3)Neural network: $K(x, x')\ = tanh(k_1 \langle x, x' \rangle + k_2)$

The original space is called observed space and feature is the space of Φ which is never required to be explicitly known.

How this concept is used in SVM and take care of non-linearity.

Equation 84 does not depend on the coordinate of the points but only on the inner product $x_i^T x_j$. Suppose that have been used a function Φ which takes points and map them to high dimensional space and hopefully points that are not linear in the original space becomes linear in that space. So, using that function the objective function would be:

beginequation $\sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j y_i y_j \Phi(x_i^T)\Phi(x_j)$

$$\sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j y_i y_j K(x_i^T, x_j) \tag{97}$$

Subject to $\alpha \geq 0, \sum_i^n \alpha_i y_i = 0$

Now the boundary is not linear. In the linear case when solved 84 and found α, the $\boldsymbol{\beta}$ vector was found using (82). In this case it should be:

$$\boldsymbol{\beta} = \sum_{i=1}^n \alpha_i y_i \Phi(x_i) \tag{98}$$

But $\Phi(x_i)$ is unknown. Therefore, $\boldsymbol{\beta}$ cannot be computed explicitly. However, $\boldsymbol{\beta}$ is computed only to find the decision boundary. Thus, if decision boundary can be computed in a different way it is not necessary to compute its value [104].

Decision boundary is given from the equation $\boldsymbol{\beta}^T x + \boldsymbol{\beta}_0$. If everything mapped in a higher dimensional space, then decision boundary is:

$$\boldsymbol{\beta}^T \Phi(x) + \boldsymbol{\beta}_0 \tag{99}$$

Thus, substituting (98) to (99) the decision boundary in Kernel case is given from the equation

$$\sum_{i=1}^n \alpha_i y_i \Phi(x_i)^T \Phi(x) + \boldsymbol{\beta}_0 \tag{100}$$

The dot product $\Phi(x_i)^T \Phi(x)$ can be replaced by the corresponding kernel $K(x_i, x)$. Thus, the decision boundary in Kernel case is given from the equation [104]

$$\sum_{i=1}^n \alpha_i y_i K(x_i, x) + \boldsymbol{\beta}_0 \tag{101}$$

8 Neural Networks

8.1 Introduction

In this chapter presented neural networks. Their name and structure are inspired by the human brain, since they are based on the way that biological neurons signal to one another. Among other things, Neural networks provide solutions for classification tasks and also for image recognition. To this end, presented also Convolutional Neural Networks which are special kind of feedforward neural networks having better performance with image or speech recognition.

8.2 Generalized Linear Regression

Multiple linear Regression is used to model the linear relationship between a continuous target variable y and $x_1, x_2, ..., x_p$ independent variables [105]. The objective of multiple linear regression is to learn a function so that this function will be a good predictor for the target variable [106]. This function is called hypothesis function and is given by $\hat{y} = \hat{\beta}_0 + \hat{\beta}_1 x_1 + ... + \hat{\beta}_p x_p$ [107]. The model parameters are $\hat{\beta}_0$, $\hat{\beta}_1$,..., $\hat{\beta}_p$ where $\hat{\beta}_0$ is the intercept and $\hat{\beta}_k$ for $k = 1, ..., p$ is the slope for each predictor x_p [106].

To describe nonlinear relationships between $x = [1 \ x_1 \ x_2 \ ... \ x_p]^T$ and z, a nonlinear scalar function called the activation function $\sigma : R \to R$ needs to be introduced. Thus, the linear regression model is modified to generalized linear regression model where the linear combination of the inputs is squashed through the (scalar) activation function [108].

$$z = \sigma(b_0 + w_1 x_1 + w_2 x_2 + \ \ + w_p x_p) \tag{102}$$

The generalized linear regression model is very simple and is itself not capable of describing very complicated relationships between the input x and the output z. Therefore, to increase the generality of the model two further extensions have been done: Firstly, several generalized linear regression models are used to build a layer and then these layers are stacked in a sequential construction which will result in a deep neural network, or simply deep learning [108].

8.3 Perceptron

The Perceptron is a linear machine learning algorithm for binary classification tasks. It is the first type of artificial neural networks and also the simplest one. It consists of a single node or neuron that takes a row of data as input and predicts a class label. This is achieved by calculating the weighted sum of the inputs and a bias as follows.

$$z = \sum_{i=1}^{n} w_i x_i + b \tag{103}$$

Where z is the weighted sum and x_i the n inputs for $i = 1, ..., n$ that are given. Moreover, the coefficients of the model w_i are referred to as input weights and each weight corresponds to a feature. They can be defined using random values in the beginning but they are updated after each training error using the stochastic gradient descent optimization algorithm. The term b is called bias, is created adding the number 1 in the first position of features vector and b in the first position of weights vector. Using bias, the boundary is adjusted away from origin without any dependence on the input value.

Subsequently, a function called activation function takes as input the weights sum and convert it to a numerical value which is the output. For example, when sigmoid activation fucntion is used the output will be 0 if $z < 0$ and will be 1 otherwise. It can also contain other values like "1" or "-1" depending on which activation function is used. The choice of the most appropriate activation function is related to the type of value is needed as output. Common activation functions are Sigmoid function, Sign function, hyperbolic tangent function and Relu function which are presented in section 8.5 [109].

8.4 Artificial Neural Networks

Deep neural network comprised of an input layer which is the leftmost layer and contains the inputs $x_1, ..., x_p$ of the model, are called input neurons. The output layer is the rightmost layer which contains the outputs of the model, are called output neurons. The middle layers are called hidden layers since the intermediate outputs α_j^l , are so-called hidden units or neurons, are neither inputs nor outputs of the whole model [110]. The vector of the intermediate outputs α_j^l (activation function) can be denoted by $a^{(l)} = [\ \alpha_1^{(l)}, ...,\ \alpha_{m_l}^{(l)}]^T$, where α_j^l denotes the activation of the j^{th} neuron in the l^{th} layer [108].

Each middle layer consists of m_l hidden neurons for $l = 1, \ldots, L-1$, and their number varies among layers. Also, by $w_{jk}^{(l)}$ is denoted the weight connects the k^{th} activation of $(l-1)$ layer to the j^{th} activation of (l) layer. Similarly, b_j^l is the bias of the j^{th} neuron in the l^{th} layer. Each layer can be parametrized using a weight matrix $W^{(l)}$ and an offset vector $b^{(l)}$. Layers are stacked such that the outputs of the first hidden layer $a^{(1)}$ are the inputs to the second layer, the outputs of the second layer $a^{(2)}$ are the inputs to thxe third layer etc. A deep neural network can mathematically be described as below [111] and an example for $L = 5$ layers illustrated in figure 26 [112] and

$$\alpha^{(1)} = \sigma(W^{(1)T} x \ + b^{(1)T})$$
$$\alpha^{(2)} = \sigma(W^{(2)T} \alpha^{(1)} \ + b^{(2)T})$$

$$\begin{aligned} &\vdots \\ \alpha^{(L-1)} &= \sigma(W^{(L-1)T}\alpha^{(L-2)} + b^{(L-1)T}) \\ \alpha^{(L)} &= \sigma(W^{(L)T}\alpha^{(L-1)} + b^{(L)T}) \end{aligned} \tag{104}$$

where $x = [x_1, \ldots, x_p]^T$

and

$$b^{(l)} = [\ b_1^{(l)} \quad \ldots \quad b_{m_l}^{(l)}\ ], W^{(l)} = \begin{bmatrix} w_{11}^{(1)} & \cdots & w_{1m_{l-1}}^{(1)} \\ \vdots & \ddots & \vdots \\ w_{m_l1}^{(1)} & \cdots & w_{m_lm_{l-1}}^{(1)} \end{bmatrix} \tag{105}$$

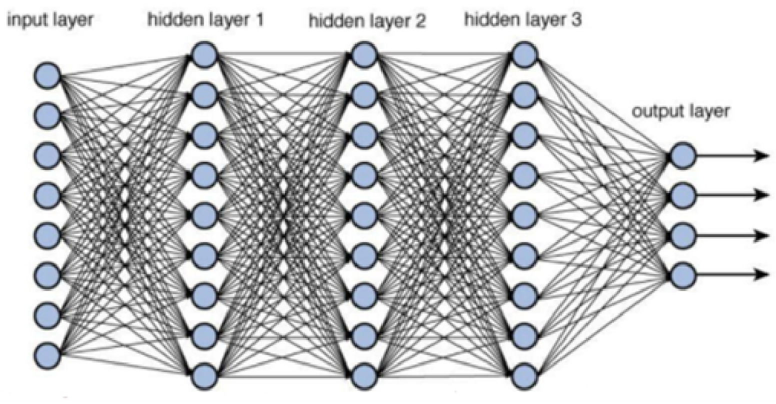


Figure 26: Deep Neural Network architecture [112].

The number of inputs p and the number of outputs K are given by the problem, but the number of layers and their dimensions $m_1, m_2, \ldots$ are user design choices that will determine the flexibility of the model.

For deep neural networks the parameters are [108]

$$\theta = [vec(W^{(1)})^T \ vec(W^{(2)})^T \ \ldots \ vec(W^{(L)})^T \quad b^{(1)}\, b^{(2)} \ \ldots \ b^{(L)}]^T$$

The wider and deeper the network is, the more parameters there are. Practical deep neural network model can easily have in the order of millions of parameters, therefore, they are extremely flexible.

8.5 Activation Functions

Activation function has an extremely significant influence on the ability and efficiency of the neural network, and in different parts of the model various activation functions can be used. However, for all nodes in the same layer only the same activation function can be used and most often the same activation function is used in each hidden layer. In the output layer typically is used a different activation function and as presented in the next sections, is dependent upon the type of prediction required by the model . Activation functions are also usually differentiable. This is required since neural networks are trained using the backpropagation (presented below) of error algorithm that calculates the derivative of prediction error in order to update the weights of the model [113].

8.5.1 Activation Functions for Hidden Layers

Most of the time, in the hidden layers is used a differentiable nonlinear activation function instead of linear functions. Using non-linear functions, the model is able to learn more complex structures in the data. The activations functions that are used in hidden layers are mostly the following three:

- Rectified Linear Activation (ReLU): $\sigma(x) = max(0, x)$. Relu is linear for values greater than zero and nonlinear function for negative values as always the output for negative inputs is zero allowing the learning of complex relationships in the data. [114]
- The Logistic (Sigmoid) activation function: $\sigma(x) = \frac{1}{1+e^{-x}}$. The input can be any real value and the outputs ranges from 0 to 1. As the inputs becomes larger the output tends to number 1, while in contrast as the input gets small values, the output tends to 0.0. [113]
- Hyperbolic Tangent (Tanh) activation function: $\sigma(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$. The input can be any real value and the outputs ranges from -1 to 1 . As the inputs becomes larger the output tends to number 1, while in contrast as the input gets small values, the output tends to -1 [113].

Sigmoid and hyperbolic tangent are two commonly used activation functions. However, both present two drawbacks. Firstly, they are only really sensitive to changes near their mid -point of their input i.e. 0.5 for sigmoid and 0.0 for tanh. Moreover they saturate, due to the fact that large values are squeezed to 1 and small values to -1 for tanh or 0 for sigmoid [114].

The vanishing gradient problem

In very deep neural networks, the learning process using these activation functions becomes a difficult task. As presented below in the backpropagation algorithm, error is back propagated across the network and used to update the weights. The amount of error is significantly reduced with each supplementary layer through which it is propagated, given the derivative of the chosen activation function. This is known as *the vanishing gradient problem* and averts deep (multi-layered) networks from learning efficiently.

On the contrary, using Relu activation function a very deep neural network can be developed without the aforementioned issue. The linearity of Relu for values greater than zero is beneficial when training a neural network using backpropagation. Relu function has become the standard activation function in most types of neural networks and also in Convolutional Neural Networks which are presented below [114].

8.6 Neural Networks for Regression

One of the drawbacks of the Linear Regression model is the assumption of linear relationship between the dependent and independent variables and therefore cannot learn complex non-linear relationships. On the contrary, Artificial Neural Networks have the ability to learn not only linear but also complex non-linear relationships between the features and target due to the presence of activation function in each layer [115]. ANN is a flexible model that can without difficulty be adjusted to the shape of the data. If the results are not satisfying, extra hidden neuron layers can be added to enhance its accuracy and turn it more complex [116]. When there is only a single output neuron, the linear or identity activation function is used in the output layer [113].

8.6.1 Multiple-Output Regression

The output neurons can be more than one and this case is called Multiple-output Regression or Multi-output Regression. This is the regression task that requires predicting more than one numeric value for each input sample, and the outputs are computed at the same time. Multi-output regression tasks can be handled using specialized machine learning algorithms that have the ability to output multiple variables for each prediction can handle and deep learning neural network is one of them. When neural networks are used for multi-output regression tasks , each node in the output layer corresponds to a target variable and in each one is applied the linear activation function [117].

8.7 Neural Networks for Classification

Neural networks can also be used for classification having qualitative outputs instead of quantitative. The classification cases that neural networks can be used are: Binary Classification, Multi-Class Classification, Multi-label Classification.

Binary Classification

In Binary classification where the result is either 0 or 1, the sigmoid function is usually used in the output layer. The output layer will have a single neuron and the value for sigmoid function lies between 0 and 1. The input is classified to category 1 if value is greater than 0.5 and is classified to category 0 otherwise.

Multiple-Class Classification

In multi-class classification is usually used the softmax activation function in the output layer.The final layer of the neural network will have one neuron corresponding to each of the classes and the returning value for each node will have a range from 0 to 1. Softmax function produces positive estimates that sum always to one. Therefore, the output from the softmax layer can be thought as a probability distribution [118].

Multiple-Label Classification

There are some classification tasks require predicting more than one class label. This means that class labels are not mutually exclusive in contrast to normal classification tasks. These tasks are called multiple label classification, or multi-label classification. In multi-label classification, the output consists of zero or more labels for each input sample and the outputs are computed simultaneously.

Multi-label classification tasks can be handled using specialized machine learning algorithms that have the abilitiy to predict multiple mutually non-exclusive classes or “labels” and deep learning neural network is one of them. When neural networks are used for Multi-label classification tasks, each node in the output layer corresponds to a target label. In each node the sigmoid activation function is used returning values from 0 to 1 and can be considered as a probability of class membership for the label [119].

8.8 Learning Process

8.8.1 Gradient Descent

Gradient descent algorithm is used for the training process. Gradient descent manages to find weights and biases so that the output of the network approximates the response value y which is given from the data, for all training inputs x. To find those parameters an update rule is applied again and again, making moves towards the minimum of the cost function, until a global minimum is reached. The update rule will be [110]

$$w_{jk}^{(l)} \rightarrow w_{jk}^{(l)\prime} = w_{jk}^{(l)} - \eta \frac{\partial C}{\partial w_{jk}^{(l)}} \tag{106}$$

$$b_{j}^{(l)} \rightarrow b_{j}^{(l)\prime} = b_{j}^{(l)} - \eta \frac{\partial C}{\partial b_{j}^{(l)}} \tag{107}$$

The rule does not always work - several things can go wrong preventing gradient descent from finding the global minimum of C. But, in practice gradient descent often works extremely well and in neural networks is a powerful way of minimizing the cost function, helping the learning procedure [110].

8.8.2 Stochastic Gradient Descent

There are several challenges in applying the gradient descent rule. The cost function (3.3) has the form $C = \frac{1}{n}\sum_x C_x$. In practice, to compute the gradient ∇C the gradients ∇C_x are computed separately for each training input, x, and then average them i.e. $\nabla C = \frac{1}{n}\sum_x \nabla C_x$. Unfortunately, when the number of training inputs is very large this can take a long time and learning thus occurs slowly.

An idea called stochastic gradient descent can be used to speed up learning. The idea is to estimate the gradient ∇C by computing ∇C_x for a small number m of randomly chosen training inputs $X_1, X_2, \ldots, X_m$ which are called mini batch. Provided the sample size m is large enough, the average value of the $\nabla C_{X_i}, i = 1, \ldots, m$ is expected to be roughly equal to the average over all ∇C_x, that is [110]

$$\frac{\sum_{i=1}^{n} \nabla C_{X_i}}{m} \approx \frac{\sum_x \nabla C_x}{n} = \nabla \mathrm{C} \tag{108}$$

The estimate will not be perfect - there will be statistical fluctuations - but it does not need to be perfect: all is really matters is moving in a general direction that will help decrease C, and so it does not need an exact computation of the gradient. Then stochastic gradient descent works by picking out a randomly chosen mini-batch of training inputs, and training with those,

$$w_{jk}^{(l)} - \longrightarrow w_{jk}^{(l)\prime} = w_{jk}^{(l)} - \frac{\eta}{m} \sum_{i=1}^{m} \frac{\partial C_{X_i}}{\partial w_{jk}^{(l)}} \tag{109}$$

$$b_j^{(l)} \rightarrow b_j^{(l)\prime} = b_j^{(l)} - \frac{\eta}{m} \sum_{i=1}^{m} \frac{\partial C_{X_i}}{\partial b_j^{(l)}} \tag{110}$$

where the sums are over all the training examples $X_i, i = 1, \ldots, m$ in the current mini-batch. Then another randomly chosen mini-batch is picked out and is trained with the same way. And so on, until the training inputs have been used, which is said to complete an epoch of training. At that point a new training epoch is started. Each epoch starts by randomly shuffling the training data, and then partitions it into mini-batches of the appropriate size. Then for each mini batch a single step is applied of gradient descent. In practice, stochastic gradient descent is a commonly used and powerful technique for learning in neural networks, and it is the basis for many of the learning techniques [110].

8.8.3 Backpropagation Algorithm

Backpropagation is the core algorithm behind how neural networks learn. As seen before, to apply update rule (106) or (109) the calculatation of the gradient vector is required .Backpropagation is a fast algorithm and is used to compute all the components of the gradient vector. The magnitude of each component ∇C indicates how sensitive the cost function is to each weight and bias [120].

The partial derivatives $\frac{\partial C_0}{\partial w_{jk}^{(l)}}, \frac{\partial C_0}{\partial b_j^{(l)}}$ are the derivatives with respect to $w_{jk}^{(l)}, b_j^{(l)}$ only for the cost of a specific training example. Since the full cost function involves averaging together all those costs across many training examples, its derivative requires averaging this expression that found over all training examples and thus, eventually they arise the final components $\frac{\partial C}{\partial w_{jk}^{(l)}}, \frac{\partial C}{\partial b_j^{(l)}}$ of the gradient vector ∇C as presented in the following equations.

$$\frac{\partial C}{\partial w_{jk}^{(l)}} = \frac{1}{n} \sum_{k=0}^{n-1} \frac{\partial C_k}{\partial w_{jk}^{(l)}} \tag{111}$$

$$\frac{\partial C}{\partial b_j^{(l)}} = \frac{1}{n} \sum_{k=0}^{n-1} \frac{\partial C_k}{\partial b_j^{(l)}} \tag{112}$$

Expression (111) shows how the overall cost of the network will change when the weight l is wiggled. Assuming that the desired values for the outputs of the model for a given training example are $y = [y_1, \dots, y_{m_L}]$. For example, y_j might be zero or one. For simplicity it will be used the quadratic cost function for the explanation of this method. Nothing changes when another cost function is used. About the cost function for a given training example are added up the squares of the differences between these last layer activations and the desired output. So, the cost function will be:

$$C_0 = \frac{1}{2} \sum_{j=1}^{m_L} (a_j^{(L)} - y_j)^2 \tag{113}$$

As has already been said layer l has m_l neurons and layer $l-1$ has m_{l-1} neurons. The weighted input to the activation function for neuron j in layer l will be:

$$z_j^{(l)} = w_{j0}^{(l)} a_0^{(l-1)} + w_{j1}^{(l)} a_1^{(l-1)} + w_{j2}^{(l)} a_2^{(l-1)} + \ldots + w_{jm_{l-1}}^{(l)} a_{m_{l-1}}^{(l-1)} + b_j^{(l)}, j = 1, 2, \ldots, m_l, l = 1, \ldots, L \tag{114}$$

So the activation of the j^{th} neuron in layer l is

$$a_j^{(l)} = \sigma(z_j^{(l)}) \tag{115}$$

Where σ is an activation function.

Partial derivatives $\frac{\partial C_0}{\partial w_{jk}^{(l)}}, \frac{\partial C_0}{\partial b_j^{(l)}}$ cannot be calculated directly since cost function formula (113) contains neither $w_{jk}^{(l)}$ nor $b_j^{(l)}$ term. However, as shown in the equation (114) $w_{jk}^{(l)}$affects $z_j^{(l)}$ which in turn affects $a_j^{(l)}$ which directly affects C_0. Consequently, the chain rule can be used in this case. Using the chain rule the partial derivatives will be:

$$\frac{\partial C_0}{\partial w_{jk}^{(l)}} = \frac{\partial z_j^{(l)}}{\partial w_{jk}^{(l)}} \frac{\partial a_j^{(l)}}{\partial z_j^{(l)}} \frac{\partial C_0}{\partial a_j^{(l)}} = a_k^{(l-1)} \sigma'(z_j^{(l)}) \frac{\partial C_0}{\partial a_j^{(l)}} \tag{116}$$

$$\frac{\partial C_0}{\partial b_j^{(l)}} = \frac{\partial z_j^{(l)}}{\partial b_j^{(l)}} \frac{\partial a_j^{(l)}}{\partial z_j^{(l)}} \frac{\partial C_0}{\partial a_j^{(l)}} = \sigma'(z_j^{(l)}) \frac{\partial C_0}{\partial a_j^{(l)}} \tag{117}$$

Due to the fact that $\frac{\partial z_j^{(l)}}{\partial w_{jk}^{(l)}} = a_k^{(l-1)}$, $\frac{\partial a_j^{(l)}}{\partial z_j^{(l)}} = \sigma'(z_j^{(l)})$, $\frac{\partial z_j^{(l)}}{\partial b_j^{(l)}} = 1$

When $l = L$ the calculation of the derivative of the cost function with respect to one of the activations of the final layer is directly calculated:

$$\frac{\partial C_0^{(L)}}{\partial a_j^{(L)}} = (a_j^{(L)} - y_j) \tag{118}$$

The same partial derivative with respect to one of the activations of previous layers l for $l = 1, \ldots, L-1$ is a little more complicated. In this case the neuron influences the cost function through multiple paths. In particular, if layer L has m_L neurons and layer $L-1$ has m_{L-1} neurons then a neuron activation $\alpha_j^{(L-1)} for j = 1, \ldots, m_L - 1$ influences the cost function through m_L paths. That is, it influences $\alpha_1^{(L)}$ which plays a role in the cost function, but it also has an influence on $\alpha_2^{(L)}, \ldots, \alpha_{m_L}^{(L)}$ which also play a role in the cost function, and those up need to be added. A relative example illustrated in figure 27 in which the number of neurons in layer L is $m_L = 2$ and seems that $\alpha_k^{(L-1)}$ influences the cost function through $\alpha_0^{(L)}$ and $\alpha_1^{(L)}$. Consequently, to calculate the partial derivatives $\frac{\partial C_0}{\partial \alpha_j^{(l)}}, l = 1, \ldots, L-1$ the chain rule can be used again using the following formula [120].

$$\frac{\partial C_0}{\partial a_j^{(l)}} = \sum_{j=1}^{n_{l+1}} w_{jk}^{(l+1)} \sigma'\left(z_j^{(l+1)}\right) \frac{\partial C_0}{\partial \alpha_j^{(l+1)}}, \; for\, l = 1, \ldots, L-1 \tag{119}$$

or

$$\frac{\partial C_0}{\partial a_j^{(l)}} = (a_j^{(l)} - y_j), \quad for\, l = L$$

Therefore, the final components of the gradient vector will be

$$\frac{\partial C}{\partial w_{jk}^{(l)}} = \frac{1}{n} \sum_{k=0}^{n-1} \frac{\partial C_k}{\partial w_{jk}^{(l)}} \tag{120}$$

$$\frac{\partial C}{\partial b_j^{(l)}} = \frac{1}{n} \sum_{k=0}^{n-1} \frac{\partial C_k}{\partial b_j^{(l)}} \tag{121}$$

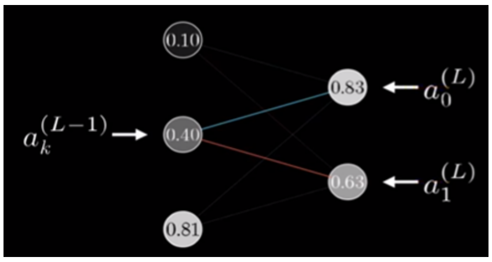


Figure 27: The activation in one layer affects all the activations in the next layer [120].

8.8.4 Learning Slowdown

The hope and expectation of neural networks is to learn fast from their errors. Artificial neuron has a lot of difficulty learning when it is badly wrong - far more difficulty than when it is just a little wrong. A neuron learns by changing the weight and bias at a rate determined by the partial derivatives of the cost function, $\frac{\partial C}{\partial w}$ and $\frac{\partial C}{\partial b}$. So, saying "learning is slow" is really the same as saying that those partial derivatives are small. Both the expressions contain the term $\sigma'(z_j^{(l)})$ which affects their behavior [121].

When σ is the sigmoid, the more the output of the neuron is close to 1, the curve gets very flat, and so $\sigma'(z)$ gets very small. Hence, the amounts $\frac{\partial C}{\partial w}$ and $\frac{\partial C}{\partial b}$ get very small values. This is the origin of the learning slowdown [121].

8.8.5 Cross Entropy Cost Function

The cross-entropy cost function is defined as $C = -\frac{1}{n} \sum_{x=1}^{n} \sum_{j=1}^{m_L} [y_j ln a_j^L + (1-y) ln(1-a_j^L)]$ where n is the total number of items of training data, the sum is over all training inputs x, y_j is the desired value at the j^{th} output neuron and α_j^L is the activation of the j^{th} neuron in the last L^{th} layer. Two properties are necessary to satisfy a cost function. To be always positive, and tends toward zero as the neuron gets better at computing the desired output, y, for all training inputs, x. Indeed, both properties are also satisfied by the quadratic and cost-entropy cost function. But the cross-entropy cost function has the benefit that, unlike the quadratic cost, it avoids the problem of learning slowing down [121].

8.8.6 Weight Initialization

The weights and biases of a neural network must be initialized. Suppose that a neuron with n_{in} input weights have been created. Then those weights must be initialized as Gaussian random variables with mean 0 and standard deviation $\frac{1}{\sqrt{n_{in}}}$ input weights. Also, the bias can be initialized as a Gaussian with mean 0 and standard deviation 1. Provided that the weights have not been initialized properly, then they may learn very slowly using the gradient descent algorithm.

Improved weight initialization speeds up learning but does not always improve the final networks performance. If a network has been trained using both the initialization ways maybe after just a few more epochs of training the accuracies will become almost the same. But also, in many examples of neural networks the long-run behavior is significantly better with the $\frac{1}{\sqrt{n_{in}}}$ weight initialization. Thus, it is not only the speed of learning is improved, it is sometimes also the final performance [121].

8.9 Overfitting and Regularization

A model that agrees well with the available data, is not necessarily a good model. It may just mean that there is enough freedom in the model that it can describe almost any data set of the given size, without capturing any genuine insights into the underlying phenomenon. When that happens, the model works well for the existing data, but fails to make predictions in situations in which has not been exposed to before [121].

A relative example illustrated in figure 28. In the first 200 epochs (not shown) the accuracy rises to just under 82 percent. The learning then gradually slows down. Finally, at around epoch 280 the classification accuracy pretty much stops improving. Later epochs merely see small stochastic fluctuations near the value of the accuracy at epoch 280. What the network learns after epoch 280 is no longer generalized to the test data meaning that it is not useful learning. Therefore, the network is *overfitting* or *overtraining* beyond epoch 280 [121].

Another sign of overfitting can be found in the classification accuracy on the training data which is illustrated in figure 29 for the same network. The accuracy rises all the way up to 100 percent. At the same time, the test accuracy tops out at just 82.27 percent. So, the network really is learning about peculiarities of the training set, not just recognizing digits in general.

Overfitting is a major problem in neural networks. This is especially true in modern networks, which often have very large numbers of weights and biases. To train effectively, it is required a way of detecting when overfitting is going on, so to avoid it. Also, are required techniques for reducing the effects of overfitting [121].

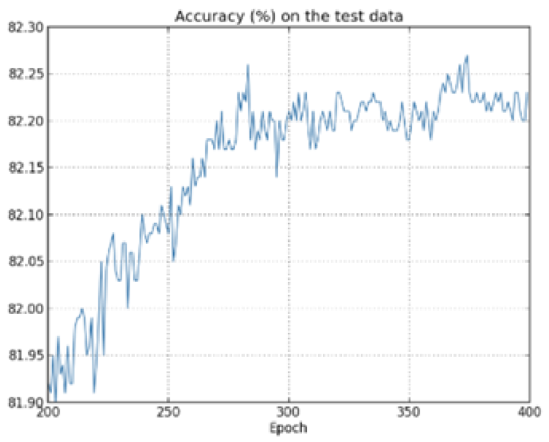


Figure 28: How the classification accuracy on the test data changes over time [121].

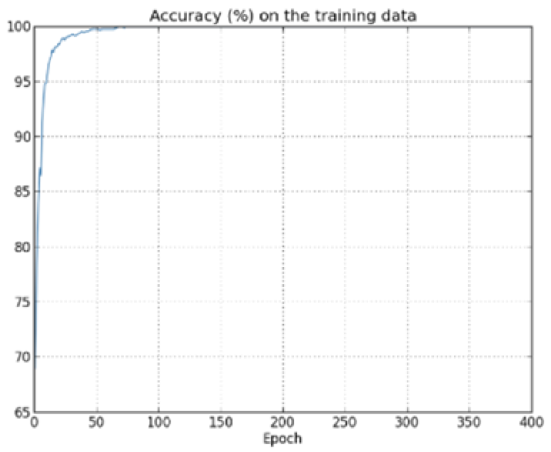


Figure 29: How the cost on the test data changes over time [121].

8.9.1 Regularization

In general, one of the best ways of reducing overfitting is to increase the size of the training data. Unfortunately, the collection of training data can be expensive or difficult process, thus this is not always the best choice. One possible approach is to reduce the size of the network. However, large networks have the potential to be more powerful than small networks and so this is an option that would be adopted only reluctantly.

Regularization techniques can reduce overfitting, keeping the existed training data and networks. The most commonly used regularization technique is known as weight decay or $L2$ regularization. The idea of $L2$ regularization is to add an extra term to the cost function, a term called the regularization term. In general, a regularized cost function can be written as [121]

$$C = C_0 + \frac{\lambda}{2n} \sum_w w^2 \tag{122}$$

where C_0 is the original, unregularized cost function.

The first term is just the usual expression for the cost function. A second term has been added containing the sum of the squares of all the weights in the network. This is scaled by a factor $\frac{\lambda}{2n}$, where $\lambda > 0$ is known as the *regularization parameter*, and n is, as usual, the size of the training set. It is also worth noting that the regularization term does not include the biases. Having a large bias does not make a neuron sensitive to its inputs in the same way as having large weights. The effect of regularization is the fact that the network prefers to learn small weights while nothing more changes. Smaller weights means that the behavior of the network will not change too much provided that a few random inputs have been changed. That makes it difficult for a regularized network to learn the effects of local noise in the data. It is an empirical fact that regularized neural networks usually generalize better than unregularized networks [121].

$L2$ regularization technique is not the only one. Many other techniques have been developed. Three significant of them are: $L1$ regularization, dropout, and artificially increasing the training set size. Their detailed presentation is beyond the scope of this project [121].

Training a regularized network

The partial derivatives $\frac{\partial C}{\partial w}$ and $\frac{\partial C}{\partial b}$ for all the weights and biases in a regularized network will be:

$$\frac{\partial C}{\partial w} = \frac{\partial C_0}{\partial w} + \frac{\lambda}{n} w \tag{123}$$

$$\frac{\partial C}{\partial b} = \frac{\partial C_0}{\partial b} \tag{124}$$

The $\frac{\partial C_0}{\partial w}$, $\frac{\partial C_0}{\partial b}$ terms can be computed using backpropagation as have been described and then $\frac{\lambda}{n} w$ is added to the partial derivative of all the weight terms. Consequently, it is easy to compute the gradient of the regularized cost function. The gradient descent learning rule for the biases does not change but the learning rule for the weights will be [121]

$$w \rightarrow w - \eta \frac{\partial C_0}{\partial w} - \eta \frac{\lambda}{n} w = (1 - \eta \frac{\lambda}{n}) w - \eta \frac{\partial C_0}{\partial w} \tag{125}$$

This is exactly the same as the usual gradient descent learning rule, except that the weight w first is rescaled by a factor $1 - \eta \frac{\lambda}{n}$. This rescaling is sometimes referred to as weight decay, since it makes the weights smaller.

Regularization does not significantly affect the stochastic gradient descent either. As in unregularized stochastic gradient descent, the amount $\frac{\partial C_0}{\partial w}$ can be estimated by averaging over a mini batch of m training examples. Thus, the regularized learning rule for stochastic gradient descent will be as follow.

$$w \rightarrow (1 - \eta\frac{\lambda}{n})w - \frac{\eta}{m}\sum_x \frac{\partial C_x}{\partial w} \tag{126}$$

8.10 Hyper-Parameters

When building a neural network is required to choose values for hyper-parameters such as the learning rate η, the $L2$ regularization parameter λ, the mini-batch size and others. In practice, it might be difficult to choose good hyper-parameters. Although it is a crucial step for model's performance. Usually, a lot of experimentation is required to get appropriate values for the parameters [121].

8.10.1 Learning Rate η

In gradient or stochastic gradient descent the update rules contain η parameter which affects the size of the steps towards the minimum of the cost function. If η is too large then the steps will be so large that they may overshoot the minimum, causing the algorithm to increase the cost function. To the other side, choosing η quite small slows down gradient descent which is also a problem. One initial value can be $\eta = 0.01$. If the cost decreases during the first few epochs, then values $\eta = 0.1, 1.0$ should successively be tried until a value for η has been found for which the cost oscillates or increases during the first few epochs. Alternately, if the cost oscillates or increases during the first few epochs when $\eta = 0.01$, then values $\eta = 0.001, 0.0001, \ldots$ can be tested until a value for η is found for which the cost decreases during the first few epochs.

It is often beneficial to vary the learning rate. During the learning process the weights can be badly wrong. And so, it is best to use a bigger learning rate that causes the weights to change quickly. Later, the learning rate can be reduced as more fine-tuned adjustments are made to the weights [121].

8.10.2 Mini-Batch Size

The choice of mini-batch size at which the speed is maximized is relatively independent of the other hyper-parameters, so it is not required to optimize them to find a good mini-batch size. Some acceptable (but not necessarily optimal) values can be used for the other hyper-parameters, and then trial several different mini-batch sizes, scaling as above. The mini-batch size can be determined from a plot of validation accuracy versus time as the size which gives the most rapid improvement in performance [121].

8.10.3 Training epochs

One method to determine the number of training epochs is called early stopping. Early stopping means that at the end of each epoch the classification accuracy should be computed on the validation data and when that stops improving, terminate. Accuracy can jump around quite a bit, even when the overall trend is to improve. The preferable rule is to terminate when the best classification accuracy is not improved for quite some epochs instead of the first time in which the accuracy decreases ensuring that the procedure does not stop too soon, but also not to late [121].

8.10.4 The Regularization Parameter λ

It is proposed to start initially with no regularization ($\lambda = 0.0$), and determining a value for η, as above. Then, an initial value can be $\lambda = 1.0$ and then increase or decrease by factors of 10, to improve performance on the validation data. Once a good order of magnitude has been found, the value of λ can be optimized.

Apart from the aforementioned tactics to optimize hyper-parameters by hand, great deal of work has been done on automating the process. A common technique is grid search, which systematically searches through a grid in hyper-parameter space [121]. Undoubtedly, it has not been covered everything about hyper-parameter optimization. That is a huge subject, and it is not, in any case, a problem that is ever fully solved, nor is there common understanding amongst practitioners on the right strategies to use. There is always one more idea someone can try to improve a bit more the effectiveness from his network. It becomes clear that neural networks require a lot of work when compared with other machine learning techniques [121].

8.10.5 Neural Network Numerical Example

An example for building a Neural Network Algorithm ”by hand” is presented below for a single training set: given inputs 0.05 and 0.10 the neural network should have output 0.01 and 0.99.

Firstly, the predicition of the network having the initial parameters values is calculated. To this end, must be calculated the weighted sum for h_1:

$net_{h1} = w_1 * i_1 + w_2 * i_2 + b_1 * 1 = 0.15 * 0.05 + 0.2 * 0.1 + 0.35 * 1 = 0.3775$ And putting this input to the logistic function is computed the output of h_1:

$out_{h1} = \frac{1}{1+e^{-net_{h1}}} = \frac{1}{1+e^{-0.3775}} = 0.593269992$

The output for h_2 by doing the same process will be:

$out_{h2} = 0.596884378$

This process is repeated for the output layer neurons named o_1 and o_2, using the output from the hidden layer neurons as inputs.

$net_{o1} = w_5 * out_{h1} + w_6 * out_{h2} + b_2 * 1 \Longrightarrow$

$net_{o1} = 0.4 * 0.593269992 + 0.45 * 0.596884378 + 0.6 * 1 = 1.105905967$

$out_{o1} = \frac{1}{1+e^{-net_{o1}}} = \frac{1}{1+e^{-1.105905967}} = 0.75136507$

And doing the same process for o_2 the output will be

$out_{o2} = 0.772928465$

Then, it can be calculated the error for each output neuron using the squared error function and by summing them to get the total error

$E_{total} = \sum \frac{1}{2}(target - output)^2$

The target output for o_1 is 0.01 but the neural network output 0.75136507, therefore its error is:

$E_{o1} = \frac{1}{2}(target_{o1} - out_{o1})^2 = \frac{1}{2}(0.01 - 0.75136507)^2 = 0.274811083$

And doing the same process for o_2 the error will be

$E_{o2} = 0.023560026$

The total error for the neural network is the sum of these errors:

$E_{total} = E_{o1} + E_{o2} = 0.274811083 + 0.023560026 = 0.298371109$

Backpropagation can be applied to update each of the weights in the network minimizing the total error as well as the error of each output neuron.

As indicative examples, backpropagation will be applied to w_1 from the hidden layer and w_5 from the output layer. For w_5 parameter, the computation of $\frac{\partial E_{total}}{\partial w_5}$ is required. To this end, chain rule must be used since there is no direct w_5 term in error function.

$$\frac{\partial E_{total}}{\partial w_5} = \frac{\partial E_{total}}{\partial out_{o1}} * \frac{\partial out_{o1}}{\partial net_{o1}} * \frac{\partial net_{o1}}{\partial w_5}$$

The terms in the above equation are calculated below. First, the change in total error with respect to the output.

$E_{total} = \frac{1}{2}(target_{o1} - out_{o1})^2 + \frac{1}{2}(target_{o2} - out_{o2})^2$

$\frac{\partial E_{total}}{\partial out_{o1}} = 2 * \frac{1}{2}(target_{o1} - out_{o1})^{2-1} * -1 + 0$

$\frac{\partial E_{total}}{\partial out_{o1}} = -(target_{o1} - out_{o1}) = -(0.01 - 0.75136507) = 0.74136507$

Next, the change output $o1$ with respect to the total weighted sum is

$\frac{\partial out_{o1}}{\partial net_{o1}} = \frac{\partial(\frac{1}{1+e^{-net_{o1}}})}{\partial net_{o1}} = out_{o1}(1 - out_{o1}) = 0.75136507(1 - 0.75136507) = 0.186815602$

and the change in the weighted sum with respect to w_5

$net_{o1} = w_5 * out_{h1} + w_6 * out_{h2} + b_2 * 1$

$\frac{\partial net_{o1}}{\partial w_5} = 1 * out_{h1} * w_5^{(1-1)} + 0 + 0 = out_{h1} = 0.593269992$

Putting it all together:

$\frac{\partial E_{total}}{\partial w_5} = \frac{\partial E_{total}}{\partial out_{o1}} * \frac{\partial out_{o1}}{\partial net_{o1}} * \frac{\partial net_{o1}}{\partial w_5} = \frac{\partial E_{total}}{\partial w_5} = 0.741 * 0.186 * 0.593 = 0.082$

Then, gradient descent can be applied for the w_5 parameter using the update rule in equation (106).

$w_5' = w_5 - \eta * \frac{\partial E_{total}}{\partial w_5} = 0.4 - 0.5 * 0.082167041 = 0.35891648$

Backpropagation process is repeated for the parameters in hidden layer w_1, w_2, w_3, and w_4. For w_1 similar to before, the computation of $\frac{\partial E_{total}}{\partial w_1}$ is required. To this end, chain rule must be used since there is no direct w_1 in error function.

$$\frac{\partial E_{total}}{\partial w_1} = \frac{\partial E_{total}}{\partial out_{h1}} * \frac{\partial out_{h1}}{\partial net_{h1}} * \frac{\partial net_{h1}}{\partial w_1}$$

As mentioned in backpropagation a hidden neuron affects all the output neurons. In this case, out_{h1} affects both out_{o1} and out_{o2} and therefore the $\frac{\partial E_{total}}{\partial out_{h1}}$ will be

$\frac{\partial E_{total}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial out_{h1}} + \frac{\partial E_{o2}}{\partial out_{h1}}$

The term $\frac{\partial E_{o1}}{\partial out_{h1}}$ is calculated as

$\frac{\partial E_{o1}}{\partial out_{o1}} = \frac{\partial E_{o1}}{\partial net_{o1}} * \frac{\partial net_{o1}}{\partial out_{o1}} = 0.74136507 * 0.186815602 = 0.138498562$

The term $\frac{\partial E_{o1}}{\partial net_{o1}}$ will be

$\frac{\partial E_{o1}}{\partial net_{o1}} = \frac{\partial E_{o1}}{\partial out_{o1}} * \frac{\partial out_{o1}}{\partial net_{o1}} = 0.74136507 * 0.186815602 = 0.138498562$

And $\frac{\partial net_{o1}}{\partial out_{h1}}$ is equal to w_5:

$net_{o1} = w_5 * out_{h1} + w_6 * out_{h2} + b_2 * 1$

$\frac{\partial net_{o1}}{\partial out_{h1}} = w_5 = 0.40$

Plugging them in:

$$\frac{\partial E_{o1}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial net_{o1}} * \frac{\partial net_{o1}}{\partial out_{h1}} = 0.138498562 * 0.40 = 0.055399425$$

With the same way, $\frac{\partial E_{o2}}{\partial out_{h1}}$ will be

$\frac{\partial E_{o2}}{\partial out_{h1}} = -0.019049119$

Therefore:

$\frac{\partial E_{total}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial out_{h1}} + \frac{\partial E_{o2}}{\partial out_{h1}} = 0.055399425 + -0.019049119 = 0.036350306$

Next, the terms $\frac{\partial out_{h1}}{\partial net_{h1}}$ and then $\frac{\partial net_{h1}}{\partial w}$ for each weight must be calculated too.

$\frac{\partial out_{h1}}{\partial net_{h1}} = \frac{\partial(\frac{1}{1+e^{-net_{h1}}})}{\partial net_{h1}} = out_{h1}(1 - out_{h1}) = 0.59326999(1 - 0.59326999) = 0.241300709$

The partial derivative of the total net input to h_1 with respect to w_1 is

$net_{h1} = w_1 * i_1 + w_3 * i_2 + b_1 * 1$

$\frac{\partial net_{h1}}{\partial w_1} = i_1 = 0.05$

Putting it all together:

$$\frac{\partial E_{total}}{\partial w_1} = \frac{\partial E_{total}}{\partial out_{h1}} * \frac{\partial out_{h1}}{\partial net_{h1}} * \frac{\partial net_{h1}}{\partial w_1}$$

$$\frac{\partial E_{total}}{\partial w_1} = 0.036350306 * 0.241300709 * 0.05 = 0.000438568$$

Then, gradient descent can be applied for the w_1 parameter using the update rule in equation (106).

$$w_1' = w_1 - \eta * \frac{\partial E_{total}}{\partial w_1} = 0.15 - 0.5 * 0.000438568 = 0.149780716$$

Repeating this for w_2, w_3, and w_4

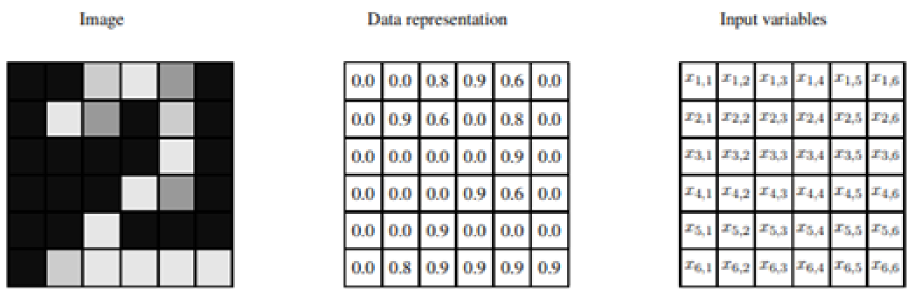
$$w_2' = 0.19956143$$

$$w_3' = 0.24975114$$

$$w_4' = 0.29950229$$

8.11 Convolutional Neural Networks

Convolutional neural networks (CNN) are specialized for applications in images (either color or grayscale) and video recognition. An image is made of the smallest not visible parts called pixels and every pixel has a force often known as the pixel intensity. Digital grayscale images are made of pixels ordered in a matrix. Each pixel can be depicted having limits from 0 (total absence, black) to 1 (total presence, white) and intermediate values to 0 and 1 correspond to different shades of gray. In figure 30 is presented an image with 6×6 pixels. In an image classification task, this image is the input and the pixels in the image are the input variables $x_{1,1}, x_{1,2}, ..., x_{6,6}$. The position of each pixel in the image is determined by the indicators j and k, as illustrated in figure 30 [108].



0.0	0.0	0.8	0.9	0.6	0.0
0.0	0.9	0.6	0.0	0.8	0.0
0.0	0.0	0.0	0.0	0.9	0.0
0.0	0.0	0.0	0.9	0.6	0.0
0.0	0.0	0.9	0.0	0.0	0.0
0.0	0.8	0.9	0.9	0.9	0.9

$x_{1,1}$	$x_{1,2}$	$x_{1,3}$	$x_{1,4}$	$x_{1,5}$	$x_{1,6}$
$x_{2,1}$	$x_{2,2}$	$x_{2,3}$	$x_{2,4}$	$x_{2,5}$	$x_{2,6}$
$x_{3,1}$	$x_{3,2}$	$x_{3,3}$	$x_{3,4}$	$x_{3,5}$	$x_{3,6}$
$x_{4,1}$	$x_{4,2}$	$x_{4,3}$	$x_{4,4}$	$x_{4,5}$	$x_{4,6}$
$x_{5,1}$	$x_{5,2}$	$x_{5,3}$	$x_{5,4}$	$x_{5,5}$	$x_{5,6}$
$x_{6,1}$	$x_{6,2}$	$x_{6,3}$	$x_{6,4}$	$x_{6,5}$	$x_{6,6}$

Figure 30: Data representation of a grayscale image with 6×6 pixels. Each pixel is represented with a number describing the grayscale color [108].

Colour images usually come with three color channels, i.e. the Red-Green-Blue channels, popularly known as the "RGB" values (Fig.31).In this case the data are 3-dimensional and an equivalent expression is that the depth is three [122]. The values of a pixel can be from 0 to 255, where "0" is white and "255" is the base color and these three layers together form a colored image where pixel can be determined in rgb terms. A combination of these three-color channels can produce all possible color pallets [123]. Color images having multiple color channels require processing of huge volumes of data making the process computationally hard and thus can be a difficult task [124].

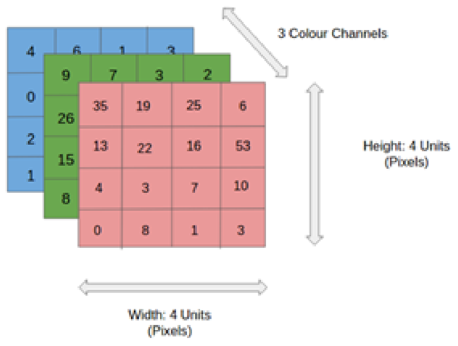


Figure 31: RGB color channels of an image [122].

The role of CNN is to reduce the images into a form that is easier to process, without losing features critical towards a good prediction. This makes convolutional networks fast to train. This, in turn, helps to train deep, many-layer networks, which are appropriate in classifying images. Convolutional neural networks use three basic ideas: local receptive fields, shared weights, and pooling [125].

8.11.1 Local Receptive Fields

In CNNs input pixels are connected to a layer of hidden neurons. Input pixels are not connected directly to every hidden neuron. Instead, connections are made in small, localized regions of the input image. More specific, each neuron in the first hidden layer will be connected to a small region of the input neurons, say, for example, a 5×5 region, corresponding to 25 input pixels. So, for a particular hidden neuron, there are connections that look like as illustrated in figure 32. That region in the input image is called the local receptive field for the hidden neuron. It is a little window on the input pixels [125].

Each connection learns a weight and a bias. The local receptive field slides into the entire input image. Each local receptive field mapped to a different hidden neuron in the first hidden layer as shown in figure 33, starting with a local receptive field in the top-left corner. Then the local receptive field is slides into by one pixel to the right (i.e., by one neuron), to connect to a second hidden neuron (Fig.34). And so on, building up the first hidden layer. In fact, sometimes a different stride length is used such as 2 pixels to the right (or down), in which case the stride length of 2 is used [125].

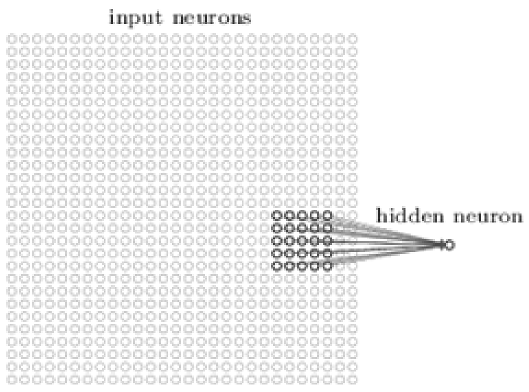


Figure 32: [125]

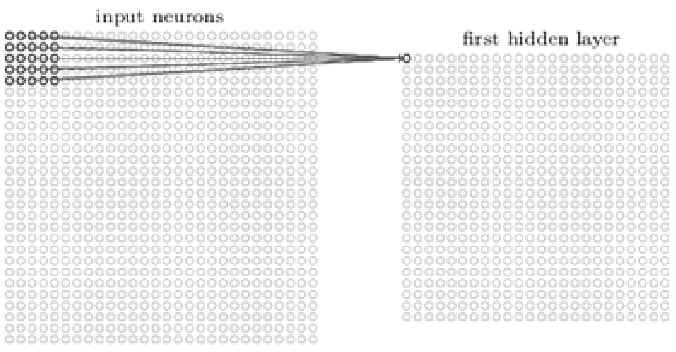


Figure 33: Sliding the local receptive field across the entire input image [125].

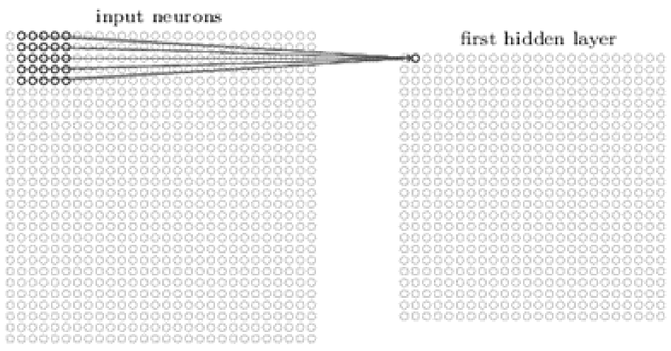


Figure 34: [125]

8.11.2 Shared Weights and Biases

Suppose that is given an input image containing 28×28 pixels and 5×5 local receptive fields. Here, each hidden neuron has a bias and 5×5 weights connected to its local receptive field and each hidden layer consists of one or more feature maps. The weights and bias that are used for each local receptive field in one feature map are the same. This feature map detects the same feature just at different locations in the input image rather than looking into every single pixel value. The weights defining the feature map are called shared weights. And the bias defining the feature map in this way is called shared bias. The shared weights and bias are often said that define a kernel or filter. Each filter produces a future map. Subsequently, the number of filters equals with the number of feature maps.

The neural networks learn those filters using the backpropagation algorithm. The more filters applied, the more information about the image content is acquired and the more features can be extracted. To do image recognition more than one feature map is necessary. And so, a complete convolutional layer consists of several different feature maps. In figure 35 illustrated an example where each feature map is defined by a set of 5×5 shared weights, a single shared bias and applied 3 filters. In practice convolutional networks may use more (and perhaps many more) feature maps [125].

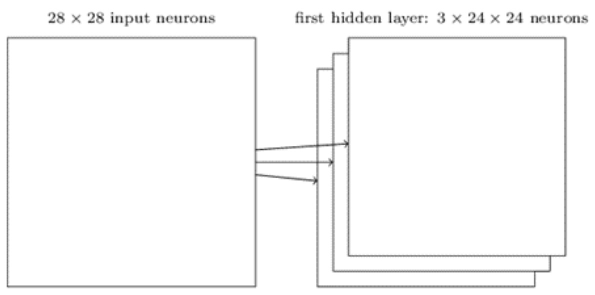


Figure 35: Several different feature maps [125].

A big advantage of sharing weights and biases is the significant reduction to the number of parameters involved in a convolutional network. In the case which there is 28×28 input image and 5×5 local receptive fields for each feature map are needed $25 = 5 \times 5$ shared weights, plus a single shared bias. So, each feature map requires 26 parameters. Suppose that 20 feature maps are required, that is a total of $20 \times 26 = 520$ parameters defining the convolutional layer. By comparison, can be defined a case with a fully connected first layer containing $784 = 28 \times 28$ input neurons, and a relatively modest 30 hidden neurons. That is a total of 784×30 weights, plus an extra 30 biases, for a total of 23,550 parameters. In other words, the fully connected layer would have more than 40 times as many parameters as the convolutional layer. It cannot be really done a direct comparison between the number of parameters, since the two models are different in essential ways. Intuitively, it seems likely that the use of convolutional layers will reduce the number of parameters to get the same performance as the fully connected model. That, in turn, will result in faster training for the convolutional model, and, ultimately, will help build deep networks using convolutional layers [125].

Inputs images might have more than one channel such as the case in the figure below which contains RGB image. Here, two filters are applied but as shown each filter has a depth of 3 which is the same depth of the input image. In general, the filter depth should always be equal to depth of the previous layer [126].

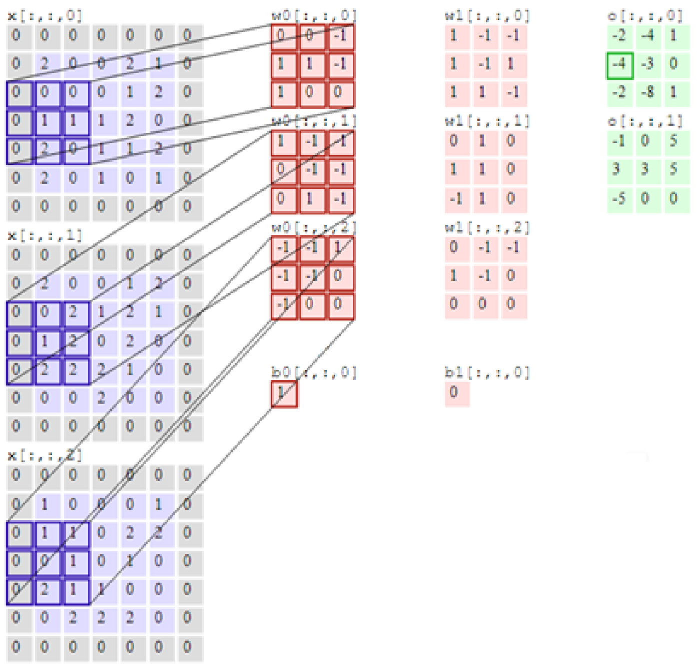


Figure 36: A Convolution filter [126].

8.11.3 Pooling Layers

Convolutional neural networks also contain pooling layers. Pooling layers are used immediately after convolutional layers. Pooling is used to reduce the size of the feature map while still maintain the important features.

In detail, a pooling layer takes each feature map output from the convolutional layer and prepares a condensed feature map. For instance, each unit in the pooling layer may summarize a region say of 2×2 neurons of the previous layer. The dimension of this region is named pooling size. In figure 37 the pooling size equals 2. One common procedure for pooling is known as max-pooling. In max pooling, a pooling unit simply outputs the maximum activation in the 2×2 input region, as illustrated in figure 37. In a similar way, when average pooling is used, the average of each block is taken. In general, the output size after performing pooling is the input size divided by the pooling size [125].

$$Outputsize = \frac{input\ size}{pooling\ size}$$

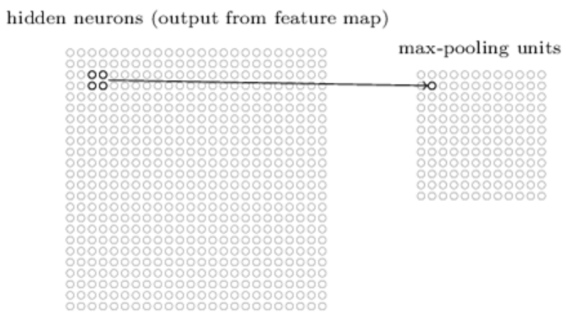


Figure 37: Max pooling [125].

As mentioned above, the convolutional layer usually involves more than a single feature map. Pooling is applied to each feature map separately. So, if there were three feature maps, the combined convolutional and pooling layers look like the example of figure 38 [125].

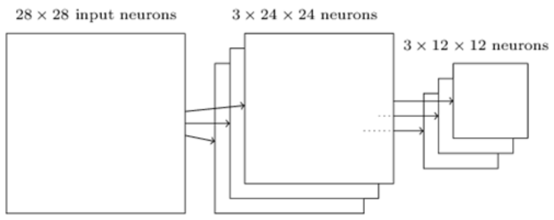


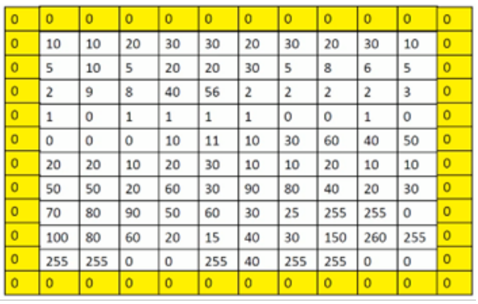
Figure 38: Combined convolutional and max-pooling layers [125].

A great benefit of pooling is that there are many fewer pooled features, and so this helps to reduce the number of parameters needed in later layers. Max-pooling and average pooling are not the only techniques used for pooling. Another common approach is known as L2 pooling. Here, instead of taking the maximum activation of a 2×2 region of neurons, is used the square root of the sum of the squares of the activations in the 2×2 region. While the details are different, the intuition is like max and average pooling: $L2$ pooling is a way of condensing information from the convolutional layer. All techniques which are used for pooling have their advantages and disadvantages depending on the application or on the layer is used, whether it is the output layer or that layer is used at the beginning, at the middle or at the end [125].

8.11.4 Zero -Padding

Sometimes is needed to add a frame with zeros around the image (Fig.39). This technique is called zero padding. After convolution the image get shrinked and after filtered the result is a very small image. Also, by sliding the filter into the image the pixel in the corner is covered only one time while the middle pixel get covered more than once. This means that there is more information on the middle pixels and probably few values are affected by pixels as the edges of an image. Zero padding is an addition frame of zeros that is added to the border of an image [127].

Whether or not use padding is specified in each convolutional layer. Sometimes is needed a double or triple border of zeros to maintain the original size of the input, depending on the size of the input and the size of the filters [128].



0	0	0	0	0	0	0	0	0	0	0	0
0	10	10	20	30	30	20	30	20	30	10	0
0	5	10	5	20	20	30	5	8	6	5	0
0	2	9	8	40	56	2	2	2	2	3	0
0	1	0	1	1	1	1	0	0	1	0	0
0	0	0	0	10	11	10	30	60	40	50	0
0	20	20	10	20	30	10	10	20	10	10	0
0	50	50	20	60	30	90	80	40	20	30	0
0	70	80	90	50	60	30	25	255	255	0	0
0	100	80	60	20	15	40	30	150	260	255	0
0	255	255	0	0	255	40	255	255	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0

Figure 39: Zero-padding with one border of zeros [127].

Zero padding allows to use a convolutional layer without necessarily shrinking the height and width of the volumes. This is important for building deeper networks, since otherwise the height/width would shrink as you go to deeper layers [129].

When building a convolutional neural network the prior knowledge of how the output size affected by the filter size and the padding amount that must be used is useful. Input size can be denoted by n, filter size by f, padding by p and stride by s. When a $n \times n \times c$ input is convolved with a $f \times f \times c$ filter with the stride s and padding p the generated output size will be [130]

$$\left(\frac{n+2p-f}{s}+1\right) \times \left(\frac{n+2p-f}{s}+1\right) \times 1$$

In some cases $\frac{n+2p-f}{s}$ is not integer thus we will take the floor value.

8.11.5 Shifting Invariance

CNNs are invariant to shifting. For example, having a training image with a cat figure 40, which is placed on the right side of the image and at testing time this image is encountered, which is also a cat, but this time is placed on the left side of the image [131]. This shifting does not affect network's prediction. So whatever shift there is for a couple of pixels or for the main object the prediction is not affected. This happens because the filters have learned to extract the relevant features anywhere in the image [132].

After having create the convolutional layers, the final step is to apply fully connected layers. The last feature maps that are gotten after pooling are flattened it out. The first fully connected

Figure 40: [131]

layer connects every neuron from the pooled layer to every one of the hidden neurons. Several processing layers can be used, just layers, just like in the feed forward. This fully connected architecture is the same as used in earlier chapters.

9 Ensemble learning

9.1 Introduction

Ensemble learning can be broken down into two tasks: developing a population of base learners from the training data that each get slightly different results on a dataset, and then combining them to form the composite predictor. Any classifier can be used as base learner in ensemble learning. The learners can be performed in different ways ensuring that they see different things. Cross validation can be used separating the data in a different way for each learner or just the data is randomly separated giving different sets to different learner when many of them are given. Ensemble methods do very well not only when there are many data, but also in the case that data are few. The results from an ensemble method can be combined in different ways. An effective one is majority voting which as described before means that the final output is the one having the majority values [133].

Ensemble methods can mainly be divided into two groups: In parallel ensemble methods where base learners are generated in parallel. In sequential ensemble methods where base learners are generated sequentially [134]. Most often a single base learning algorithm is used so that the weak learners that are trained in different ways are homogeneous. The ensemble model which obtained is then called “homogeneous”. However, some methods use not a single of base learning algorithms: some heterogeneous weak learners are then combined into an “heterogeneous ensembles model”[135].

9.2 Boosting

The most popular ensemble method is boosting. In boosting a random sample of data is selected, fitted with a base learner and then trained sequentially— that is, a series of learners are constructed and with each new iteration, the weights of the misclassified data in the previous learner are increased. This redistribution of weights helps the algorithm identify the parameters that it needs to focus on to improve its performance [136].

These learners are “weak” classifiers each performing only just better than chance and their outputs are combined to produce an ensemble learner that can perform arbitrarily well [133]. The principal algorithm of boosting is named AdaBoost.

9.2.1 AdaBoost

The innovation that AdaBoost (which stands for adaptive boosting) uses is to give weights to each datapoint according to how difficult previous classifiers have found to get it correct. These weights are given to the classifier as part of the input when it is trained. Thereby, a sequence of weak classifiers $G_m(x), m = 1, 2, ..., M$ is produced to repeatedly modified versions of data.

The weights are initially all set to the same value, $1/N$, where N is the number of datapoints in the training set. At step m, those observations that have been misclassified by the classifier

$G_{m-1}(x)$ induced at the previous step have their weights increased, whereas the weights are decreased for those that were classified correctly. At each iteration m, the error (ϵrrm) is computed as the sum of the weights of the misclassified points, divided by the weights of all points and the weights for incorrect examples are updated by being multiplied by $(1-errm)/errm$. Thus, as iterations proceed, observations that are difficult to classify correctly receive ever-increasing influence. Training terminates after a set number of iterations, or when either all the datapoints are classified correctly, or one point contains more than half of the available weight. The predictions from all of them are then combined through a weighted majority vote to produce the final prediction [133]:

$$G(x) = sign(\sum_{m=1}^{M} a_m G_m(x)) \tag{127}$$

Coefficients $\alpha_1, \alpha_2, ..., \alpha_M$ are computed using the formula

$$a_m = log(\frac{1 - err_m}{err_m}) \tag{128}$$

and weight the contribution of each respective $G_m(x)$. Their effect is to give higher influence on the more accurate classifiers in the sequence. In summary the steps of Adaboost algorithm described above [137]:

```
1. Initialize the observation weights $w_i = 1/N, i = 1, 2, ..., N$.
2. For $m = 1$ to $M$:
    (a) Fit a classifier $G_m(x)$ to the training data using weights $w_i$ .
    (b)Compute $Errm = \frac{\sum_{i=1}^{N} w_i I(y_i \neq G_m(x_i))}{\sum_{i=1}^{N} w_i}$
    (c)Compute $a_m = log\frac{1-err_m}{err_m}$
    (d)Set $w_i = w_i e^{a_m I(y_i \neq G_m(x_i))}$
3. Output $G(x) = sign(\sum_{m=1}^{M} a_m G_m(x))$
```

9.3 Bagging

One of the most common methods in the Parallel ensemble technique is Bootstrap Aggregating (Bagging). Bagging can be applied both for regression and classification problems. Lots

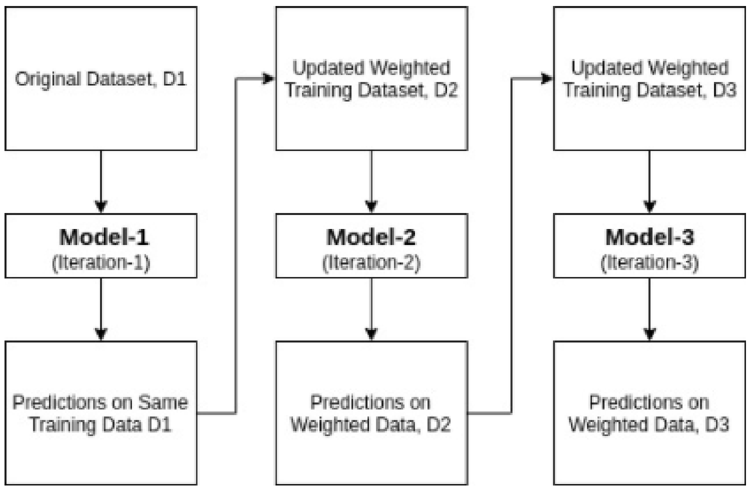

Original Dataset, D1
Updated Weighted Training Dataset, D2
Updated Weighted Training Dataset, D3
Model-1 (Iteration-1)
Model-2 (Iteration-2)
Model-3 (Iteration-3)
Predictions on Same Training Data D1
Predictions on Weighted Data, D2
Predictions on Weighted Data, D3

Figure 41: Adaboost training process [138].

of learners are generated in paraller by training each one in slightly different bootstrap samples. Then in classification problem, they are combined by taking the output to be the majority vote of all the classifiers [139]. In regression problems the bagged estimate is the average prediction of all the classifiers' outputs. As a result, bagging algorithm helps to reduce the variance of the prediction of a model [140]. An example of bagging ensemble method is Random Forests where m different trees are trained on different random subsets of the data and perform voting for final prediction. Random forests are presented below precisely.

Bootstrap sample

A bootstrap sample is a sample taken from the original dataset with replacement, so that some data may be obtained several times and others not at all. The bootstrap sample is the same size as the original, and lots and lots of these samples are taken, at least 50, and could even be in the thousands [139].

9.3.1 Random forests

In bagging, a number of decision trees are created where each one is created from a different bootstrap sample of the training dataset and in turn, has a slightly different performance. Trees used in the ensemble are unpruned, making them slightly overfitted to the training dataset. This is not a problem, as it makes the trees dissimilar and resulting in less correlated predictions or prediction errors. The final prediction is the average of all trees and thus, a better performance is achieved compare to any single tree in the model [141].

Random forests is a substantial modification of bagging that builds a large collection of de-correlated trees, and then averages them [142]. To create a forest, different trees are created by training them on slightly different data, so bootstrap samples are taken from the dataset as inputs to each tree. Unlike bagging, random forest limits the choices that the decision tree can make. This means that at each node, a random subset of the features is given to the tree, and it can only pick from that subset rather than from the whole set [143]. In practice, given that p is the number of features, a subset size $m = \sqrt{p}$ is used for classification problems while $m = \frac{p}{3}$ is used for regression problems [142].

By reducing the features to a random subset that may be considered at each split point, each decision tree in the ensemble becomes more different. The effect is that the predictions, and in turn, prediction errors, made by each tree in the ensemble are more different or less correlated. When the predictions from these less correlated trees are averaged to make a prediction, it often results in better performance than bagged decision trees [141].

Increasing the randomness in the training of each tree also speeds up the training, since there are fewer features to search over at each stage and reduces the variance without effecting the bias. Another benefit of this is that there is no need to prune the trees. However, there is no way to choose the number of trees to put into the forest it, instead trees are built until the error stops decreasing. When they are used for classification, a random forest obtains a class vote from each tree, and then classifies using majority vote. When they are used for regression the predictions from each tree at a target point x are simply averaged [143].

The steps of the algorithm in summary [143].

```
• For each of $N$ trees:

– create a new bootstrap sample of the training set

– use this bootstrap sample to train a decision tree

– at each node of the decision tree, randomly select $m$ features, and compute the information gain (or Gini impurity) only on that set of features, selecting the optimal one

– repeat until the tree is complete
```

9.4 Bagging vs Boosting

As already presented one main difference between bagging and boosting methods is the fact that in bagging the algorithms are generated in paraller while in boosting are generated sequentially. The following figure illustrates the different processes [134] .

Another difference between bagging and boosting is the way they are used. Bagging methods are usually used on weak learners that show signs of high variance and low bias, while on the contrary boosting methods are applied when low variance and high bias exists. While bagging

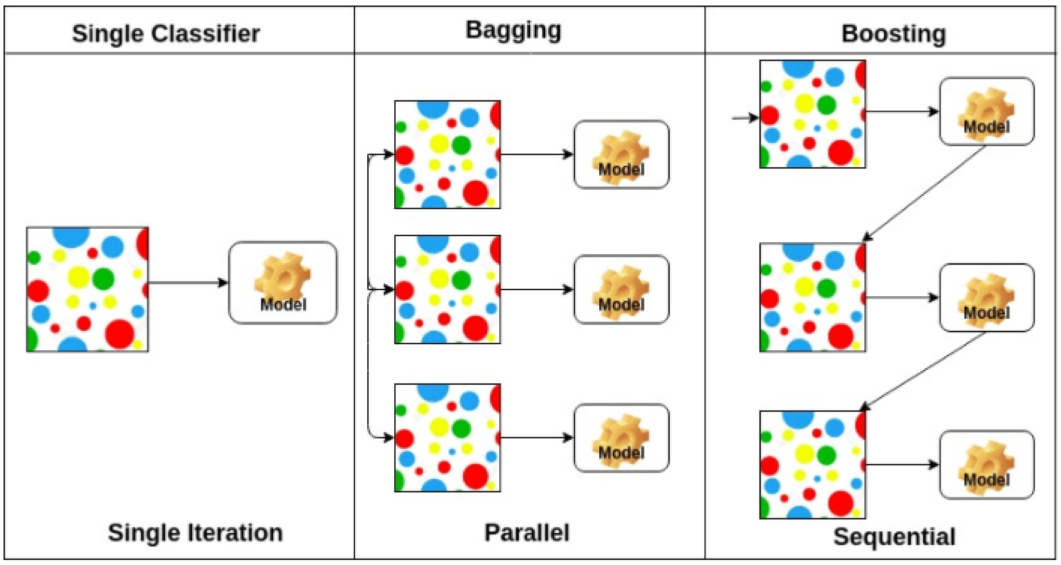


Figure 42: Bagging vs Boosting [134].

can be used to avoid overfitting, boosting methods can be more prone to this although it really depends on the dataset. However, parameter tuning can help avoid the issue. As a result, bagging and boosting have different real-world applications as well. Bagging has been leveraged for loan approval processes and statistical genomics while boosting has been used more within image recognition apps and search engines [136].

10 Hands-on Projects

10.1 Introduction

Three data sets were used for building in practice some of the aforementioned methods and others beyond them. Python programming language was used for algorithms' development. All the methods and steps that applied in the building process are described in (10.2) section.

10.2 Methods and Techniques

In the beginning of each process were calculated some descriptive statistics and graphs like barplots and histograms for understanding the distribution of the data according to the target variable.

Splitting the data

The data were divided into training and test set. Each algorithm was built exclusively using the training data. The test set was used to evaluate the final model's performance on completely unseen data.

Imputing

The first pre-processing step was to search for missing values. Missing values for a categorical variable can be replaced by the majority value of the feature. Missing values for a numerical value can be replaced by the median or mean value of each variable. Columns and rows containing more than 80% and 70% missing values respectively were deleted.

Onehotencoding

Subsequently, categorical variables having more than two categories were represented by dummy variables and the numerical variables were standardized. However, these steps are not necessary for all algorithms. Tree-based algorithms like Random Forests, XGBoost etc require neither dummies nor standardized values. On the contrary , algorithms like Logistic Regression, Support Vector Machines, and K nearest Neighbors which were also developed require both one-hot encode and standardization. Hence, different pre-processing strategies were used depending on the algorithm. For a large number of features one-hot-encoding can lead to a dramatically dimensionality and complexity increase. One way to deal this issue is to select the first *k* most frequent categories and represent them with dummies.

Recursive feature elimination

Next, feature selection was applied using Recursive feature elimination which is a backward selection method and is an efficient approach for eliminating features from a training dataset for feature selection. It works by searching for a subset of features by starting with all features in the training dataset and successfully removing features until the desired number remains. The features are ranked by importance according to an algorithm called estimator, the least important features are dropped, and the estimator is re-fitted until the specified number of features remains.

Rfe hyperparameters

The number of features and the estimator are tuning parameters. They were selected based on a specific strategy. The algorithm with the best results in the model selection process was applied in combination with RFE for different numbers of features to select. The number of features that produced the best result was not the final option but was used to select the estimator for ranking features by importance among many candidates like Decision Tree, Perceptron, Random Forests, Gradient Boosting, Logistic Regression etc. The estimator was chosen comparing the performance of the best model in model selection process in combination with feature selection for different candidates. The candidate that produce the best results was the final choice. However, if the estimator was a 'slow' algorithm and there was no significant different with a faster candidate then the second would be selected to speed-up the training process.

The final number of features was determined next using Bayesian optimization. The model with the optimal hyperparameters is a different algorithm and thus, the number of features must be explored again and determined at the same time with the hyperparametes of the model. Similarly, the estimator for ranking features could be explored using Bayesian optimization. However, it was noticed that the same algorithm produced the best results irrespective of the hyperparameters of the classifier. Hence, there was no reason to put extra candidates hyperparameters when doing Bayesian optimization making the process more computationally expensive than it was already.

Imbalanced data

Two of three data sets were imbalanced meaning that the one class called majority had a significantly higher number of observations compared to the other called minority. When that happens, prediction of the minority is harder since few data points are given and the algorithm ends up over-biasing the majority class. However, when working with an imbalanced classification problem, the minority class is typical of the most interest. To deal this issue, different preprocessing strategy was applied including resampling techniques, stratified kfold was used instead of kfold for model selection and hyperparameter tuning based on different evaluation metric. These methods are presented below extensively.

Evaluation metrics

Accuracy is not an appropriate evaluation metric for imbalanced data and can lead to misleading results. On the contrary, the F1 score is a more robust evaluation metric in imbalanced data and combines the precision and recall metrics into a single metric. As a consequence, a high F1 score ensures that both classes are taken into account by the algorithm. There are four types of results in binary classification: True Positives (TP), True Negatives (TN), False Positives (FP), and False Negatives (FN). Accuracy and F1 metrics are defined as

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN} \tag{129}$$

$$F1 = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{130}$$

where

$$Precision = \frac{TP}{TP + FP} \tag{131}$$

and

$$Recall = \frac{TP}{TP + FN} \tag{132}$$

K-fold/Stratified k-fold

K fold cross validation was used for model selection and hyperparameter tuning. In K fold cross validation,the data is separated in k parts and in each iteration $k - 1$ parts are used as a training set and the remaining as a test set. This process is repeated until all parts used as a train and test set. The mean of k estimates is considered as the final result. In the case of imbalanced data is dangerous to use k fold cross validation because the distribution of the data is not taken into account. Consequently, the amount of the minority class to some of k parts can be even lower than the original. To this end, stratified k-fold was used which is a more appropriate method, operates in the same way as k-fold, but ensures that the ratio of each of the k parts will be equal to the original ratio.

Resampling

Undersampling is a useful technique to balance the imbalanced data. It was performed during the training process using Random Undersampling. Random undersampling simply deletes data randomly until it gets the desired ratio. This ratio is a hyperparameter and was tuned during the training process.Although is a very simple method, is also very fast and in the cases that are presented below produced equal results to others undersampling methods like Onesidedselection, Tomek or oversampling methods like Smote which are more time-consuming.

Data leakage

Resampling techniques like undersampling or oversampling must be performed exclusively in the training data. Otherwise, the distribution of test data will change dramatically leading to completely inaccurate results. Similarly, standardscaler or feature selection algorithm must be fitted only in the training data. This serious mistake is called data leakage and it means that the test data have been exposed and have been used accidentally during the training process. When cross-validation is not used is easy to avoid data leakage. The data are just split into train and test set, the pre-processing steps and the model are fitted on the training data, then the test data are transformed using the necessary steps like standardization and the model is applied to the test data. However, in the case that $k-$ fold cross-validation or some of its variations is used the training process is more complex. To avoid data leakage is required to ensure that in each of k iterations the pre-processing steps will be fitted on the $k-1$ parts which is the training set and not to the remaining k part. Data leakage was prevented using pipelines.

Pipelines help to avoid data leakage ensuring that during cross-validation the test data are not exposed and do not learn anything from training data. They have been used for the development of all three algorithms. They are objects that hold all the steps of data preparation and modeling and execute them sequentially. Pipelines also ensure that the sequence of the steps are executed once and make the training process easier since there is no need to execute manually each step [144].

Anomaly Detection

Isolation forest was applied to search for anomaly data points in training sets, but did not improve the results. The performance not only was not improved but decreased probably because the values considered anomalies and deleted provided valuable information to the algorithms.

Isolation Forest uses an ensemble of Isolation Trees to detect anomaly data points. The Isolation Forest algorithm is based on the idea that anomalies observations are fewer and different, therefore they can be recognized faster and easily. The isolation tree is built dividing the training data at random value from a randomly selected feature in an iterative manner, to isolate data points from each other. An anomalous point is considered the one which have been isolated much faster than the other points. This is quantified through an outlier score corresponding to each point which is calculated by the algorithm, based on the number of splits are needed to its isolation. If the score is less than zero the points is considered as anomally. Otherwise, a threshold can be defined instead of zero [145].

Model selection

In each of the three data sets, various algorithms were performed and evaluated. In this process, all had their default hyperparameters and they were trained using k-fold or stratified k-fold cross validation. Two algorithms with the best results were selected for further optimization.

Bayesian Optimization

Bayesian Optimization was used for hyperparameter tuning. This method can importantly speed-up the optimization process compared to common optimization techniques like Grid Search and brings better generalization performance on the test set. This is achieved by taking into account information by previous hyperparameter combinations when choosing the hyperparameter set to evaluate next. Two popular libraries for Bayesian Optimization include Scikit-Optimize and HyperOpt. In the following three applications Scikit-Optimize was used with BayesSearchCV class.

10.3 Project 1

The first data set comes from a database called Cleveland [146] and consists of 13 features and a target variable collected by 303 patients. The aim was to create a classifier for predicting patients with more chance to heart attack. The target variable is binary having classes 0 = no/less chance of heart attack and 1 =more chance of heart attack. The whole data is represented in figure 43.

Some descriptive statistics about the numerical variables are presented in table 31. The distribution of the target variable is presented in the first graph in figure 44 and it seems that the data are not imbalanced. In the next graphs from the same figure are presented the distributions of the categorical variables according to the target variable. It seems that many of them have different distributions between the 2 levels of the target variable. Moreover, in figure 45 are presented the means of each numerical variable according to the target variable. The next graphs from the same figure are histograms for the numerical variables according to the target variable. Again, there are variables that appear to be differently distributed between the 2 levels of the target variable.

The target variable was separated from the dataset to a variable named y and the remaining to a data frame named X. The pre-processing strategy was not the same for each model. One hot encoding was applied for categorical features having equal or more than 3 classes. Standardization was applied before Support Vector Machine and Logistic Regression. Then, feature selection using RFE was applied. For model selection and hyperparameter tuning 10-Fold cross validation was used based on accuracy metric. All the pre-processing steps and classifiers were included in pipeline in the order mentioned above to avoid data leakage.

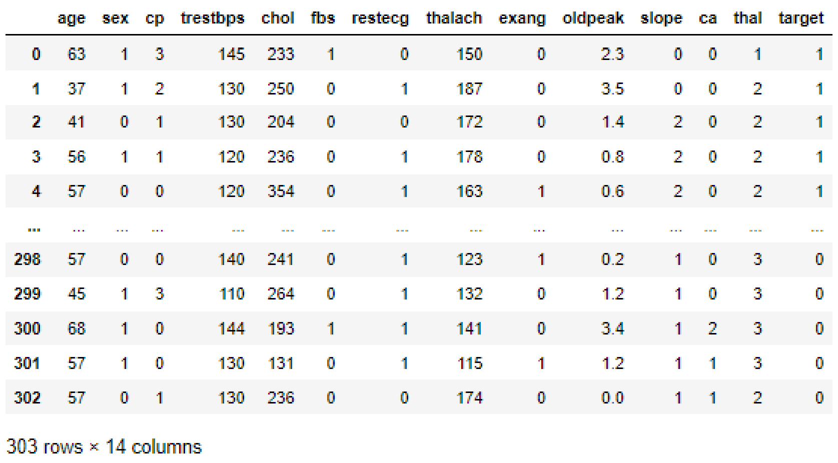
Random Forest, XGBoost, Adaboost using base classifier Decision Tree, Support Vector Machine and Logistic Regression, Gaussian Naïve Bayes and a Voting classifier combining Logistic Regression, Decision Tree and KNeighborsClassifier, were performed and evaluated having their default hyperparameters. In this process, all the pre-processing steps were applied except for feature selection. Support Vector Machine and Logistic Regression had the best results, and they were chosen for the next steps.

The next step was to select the estimator for ranking the features using Recursive feature elimination. Logistic Regression was chosen among Perceptron, Decision Tree, XGBoost and Random Forest. Next, hyperparameter tuning using Bayesian optimization for Logistic Regression and SVM was applied.The models with the default and the optimized parameters were fitted on the training set and then applied to the test set. Having deafault hyperparameters, accuracy score for Logistic Regression and SVM was 0.803 and 0.819 respectively. The results of the two models are presented in table 33.

The optimized parameters for the SVM model using all the features was $C = 10$ and $gamma = 0.001$ and achieved accuracy equal to 0.836. Using feature selection, 6 features were selected to be the most important. The best parameters for the SVM model using those 6 features was the same as the full model and achieved accuracy equal to 0.852. Also, for the same model the ROC curve is presented in figure 45 and the AUC was 0.91.

The optimized parameters for the Logistic Regression model using all the features was $C = 0.01, solver =' liblinear'$ and achieved accuracy equal to 0.836. Using feature selection, 6 features were selected to be the most important. The best parameters for the Logistic Regression model using those 6 features was $C = 0.1, solver =' lbfgs'$ and achieved accuracy equal to 0.836. Also, for the same model the ROC curve is presented in figure 46 and the AUC was 0.912.

The features which selected from the two models was the same and are presented in table 34. 'Slope' was the slope of the peak exercise ST segment and had 3 categories, 'thal' was Thalium Stress Test result with 3 categories (0 = normal; 1 = fixed defect; 2 = reversable defect), 'cp' was chest pain type such that 0 = Typical Angina, 1 = Atypical Angina, 2 = Non-anginal Pain, 3 = Asymptomati,'ca' was number of major vessels, 'oldpeak' was the Previous peak and 'exang' was Exercise induced angina such that 1 = Yes, 0 = No.



	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1	1
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2	1
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2	1
3	56	1	1	120	236	0	1	178	0	0.8	2	0	2	1
4	57	0	0	120	354	0	1	163	1	0.6	2	0	2	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
298	57	0	0	140	241	0	1	123	1	0.2	1	0	3	0
299	45	1	3	110	264	0	1	132	0	1.2	1	0	3	0
300	68	1	0	144	193	1	1	141	0	3.4	1	2	3	0
301	57	1	0	130	131	0	1	115	1	1.2	1	1	3	0
302	57	0	1	130	236	0	0	174	0	0.0	1	1	2	0

303 rows × 14 columns

Figure 43: Dataset project 1.

	age	**trestbps**	**chol**	**thalach**	**oldpeak**
count	303.000000	303.000000	303.000000	303.000000	303.000000
mean	54.366337	131.623762	246.264026	149.646865	1.039604
std	9.082101	17.538143	51.830751	22.905161	1.161075
min	29.000000	94.000000	126.000000	71.000000	0.000000
25%	47.500000	120.000000	211.000000	133.500000	0.000000
50%	55.000000	130.000000	240.000000	153.000000	0.800000
75%	61.000000	140.000000	274.500000	166.000000	1.600000
max	77.000000	200.000000	564.000000	202.000000	6.200000
skew	-0.201	0.710	1.138	-0.535	1.263
kurtosis	-0.553	0.894	4.412	-0.081	1.530

Table 31: Descriptive statistics.

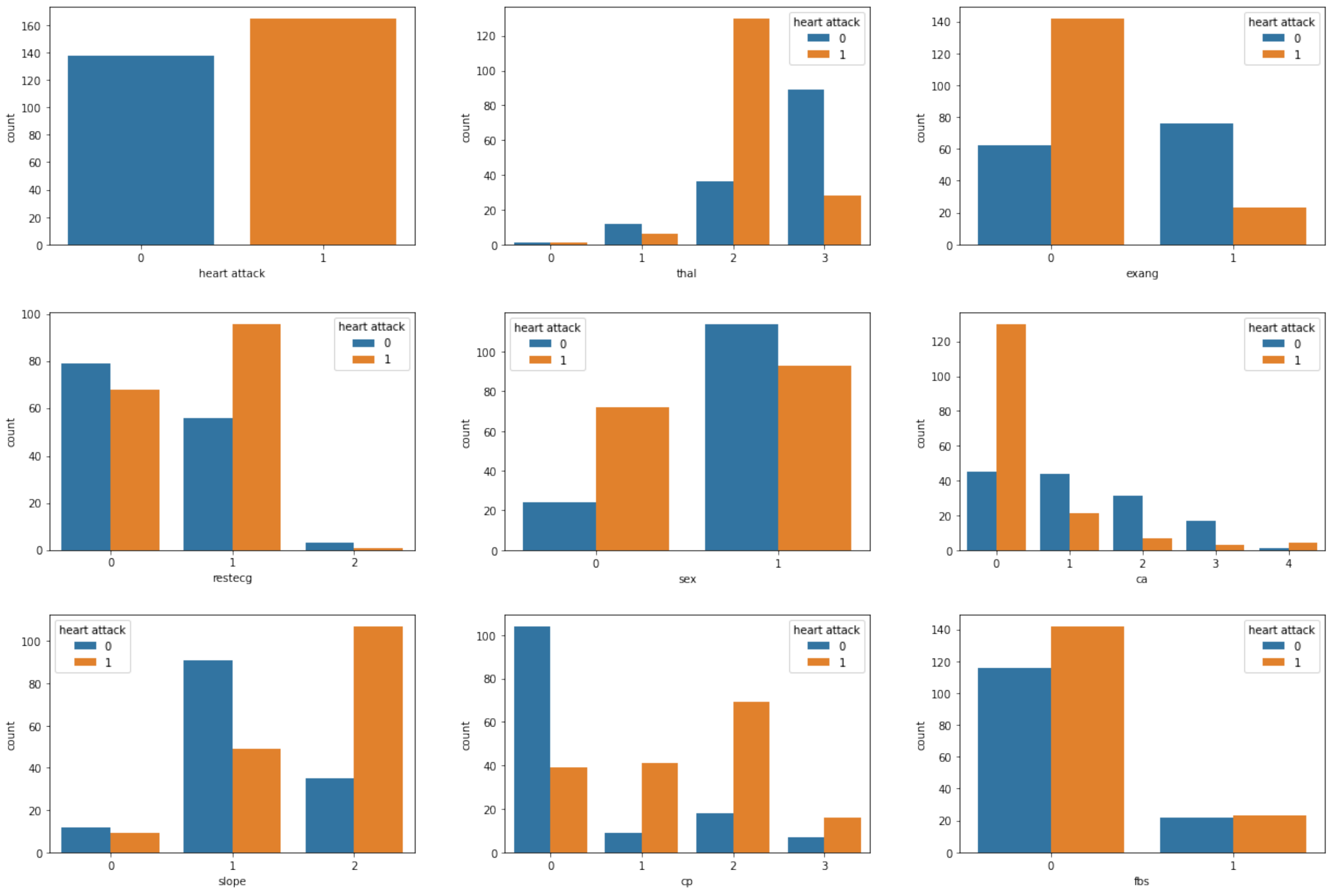


Figure 44: Plots for categorical variables.

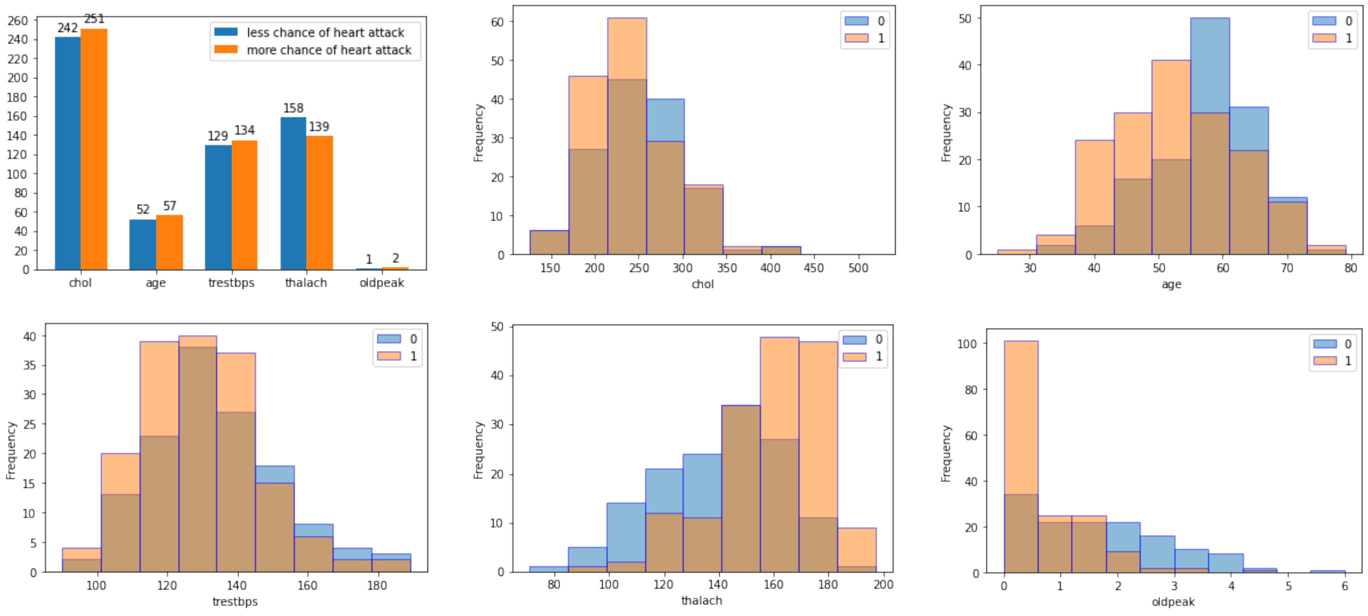


Figure 45: Plots for numerical variables.

Model	Accuracy
Random Forest	0.825
Decision Tree	0.761
Xgboost	0.802
SVM	**0.835**
Logistic Regression	**0.843**
Adaboost	0.786
Voting	0.81

Table 32: Models comparison.

Model	Default Parameters	Full optimized model	Using feature selection		
	Accuracy	Accuracy	Accuracy	Sens	Spec
SVM	0.819	0.836	0.852	97%	71.42%
Logistic Regression	0.803	0.803	0.836	97%	67.85%

Table 33: Evaluation process.

SVM/Logistic Regression
Slope
thal
cp
exang
oldpeak
ca

Table 34: Selected Features.

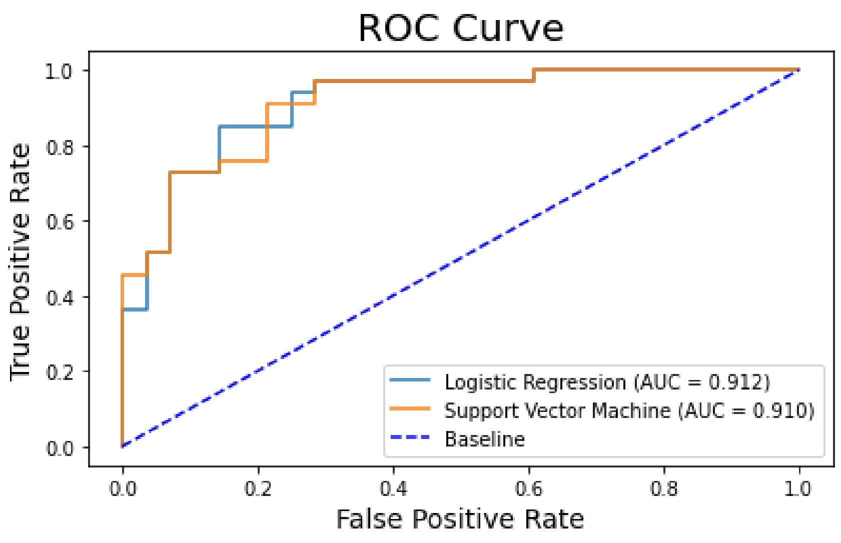


Figure 46: ROC Curves for the 2 optimal models.

10.4 Project 2

The second dataset comes from The Behavioral Risk Factor Surveillance System (BRFSS) which is a health-related telephone survey that is collected annually by the CDC. Each year, the survey collects responses from over 400,000 Americans on health-related risk behaviors, chronic health conditions, and are used for preventative services. For this project, it was chosen the data set for the year 2015 containing 21 feature variables and a target variable to be used for the binary classification of heart disease. The target variable is binary having classes 0 =do not have/have not had heart disease and 1 = had heart disease. The features are either binary or ordinal and there are no missing values. The whole data is represented in figure 47.

Some descriptive statistics are presented in table 36. The distribution of the target variable is presented in the first graph in figure 48 and it seems that the data are highly imbalanced. Actually, there is a strong class imbalance since 229,787 participants do not have/have not had heart disease while 23,893 have had heart disease. In the next graphs from the same figure are presented the distributions of some categorical variables according to the target variable. Moreover, in figure 49 are presented histograms for some variables according to the target variable.

The target variable was separated from the dataset to a variable named y and the remaining to a data frame named X. The data was separated into train 80% and test data 20% and using the command $stratify = y$ the train and test set had a ratio equal to the target variable. Standardization was applied before Logistic Regression,Adaboost using base classifier Linear SVM and Adaboost using base classifier Logistic Regression. After standardization, feature selection using RFE was performed. The next and final preprocessing step before fitting the algorithm was to undersample the data using RandomUnderSampler in python creating a ratio in $1 : 2$ in training set. For model evaluation and hyperparameter tuning Stratified 5-fold cross validation was used based on $f1$-measure metric. All the preprocessing steps and classifiers were included in the pipeline in the order mentioned above to avoid data leakage.

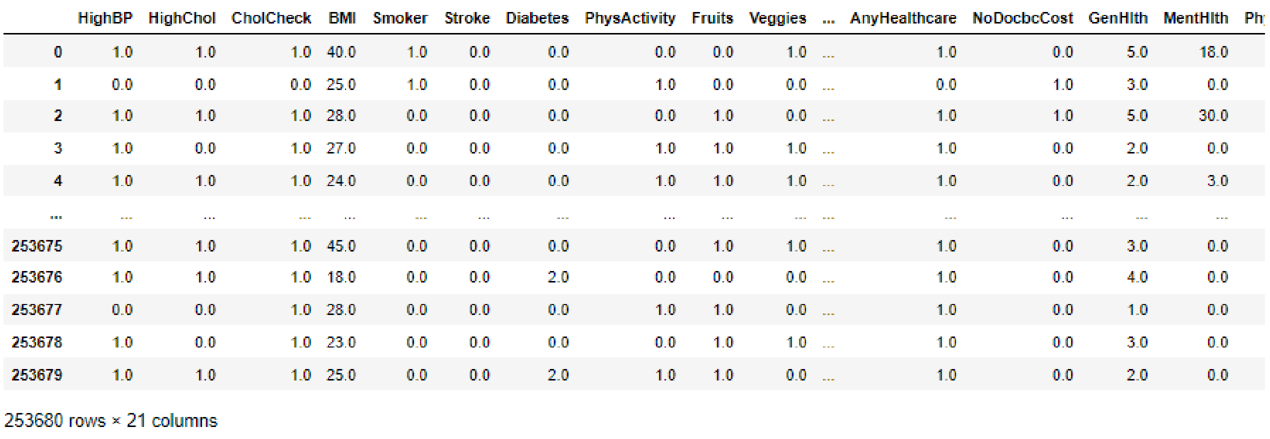
Balanced Bagging Classifier,Logistic Regression, Decision Tree, Random Forest, XGBoost, Adaboost using base classifier Decision Tree, Adaboost using base classifier Linear SVM and Adaboost using base classifier Logistic Regression were performed and evaluated having their default hyperparameters. In this process, all the preprocessing steps were applied except for feature selection. Adaboost using base classifier Logistic Regression and XGboost had the best results and they were selected for further optimization.

The next step was to select the best estimator for ranking the features using Recursive feature elimination.Logistic Regression was chosen between Perceptron and Decision Tree. Next, hyperparameter tuning using Bayesian optimization for Adaboost and XGBoost was applied. The models were fitted on the training set and then applied to the test set. Having default hyperparamaters, F1 score for Adaboost and Xgboost without undersampling was just 0.09 and 0.19 respectively. After undersampling there was a significant improvement to 0.377 and 0.405 respectively as illustrated in figure 66. The results of the two models are presented in table 38.

The optimized parameters for the Adaboost model using all the features was C=100,learning_rate=0.6, n_estimators=400 and achieved F1 equal to 0.418 and accuracy 0.85. Using feature selection, 9 features (table 39) were selected to be the most important. The best parameters for the Logistic Regression model using those 9 features was C=1000, max_iter=400, learning_rate=0.6, n_estimators=200 and achieved F1 equal to 0.417 and accuracy was 0.84. Also, for the same model the ROC curve is presented in figure 50 and the AUC was 0.748.

The optimized parameters for the XGBoost model using all the features was colsample_bylevel=0.01, colsample_bytree=0.7, gamma=0.2, learning_rate=0.05, max_depth=3, min_child_weight=1, n_estimators=300, subsample=1 and achieved F1 equal to 0.41 and accuracy 0.83. Using feature selection, 11 features (table 38) were selected to be the most important. The best parameters for the XGBoost model using those 11 features was colsample_bylevel=1, colsample_bytree=0.7, gamma=0.4, learning_rate=0.05, max_depth=3, min_child_weight=7, n_estimators=200, subsample=1 and achieved F1 equal to 0.42 and accuracy was 0.83. Also, for the same model the ROC curve is presented in figure 50 and the AUC was 0.85.

The features which selected from both models are presented in table 39. Using Adaboost were selected the followin 'HighBP' was the blood pressure (high) , 'HighChol' was cholesterol (high), 'Smoker' was smoking, 'HvyAlcoholConsump' was alcohol consumption and 'GenHlth' was general health,'stroke' was (Ever told) someone had a stroke , 'sex' was Indicate sex of respondent,' DIFFWALK' was the difficulty walking or climbing , 'CholCheck' was Cholesterol check within past five years. The same features were selected from Xgboost was the above mentioned without the 'CholCheck' and additionally: 'Diabetes' was (Ever told) the presence of diabetes,'age' and 'NoDocbcCost' was there a time in the past 12 months when the participant needed to see a doctor but could not because of cost.



	HighBP	HighChol	CholCheck	BMI	Smoker	Stroke	Diabetes	PhysActivity	Fruits	Veggies	...	AnyHealthcare	NoDocbcCost	GenHlth	MentHlth	Ph
0	1.0	1.0	1.0	40.0	1.0	0.0	0.0	0.0	0.0	1.0	...	1.0	0.0	5.0	18.0	
1	0.0	0.0	0.0	25.0	1.0	0.0	0.0	1.0	0.0	0.0	...	0.0	1.0	3.0	0.0	
2	1.0	1.0	1.0	28.0	0.0	0.0	0.0	0.0	1.0	0.0	...	1.0	1.0	5.0	30.0	
3	1.0	0.0	1.0	27.0	0.0	0.0	0.0	1.0	1.0	1.0	...	1.0	0.0	2.0	0.0	
4	1.0	1.0	1.0	24.0	0.0	0.0	0.0	1.0	1.0	1.0	...	1.0	0.0	2.0	3.0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
253675	1.0	1.0	1.0	45.0	0.0	0.0	0.0	0.0	1.0	1.0	...	1.0	0.0	3.0	0.0	
253676	1.0	1.0	1.0	18.0	0.0	0.0	2.0	0.0	0.0	0.0	...	1.0	0.0	4.0	0.0	
253677	0.0	0.0	1.0	28.0	0.0	0.0	0.0	1.0	1.0	0.0	...	1.0	0.0	1.0	0.0	
253678	1.0	0.0	1.0	23.0	0.0	0.0	0.0	0.0	1.0	1.0	...	1.0	0.0	3.0	0.0	
253679	1.0	1.0	1.0	25.0	0.0	0.0	2.0	1.0	1.0	0.0	...	1.0	0.0	2.0	0.0	

253680 rows × 21 columns

Figure 47: Dataset project 2.

Income	HighBP	HighChol
CholCheck	BMI	Smoker
Stroke	Diabetes	PhysActivity
Fruits	Veggies	HvyAlcoholConsump
AnyHealthcare	NoDocbcCost	GenHlth
MentHlth	PhysHlth	DiffWalk
Sex	Age	Education

Table 35: Dataset features.

	BMI	**GenHlth**	**MentHlth**	**PhysHlth**	**Age**	**Education**	**Income**
count	253680	253680	253680	253680	253680	253680	253680
mean	28.382364	2.511392	3.184772	4.242081	8.032119	5.050434	6.053875
std	6.608694	1.068477	7.412847	8.717951	3.054220	0.985774	2.071148
min	12	1	0	0	1	1	1
25%	24	2	0	0	6	4	5
50%	27	2	0	0	8	5	7
75%	31	3	2	3	10	6	8
max	98	5	30	30	13	6	8

Table 36: Descriptive statistics.

Model	F1
Balanced Bagging	0.29
Xgboost	**0.414**
Random Forest	0.393
Adaboost (SVM)	0.4
Logistic Regression	0.41
Adaboost(DT)	0.413
Adaboost(LR)	**0.415**

Table 37: Models comparison.

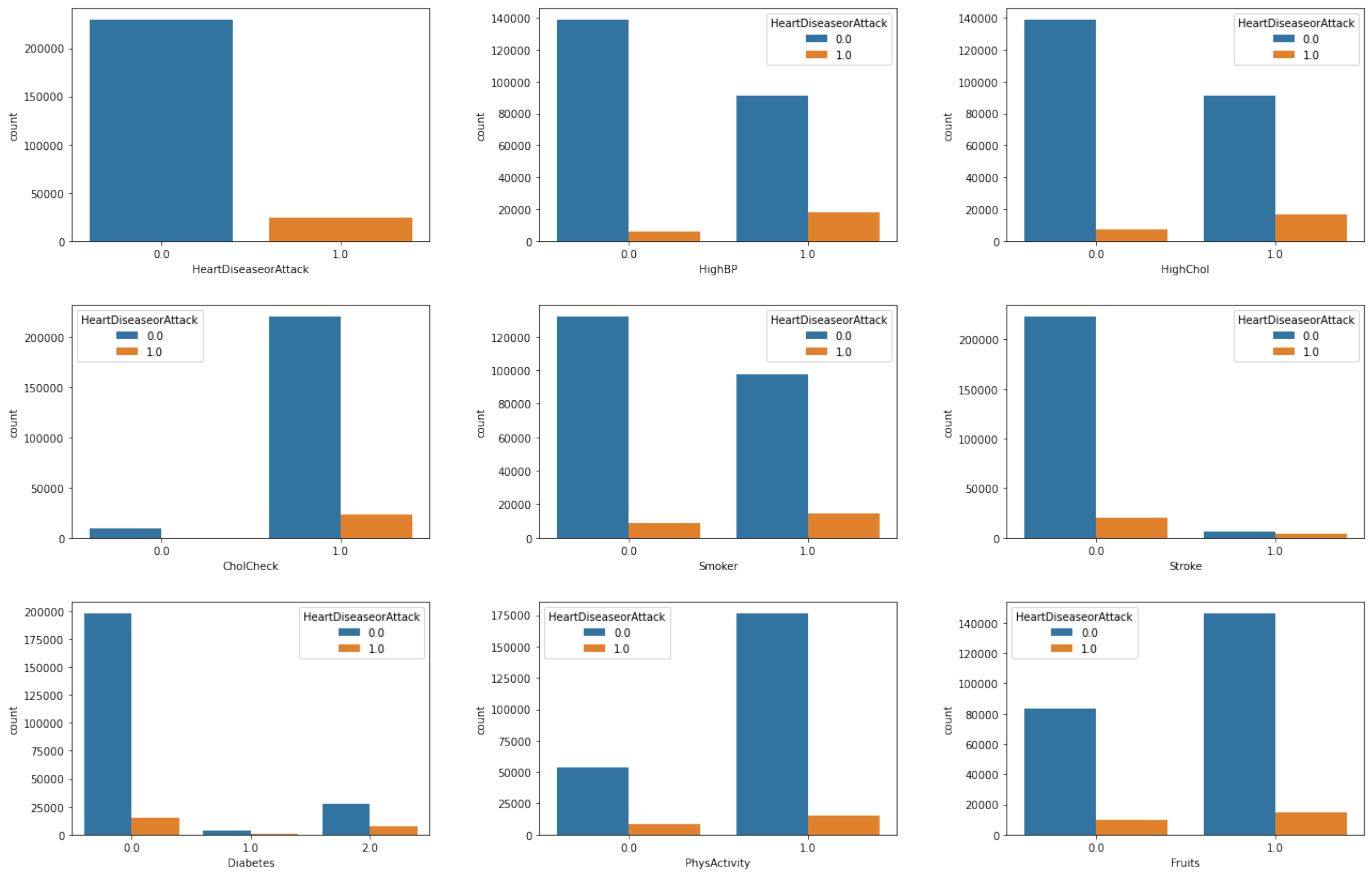


Figure 48: Plots for categorical variables.

Model	No Resampling		Default Parameters		Full optimized model		Using feature selection			
	Accuracy	F1	Accuracy	F1	Accuracy	F1	Accuracy	F1	Sens	Spec
Adaboost	0.9	0.09	0.8	0.377	0.85	0.418	0.84	0.417	59.82%	86.83%
Xgboost	0.91	0.19	0.83	0.405	0.83	0.41	0.83	0.42	62.77%	87.84%

Table 38: Evaluation process.

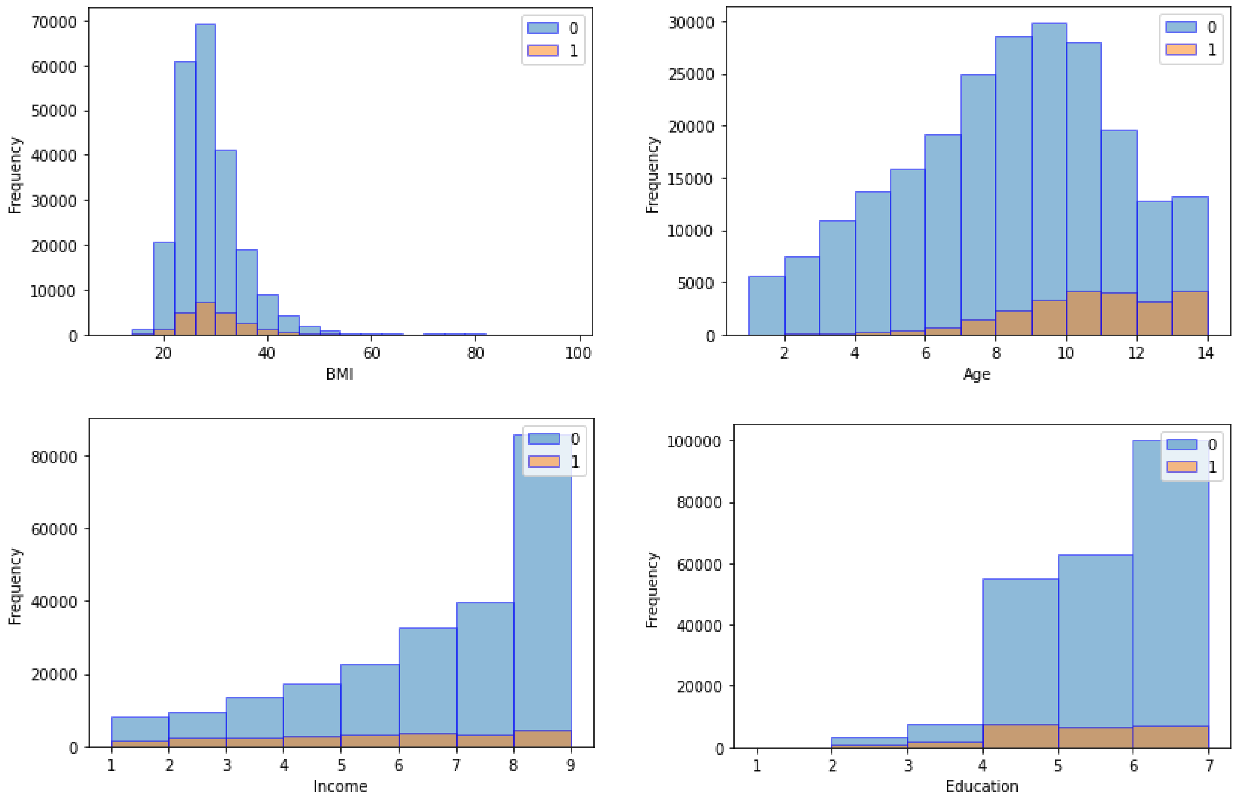


Figure 49: Plots for numerical variables.

Adaboost	Xgboost
HighBP	HighBP
HighChol	HighChol
CholCheck	Smoker
Smoker	Stroke
Stroke	Diabetes
HvyAlcoholConsump	HvyAlcoholConsump
GenHlth	NoDocbcCost
DiffWalk	GenHlth
Sex	DiffWalk
	Sex
	Age

Table 39: Selected Features.

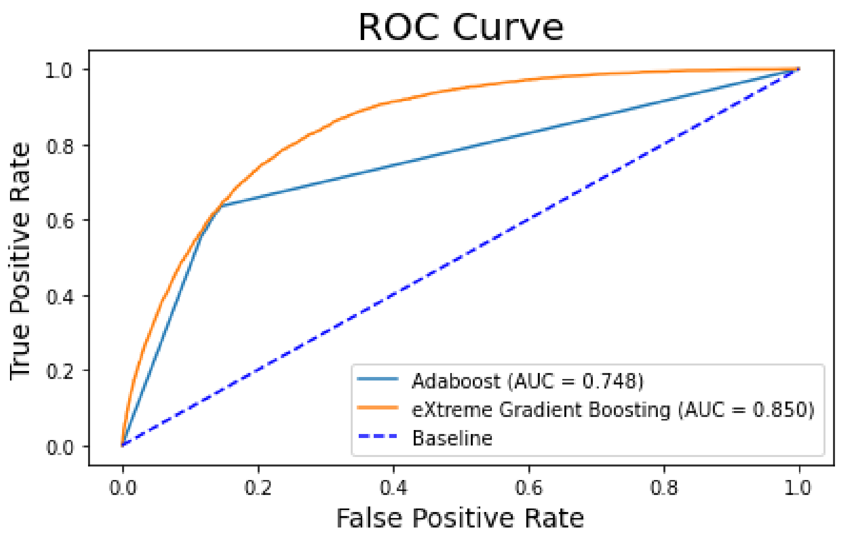


Figure 50: ROC Curves for the 2 optimal models.

10.5 Project 3

The dataset used for the third project is from the published research [147] and contains 48 variables collected by 4313 patients. The whole data is represented in figure 51. The goal was to create an algorithm for mortality prediction of patients with COVID-19.

Some descriptive statistics are presented in table 41. The target variable was binary named 'death' with classes $0 = no$ and $1 = yes$. The distribution of the target variable is presented in the first graph in figure 52 and it seems that the data are imbalanced. The two classes had a ratio frequency 1:3 so the data was considered slightly imbalanced. In the next graphs from the same figure are presented the distributions of some categorical variables according to the target variable. Moreover, in figure 53 are presented histograms for some variables according to the target variable.

Many columns and rows had missing values. The variables and the rows which had more than 70% and 80% missing values were deleted. As a consequence, 253 data points and 9 features were eliminated. The target variable was separated from the dataset to a variable named y and the remaining to a data frame named X. Then, they were separated into train 80% and test 20% data and using the command stratify=y the train and test set had a ratio equal to the target variable.

Missing values contained only the numerical features and they were replaced by the median of each variable using SimpleImputer. One hot encoding and Standardization were applied only before Support Vector Machine and Logistic Regression. Then, feature selection using RFE was applied. The next and final pre-processing step before fitting the algorithm was to undersample the data using RandomUnderSampler in python. For model evaluation and hyperparameter tuning Stratified 5-fold cross validation was used based on f1-measure. All the pre-processing steps and classifiers were included in pipeline in the order mentioned above to avoid data leakage.

In the model selection process Random Forest, XGBoost, Adaboost with base classifier Decision Tree, Support Vector Machine and Logistic Regression were performed and evaluated having their default hyperparameters. In this process, all the pre-processing steps were applied except for feature selection. Random Forest and XGboost had the best results and they were selected for further optimization.

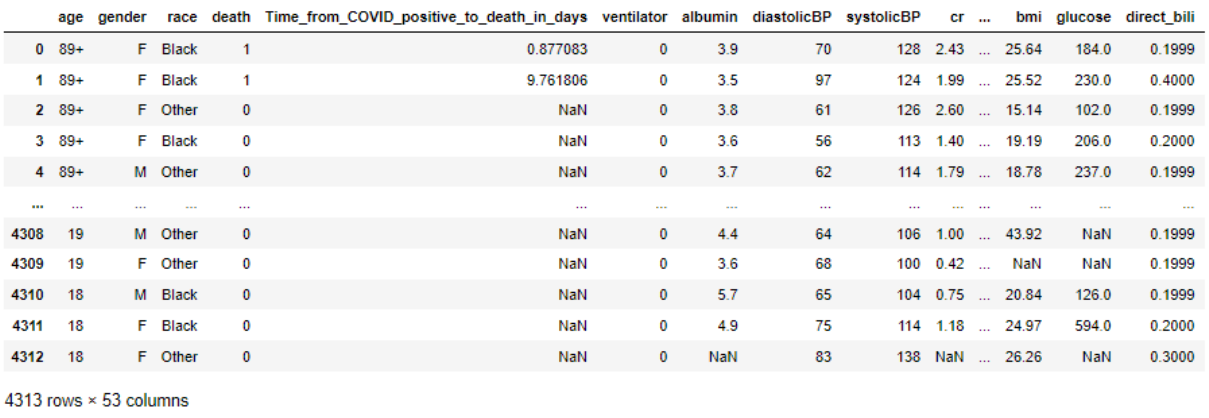
The next step was to select the best estimator for ranking the features using Recursive feature elimination. Decision Tree was chosen between Perceptron, Logistic Regression and Random Forest. Random Forest produced a slightly better result, but was rejected since it is more time consuming, and the benefit compared to Decision Tree was not important. Machine learning is a trade-off process between time and more accurate results and sometimes is preferable to sacrifice a little efficiency to speed up the process.

Next, hyperparameter tuning using Bayesian optimization for Random Forest and XGBoost was applied. The models were fitted on the training set and then applied to the test set. Having default hyperparamaters F1 score for Random Forest and Xgboost was 0.75 and 0.727 respectively. The results of the two models are presented in table 43.

The optimized parameters for the Random Forest model using all the features was bootstrap=True, max_depth=12, max_features='auto', n_estimators=150,min_samples_leaf=1, min_samples_split=2, random_state=42 C=100, learning_rate=0.6, n_estimators=400 and achieved F1 equal to 0.75 and accuracy 0.86. Using feature selection, 15 features (table 44) were selected to be the most important. The best parameters for the Random Forest model using those 15 features was bootstrap=True, max_depth=10, max_features='sqrt',n_estimators=450, min_samples_leaf=3, min_samples_split=2, n_estimators=200 and achieved F1 equal to 0.757 and accuracy was 0.87. Also, for the same model the ROC curve is presented in figure 54 and the AUC was 0.916.

The optimized parameters for the XGBoost model using all the features was use_label_encoder=False,n_jobs = 1,objective = 'binary:logistic',eval_metric = 'auc', colsample_bylevel=1, colsample_bytree=0.7, gamma=0.0, learning_rate=0.05, max_depth=6, min_child_weight=1, n_estimators=300,subsample=1, silent=1, tree_method='approx' and achieved F1 equal to 0.74 and accuracy 0.87. Using feature selection, 15 features (table 44) were selected to be the most important. The best parameters for the XGBoost model using those 15 features was use_label_encoder=False, n_jobs = 1,objective = 'binary:logistic', eval_metric = 'auc', colsample_bylevel=0.01, colsample_bytree=0.7, gamma=0.4, learning_rate=0.05, max_depth=3, min_child_weight=1, n_estimators=100, subsample=0.7173804504207143, silent=1, tree_method='approx' and achieved F1 equal to 0.73 and accuracy was 0.85. Also, for the same model the ROC curve is presented in figure 54 and the AUC was 0.917.

The features which selected from both models are presented in table 44. Using Random Forest were selected the following: 'Age','race', 'ventilator', 'diastolicBP' was the diastolic blood pressure, 'systolicBP' was systolic blood pressure and 'cr' was creatinine,' Pulse0x' was pulse oximetry level, 'NLratio' , was neutrophil-lymphocyte ratio , 'Temperature','Bun' was blood urea nitrogen,'Troponin' was troponin level,'Ptt' was partial thromboplastin time,'Bmi' was body mass index, 'Creatine kinase' was creatine phosphokinase and 'potassium' . The same features were selected from both models was 'ventilator', 'diastolicBP', 'systolicBP', 'cr', ' Pulse0x', 'NLratio', 'Troponin', 'Bmi', 'Creatine kinase' and additionally using XGBoost : 'Hgb' was haemoglobin, 'lymphocyte', 'Monocyte' ,'protein' and 'pulse'.



	age	gender	race	death	Time_from_COVID_positive_to_death_in_days	ventilator	albumin	diastolicBP	systolicBP	cr	...	bmi	glucose	direct_bili
0	89+	F	Black	1	0.877083	0	3.9	70	128	2.43	...	25.64	184.0	0.1999
1	89+	F	Black	1	9.761806	0	3.5	97	124	1.99	...	25.52	230.0	0.4000
2	89+	F	Other	0	NaN	0	3.8	61	126	2.60	...	15.14	102.0	0.1999
3	89+	F	Black	0	NaN	0	3.6	56	113	1.40	...	19.19	206.0	0.2000
4	89+	M	Other	0	NaN	0	3.7	62	114	1.79	...	18.78	237.0	0.1999
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
4308	19	M	Other	0	NaN	0	4.4	64	106	1.00	...	43.92	NaN	0.1999
4309	19	F	Other	0	NaN	0	3.6	68	100	0.42	...	NaN	NaN	0.1999
4310	18	M	Black	0	NaN	0	5.7	65	104	0.75	...	20.84	126.0	0.1999
4311	18	F	Black	0	NaN	0	4.9	75	114	1.18	...	24.97	594.0	0.2000
4312	18	F	Other	0	NaN	0	NaN	83	138	NaN	...	26.26	NaN	0.3000

4313 rows × 53 columns

Figure 51: Dataset project 3.

age	gender	death	glucose
race	ct value	potassium	pro bnp
ventilator	egfr	COVID to death (days)	indirect bili
albumin	diastolicBP	cr	direct bili
systolicBP	charlson score	ddimer	bmi
eosinophil	ferritin	hgb	rdw
inr	lymphocyte	NLratio	troponin
neutrophil	fibrinogen	platelet	wbc
pulse	pulseOx	protein	monocyte
rr	temperature	alt	total bili
ast	bun	chloride	ptt
crp	interleukin6	calcium	tnf
ldh	mcv	mpv	procalcitonin
creatine kinase			

Table 40: Dataset features.

	age	diastolicBP	systolicBP	cr	NLratio	pulseOx	temperature	bun	troponin	ptt	bmi	creatine_kinase	potassium
count	4060	4060	4060	4014	4060	4059	4058	4014	3549	3252	3834	3299	3984
mean	64.386	67.506	120.091	2.041	7.967	92.503	99.217	32.930	0.058	35.4194	30.448	621.985	4.427
std	16.317	16.718	24.840	2.585	8.892	8.478	1.694	32.782	0.264	14.719	49.796	2804.497	0.741
min	18	0	0	0.19	0	11	85.3	4.999	0.0099	19	9.9	19.999	1.9999
25%	55	59	107	0.81	3.461	90	98.2	13	0.01	29.6	24.622	86	4
50%	66	69	122	1.13	5.777	95	98.9	20.5	0.01	32.8	28.42	168	4.4
75%	77	79	136	1.96	9.603846	98	100	40	0.03	37	33.19	416	4.8
max	91	120	215	31.66	208.5	100	122	301	9.56	200	3069.26	80000	9.0001

Table 41: Descriptive statistics.

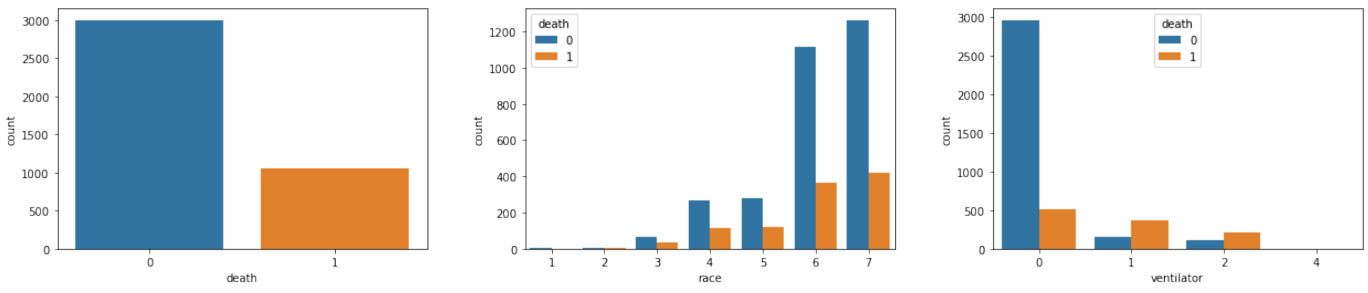


Figure 52: Plots for categorical variables.

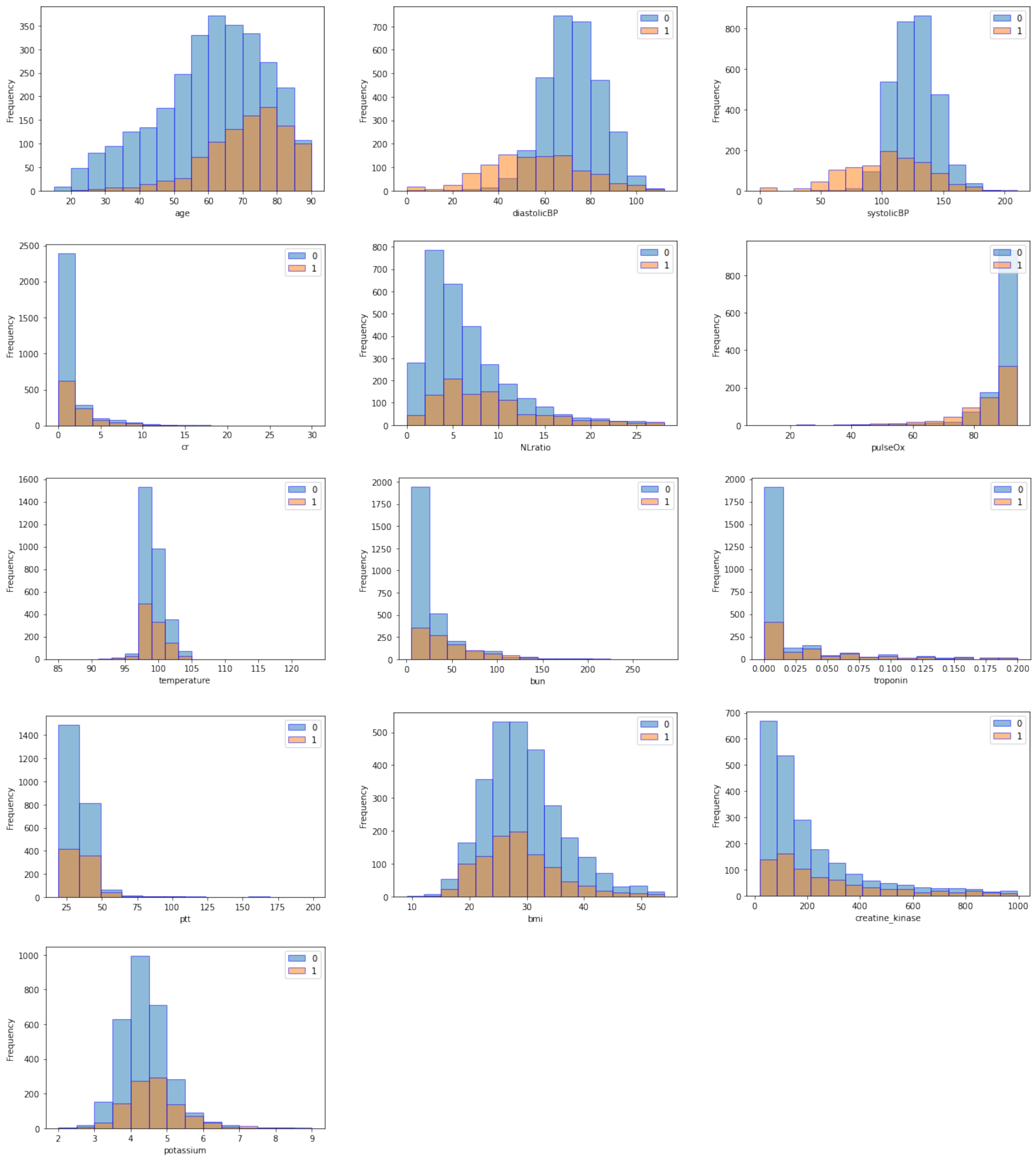


Figure 53: Plots for numerical variables.

Model	F1
Random Forest	**0.723**
Decision Tree	0.694
Xgboost	**0.716**
SVM	0.634
Adaboost	0.699
Voting	0.642

Table 42: Models comparison.

Model	Default Parameters		Full optimized Model		Using feature selection			
	Accuracy	F1	Accuracy	F1	Accuracy	F1	Sens	Spec
Random Forest	0.87	0.75	0.86	0.75	0.87	0.757	77.72%	90.34%
Xgboost	0.86	0.727	0.87	0.74	0.85	0.73	77.77%	87.52%

Table 43: Evaluation process.

Random Forest	Xgboost
Age	Age
race	ventilator
ventilator	diastolicBP
diastolicBP	systolicBP
systolicBP	cr
cr	Hgb
NLratio	lymphocyte
Pulse0x	NLratio
Temperature	Protein
Bun	Pulse
Troponin	Pulse0x
Ptt	Monocyte
Bmi	Troponin
Creatine_kinase	Bmi
potassium	Creatine_kinase

Table 44: Selected features.

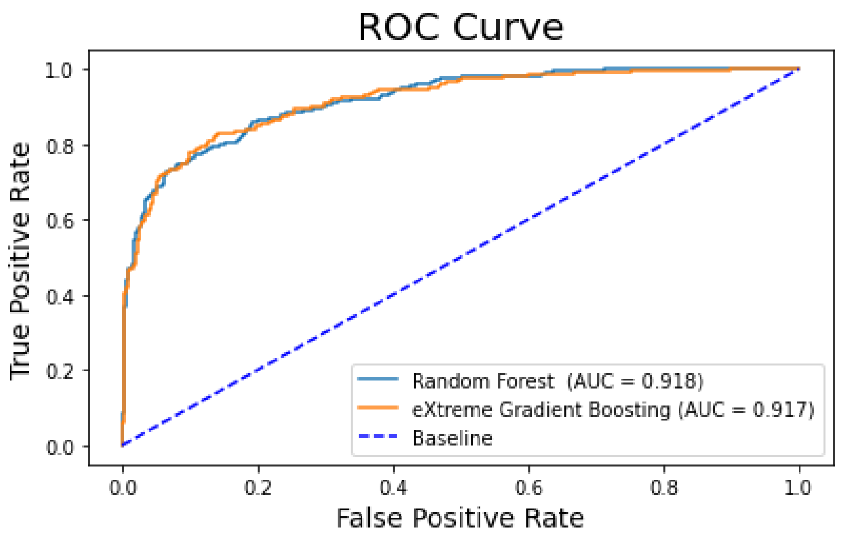


Figure 54: ROC Curves for the 2 optimal models.

10.6 Summary

After the above analyzes is concluded that even a powerful machine learning algorithm its self does not ensure good results. Data preprocessing, feature selection or hyperparameters optimization are crucial steps in training process and can improve importantly the predictive ability of the model. However, the choice of the most suitable algorithm undoubtedly can play a vital role in the model's efficiency. The results in the above three applications are considered satisfying based on the evaluation metrics. They indicate the potentials of machine learning algorithms to discover new biomarkers for various diseases utilizing real world data.

A Code for Project 1

```
import pandas as pd
from sklearn.feature_selection import RFE
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.model_selection import cross_val_score ,KFold
from sklearn.naive_bayes import GaussianNB
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, precision_score, recall_score
from sklearn.ensemble import RandomForestClassifier
from imblearn.pipeline import Pipeline
from skopt import BayesSearchCV
from sklearn.compose import make_column_transformer
from numpy import mean
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from numpy import std
from sklearn.compose import ColumnTransformer
import xgboost as xgb
from sklearn.ensemble import AdaBoostClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import VotingClassifier
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.stats import skew, kurtosis
df = pd.read_csv('heart.csv')
df
df.rename(columns={'target':'heart attack'},inplace=True)
X = df.drop(columns = ['heart attack'])
y = df['heart attack']
X.describe()
skew(df.age)
cols=['chol','age','trestbps','thalach','oldpeak']
for col in df[cols]:

  print("Skew: %s %.3f" % (col,skew(df[col])))
  print('>%s %.3f (%.3f)' % (name, mean(scores), std(scores)))
  print (skew(col))

for col in df.columns:

      print("kurtosis: %s %.3f" % (col,kurtosis(df[col])))
```

```
sns.countplot(x='heart attack',data=df)
sns.countplot(x='thal',hue='heart attack',data=df)
sns.countplot(x='cp',hue='heart attack',data=df)
sns.countplot(x='exang',hue='heart attack',data=df)
sns.countplot(x='restecg',hue='heart attack',data=df)
sns.countplot(x='sex',hue='heart attack',data=df)
sns.countplot(x='ca',hue='heart attack',data=df)
sns.countplot(x='fbs',hue='heart attack',data=df)
sns.countplot(x='slope',hue='heart attack',data=df)

myheart = df['heart attack']
mychol = df['chol']

heart_0 = myheart == 0
heart_1 = myheart == 1

plt.hist(mychol[heart_0], edgecolor='blue', alpha=0.5,
         bins =range(126,564,44) , label='0')
plt.hist(mychol[heart_1], edgecolor='blue', alpha=0.5,
         bins = range(126,564,44) , label='1')

plt.legend(loc='upper right')
plt.xlabel('chol')
plt.ylabel('Frequency')
plt.show()

myage = df['age']

plt.hist(myage[heart_0], edgecolor='blue', alpha=0.5,
         bins = range(25,80,6), label='0')
plt.hist(myage[heart_1], edgecolor='blue', alpha=0.5,
         bins = range(25,80,6), label='1')
plt.legend(loc='upper right')
plt.xlabel('age')
plt.ylabel('Frequency')
plt.show()

cols=['chol','age','trestbps','thalach','oldpeak']

mytrestbps = df['trestbps']

plt.hist(mytrestbps[heart_0], edgecolor='blue', alpha=0.5,
         bins =  range(90,200,11), label='0')
plt.hist(mytrestbps[heart_1], edgecolor='blue', alpha=0.5,
         bins = range(90,200,11), label='1')
plt.legend(loc='upper right')
plt.xlabel('trestbps')
plt.ylabel('Frequency')
plt.show()

mythalach = df['thalach']

plt.hist(mythalach[heart_0], edgecolor='blue', alpha=0.5,
         bins = range(71,205,14), label='0')
plt.hist(mythalach[heart_1], edgecolor='blue', alpha=0.5,
         bins =  range(71,205,14), label='1')
plt.legend(loc='upper right')
plt.xlabel('thalach')
plt.ylabel('Frequency')
plt.show()

myoldpeak = df['oldpeak']

plt.hist(myoldpeak[heart_0], edgecolor='blue', alpha=0.5,
         bins = [0,0.6,1.2,1.8,2.4,3,3.6,4.2,4.8,5.4,6], label='0')

plt.hist(myoldpeak[heart_1], edgecolor='blue', alpha=0.5,
         bins =  [0,0.6,1.2,1.8,2.4,3,3.6,4.2,4.8,5.4,6], label='1')

plt.legend(loc='upper right')
plt.xlabel('oldpeak')
plt.ylabel('Frequency')
plt.show()
dataset=df.groupby('heart attack')[cols].mean()

#dataset=df1.groupby('target').mean()

#Differentiate Discrete and Continuous features
#Print Discrete Feature Data
discrete_feature=[feature for feature in feature_list if
                  len(df[feature].unique())<25]
print("Discrete Variables Count: {}".format(len(discrete_feature)))
print("Discrete features are ",discrete_feature)
#Print Continuous Feature Data
cont_feature=[feature for feature in feature_list if
              len(df[feature].unique())>25]
print("Continuous Variables Count: {}".format(len(cont_feature)))
print("Continuous features are ",cont_feature)

indx=np.arange(len(cols))
score_label=np.arange(0,270,20)
```

```
col1=list(dataset.T[1])
col2=list(dataset.T[0])
bar_width=0.35

fig,ax=plt.subplots()
bar_1=ax.bar(indx-bar_width/2, col1, bar_width,
             label='less chance of heart attack')
bar_2=ax.bar(indx+bar_width/2, col2, bar_width,
             label='more chance of heart attack ')

#inserting x axis label
ax.set_xticks(indx)
ax.set_xticklabels(cols)

ax.legend()

#inserting y axis label
ax.set_yticks(score_label)
ax.set_yticklabels(score_label)

def insert_data_labels(bars):
    for bar in bars:
        bar_height = bar.get_height()
        ax.annotate('{0:.0f}'.format(bar.get_height()),
            xy=(bar.get_x() + bar.get_width() / 2, bar_height),
            xytext=(0, 3),
            textcoords='offset points',
            ha='center',
            va='bottom'
        )

insert_data_labels(bar_1)
insert_data_labels(bar_2)

plt.show()

def encoding(X):
    column_transformer = make_column_transformer( (OneHotEncoder(),
        ["slope","ca","thal","cp","restecg"]),remainder='passthrough')
    X = column_transformer.fit_transform(X)
    X = pd.DataFrame(data=X, columns=
                    column_transformer.get_feature_names_out())
    return X

X=encoding(X)
X_train, X_test, y_train, y_test = train_test_split( X, y, test_size=0.2
                                        ,stratify=y, random_state=42)

numerical_ix=['remainder__thalach','remainder__age',
      'remainder__trestbps', 'remainder__chol','remainder__oldpeak']
preprocessor_ = ColumnTransformer(transformers =
   [('scale',StandardScaler(),numerical_ix)],remainder ='passthrough')


def get_models():
    models = dict()
    models['Random forest'] = RandomForestClassifier()
    models['DecisionTree'] = DecisionTreeClassifier()
    models['xgboost'] = xgb.XGBClassifier(random_state=42)
    models['svm'] = Pipeline(steps=[('preprocessor_',preprocessor_),
    ('m',SVC())])
    models['log_reg'] = Pipeline(steps=
            [('preprocessor_',preprocessor_),('m',LogisticRegression())])
    models['GaussianNB'] = Pipeline(steps=
            [('preprocessor_',preprocessor_),('m', GaussianNB())])
    models['Adaboost'] = AdaBoostClassifier(random_state=42)

    lr= LogisticRegression()
    dtc = DecisionTreeClassifier()
    kn=KNeighborsClassifier(n_neighbors=1)
    base_methods=[('pipe_lr', lr),('DecisionTree',dtc),('pipe_Kn',kn)]
    vote_model=VotingClassifier(estimators=base_methods,voting='hard')
    models['Voting'] = Pipeline(steps=
             [('preprocessor_',preprocessor_),('m',vote_model)])
    return models

def evaluate_model(model, X, y):
    cv = KFold(n_splits=10,shuffle=True,random_state=40)
    scores = cross_val_score(model, X, y, scoring='accuracy',
                cv=cv, n_jobs=-1, error_score='raise')
    return scores



models = get_models()
results, names = list(), list()
for name, model in models.items():
    scores = evaluate_model(model, X_train, y_train)
    results.append(scores)
    names.append(name)
    print('>%s %.3f (%.3f)' % (name, mean(scores), std(scores)))

def get_models():
    models = dict()
    for i in range(5,24):
        rfe = RFE(estimator=DecisionTreeClassifier()
                    , n_features_to_select=i)
        model = LogisticRegression()
        models[str(i)] = Pipeline(
      steps=[("preprocessor_", preprocessor_),('s',rfe),('m',model)])
    return models

def evaluate_model(model, X, y):
    cv = KFold(n_splits=10,shuffle=True,random_state=40)
    scores = cross_val_score(model, X, y, scoring='accuracy',
                cv=cv, n_jobs=-1, error_score='raise')
    return scores

models = get_models()
results, names = list(), list()
for name, model in models.items():
    scores = evaluate_model(model, X_train, y_train)
    results.append(scores)
    names.append(name)
    print('>%s %.3f (%.3f)' % (name, mean(scores), std(scores)))

model=SVC()
import xgboost as xgb
from sklearn.linear_model import Perceptron

def get_models():
    models = dict()
    rfe = RFE(estimator=
    LogisticRegression(max_iter=400), n_features_to_select=7)
    models['lr'] = Pipeline(steps=
    [('preprocessor_',preprocessor_),('rfe',rfe),('m',model)])

    rfe = RFE(estimator=Perceptron(), n_features_to_select=7)
    models['per'] = Pipeline(steps=
    [('preprocessor_',preprocessor_),('rfe',rfe),('m',model)])

    rfe = RFE(estimator=DecisionTreeClassifier(), n_features_to_select=7)
    models['dtc'] = Pipeline(steps=
    [('preprocessor_',preprocessor_),('rfe',rfe),('m',model)])

    rfe = RFE(estimator=RandomForestClassifier(), n_features_to_select=7)
    models['rf'] = Pipeline(steps=
    [('preprocessor_',preprocessor_),('rfe',rfe),('m',model)])
    rfe = RFE(estimator=xgb.XGBClassifier(), n_features_to_select=7)
    models['xgb'] = Pipeline(steps=
    [('preprocessor_',preprocessor_),('rfe',rfe),('m',model)])

    return models

def evaluate_model(model, X, y):
    cv = KFold(n_splits=10,shuffle=True)
    scores = cross_val_score(model, X, y,
                scoring='accuracy', cv=cv, n_jobs=-1)
    return scores

models = get_models()
results, names = list(), list()
for name, model in models.items():
    scores = evaluate_model(model, X_train, y_train)
    results.append(scores)
    names.append(name)
    print('>%s %.3f (%.3f)' % (name, mean(scores), std(scores)))

solver=['newton-cg', 'lbfgs', 'liblinear']

random_grid = {#'rfe__estimator__C':[0.001,0.01,0.1,1,10,100,1000],
#'rfe__estimator__solver':solver, 'rfe__n_features_to_select':[4,5,6,7,8]
 'model__C': [0.01,0.1, 1, 10, 100, 1000],
'model__gamma': [1, 0.1, 0.01, 0.001, 0.0001],'model__kernel': ['rbf']}

rfe = RFE(estimator=LogisticRegression(max_iter=400))
model = SVC()

model_pipeline = Pipeline(steps =[("preprocessor_", preprocessor_),
                                  #        ('rfe',rfe),
                                  ('model', model)])
cv = KFold(n_splits=10,shuffle=True)
opt = BayesSearchCV( model_pipeline,[(random_grid, 20)],cv=cv,scoring='ac
opt.fit(X_train, y_train)

print(opt.best_params_)
print(opt.best_score_)

solver=['newton-cg', 'lbfgs', 'liblinear']
n_estimators=[150,200,250,300,450]
max_features = ['auto', 'sqrt']
min_samples_split = [2,5, 10]
min_samples_leaf = [1, 2,3, 4]
bootstrap = [True, False]
```

```
random_grid = {#'rfe__estimator__C':[0.001,0.01,0.1,1,10,100,1000],
#'rfe__estimator__solver':solver,'rfe__n_features_to_select':[5,6,7,8,9],
'model__solver':solver,'model__C':[0.001,0.01,0.1,1,10,100,1000]}

rfe = RFE(estimator=LogisticRegression(max_iter=400))
model = LogisticRegression(max_iter=400)

model_pipeline = Pipeline(steps =[("preprocessor_", preprocessor_),
                    ('rfe',rfe),('model', model)])
cv = KFold(n_splits=10,shuffle=True)
opt = BayesSearchCV( model_pipeline,[(random_grid, 30)],cv=cv,scoring='ac
opt.fit(X_train, y_train)

print(opt.best_params_)
print(opt.best_score_)


#rfe = RFE(estimator=LogisticRegression(C=0.01,
#solver='liblinear',max_iter=400), n_features_to_select=8) #SVC
rfe = RFE(estimator=LogisticRegression(C=0.01,
solver='newton-cg',max_iter=400), n_features_to_select=7) #LR

#model = LogisticRegression(C=0.01,solver='liblinear',max_iter=400)#witho
#model = LogisticRegression()#without rfe
model = LogisticRegression(C=0.1,solver='lbfgs',max_iter=400)#with rfe

#model=SVC()
#model = SVC(C=10,gamma=0.001)#without rfe
#model= SVC(C=10,gamma=0.001)#with rfe

pipeline_model = Pipeline(steps=[ ("preprocessor_", preprocessor_),
          ('rfe',rfe),('model',model)])


pipeline_model.fit(X_train, y_train)

y_pred_train=pipeline_model.predict(X_train)

conf_pred_train=confusion_matrix(y_train,y_pred_train)
print(conf_pred_train)

score_train=accuracy_score(y_train,y_pred_train)
print(" Accuracy on training set are {}".format(score_train))
y_pred_test = pipeline_model.predict(X_test)

conf_pred_test = confusion_matrix(y_test, y_pred_test)
```

```
print(conf_pred_test)

score=accuracy_score(y_test,y_pred_test)
print(" Accuracy on test set are {}".format(score))
print(classification_report(y_test, y_pred_test))

recall_sensitivity = recall_score(y_test, y_pred_test, pos_label=1)

recall_specificity = recall_score(y_test, y_pred_test, pos_label=0)

recall_sensitivity, recall_specificity

rfe_svc = RFE(estimator=LogisticRegression(
  C=0.01,solver='liblinear',max_iter=400), n_features_to_select=8) #SVC
rfe_lr = RFE(estimator=LogisticRegression(
  C=0.01,solver='newton-cg',max_iter=400), n_features_to_select=7) #LR

model_svc= SVC(C=10,gamma=0.001,probability=True)#with rfe
pipeline_model_svc = Pipeline(steps=[ ("preprocessor_", preprocessor_)
                  ,('rfe_svc',rfe_svc),('model_svc',model_svc)])

model_lr = LogisticRegression(C=0.1,solver='lbfgs',max_iter=400)#with rfe
pipeline_model_lr = Pipeline(steps=[ ("preprocessor_", preprocessor_)
                  ,('rfe_lr',rfe_lr),('model_lr',model_lr)])

model_lr=pipeline_model_lr.fit(X_train, y_train)
probs_lr = model_lr.predict_proba(X_test)[:, 1]

model_svc =  pipeline_model_svc.fit(X_train, y_train)
probs_svc = model_svc.predict_proba(X_test)[:, 1]

from sklearn.metrics import roc_auc_score, roc_curve

auc_lr = roc_auc_score(y_test, probs_lr)
fpr_lr, tpr_lr, thresholds_lr = roc_curve(y_test, probs_lr)

auc_svc = roc_auc_score(y_test, probs_svc)
fpr_svc, tpr_svc, thresholds_svc = roc_curve(y_test, probs_svc)

plt.figure(figsize=(7, 4))

plt.plot(fpr_lr, tpr_lr, label=f'Logistic Regression (AUC = {auc_lr:.3f})
plt.plot(fpr_svc, tpr_svc, label=
f'Support Vector Machine (AUC = {auc_svc:.3f})')

plt.plot([0, 1], [0, 1], color='blue', linestyle='--', label='Baseline')
plt.title('ROC Curve', size=20)
plt.xlabel('False Positive Rate', size=14)
plt.ylabel('True Positive Rate', size=14)
plt.legend();

rfe = RFE(estimator=LogisticRegression(C=0.01,
    solver='liblinear',max_iter=400), n_features_to_select=8) #SVC
#rfe = RFE(estimator=LogisticRegression(C=0.01,
#solver='newton-cg',max_iter=400), n_features_to_select=7) #LR

fit = rfe.fit(X_train, y_train)
print("Num Features: %d" % fit.n_features_)
print("Selected Features: %s" % fit.support_)
print("Feature Ranking: %s" % fit.ranking_)
print(X_train.columns[rfe.support_])

\section{Code for Project 1}\\
\begin{lstlisting}[xleftmargin=0em]
```

B Code for Project 2

```
import pandas as pd
from sklearn.preprocessing import OneHotEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_score,KFold

import numpy as np
from numpy import std
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, precision_score,
recall_score, f1_score,classification_report

from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from imblearn.pipeline import Pipeline
from skopt import BayesSearchCV
from sklearn.compose import make_column_transformer

from numpy import mean
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn.compose import ColumnTransformer
import xgboost as xgb
from sklearn.ensemble import AdaBoostClassifier
from sklearn.svm import LinearSVC
from sklearn.model_selection import StratifiedKFold
from imblearn.under_sampling import RandomUnderSampler
from sklearn.feature_selection import RFE
from sklearn.tree import DecisionTreeClassifier
from imblearn.ensemble import BalancedBaggingClassifier
from imblearn.ensemble import BalancedRandomForestClassifier
from sklearn.linear_model import Perceptron
from skopt import BayesSearchCV
from numpy import arange
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.stats import skew, kurtosis

df = pd.read_csv('heart_2.csv')
y = df['HeartDiseaseorAttack']
X=df.drop(columns='HeartDiseaseorAttack')

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, stratify=y, random_state=42)
X[cols].describe()
cols=['BMI','GenHlth','MentHlth','PhysHlth','Age','Education','Income']
for col in df[cols]:
    print("Skew: %s %.3f" % (col,skew(df[col])))

for col in  df[cols]:
    print("kurtosis: %s %.3f" % (col,kurtosis(df[col])))

sns.countplot(x='HeartDiseaseorAttack',data=df)
sns.countplot(x='thal',hue='heart attack',data=df)
cols2=['HighBP','HighChol','CholCheck','Smoker',
    'Stroke','Diabetes','PhysActivity','Fruits','Veggies']

sns.countplot(x='HighBP',hue='HeartDiseaseorAttack',data=df)
sns.countplot(x='HighChol',hue='HeartDiseaseorAttack',data=df)
sns.countplot(x='CholCheck',hue='HeartDiseaseorAttack',data=df)
sns.countplot(x='Smoker',hue='HeartDiseaseorAttack',data=df)
sns.countplot(x='Stroke',hue='HeartDiseaseorAttack',data=df)
sns.countplot(x='Diabetes',hue='HeartDiseaseorAttack',data=df)
sns.countplot(x='PhysActivity',hue='HeartDiseaseorAttack',data=df)
sns.countplot(x='Fruits',hue='HeartDiseaseorAttack',data=df)
sns.countplot(x='Veggies',hue='HeartDiseaseorAttack',data=df)

cols=['BMI','GenHlth','MentHlth','PhysHlth','Age','Education','Income']

myheart = df['HeartDiseaseorAttack']
myBMI = df['BMI']
```

```
heart_0 = myheart == 0
heart_1 = myheart == 1

plt.hist(myBMI[heart_0], edgecolor='blue', alpha=0.5,
         bins =range(10,100,4) , label='0')
plt.hist(myBMI[heart_1], edgecolor='blue', alpha=0.5,
         bins = range(10,100,4), label='1')
plt.legend(loc='upper right')
plt.xlabel('BMI')
plt.ylabel('Frequency')
plt.show()

myheart = df['HeartDiseaseorAttack']
myAge = df['Age']

plt.hist(myAge[heart_0], edgecolor='blue', alpha=0.5,
         bins = [1,2,3,4,5,6,7,8,9,10,11,12,13,14], label='0')
plt.hist(myAge[heart_1], edgecolor='blue', alpha=0.5,
         bins = [1,2,3,4,5,6,7,8,9,10,11,12,13,14], label='1')
plt.legend(loc='upper right')
plt.xlabel('Age')
plt.ylabel('Frequency')
plt.show()

myheart = df['HeartDiseaseorAttack']
myIncome = df['Income']

plt.hist(myIncome[heart_0], edgecolor='blue',
         alpha=0.5, bins = [1,2,3,4,5,6,7,8,9], label='0')
plt.hist(myIncome[heart_1], edgecolor='blue',
         alpha=0.5, bins =[1,2,3,4,5,6,7,8,9], label='1')
plt.legend(loc='upper right')
plt.xlabel('Income')
plt.ylabel('Frequency')
plt.show()

myheart = df['HeartDiseaseorAttack']
myIncome = df['Education']

plt.hist(myIncome[heart_0], edgecolor='blue', alpha=0.5,
         bins = [1,2,3,4,5,6,7], label='0')
plt.hist(myIncome[heart_1], edgecolor='blue', alpha=0.5,
         bins =[1,2,3,4,5,6,7], label='1')
plt.legend(loc='upper right')
plt.xlabel('Education')
plt.ylabel('Frequency')
plt.show()

cols=['HighBP','HighChol','Smoker','Stroke','CholCheck',
      'Fruits','Veggies','Diabetes']
dataset=df.groupby('HeartDiseaseorAttack')[cols].mean()
#dataset

indx=np.arange(len(cols))
score_label=np.arange(0,1.2,0.2)

col1=list(dataset.T[1])
col2=list(dataset.T[0])
bar_width=0.35

fig,ax=plt.subplots()
bar_1=ax.bar(indx-bar_width/2, col1, bar_width,
             label='more chance of heart attack')
bar_2=ax.bar(indx+bar_width/2, col2, bar_width,
             label='less chance of heart attack ')

#inserting x axis label
ax.set_xticks(indx)
ax.set_xticklabels(cols)

ax.legend()

#inserting y axis label
ax.set_yticks(score_label)
ax.set_yticklabels(score_label)

def insert_data_labels(bars):
    for bar in bars:
        bar_height = bar.get_height()
        ax.annotate('{0:.0f}'.format(bar.get_height()),
            xy=(bar.get_x() + bar.get_width() / 2, bar_height),
            xytext=(0, 3),
            textcoords='offset points',
            ha='center',
            va='bottom'
        )

insert_data_labels(bar_1)
insert_data_labels(bar_2)

plt.show()
```

```
preprocessor_ = ColumnTransformer(transformers =[
    ('scale',StandardScaler(), X_train.columns)], remainder ='passthrough
under = RandomUnderSampler(sampling_strategy =0.5,random_state =40)
def get_models():
    models = dict()
    model= BalancedBaggingClassifier()
    models['balanced']= Pipeline(steps =[('under', under),('m',model)])

    model= DecisionTreeClassifier()
    models['balanced']= Pipeline(steps =[('under', under),('m',model)])

    models['xgboost'] = Pipeline(steps =[('under',
        under),('m',xgb.XGBClassifier())])

    models['Random forest'] = Pipeline(steps =[('under', under),
                    ('m',RandomForestClassifier(random_state =42))])

    model=AdaBoostClassifier(base_estimator=LinearSVC(),algorithm='SAMME'
    models['Ada_SVC'] = Pipeline(steps =[('preprocessor_',preprocessor_),
                                ('under', under),('m',model)])

    models['log_reg'] = Pipeline(steps =[('preprocessor_',preprocessor_),
                        ('under', under),('m',LogisticRegression())])

    model = AdaBoostClassifier()
    models['Adaboost'] = Pipeline(steps =[('under', under),('m',model)])

    model = AdaBoostClassifier(base_estimator = LogisticRegression(
                                                max_iter =400))
    models['Adaboost_LR'] = Pipeline(steps=
     [('preprocessor_',preprocessor_),('under', under),('m',model)])
    return models

def evaluate_model(model, X, y):
    cv = StratifiedKFold(n_splits =5,shuffle=True )
    scores = cross_val_score(model, X, y, scoring='f1', cv=cv, n_jobs=-1)
    return scores

models = get_models()
results, names = list(), list()
for name, model in models.items():
    scores = evaluate_model(model, X_train, y_train)
    results.append(scores)
    names.append(name)
    print('>%s %.3f (%.3f)' % (name, mean(scores), std(scores)))
```

```
under = RandomUnderSampler(sampling_strategy=1,random_state=40)
preprocessor_ = ColumnTransformer(transformers =[
    ('scale',StandardScaler(), X_train.columns)], remainder ='passthrough

def get_models():
    models = dict()
    for i in range(5,21):
        rfe = RFE(estimator=LogisticRegression(max_iter=400),
                  n_features_to_select=i)
#          model = AdaBoostClassifier(base_estimator =
         #LogisticRegression(max_iter=400),algorithm='SAMME')
        model= xgb.XGBClassifier(use_label_encoder=False)
        models[str(i)] = Pipeline(steps=[("preprocessor_", preprocessor_)
                        ('s',rfe),('under', under),('m',model)])
    return models

def evaluate_model(model, X, y):
    cv = StratifiedKFold(n_splits=5,shuffle=True,random_state=42)
    scores = cross_val_score(model, X, y,
           scoring='f1', cv=cv, n_jobs=-1, error_score='raise')
    return scores

models = get_models()
results, names = list(), list()
for name, model in models.items():
    scores = evaluate_model(model, X_train, y_train)
    results.append(scores)
    names.append(name)
    print('>%s %.3f (%.3f)' % (name, mean(scores), std(scores)))

preprocessor_ = ColumnTransformer(transformers =[
    ('scale',StandardScaler(), X_train.columns)], remainder ='passthrough
under = RandomUnderSampler(sampling_strategy=0.5,random_state=40)
model=AdaBoostClassifier(base_estimator=
                    LogisticRegression(max_iter=400),algorithm='SAMME')
model= xgb.XGBClassifier(use_label_encoder=False)

def get_models():
    models = dict()
    rfe = RFE(estimator=LogisticRegression(max_iter=400),
                  n_features_to_select=9)
    models['lr'] = Pipeline(steps=[('preprocessor_',preprocessor_),
           ('rfe',rfe),('under', under),('m',model)])

    rfe = RFE(estimator=Perceptron(), n_features_to_select=9)
    models['per'] = Pipeline(steps=[('preprocessor_',preprocessor_),
                        ('rfe',rfe),('under', under),('m',model)])

    rfe = RFE(estimator=DecisionTreeClassifier(), n_features_to_select=9)
    models['dtc'] = Pipeline(steps=[('preprocessor_',preprocessor_),
                                ('rfe',rfe),('under', under),('m',model)]
    return models

def evaluate_model(model, X, y):
    cv = StratifiedKFold(n_splits=5,shuffle=True)
    scores = cross_val_score(model, X, y, scoring='f1', cv=cv, n_jobs=-1)
    return scores

models = get_models()
results, names = list(), list()
for name, model in models.items():
    scores = evaluate_model(model, X_train, y_train)
    results.append(scores)
    names.append(name)
    print('>%s %.3f (%.3f)' % (name, mean(scores), std(scores)))

solver=['newton-cg', 'lbfgs', 'liblinear']
random_grid = {'under__sampling_strategy':[0.5,0.8,1],
    'rfe__estimator__C':[0.001,0.01,0.1,1,10,100,1000],
'rfe__estimator__solver':solver,'rfe__n_features_to_select':
[6,7,8,9,10,11],
"model__learning_rate":[0.05,0.10,0.15,0.20,0.25,0.30],
"model__max_depth":[3,4,5,6,8,10,12,15],
    "model__min_child_weight":[1,3,5,7],"model__gamma":
[0.0,0.1,0.2,0.3,0.4],"model__colsample_bytree":[0.3,0.4,0.5,0.7],
'model__subsample': (0.01, 1.0, 'uniform'),
'model__colsample_bytree': [0.3,0.4,0.5,0.7],
'model__colsample_bylevel': (0.01, 1.0, 'uniform'),
    'model__n_estimators': (50, 100,200,300)}

rfe = RFE(estimator=LogisticRegression(max_iter=600))
under = RandomUnderSampler(random_state=40)
preprocessor_ = ColumnTransformer(transformers =[
    ('scale',StandardScaler(), X_train.columns)], remainder ='passthrough

model= xgb.XGBClassifier(use_label_encoder=False, n_jobs = 1,
   objective = 'binary:logistic', eval_metric = 'error', tree_method='app

model_pipeline=Pipeline(steps=[("preprocessor_", preprocessor_),
               ('rfe',rfe),('under', under),('model', model)])

cv = StratifiedKFold(n_splits=5,shuffle=True,random_state=42)
```

```
opt = BayesSearchCV( model_pipeline ,
          [(random_grid , 80)],cv=cv,scoring='f1')
opt.fit(X_train , y_train)

print(opt.best_params_)
print(opt.best_score_)

random_grid = {'rfe__n_features_to_select':[6,7,8,9,10],
'under__sampling_strategy':[0.5,0.8,1],'model__base_estimator__C':
    [0.0001,0.001,0.01,0.1,1,10,100,1000],
    'model__base_estimator__max_iter':[500,1000,5000,10000],
    'model__learning_rate': arange(0.1, 1, 0.1),
    "model__n_estimators": [ 50, 100,200,250,400]  }

rfe = RFE(estimator=LogisticRegression(max_iter=400))
under = RandomUnderSampler(random_state=40)
preprocessor_ = ColumnTransformer(transformers =[
    ('scale',StandardScaler(), X_train.columns)], remainder ='passthrough

model = AdaBoostClassifier(base_estimator = LogisticRegression(),
                        algorithm='SAMME')

model_pipeline = Pipeline(steps =[("preprocessor_", preprocessor_),
                        ('rfe',rfe),('under', under),('model', model)])

cv = StratifiedKFold(n_splits=5,shuffle=True,random_state=42)

opt = BayesSearchCV( model_pipeline ,
           [(random_grid , 70)],cv=cv,scoring='f1')
opt.fit(X_train , y_train)

print(opt.best_params_)
print(opt.best_score_)

under = RandomUnderSampler(sampling_strategy=0.5,random_state=10)
#rfe = RFE(estimator=LogisticRegression(C=0.001,
#solver='lbfgs',max_iter=400), n_features_to_select=11)#xgb
rfe = RFE(estimator=LogisticRegression(max_iter=400),
          n_features_to_select=9)#Adaboost with rfe

model=AdaBoostClassifier(base_estimator=LogisticRegression(
    C=1000,max_iter=400),learning_rate=0.6,
n_estimators=200,algorithm='SAMME')#with rfe
#model=AdaBoostClassifier(base_estimator=LogisticRegression(),
#algorithm='SAMME') #Adaboost default
```

```
#model=AdaBoostClassifier(base_estimator=LogisticRegression(C=100),
#learning_rate=0.6,n_estimators=400,algorithm='SAMME')# without rfe

#model= xgb.XGBClassifier(colsample_bylevel=1,
#colsample_bytree=0.7,gamma=0.4,learning_rate=0.05,
#max_depth=3,min_child_weight=7,n_estimators=200,subsample=1,
#use_label_encoder=False,n_jobs = 1,objective = 'binary:logistic',
#eval_metric = 'error', tree_method='approx')# with rfe

#model= xgb.XGBClassifier(colsample_bylevel=0.01,colsample_bytree=0.7,
#gamma=0.2,learning_rate=0.05, max_depth=3,min_child_weight=1,
#n_estimators=300,subsample=1,use_label_encoder=False,n_jobs=1,
#objective='binary:logistic',eval_metric='error',
                  tree_method='approx')# without rfe

#model= xgb.XGBClassifier(use_label_encoder=False,n_jobs=1,
#objective='binary:logistic',eval_metric='error',
                    tree_method='approx') #default

pipeline_model = Pipeline(steps=[ ("preprocessor_", preprocessor_),
     ('rfe',rfe),('under',under),('model',model)])
#pipeline_model = Pipeline(steps=[ ("preprocessor_", preprocessor_),
#('under',under),('model',model)]) #without rfe

pipeline_model.fit(X_train, y_train)

y_pred_train=pipeline_model.predict(X_train)

conf_pred_train=confusion_matrix(y_train,y_pred_train)
print(conf_pred_train)

score_train=f1_score(y_train,y_pred_train)
print("F1 on training set are {}".format(score_train))
y_pred_test = pipeline_model.predict(X_test)

conf_pred_test = confusion_matrix(y_test, y_pred_test)
print(conf_pred_test)
score=f1_score(y_test,y_pred_test)
print(" F1 on test set are {}".format(score))
print(classification_report(y_test, y_pred_test))

recall_sensitivity = recall_score(y_test, y_pred_test, pos_label=1)

recall_specificity = recall_score(y_test, y_pred_test, pos_label=0)
```

```
recall_sensitivity , recall_specificity

preprocessor_ = ColumnTransformer(transformers =[
    ('scale',StandardScaler(), X_train.columns)], remainder ='passthrough
under = RandomUnderSampler(sampling_strategy =0.5,random_state =10)

rfe_xgb =  RFE(estimator=LogisticRegression(C=0.001,solver='lbfgs'
                        ,max_iter =400), n_features_to_select =11)#xgb
rfe_ada = RFE(estimator=LogisticRegression(max_iter =400),
              n_features_to_select =9)#Adaboost with rfe

model_xgb= xgb.XGBClassifier(colsample_bylevel =1,colsample_bytree =0.7,
gamma=0.4,learning_rate =0.05, max_depth =3,min_child_weight =7,
n_estimators =200,subsample =1, use_label_encoder=False,
n_jobs = 1,objective = 'binary:logistic',eval_metric = 'error',
                               tree_method='approx')# with rfe

model_ada=AdaBoostClassifier(base_estimator=LogisticRegression(
    C=1000,max_iter =400), learning_rate =0.6,
                              n_estimators =200,algorithm ='SAMME')

pipeline_model_xgb = Pipeline(steps=[ ("preprocessor_", preprocessor_),
              ('rfe_xgb',rfe_xgb),('under',under),('model_xgb',model_xgb)
pipeline_model_ada = Pipeline(steps=[ ("preprocessor_", preprocessor_),
           ('rfe_ada',rfe_ada),('under',under),('model_ada',model_ada)])


model_ada=pipeline_model_ada.fit(X_train, y_train)

probs_ada = model_ada.predict_proba(X_test)[:, 1]

model_xgb =  pipeline_model_xgb.fit(X_train, y_train)

probs_xgb = model_xg.predict_proba(X_test)[:, 1]

from sklearn.metrics import roc_auc_score, roc_curve

auc_ada = roc_auc_score(y_test, probs_ada)
fpr_ada, tpr_ada, thresholds_rf = roc_curve(y_test, probs_ada)

auc_xgb = roc_auc_score(y_test, probs_xgb)
fpr_xgb, tpr_xgb, thresholds_xgb = roc_curve(y_test, probs_xgb)

plt.figure(figsize =(7, 4))

plt.plot(fpr_ada, tpr_ada, label=f'Adaboost (AUC = {auc_ada:.3f})')
plt.plot(fpr_xg, tpr_xg,
     label=f'eXtreme Gradient Boosting (AUC = {auc_xgb:.3f})')
plt.plot([0, 1], [0, 1], color='blue', linestyle='--', label='Baseline')
plt.title('ROC Curve', size=20)
plt.xlabel('False Positive Rate', size=14)
plt.ylabel('True Positive Rate', size=14)
plt.legend();

rfe = RFE(estimator=LogisticRegression(C=0.001,solver='lbfgs',
                      max_iter=400), n_features_to_select=11)#xgb
#rfe = RFE(estimator=LogisticRegression(max_iter=400),
                          #n_features_to_select=9)#Adaboost

fit = rfe.fit(X_train, y_train)
print("Num Features: %d" % fit.n_features_)
print("Selected Features: %s" % fit.support_)
print("Feature Ranking: %s" % fit.ranking_)

print(X_train.columns[rfe.support_])
```

C Code for Project 3

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import make_column_transformer
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score,
precision_score, recall_score, f1_score

from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import StratifiedKFold
from imblearn.pipeline import Pipeline
from imblearn.under_sampling import RandomUnderSampler
from sklearn.metrics import confusion_matrix
from sklearn.naive_bayes import GaussianNB
from numpy import mean
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import Perceptron

from numpy import std
from sklearn.impute import SimpleImputer
from sklearn.compose import ColumnTransformer
from sklearn.feature_selection import RFE
from sklearn.metrics import classification_report
import xgboost as xgb
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import VotingClassifier
from imblearn.ensemble import BalancedBaggingClassifier
from skopt import BayesSearchCV
from imblearn.ensemble import BalancedRandomForestClassifier
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.stats import skew, kurtosis

df = pd.read_csv('Deidentified_data(approved)_.csv')
#numerical_ix = df.select_dtypes(include=
#['int64', 'float64']).columns

df['age']=df['age'].replace('89+', 91)
df.age = df.age.astype(int)

cleanup_nums = {"race":     {"Other_Pacific_Islander": 1,
    "Native_American_Alaskan": 2, "Asian":3 , "Declined":4,
    "White":5,  "Black":6, "Other":7}, "gender": {"F": 1, "M": 0}}

df=df.replace(cleanup_nums)
df=df.dropna(how='any',axis=0,thresh=df.shape[1]*0.7)
df=df.dropna(how='any',axis=1,thresh=df.shape[0]*0.8)

y=df['death']
X=df.drop(columns='death')

cols=["age","diastolicBP","systolicBP","cr","NLratio","pulseOx","bmi",
"temperature","bun","troponin","ptt","creatine_kinase" ,"potassium"]
X[cols].describe()

for col in df[cols]:
    print("Skew: %s %.3f" % (col,skew(df[col])))

for col in  df[cols]:
    print("kurtosis: %s %.3f" % (col,kurtosis(df[col])))

cols=['age',  'ventilator', 'diastolicBP', 'systolicBP', 'cr','NLratio',
'pulseOx','temperature', 'bun', 'troponin', 'ptt',
 'bmi', 'creatine_kinase', 'potassium']
```

```
myheart = df['death']
myage = df['age']
heart_0 = myheart == 0
heart_1 = myheart == 1

sns.countplot(x='death',data=df)

sns.countplot(x='ventilator',hue='death',data=df)

plt.hist(myage[heart_0], edgecolor='blue', alpha=0.5,
bins =range(15,95,5), label='0')
plt.hist(myage[heart_1], edgecolor='blue', alpha=0.5,
bins = range(15,95,5) , label='1')


plt.xlabel('age')
plt.ylabel('Frequency')
plt.show()

myheart = df['death']
mydiastolicBP = df['diastolicBP']

plt.hist(mydiastolicBP[heart_0], edgecolor='blue', alpha=0.5,
bins = range(0,120,8), label='0')
plt.hist(mydiastolicBP[heart_1], edgecolor='blue', alpha=0.5,
bins = range(0,120,8), label='1')
plt.legend(loc='upper right')
plt.xlabel('diastolicBP')
plt.ylabel('Frequency')
plt.show()

myheart = df['death']
mysystolicBP = df['systolicBP']

plt.hist(mysystolicBP[heart_0], edgecolor='blue', alpha=0.5,
bins = range(0,215,14), label='0')
plt.hist(mysystolicBP[heart_1], edgecolor='blue', alpha=0.5,
bins = range(0,215,14), label='1')
plt.legend(loc='upper right')
plt.xlabel('systolicBP')
plt.ylabel('Frequency')
plt.show()

myheart = df['death']
mycr = df['cr']
```

```
plt.hist(mycr[heart_0], edgecolor='blue', alpha=0.5,
bins = range(0,32,2), label='0')
plt.hist(mycr[heart_1], edgecolor='blue', alpha=0.5,
bins = range(0,32,2), label='1')
plt.legend(loc='upper right')
plt.xlabel('cr')
plt.ylabel('Frequency')
plt.show()

myheart = df['death']
myNLratio = df['NLratio']

plt.hist(myNLratio[heart_0], edgecolor='blue', alpha=0.5,
bins = range(0,30,2), label='0')
plt.hist(myNLratio[heart_1], edgecolor='blue', alpha=0.5,
bins = range(0,30,2), label='1')
plt.legend(loc='upper right')
plt.xlabel('NLratio')
plt.ylabel('Frequency')
plt.show()

myheart = df['death']
mypulseOx = df['pulseOx']

plt.hist(mypulseOx[heart_0], edgecolor='blue', alpha=0.5,
bins = range(10,95,6), label='0')
plt.hist(mypulseOx[heart_1], edgecolor='blue', alpha=0.5,
bins = range(10,95,6), label='1')
plt.legend(loc='upper right')
plt.xlabel('pulseOx')
plt.ylabel('Frequency')
plt.show()

myheart = df['death']
mytemperature = df['temperature']

plt.hist(mytemperature[heart_0], edgecolor='blue',
alpha=0.5, bins = range(85,125,2), label='0')
plt.hist(mytemperature[heart_1], edgecolor='blue',
alpha=0.5, bins = range(85,125,2), label='1')
plt.legend(loc='upper right')
plt.xlabel('temperature')
plt.ylabel('Frequency')
plt.show()
```

```
myheart = df['death']
mybun = df['bun']

plt.hist(mybun[heart_0], edgecolor='blue', alpha=0.5,
bins =  range(5,305,20), label='0')
plt.hist(mybun[heart_1], edgecolor='blue', alpha=0.5,
bins =  range(5,305,20), label='1')
plt.legend(loc='upper right')
plt.xlabel('bun')
plt.ylabel('Frequency')
plt.show()

cols=['age',  'ventilator', 'diastolicBP', 'systolicBP', 'cr','ptt',
 'bmi','NLratio', 'pulseOx', 'temperature', 'bun', 'troponin',
       'creatine_kinase', 'potassium']

myheart = df['death']
myage = df['age']

myheart = df['death']
mytroponin = df['troponin']

plt.hist(mytroponin[heart_0], edgecolor='blue', alpha=0.5, bins =
[0,0.015,0.03,0.045,0.06,0.075,0.09,0.105,0.12,0.135,
0.15,0.165,0.17,0.185,0.2], label='0')
plt.hist(mytroponin[heart_1], edgecolor='blue', alpha=0.5, bins
=[0,0.015,0.03,0.045,0.06,0.075,0.09,0.105,0.12,0.135,0.15,0.165,
0.17,0.185,0.2], label='1')
plt.legend(loc='upper right')
plt.xlabel('troponin')
plt.ylabel('Frequency')
plt.show()

myheart = df['death']
myptt = df['ptt']

plt.hist(myptt[heart_0], edgecolor='blue', alpha=0.5,
bins = range(19,200,15), label='0')
plt.hist(myptt[heart_1], edgecolor='blue', alpha=0.5,
bins = range(19,200,15), label='1')
plt.legend(loc='upper right')
plt.xlabel('ptt')
plt.ylabel('Frequency')
plt.show()

myheart = df['death']
```

```
mybmi = df['bmi']
plt.hist(mybmi[heart_0], edgecolor='blue', alpha=0.5,
bins =range(9,55,3) , label='0')
plt.hist(mybmi[heart_1], edgecolor='blue', alpha=0.5,
bins =range(9,55,3) ,label='1')

#plt.hist(mybmi[heart_0], edgecolor='blue', alpha=0.5,
bins = [10,15,20,25,30,35,40,45,50,55,60,65,70,75], label='0')
#plt.hist(mybmi[heart_1], edgecolor='blue', alpha=0.5,
bins =  [10,15,20,25,30,35,40,45,50,55,60,65,70,75], label='1')
plt.legend(loc='upper right')
plt.xlabel('bmi')
plt.ylabel('Frequency')
plt.show()

myheart = df['death']
mycreatine_kinase = df['creatine_kinase']

plt.hist(mycreatine_kinase[heart_0], edgecolor='blue', alpha=0.5,
bins = range(19,1000,65), label='0')
plt.hist(mycreatine_kinase[heart_1], edgecolor='blue', alpha=0.5,
bins =  range(19,1000,65), label='1')
plt.legend(loc='upper right')
plt.xlabel('creatine_kinase')
plt.ylabel('Frequency')
plt.show()

myheart = df['death']
mypotassium = df['potassium']

plt.hist(mypotassium[heart_0], edgecolor='blue', alpha=0.5,
bins = [2,2.5,3,3.5,4,4.5,5,5.5,6,6.5,7,7.5,8,8.5,9], label='0')
plt.hist(mypotassium[heart_1], edgecolor='blue', alpha=0.5,
bins =  [2,2.5,3,3.5,4,4.5,5,5.5,6,6.5,7,7.5,8,8.5,9], label='1')
plt.legend(loc='upper right')
plt.xlabel('potassium')
plt.ylabel('Frequency')
plt.show()

X_train, X_test, y_train, y_test = train_test_split( X, y,
test_size=0.2,stratify=y, random_state=42)

numerical_ix= ['ventilator', 'albumin', 'diastolicBP', 'systolicBP',
       'cr', 'egfr', 'eosinophil', 'hgb', 'inr', 'lymphocyte',
       'NLratio', 'platelet', 'protein', 'pulse', 'pulseOx',
       'temperature', 'wbc', 'alt', 'ast', 'bun', 'calcium',
        'interleukin6', 'mcv', 'monocyte', 'mpv', 'rdw', 'troponin',
        'bmi', 'glucose', 'direct_bili', 'total_bili',
        'creatine_kinase', 'potassium', 'charlson_score',
        'neutrophil','rr','chloride', 'crp', 'ptt','indirect_bili']

categorical_features=['race']

numeric_transformer = Pipeline ( steps = [("imputer",
SimpleImputer(strategy="median")),('scale',StandardScaler())])
preprocessor_1 = ColumnTransformer(transformers
=[('numeric_transformer',numeric_transformer, numerical_ix),
    ('encode', OneHotEncoder(),categorical_features)],
    remainder='passthrough')

preprocessor_2 = ColumnTransformer(transformers =[("imputer",
SimpleImputer(strategy="median"), numerical_ix)],remainder='passthrough')
under = RandomUnderSampler(sampling_strategy=0.5,random_state=40)

def get_models():
    models = dict()

    model = RandomForestClassifier()
    models['Random forest']=Pipeline(steps=[('preprocessor_2'
    ,preprocessor_2),('under', under),('m',model)])

    model = LogisticRegression()
    models['DecisionTree']=Pipeline(steps=[('preprocessor_1',
    preprocessor_1),('under',under),('m',model)])

    model= xgb.XGBClassifier()
    models['xgboost']=Pipeline(steps=[('preprocessor_2',
    preprocessor_2),('under',under),('m',model)])

    model = SVC()
    models['svm']=Pipeline(steps=[('preprocessor_1',preprocessor_1),
    ('under', under),('m',model)])

    model = AdaBoostClassifier(random_state=42)
    models['Adaboost'] = Pipeline(steps=[('preprocessor_2',
   preprocessor_2),('under',under),('m',model)])

    lr= LogisticRegression()
    gnb=GaussianNB()
    kn=KNeighborsClassifier(n_neighbors=1)
    base_methods=[('pipe_lr', lr),('pipe_gnb', gnb),('pipe_Kn',kn)]
    vote_model=VotingClassifier(estimators=base_methods,voting='hard')
    models['Voting'] =Pipeline(steps=[('preprocessor_1',
    preprocessor_1),('under', under),('m',vote_model)])

    return models

def evaluate_model(model, X, y):
    cv = StratifiedKFold(n_splits=5,shuffle=True)
    scores = cross_val_score(model, X, y, scoring='f1', cv=cv, n_jobs=-1)
    return scores

models = get_models()
results, names = list(), list()
for name, model in models.items():
    scores = evaluate_model(model, X_train, y_train)
    results.append(scores)
    names.append(name)
    print('>%s %.3f (%.3f)' % (name, mean(scores), std(scores)))

def get_models():
    models = dict()
    for i in range(5,30):
        rfe = RFE(estimator=DecisionTreeClassifier(),
        n_features_to_select=i)
        model = xgb.XGBClassifier()
        models[str(i)] = Pipeline(steps=[("preprocessor_2",
        preprocessor_2),('s',rfe),('under', under),('m',model)])
    return models

def evaluate_model(model, X, y):
    cv = StratifiedKFold(n_splits=5,shuffle=True,random_state=42)
    scores = cross_val_score(model, X, y, scoring='recall', cv=cv,
    n_jobs=-1, error_score='raise')
    return scores

models = get_models()
results, names = list(), list()
for name, model in models.items():
    scores = evaluate_model(model, X_train, y_train)
    results.append(scores)
    names.append(name)
    print('>%s %.3f (%.3f)' % (name, mean(scores), std(scores)))

numeric_transformer = Pipeline ( steps = [("imputer",
SimpleImputer(strategy="median")),('scale',StandardScaler())])
preprocessor_1 = ColumnTransformer(transformers
=[('numeric_transformer',numeric_transformer, numerical_ix),
 ('encode', OneHotEncoder(),categorical_features)],
                                    remainder='passthrough')

preprocessor_2 = ColumnTransformer(transformers =[("imputer",
SimpleImputer(strategy="median"), numerical_ix)],
                          remainder='passthrough')
under = RandomUnderSampler(sampling_strategy=0.5,random_state=40)
def get_models():
    models = dict()

    model = RandomForestClassifier()
    models['Random forest'] = Pipeline(steps=[('preprocessor_2'
    ,preprocessor_2),('under', under),('m',model)])

    model = LogisticRegression()
    models['DecisionTree'] = Pipeline(steps=[('preprocessor_1',
    preprocessor_1),('under', under),('m',model)])

    model= xgb.XGBClassifier()
    models['xgboost'] = Pipeline(steps=[('preprocessor_2',
    preprocessor_2),('under', under),('m',model)])

    model = SVC()
    models['svm'] = Pipeline(steps=[('preprocessor_1',preprocessor_1),
    ('under', under),('m',model)])

    model = AdaBoostClassifier(random_state=42)
    models['Adaboost'] = Pipeline(steps=[('preprocessor_2',
    preprocessor_2),('under', under),('m',model)])

    lr= LogisticRegression()
    gnb=GaussianNB()
    kn=KNeighborsClassifier(n_neighbors=1)
    base_methods=[('pipe_lr', lr),('pipe_gnb', gnb),('pipe_Kn',kn)]
    vote_model=VotingClassifier(estimators=base_methods,voting='hard')

    models['Voting'] =Pipeline(steps=[('preprocessor_1',
    preprocessor_1),('under', under),('m',vote_model)])

    return models

def evaluate_model(model, X, y):
    cv = StratifiedKFold(n_splits=5,shuffle=True)
    scores = cross_val_score(model, X, y, scoring='f1',
    cv=cv, n_jobs=-1)
    return scores

models = get_models()
results, names = list(), list()
for name, model in models.items():
    scores = evaluate_model(model, X_train, y_train)
    results.append(scores)
    names.append(name)
    print('>%s %.3f (%.3f)' % (name, mean(scores), std(scores)))

def get_models():
    models = dict()
    for i in range(5,30):
        rfe = RFE(estimator=DecisionTreeClassifier(),
                                n_features_to_select=i)
        model = xgb.XGBClassifier()
        models[str(i)] = Pipeline(steps=[("preprocessor_2",
        preprocessor_2),('s',rfe),('under', under),('m',model)])
    return models

def evaluate_model(model, X, y):
    cv = StratifiedKFold(n_splits=5,shuffle=True,random_state=42)
    scores = cross_val_score(model, X, y, scoring='recall', cv=cv,
    n_jobs=-1, error_score='raise')
    return scores

models = get_models()
results, names = list(), list()
for name, model in models.items():
    scores = evaluate_model(model, X_train, y_train)
    results.append(scores)
    names.append(name)
    print('>%s %.3f (%.3f)' % (name, mean(scores), std(scores)))

n_estimators=[150,200,250,300,450]
max_features = ['auto', 'sqrt']
max_depth=[8,9,10,11,12]
min_samples_split = [2,5, 10]
min_samples_leaf = [1, 2,3, 4]
bootstrap = [True, False]
random_grid = {'rfe__estimator__max_depth':[2,4,6,8,10,12,None],
'rfe__n_features_to_select':[10,11,12,13,14,15],
    'under__sampling_strategy':[0.5,0.8],'model__n_estimators':
    n_estimators, 'model__max_features': max_features,
 'model__max_depth': max_depth, 'model__min_samples_split':
 min_samples_split, 'model__min_samples_leaf': min_samples_leaf,
  'model__bootstrap': bootstrap}

rfe = RFE(estimator=DecisionTreeClassifier())
under = RandomUnderSampler(random_state=40)
preprocessor_2=ColumnTransformer(transformers=[("imputer",
SimpleImputer(strategy="median"),numerical_ix)],
                              remainder='passthrough')

model = RandomForestClassifier(random_state=42)

model_pipeline = Pipeline(steps =[("preprocessor_2",
preprocessor_2),#('rfe',rfe), ('under', under),('model', model)])

cv = StratifiedKFold(n_splits=5,shuffle=True,random_state=42)

opt = BayesSearchCV( model_pipeline,[(random_grid, 60)],
                                            cv=cv,scoring='f1')
opt.fit(X_train, y_train)

print(opt.best_params_)
print(opt.best_score_)


model= xgb.XGBClassifier(use_label_encoder=False,n_jobs = 1,
objective = 'binary:logistic',eval_metric = 'auc',
silent=1, tree_method='approx')

under = RandomUnderSampler(random_state=40)
rfe = RFE(estimator=DecisionTreeClassifier())

preprocessor_2=ColumnTransformer(transformers=[("imputer",
SimpleImputer(strategy="median"),numerical_ix)],
                                    remainder='passthrough')

random_grid = {'rfe__estimator__max_depth':[2,4,6,8,10,12,None],
'rfe__n_features_to_select':[10,11,12,13,14,15],
'under__sampling_strategy':[0.5,0.8],"model__learning_rate":
[0.05,0.10,0.15,0.20,0.25,0.30],
"model__max_depth":[3,4,5,6,8,10,12,15],
"model__min_child_weight":[1,3,5,7],
"model__gamma":[0.0,0.1,0.2,0.3,0.4],
"model__colsample_bytree":[0.3,0.4,0.5,0.7],'model__subsample':
(0.01, 1.0, 'uniform'),
'model__colsample_bytree': [0.3,0.4,0.5,0.7],
'model__colsample_bylevel': (0.01, 1.0, 'uniform'),
'model__n_estimators': (50, 100,200,300)}
```

```
#model_pipeline = Pipeline(steps =[("preprocessor_2", preprocessor_2),
('rfe',rfe),('under', under),('model',model)])
model_pipeline = Pipeline(steps =[("preprocessor_2", preprocessor_2),
('under', under),('model',model)])

cv = StratifiedKFold(n_splits=5,shuffle=True)

opt = BayesSearchCV( model_pipeline,[(random_grid, 100)],cv=cv,
scoring='f1',n_jobs = 3,verbose = 0,refit = True,random_state = 42)

opt.fit(X_train, y_train)

print(opt.best_params_)
print(opt.best_score_)

rfe = RFE(estimator=DecisionTreeClassifier(max_depth=8),
                              n_features_to_select=15)#xgb
#rfe = RFE(estimator=DecisionTreeClassifier(max_depth=6),
                                  n_features_to_select=15)#rf

preprocessor_2=ColumnTransformer(transformers=[("imputer",
 SimpleImputer(strategy="median"),numerical_ix)],
          remainder='passthrough')

#under = RandomUnderSampler(sampling_strategy=0.5,
                               random_state=40)#xgb default
#under = RandomUnderSampler(sampling_strategy=0.728126631947327,
                             random_state=40)#rf without rfe

#under = RandomUnderSampler(sampling_strategy=0.5839989677305552,
#random_state=40)#rf with rfe
under = RandomUnderSampler(sampling_strategy=0.8,random_state=40)#xgb

#model = RandomForestClassifier(bootstrap=True,max_depth=10,
#max_features='sqrt',n_estimators=450,min_samples_leaf=3,
    #  min_samples_split=2,random_state=42) #with rfe

#model = RandomForestClassifier(bootstrap=True,max_depth=12,
#max_features='auto',n_estimators=150,min_samples_leaf=1,
#  min_samples_split=2,random_state=42) #without rfe

#model = RandomForestClassifier() #default

#model= xgb.XGBClassifier(use_label_encoder=False,n_jobs = 1,
objective = 'binary:logistic',eval_metric = 'auc',silent=1,
                tree_method='approx')# defeault hyperparameters
#model= xgb.XGBClassifier(use_label_encoder=False,n_jobs = 1,
#objective ='binary:logistic',eval_metric = 'auc',
#colsample_bylevel=1,colsample_bytree=0.7,gamma=0.0,
learning_rate=0.05,max_depth=6,min_child_weight=1,n_estimators=300,
subsample=1, silent=1, tree_method='approx') #Without rfe

model= xgb.XGBClassifier(use_label_encoder=False,n_jobs = 1,
objective = 'binary:logistic',eval_metric = 'auc',
colsample_bylevel=0.01,colsample_bytree=0.7,gamma=0.4,
learning_rate=0.05,max_depth=3,min_child_weight=1,
n_estimators=100,subsample=0.7173804504207143, silent=1,
tree_method='approx') #With rfe

pipeline_model = Pipeline(steps=[ ("preprocessor_2",
preprocessor_2),('rfe',rfe),('under',under),('model',model)])
#pipeline_model = Pipeline(steps=[ ("preprocessor_2",
preprocessor_2),('under',under),('model',model)])

pipeline_model.fit(X_train, y_train)

y_pred_train=pipeline_model.predict(X_train)

conf_pred_train=confusion_matrix(y_train,y_pred_train)
print(conf_pred_train)

score_train=f1_score(y_train,y_pred_train)
print(" F1 on training set {}".format(score_train))
y_pred_test = pipeline_model.predict(X_test)

conf_pred_test = confusion_matrix(y_test, y_pred_test)
print(conf_pred_test)
score=f1_score(y_test,y_pred_test)
print(" F1 on test set {}".format(score))
print(classification_report(y_test, y_pred_test))

recall_sensitivity = recall_score(y_test, y_pred_test, pos_label=1)

recall_specificity = recall_score(y_test, y_pred_test, pos_label=0)

recall_sensitivity, recall_specificity

preprocessor_2 = ColumnTransformer(transformers =[("imputer",
SimpleImputer(strategy="median"), numerical_ix)],
                             remainder='passthrough')
under_xgb = RandomUnderSampler(sampling_strategy=0.8,random_state=40)
```

```
under_rf = RandomUnderSampler(sampling_strategy=0.5839989677305552,
random_state=40)

rfe_xgb = RFE(estimator=DecisionTreeClassifier(max_depth=8),
n_features_to_select=15)#xgb
rfe_rf = RFE(estimator=DecisionTreeClassifier(max_depth=6),
n_features_to_select=15)#rf

model_xgb= xgb.XGBClassifier(use_label_encoder=False,n_jobs = 1,
objective = 'binary:logistic',eval_metric = 'auc',
colsample_bylevel=0.01,colsample_bytree=0.7,gamma=0.4,
learning_rate=0.05,max_depth=3,min_child_weight=1,
n_estimators=100,subsample=0.7173804504207143, silent=1,
                tree_method='approx')


model_rf = RandomForestClassifier(bootstrap=True,
max_depth=10,max_features='sqrt',n_estimators=450,min_samples_leaf=3,
                 min_samples_split=2,random_state=42) #with rfe

pipeline_model_xgb = Pipeline(steps=[ ("preprocessor_2",
preprocessor_2),('rfe_xgb',rfe_xgb),('under_xgb',under_xgb),
('model_xgb',model_xgb)])
pipeline_model_rf = Pipeline(steps=[ ("preprocessor_2",
preprocessor_2),('rfe_rf',rfe_rf),('under_rf',under_rf),
('model_rf',model_rf)])


model_rf = pipeline_model_xgb.fit(X_train, y_train)

probs_rf = model_rf.predict_proba(X_test)[:, 1]

model_xg = pipeline_model_rf.fit(X_train, y_train)

probs_xg = model_xg.predict_proba(X_test)[:, 1]

from sklearn.metrics import roc_auc_score, roc_curve


auc_rf = roc_auc_score(y_test, probs_rf)
fpr_rf, tpr_rf, thresholds_rf = roc_curve(y_test, probs_rf)

auc_xg = roc_auc_score(y_test, probs_xg)
fpr_xg, tpr_xg, thresholds_xg = roc_curve(y_test, probs_xg)

plt.figure(figsize=(7, 4))
plt.plot(fpr_rf, tpr_rf, label=f'Random Forest
(AUC = {auc_rf:.3f})')
plt.plot(fpr_xg, tpr_xg, label=f'eXtreme Gradient Boosting (AUC =
{auc_xg:.3f})')
plt.plot([0, 1], [0, 1], color='blue', linestyle='--', label='Baseline')
plt.title('ROC Curve', size=20)
plt.xlabel('False Positive Rate', size=14)
plt.ylabel('True Positive Rate', size=14)
plt.legend();

imputer = SimpleImputer(strategy="median")
X=imputer.fit_transform(X_train)
X_train = pd.DataFrame(X, columns=X_train.columns)

#rfe = RFE(estimator=DecisionTreeClassifier(max_depth=8),
n_features_to_select=15)#xgb
rfe = RFE(estimator=DecisionTreeClassifier(max_depth=6),
n_features_to_select=15)#rf

fit = rfe.fit(X_train, y_train)

print("Num Features: %d" % fit.n_features_)
print("Selected Features: %s" % fit.support_)
print("Feature Ranking: %s" % fit.ranking_)

print(X_train.columns[rfe.support_])
```

References

[1] *Precision Medicine*. bit.ly/3AG6n4d. accessed November ,2018. 2018.

[2] G. S. Ginsburg and K. A. Phillips. “Precision medicine: from science to value”. In: *Encyclopedia of Bioinformatics and Computational Biology*. Ed. by S. Ranganathan, M. Gribskov, K. Nakai, and C. Schönbach. Elsevier, 2018, pp. 364–371.

[3] F. Li. “Introduction to Machine Learning”. In: *IBM J. Res. Dev* 3 (1959), p. 210.

[4] T. T. Mark Esposito Kariappa Bheemaiah. *What is machine learning?* bit.ly/3TlGldA. accessed November ,2021. 2017.

[5] J. Brownlee. *How to Prepare Data For Machine Learning*. bit.ly/3PRZzVw. accessed July ,2022.

[6] I. Sydorenko. *How to Choose the Right Machine Learning Algorithm: A Pragmatic Approach*. bit.ly/3clA3dE. accessed November ,2021. 2021.

[7] *Machine Learning Algorithms—Top 5 Examples in Real Life*. bit.ly/3AOuzl2. accessed July ,2022.

[8] W. Slikker Jr. “Biomarkers and their impact on precision medicine”. In: *Experimental Biology and Medicine* 243.3 (2018), p. 211.

[9] R. Dhingra and R. S. Vasan. “Biomarkers in cardiovascular disease: Statistical assessment and section on key novel heart failure biomarkers”. In: *Trends in cardiovascular medicine* 27.2 (2017), pp. 123–133.

[10] *Biomarkers Diagnostics*. bit.ly/3y86nsf. accessed June ,2022.

[11] K. McCauley. *Biomarkers in Precision Medicine*. bit.ly/3QDvryv. accessed July ,2021.

[12] L. Bravo-Merodio, A. Acharjee, D. Russ, V. Bisht, J. A. Williams, L. G. Tsaprouni, and G. V. Gkoutos. “Translational biomarkers in the era of precision medicine”. In: *Advances in clinical chemistry* 102 (2021), pp. 191–232.

[13] R. Tandon. *What Is A Biomarker? Definitions, Types, And Research Applications Explained*. bit.ly/3OF06JQ. accessed July ,2022.

[14] A. J. Vargas and C. C. Harris. “Biomarker development in the precision medicine era: lung cancer as a case study”. In: *Nature Reviews Cancer* 16.8 (2016), pp. 525–537.

[15] M. Olivier, R. Asmis, G. A. Hawkins, T. D. Howard, and L. A. Cox. “The need for multi-omics biomarker signatures in precision medicine”. In: *International journal of molecular sciences* 20.19 (2019), p. 4781.

[16] *A brief guide to genomics*. bit.ly/3ww44hq. accessed November ,2021.

[17] A. Telenti. “Integrating metabolomics with genomics”. In: *Pharmacogenomics* 19.18 (2018), pp. 1377–1381.

[18] E. Milward, A. Shahandeh, M. Heidari, D. Johnstone, N. Daneshi, and H. Hondermarck. “Transcriptomics”. In: *Encyclopedia of Cell Biology*. Ed. by R. A. Bradshaw and P. D. Stahl. Academic Press, 2016, pp. 160–165.

[19] Liang and Kung-Hao. “Bioinformatics for biomedical science and clinical applications”. In: Elsevier, 2013.

[20] F. Vafaee, H. Dashti, and H. Alinejad-Rokny. “Transcriptomic Data Normalization”. In: *Encyclopedia of Bioinformatics and Computational Biology*. Ed. by S. Ranganathan, M. Gribskov, K. Nakai, and C. Schönbach. Academic Press, 2019, pp. 364–371.

[21] N. Amiri-Dashatan, M. Koushki, H.-A. Abbaszadeh, M. Rostami-Nejad, and M. Rezaei-Tavirani. “Proteomics applications in health: biomarker and drug discovery and

food industry”. In: *Iranian journal of pharmaceutical research: IJPR* 17.4 (2018), p. 1523.

[22] R. Kalluri. “The biology and function of fibroblasts in cancer”. In: *Nature Reviews Cancer* 16.9 (2016), pp. 582–598.

[23] Y. Xie, W.-Y. Meng, R.-Z. Li, Y.-W. Wang, X. Qian, C. Chan, Z.-F. Yu, X.-X. Fan, H.-D. Pan, C. Xie, et al. “Early lung cancer diagnostic biomarker discovery by machine learning methods”. In: *Translational oncology* 14.1 (2021), p. 100907.

[24] J. Zhang, X. Han, C. Gao, Y. Xing, Z. Qi, R. Liu, Y. Wang, X. Zhang, Y.-G. Yang, X. Li, et al. “5-Hydroxymethylome in circulating cell-free DNA as a potential biomarker for non-small-cell lung cancer”. In: *Genomics, proteomics & bioinformatics* 16.3 (2018), pp. 187–199.

[25] N. Tsoukalas, E. Aravantinou-Fatorou, P. Baxevanos, M. Tolia, K. Tsapakidis, M. Galanopoulos, M. Liontos, and G. Kyrgias. “Advanced small cell lung cancer (SCLC): new challenges and new expectations”. In: *Annals of translational medicine* 6.8 (2018).

[26] B. Senliol, G. Gulgezen, L. Yu, and Z. Cataltepe. “Fast Correlation Based Filter (FCBF) with a different search strategy”. In: *2008 23rd international symposium on computer and information sciences*. IEEE. 2008, pp. 1–4.

[27] Y.-H. Zhang, M. Jin, J. Li, and X. Kong. “Identifying circulating miRNA biomarkers for early diagnosis and monitoring of lung cancer”. In: *Biochimica et Biophysica Acta (BBA)-Molecular Basis of Disease* 1866.10 (2020), p. 165847.

[28] bit.ly/3OJXQS4.

[29] J. Yu, Y. Hu, Y. Xu, J. Wang, J. Kuang, W. Zhang, J. Shao, D. Guo, and Y. Wang. “LUADpp: an effective prediction model on prognosis of lung adenocarcinomas based on somatic mutational features”. In: *BMC cancer* 19.1 (2019), pp. 1–10.

[30] J. Wu, X. Zan, L. Gao, J. Zhao, J. Fan, H. Shi, Y. Wan, E. Yu, S. Li, X. Xie, et al. “A machine learning method for identifying lung cancer based on routine blood indices: qualitative feasibility study”. In: *JMIR medical informatics* 7.3 (2019), e13476.

[31] B. Roman-Canal, C. P. Moiola, S. Gatius, S. Bonnin, M. Ruiz-Miró, E. González, A. Ojanguren, J. L. Recuero, A. Gil-Moreno, J. M. Falcón-Pérez, et al. “EV-associated miRNAs from pleural lavage as potential diagnostic biomarkers in lung cancer”. In: *Scientific reports* 9.1 (2019), pp. 1–9.

[32] C. Zhang, W. Leng, C. Sun, T. Lu, Z. Chen, X. Men, Y. Wang, G. Wang, B. Zhen, and J. Qin. “Urine proteome profiling predicts lung cancer from control cases and other tumors”. In: *EBioMedicine* 30 (2018), pp. 120–128.

[33] L. Aprupe, G. Litjens, T. J. Brinker, J. van der Laak, and N. Grabe. “Robust and accurate quantification of biomarkers of immune cells in lung cancer micro-environment using deep convolutional neural networks”. In: *PeerJ* 7 (2019), e6335.

[34] K. O’Shea, S. J. Cameron, K. E. Lewis, C. Lu, and L. A. Mur. “Metabolomic-based biomarker discovery for non-invasive lung cancer screening: A case study”. In: *Biochimica et Biophysica Acta (BBA)-General Subjects* 1860.11 (2016), pp. 2682–2687.

[35] R. Casanova, S. Saldana, S. L. Simpson, M. E. Lacy, A. R. Subauste, C. Blackshear, L. Wagenknecht, and A. G. Bertoni. “Prediction of incident diabetes in the Jackson Heart Study using high-dimensional machine learning”. In: *PloS one* 11.10 (2016), e0163942.

[36] L. Yoffe, A. Polsky, A. Gilam, C. Raff, F. Mecacci, A. Ognibene, F. Crispi, E. Gratacós, H. Kanety, S. Mazaki-Tovi, et al. “Early diagnosis of gestational diabetes mellitus using

circulating microRNAs". In: *European journal of endocrinology* 181.5 (2019), pp. 565–577.

[37] Y. Xiong, L. Lin, Y. Chen, S. Salerno, Y. Li, X. Zeng, and H. Li. "Prediction of gestational diabetes mellitus in the first 19 weeks of pregnancy using machine learning techniques". In: *The Journal of Maternal-Fetal & Neonatal Medicine* (2020), pp. 1–7.

[38] M. Tayefi, M. Tajfard, S. Saffar, P. Hanachi, A. R. Amirabadizadeh, H. Esmaeily, A. Taghipour, G. A. Ferns, M. Moohebati, and M. Ghayour-Mobarhan. "hs-CRP is strongly associated with coronary heart disease (CHD): A data mining approach using decision tree algorithm". In: *Computer methods and programs in biomedicine* 141 (2017), pp. 105–109.

[39] L. Zhang, F. Ma, A. Qi, L. Liu, J. Zhang, S. Xu, Q. Zhong, Y. Chen, C.-y. Zhang, and C. Cai. "Integration of ultra-high-pressure liquid chromatography–tandem mass spectrometry with machine learning for identifying fatty acid metabolite biomarkers of ischemic stroke". In: *Chemical Communications* 56.49 (2020), pp. 6656–6659.

[40] Z. Jaadi. *A Step-by-Step Explanation of Principal Component Analysis (PCA).* `bit.ly/3qYt8vv`. accessed November ,2021. 2021.

[41] M. Brems. *A One-Stop Shop for Principal Component Analysis*. `bit.ly/3fT7SRq`. accessed November ,2021. 2017.

[42] M. Stephen. "Principal component analysis". In: *Machine learning: an algorithmic perspective*. 2nd ed. Chapman and Hall/CRC, 2014, pp. 133–136.

[43] A. Ghodsi. "Principal Component Analysis". Sept. 2006.

[44] A. Ghodsi. *Principal Components Analysis*. `bit.ly/3GPcrry`. accessed November ,2021. 2006.

[45] S. Jensen. *An Introduction to Lagrange Multipliers*. `bit.ly/3u7RhBN`. accessed November ,2021.

[46] *Singular Value decomposition*. `bit.ly/3qLsPDZ`. accessed October,2021. 2011.

[47] B. Ghojogh and M. Crowley. *Principal Component Analysis*. `bit.ly/33iVMxZ`. accessed October,2021. 2021.

[48] *Chapter 7 The Singular Value Decomposition (SVD)*. `bit.ly/3AvOu6v`. accessed November ,2021.

[49] Taotao. *Calculate PCA by hand*. `bit.ly/3Bs72qx`. accessed July ,2022.

[50] A. Ghodsi. *Data Visualization (Lecture 6)*. `bit.ly/3nKpRxt`. accessed November ,2021. 2006.

[51] K. Ezukwoke and S. Zareian. "Kernel methods for principal component analysis (PCA) A comparative study of classical and kernel PCA". In: *A preprint* (2019).

[52] E. Tai. *Kernel Functions in Non-linear Classification*. `bit.ly/3G5vhd7`. accessed November ,2021. 2019.

[53] J. Boccard and S. Rudaz. In: *Proteomic and Metabolomic Approaches to Biomarker Discovery*. Academic Press, 2013, pp. 437–438.

[54] A. ghodsi. *Lecture 19: Clustering*. `bit.ly/3g6zB11`. accessed December ,2021. 2006.

[55] J. Guttag. *Lecture 12: Clustering*. `bit.ly/35yFLoD`. MIT OpenCourseWare. 2016.

[56] A. S. S. A. Y. Wah. "A Comparison Study on Similarity and Dissimilarity Measures in Clustering Continuous Data". In: (2015).

[57] *Hierarchical Clustering / Dendrograms*. `bit.ly/3r9q2U1`. accessed December ,2021.

[58] P. B. G. Alvez. "Inference of a human brain fiber bundle atlas from high angular resolution diffusion imaging". PhD thesis. Université Paris Sud-Paris XI, 2011, p. 84.

[59] G. L. Team. *Hierarchical Clustering: What is a Dendrogram?* bit.ly/3ff9ogg. accessed December ,2021. 2020.

[60] C. R. Patlolla. *Understanding the concept of Hierarchical clustering Technique*. bit.ly/3IILiqW. accessed December ,2021. 2018.

[61] bit.ly/3fo0UmU. accessed December ,2021.

[62] A. Apon, F. Robinson, D. Brewer, L. Dowdy, D. Hoffman, and B. Lu. "Inital starting point analysis for k-means clustering: A case study". In: (2006).

[63] V. Kumar. *What is K-Means algorithm and how it works*. bit.ly/3R69xo1. accessed June ,2022.

[64] N. Sharma. *K-Means Clustering Explained*. bit.ly/34l7o3I. accessed December ,2021. 2021.

[65] *2.3. Clustering*. bit.ly/3ITOsZ9. accessed December ,2021.

[66] A. Goel. *Are you solving ML Clustering problems using K-Means?* bit.ly/3gcDJwm. accessed December ,2021. 2020.

[67] A. Soetewey. *The complete guide to clustering analysis: k-means and hierarchical clustering by hand and in R*. bit.ly/3p0X08s. accessed July ,2022.

[68] A. Kumar. *KMeans Silhouette Score Explained With Python Example*. bit.ly/3GCyDp3. accessed December ,2021. 2020.

[69] *Selecting the number of clusters with silhouette analysis on KMeans clustering*. bit.ly/3tmSfcU. accessed December ,2021.

[70] D. Dey. *Dunn index and DB index – Cluster Validity indices — Set 1*. bit.ly/3umFLSk. accessed July ,2022.

[71] T. Hastie, R. Tibshirani, and J. Friedman. In: *The elements of statistical learning*. Springer, 2009, pp. 119–121.

[72] A. A. Tokuç. *Gradient Descent Equation in Logistic Regression*. bit.ly/3IM9GZX. accessed July ,2022.

[73] A. Zhang. *Generalized Linear Models*. bit.ly/3o1aRvh. accessed December ,2021.

[74] D. Jurafsky and J. H. Martin. In: *Speech and Language Processing (An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition)*. 3rd ed. Stanford Univ, 2021, pp. 78–101.

[75] *Binary vs. Multi-Class Logistic Regression*. bit.ly/33oUamc. accessed December ,2021. 2018.

[76] R. Xia. *Lecture 3 Logistic Regression Softmax Regression*. bit.ly/3r40Orm. accessed December ,2021.

[77] S. N. Srihari. *Multiclass Logistic Regression*. bit.ly/3AAIjy5. accessed December ,2021.

[78] L. Chen. *Multiclass classification with softmax regression and gradient descent*. bit.ly/3Pe8Pnq. accessed July ,2022.

[79] M. Stephen. "Learning with Trees". In: *Machine learning: an algorithmic perspective (2nd ed.)* Chapman and Hall/CRC, 2014, pp. 249–266.

[80] M. Galarnyk. *Understanding Decision Trees for Classification (Python)*. bit.ly/33uZKDX. accessed September ,2021. 2019.

[81] E. Lisowski. *Entropy in Machine Learning*. bit.ly/3HVTipt. accessed November ,2021.

[82] P. Katiyar. *Decision Tree Pruning Techniques In Python*. bit.ly/3HUInMs. accessed July ,2021. 2021.

[83] P. Katiyar. *Different Classification Trees in Machine Learning*. bit.ly/3bpJBmT. accessed July ,2021.

[84] K. M. Leung. “Decision Trees and Decision Rules”. In: 2007.

[85] T. Hastie, R. Tibshirani, and J. Friedman. “Additive Models, Trees, and Related Methods”. In: *The elements of statistical learning*. Springer, 2009, pp. 295–336.

[86] J. Obukwelu. *What’s the KNN?* bit.ly/3fnquJ4. accessed December ,2021. 2020.

[87] B. Mirbozorgi. *Theory of K-Nearest Neighbors (KNN)*. bit.ly/3I2fkFL. accessed December ,2021. 2020.

[88] *K-Nearest Neighbor(KNN) Algorithm for Machine Learning*. bit.ly/33gOjjd. accessed December ,2021.

[89] D. Nelson. *What is a KNN (K-Nearest Neighbors)?* bit.ly/3toH1V6. accessed December ,2021. 2020.

[90] Nagarajramachandran. *K- Nearest Neighbor Explanation With Example*. bit.ly/3oMaUv9. accessed July ,2022.

[91] M. Jain. *What are Generative Learning Algorithms?* bit.ly/3I0qEDq. accessed November ,2021.

[92] M. Stephen. “Turning data into probabilities”. In: *Machine learning: an algorithmic perspective*. 2nd ed. Chapman and Hall/CRC, 2014, pp. 27–31.

[93] S. Prabhakaran. *How Naive Bayes Algorithm Works? (with example and full code)*. bit.ly/3zDlf0y. accessed October, 2021. 2018.

[94] J. Brownlee. *A Gentle Introduction to Bayes Theorem for Machine Learning*. bit.ly/33j0b3W. accessed October, 2021. 2019.

[95] S. Prabhakaran. *How Naive Bayes Algorithm Works? (with example and full code)*. bit.ly/3zDlf0y. accessed July ,2022.

[96] T. Hastie, R. Tibshirani, and J. Friedman. In: *The elements of statistical learning*. Springer, 2009, pp. 106–110.

[97] W. Wu, Y. Mallet, B. Walczak, W. Penninckx, D. Massart, S. Heuerding, and F. Erni. “Comparison of regularized discriminant analysis linear discriminant analysis and quadratic discriminant analysis applied to NIR data”. In: *Analytica Chimica Acta* 329.3 (1996), pp. 257–265.

[98] A. Lindholm, N. Wahlström, F. Lindsten, and T. B. Schön. “Supervised Machine Learning(Lecture notes for the Statistical Machine Learning course)”. The classification problem and three parametric classifiers (Chapter 3). Mar. 2019.

[99] A. Ghodsi. *Lec 3: QDA, Principal Component Analysis (PCA)*. bit.ly/35uzvOH. accessed November ,2021. 2015.

[100] K. Teknomo. *Linear Discriminant Analysis Numerical Example*. bit.ly/3C1I95u. accessed July ,2022. 2019.

[101] M. Stephen. “Support Vector Machines”. In: *Machine learning: an algorithmic perspective*. 2nd ed. Chapman and Hall/CRC, 2014, pp. 169–177.

[102] T. Hastie, R. Tibshirani, and J. Friedman. In: *The elements of statistical learning*. Springer, 2009, pp. 132–134.

[103] Abu-Mostafa. *e-Chapter 8 Support Vector Machine*. bit.ly/3A1bGuI. accessed July ,2022. 2015.

[104] T. Hastie, R. Tibshirani, and J. Friedman. In: *The elements of statistical learning*. Springer, 2009, pp. 418–424.

[105] F. Mh. *Multiple Linear Regression*. bit.ly/3sgHhna. accessed October,2021. 2021.

[106] *What is Multiple Linear Regression?* bit.ly/3uxGBML. accessed October,2021.
[107] S. S. *Multivariate Linear Regression.* bit.ly/333tQOM. accessed October,2021.
[108] A. Lindholm, N. Wahlström, F. Lindsten, and T. B. Schön. "Supervised Machine Learning(Lecture notes for the Statistical Machine Learning course)". Neural networks and deep learning (Chapter 7). Mar. 2019.
[109] Simplilearn. *What is Perceptron: A Beginners Guide for Perceptron.* bit.ly/3vjHTuD. accessed July ,2022.
[110] M. Nielsen. *Neural Networks and Deep Learning.* bit.ly/3F9Zvep. accessed June,2021. 2019.
[111] A. Kumar. *Neural Networks and Mathematical Models Examples.* bit.ly/3gj7FqP. accessed September,2021. 2020.
[112] N. Faruqui. *Implementation of A Deep Neural Network using MATLAB.* bit.ly/3GJ2qMR. accessed June,2021. 2019.
[113] J. Brownlee. *How to Choose an Activation Function for Deep Learning.* bit.ly/3HpiNhM. accessed October,2021. 2021.
[114] J. Brownlee. *A Gentle Introduction to the Rectified Linear Unit (ReLU).* bit.ly/3go6rul. accessed October,2021. 2020.
[115] S. Rajan. *A Walk-through of Regression Analysis Using Artificial Neural Networks in Tensorflow.* bit.ly/3AUyCud. accessed September,2021. 2021.
[116] *Applying Artificial Neural Networks (ANNs) for Linear Regression: Yay or Nay?* bit.ly/34caNST. accessed September,2021. 2020.
[117] J. Brownlee. *Deep Learning Models for Multi-Output Regression.* bit.ly/3onfX5h. accessed October,2021. 2020.
[118] S. Ronaghan. *Deep Learning: Which Loss and Activation Functions should I use?* bit.ly/3L0PrIR. accessed October,2021. 2018.
[119] J. Brownlee. *Multi-Label Classification with Deep Learning.* bit.ly/3AUKOeL. accessed October,2021. 2020.
[120] G. Sanderson and J. Pullen. *Backpropagation calculus.* bit.ly/33kfUQz. accessed June,2021. 2017.
[121] M. Nielsen. *Neural Networks and Deep Learning.* bit.ly/3zDf50Z. accessed June,2021. 2019.
[122] Raycad. *Convolutional Neural Network (CNN).* bit.ly/3IcOATb. accessed June,2021. 2017.
[123] A. Dwivedi. *Starting with convolutional neural network (CNN).* bit.ly/3fqfevt. accessed June,2021. 2019.
[124] A. Das. *Convolution Neural Network for Image Processing — Using Keras.* bit.ly/3GCstVJ. accessed June,2021. 2020.
[125] M. Nielsen. *Neural Networks and Deep Learning.* bit.ly/34pOPvk. accessed June,2021. 2019.
[126] V. K. Solegaonkar. *Convolutional Neural Networks.* bit.ly/350q0GV. accessed September,2021. 2019.
[127] A. Perera. *What is Padding in Convolutional Neural Network's(CNN's) padding.* bit.ly/3nxt97i. accessed September,2021. 2018.
[128] *Zero Padding In Convolutional Neural Networks Explained.* bit.ly/3rquYnL. accessed September,2021. 2018.

[129] A. Ng. *Convolutional Neural Networks: Step by Step*. bit.ly/33tmgNy. accessed September,2021. 2017.
[130] D. Guan. *Padding in Convolutional Neural Networks*. bit.ly/3Acjd8v. accessed September,2021. 2020.
[131] bit.ly/3tAFdZm. accessed September,2021.
[132] J. Sun. *Convolutional Neural Networks*. b.gatech.edu/3GJS4Mo. accessed September,2021. 2017.
[133] M. Stephen. "Decision by Committee: Ensemble Learning". In: *Machine learning: an algorithmic perspective (2nd ed.)* Chapman and Hall/CRC, 2014, pp. 267–280.
[134] Y. Pandya. *Ensemble Methods in Machine Learning*. bit.ly/3Kcrf4W. accessed November, 2021. 2021.
[135] J. Rocca. *Ensemble methods: bagging, boosting and stacking*. bit.ly/3Mo27du. accessed November, 2021. 2019.
[136] I. C. Education. *What is boosting?* ibm.co/3pwMvdW. accessed November,2021. 2021.
[137] T. Hastie, R. Tibshirani, and J. Friedman. "Boosting and Additive Trees". In: *The elements of statistical learning*. Springer, 2009, pp. 337–387.
[138] A. Navlani. *AdaBoost Classifier in Python*. bit.ly/3dymQid. accessed July ,2022.
[139] M. Stephen. "Bagging". In: *Machine learning: an algorithmic perspective (2nd ed.)* Chapman and Hall/CRC, 2014, pp. 273–274.
[140] T. Hastie, R. Tibshirani, and J. Friedman. "Bagging". In: *The elements of statistical learning*. Springer, 2009, pp. 282–283.
[141] J. Brownlee. *How to Develop a Random Forest Ensemble in Python*. bit.ly/3pBpAy1. accessed November, 2021. 2021.
[142] T. Hastie, R. Tibshirani, and J. Friedman. "Ramdom Forests". In: *The elements of statistical learning*. Springer, 2009, pp. 587–592.
[143] M. Stephen. "Random Forests". In: *Machine learning: an algorithmic perspective (2nd ed.)* Chapman and Hall/CRC, 2014, pp. 275–277.
[144] J. Brownlee. *A Gentle Introduction to Machine Learning Modeling Pipelines*. bit.ly/3orjwqJ. accessed July ,2022.
[145] D. K. *Anomaly Detection Using Isolation Forest in Python*. bit.ly/3yU7FYg. accessed July ,2022.
[146] J. Andras, W. Steinbrunn, P. Matthias, and D. Robert. *Heart Disease Data Set*. bit.ly/3RlS3TI. accessed July ,2022.
[147] K. Ikemura, E. Bellin, Y. Yagi, H. Billett, M. Saada, K. Simone, L. Stahl, J. Szymanski, D. Goldstein, M. R. Gil, et al. "Using automated machine learning to predict the mortality of patients with COVID-19: prediction model development study". In: *Journal of medical Internet research* 23.2 (2021), e23458.