ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ

Σχολή Χρηματοοικονομικής και Στατιστικής

Τμήμα Στατιστικής και Ασφαλιστικής Επιστήμης

ΜΕΤΑΠΤΥΧΙΑΚΟ ΠΡΟΓΡΑΜΜΑ ΣΠΟΥΔΩΝ
ΣΤΗΝ **ΕΦΑΡΜΟΣΜΕΝΗ ΣΤΑΤΙΣΤΙΚΗ**

ΕΦΑΡΜΟΓΗ ΜΕΘΟΔΩΝ ΣΤΑΤΙΣΤΙΚΗΣ ΜΗΧΑΝΙΚΗΣ ΜΑΘΗΣΗΣ ΣΤΗΝ ΑΝΑΛΥΣΗ ΚΕΙΜΕΝΩΝ ΚΑΙ ΕΙΚΟΝΩΝ ΥΓΕΙΑΣ

Ιωάννης Αρβανιτόπουλος

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

Μάρτιος 2023

Π Α Ν Ε Π Ι Σ Τ Η Μ Ι Ο Π Ε Ι Ρ Α Ι Ω Σ

Σχολή Χρηματοοικονομικής και Στατιστικής

Τμήμα Στατιστικής και Ασφαλιστικής Επιστήμης

ΜΕΤΑΠΤΥΧΙΑΚΟ ΠΡΟΓΡΑΜΜΑ ΣΠΟΥΔΩΝ
ΣΤΗΝ **ΕΦΑΡΜΟΣΜΕΝΗ ΣΤΑΤΙΣΤΙΚΗ**

ΕΦΑΡΜΟΓΗ ΜΕΘΟΔΩΝ ΣΤΑΤΙΣΤΙΚΗΣ ΜΗΧΑΝΙΚΗΣ ΜΑΘΗΣΗΣ ΣΤΗΝ ΑΝΑΛΥΣΗ ΚΕΙΜΕΝΩΝ ΚΑΙ ΕΙΚΟΝΩΝ ΥΓΕΙΑΣ

Ιωάννης Αρβανιτόπουλος

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

Μάρτιος 2023

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:

- Αναπληρωτής Καθηγητής Σωτήριος Μπερσίμης (Επιβλέπων)
- Καθηγητής Βασίλειος Πλαγιανάκος
- Επίκουρος Καθηγητής Σωτήριος Τασουλής

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

UNIVERSITY OF PIRAEUS

School of Finance and Statistics

Department of Statistics and Insurance Science

POSTGRADUATE PROGRAM IN
APPLIED STATISTICS

STATISTICAL MACHINE LEARNING USED IN HEALTH IMAGE AND TEXT ANALYSIS

By

Ioannis Arvanitopoulos

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

March 2023

Στους παππούδες μου

Σαργκίς & Καρινέ

Ευχαριστίες

Η παρούσα διπλωματική εργασία σηματοδοτεί την ολοκλήρωση των μεταπτυχιακών μου σπουδών με τίτλο «Εφαρμοσμένη Στατιστική» στο τμήμα Στατιστικής και Ασφαλιστικής Επιστήμης της σχολής Χρηματοοικονομικής και Στατικής του πανεπιστημίου Πειραιώς.

Νιώθω την ανάγκη να ευχαριστήσω θερμά τους καθηγητές και το διδακτικό προσωπικό του προγράμματος, καθώς μου μετέφεραν τις κατάλληλες γνώσεις για να μπορέσω να εκπονήσω την διπλωματική μου εργασία. Ειδικότερα θα ήθελα να εκφράσω ένα θερμό και ειλικρινές ευχαριστώ στον επιβλέποντα καθηγητή μου κ. Σωτήριο Μπερσίμη, αναπληρωτή καθηγητή του πανεπιστημίου Πειραιά, για την εμπιστοσύνη που μου έδειξε κατά την ανάθεση της συγκεκριμένης εργασίας, για την άρτια επιστημονική του καθοδήγηση καθώς και για την εξαιρετική συνεργασία που είχαμε καθ' όλο το διάστημα αυτό.

Ευχαριστώ πολύ την οικογένεια μου για την στήριξη και την ενθάρρυνση καθ' όλη τη διάρκεια των σπουδών μου. Θέλω ακόμα να ευχαριστήσω τους φίλους μου για την υποστήριξη στις θυσίες που έκανα.

Δεν θα μπορούσα να παραλείψω τη σύντροφο και αρραβωνιαστικιά μου, Αναστασία Νάτση, για την ιδιαίτερη συμβολή και στήριξη σε αυτή μου τη προσπάθεια.

Τέλος ευχαριστώ μέσα από την καρδιά μου τον άνθρωπο που ήταν "φάρος" σε όλο το ταξίδι της γνώσης και εκπαίδευσης μου, τον παππού μου Αρβανιτόπουλο Σαργκίς.

Πειραιάς, Μάρτιος 2023

Περίληψη

Η παρούσα διπλωματική εργασία στόχο έχει να καλύψει ένα μεγάλο μέρος των τεχνικών προ επεξεργασίας δεδομένων κειμένων και εικόνων στον τομέα της υγείας, με σκοπό τη βελτίωση της απόδοσης των αλγορίθμων κατηγοριοποίησης αυτών σε προκαθορισμένες κατηγορίες. Για την κατηγοριοποίηση των κειμένων δοκιμάστηκαν τόσο τεχνικές μηχανικής μάθησης όσο και τεχνικές βαθιάς μηχανικής μάθησης. Στις τεχνικές μηχανικές μάθησης χρησιμοποιήθηκαν οι αλγόριθμοι Logistic Regression, Multinomial Naïve Bayes και Support Vector Machines (SVM). Τα αποτελέσματα αυτών συγκρίθηκαν και ο αλγόριθμος με την καλύτερη απόδοση βρέθηκε να είναι ο SVM με ακρίβεια κατηγοριοποίησης 88.98% Στις τεχνικές βαθιάς μηχανικής μάθησης χρησιμοποιηθήκαν αρχιτεκτονικές νευρωνικών δικτύων όπως Multilayer Perceptron και Convolutional Neural Network (CNN). Τα αποτελέσματα, αφού συγκρίθηκαν οι μέθοδοι μεταξύ τους, έδειξαν πως καλύτερη κατηγοριοποίηση των κειμένων επιτυγχάνει το δίκτυο Multilayer Perceptron με ακρίβεια κατηγοριοποίησης 84.85%. Η κατηγοριοποίηση εικόνων δοκιμάστηκε να επιτευχθεί με CNNs και Transfer Learning. Συγκεκριμένα ένα απλό CNN και ένα προ εκπαιδευμένο CNN γνωστό ως VGG16 δοκιμάστηκαν με καλύτερη απόδοση το VGG16 που πέτυχε ακρίβεια κατηγοριοποίησης 97.83%

Abstract

The current thesis aims to cover a wide range of pre-processing techniques on text and image data on health sector, with the scope of improving the performance of classification algorithms. For text classification, Machine Learning algorithms such as Logistic Regression, Multinomial Naïve Bayes, and Support Vector Machines (SVM) were used and the results of them were compared to each other. The best performing algorithm was SVM with a classification accuracy of 88.98%. Deep Learning techniques were used as well, for the same task. A Multilayer Perceptron and a 1-Dimentional Convolutional Neural Network (1-D CNN) were trained and evaluated. The best performance was achieved from the Multilayer Perceptron with a classification accuracy of 84.85%. Image classification task was implemented using CNNs and Transfer Learning. Specifically, a simple CNN and a pre-trained CNN known as VGG16 were used and the best performing was the VGG16 with a classification accuracy of 97.83%.

Table of Contents

List of Figures i

List of Tables iii

1. Introduction 1

2. Text and image classification applications on healthcare 3

2.1. Applications on text classification 3

2.2. Applications on image classifications 7

3. Text Data Preprocessing steps and techniques 13

3.1. Tokenization 13

3.2. Stop Word Removal 13

3.3. Stemming 14

3.3.1. Porters Stemmer 14

3.3.2. N-Gram Stemmer (Statistical Methods) 15

3.3.3. Context Sensitive Stemmer (Mixed Methods) 15

3.4. Word Vectorization 17

3.4.1. Bag of Words 17

3.4.2. Term Frequency - Inverse Document Frequency Vectorization 18

3.4.3. Word Embeddings 19

4. Classification Algorithms 25

4.1. Multinomial Naive Bayes Classifier 25

4.2. Logistic Regression 26

4.3. Support Vector Machines (SVM) 28

4.3.1. Binary-Class SVM 28

4.3.2. Multi-Class SVM 29

4.4. Artificial Neural Networks 31

4.4.1. Perceptron ANN 34

4.4.2. Convolutional Neural Networks (CNN) 34

4.4.3. Recurrent Neural Networks (RNN) 37

4.4.4. Long Short-Term Memory (LSTM) 40

5. Clinical Text Analysis 43
5.1. Dataset 43
5.2. Data Preprocessing 45
5.3. Modeling 47
5.3.1. Initial Experiment 47
5.3.2. Dropping Categories Experiment 50
5.3.3. Dropping Categories and Feature Selection using $X2$ 56
5.3.4. Oversampling with SMOTE & Feature Selection 60
5.4. Deep Learning Approach 64
5.4.1. Deep Neural Network 65
5.4.2. Convolutional Neural Network 67
5.4.3. Convolutional Neural Network with Glove Embeddings 71
5.4.4. Text Generation for balancing minority categories 72
5.4.5. Deep Neural Network with oversampled data 77
5.4.6. Convolutional Neural Network with oversampled data 78
5.4.7. Summary 79
6. Chest X-rays Covid-19 Classification 81
6.1. Dataset 81
6.2. Data Exploration and Preprocessing 81
6.3. Deep Neural Network 83
6.4. VGG16 Network 84
6.5. Summary 86
7. Conclusion 88
8. Appendix 89
8.1. Medical Text Classification Code 89
8.2. Medical Text Classification (Deep Learning Approach) Code 95
8.3. Medical Image Classification Code 109
9. References 119

List of Figures

Figure 1 - Brain tumor types 8
Figure 2 - Roc Curve of the classification 8
Figure 3 - Confusion Matrix of Brain tumor classification 8
Figure 4 - Discriminative patch prediction 11
Figure 5 - Context - aware feature selection and aggregation 11
Figure 6 - Cosine Distance 19
Figure 7 - A simple CBOW model 20
Figure 8 - A Skip-Gram model 21
Figure 9 - Weighting function f with a =3/4. 24
Figure 10 - Sigmoid Function 26
Figure 11 - Linear Support Vector Machine (SVM) for a 2D data. 28
Figure 12 - One vs One Approach 30
Figure 13 - Three different classes scatter plot 30
Figure 14 - One vs All Approach 31
Figure 15 - human brain's neuron 31
Figure 16 - ANN Architecture 32
Figure 17 - Convolution Filter 35
Figure 18 - Max and Average Pooling 35
Figure 19 - Fully Connected layer 36
Figure 20 - General CNN architecture for text classification 37
Figure 21 - Recurrent Neural Network 38
Figure 22 - Types of RNNs 39
Figure 23 - Unrolled Recurrent Neural Network 39
Figure 24 - Forget Gate 40
Figure 25 - Input Gate 41
Figure 26 - Cell State Update 41
Figure 27 - Output Gate 42
Figure 28 - Distribution of medical transcription text lengths 43
Figure 29 - Counts of each category in target variable 44
Figure 30 - NaN Values for "transcription" column 44
Figure 31 - Counts of each category in target variable after reduction 45
Figure 32 - Most frequent occurring words 46
Figure 33 - Confusion Matrix of Logistic Regression (Dropped Categories experiment) 55
Figure 34 - Confusion Matrix of SVM classifier 63
Figure 35 - Final medical specialties counts 64
Figure 36 - DNN Architecture 65

Figure 37 - Confusion Matrix of DNN 66
Figure 38 - CNN Architecture 69
Figure 39 - Confusion Matrix of CNN 70
Figure 40 - Models performance over epochs 72
Figure 41 - LSTM model for text generation 75
Figure 42 - Text categories counts after text generation 76
Figure 43 - Confusion Matrix of DNN on generated texts 77
Figure 44 - All models performance over epochs 79
Figure 45 - Images of Normal and Covid-19 X-rays 82
Figure 46 - DNN Architecture 83
Figure 47 - Receiver Operating Characteristic Curve of DNN 84
Figure 48 - Confusion matrix of DNN 84
Figure 49 - VGG16 Architecture 84
Figure 50 - Confusion Matrix of VGG16 85
Figure 51 - Receiver Operating Characteristic Curve of VGG16 86
Figure 52 - Models performance over training 86
Figure 53 - Receiver Operating Characteristic of models 86

List of Tables

Table 1 - Image distribution in the BreakHis dataset 9
Table 2 - DenseNet Vs Popular CNNs in the multiclass breast cancer classification .. 10
Table 3 - Data description in the dataset 11
Table 4 - Vocabulary inside the corpus 18
Table 5 - Co-occurance Matrix 21
Table 6 - Data set columns 43
Table 7 - Classification report of NB 47
Table 8 - Classification report of Logistic Regression 48
Table 9 - Classification report of SVM 49
Table 10 - Summary of the "Initial Experiment" 50
Table 11 - Classification report of NB 52
Table 12 - Classification report of Logistic Regression 53
Table 13 - Classification report of SVM 54
Table 14 - Summary of the "Dropping Categories Experiment" 54
Table 15 - Classification report of NB 57
Table 16 - Classification report of Logistic Regression 58
Table 17 - Classification report of SVM 59
Table 18 - Summary of "Dropping categories and Feature Selection" experiment 59
Table 19 - Categories counts before and after SMOTE 60
Table 20 - Classification Report of Multinomial NB 61
Table 21 - Classification report of Logistic regression 61
Table 22 - Classification report of SVM 62
Table 23 - Summary of "Oversampling with SMOTE & Feature Selection" experiment 62
Table 24 - Before and After text cleaning 64
Table 25 - Classification report of DNN 66
Table 26 - Text encoded into sequence of integers 67
Table 27 - Padding sequences into 100 maximum length 68
Table 28 - Classification report of CNN 70
Table 29 -Classification report of CNN wih Glove Embeddings 71
Table 30 - Feature and Labels extraction from given text. 74
Table 31 - Classification report of DNN on generated texts 77
Table 32 - Classification report of CNN on generated texts 78
Table 33 - Summary of all DL experiments 79
Table 34 - Dataset Description 81
Table 35 - Final Dataset Description 81
Table 36 - Classification report of DNN 83

Table 37 - Classification report of VGG16 ..... *85*
Table 38 - Models Summary Table 86

1. Introduction

Text mining is the process of extracting interesting and non-trivial patterns or knowledge from unstructured text documents. It can be viewed as an extension of data mining or knowledge discovery from databases.

Text mining is believed to have a commercial potential higher than that of data mining. In fact, a recent study indicated that 80% of a company's information is contained in text documents. Text mining, in comparison with data mining, is a much more complex task as it involves dealing with text data that is inherently unstructured and fuzzy. Text mining is a multidisciplinary field, involving information retrieval (IR), text analysis, information extraction, clustering, categorization, visualization, database technology, machine learning, and data mining.

In healthcare, clinical records are largely maintained in free-text form. For that reason, a reliable and efficient method to extract structured information for future data mining from free text using information extraction techniques may greatly benefit research endeavors. As information technology and HIS (Hospital Information System) have been developed much, EMR (Electronic Medical Record) has also been popularized. EMR, which medical staff uses to record texts, symbols, charts, graphics, data, and other digital information generated by HIS, refers to medical records, which could be stored, managed, transmitted, and reproduced efficiently. With the tremendous growth of the adoption of EMR, various sources of clinical are becoming available, which has established EMR as an important factor for large-scale analysis of health data.

EMR also contains medical images such as MRI (Magnetic Resonance Imaging), X-rays, CT (Computed Tomography), sonograms etc. which are also unstructured data and can be used for the purpose of high-precision diagnosis. The needs for AI (Artificial Intelligence), in healthcare have established automated systems which scan medical images and configure big data. These data in combination with suitable techniques are used for early detection of some of the most harmful diseases such as Alzheimer's disease, brain tumors, breast, and lung cancer detection etc.

2. Text and image classification applications on healthcare

2.1. Applications on text classification

Data Analytics and Machine Learning (ML) have been increasingly considered as an enabling artefact to leverage health data for competitive advantage. The use of ML techniques and data analytics have been widely utilized to summarize, explain, and get insights into the interrelationships underlying complex datasets in novel ways. Such insights can play a positive role in various medical and operational aspects including diagnosis, health monitoring and assessment, healthcare planning, and management of hospitals and health services. However, one of the key challenges for healthcare analytics is to deal with huge data volumes in the form of unstructured text. Examples include nursing notes, clinical protocols, medical transcriptions, medical publications, and many others. In this respect, the use of Text Analytics has increasingly come into prominence to deliver benefits for health organizations in a wide range of applications.

In this section, the state-of-the-art approaches, and applications of Text Analytics in the healthcare context is going to be explored. The review is organized into two broad categories of Text Analytics. On one hand, the first part presents selective studies that applied Text Mining in the context of healthcare. On the other hand, the second part describes Text Analytics in a diversity of predictive applications to support clinical decision making. The review is unavoidably selective rather than exhaustive. However, it is believed that the study could adequately provide representative studies in each category.

Text Mining consists of two phases as follows. The initial phase typically includes the application of text refining procedures, which transform free-text documents into another intermediate form. Subsequently, the process of knowledge extraction, which attempts to learn patterns or insights from that intermediate form (Tan, 1999). This section provides selective studies that applied Text Mining with different modalities and for various purposes in the healthcare context.

Han et al. (2015), have presented a rule-based system for question retrieval. They aimed to search for similar questions in a large corpus of questions posted on online health forums. The system was mainly based on the RAKE algorithm (Rose et al. 2010) to perform the automatic extraction of keywords. Additional NLP methods were applied using the popular NLTK library.

Martínez et al. (2016), have exploited health-related online content into actionable knowledge using Text Mining. To reach this, they developed an approach to help monitor online user generated streams on social media. The transformed

information was extracted by an NLP-based processing pipeline which was applied on real-time streams of social media. The system could not only extract the mention of diseases and drugs, but also it could identify useful relationships among medications, indications, and adverse drug reactions.

Chang et al. (2016), have worked to develop a workflow using Text Mining to search, extract, and synthesize information about Comparative Effectiveness Research (CER) in healthcare. They developed a pipeline based on Natural Language Processing (NLP-based pipeline) to extract information from unstructured CER data sources. This solution could allow for the generation of timely alerts, and the collection of systematic reviews as well. They used trial data from multiple sources including ClinicalTrials.gov, WHO International Clinical Trials Registry Platform (ICTRP), and Cite line Trial trove.

Brown and Marotta, (2017), another interesting application set up, was intended to develop a set of classification models to predict the protocol and priority of magnetic resonance imaging (MRI) brain examinations. They used the narrative clinical information provided by clinicians. The models were trained to make predictions on three tasks including: i) Selection of examination protocols, ii) Evaluation of the need for contrast administration, and iii) Estimation of priority. The dataset consisted of about 14K MRI brain examinations over the period of January 2013 to June 2015. They created three models for each prediction task, each using a different classification algorithm (Random Forest, Support Vector machine and K-Nearest neighbor). They got an accuracy of 82,9%, 83% and 88,2% for each task. The empirical results largely demonstrated that the models could be effectively employed to assist the clinical decision support in this regard.

Castro et al. (2017), in the context of radiology, have developed a system to automate the annotation and classification of the Breast Imaging Reporting and Data System (BI - RADS) categories. They tried to develop an NLP system so it can automate BI-RADS categories extraction from breast radiology reports. Specifically, the system tackled two tasks including: w) Annotation of the BI-RADS categories, and ii) Classification of the laterality for each BI-RADS category. The study included 2K radiology reports collected from 18 hospitals of the University of Pittsburgh from 2003 to 2015.

Pendyala, and Figueira (2017), explored the potential of Text Mining for automating medical diagnosis. They applied the Bag-of-Words representation to medical documents. To simplify the text representation, the Bag-of-Words model builds a histogram of the words, while each word count is considered as a feature

(Goldberg, 2017). As such, each document can be simply represented as a “bag” of words, while disregarding the order, sequence, and grammar of text. Though using a small dataset, their experiments demonstrated promising results for that application. More recently, (van Dijk et al., 2020) applied Text Mining to EHR data to validate the screening eligibility of trial patients. The study was based on a multi-center, and multi-EHR systems as well. The accuracy of the Text-Ming approach was compared to the standard process produced by research personnel. The accuracy of the automatically extracted data was about 88.0%.

Jelodar et al. (2020), used Text Mining to extract the COVID-19 discussions from social media. They applied topic modeling of public opinions to gain insights into the various issues pertaining to the COVID-19 pandemic. In addition, they implemented an LSTM recurrent neural network for sentiment classification of COVID-19 comments. Their findings put light on the importance of using public opinions and the appropriate techniques to understand issues surrounding COVID-19. The model achieved an accuracy of 81,15% which was higher than that of other well-known machine-learning algorithms for COVID-19 sentiment classification.

Tvardik et al. (2018), developed a Text-Analytics solution for the automatic detection of medical events. The textual records included data collected from three University hospitals based in France over the period October 2009 to December 2010. The dataset spanned a variety of medical surgical specialties including neurosurgery, orthopedic surgery, and digestive surgery. The system performance was compared with standard methods. The overall sensitivity and specificity were about 84%. The study generally confirmed the feasibility of using NLP-based methods to automate the detection and monitoring of healthcare-associated events in hospital facilities.

Afzal et al. (2018), applied NLP for the automatic identification of Critical limb ischemia (CLI). Critical limb ischemia is a complication of advanced peripheral artery disease (PAD) with diagnosis based on the presence of clinical signs and symptoms. The dataset included narrative clinical notes retrieved from the HER (Electronic Health Record) database. They tried to extend a previous NLP algorithm for PAD by developing and validating a sub phenotyping NLP algorithm to identify the CLI cases from clinical notes. The model performance was validated compared to the human abstraction of clinical notes. Specifically, a physician reviewed and interpreted the information in the EHR data for each patient in the dataset. Overall, the method could achieve an excellent F1-score of about 90%.

Sterling et al. (2019), utilized the bag-of-words representation of nursing triage free-text notes which are the first text data created at the start of an emergency department (ED) visit. The study aimed to predict the final ED deposition using three NLP preprocessing techniques. Using a dataset of over 250K ED visits they defined the target variable as 1: admission, transfer, or death and 2: discharged, 'left without being seen' and 'left against medical advice'. Neural network regression models were trained to predict hospital admissions. They could achieve a promising accuracy with ROC-AUC≈0.74.

Ge et al. (2019), came through with another recent study developed a framework to realize scalable Text Analytics. The framework aimed to support real-time analytics for decision support in a variety of domains such as healthcare for example. Deep Learning was applied for NLP tasks including language understanding and sentiment analysis. The framework utilized a set of open-source tools including Spark Streaming for real-time text processing along with Zeppelin and Banana for data visualization. In addition, an LSTM model was trained for sentiment analysis. They practically demonstrated the functionality of the framework using a scenario with Twitter data.

Kidwai, and Nadesh (2020), discussed the application of diagnostic chatbots in healthcare. They developed a chatbot that makes use of NLP methods to understand the user queries. After collecting the initial symptoms, the chatbot would guide the user through a sequence of questions towards making the appropriate diagnosis. The system uses decision trees and follows a top-down approach to conclude the diagnosis. The chatbot was experimented with using a medical database of about 150 diseases.

Chen et al. (2020), managed to deal with the problem of overcrowding in emergency department (ED) which has serious issues and demands effective clinical decision-making of patient disposition. Their study included the development of a disposition prediction model using Deep Learning. They gathered approximately 105K ED visits during 2017-2018. The class to be predicted a deep neural network model was developed with word embedding. They aimed to compare ML models as they put the DNN against a Logistic Regression model with structured data. The metric used to measure the predictive performance in both cases was F1 score.

Arnaud et al. (2020), presented an approach based on integrating structured data with unstructured textual notes recorded at the triage stage. The key idea was to apply a multi-input of mixed data for training a classification model to predict hospitalization. On one hand, a standard Multi-Layer Perceptron (MLP) model was used with the standard set of features (i.e., numeric, and categorical). On the other hand, a Convolutional Neural Network (CNN) was used to operate over textual data. Their

empirical results demonstrated that the classifier could achieve a very good accuracy with ROC-AUC≈0.83.

2.2. Applications on image classifications

Image classification is the ability to assign a label to a given image of any size. This task belongs to the subset of artificial intelligence that is well known as computer vision. Computer vision is widely used nowadays in many different industries with very interesting and useful applications, one of those is healthcare.

Modern hospitals hold a diverse range of imaging data for diagnosis, treatment planning, and assessing response to treatment. There are many cases where the human eye is not possible to detect organ damage or a type of disease that is at its earliest stage. Medical image analysis aims to extract the most important features in health-related images to improve clinical diagnosis. In all medical specialties there are plenty of image processing and classification applications with most of them related to early cancer detection.

Ali Ari and Davut Hanbay (2018), developed a system for brain tumor detection and segmentation. They proposed a method which had three stages. At the first stage nonlocal means and local smoothing methods were used to remove possible noises. In the second stage, cranial magnetic resonance images were classified as benign or malignant using extreme learning machine local receptive fields (ELM-LRF). In the third stage the detected tumors were segmented. The dataset they have used was comprised of sixteen patients' images digitized at 256x256 pixels. The training set had 9 images while the testing was 7. All input images were resized to 32x32 before feeding into the ELM-LRF. The ELM-RF had four tunable parameters: convolution filter size r, convolution filter number K, pooling size, and C regulation coefficient. The values of the parameters were set to 5 to be the convolution filter size, the K value was chosen as 16 and the pooling size was chosen as 3. In addition, for identifying the most appropriate C value, an interval search was performed among $2^{-10}, 2^{-8}, \dots, 2^{8}, 2^{10}$ and the C value with the minimum fault was chosen. In the experimental studies the classification accuracy of cranial MR images is 97.18%. Evaluated results showed that the proposed method's performance was better than the other recent studies in literature. Experimental results also proved that the proposed method is effective and can be used in computer aided brain tumor detection.

Tariq Sadad et al. (2020), developed a model for tumor detection and classification. The dataset that they used was from Figshare source which contained 3,064 brain MRI images of size 512x512 pixels. These images were obtained from 233 patients. It contains three brain tumors: glioma, pituitary, meningioma tumor and three distinct views: sagittal, axial, coronal views.

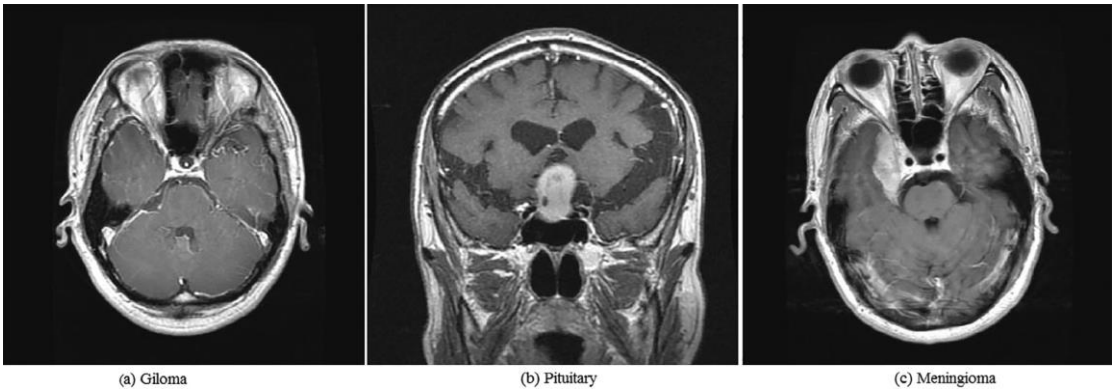


Figure 1 - Brain tumor types

For phase one, brain tumor detection, a U-Net architecture was employed and achieved a remarkable efficiency in detecting tumors in medical images. The backbone that was applied is an ResNet50 which is comprised of an encoder and decoder. For phase two, brain tumor classification, transfer learning and NASNet architecture were used. NASNet comprises of CNN and Controller Recurrent Neural Network (CRNN). They managed to achieve the highest IoU score of 0.9504 for brain tumor detection and the highest accuracy of 99.6% for the detected tumor classification.

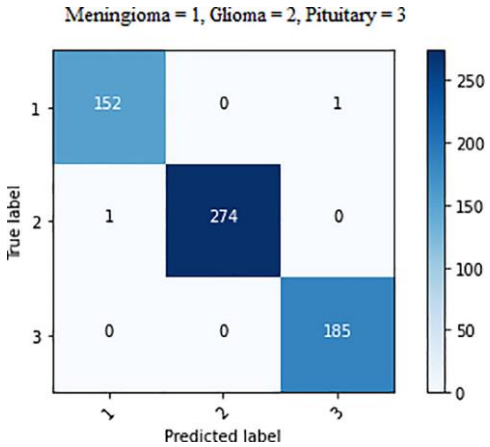


Figure 3 - Confusion Matrix of Brain tumor classification

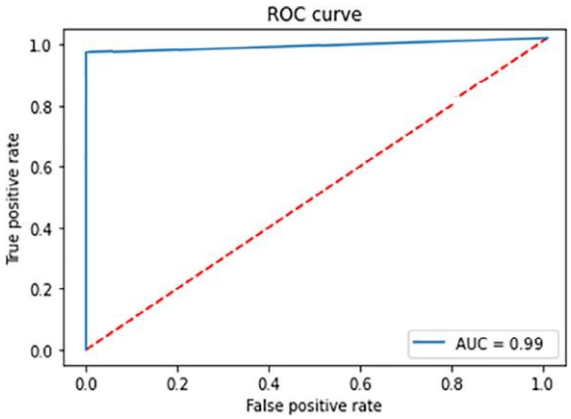


Figure 2 - Roc Curve of the

Majid Nawaz et al. (2018), in the context of breast tumor detection, presented a deep learning approach that was based on a Convolutional Neural Network (CNN) model for multiclass breast cancer classification. They aimed not only to classify the breast tumors in non-benign or malignant, but they proposed an approach that was able to predict the subclass of the tumors like Fibroadenoma, Lobular carcinoma, etc. The dataset used for training and testing is BreakHis dataset which contains images collected through a clinical study from January 2014 to December 2014 that took place in Brazil. The dataset contains 7909 colored microscopic biopsy images of benign and malignant breast tumors in four magnifying factors.

Magnification	Benign	Malignant	Total
40X	652	1370	1995
100X	644	1437	2081
200X	623	1390	2013
400X	588	1232	1820
Total of Images	2480	5429	7909

Table 1 - Image distribution in the BreakHis dataset

The proposed model is a convolutional neural network where the convolution non-linear and pooling layers were replaced with dense blocks and transition layers. This kind of CNN is known as DenseNet model. The DenseNet has three dense blocks, and two transition layers. They used 7x7 kernels for the first convolution to detect small variation and substance in the image and extract more important features. An average 7x7 kernel size pooling layer with stride 2 is used before the fully connected layer and finally the softmax layer for the eight classes of breast cancer histopathological images was configured. The weights of different layers were initialized by using a pre-trained model on ImageNet. The last layer was fine-tuned on BreakHis cancer images dataset. The first convolutional layer was then unfrozen, and the entire network was fine-tuned on the BreakHis training data. The results of the model were reported in two ways: In patient level and in image level. In patient let, N_p be the number of histopathological images of patient P. For each patient, if N cancer images are correctly classified, the patient score and the patient recognition rate are defined as:

$$Patient\ score = \frac{N}{N_p} \tag{1}$$

$$Patient\ recognition\ rate = \frac{\sum patient\ Score}{total\ patient\ number} \tag{1}$$

In image level reporting the recognition rate is calculated in image level. Let, N_t be the number of histopathological images inside the testing set. If N_r, cancer images are correctly classified then the recognition rate in image level is:

$$Image\ recognition\ rate = \frac{N_r}{N_t} \tag{3}$$

Accuracy (%)	Model	Magnification Factors				
		40x	*100x*	*200x*	*400x*	*average*
Image level	*LeNet [43]*	46.4	47.34	46.5	45.2	46.36
	AlexNet [28]	86.4	75.8	72.6	84.6	79.85
	CSDCNN[30]	92.8	93.9	93.4	92.9	93.25
	DenseNet (our's)	93.64	97.42	95.87	94.67	95.4
Patient level	*LeNet [43]*	48.2	47.6	45.5	45.2	46.62
	AlexNet [28]	74.6	73.8	76.4	79.2	76
	CSDCNN[30]	94.1	93.2	94.7	93.5	93.87
	DenseNet (our's)	94.23	97.86	96.35	95.24	96.48

Table 2 - DenseNet Vs Popular CNNs in the multiclass breast cancer classification

In image level the DenseNet CNN model achieved high performance with 95.4% accuracy in the multiclass breast cancer classification task when compared with other state-of-the-art models.

Xi Wang, et al. (2018), proposed an approach for fast and effective classification on whole slide lung cancer images. Their method takes advantage of a patch-based fully convolutional network for discriminative block retrieval. The network architecture they proposed was split into three parts. The first part is a patch-based CNN that aims to predict the cancer likelihood from whole slide images, the second part the spatially contextual information is considered when selecting features from these blocks and the

third part the global feature descriptor is fed into a standard Random Forest for the final whole slide image classification.

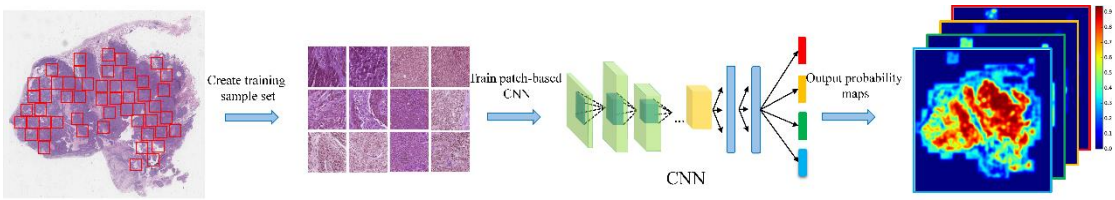


Figure 4 - Discriminative patch prediction

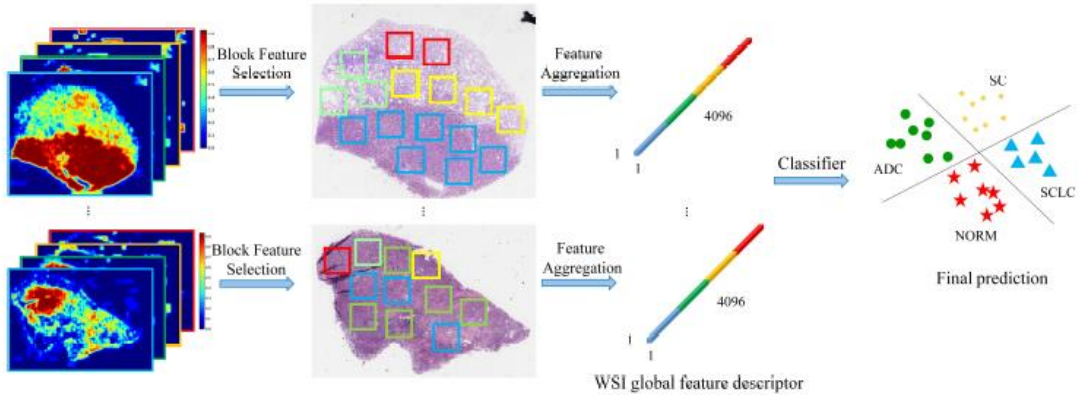


Figure 5 - Context - aware feature selection and aggregation

The data used for the experiments were provided by Sun Yat-Sen University Cancer Center. The dataset consists of 871 digitalized histology WSIs with lung carcinomas and 68 healthy WSIs. The 871 WSIs are split into three subcategories of lung cancer, Squamous cell Carcinoma (SC), Adenocarcinoma (ADC) or Small Cell Lung Carcinoma (SCLC). Inside this set, 59 images are commented by a group of experienced pathologists. These 59 images comprise as D1 dataset. The rest 812 cancer images are further split into 642 (D2) and 170 (D4) images for training and testing accordingly. The normal images are also split into two datasets containing 53 (D3) and 15 (D4) images for training and testing respectively. The table below shows the split of the dataset they had done.

		Carcinoma	***SC***	***ADC***	***SCLC***	***Normal***
Training	D1	59	21	20	18	-
	D2	642	267	293	82	-
	D3	-	-	-	-	53
Testing	D4	170	73	77	20	15

Table 3 - Data description in the dataset

With these data as shown above they set 4 experiments, M1 where D1 and D3 were used for patch-based CNN training. All the patches extracted from D1 and D3 only convey the WSI-level labels. During inference, CNN scanned the training set and output the patch-wise probabilities as well as the features from the last convolutional layer. The M2 experiment is like M1 except that the weighted loss function is employed during training. In M# experiment the training dataset was constructed from D1, D2 and D3 and the coarse annotation masks were not utilized during CNN training. Finally, in M4 experiment where the training dataset was the same as in M3 with the difference that the weighted loss function was utilized on coarse annotation regions. After the extensive experiments, the proposed method achieved an accuracy score of 97.1%

3. Text Data Preprocessing steps and techniques

Text pre-processing is an essential part of any Text Mining and NLP system, since the characters, words, and sentences identified at this stage are the fundamental units passed to all further processing stages, from analysis and tagging components, such as morphological analyzers and part-of-speech taggers, through applications, such as information retrieval and machine translation systems. It is a collection of activities in which text documents are pre-processed. Because the text data often contains some special formats like number formats, date formats and the most common words that are unlikely to help Text mining such as prepositions, articles, and pro-nouns can be eliminated. Text data pre-processing is needed to reduce indexing or data file size of the text documents as well as to improve the efficiency of the IR system. The 20%-30% of total word counts in a particular document is composed of stop words that do not add any value. Words of the same root or meaning, in the natural human language, are presented in different forms with different suffixes which affect the total number of significant words inside a document by presenting the same word with different tokens.

3.1. Tokenization

Tokenization is the process of breaking a stream of text into words, phrases, symbols, or other meaningful elements called tokens. The aim of the tokenization is the exploration of the words in a sentence. The list of tokens becomes input for further processing such as parsing or text mining. Tokenization is useful both in linguistics (where it is a form of text segmentation), and in computer science, where it forms part of lexical analysis. Textual data is only a block of characters at the beginning. All processes in information retrieval require the words of the data set. Hence, the requirement for a parser is the tokenization of documents. This may sound trivial as the text is already stored in machine-readable formats. Nevertheless, some problems are still left, like the removal of punctuation marks. Other characters like brackets, hyphens, etc. require processing as well. Furthermore, tokenizer can cater for consistency in the documents. The main use of tokenization is identifying meaningful keywords. The inconsistency can be different number and time formats. Another problem is abbreviations and acronyms which must be transformed into a standard form.

3.2. Stop Word Removal

Many words in documents recur very frequently but are meaningless as they are used to join words together in a sentence. It is commonly understood that stop words do not contribute to the context or content of textual documents. Due to their high

frequency of occurrence, their presence in text mining presents an obstacle in understanding the content of the documents.

Stop words are very frequently used common words like ‘and’ ‘are’, ‘this’ etc. They are not useful in classification of documents. So, they must be removed. However, the development of such stop words list is difficult and inconsistent between textual sources. This process also reduces the text data and improves the system performance. Every text document deal with these words which are not necessary for text mining applications.

3.3. Stemming

Stemming is the process of conflating the variant forms of a word into a common representation, the stem. For example, the words: “presentation”, “presented”, “presenting” could all be reduced to a common representation “present”. This is a widely used procedure in text processing for information retrieval (IR) based on the assumption that posing a query with the term presenting implies an interest in documents containing the words presentation and presented. Although there are some errors when it comes to stemming. There are mainly two errors in stemming, over stemming and stemming.

Over-stemming is when two words with different stems are stemmed to the same root. This is also known as a false positive. Under-stemming is when two words that should be stemmed to the same root are not. This is also known as a false negative.

Usually, stemming algorithms can be classified into three groups: **truncating methods**, **statistical methods**, and **mixed methods**. Each of these groups has a typical way of finding the stems of the word variants. Some of the stemming algorithms are going to be analyzed below.

3.3.1. Porters Stemmer

Porters stemming algorithm is one of the most popular stemming algorithms proposed in 1980. Many modifications and enhancements have been made and suggested on the basic algorithm. It is based on the idea that the suffixes in the English language (approximately 1200) are mostly made up of grouping of smaller and simpler suffixes. It has five steps, and within each step, rules are applied until one of them passes the conditions. If a rule is accepted, the suffix is removed consequently, and the next step is performed. The resultant stem at the end of the fifth step is returned.

The rule looks like the following:

<condition> <suffix> → <new suffix>

For example, a rule (m>0) EED → EE means "if the word has at least one vowel and consonant plus EED ending, change the ending to EE". So "agreed" becomes "agree" while "feed" remains unchanged. This algorithm has about 60 rules and very easy to understand. Porter designed a detailed framework of stemming which is known as "Snowball". The main purpose of the framework is to allow programmers to develop their own stemmers for other character sets or languages.

3.3.2. N-Gram Stemmer (Statistical Methods)

It is language independent stemmer. The string-similarity approach is used to convert word inflation to its stem. N-gram is a string of n, usually adjacent, characters extracted from a section of continuous text. N-gram is a set of n following characters extracted from a word. The main idea behind this approach is, similar words will have a high quantity of n-grams in common. For n equals to 2 or 3, the words extracted are called diagrams or trigrams, respectively.

For example, the word "INTRODUCTIONS" results in the generation of the diagrams:

***I, IN, NT, TR, RO, OD, DU, UC, CT, TI, IO, ON, NS, S* and the trigrams:**

****I, *IN, INT, NTR, TRO, ROD, ODU, DUC, UCT, CTI, TIO, ION, ONS, NS*, S****

Where '*' denotes a padding space. There is n+1 such diagram and n+2 such trigrams in a word containing n characters. Most stemmers are language specific. Usually, a value of 4 or 5 is selected for n. After that textual data or document is analyzed for all the n-grams. A word root generally occurs less frequently than its morphological form. This means a word generally has an affix associated with it. This stemmer has an advantage that it is language independent and hence very useful in many applications. The disadvantage is it requires huge memory and storage for creating and storing the n grams and indexes and hence it is not a practical approach.

3.3.3. Context Sensitive Stemmer (Mixed Methods)

This is a remarkably interesting method of stemming unlike the usual method where stemming is done before indexing a document, over here for a Web Search, context sensitive analysis is done using statistical modeling on the query side. This method was proposed by Funchun Peng et al. (2007).

Basically, for the words of the input query, the morphological variants which would be useful for the search are predicted before the query is submitted to the search engine. This severely reduces the number of bad expansions, which in turn reduces the cost of additional computation and improves the precision at the same time. After the predicted word variants of the query have been derived, a context sensitive document matching is done for these variants. This conservative strategy serves as a safeguard against spurious stemming, and it turns out to be very important for improving precision. This stemming process is divided into four steps after the query is fired:

a. Candidate generation:

Over here the Porter stemmer is used to generate the stems from the query words. This has completely no relation to the semantics of the words. For a better output the corpus-based analysis based on distributional similarity is used. The foundation of using distributional word similarity is that true variants tend to be used in similar contexts. In the distributional word similarity calculation, each word is represented by a vector of features derived from the context of the word. We use the bigrams to the left and right of the word as its context features, by mining a huge Web corpus. The similarity between two words is the cosine similarity between the two corresponding feature vectors.

b. Query Segmentation and head word detection:

When the queries are long, it is important to detect the major concept of the query. The query is broken into segments which are normally noun phrases. For each noun phrase the most important word is detected which is the head word. Sometimes a word is split to know the content. The mutual information of two adjacent words is found and if it passes a threshold value, they are kept in the same segment. Finding the headword is by using a syntactical parser.

c. Context sensitive word expansion:

The keywords words are obtained by using probability measures and it decided which word variants would be most useful – generally they are the plural forms of the words. This is done using the simplest and most successful approach to language modeling, which is the one based on the n-gram model which uses the chain rule of probability. In this step all the important headword variants are obtained. The traditional way of using stemming for Web search, is referred to as the naïve model. This is to treat every word variant equivalent for all possible words in the query. The query "bookstore" will be transformed into "(book OR books) (store OR stores)" when limiting stemming to pluralization handling only, where OR is an operator that denotes the equivalence of the left and right arguments.

d. Context sensitive document matching:

The context is the left or the right non-stop segments of the original word. Since queries and documents may not represent the intent in the same way, this proximity constraint is to allow variant occurrences within a window of some fixed size. The smaller the window size is, the more restrictive the matching. The advantage of this stemmer is it improves selective word expansion on the query side and conservative word occurrence matching on the document side. The disadvantage is the processing time and the complex nature of the stemmer. There can be errors in finding the noun phrases in the query and the nearest words.

3.4. Word Vectorization

For as long as computers have existed, there has been the question of how to represent data in a way that machines can understand and work with. A problem with modeling text is that it is messy, and techniques like machine learning algorithms prefer well defined fixed-length inputs and outputs. Machine learning algorithms cannot work with raw text directly; the text must be converted into numbers. Specifically, vectors of numbers. Below three of the most used techniques namely Bag of Words, tf-idf vectorization and word embedding are presented to convert text into numeric feature vectors.

3.4.1. Bag of Words

When dealing with text data it is necessary to understand terms like *Document*, *Corpus* and *Feature*. A document is a single text data point e.g., a medical prescription. The corpus is the collection of all the documents and the features are every unique word inside the corpus.

A bag of words (BoW) is the text representation that describes the occurrence of words inside a document. This representation is product of two things, a vocabulary of known words and a measure of the presence of known words. For example, let's say that 2 documents exist as below:

Doc1: *"I like to watch football despite my wife's disagreement."*

Doc2: *"My wife does not enjoy the time I watch football."*

The corpus can be built by combining the above three documents:
Corpus = "*I like to watch football despite my wife's disagreement. My wife does not enjoy the time I watch football."*

And features will be all unique words: *{"I", "like", "to", "watch", "football", "despite", "My", "wife", "disagreement", "does", "not", "enjoy", "the", "time"}*.

In *table 4* is the vocabulary of 14 words out of 19 inside the corpus.

	I	like	to	watch	football	despite	my	wife	disagreement	does	not	enjoy	the	time
my	0	0	0	0	0	0	**1**	0	0	0	0	0	0	0
wife	0	0	0	0	0	0	0	**1**	0	0	0	0	0	0
does	0	0	0	0	0	0	0	0	0	**1**	0	0	0	0
not	0	0	0	0	0	0	0	0	0	0	**1**	0	0	0
enjoy	0	0	0	0	0	0	0	0	0	0	0	**1**	0	0
the	0	0	0	0	0	0	0	0	0	0	0	0	**1**	0
time	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**
I	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0
watch	0	0	0	**1**	0	0	0	0	0	0	0	0	0	0
football	0	0	0	0	**1**	0	0	0	0	0	0	0	0	0
Bow for Document1	**1**	0	0	**1**	**1**	0	**1**	**1**	0	**1**	**1**	**1**	**1**	**1**

Table 4 - Vocabulary inside the corpus

Hence, Doc2 would look as follows as a binary vector:

$$doc2 = [1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1,1]$$

3.4.2. Term Frequency - Inverse Document Frequency Vectorization

The Bag-of words is simple and works properly, but its main disadvantage is that it treats all words equally and cannot distinguish quite common words or rare words. Term Frequency – Inverse Document Frequency (Tf-idf) solves this problem.

Term Frequency–Inverse Document Frequency (tf-idf) is a numerical statistic which reveals how important a word is to a document in a collection. The Tf - IDF is often used as a weighting factor in information retrieval and text mining. The value of tf-idf increases proportionally to the number of times a word appears in the document but is counteracting by the frequency of the word in the corpus. This can help to control the fact that some words are generally more common than others. Tf–IDF can be successfully used for stop-words filtering in various subject fields including text summarization and classification. Tf-IDF is the product of two statistics which are termed frequency and inverse document frequency. To further distinguish them, the number of times each term occurs in each document is counted and sums them all together. Term Frequency (TF) is defined as the number of times a term occurs in a document.

$$Tf(t,d) = \frac{No.of\ times\ a\ word\ appears\ in\ a\ document}{Total\ No.of\ words\ in\ a\ document}$$

An Inverse Document Frequency (IDF) is a statistical weight used for measuring the importance of a term in a text document collection. IDF feature is incorporated which reduces the weight of terms that occur very frequently in the document set and increases the weight of terms that occur rarely.

$$IDF(t,d) = log\frac{No.of\ total\ documents}{no.of\ doc.,term\ t\ appears\ into}$$

The more files a word appears in the smaller IDF.

Then TF-IDF is calculated for each word using the formula,

$$Tf - idf(t,d) = tf(t,d) * idf(t,d)$$

3.4.3. Word Embeddings

Both Vectorization techniques, BoW and tf-idf work well but it fails to suggest a relation between two words. E.g., king and queen are two related words, but these methods fail to recreate that relation in Vectorization. Vectorization using word embedding solves this problem.

Word embedding or word vector is an approach in NLP to map words or phrases from a vocabulary to a corresponding vector of real numbers. It is defined as a type of word representation that allows words with similar meaning to have a similar representation. The process of converting words into numbers is called vectorization and after vectorization it is needed a technique to identify similar words; such technique is the Cosine Similarity.

Cosine similarity measures the cosine of the angle between two vectors (item1, item2) projected in an N-dimensional vector space. The smaller the angle, the higher the similarity. It is defined as follows:

$$sim(item1, item2) = cos\theta = \frac{A \cdot B}{||A|| \cdot ||B||}$$

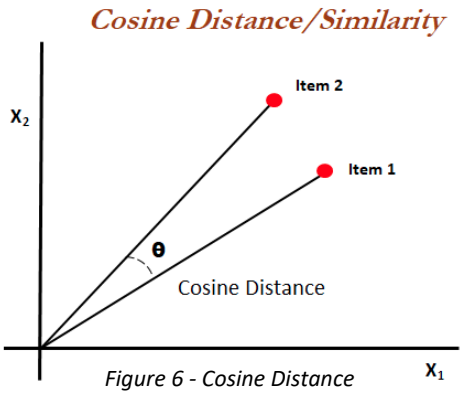


Figure 6 - Cosine Distance

These embeddings can be learned, or precomputed embeddings can be used. Some of the most used embeddings are:

- Word2Vec
- Glove

3.4.3.1. Word2Vec

Word2Vec is a statistical method for efficiently learning a standalone word embedding from a text corpus. It was developed by Tomas Mikolov, et al. (2013), at Google as a response to make the neural-network-based training of the embedding more efficient and since then has become the de facto standard for developing pre-trained word embedding. Two different learning models were introduced that can be used as part of the word2vec approach to learn the word embedding; they are:

- Continuous Bag-of-Words, or CBOW model
- Continuous Skip-Gram Model

CBOW model takes the context of each word as the input and tries o predict the word corresponding to the context. More specifically, one hot encoding is used for the input word and measures the output error compared to one hot encoding of the target word. In the process of predicting the target word, the vector representation of the target word is learned.

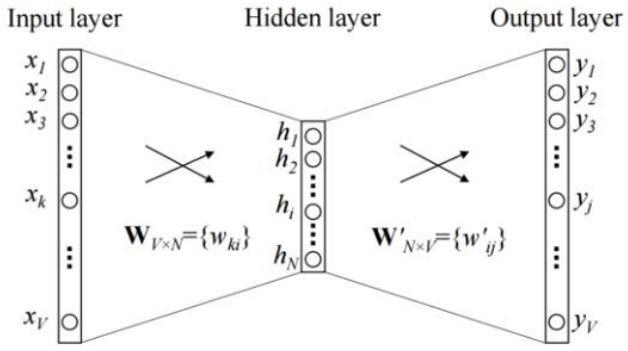


Figure 7 - A simple CBOW model

The input word is a one hot encoded vector of size V. The hidden layer contains N neurons, and the output is again a V length vector with the elements being the softmax values. $W_{V\times N}$ is the weight matrix that maps the input x to the hidden layer while $W'_{N\times V}$ is the weight matrix that maps the hidden layer outputs to the final layer. There is no activation like sigmoid, tanh or ReLU. The only non-linearity is the softmax calculations in the output layer.

Skip-gram is a slightly different word embedding technique in comparison to CBOW as it does not predict the current word based on the context. Instead, each current word is used as an input to a log-linear classifier along with a continuous projection layer. This way, it predicts words in a certain range before and after the current word. This variant takes only one word as an input and then predicts the closely related context words. That is the reason it can efficiently represent rare words.

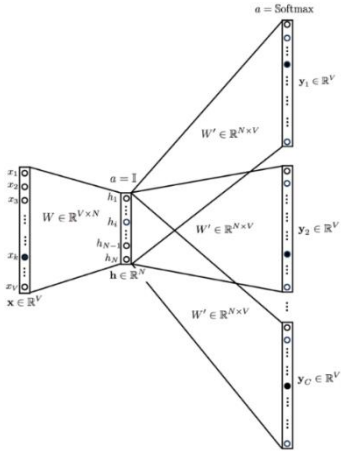


Figure 8 - A Skip-Gram model

3.4.3.2. Glove

The Global Vectors for Word Representation, or GloVe, algorithm is an extension to the word2vec method for efficiently learning word vectors, developed by Pennington, et al. at Stanford. It is an unsupervised learning algorithm which aims to generate word embeddings by aggregating global word co-occurrence matrices from a given corpus. The basic idea behind the GloVe word embedding is to derive the relationship between the words from statistics. Unlike the occurrence matrix, the co-occurrence matrix presents how often a particular word pair occurs together. Each value in the co-occurrence matrix represents a pair of words occurring together. GloVe focuses on global context to create word embeddings.

Let's say a corpus of one document that says, "The quick brown fox jumps over the lazy dog". Then the co-occurrence matrix window size of1 would be as follows:

	brown	**dog**	**fox**	**jumps**	**lazy**	**over**	**quick**	**the**
brown	0	0	1	0	0	0	1	0
dog	0	0	0	0	1	0	0	0
fox	1	0	0	1	0	0	0	0
umps	0	0	1	0	0	1	0	0
lazy	0	1	0	0	0	0	0	1
over	0	0	0	1	0	0	0	1
quick	1	0	0	0	0	0	0	1
the	0	0	0	0	1	1	1	0

Table 5 - Co-occurance Matrix

Let X be the matrix of word-word co-occurrence counts with entries X_{ij}. X_{ij} is the number of times word j occurs in the context of word i. Moreover, $X_i = \sum_k X_{ik}$ is the number of times any word appears in the context of word i. Finally let $P_{ij} = P(j|i) = \frac{X_{ij}}{X_i}$ be the probability that word j appears in the context of word i.

The idea of GloVe is to convert X into feature matrices W whose rows are populated by w_i or w_j, where $w_i, w_j \in \mathbb{R}^d$ are the feature vectors of word i and j. Also into matrix V whose rows are populated by v_k, where $v_k \in \mathbb{R}^d$ is a separate context feature vector of word k.

The model in the most general form is:

$$F(w_i, w_j, v_k) = \frac{P_{ik}}{P_{jk}} \quad (1)$$

Where F relates W and V to X. If word k is related to both word i and j which means that P_{ik} and P_{jk} are large, or unrelated to both word i and j which means that P_{ik} and P_{jk} are small, then the value of F would be close to 1. On the other hand, if word k is related to exactly one of the words i and k, which means that one of P_{ik} andP_{jk} is small and the other is large, then the value of F would be far from 1. As each model, GloVe needs a Cost Function to be defined. Since vector spaces are inherently linear structures, the most natural way to encode the information present in $\frac{P_{ik}}{P_{jk}}$ is with vector differences. Hence, you could restrict F to be:

$$F(w_i - w_j, v_k) = \frac{P_{ik}}{P_{jk}} \quad (2)$$

To maintain the linear structure of the vectors, F will be restricted one more time such that it now receives the dot product of its arguments,

$$F((w_i - w_j)^T v_k) = \frac{P_{ik}}{P_{jk}} \quad (3)$$

There is nothing special about i and j, they point to arbitrary words in the corpus. Hence, the role of i and j in equation can be flipped to obtain:

$$F((w_j - w_i)^T v_k) = \frac{P_{jk}}{P_{ik}} = \frac{1}{F((w_i - w_j)^T v_k)} \quad (4)$$

The model should be invariant under this relabeling, but equation (2) is not. To achieve invariance, F is assumed to be a **group homomorphism** from $(\mathbb{R}, +)$ to $(\mathbb{R}, \times)$. Hence,

$$\frac{P_{ik}}{P_{jk}} = F\left(\left(w_i - w_j\right)^T v_k\right)$$

$$= F\left(w_i^T v_k + \left(-w_j^T v_k\right)\right)$$

$$= F(w_i^T v_k) \times F\left(-w_j^T v_k\right)$$

$$= \frac{F(w_I^T v_k)}{F\left(w_j^T v_k\right)} \quad (5)$$

Where $F(w_i^T v_k) = P_{ik} = \frac{X_{ik}}{X_i}$

The solution of equation (5) is:

$$w_i^T v_k = \log(P_{ik}) = \log(X_{ik}) - \log(X_i)$$

Since the term $\log(X_i)$ does not depend on k, it could be considered as a bias term b_i for w_i. To restore exchange symmetry, a bias a_k for v_k will be also added. The final model becomes:

$$w_i^T + b_i + a_K = \log(X_{ik}) \quad (7)$$

The model in part (7) is called the log-bilinear regression model. The optimal values for parameters w_i, v_k, b_i and a_k can be found using the least squares method.

A weighting function $f(X_{ik})$ is introduced to compensate whenever $X_{ik} = 0$ since $log(X_{ik})$ is undefined at that point, and to balance out the contribution of frequent and infrequent words to the model. The cost function of the model becomes:

$$J = \sum_{i,k=1}^{vocabulary\ size} f(X_{ik})(w_i^T v_k + b_i + a_k - logX_{ik})^2$$

The weighting function should obey the following properties:

1. $f(0) = 0$. If f is continuous function, it should vanish as $x \to 0$ fast enough that the $\lim_{x \to 0} f(x) \log^2 x$ is finite.

2. $f(x)$ should be non-decreasing so that rare co-occurrences are not overweighted.
3. $f(x)$ should be relatively small for large values of x , so that frequent co-occurrences are not overweighted.

   One good simple choice of f is:

$$f(x) = \begin{cases} \left(\frac{x}{x_{max}}\right)^a & if\ x < x_{max} \\ 1 & otherwise \end{cases}$$

   Where x_{max} and a are hyperparameters.

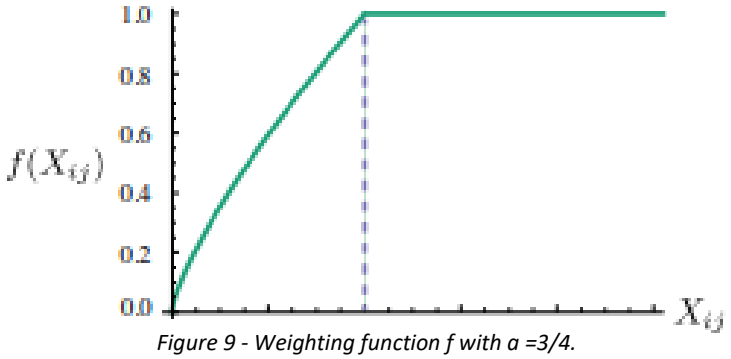


Figure 9 - Weighting function f with a =3/4.

The performance of the model depends weakly on the cutoff, which is fixed to $x_{max} = 100$ for the experiments. An $a = \frac{3}{4}$ gives a modest improvement over a linear version with $a = 1$. Although only empirical motivation is offered for choosing the value $a\ at \frac{3}{4}$.

4. Classification Algorithms

In this chapter the theory behind various ML and Deep Learning (DL) algorithms will be presented in the context of text classification.

4.1. Multinomial Naive Bayes Classifier

Multinomial NB is a probabilistic classifier which means that it is designed to use an implicit mixture model for the generation of the documents. That is the reason these kinds of classifiers are called generative classifiers. The naïve Bayes classifier is the simplest and most used generative classifier. Multinomial Naïve Bayes algorithm is the statistical classification algorithm which is based on the Bayes' theorem and helps us find the conditional probability of happening of two events based on the probabilities of happening of each individual event. Multinomial Naïve Bayes classifier works on the concept of term frequency that means the number of times the word occurs within a document.

The Multinomial model aims at determining the term frequency i.e., the number of times a term occurs in a document. The multinomial Naïve Bayes Classifier can be formulated as follows:

Assuming that there is a document 'd'. The probability of being a document of class c, $c \in \{1, \dots, C\}$, is calculated as:

$$P(c|d) = argmax\, P(c) \prod_{k=1}^{n_d} P(t_k|c) \quad (1)$$

n_d: *represents the number of tokens in a document*

t_k: *represents the* k^{th} *token in the document*

where $P(t_k|c)$ represents the conditional probability that whether the term t_k occurs in a document of class 'c', and it is calculated as follows:

$$P(t_k|c) = \frac{count(t_k|c) + 1}{count(t_c) + |V|} \quad (2)$$

where $count(t_k|c)$ means the number of times the word t_k is used in the documents among class c and $count(t_c)$ means the total number of tokens present in

the documents of class c. Furthermore 1 and $|V|$ are added as smoothing constants which are added to avoid setbacks in the calculation when the term does not appear at all in the document, or the documents is empty or null. $|V|$ is the number of tokens inside the total vocabulary of documents.

$P(c)$: is the prior probability of document being of class c which is calculated as taken after:

$$P(c) = \frac{Number\ of\ documents\ of\ class\ c}{Total\ number\ of\ documents} \quad (3)$$

The probability $P(c|d)$ is calculated for all the classes and the maximum of it, is the predicted class for a document.

4.2. Logistic Regression

Logistic Regression is a probability-based machine learning algorithm which is used for classification problems.

Logistic Regression can be compared with Linear Regression model with the difference that logistic regression uses a more complex cost function, which is known as "Sigmoid function". This function is a mathematical function which is used to map the predictive values to probabilities. The sigmoid function is:

$$g(y) = \frac{1}{1 + e^{-y}}$$

It always gives a value of probability ranging from $0 \leq p \leq 1$. The function can take any real number and map it into a value between 0 and 1. For example, if the output of the sigmoid function is more than 0.5 then the data point is classified as 1.

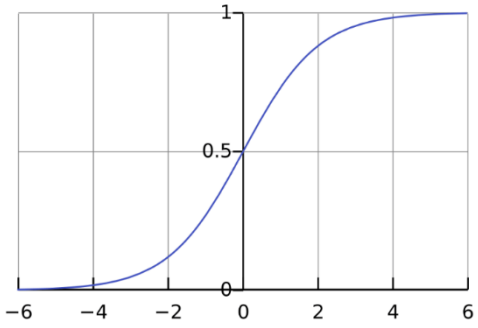


Figure 10 - Sigmoid Function

The linear regression function is:

$$Y = b_0 + b_1X_1 + b_2X_2 + \cdots + b_nX_n$$

Y is the dependent variable and $X_1, X_2, \ldots, X_n$ are the explanator variables.

If the sigmoid function is applied to the linear regression equation the ended p function is:

$$p = \frac{e^{b_0+b_1X_1+b_2X_2+\cdots+b_nX_n}}{1 + e^{b_0+b_1X_1+b_2X_2+\cdots+b_nX_n}}$$

Let $u = b_0 + b_1X_1 + b_2X_2 + \cdots + b_nX_n$,

$$\frac{p}{1-p} = \frac{\frac{e^u}{1+e^u}}{1-\frac{e^u}{1+e^u}} = \frac{\frac{e^u}{1+e^u}}{\frac{1}{1+e^u}} = \frac{e^u(1+e^u)}{1+e^u} = e^u$$

$$\frac{p}{1-p} = e^{b_0+b_1X_1+b_2X_2+\cdots+b_nX_n} = \log\left(\frac{p}{1-p}\right) = b_0 + b_1X_1 + b_2X_2 + \cdots + b_nX_n$$

Logistic Regression can be expressed as:

$$\log\left(\frac{p(X)}{1-p(X)}\right) = b_0 + b_1X_1 + b_2X_2 + \cdots + b_nX_n$$

Where the left side is called logit and $\frac{p(X)}{1-p(X)}$ is called odds. The odds signify the ratio of probability of success to probability of failure. Therefore, in Logistic Regression, linear combination of inputs is mapped to the log(odds) – The output to be equal to 1.

Logistic Regression in its straightforward way is a classification method for binary target variable. When it comes to multiclass categorical target variable, there are some techniques thar are going to be presented in the next section.

4.3. Support Vector Machines (SVM)

The original version of SVM is developed by Vapnik and Chervonenkis in 1963. SVM was originally designed for binary classification tasks. The SVM problem is to find the decision surface that maximizes the margin between the data points of the two classes while minimizing the penalty associated with the misclassifications in the training set.

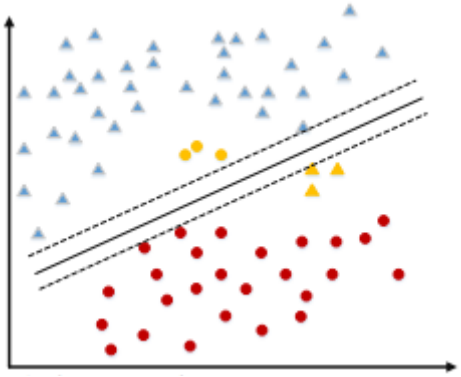


Figure 11 - Linear Support Vector Machine (SVM) for a 2D

4.3.1. Binary-Class SVM

In the context of text classification, let $d_1, d_2, \ldots, d_l$ be training examples belonging to one class D, where D is a compact subset of R^N. A set of examples $(d_1, y_1), (d_2, y_2), \ldots, (d_l, y_l)$, $y_i \in \{-1.1\}$ are given. The optimum hyperplane, defined by $(w \cdot x) + b = 0$, is the solution of the following quadratic programming problem:

$$\min \frac{1}{2}\left|\left|w\right|\right|^2 + C\sum_{i=1}^{l}\xi_i \tag{4}$$

$$s.t. \qquad y_i(d_i w + b) \geq 1 - \xi_i$$

$$\xi_i \geq 0 \; i = 1, \ldots, l$$

The above minimization problem is equivalent to solving the following Lagrangian dual problem:

$$\max_a \sum_{i=1}^{l} a_i - \frac{1}{2}\sum_{i=1}^{l}\sum_{j=1}^{l} a_i a_j y_i y_j d_i d_j \tag{5}$$

$$s.t. \sum_{i=1}^{l} a_i y_i = 0$$

$$0 \leq a_i \leq C \quad i = 1, \dots, l$$

where $a_i \ , i = 1, \dots, l$ are the Lagrange multiplier.

4.3.2. Multi-Class SVM

A multiple SVM (MSVM) must be created for multi-class situations as SVM are often used for binary classification problems. Below two techniques, OVO (One vs One) and OVA (One vs All), are going to be presented that shift traditional SVM to multi-class SVM (MSVM).

- **One vs One (OVO)**

By using this method, our multiclass classification problem is divided into smaller binary classification problems. Thus, following the application of this technique, binary classifiers are obtained for each pair of classes. The idea of majority voting combined with the distance from the margin as its confidence criterion when making a final forecast for any input is used. The major problem of this approach is that too many SVMs must be trained.

For multi-class problems with L categories:

- Positive Samples: all the data point in class $s(\{x_i : s \in y_i\})$
- Negative Samples: all the data point in class $t(\{x_i : t \in y_i\})$
- $f_{s,t}(x)$: the decision value of this specific classifier

  (Large value of $f_{s,t}(x)$ then label s has a higher probability than the label t)
- $f_{s,t}(x) = -f_{t,s}(x)$
- Prediction: $f(x) = argmax_s\left(\sum_t f_{s,t}(x)\right)$

Let it be a classification problem with 3 classes. In the One vs One strategy, the data points of the third class are ignored in the process of the search for the hyperplane that divides every pair of classes.

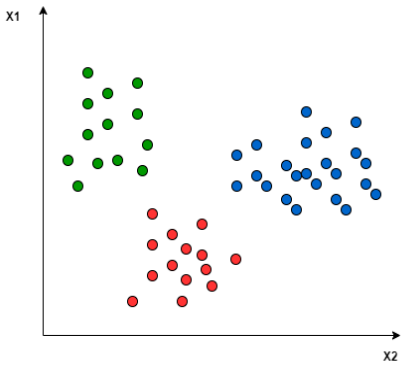


Figure 13 - Three different classes scatter plot

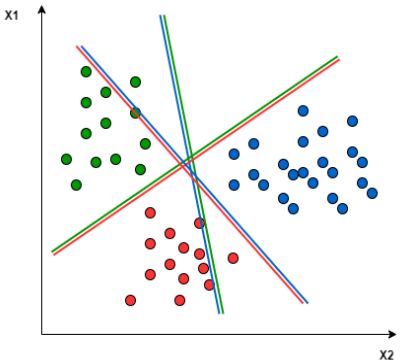


Figure 12 - One vs One Approach

Red-Blue line tries to maximize the separation only between blue and red points while it has nothing to do with the green points.

- **One vs All (OVA)**

In this technique N SVMs are being trained to deal with the classification problem of N classes. For example:

- SVM_1: $learns$ "$class\ output = 1\ vs\ class\ output \neq 1$"
- SVM_2: $learns$ "$class\ output = 2\ vs\ class\ output \neq 2$"
- SVM_3: $learns$ "$class\ output = 3\ vs\ class\ output \neq 3$"

and so on.

For the prediction of a new input, each of the bult-in SVMs are used and the predicted class is that one the SVM of which placed the prediction the furthest into the positive region.

In this strategy, challenges like too much computation time and unbalanced data problems must be overcome. For example, if a dataset contains a target variable in which there are 10 classes and each of the class consists of 1000 data points, then for any one of the SVM having two classes, one class will have 9000 points and the other will have only 1000 data points, so the problem becomes unbalanced.

For multi-class problems with L categories:

- Positive Samples: all the points in class $t(\{x_i: t \in y_i\})$
- Negative Samples: all the points not in class $t(\{x_i: t \notin y_i\})$
- $f_t(x)$: The decision value for the $t-th$ classifier

  (Large value of f_t then higher probability that x belongs in the class t)
- Prediction: $argmax_t f_t(x)$

In the same classification problem as before, the One vs All approach attempts to discover hyperplanes to isolate the classes. The separation takes all points into account and then divides them into two groups in which there is a group for the one class points and the other group for all other points.

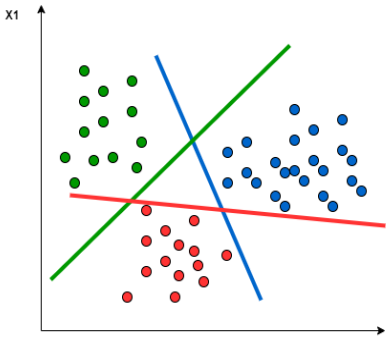


Figure 14 - One vs All Approach

4.4. Artificial Neural Networks

Simultaneously with Machine Learning techniques, Neural Networks have deep impact in many classification techniques especially in applications which deal with nontabular data like images, voice records, handwritten documents, videos provided by traffic cameras etc.

Artificial Neural Networks (ANNs) are a simulation of human brain's biological neural networks as seen in Figure 14:

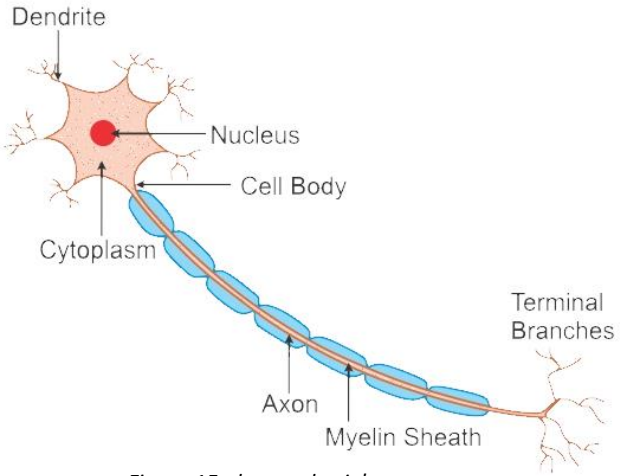


Figure 15 - human brain's neuron

A biological neuron consists of a cell body and two types of out-reaching branches, axon and terminal branches. Inside the cell body nucleus contains information about hereditary traits. Each neuron receives signals from other neurons through its

dendrites and transmits signals which are generated by its cell bod along the axon. At the terminals of these strands the functional parts between two neurons called synapses exist which release certain chemicals called neurotransmitters as the signal reaches them. The neurotransmitters diffuse across the synaptic gap, to enhance or inhibit, depending on the type of the synapse, the receptor neuron's own tendency to emit electrical impulses. The synapse's effectiveness can be adjusted by the signals passing through it so that the synapses can learn from the activities in which they participate.

Artificial Neural Networks work similarly like biological neural networks. ANNs are processors that are made up of simple processing units (neurons). They are capable of learning experiential knowledge expressed through interunit connection strengths and making such knowledge available for use. The major developments behind this rise were done back in 1982 with Hopfield's energy approach and the backpropagation learning algorithm for multilayer perceptron which first proposed by Werbos.

The architecture of an ANN which consists of a set of neurons and weighted links that connect the neurons is combined with a learning algorithm that is used for training the neural network by calculating the weights in order to model a particular learning task correctly on training examples. A regular ANN looks like as follows:

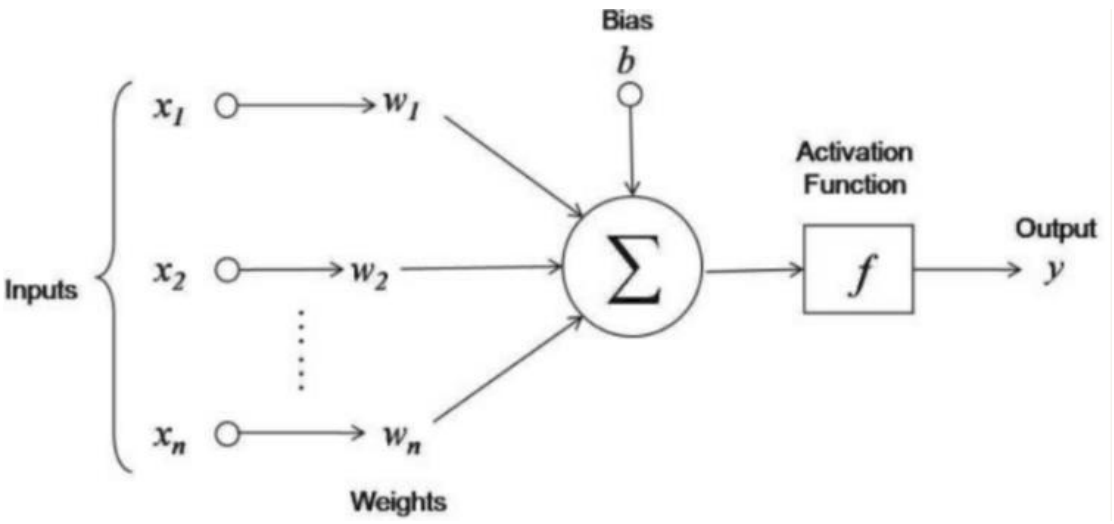


Figure 16 - ANN Architecture

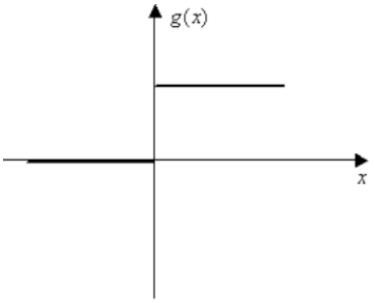
$x_1, x_2, \dots, x_n$ are the input variables while $w_1, w_2, \dots, w_n$ are weights of respective inputs. b is the bias which is summed with the weighted inputs to form the net inputs. Bias and weights are both adjustable parameters of neurons which are being adjusted using some learning rules. The output of a neuron can range from -

inf to $+inf$. Activation function which works like a mapping mechanism between the input and output decides the final value of the target variable.

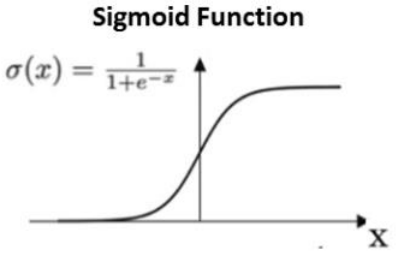
Activation function defines the output of a neuron in terms of a local induced field. There are many activation functions. Some of them are as follows:

Hard Limit:

$$y = \begin{cases} \mathbf{1}, & x \geq 0 \\ \mathbf{0}, & x < 0 \end{cases}$$

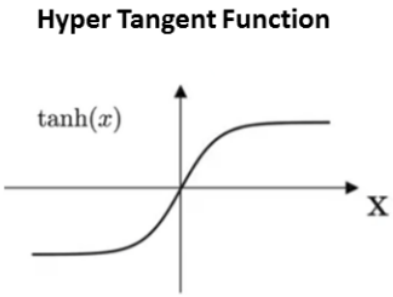


Binary Sigmoid Function is a logistic function where the output values are either binary or vary from 0 to 1.

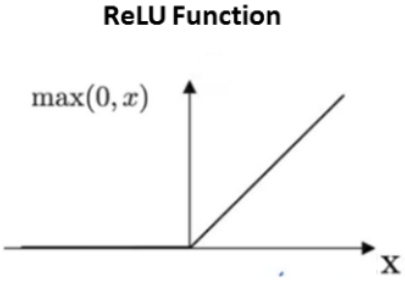


Hyper Tangent Function:

$$y = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$



ReLu stands for the rectified linear unit (**ReLu**). It is the most used activation function in the world.it output o for negative values of x.



4.4.1. Perceptron ANN

The perceptron is a single processing unit of any neural network. It is a neural network unit that does a precise computation to detect features in the input data. Perceptron is mainly used to classify the data into two parts. That is the reason it is also known as Linear Binary Classifier. The training algorithm of a Perceptron ANN is as follows:

I. Initialize weights and threshold in range $[-0.5\ ,0.5]$ using a random generator.
II. For each example $\widetilde{x_j} = (x_1, x_2, \dots, x_n)$ inside the training dataset labeled as l_j perform the following steps:
    a. calculate the following:

$$u(j) = \sum_{i=1}^{n} w_i(j) x_i(j)$$

    b. Use the activation function to map the value of $u(j)$ with the output value y_j.
    c. Update the weights as follows:

$$w_i(j+1) = w_i(j) + \eta \cdot \left(l_j - y_j\right) \cdot x_j$$

    where η is the learning rate

4.4.2. Convolutional Neural Networks (CNN)

Convolutional Neural Networks, also known as CNN or ConvNet are a subtype of neural networks, but they are distinguished from them by their extremely good performance with image, speech, or audio signal inputs. They are developed to work as the human brain does when it comes to recognize shapes, patterns, angles, lines, etc. For this reason, CNN architectures do not necessarily need hand-crafted feature extraction by humans.

A CNN architecture typically has three layers: a convolutional layer, a pooling layer, and a fully connected layer.

The Convolutional layer is the most important layer of CNN as this layer is responsible for detecting the most important features and as a matter of fact in this layer is where the most computations take place. The convolutional layer performs a dot product between two matrices, where one matrix is the set of learnable parameters which is also known as kernel or filter, and the other is the slice of the input resulting

as the filter is sliding over the input. The output matrix is a convolved feature also known as feature map. The number of pixels that the kernel slides over the input is called stride. Usually, it is equal to 1.

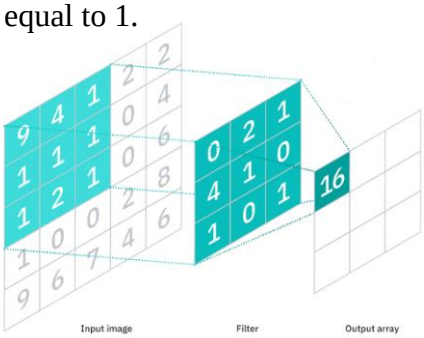


Figure 17 - Convolution Filter

The **Pooling Layer** comes after convolution layer and applies in the feature map. This layer is responsible for reducing the spatial size of the convolved feature to decrease the computational power required to process the data through dimensionality reduction. There are two types of pooling: **Max Pooling** and **Average Pooling.**

Max Pooling returns the maximum value from the portion of the image covered by the pooling kernel. While Average pooling returns the average of all he values from the portion of the image covered by the pooling kernel.

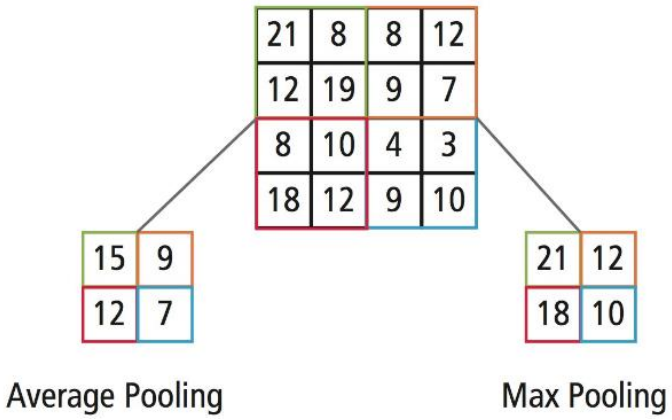


Figure 18 - Max and Average Pooling

The pooled feature map is being converted into a flattened reshaped vector of one row that it will be passed into the fully connected layer.

The **Fully Connected Layer** performs the task of classification based on the features extracted through the previous layers and their different filters. It consists of the weights and biases along with the neurons and is used to connect the neurons between two different layers. The flattened feature map becomes the input of the neural network. This is then passed to a fully connected layer. Before the final output is obtained, the result is passed to an appropriate activation function. For instance, the sigmoid activation function in the case of binary problems and softmax in the case of multi-class problems.

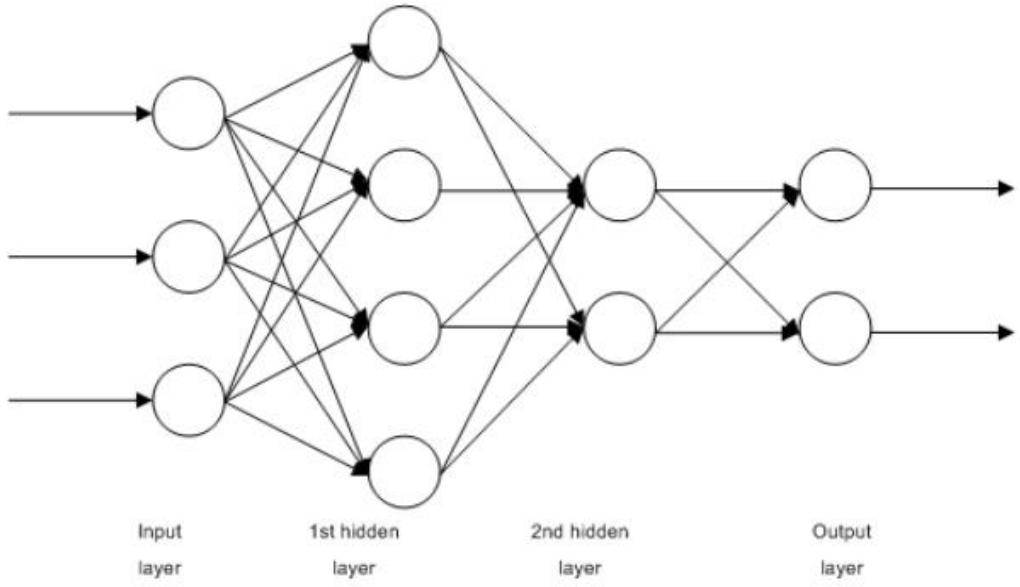


Figure 19 - Fully Connected layer

Convolutional Neural Networks for NLP

When it comes to NLP tasks, such as document classification, sentiment classification etc., the architecture of CNN is changed to 1 dimension convolutional and pooling operations. Given a sequence of words $w_{1:n} = w_1, \dots , w_n$, where each is associated with an embedding vector of dimension *d*, the 1D convolution of width-*k* is the resulting of moving a sliding-window of size *k* over a sentence and applying the same convolution filter or kernel to each window in the sequence. Considering a window of words $w_i, \dots, w_{i+k}$ the concatenated vector of the *i*th window is:

$$x_i = [w_i, w_{i+1}, \dots, w_{i+k}] \in \mathbb{R}^{k \times d}$$

The convolution filter is applied to each window, resulting in scalar values r_i, each of the *i*-th window:

$$r_i = g(x_i \cdot u) \in \mathbb{R}$$

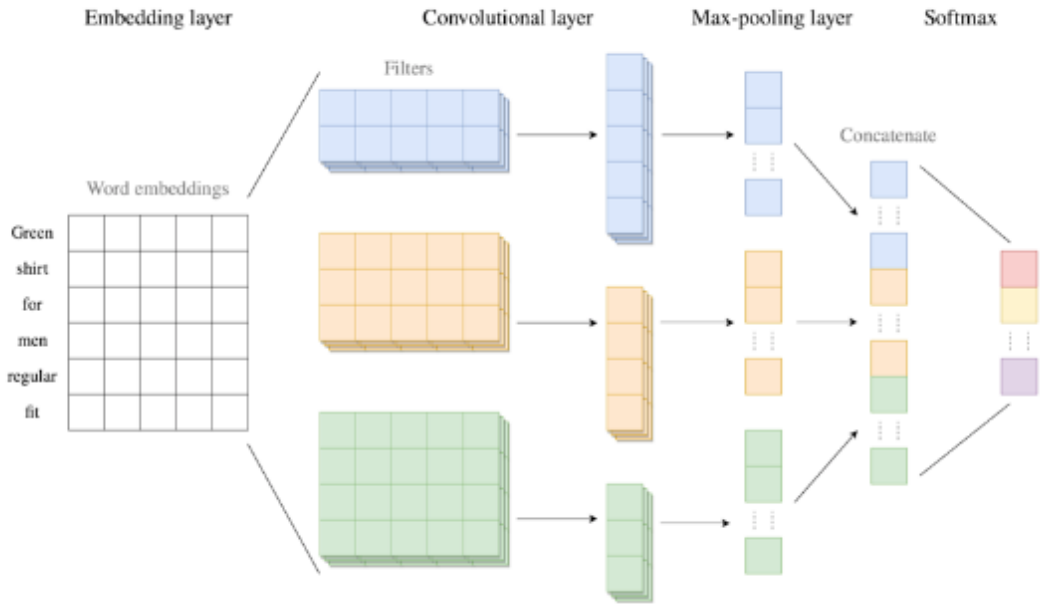


Figure 20 - General CNN architecture for text classification

Let's see how a 1-D CNN works. For a given sentence "Green shirt for men regular fit" consisting of 6 words, each word is represented by a 5-dimensional word vector, hence the sentence matrix is of 6×5 shape.

In the graph above there are three filter regions each of which has k filters. Filters perform convolutions on the sentence matrix and generate feature maps. The number of feature maps inside a region depends on the number of filters in each region size. The total number of feature maps is $r \times k$, where r is the number of regions, in our example is 3. One max-pooling layer is performed over each feature map recording the maximum value of each feature map. Thus, a univariate feature vector is generated from all feature maps, and these features are concatenated to form a feature vector the length of which is equal to the total number of filters ($number\ of\ filters\ per\ region\ size \times number\ of\ region\ size$**).** This feature vector can then be fed into a fully connected layer to perform classification.

4.4.3. Recurrent Neural Networks (RNN)

Recurrent Neural Networks are a well-known Deep Learning methodology which was initially created in the 1980s based on David Rumelhart's work in 1996. David Rumelhart was an American psychologist who made many contributions to the formal analysis of human cognition, working primarily within the frameworks of mathematical psychology, symbolic artificial intelligence, and parallel distributed processing.

RNNs work on the guideline of saving the output of a specific layer and feeding this back to the input in arrange to predict the output layer. Since of their inner memory, RNNs can remember vital things about the input they have received, which allows them to be exceptionally exact in predicting what's coming next. This is why they're the preferred algorithm for sequential data like time series, speech, text, financial data, audio, video, weather and much more.

The information in a Recurrent Neural Network cycles through a loop to the middle hidden layer. The middle layer "h" which can consist of multiple hidden layers, each of them with its own activation functions, weights, and biases, takes the processed input from the input layer "x". Finally, the RNN will define activation functions and will standardize all the weights and biases in a way that each hidden layer consists of the same parameters.

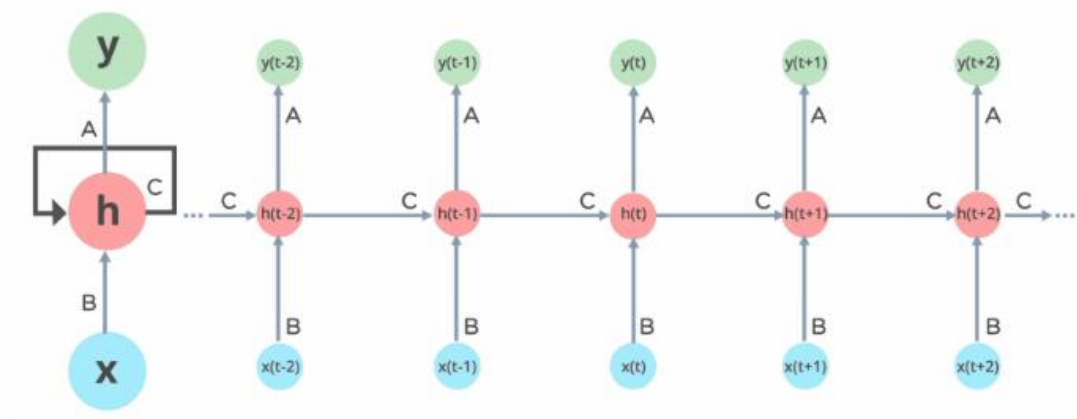


Figure 21 - Recurrent Neural Network

Two inputs are used in a Recurrent Neural Network, one for the present and another for the recent past. This is because sequential data contains important information about what is coming next. This is why RNN can deal with problems that other algorithms cannot handle in an efficient way. A feed-forward neural network assigns a weight matrix to its inputs and then produces the output. RNNs apply weights to the current and to the previous input. Furthermore, a recurrent neural network will also tweak the weights for both gradient descent and backpropagation through time.

Feed-forward neural networks map one input tone output while RNNs can map one to many, many to one, many to many.

One to One RNN, also known as Vanilla Neural Network, is used for general machine learning problems, which has a single input and a single output. One to Many

RNN has a single input and multiple outputs. An example of its usage I the image caption. Many to One RNN generates a single output by receiving a sequence of inputs. Sentiment analysis is an example where this kind of RNN can be used as a classifier of a sentence to be positive or negative. Finally, Many to Many RNN takes a sequence of inputs and generates a sequence of outputs. This kind of RNN is used in tasks such as machine translation.

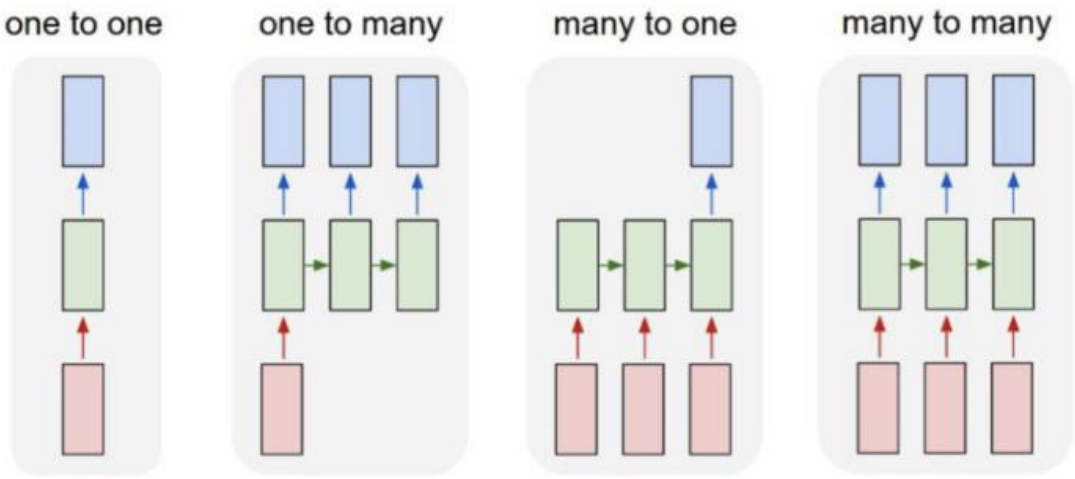


Figure 22 - Types of RNNs

Recurrent Neural Networks is a sequence of neural networks that are trained one after another with backpropagation. Backpropagation is an algorithm which is used for calculating the gradient of an error function with respect to network's weights. This process is done by "walking" the algorithm its way backwards through the various layers of gradients to find the derivative of the errors with respect to the weights. As RNN is a network with a loop inside it, the backpropagation is done on an unrolled RNN, and it is called **Backpropagation Through Time (BPTT).**

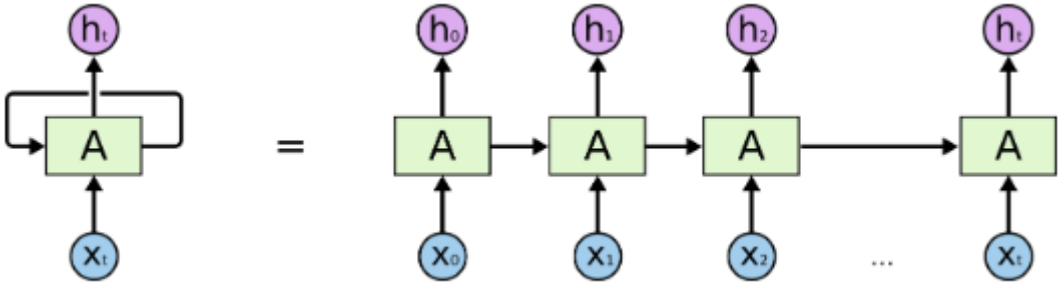


Figure 23 - Unrolled Recurrent Neural Network

There are two significant issues of standard RNNs. Exploding gradients and vanishing gradients. Exploding gradients are when the algorithm assigns a high importance to the weights. This problem can be easily solved by truncating or squashing the gradients. Vanishing gradients occur when the values of a gradient are too small, and the model stops learning or takes way too long as a result. This was a major problem in the 1990s and much harder to solve than the exploding gradients. Fortunately, it was solved through the concept of LSTM by Sepp Hochreiter and Juergen Schmidhuber.

4.4.4. Long Short-Term Memory (LSTM)

Long-Short Term Memory (LSTM) is a type of recurrent neural network but in terms of memory is better. LSTMs are developed to avoid the long-term dependency problem. Like all neural networks, LSTMs can have multiple hidden layers where the relevant information is kept, and all the irrelevant information gets discarded as it passe through every hidden layer. The main role of LSTM model is described by a memory cell known as "cell state" which maintains its state over time. LSTM has three main gates: **Forget Gate**, **Input Gate** and **Output Gate**.

Forget gate layer is a sigmoid layer which is responsible for deciding which information is kept for calculating the cell state and which is up to be discarded. It takes as an input h_{t-1} which is the information from the previous cell and x_t which is the information from the current cell and outputs a number between 0 and 1 for each number in the cell state C_{t-1}. Number close to 1 means that the information must be kept while a number close to 0 means that the information must be discarded.

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$

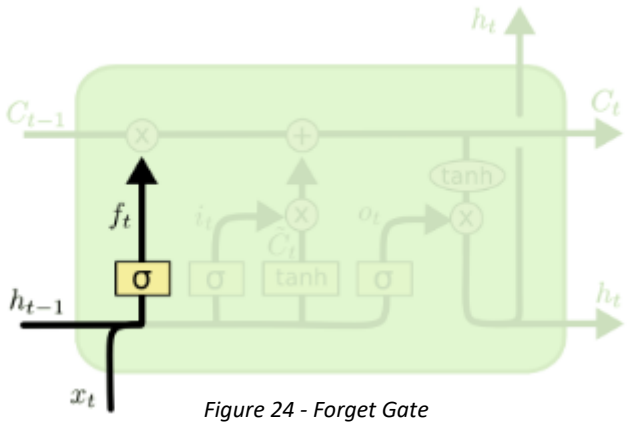


Figure 24 - Forget Gate

The input gate is responsible for deciding which information is important and which is not to be stored in the cell state. This is done by two layers, a sigmoid layer called “input gate layer” which decides which values are going to be updated and a $tanh$ layer that creates a vector of new candidate values, $\tilde{C}_t$, that could be added to the state.

$$i_t = s(W_i \cdot [h_{t-1}, x_t] + b_i)$$

$$\tilde{C}_t = tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

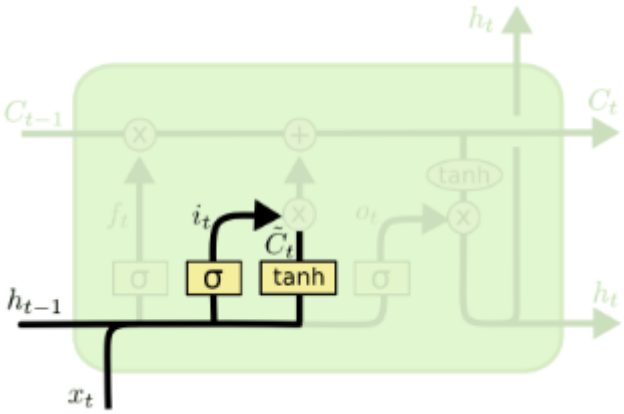


Figure 25 - Input Gate

After the decisions done from the previous gates, the old cell state, C_{t-1}, is going to be updated into the new cell state C_t. This is done by multiplying the old state by f_t and adding $i_t * \tilde{C}_t$. First multiplication helps in forgetting all the information that was decided to be discarded. The second multiplication contains the new candidate values, scaled by how much each state was decided to be updated.

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

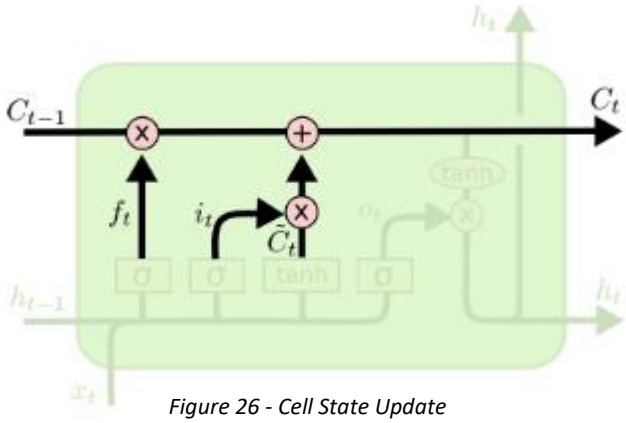


Figure 26 - Cell State Update

The output gate decides what the next state should be. h_{t-1} and x_t are passed to a sigmoid layer which decides what parts of the cell state are going to be outputted. Then the new cell state, C_t, is going through a $tanh$ layer and is multiplied by the output of the sigmoid layer to decide what information the hidden state should carry.

$$o_t = \sigma(W_0[h_{t-1}, x_t] + b_o)$$

$$h_t = o_t * \tanh(C_t)$$

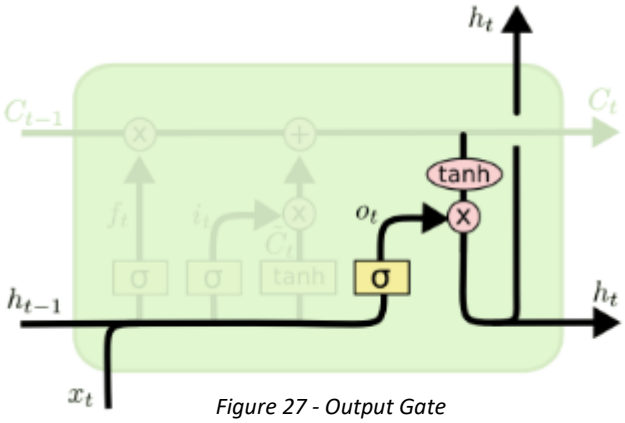


Figure 27 - Output Gate

5. Clinical Text Analysis

In this chapter all text preprocessing and classification techniques are going to be presented under the shade of a clinical text dataset with the scope of the development of a model that classifies clinical documents with the best performance.

5.1. Dataset

The dataset that is going to be used for this purpose is mtsamples.csv gathered from Kaggle datasets. The dataset consists of 5 columns and 4999 rows and contains sample transcription reports for many specialties and different work types.

Column	Description
description	Short description of transcription
medical_specialty	Medical specialty of transcription
sample_name	Transcription Title
transcription	Sample medical transcription
keywords	Relevant keywords from transcription

Table 6 - Data set columns

For text analytics and text classification, all columns are going to be dropped from the dataset except column “transcription” as it contains all the text that is needed, and column “medical_specialty” as it is the target variable that is going to be used for the classification of the transcriptions. “transcription” column is string column and contains various length strings with the shortest at 11 characters and the longest at 18425 characters. The distribution of the lengths of the medical transcription is shown in the graph below.

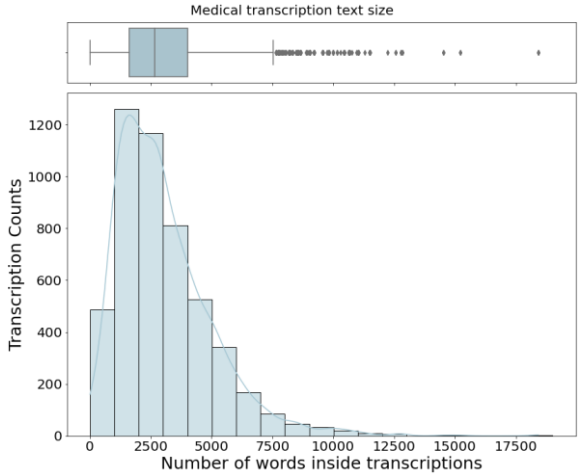


Figure 28 - Distribution of medical transcription text lengths

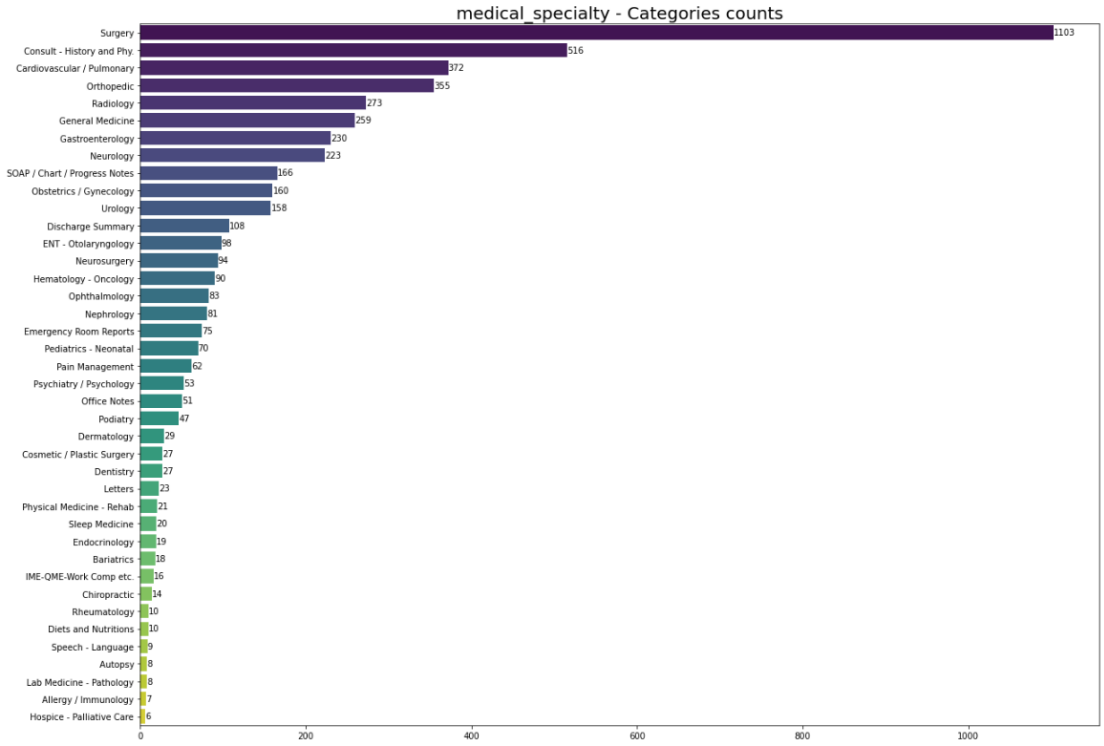


Figure 29 - Counts of each category in target variable

As it is observed from the figure above, "medical_specialty" is categorical variable with 40 unique categories. Some of the categories are the minority as they appear less than 50 times in the entire dataset. The most frequent category is "Surgery" as it appears in 1103 documents out of 4999.

33 rows have NaN values in the "transcription" column although there are not "transactions" that contain only whitespaces or are empty strings. These rows are going to be dropped as they do not contribute to the analysis.

	transcription	medical_specialty
97	NaN	Urology
116	NaN	Urology
205	NaN	Surgery
263	NaN	Surgery
459	NaN	Surgery
622	NaN	Surgery
628	NaN	Surgery
680	NaN	Surgery
729	NaN	Surgery
871	NaN	Surgery
879	NaN	Surgery

Figure 30 - NaN Values for "transcription" column

5.2. Data Preprocessing

Categories with less than 50 appearances in the dataset are going to be removed, this will reduce the medical specialty categories from 40 to 21.

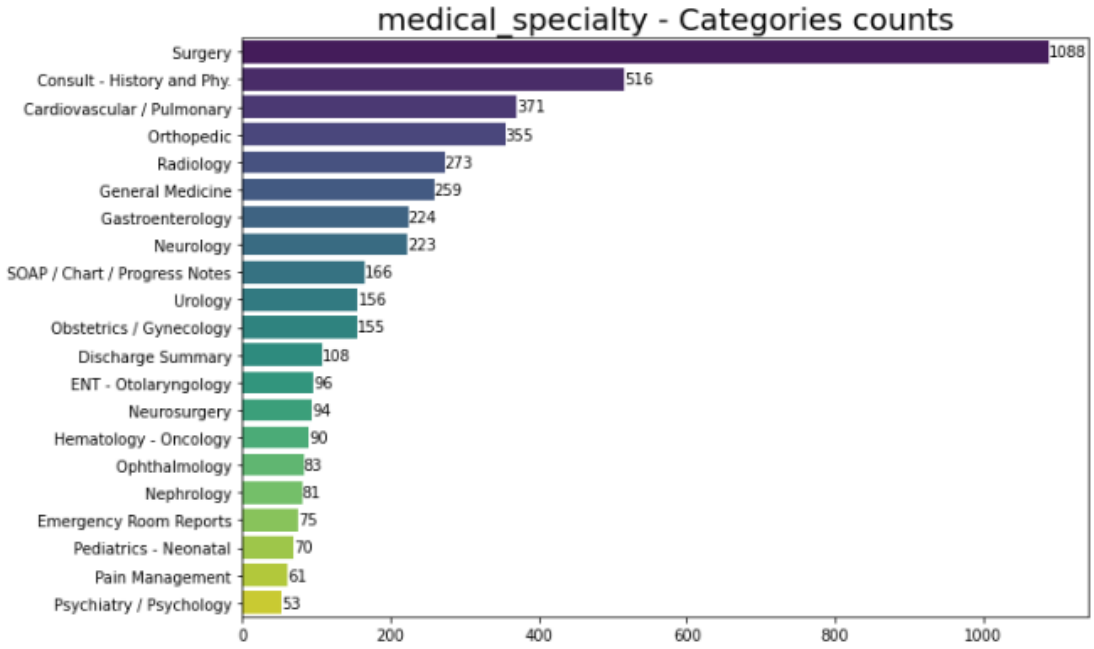


Figure 31 - Counts of each category in target variable after reduction

It is obvious that the dataset is imbalanced. This might cause overfitting while a model is getting trained. For this reason, later, oversampling and under sampling techniques may be used to avoid overfitting.

A random row of the dataset consists of two fields, the free text of the transcription and the category in which the transcription belongs to.

index	text	category
0	2-D M-MODE: , ,1. Left atrial enlargement with left atrial diameter of 4.7 cm.,2. Normal size right and left ventricle.,3. Normal LV systolic function with left ventricular ejection fraction of 51%.,4. Normal LV diastolic function.,5. No pericardial effusion.,6. Normal morphology of aortic valve, mitral valve, tricuspid valve, and pulmonary valve.,7. PA systolic pressure is 36 mmHg.,DOPPLER: , ,1. Mild mitral and tricuspid regurgitation.,2. Trace aortic and pulmonary regurgitation.	Cardiovascular / Pulmonary
1	1. The left ventricular cavity size and wall thickness appear normal. The wall motion and left ventricular systolic function appears hyperdynamic with estimated ejection fraction of 70% to 75%. There is near-cavity obliteration seen. There also appears to be increased left ventricular outflow tract gradient at the mid cavity level consistent with hyperdynamic left ventricular systolic function. There is abnormal left ventricular relaxation pattern seen as well as elevated left atrial pressures seen by Doppler examination.,2. The left atrium appears mildly dilated.,3. The right atrium and right ventricle appear normal.,4. The aortic root appears normal.,5. The aortic valve appears calcified with mild aortic valve stenosis, calculated aortic valve area is 1.3 cm square with a maximum instantaneous gradient of 34 and a mean gradient of 19 mm.,6. There is mitral annular calcification extending to leaflets and supportive structures with thickening of mitral valve leaflets with mild mitral regurgitation.,7. The tricuspid valve appears normal with trace tricuspid regurgitation with moderate pulmonary artery hypertension. Estimated pulmonary artery systolic pressure is 49 mmHg. Estimated right atrial pressure of 10 mmHg.,8. The pulmonary valve appears normal with trace pulmonary insufficiency.,9. There is no pericardial effusion or intracardiac mass seen.,10. There is a color Doppler suggestive of a patent foramen ovale with lipomatous hypertrophy of the interatrial septum.,11. The study was somewhat technically limited and hence subtle abnormalities could be missed from the study.,	Cardiovascular / Pulmonary
2	2-D ECHOCARDIOGRAM,Multiple views of the heart and great vessels reveal normal intracardiac and great vessel relationships. Cardiac function is normal. There is no significant chamber enlargement or hypertrophy. There is no pericardial effusion or vegetations seen. Doppler interrogation, including color flow imaging, reveals systemic venous return to the right atrium with normal tricuspid inflow. Pulmonary outflow is normal at the valve. Pulmonary venous return is to the left atrium. The interatrial septum is intact. Mitral inflow and ascending aorta flow are normal. The aortic valve is trileaflet. The coronary arteries appear to be normal in their origins. The aortic arch is left-sided and patent with normal descending aorta pulsatility.	Cardiovascular / Pulmonary

The free text of the transcription is a string that contains punctuations, numbers and words that are very common in all the transcription provided through the dataset, those words are called stop words. In the next steps each transcription row will be cleaned which means that all the punctuations, alphanumeric characters and stop words are going to be removed.

index	text	category
0	left atrial enlargement left atrial diameter normal size right left normal lv systolic function left ventricular ejection fraction normal lv diastolic pericardial normal morphology aortic valve mitral valve tricuspid valve pulmonary pa systolic pressure doppler mild mitral tricuspid trace aortic pulmonary regurgitation	Cardiovascular / Pulmonary
1	left ventricular cavity size wall thickness appear normal wall motion left ventricular systolic function appears hyperdynamic estimated ejection fraction obliteration seen also appears increased left ventricular outflow tract gradient mid cavity level consistent hyperdynamic left ventricular systolic function abnormal left ventricular relaxation pattern seen well elevated left atrial pressures seen doppler left atrium appears mildly right atrium right ventricle appear aortic root appears aortic valve appears calcified mild aortic valve stenosis calculated aortic valve area cm square maximum instantaneous gradient mean gradient mitral annular calcification extending leaflets supportive structures thickening mitral valve leaflets mild mitral tricuspid valve appears normal trace tricuspid regurgitation moderate pulmonary artery hypertension estimated pulmonary artery systolic pressure mmhg estimated right atrial pressure pulmonary valve appears normal trace pulmonary pericardial effusion intracardiac mass color doppler suggestive patent foramen ovale lipomatous hypertrophy interatrial study somewhat technically limited hence subtle abnormalities could missed	Cardiovascular / Pulmonary
2	echocardiogram multiple views heart great vessels reveal normal intracardiac great vessel relationships cardiac function normal significant chamber enlargement hypertrophy pericardial effusion vegetations seen doppler interrogation including color flow imaging reveals systemic venous return right atrium normal tricuspid inflow pulmonary outflow normal valve pulmonary venous return left atrium interatrial septum intact mitral inflow ascending aorta flow normal aortic valve trileaflet coronary arteries appear normal origins aortic arch patent normal descending aorta pulsatility	Cardiovascular / Pulmonary

The text field now looks cleaner as all the punctuation, alphanumeric characters and stop words are removed. There are 13716 unique words inside the cleaned transcription column and the 20 most frequent words are:

```
Nr of unique words is :  13716
======================================
20 most used words in the dataset are:
======================================
patient     22289
right       10318
left        10077
histori      8664
use          8361
place        7922
procedur     7338
normal       6444
well         5440
pain         4892
medic        4453
remov        4364
note         4319
time         4315
incis        4121
perform      3945
oper         3924
also         3923
posit        3771
blood        3678
```

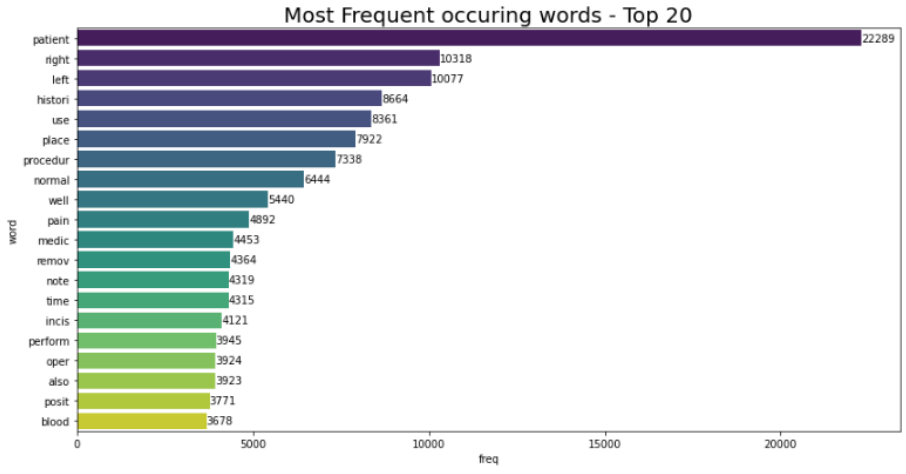


Figure 32 - Most frequent occurring words

5.3. Modeling

5.3.1. Initial Experiment

All the three algorithms (Logistic Regression, SVM and Multinomial Naïve Bayes) will be fitted and evaluated with the preprocessed data. The parameters of each model and vectorization algorithms, as well, are decided by applying grid search with cross validation through pipeline. All the experiments will be tracked using MLflow opensource platform.

5.3.1.1. Multinomial Naïve Bayes:

The corpus is vectorized using **Tfidf Vectorizer** with ngram range set to (1,2) and with use of idf term.

label	*precision*	*recall*	*f1-score*	*support*
Cardiovascular / Pulmonary	49.06%	35.14%	40.94%	74
Neurology	33.33%	26.67%	29.63%	45
Urology	100.00%	3.23%	6.25%	31
General Medicine	0.00%	0.00%	0.00%	52
Surgery	45.83%	80.73%	58.47%	218
SOAP / Chart / Progress Notes	0.00%	0.00%	0.00%	33
Radiology	33.33%	32.73%	33.03%	55
Psychiatry / Psychology	0.00%	0.00%	0.00%	10
Pediatrics - Neonatal	0.00%	0.00%	0.00%	14
Pain Management	0.00%	0.00%	0.00%	12
Orthopedic	26.19%	15.49%	19.47%	71
Ophthalmology	0.00%	0.00%	0.00%	17
Obstetrics / Gynecology	37.50%	9.68%	15.38%	31
Neurosurgery	0.00%	0.00%	0.00%	19
Nephrology	0.00%	0.00%	0.00%	16
Hematology - Oncology	0.00%	0.00%	0.00%	18
Gastroenterology	35.29%	13.33%	19.35%	45
ENT - Otolaryngology	0.00%	0.00%	0.00%	19
Emergency Room Reports	0.00%	0.00%	0.00%	15
Discharge Summary	20.00%	4.55%	7.41%	22
Consult - History and Phy.	28.93%	89.32%	43.71%	103
accuracy		37.61%		
macro avg	19.50%	14.80%	13.03%	920
weighted avg	30.53%	37.61%	28.82%	920

Table 7 - Classification report of NB

From *table 7*, can be observed that the initial Multinomial Naïve Bayes classifier achieved poor results trying to classify each specialty into the correct label. With an accuracy of 37.61% and extremely imbalanced data, the classifier predicted correctly 80.73% of true "Surgery" specialty, and 89.32% of true "Consult History and Phy." specialty. All the other labels had less than 50% recall.

5.3.1.2. Logistic Regression:

Again, the corpus is vectorized using **Tfidf Vectorizer** with ngram range set to (1,2) and without use of idf term as grid search decided those to be the optimal parameters.

label	precision	recall	f1-score	support
Cardiovascular / Pulmonary	37.74%	27.03%	31.50%	74
Neurology	31.58%	26.67%	28.92%	45
Urology	25.00%	3.23%	5.71%	31
General Medicine	13.33%	7.69%	9.76%	52
Surgery	43.52%	77.06%	55.63%	218
SOAP / Chart / Progress Notes	35.48%	33.33%	34.38%	33
Radiology	32.35%	40.00%	35.77%	55
Psychiatry / Psychology	0.00%	0.00%	0.00%	10
Pediatrics - Neonatal	33.33%	7.14%	11.76%	14
Pain Management	50.00%	33.33%	40.00%	12
Orthopedic	18.75%	12.68%	15.13%	71
Ophthalmology	40.00%	11.76%	18.18%	17
Obstetrics / Gynecology	25.00%	3.23%	5.71%	31
Neurosurgery	0.00%	0.00%	0.00%	19
Nephrology	0.00%	0.00%	0.00%	16
Hematology - Oncology	0.00%	0.00%	0.00%	18
Gastroenterology	9.09%	2.22%	3.57%	45
ENT - Otolaryngology	100.00%	5.26%	10.00%	19
Emergency Room Reports	0.00%	0.00%	0.00%	15
Discharge Summary	39.02%	72.73%	50.79%	22
Consult - History and Phy.	34.76%	63.11%	44.83%	103
accuracy		36.74%		
macro avg	27.09%	20.31%	19.13%	920
weighted avg	31.22%	36.74%	30.26%	920

Table 8 - Classification report of Logistic Regression

From *table 8*, can be observed that the initial Logistic Regression classifier achieved poor results as well as Multinomial Naïve Bayes, trying to classify each specialty into the correct label. With an accuracy of 36.74% and extremely imbalanced data, the classifier predicted correctly 77.06% of true "Surgery" specialty, 72.73% of true "Discharge Summary" specialty and 62.11% of true "Consult History and Phy." specialty. All the other labels had less than 50% recall.

5.3.1.3. Support Vector Machines

As the other two algorithms, SVM trained with the corpus vectorized again with ngram range at (1,2) and without idf term.

label	***precision***	***recall***	***f1-score***	***support***
Cardiovascular / Pulmonary	33.93%	25.68%	29.23%	74
Neurology	26.19%	24.44%	25.29%	45
Urology	0.00%	0.00%	0.00%	31
General Medicine	12.90%	7.69%	9.64%	52
Surgery	41.37%	69.27%	51.80%	218
SOAP / Chart / Progress Notes	32.14%	27.27%	29.51%	33
Radiology	23.73%	25.45%	24.56%	55
Psychiatry / Psychology	33.33%	10.00%	15.38%	10
Pediatrics - Neonatal	11.11%	7.14%	8.70%	14
Pain Management	60.00%	75.00%	66.67%	12
Orthopedic	13.46%	9.86%	11.38%	71
Ophthalmology	14.29%	5.88%	8.33%	17
Obstetrics / Gynecology	0.00%	0.00%	0.00%	31
Neurosurgery	0.00%	0.00%	0.00%	19
Nephrology	0.00%	0.00%	0.00%	16
Hematology - Oncology	0.00%	0.00%	0.00%	18
Gastroenterology	0.00%	0.00%	0.00%	45
ENT - Otolaryngology	16.67%	5.26%	8.00%	19
Emergency Room Reports	0.00%	0.00%	0.00%	15
Discharge Summary	34.15%	63.64%	44.44%	22
Consult - History and Phy.	32.34%	52.43%	40.00%	103
accuracy			32.17%	
macro avg	18.36%	19.48%	17.76%	920
weighted avg	24.51%	32.17%	26.84%	920

Table 9 - Classification report of SVM

Specialties like “Surgery”, “Pain Management”, “Discharge Summary” and “Consult-History and Phy.” achieved a recall of more than 50%, which means that support vector machines could predict more than 50% of the true labels of them. Although model accuracy remains at a low level.

5.3.1.4. Experiment Summary

Model	*Accuracy*	*Precision*	*Recall*	*f-1-score*
Multinomial NB	37.61%	30.53%	37.61%	28.82%
Logistic Regression	36.74%	31.22%	36.74%	30.26%
SVM	32.17%	24.51%	32.17%	26.84%

Table 10 - Summary of the “Initial Experiment”

The best performance is that of Multinomial Naïve Bayes classifier with an accuracy of 37.61%, precision weighted average 30.53%, recall weighted average 37.61% and f1-score weighted average 28.82%. Although the results are extremely poor.

5.3.2. Dropping Categories Experiment

From the previous experiment, the results were too poor so some domain knowledge will be applied to improve the results. As mentioned in the previous experiment, the “surgery” specialty is a superset inside of which are more specialties included, also some other specialties like “SOAP / Chart / Progress Notes”, “Discharge Summary” etc. overlap with other specialties. Rows of those specialties will be removed. Additionally, two specialties will be combined int one. More specifically, “Neurosurgery” will be transformed into “Neurology” and “Nephrology” will be transformed into “Urology”.

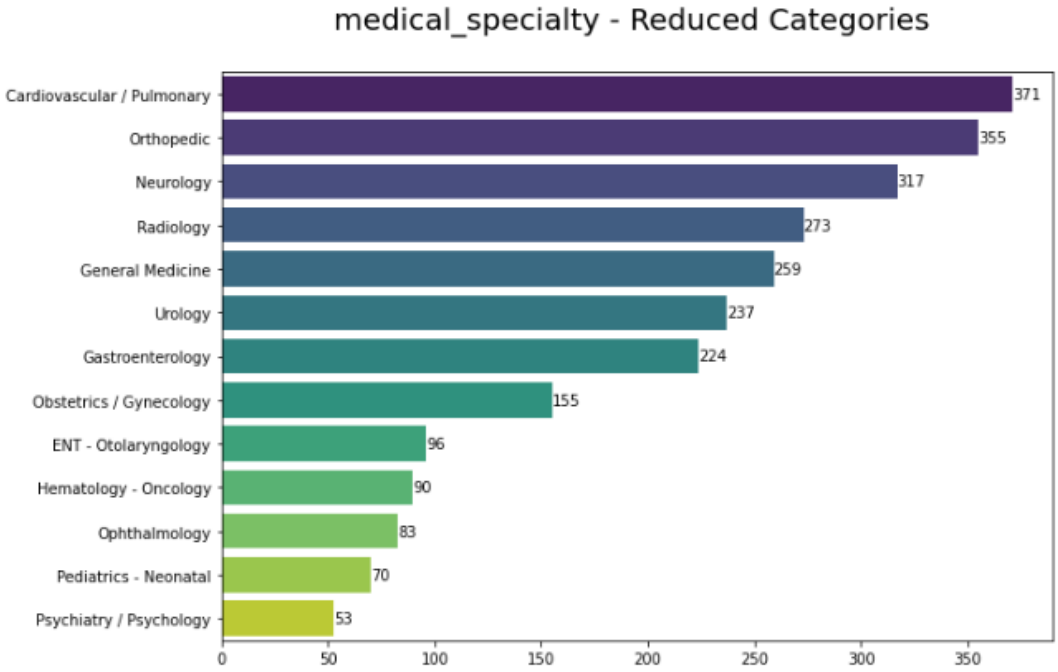


Figure 32 - Reduced Categories of specialties

After dropping and merging categories, the dataset contains 2583 documents of 13 categories. The most frequent category is “Cardiovascular/Pulmonary” with 371 appearances and the least frequent is “Psychiatry/Psychology” with 53 appearances in the dataset.

5.3.2.1. Multinomial Naïve Bayes:

The corpus is vectorized using **Tfidf Vectorizer** with ngram range set to (1,1) and the use of idf term.

label	precision	recall	f1-score	support
Cardiovascular / Pulmonary	71.88%	62.16%	66.67%	74
Neurology	56.36%	49.21%	52.54%	63
Urology	75.56%	72.34%	73.91%	47
General Medicine	37.90%	90.38%	53.41%	52
Radiology	31.91%	27.27%	29.41%	55
Psychiatry / Psychology	75.00%	81.82%	78.26%	11
Pediatrics - Neonatal	50.00%	14.29%	22.22%	14
Orthopedic	72.06%	69.01%	70.50%	71
Ophthalmology	100.00%	70.59%	82.76%	17
Obstetrics / Gynecology	73.33%	70.97%	72.13%	31
Hematology - Oncology	28.57%	22.22%	25.00%	18
Gastroenterology	96.55%	62.22%	75.68%	45
ENT - Otolaryngology	84.62%	57.89%	68.75%	19
accuracy		59.96%		
macro avg	65.67%	57.72%	59.33%	517
weighted avg	64.27%	59.96%	60.14%	517

Table 11 - Classification report of NB

The results are quite better than the initial experiment with a model accuracy close to 60%. It is remarkable that the "Ophthalmology" category reached a precision of 100% which means that all the documents that were predicted as "Ophthalmology" were actually, "Ophthalmology". Some categories like "Pediatrics - Neonatal" and "Radiology" still do not perform as expected with an f-1 score less than 50%.

5.3.2.2. Logistic Regression:

Again, the corpus is vectorized using **Tfidf Vectorizer** with ngram range set to (1,1) and with the use of idf term as Grid Search decided as best parameters. Fitting Logistic Regression model with "newton-cg" set as solver and "l2" as penalty parameters:

label	precision	recall	f1-score	support
Cardiovascular / Pulmonary	69.23%	72.97%	71.05%	74
Neurology	58.57%	65.08%	61.65%	63
Urology	72.92%	74.47%	73.68%	47
General Medicine	47.25%	82.69%	60.14%	52
Radiology	22.45%	20.00%	21.15%	55
Psychiatry / Psychology	80.00%	36.36%	50.00%	11
Pediatrics - Neonatal	50.00%	7.14%	12.50%	14
Orthopedic	71.05%	76.06%	73.47%	71
Ophthalmology	100.00%	70.59%	82.76%	17
Obstetrics / Gynecology	77.78%	67.74%	72.41%	31
Hematology - Oncology	20.00%	11.11%	14.29%	18
Gastroenterology	94.12%	71.11%	81.01%	45
ENT - Otolaryngology	93.33%	73.68%	82.35%	19
accuracy		62.67%		
macro avg	65.90%	56.08%	58.19%	517
weighted avg	63.90%	62.67%	61.81%	517

Table 12 - Classification report of Logistic Regression

As shown from *table 12*, the Logistic Regression classifier achieved an accuracy of 62.67% which is better than Multinomial Naïve Bayes classifier. The same problem with "Pediatrics – Neonatal" and "Radiology" categories occurs also with Logistic Regression.

5.3.2.3. Support Vector Machines

label	precision	recall	f1-score	support
Cardiovascular / Pulmonary	64.10%	67.57%	65.79%	74
Neurology	56.72%	60.32%	58.46%	63
Urology	69.39%	72.34%	70.83%	47
General Medicine	45.88%	75.00%	56.93%	52
Radiology	27.78%	27.27%	27.52%	55
Psychiatry / Psychology	80.00%	36.36%	50.00%	11
Pediatrics - Neonatal	60.00%	21.43%	31.58%	14
Orthopedic	72.86%	71.83%	72.34%	71
Ophthalmology	92.86%	76.47%	83.87%	17
Obstetrics / Gynecology	86.36%	61.29%	71.70%	31
Hematology - Oncology	33.33%	33.33%	33.33%	18
Gastroenterology	93.94%	68.89%	79.49%	45
ENT - Otolaryngology	88.24%	78.95%	83.33%	19
accuracy	61.51%			
macro avg	67.03%	57.77%	60.40%	517
weighted avg	64.11%	61.51%	61.69%	517

Table 13 - Classification report of SVM

The accuracy of Support Vector Machines is 61.51%, slightly less than Logistic Regression's. The category of "Pediatrics -Neonatal" has a recall of 21.43% which is better than the previous models but still very poor.

5.3.2.4. Experiment Summary

Model	*Accuracy*	*Precision*	*Recall*	*F1*
Multinomial NB	59.96%	64.27%	59.96%	60.14%
Logistic Regression	62.67%	63.90%	62.67%	61.81%
SVM	61.51%	64.11%	61.51%	61.69%

Table 14 - Summary of the "Dropping Categories Experiment"

The best performance with this experiment is Logistic Regression's classifier with an accuracy of 62.67%, precision weighted average 63.90%, recall weighted average 62.67% and f1-score weighted average 61.81%. The results are pretty good but there are some issues that must be solved. All three models fail to identify correctly labels like "Radiology", "Psychiatry / Psychology", "Pediatrics - Neonatal" and "Hematology - Oncology".

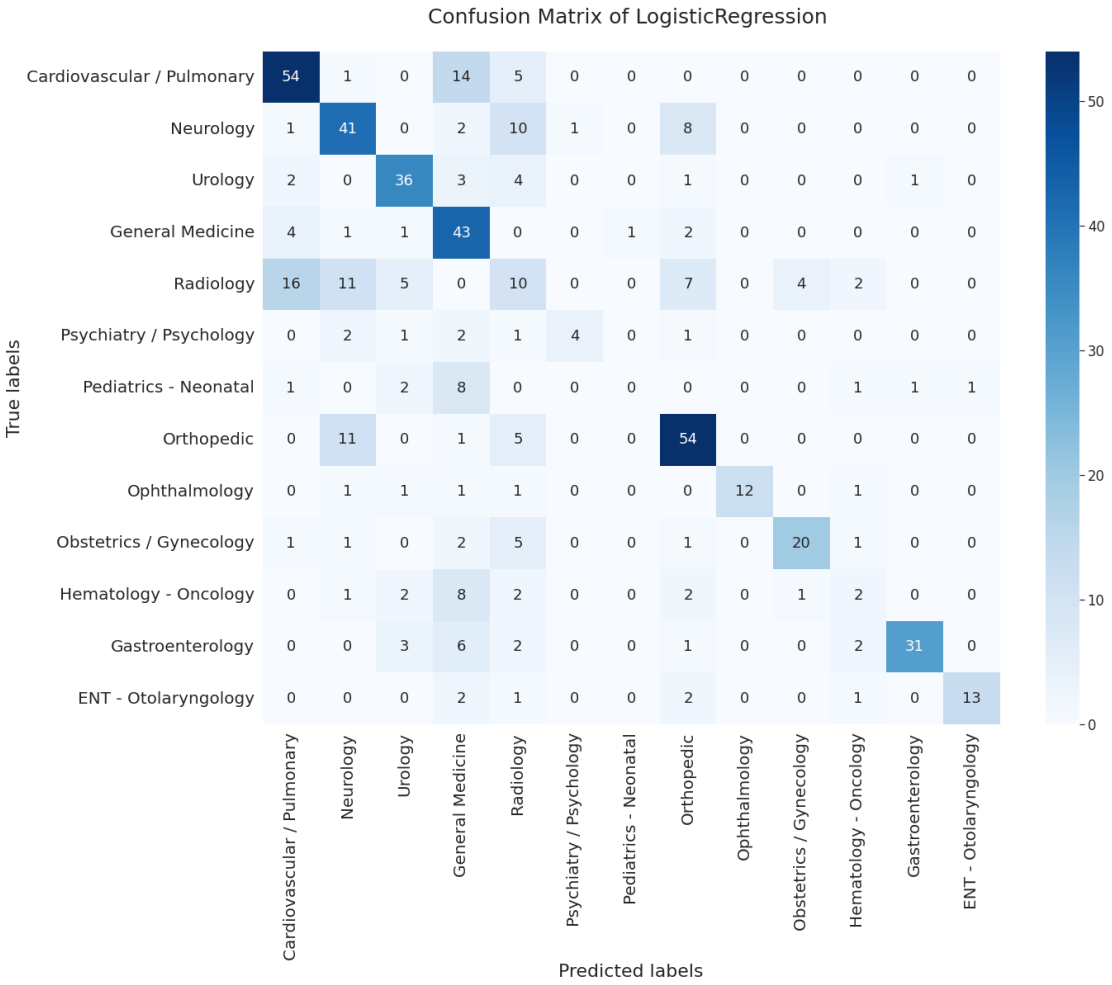


Figure 33 - Confusion Matrix of Logistic Regression (Dropped Categories experiment)

In general, the results are pretty good with the SVM classifier as they appear to be from the confusion matrix. The diagonal is full of correctly predicted documents but still there are documents that are being misclassified to some labels like “General Medicine” and “Radiology” mainly.

5.3.3. Dropping Categories and Feature Selection using X^2

In this experiment, feature selection will be implemented with scope of dimensionality reduction. X^2 test will be used to test the independence between each feature and the target variable. All the features out of 2500 with p-value more than 0.05 will be dropped. After selecting most relevant features using X^2 test, the modeling data frame know includes only 1024 features. Below are the 10 most significant features of each category:

```
# Cardiovascular / Pulmonary:
  . selected features: 127
  . top features: coronari,arteri,atrial,ventricular,pulmonari,circumflex,aortic,main,bronchoscop,chest

# Neurology:
  . selected features: 105
  . top features: dura,brain,tempor,subdur,csf,frontal,mri,hydrocephalu,sensori,motor

# Urology:
  . selected features: 60
  . top features: bladder,prostat,urethra,renal,ureter,va,peni,kidney,inguin,testi

# General Medicine:
  . selected features: 128
  . top features: neg,histori,sound,tender,subject,mg,cough,throat,regular,diabet

# Radiology:
  . selected features: 63
  . top features: imag,exam,fetal,mci,comparison,contrast,ct,axial,mild,signal

# Psychiatry / Psychology:
  . selected features: 105
  . top features: axi,suicid,ideat,behavior,psychiatr,disord,substanc,depress,hallucin,mental

# Pediatrics - Neonatal:
  . selected features: 49
  . top features: mom,feed,babi,child,infant,immun,ounc,ductu,dad,otiti

# Orthopedic:
  . selected features: 166
  . top features: tourniquet,carpal,knee,tendon,screw,joint,fractur,medial,cement,metatars

# Ophthalmology:
  . selected features: 65
  . top features: cataract,chamber,eye,intraocular,len,lid,limbu,phacoemulsif,scleral,viscoelast

# Obstetrics / Gynecology:
  . selected features: 107
  . top features: cervix,uterin,uteru,vagin,fallopian,deliveri,ovari,fetal,placenta,pelvic

# Hematology - Oncology:
  . selected features: 28
  . top features: carcinoma,chemotherapi,breast,squamou,cancer,cell,basal,node,cycl,margin

# Gastroenterology:
  . selected features: 89
  . top features: colon,cecum,scope,polyp,gallbladd,stomach,colonoscopi,esophagu,rectum,colonoscop

# ENT - Otolaryngology:
  . selected features: 57
  . top features: adenoid,ear,nasal,tonsil,otiti,nasopharynx,myringotomi,media,tonsillar,turbin
```

5.3.3.1. Multinomial Naïve Bayes:

Multinomial Naïve Bayes classifier fitted with alpha parameter set to 0.01 while the corpus vectorized using ngram range of (1,1) and the use of idf term.

label	precision	recall	f1-score	support
Cardiovascular / Pulmonary	68.00%	68.92%	68.46%	74
Neurology	61.82%	53.97%	57.63%	63
Urology	71.43%	74.47%	72.92%	47
General Medicine	38.79%	86.54%	53.57%	52
Radiology	21.88%	12.73%	16.09%	55
Psychiatry / Psychology	83.33%	90.91%	86.96%	11
Pediatrics - Neonatal	33.33%	7.14%	11.76%	14
Orthopedic	74.67%	78.87%	76.71%	71
Ophthalmology	100.00%	76.47%	86.67%	17
Obstetrics / Gynecology	73.33%	70.97%	72.13%	31
Hematology - Oncology	25.00%	16.67%	20.00%	18
Gastroenterology	93.10%	60.00%	72.97%	45
ENT - Otolaryngology	81.25%	68.42%	74.29%	19
accuracy		61.32%		
macro avg	63.53%	58.93%	59.24%	517
weighted avg	62.56%	61.32%	60.21%	517

Table 15 - Classification report of NB

The accuracy of Multinomial Naïve Bayes classifier improved from 59.96% to 61.32%. The model deals good with categories like “Urology”, “Psychiatry / Psychology”, “Orthopedic”, “Ophthalmology”, “Obstetrics / Gynecology”, “Gastroenterology” and “ENT - Otolaryngology” reaching an f-1 score more than 70%.

5.3.3.2. Logistic Regression:

Again, the corpus is vectorized using **Tfidf Vectorizer** with ngram range set to (1,1) and with the use of idf term as Grid Search decided as best parameters. Fitting Logistic Regression model with "newton-cg" set as solver and "l2" as penalty parameters, the classification report is as follows:

label	precision	recall	f1-score	support
Cardiovascular / Pulmonary	69.14%	75.68%	72.26%	74
Neurology	59.02%	57.14%	58.06%	63
Urology	73.47%	76.60%	75.00%	47
General Medicine	48.24%	78.85%	59.85%	52
Radiology	23.91%	20.00%	21.78%	55
Psychiatry / Psychology	85.71%	54.55%	66.67%	11
Pediatrics - Neonatal	50.00%	7.14%	12.50%	14
Orthopedic	67.47%	78.87%	72.73%	71
Ophthalmology	100.00%	76.47%	86.67%	17
Obstetrics / Gynecology	84.62%	70.97%	77.19%	31
Hematology - Oncology	30.77%	22.22%	25.81%	18
Gastroenterology	91.43%	71.11%	80.00%	45
ENT - Otolaryngology	87.50%	73.68%	80.00%	19
accuracy		63.44%		
macro avg	67.02%	58.71%	60.66%	517
weighted avg	64.21%	63.44%	62.60%	517

Table 16 - Classification report of Logistic Regression

Results are much better with Logistic Regression than Multinomial NB. The model achieved a slight increase of accuracy from 62.67% to 63.44%. Although again is visible that the model can correctly identify many categories, 7 out of 13 categories have f-1 score more than 70%.

5.3.3.3. Support Vector Machines

label	precision	recall	f1-score	support
Cardiovascular / Pulmonary	65.06%	72.97%	68.79%	74
Neurology	58.73%	58.73%	58.73%	63
Urology	70.00%	74.47%	72.16%	47
General Medicine	46.59%	78.85%	58.57%	52
Radiology	26.53%	23.64%	25.00%	55
Psychiatry / Psychology	75.00%	54.55%	63.16%	11
Pediatrics - Neonatal	75.00%	21.43%	33.33%	14
Orthopedic	72.86%	71.83%	72.34%	71
Ophthalmology	93.33%	82.35%	87.50%	17
Obstetrics / Gynecology	86.36%	61.29%	71.70%	31
Hematology - Oncology	25.00%	22.22%	23.53%	18
Gastroenterology	93.75%	66.67%	77.92%	45
ENT - Otolaryngology	88.24%	78.95%	83.33%	19
accuracy		62.28%		
macro avg	67.42%	59.07%	61.24%	517
weighted avg	64.49%	62.28%	62.14%	517

Table 17 - Classification report of SVM

5.3.3.4. Experiment Summary

Model	*Accuracy*	*Precision*	*Recall*	*f1-score*
Multinomial NB	61.32%	62.56%	61.32%	60.21%
Logistic Regression	63.44%	64.21%	63.44%	62.60%
SVM	62.28%	64.49%	62.28%	62.14%

Table 18 - Summary of "Dropping categories and Feature Selection" experiment

The best performance with this experiment is, again, Logistic Regression's classifier with an accuracy of 63.64%, precision weighted average 64.51%, recall weighted average 63.64% and f1-score weighted average 62.91%. The results are pretty good but there are some issues that must be solved. There is enough noise added mainly from categories like "General Medicine" and "Radiology". During the next experiment those categories will be dropped and an oversampling technique that handles imbalanced training datasets, will be presented as a way of improvement of classification models.

5.3.4. Oversampling with SMOTE & Feature Selection

In this experiment, "General Medicine" and "Radiology" category will be dropped as they add a lot of noise to the dataset. Also, the problem of imbalanced categories will be handled. The new dataset will be fed into SMOTE (Synthetic Minority Oversampling Technique) algorithm to be resampled. SMOTE aims to balance class distribution by randomly increasing minority class examples by replicating them. It generates virtual training records by linear interpolation for the minority class. These synthetic training records are generated by randomly selecting one or more of the k-nearest neighbors for each example in the minority class.

The initial number of records before oversampling was 2051 while after oversampling that number turned to 4081.

Category	***Counts before SMOTE***	***Counts after SMOTE***
Cardiovascular / Pulmonary	371	371
Orthopedic	355	371
Neurology	317	371
Urology	237	371
Gastroenterology	224	371
Obstetrics / Gynecology	155	371
ENT – Otolaryngology	96	371
Hematology- Oncology	90	371
Ophthalmology	83	371
Pediatrics - Neonatal	70	371
Psychiatry / Psychology	53	371

Table 19 - Categories counts before and after SMOTE

5.3.4.1. Multinomial Naïve Bayes:

Naïve Bayes classifier fitted on Tfidf vectorized texts with ngram range (1,1), X^2 test for feature selection ended up with 941 features out of 2500.

Dropping "General Medicine" and "Radiology" category combined with oversampling to generate artificial data points, proved to be an efficient technique as the accuracy of the model increased to 85.68%. The classifier seems to recognize documents from all the categories properly.

label	precision	recall	f1-score	support
Cardiovascular / Pulmonary	87.88%	77.33%	82.27%	75
Neurology	77.61%	70.27%	73.76%	74
Urology	91.18%	83.78%	87.32%	74
Psychiatry / Psychology	92.50%	100.00%	96.10%	74
Pediatrics - Neonatal	74.71%	86.67%	80.25%	75
Orthopedic	75.29%	86.49%	80.50%	74
Ophthalmology	97.33%	98.65%	97.99%	74
Obstetrics / Gynecology	98.36%	81.08%	88.89%	74
Hematology - Oncology	70.33%	86.49%	77.58%	74
Gastroenterology	90.91%	80.00%	85.11%	75
ENT - Otolaryngology	95.77%	91.89%	93.79%	74
accuracy		85.68%		
macro avg	86.53%	85.70%	85.78%	817
weighted avg	86.53%	85.68%	85.77%	817

Table 20 - Classification Report of Multinomial NB

5.3.4.2. Logistic Regression:

Logistic Regression classifier trained with "l2" penalty and "newton-cg" solver. The vectorized corpus's ngram range was set at (1,1) and the selected features were 922.

label	precision	recall	f1-score	support
Cardiovascular / Pulmonary	87.50%	84.00%	85.71%	75
Neurology	76.47%	70.27%	73.24%	74
Urology	91.30%	85.14%	88.11%	74
Psychiatry / Psychology	92.50%	100.00%	96.10%	74
Pediatrics - Neonatal	79.75%	84.00%	81.82%	75
Orthopedic	75.90%	85.14%	80.25%	74
Ophthalmology	96.05%	98.65%	97.33%	74
Obstetrics / Gynecology	97.10%	90.54%	93.71%	74
Hematology - Oncology	86.49%	86.49%	86.49%	74
Gastroenterology	90.79%	92.00%	91.39%	75
ENT - Otolaryngology	97.18%	93.24%	95.17%	74
accuracy		88.13%		
macro avg	88.28%	88.13%	88.12%	817
weighted avg	88.27%	88.13%	88.11%	817

Table 21 - Classification report of Logistic regression

The model archives an accuracy of 87.88% which is way better than the equivalent model of the previous experiment.

5.3.4.3. Support Vector Machines

label	precision	recall	f1-score	support
Cardiovascular / Pulmonary	91.30%	84.00%	87.50%	75
Neurology	75.76%	67.57%	71.43%	74
Urology	94.20%	87.84%	90.91%	74
Psychiatry / Psychology	93.67%	100.00%	96.73%	74
Pediatrics - Neonatal	86.42%	93.33%	89.74%	75
Orthopedic	73.86%	87.84%	80.25%	74
Ophthalmology	97.30%	97.30%	97.30%	74
Obstetrics / Gynecology	92.11%	94.59%	93.33%	74
Hematology - Oncology	86.11%	83.78%	84.93%	74
Gastroenterology	91.67%	88.00%	89.80%	75
ENT - Otolaryngology	98.59%	94.59%	96.55%	74
accuracy		88.98%		
macro avg	89.18%	88.99%	88.95%	817
weighted avg	89.18%	88.98%	88.95%	817

Table 22 - Classification report of SVM

5.3.4.4. Experiment Summary

Model	Accuracy	Precision	Recall	F1
Multinomial NB	85.07%	86.07%	85.07%	85.21%
Logistic Regression	88.13%	88.27%	88.13%	88.11%
SVM	88.98%	89.18%	88.98%	88.95%

Table 23 - Summary of "Oversampling with SMOTE & Feature Selection" experiment

The best performance with this experiment is of SVM classifier with an accuracy of 88.98%, precision weighted average 89.18%, recall weighted average 88.98% and f1-score weighted average 88.95%. The results of this experiment were undoubtedly good enough. Categories like “Cardiovascular / Pulmonary”, “Psychiatry / Psychology”, “Orthopedics”, “Ophthalmology”, “Obstetrics / Gynecology” and “ENT - Otolaryngology” have f1-score more than 80%, some of them are even more than 90% which means that SVM classifier does a perfect separation between each of them and the others.

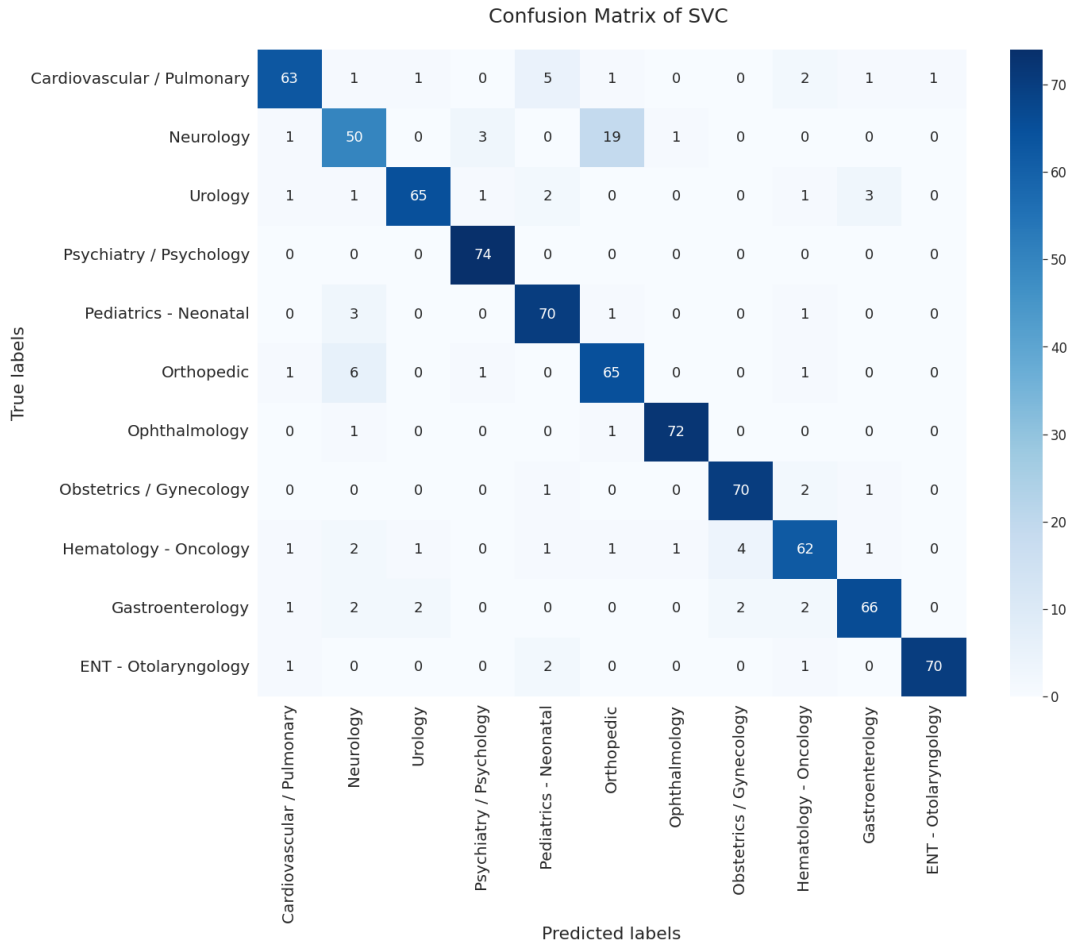


Figure 34 - Confusion Matrix of SVM classifier

5.4. Deep Learning Approach

In this section, deep learning algorithms and techniques will be presented as a way of text preprocessing and classification. All the gained knowledge from the previous section will be used during the preprocessing steps. The target variable finally consists of 11 unique categories to be predicted.

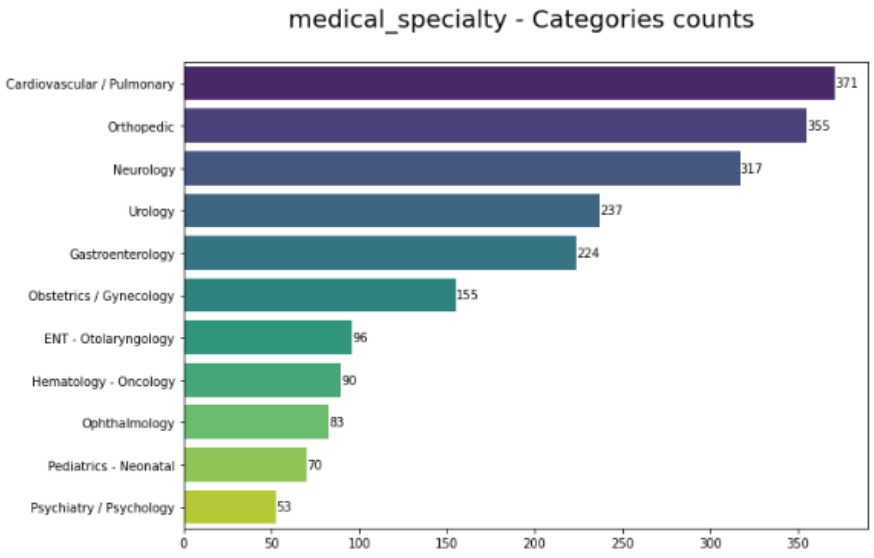


Figure 35 - Final medical specialties counts

Text before cleaning	Text after cleaning
2-D M-MODE: , ,1. Left atrial enlargement with left atrial diameter of 4.7 cm.,2. Normal size right and left ventricle.,3. Normal LV systolic function with left ventricular ejection fraction of 51%.,4. Normal LV diastolic function.,5. No pericardial effusion.,6. Normal morphology of aortic valve, mitral valve, tricuspid valve, and pulmonary valve.,7. PA systolic pressure is 36 mmHg.,DOPPLER: , ,1. Mild mitral and tricuspid regurgitation.,2. Trace aortic and pulmonary regurgitation.	left atrial enlargement left atrial diameter normal size right left normal lv systolic function left ventricular ejection fraction normal lv diastolic pericardial normal morphology aortic valve mitral valve tricuspid valve pulmonary pa systolic pressure doppler mild mitral tricuspid trace aortic pulmonary regurgitation

Table 24 - Before and After text cleaning

From *Table 24*, easily can be observed that all punctuation, stop words and non-alphabetical characters removed from each document while simultaneously, instead of using the porter stemmer to stem them the tokens inside each document were lemmatized and transformed into lowercase.

5.4.1. Deep Neural Network

The corpus of total 2051 documents vectorized using Tf-idf vectorizer with 5000 maximum features. The training sample consists of 1640 documents and 5000 features while the test set consists of 411 documents and 5000 features.

Network Achitecture

The input layer of the neural network consists of 5000 nodes. Four layers of 512 nodes followed by one dropout layer of 0.25 dropping rate form the hidden layers. Finally, the output layer comprised of 11 nodes and softmax activation function completes the architecture of the neural network. The total trainable parameters are 3,616,779.

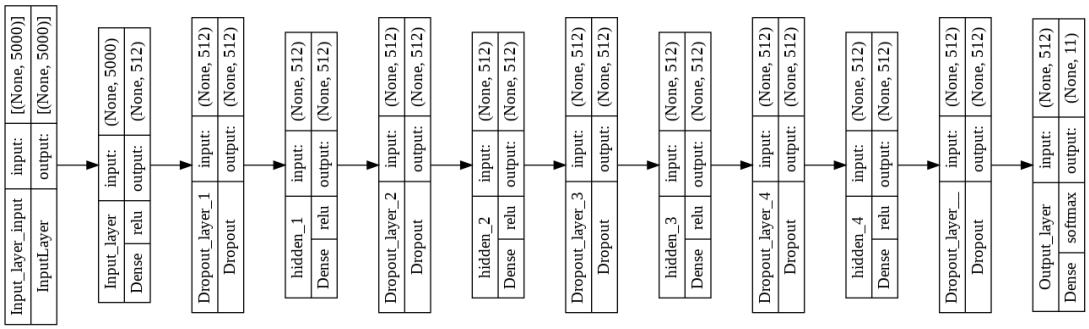


Figure 36 - DNN Architecture

Model Evaluation

label	precision	recall	f1-score	support
Cardiovascular / Pulmonary	91.36%	81.32%	86.05%	91
Neurology	73.08%	65.52%	69.09%	58
Urology	84.00%	85.71%	84.85%	49
Psychiatry / Psychology	66.67%	15.38%	25.00%	13
Pediatrics - Neonatal	100.00%	7.14%	13.33%	14
Orthopedic	72.62%	88.41%	79.74%	69
Ophthalmology	72.22%	86.67%	78.79%	15
Obstetrics / Gynecology	87.88%	90.63%	89.23%	32
Hematology - Oncology	28.13%	60.00%	38.30%	15
Gastroenterology	76.32%	78.38%	77.33%	37
ENT - Otolaryngology	78.95%	83.33%	81.08%	18
accuracy		76.16%		
macro avg	75.56%	67.50%	65.71%	411
weighted avg	79.09%	76.16%	75.28%	411

Table 25 - Classification report of DNN

The accuracy of the DNN is 76.16%. The remarkable note is that the network could not correctly predict most of the true labels of "Pediatrics – Neonatal" and "Psychiatry / Psychology" categories inside the test set. This is also visible from *Figure 36*. The diagonal cells of confusion matrix corresponding to those categories contain1 and 2 correctly classified documents, respectively.

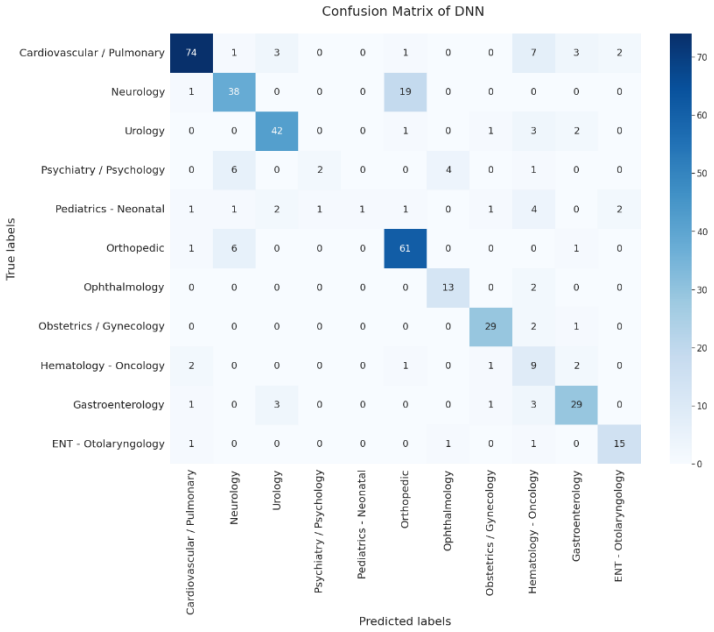


Figure 37 - Confusion Matrix of DNN

5.4.2. Convolutional Neural Network

Preprocessing

To use Convolutional Neural Networks for text classification some extra preprocessing is required. After tokenizing the documents, the ended up vocabulary size is 16,513 unique tokens. These tokens need to be presented as sequence of integers.

Natural Language Text	Text encoded into sequence of integers
left atrial enlargement left atrial diameter normal size right left normal lv systolic function left ventricular ejection fraction normal lv diastolic pericardial normal morphology aortic valve mitral valve tricuspid valve pulmonary pa systolic pressure doppler mild mitral tricuspid trace aortic pulmonary regurgitation	[3, 342, 1543, 3, 342, 966, 7, 224, 2, 3, 7, 1326, 752, 451, 3, 255, 886, 853, 7, 1326, 2492, 1293, 7, 3439, 331, 308, 913, 308, 1567, 308, 150, 3551, 752, 49, 2031, 91, 913, 1567, 1878, 331, 150, 1327]

Table 26 - Text encoded into sequence of integers

The word “left” is represented by the number 3 while the word “artial” by the number 342 and so on. All the corpus is being encoded into lists of encoded texts. As different texts have different lengths of words, the lists of encoded documents are also of different lengths. A deep learning model will often want input of uniform size which means that the documents of different lengths will raise a problem. All the documents must be of the same pre-defined length, this means that the longer documents must become shorter, and the shorter documents must become longer by adding a pre-defined numeric value (usually 0), This is going to be resolved using padding. Deciding the maximum length of each encoded document to be 100 and the padding value to be 0, the padded encoded documents will be as the example I the table below.

Before Padding	After Padding
[3, 342, 1543, 3, 342, 966, 7, 224, 2, 3, 7, 1326, 752, 451, 3, 255, 886, 853, 7, 1326, 2492, 1293, 7, 3439, 331, 308, 913, 308, 1567, 308, 150, 3551, 752, 49, 2031, 91, 913, 1567, 1878, 331, 150, 1327]	**[3, 342, 1543, 3, 342, 966, 7, 224, 2, 3, 7, 1326, 752, 451, 3, 255, 886, 853, 7, 1326, 2492, 1293, 7, 3439, 331, 308, 913, 308, 1567, 308, 150, 3551, 752, 49, 2031, 91, 913, 1567, 1878, 331, 150, 1327, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]**
[3, 255, 326, 224, 192, 1999, 675, 7, 192, 364, 3, 255, 752, 451, 410, 8449, 218, 886, 853, 6298, 95, 15, 410, 304, 3, 255, 2944, 940, 1527, 532, 326, 72, 316, 8449, 3, 255, 752, 451, 523, 3, 255, 8450, 1429, 95, 8, 262, 3, 342, 49, 95, 2031, 3, 1061, 410, 954, 2, 1061, 2, 579, 675, 331, 441, 410, 331, 308, 410, 1789, 91, 331, 308, 185, 1790, 331, 308, 30, 71, 5865, 1405, 12108, 1527, 1568, 1527, 913, 1718, 1051, 778, 3087, 3173, 587, 1273, 913, 308, 3087, 91, 913, 1567, 308, 410, 7, 1878, 1567, 1327, 361, 150, 20, 398, 218, 150, 20, 752, 49, 541, 218, 2, 342, 49, 150, 308, 410, 7, 1878, 150, 1293, 648, 3088, 83, 1791, 2031, 1569, 596, 717, 5187, 7440, 1544, 6799, 296, 747, 4129, 829, 5495, 2537, 496, 191, 3821]	**[8, 262, 3, 342, 49, 95, 2031, 3, 1061, 410, 954, 2, 1061, 2, 579, 675, 331, 441, 410, 331, 308, 410, 1789, 91, 331, 308, 185, 1790, 331, 308, 30, 71, 5865, 1405, 12108, 1527, 1568, 1527, 913, 1718, 1051, 778, 3087, 3173, 587, 1273, 913, 308, 3087, 91, 913, 1567, 308, 410, 7, 1878, 1567, 1327, 361, 150, 20, 398, 218, 150, 20, 752, 49, 541, 218, 2, 342, 49, 150, 308, 410, 7, 1878, 150, 1293, 648, 3088, 83, 1791, 2031, 1569, 596, 717, 5187, 7440, 1544, 6799, 296, 747, 4129, 829, 5495, 2537, 496, 191, 3821]**

Table 27 - Padding sequences into 100 maximum length

The input of the model is going to be each of these encoded documents of 100 length. Each token represented by an integer is going to be transformed into a vector of a specific size. The values of the vectors are real numbers called weights and are assigned throughout an embedding layer which assign numbers close to each other depending on the similarity of the tokens. The actual input of the CNN is an array of (100, vector size) shape. The vector size of the embedding layer chosen for the model is 300.

Network Achitecture

As mentioned above the first layer of the CNN model is an embedding layer with input size 100 and the output size is an array of size (100,300). Following the embedding layer, a 1-dimensional convolution layer with 80 filters ss assigned to the model architecture. Each filter's kernel size is 5. A Global Max-Pooling layer is assigned to the convolution and one dense layer with 64 nodes and 0.2 dropping rate dropout layer follows.

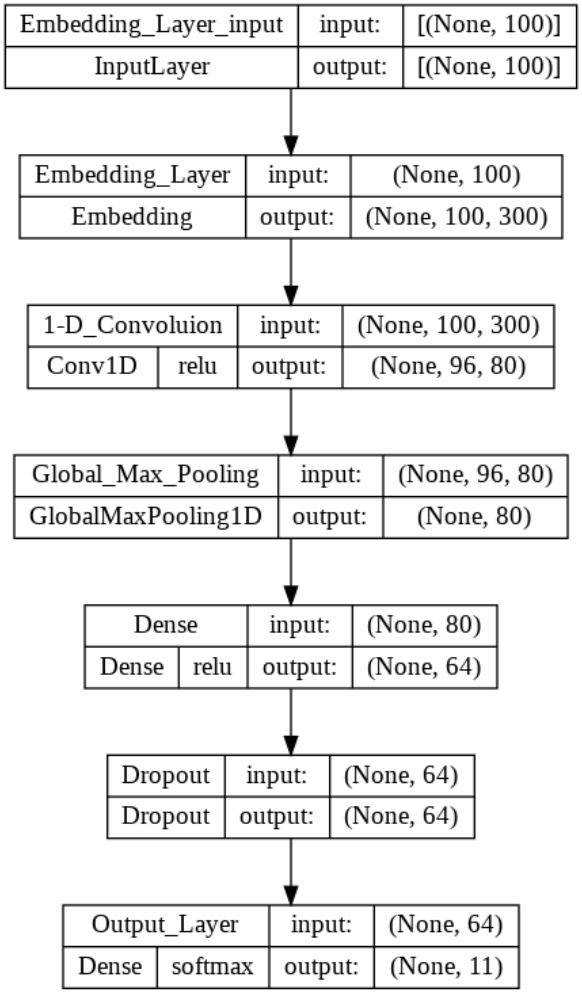


Figure 38 - CNN Architecture

Model Evaluation

label	precision	recall	f1-score	support
Cardiovascular / Pulmonary	74.76%	84.62%	79.38%	91
Neurology	64.29%	46.55%	54.00%	58
Urology	70.69%	83.67%	76.64%	49
Psychiatry / Psychology	100.00%	46.15%	63.16%	13
Pediatrics - Neonatal	50.00%	28.57%	36.36%	14
Orthopedic	65.82%	75.36%	70.27%	69
Ophthalmology	100.00%	80.00%	88.89%	15
Obstetrics / Gynecology	88.24%	93.75%	90.91%	32
Hematology - Oncology	33.33%	33.33%	33.33%	15
Gastroenterology	60.00%	64.86%	62.34%	37
ENT - Otolaryngology	78.57%	61.11%	68.75%	18
accuracy		70.32%		
macro avg	71.43%	63.45%	65.82%	411
weighted avg	70.55%	70.32%	69.53%	411

Table 28 - Classification report of CNN

The CNN model's predictions are less accurate than DNN's. Although The CNN model can correctly predict some more of the documents that belong to "Pediatric - Neonatal" category.

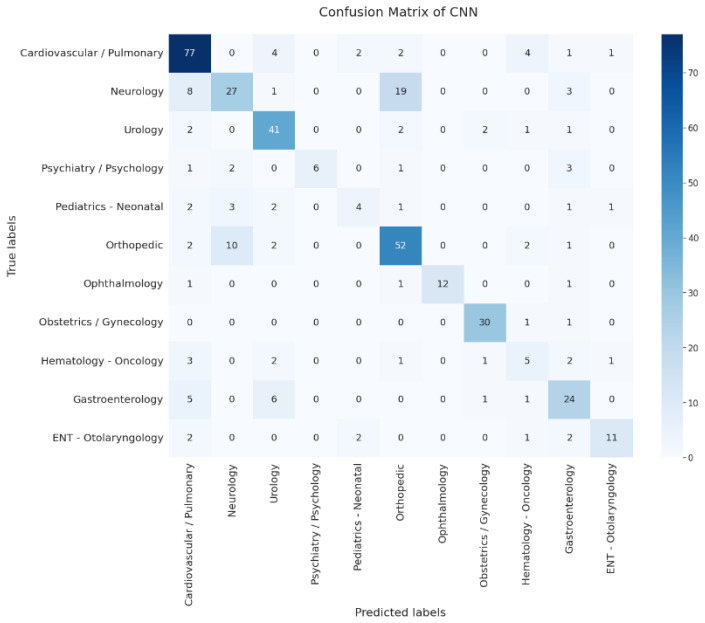


Figure 39 - Confusion Matrix of CNN

5.4.3. Convolutional Neural Network with Glove Embeddings

label	precision	recall	f1-score	support
Cardiovascular / Pulmonary	81.32%	81.32%	81.32%	91
Neurology	52.24%	60.34%	56.00%	58
Urology	63.08%	83.67%	71.93%	49
Psychiatry / Psychology	100.00%	46.15%	63.16%	13
Pediatrics - Neonatal	42.86%	21.43%	28.57%	14
Orthopedic	71.23%	75.36%	73.24%	69
Ophthalmology	92.31%	80.00%	85.71%	15
Obstetrics / Gynecology	85.29%	90.63%	87.88%	32
Hematology - Oncology	28.57%	26.67%	27.59%	15
Gastroenterology	80.00%	64.86%	71.64%	37
ENT - Otolaryngology	90.91%	55.56%	68.97%	18
accuracy		70.56%		
macro avg	71.62%	62.36%	65.09%	411
weighted avg	71.71%	70.56%	70.20%	411

Table 29 -Classification report of CNN wih Glove Embeddings

Using pre-trained Glove word vectors as embedding layer in CNN did not bring any significant change in the overall classification. However, CNN with pre-trained Glove embedding dealt with some categories in a more successful way. Categories like “Cardiovascular / Pulmonary”, “Neurology”, “Orthopedic”, “Gastroenterology”, “Gastroenterology” and “ENT - Otolaryngology” got higher f-1 score than the CNN without pre-trained Glove embeddings.

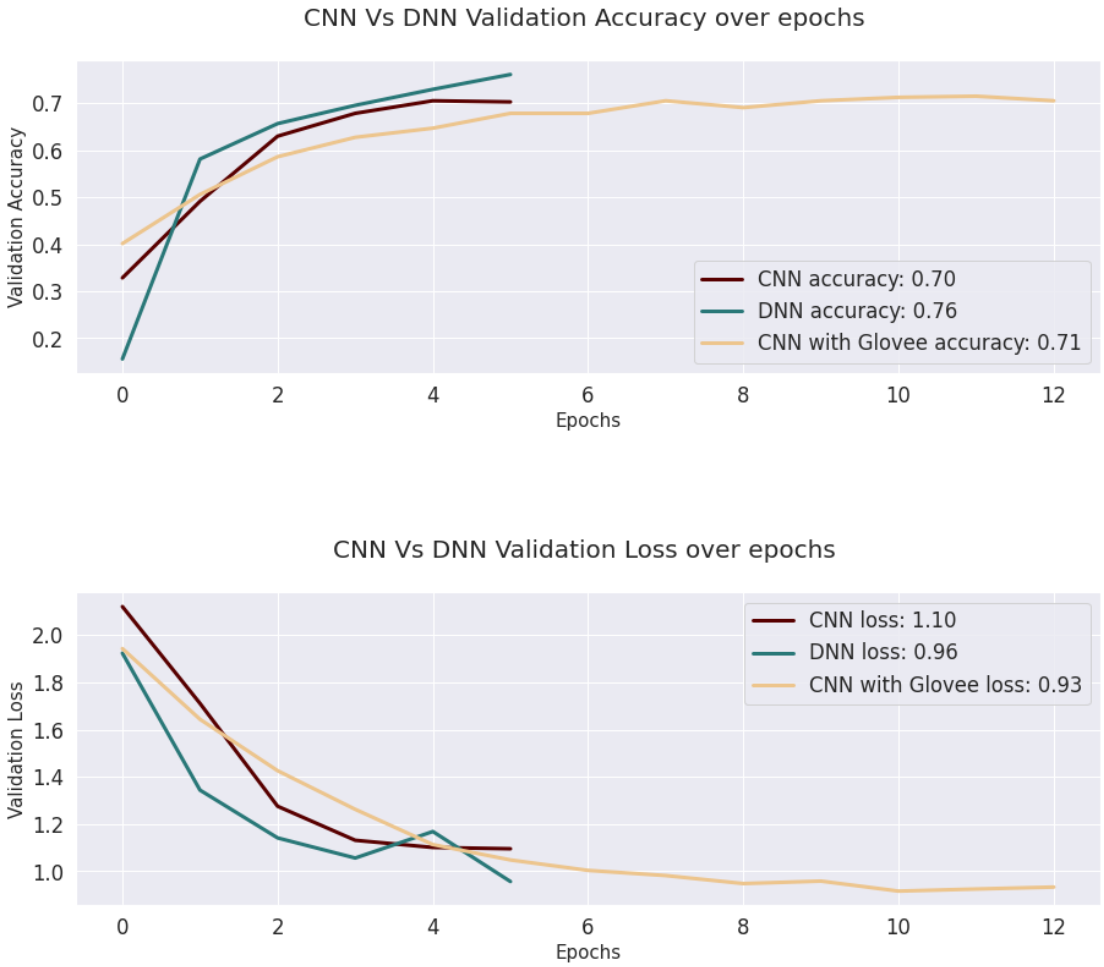


Figure 40 - Models performance over epochs

The best performing model, according to validation loss and validation accuracy (*Figure 40*), is of the Deep neural network which was trained over 6 epochs add achieved an accuracy of 0.76 and validation loss 0.96.

5.4.4. Text Generation for balancing minority categories

In this section one LSTM model is going to be presented as a technique for artificial test generation. With this method the data imbalance problem is going to be resolved.

For the generation of artificial texts, extra preprocessing steps are required to prepare the input of the model. The model will take as an input a sequence of 15 tokens, and it will try to predict the next token. The features and the labels will be extracted from the texts in a recurrent way. An algorithm will walk through each text and will extract the first 15 tokens as features and the 16th feature as label. This will be done for all consecutive 15-length tokens as shown above:

For example, the following sentence represent a text:

"Natural language processing refers to the branch of computer science and more specifically, the branch of artificial intelligence or AI concerned with giving computers the ability to understand text and spoken words as human beings can"

Features	Labels
Natural language processing refers to the branch of computer science and more specifically, the branch"	*of*
language processing refers to the branch of computer science and more specifically, the branch of	*artificial*
processing refers to the branch of computer science and more specifically, the branch of artificial	*intelligence*
refers to the branch of computer science and more specifically, the branch of artificial intelligence	*or*
to the branch of computer science and more specifically, the branch of artificial intelligence or	*AI*
the branch of computer science and more specifically, the branch of artificial intelligence or AI	*concerned*
branch of computer science and more specifically, the branch of artificial intelligence or AI concerned	*with*
of computer science and more specifically, the branch of artificial intelligence or AI concerned with	*giving*
computer science and more specifically, the branch of artificial intelligence or AI concerned with giving	*computers*
science and more specifically, the branch of artificial intelligence or AI concerned with giving computers	*the*
and more specifically, the branch of artificial intelligence or AI concerned with giving computers the	*ability*
more specifically, the branch of artificial intelligence or AI concerned with giving computers the ability	*to*

specifically, the branch of artificial intelligence or AI concerned with giving computers the ability to	*understand*
the branch of artificial intelligence or AI concerned with giving computers the ability to understand	*text*
branch of artificial intelligence or AI concerned with giving computers the ability to understand text	*and*
of artificial intelligence or AI concerned with giving computers the ability to understand text and	*spoken*
artificial intelligence or AI concerned with giving computers the ability to understand text and spoken	*words*
intelligence or AI concerned with giving computers the ability to understand text and spoken words	*as*
or AI concerned with giving computers the ability to understand text and spoken words as	*human*
AI concerned with giving computers the ability to understand text and spoken words as human	*beings*
concerned with giving computers the ability to understand text and spoken words as human beings	*can*

Table 30 - Feature and Labels extraction from given text.

This will be applied to all texts after they have been cleaned and all the preprocessing steps are done. Short texts, those that contain less than 30 words, are not going to be included in the process of feature and label creation.

After the creation of the features and the labels, the tokens will be encoded into integers and the data are going to be fed into the LSTM model.

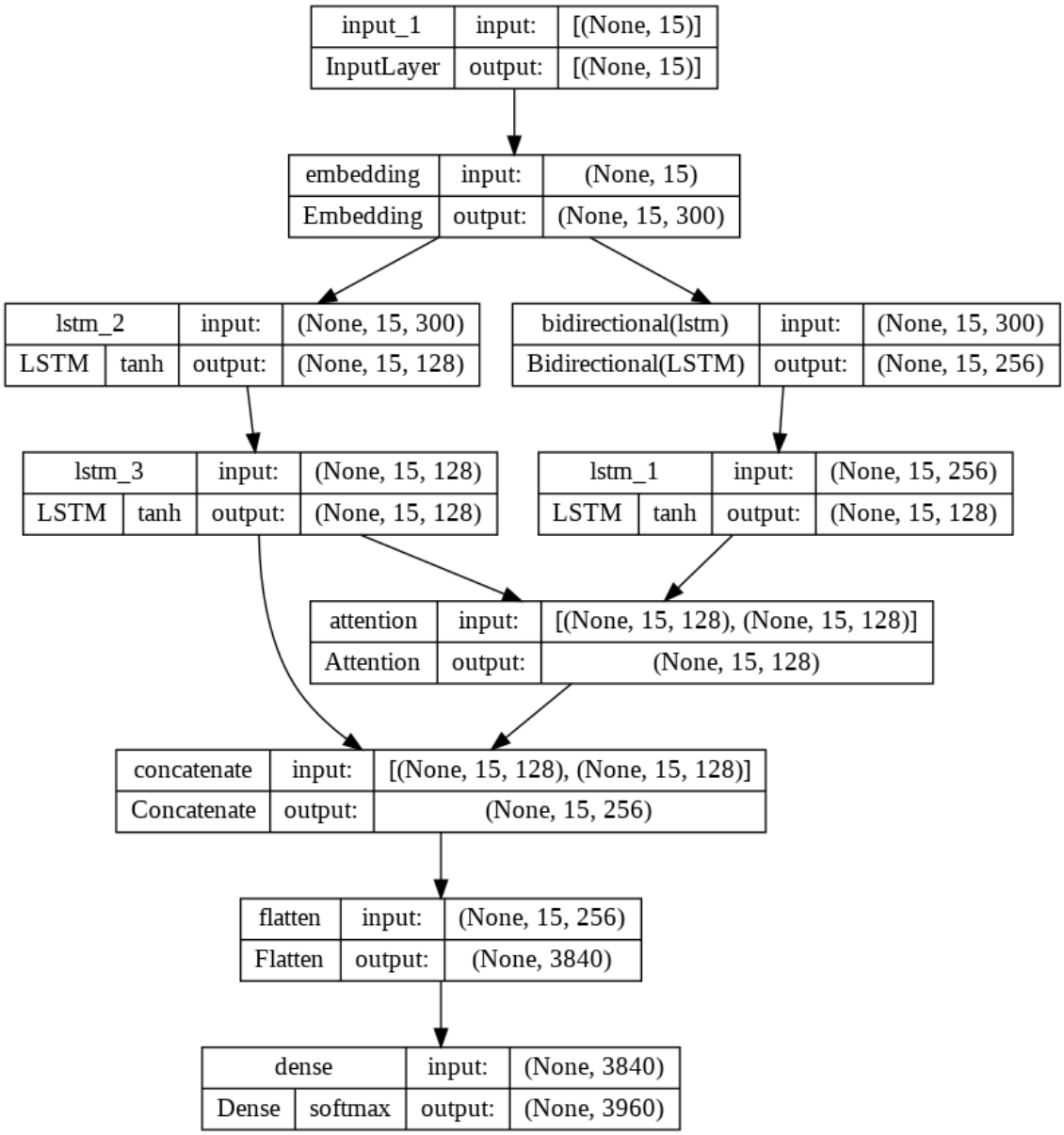


Figure 41 - LSTM model for text generation

After training the model with the created features and labels, text generation process will take place.

The lengths of the generated texts are going to be randomly selected from 50 to 150 words maximum. After defining the length of the generated text, an encoded text is randomly selected from the corpus and a sequence of 15 consecutive words inside of it is extracted from any random position. This 15 word sequence is going to be the starting point for generating the artificial text. The model that was trained before, is

going to be called as many times as the length of generated text is defined. The model will take as an input the 15 sequences that randomly were selected and will predict the 16th word. That 16th word constitutes the first word of the artificial text. During the next iteration, the first element of the 15-length sequence is dropped and the 16th element that was predicted in the previous iteration is being assigned to the sequence, know the new 15-length sequence is being fed into the model and again the 16th is predicted and assigned next to the first prediction and so on. This process keeps going until the maximum generated text length is reached.

The process mentioned above generates one artificial text. To handle the problem of imbalanced categories, the corpus is going to be split into 11 sub corpuses each for the 11unique categories. Then the LSTM model is going to be trained with each of the subcategories. Having the model trained on each subcategory a repeating structure will generate N artificial texts of random length between 50 to 150 words.

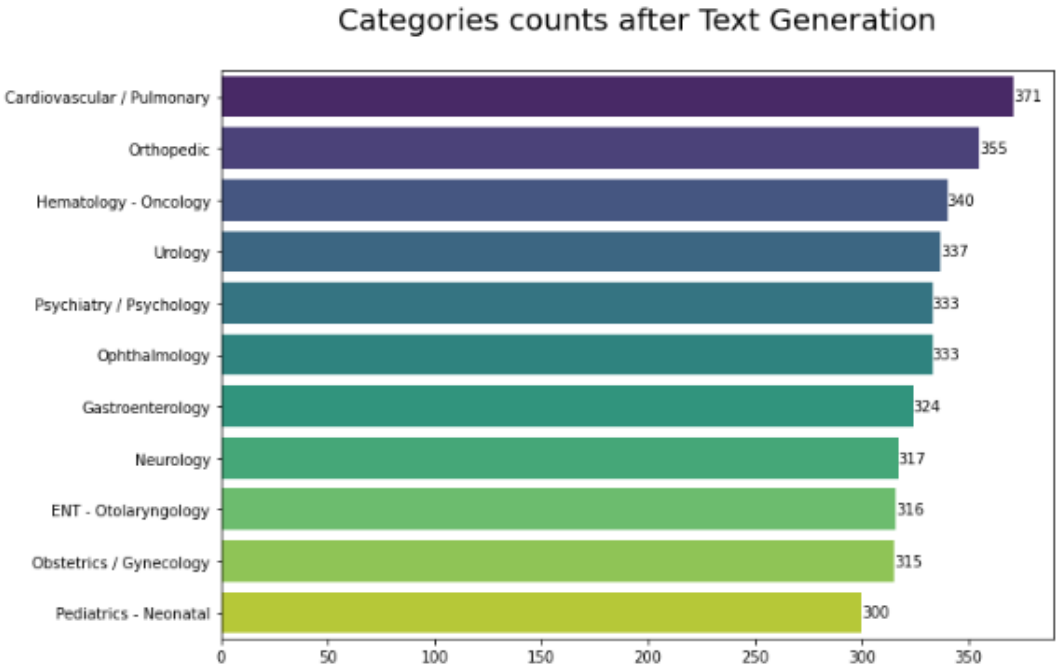


Figure 42 - Text categories counts after text generation

5.4.5. Deep Neural Network with oversampled data

label	precision	recall	f1-score	support
Cardiovascular / Pulmonary	89.53%	85.56%	87.50%	90
Neurology	77.14%	62.07%	68.79%	87
Urology	90.00%	83.51%	86.63%	97
Psychiatry / Psychology	79.38%	90.59%	84.62%	85
Pediatrics - Neonatal	89.19%	86.84%	88.00%	76
Orthopedic	67.31%	90.91%	77.35%	77
Ophthalmology	88.33%	74.65%	80.92%	71
Obstetrics / Gynecology	90.80%	94.05%	92.40%	84
Hematology - Oncology	89.55%	83.33%	86.33%	72
Gastroenterology	82.47%	90.91%	86.49%	88
ENT - Otolaryngology	96.20%	90.48%	93.25%	84
accuracy	84.85%			
macro avg	85.45%	84.81%	84.75%	911
weighted avg	85.50%	84.85%	84.81%	911

Table 31 - Classification report of DNN on generated texts

The DNN which was fitted on the mixed dataset which includes the artificial texts that were generated using the LSTM model, seems to perform well reaching an accuracy almost 85%. The weighted average of F-1 scores is 84.81. The categories that did not perform well during the previous experiments such as “Neurology”, “Pediatrics - Neonatal” etc., are now well performing categories. This is because of the extra training and testing samples that were created after the oversampling of the dataset. The testing samples increased from 411 to 911, this is an increase of 122% by the initial testing samples.

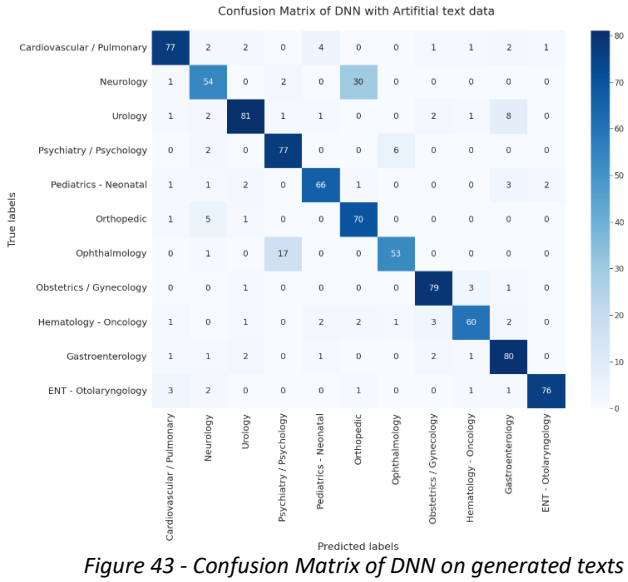


Figure 43 - Confusion Matrix of DNN on generated texts

The improvement is also visible from the confusion matrix as the diagonal includes almost all the data points. Very few misclassifications are spotted and those are for the categories of "Neurology", "Psychiatry - Psychology" and "Ophthalmology". Many categories that suffered from misclassification seem to have dealt with this as more texts were generated to support the classification. One good example is the category of "Pediatrics – Neonatal".

5.4.6. Convolutional Neural Network with oversampled data

label	*precision*	*recall*	*f1-score*	*support*
Cardiovascular / Pulmonary	83.58%	62.22%	71.34%	90
Neurology	66.20%	54.02%	59.49%	87
Urology	81.00%	83.51%	82.23%	97
Psychiatry / Psychology	63.89%	81.18%	71.50%	85
Pediatrics - Neonatal	64.84%	77.63%	70.66%	76
Orthopedic	62.92%	72.73%	67.47%	77
Ophthalmology	71.15%	52.11%	60.16%	71
Obstetrics / Gynecology	89.29%	89.29%	89.29%	84
Hematology - Oncology	60.44%	76.39%	67.48%	72
Gastroenterology	84.42%	73.86%	78.79%	88
ENT - Otolaryngology	88.89%	85.71%	87.27%	84
accuracy		73.77%		
macro avg	74.24%	73.51%	73.24%	911
weighted avg	74.80%	73.77%	73.67%	911

Table 32 - Classification report of CNN on generated texts

The Convolutional Neural Network did not perform that well in comparison with the Deep Neural Network. The oversampling of the dataset had no impact on the overall model's accuracy. The F-1 scores of some categories such as "Pediatrics Neonatal", "Neurology", "Obstetrics / Gynecology", "ENT - Otolaryngology" and "Hematology - Oncology" improved but the others remained the same or did not improve. This is visible also from the following confusion matrix. Many more misclassifications are visible from almost all the categories.

5.4.7. Summary

Model	*Accuracy*	*Precision*	*Recall*	*f-1 score*
DNN	76.16%	79.09%	76.16%	75.28%
CNN	70.32%	70.55%	70.32%	69.53%
CNN-Glovee	70.56%	71.71%	70.56%	70.20%
DNN-Oversampled	84.85%	85.50%	84.85%	84.81%
CNN-Glovee-Oversampled	73.77%	74.80%	73.77%	73.67%

Table 33 - Summary of all DL experiments

The best performing model is the deep neural network that was fitted on the dataset that also included the artificial text data which were generated from the LSTM model. That model has an accuracy of 84.85%. The weighted average of the precision is 85.5%, the recall is 84.85% and of F-1 score is 84.81%. Considering that the target variable is a multiclass category, The performance of the model is very satisfactory.

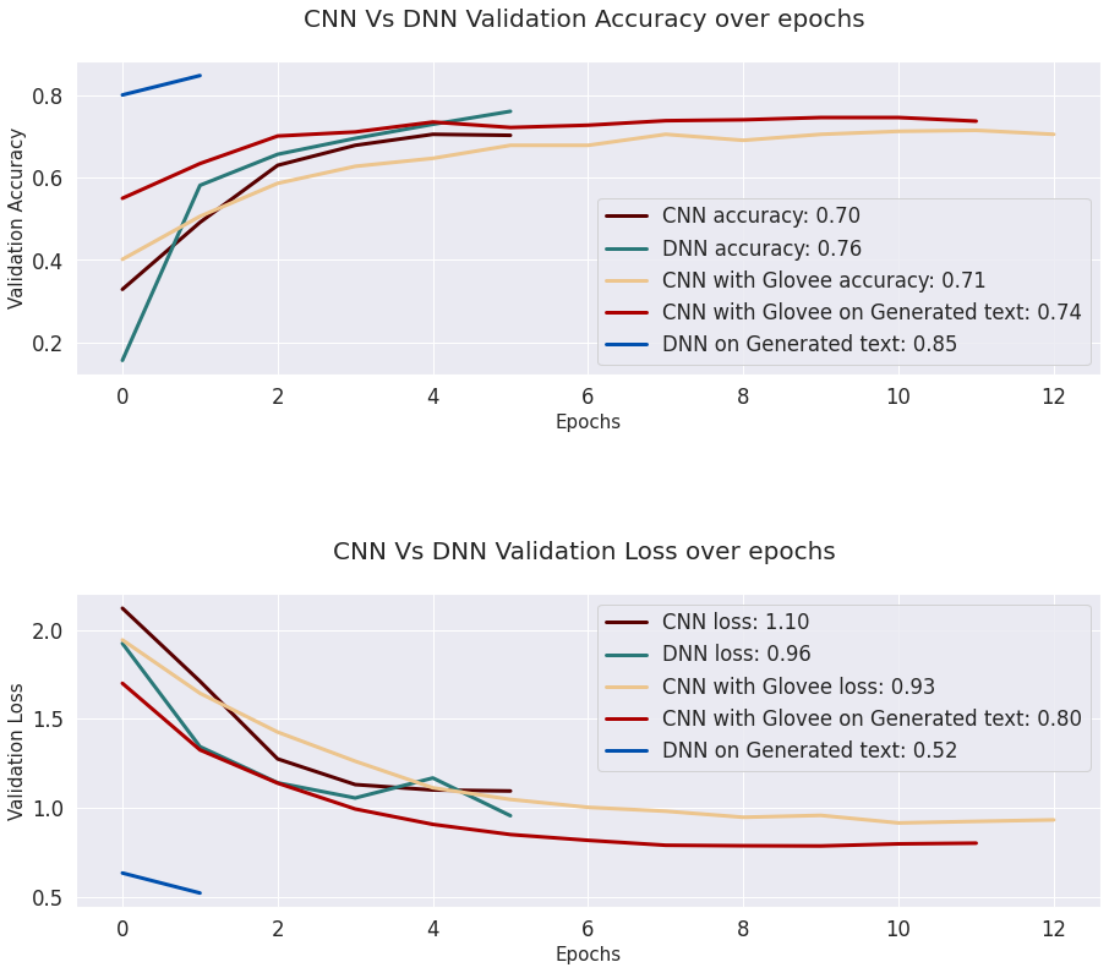


Figure 44 - All models performance over epochs

Figure 43 shows the validation accuracy and loss, over the epochs, for each one of all the DL models. All the models except the candidate perform almost in the same way. The candidate model has a validation loss of 0.52 which is almost half of the others validation loss.

6. Chest X-rays Covid-19 Classification

In this chapter image classification techniques will be presented as a way to detect Covid-19 infection by analyzing human chest x-rays.

6.1. Dataset

The used dataset was extracted from Kaggle and refers to chest x-rays which were collected from publicly released GitHub repository by the University of Montreal professors. The dataset consists of a total of 317 images belonging to three classes, “Normal”, “Covid-19” and “Viral Pneumonia”.

	Train set	***Test set***	***Totals***
Normal	70	20	90
Covid-19	111	26	137
Viral Pneumonia	70	20	90
Totals	251	66	317

Table 34 - Dataset Description

For the current analysis the “Viral Pneumonia” images were not taken into consideration, and they have been dropped. The classification task turned into binary classification.

	Train set	***Test set***	***Totals***
Normal	70	20	90
Covid-19	111	26	137
Totals	181	46	227

Table 35 - Final Dataset Description

6.2. Data Exploration and Preprocessing

The images that were extracted from the source seem to be well separated between the classes although the x-ray images are of different sizes.

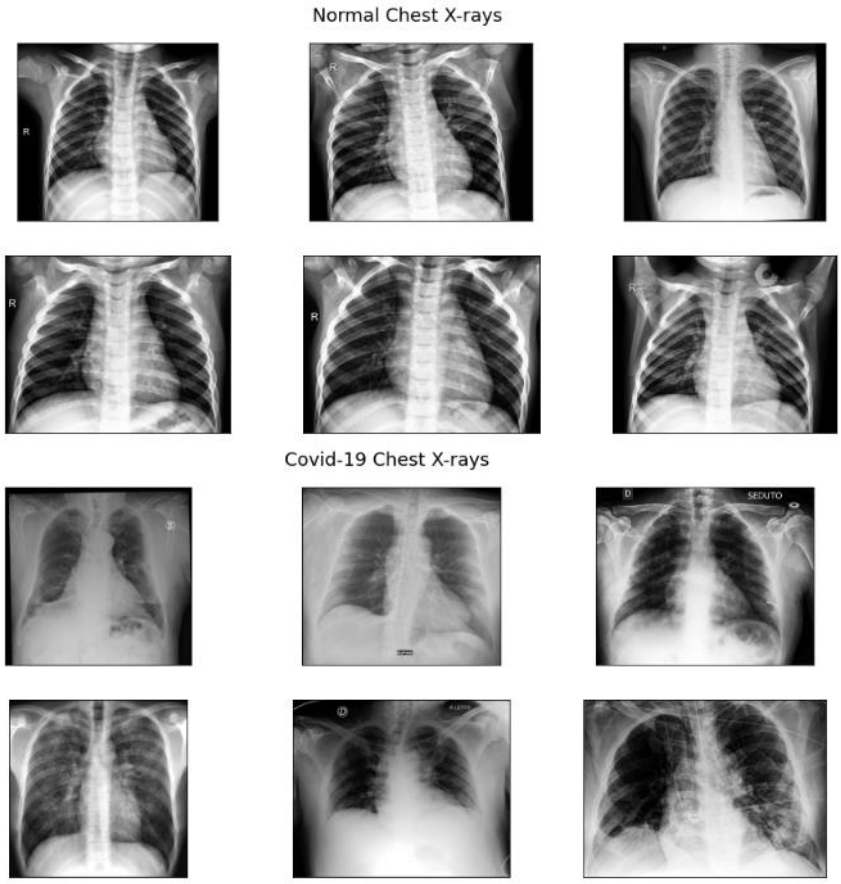


Figure 45 - Images of Normal and Covid-19 X-rays

Deep learning requires all the inputs to be of the same size. For that reason and for less complexity and computational time, al the images resized to 224×224 pixel images. The ended up training and testing samples converted into 4 dimensional arrays of (181, 224, 224, 3) and (46, 224, 224, 3) shape respectively. The 1st dimension refers to the number of samples, the 2nd, and the 3rd to the image size while the 4th dimension refers to the three color channels of each pixel.

6.3. Deep Neural Network

To train a deep neural network with images, transformed to n dimensional array, the input must be reshaped into two - dimensional array where the first dimension is the number of samples and the second is the product of the total pixels and the color channels. Hence, the training set which is of (181, 224, 224, 3) shape will be reshaped into (181, 150528) and the test set into (46, 150528).

The multilayer perceptron, between input layer and output layer, contains 6 hidden layers each of them followed by a dropout layer with 10% drop out percentage, as shown in *figure 46*:

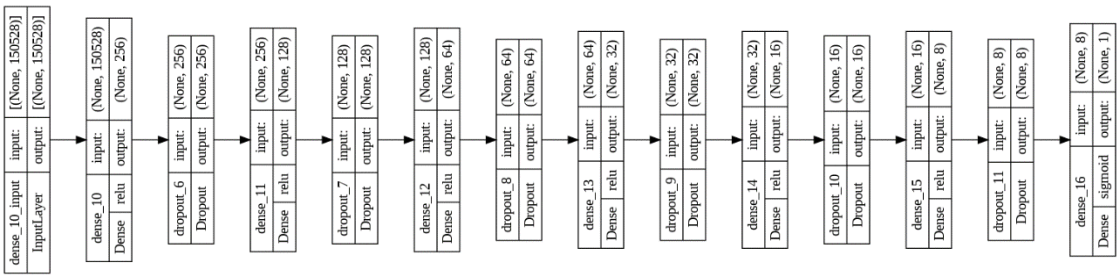


Figure 46 - DNN Architecture

label	precision	recall	f1-score	support
Normal	94.44%	85.00%	89.47%	20
Covid-19	89.29%	96.15%	92.59%	26
accuracy		91.30%		
macro avg	91.87%	90.58%	91.03%	46
weighted avg	91.53%	91.30%	91.24%	46

Table 36 - Classification report of DNN

The accuracy of the model is 91.30% which means that, approximately, 9 out of 10 chest x-rays are classified correctly. It is important to detect correctly as many Covid-19 cases as possible. If positive is set to be the presence of Covid-19 infection, it is important to minimize False Negative cases. Hence, the metric that is more important is the sensitivity or recall of the model to predict Covid-19 cases. The model has a recall of 91.30% which means that it was able to detect 96.15% of the x-rays with covid infection.

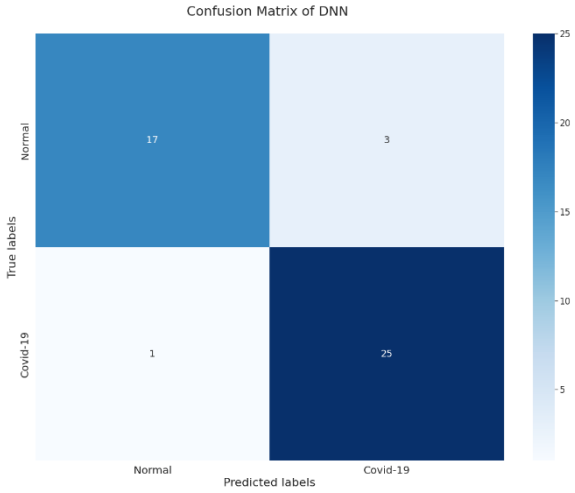


Figure 48 - Confusion matrix of DNN

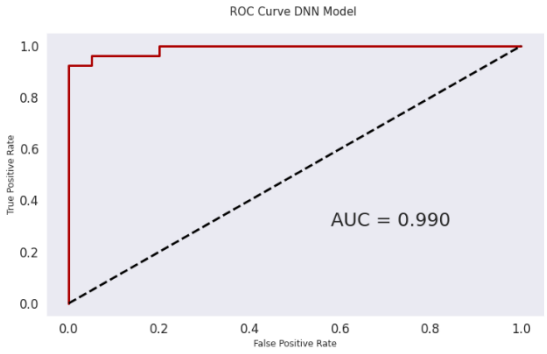


Figure 47 - Receiver Operating Characteristic Curve of DNN

6.4. VGG16 Network

VGG16, also known as OxfordNet, is a Convolutional Neural Network architecture developed by Visual Geometry Group of Oxford University. The model that was proposed by Andrew Zisserman and Karen Simonyan (2013) and published in a paper called "*Very Deep Convolutional Networks for Large-Scale Image Recognition*" was trained on ImageNet database which contains more than 14 million images of total 1000 classes. VGG16 can achieve a test accuracy of 92.7%.

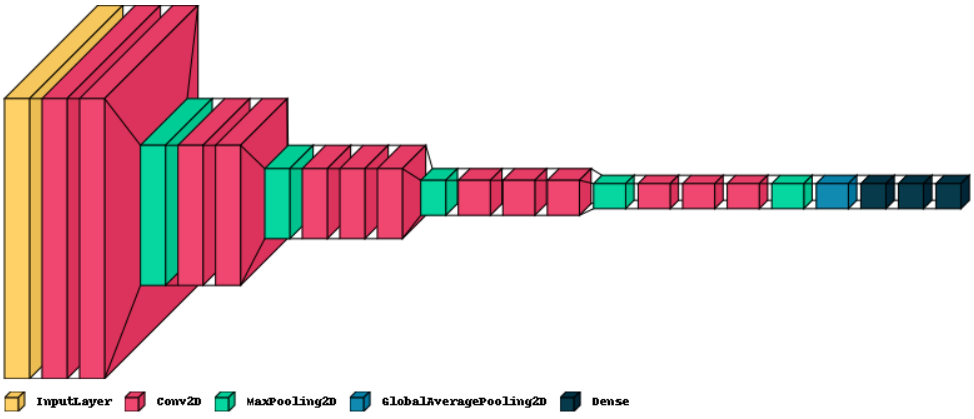


Figure 49 - VGG16 Architecture

The model architecture as shown in *Figure 49*, consists of 16 weighted layers of which, thirteen are Convolution layers and three Dense layers. Also, it has five Max Pooling layers which sum up 21 layers in total. The input of the network is of size 224×224 with 3 RGB channels image. The Convolutional layer uses 3×3 convolution

filters with stride 1 and the same max pooling layer of 2×2 filter of stride 2. In this experiment the pre-trained VGG16 will be used with the difference of adding 3 more dense layers at the output of it. This technique is known as Transfer Learning. Transfer Learning is a Machine Learning technique where a pre-trained model is used as the starting point of the training of another model. This optimization task promises rapid progress while training the second model and can achieve significantly higher performance than training from scratch a model with small amount of data. Transfer Learning is very common nowadays as most image and NLP are not implemented from scratch anymore.

label	precision	recall	f1-score	support
Normal	95.24%	100.00%	97.44%	20
Covid-19	100.00%	96.15%	98.11%	26
accuracy		97.83%		
macro avg	97.62%	98.03%	97.80%	46
weighted avg	97.93%	97.83%	97.83%	46

Table 37 - Classification report of VGG16

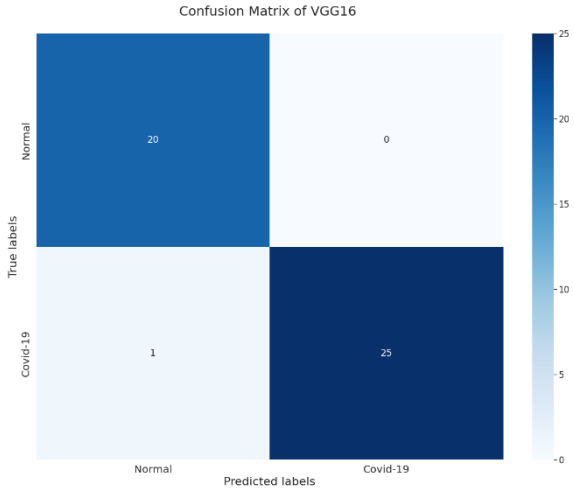


Figure 50 - Confusion Matrix of VGG16

The model accuracy is 97.83% which is very good. The recall of the Covid-19 class is 96.15% which means that the model can identify almost all of the x-rays of infected patients.

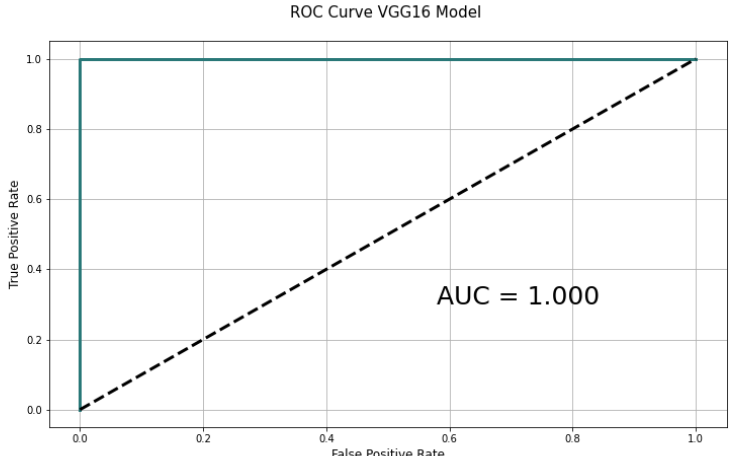


Figure 51 - Receiver Operating Characteristic Curve of VGG16

6.5. Summary

The best performance is achieved from the VGG16 model with the following performance metrics:

Model	*accuracy*	*Precision*	*Recall*	*F-1 Score*
DNN	91.30%	91.53%	91.30%	91.24%
VGG16	97.83%	97.93%	97.83%	97.83%

Table 38 - Models Summary Table

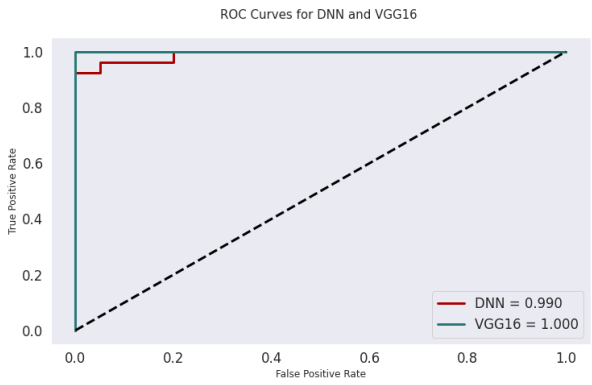


Figure 53 - Receiver Operating Characteristic of models

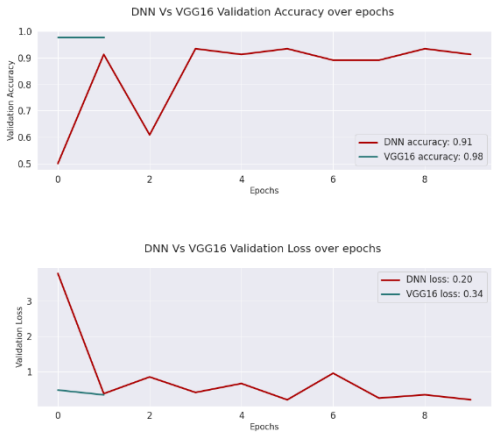


Figure 52 - Models performance over training

In *Figure 52* the validation loss and accuracy for both models are presented over the epochs. Validation loss of VGG16 is 0.34 while the validation accuracy is 0.98 in just two epochs training. This is normal as the weights of the VGG16 were not trained from scratch, but the pre-trained weights were used.

7. Conclusion

The aim of this thesis was to develop Machine Learning and Deep Learning models to classify medical text documents and medical images as well, into pre-defined categories. The purpose is to present a concrete solution which, combined with the medical staff, would transform that time consuming task into a handy and time saving one.

During the analysis a dataset of 4999 total medical transcriptions belonging to 40 distinct categories was taken as a starting point. The dataset is quite noisy, and the categories are imbalanced over the transcriptions. A lot of medical transcriptions overlap across the categories. After all the text cleaning, vectorization and SMOTE algorithm for data balancing, three Machine Learning algorithms were tested and evaluated over multiple experiments. SVM algorithm that achieved a classification accuracy of 88.98% and 0.89 f-1 score brought the best classification performance.

Many categories were dropped as overlapped medical specialties. This action reduced the number of unique classes increasing the performance of the algorithms despite the loss of significant number of documents. The same classification is proposed to be split into two different classifications in the future. One primary classification algorithm that classifies the text documents into a medical specialty and a secondary that classifies each medical specialty's document into a subspecialty.

A model to recognize and classify chest X-rays into two categories was presented, in the context of image classification. The images were discretized into two categories. Patients with normal chest X-rays and patients whose lungs were infected with Covid-19 virus. The task was carried out using two neural network architectures. A Multilayer Perceptron which achieved an accuracy of 91.3% and 0.96 recall of the positive class (Covid-19). The results were quite good, but the best performance achieved from a pre-trained CNN, also known as VGG16 network, which achieved an accuracy of 98.03% and 0.96 recall of the positive class.

This task can go one step further in the future, in the context of computer vision, by splitting it into two tasks. After the classification of the Covid-19 cases one secondary model could recognize the type and the level of the damage that has been caused by the infection to the patient.

8. Appendix

8.1. Medical Text Classification Code

Imports

In [1]:
```
# !pip install matplotlib==3.5.3 --quiet
# !pip install pyngrok --quiet
# !pip install mlflow --quiet
```

In [2]:
```
import pandas as pd
import numpy as np

import seaborn as sns
import matplotlib.pyplot as plt
import datetime


import os
os.chdir("drive/MyDrive/Github/Medical_Text_and_Image_Classification_v1")

import warnings
warnings.filterwarnings("ignore")

from functions.preprocessing import clean_transcription

from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from functions.utils import get_target_variable_mapping
from functions.modeling.modeling import produce_model

from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import MultinomialNB
```

In [3]:
```
pd.set_option('display.max_rows', 100)
pd.set_option('display.max_columns', 100)
```

Notebook parameters

In [4]:
```
export=True
verbose = True
```

In [5]:
```
# Folders
export_folder = 'Run_{}'.format(str(datetime.datetime.now()).replace(' ','_')[:19])

# Export directory
export_directory = './exports/'+export_folder

if export:
  if not os.path.exists(export_directory):
    os.makedirs(export_directory)
```

Impot And Explore the dataset

In [6]:
```
data = pd.read_csv("Data/mtsamples.csv",delimiter=',',index_col=0)
data.head()
```

In [7]:
```
print("dataset_rows: ", data.shape[0])
print("dataset_columns: ", data.shape[1])
print("column Names: ", list(data.columns))
```

In [8]:
```
# Dropping unnecessary columns
data = data.drop(['description', 'sample_name', 'keywords' ], axis=1)[["transcription"
data
```

medical_specialty - Visualization of categories counts

In [9]:
```
# plt.figure(figsize=(20,15))

# ax = sns.barplot(x=data['medical_specialty'].value_counts().values,
#             y=data['medical_specialty'].value_counts().index,
#             palette="viridis")

# for i in ax.containers:
#     ax.bar_label(i,)

# plt.title("medical_specialty - Categories counts",fontsize=30)
# plt.show()
```

Data Preprocessing

In [10]:
```
# # Dropping rows with NaN transcription
# data = data.dropna(subset=["transcription"]).reset_index(drop=True)
```

In [11]:
```
# text_lengths = [len(x) for x in data['transcription']]
```

In [11]:
```
# text_lengths = [len(x) for x in data['transcription']]
```

In [12]:
```
# figure, (ax_box, ax_hist) = plt.subplots(2, sharex=True, figsize=(12,10), gridspec_k

# sns.boxplot(text_lengths,ax=ax_box, color='#A3C6D3')
# sns.histplot(text_lengths, bins=50, binwidth=1000, kde=True, ax=ax_hist, color='#A3C

# plt.suptitle('Medical transcription text size', fontsize=20)
# plt.xlabel('Number of words inside transcriptions', fontsize=25)
# plt.ylabel('Transcription Counts', fontsize=25)
# plt.xticks(fontsize=20)
# plt.yticks(fontsize=20)

# figure.tight_layout()
```

In [13]:
```
# # Rename of columns
# data = data.rename(columns = {"transcription":"text",
#                         "medical_specialty":"category"})
```

In [14]:
```
# # Keeping only categories with more than 50 appearances in the dataset
# category_counts = data.groupby(by="category")['category'].count()
# filtered_categories = category_counts[category_counts>50]
```

In [15]:
```
# data = data.loc[data["category"].isin(list(filtered_categories.index))].reset_index(
```

In [16]:
```
plt.figure(figsize=(10,7))

ax = sns.barplot(x=data['category'].value_counts().values,
            y=data['category'].value_counts().index,
            palette="viridis")

for i in ax.containers:
    ax.bar_label(i,)

plt.title("medical_specialty - Categories counts",fontsize=20)
plt.show()
```

In [17]:
```
target_variable_mapping, inverse_target_variable_mapping = get_target_variable_mapping
```

In [18]:
```
# Getting the cleaned and stemmed text into list of tokens
data["text"] = data["text"].apply(lambda x: clean_transcription(x,stemming_method='por
data["text"]
```

In [19]:
```
nested_list_of_words = []
for i in range(len(data["text"])):
    text_words = data["text"][i].split()
    nested_list_of_words.append(text_words)

#All the words from texts
flat_stemmed_list = [item for sublist in nested_list_of_words for item in sublist]

#Summing up the unique words in the texts
unique_words = pd.Series(flat_stemmed_list).value_counts()
print('Nr of unique words is : ',unique_words.shape[0])
print("=======================================")
print('20 most used words in the dataset are: ')
print("=======================================")
unique_words.head(20)
```

In [20]:
```
frequent_words = pd.DataFrame(unique_words)
frequent_words.reset_index(inplace=True)
frequent_words.columns = ['word','freq']

plt.figure(figsize=(14,7))

ax = sns.barplot(x=frequent_words['freq'][:20],
                 y=frequent_words['word'][:20],
                 palette="viridis")

for i in ax.containers:
    ax.bar_label(i,)

plt.title("Most Frequent occuring words - Top 20",fontsize=20)
plt.show()
```

Initial_Experiment

In [21]:
```
# Defning the corpus of the transcriptions
corpus = data['text']
corpus
```

In [23]:
```
# produce_model(dataset=data,
                labels=target_variable_mapping,
                inverse_labels = inverse_target_variable_mapping,
                random_state=10,
                test_size=0.2,
                vectorizer=TfidfVectorizer(),
                classifier=SVC(),
                feature_selection=None,
                dim_red=False,
                verbose=verbose,
                export_dir=export_directory
                )
```

Experiment II (Droping Overlaping labels)

In [24]:
```
data['category'] = data['category'].apply(lambda x:x.strip())
```

In [25]:
```
#Leaving out specialty of Surgery and other overlaping categories
labels_to_drop = ['Surgery', 'Consult - History and Phy.', 'SOAP / Chart / Progress No
                  'Emergency Room Reports', 'Pain Management']

data = data.loc[~data['category'].isin(labels_to_drop)]
```

In [26]:
```
# Replacing 'Neurosurgery' category with 'Neurology'
mask = data['category']=='Neurosurgery'
data.loc[mask,'category'] = 'Neurology'

# Replacing 'Nephrology' category with 'Urology'
mask = data['category']=='Nephrology'
data.loc[mask,'category'] = 'Urology'

data = data.reset_index(drop=True)
print(f'The new dataset is of shape: {data.shape}')
data['category'].value_counts()
```

In [27]:
```
target_variable_mapping, inverse_target_variable_mapping = get_target_variable_mapping
```

In [28]:
```
plt.figure(figsize=(10,7))

ax = sns.barplot(x=data['category'].value_counts().values,
            y=data['category'].value_counts().index,
            palette="viridis")

for i in ax.containers:
    ax.bar_label(i,)

plt.title("medical_specialty - Reduced Categories \n" ,fontsize=20)
plt.show()
```

In [29]:
```
nested_list_of_words = []
for i in range(len(data["text"])):
    text_words = data["text"][i].split()
    nested_list_of_words.append(text_words)

#All the words from texts
flat_stemmed_list = [item for sublist in nested_list_of_words for item in sublist]

#Summing up the unique words in the texts
unique_words = pd.Series(flat_stemmed_list).value_counts()
print('Nr of unique words is : ',unique_words.shape[0])
print("=====================================")
print('20 most used words in the dataset are: ')
print("=====================================")
unique_words.head(20)
```

In [30]:
```
frequent_words = pd.DataFrame(unique_words)
frequent_words.reset_index(inplace=True)
frequent_words.columns = ['word','freq']

plt.figure(figsize=(14,7))

ax = sns.barplot(x=frequent_words['freq'][:20],
                 y=frequent_words['word'][:20],
                 palette="viridis")

for i in ax.containers:
    ax.bar_label(i,)

plt.title("Most Frequent occuring words - Top 20",fontsize=20)
plt.show()
```

In [31]:
```
selected_features = produce_model(dataset=data,
                                  labels=target_variable_mapping,
                                  inverse_labels=inverse_target_variable_mapping,
                                  random_state=10,
                                  test_size=0.2,
                                  vectorizer=TfidfVectorizer(),
                                  classifier=SVC(),
                                  feature_selection= 'chi2',
                                  dim_red=False,
                                  verbose=verbose,
                                  export_dir=export_directory
                                  )
```

Experiment III Oversamplening with SMOTE

In [32]:
```
#Removing General Medicine and Radiology Catgory documents

data = data[~data['category'].isin(['General Medicine', 'Radiology'])]
data = data.reset_index(drop=True)
print(f'The new dataset is of shape: {data.shape}')
```

In [33]:
```
plt.figure(figsize=(10,7))

ax = sns.barplot(x=data['category'].value_counts().values,
            y=data['category'].value_counts().index,
            palette="viridis")

for i in ax.containers:
    ax.bar_label(i,)

plt.title("medical_specialty Counts Before SMOTE Oversampling \n" ,fontsize=20)
plt.show()
```

In [34]:
```
nested_list_of_words = []
for i in range(len(data["text"])):
    text_words = data["text"][i].split()
    nested_list_of_words.append(text_words)

#All the words from texts
flat_stemmed_list = [item for sublist in nested_list_of_words for item in sublist]

#Summing up the unique words in the texts
unique_words = pd.Series(flat_stemmed_list).value_counts()
print('Nr of unique words is : ',unique_words.shape[0])
print("=======================================")
print('20 most used words in the dataset are: ')
print("=======================================")
unique_words.head(20)
```

In [35]:
```
frequent_words = pd.DataFrame(unique_words)
frequent_words.reset_index(inplace=True)
frequent_words.columns = ['word','freq']

plt.figure(figsize=(14,7))

ax = sns.barplot(x=frequent_words['freq'][:20],
                 y=frequent_words['word'][:20],
                 palette="viridis")

for i in ax.containers:
    ax.bar_label(i,)

plt.title("Most Frequent occuring words - Top 20",fontsize=20)
plt.show()
```

In [36]:
```
target_variable_mapping, inverse_target_variable_mapping = get_target_variable_mapping
```

In [38]:
```
model = produce_model(dataset=data,
                      labels=target_variable_mapping,
                      inverse_labels=inverse_target_variable_mapping,
                      random_state=10,
                      test_size=0.2,
                      vectorizer=TfidfVectorizer(),
                      classifier=SVC(),
                      oversampling=True,
                      feature_selection= 'chi2',
                      dim_red=False,
                      verbose=verbose,
                      export_dir=export_directory
                      )
```

8.2. Medical Text Classification (Deep Learning Approach) Code

In [1]:
```
# !pip install matplotlib==3.5.3 --quiet
# !pip install pyngrok --quiet
# !pip install mlflow --quiet
# !pip install keras_preprocessing
```

In [2]:
```
import pandas as pd
import numpy as np

import random
import datetime
from time import strftime
from functools import partial
from tqdm.auto import tqdm


import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.feature_extraction.text import TfidfVectorizer
from tensorflow.keras.preprocessing.text import Tokenizer
from keras_preprocessing.sequence import pad_sequences
from keras.models import Sequential
from keras import layers, Model
from keras.callbacks import EarlyStopping, ModelCheckpoint


from keras.utils.vis_utils import plot_model

from sklearn.model_selection import train_test_split

from keras.metrics import Recall, Precision, AUC

import os
os.chdir("drive/MyDrive/Github/Medical_Text_and_Image_Classification_v1")

from functions.preprocessing import clean_transcription, get_features_labels, split_da
from functions.utils import get_target_variable_mapping, get_tensorboard
from functions.model_evaluation import plot_confusion_matrix
from functions.modeling import generate_text

from sklearn.metrics import classification_report,confusion_matrix
```

Notebook parameters

In [3]:
```
log_dir = 'tensorboard_logs/'

vocab_size = 2500
export=True
```

In [5]:
```
gtqdm = partial(tqdm, position=0, leave=True)

# Folders
export_folder = 'Run_{}'.format(str(datetime.datetime.now()).replace(' ','_')[:19])

# Export directory
export_directory = './exports/'+export_folder

if export:
  if not os.path.exists(export_directory):
    os.makedirs(export_directory)
```

Import And Explore the dataset

In [6]:
```
data = pd.read_csv("Data/mtsamples.csv",delimiter=',',index_col=0)
data.head()
```

In [7]:
```
# Dropping unnecessary columns
data = data.drop(['description', 'sample_name', 'keywords' ], axis=1)[["transcription"
data
```

Data Preprocessing

In [9]:
```
data = data.dropna(subset=["transcription"]).reset_index(drop=True)
```

In [10]:
```
# Rename of columns
data = data.rename(columns = {"transcription":"text",
                        "medical_specialty":"category"})
```

In [11]:
```
# Keeping only categories with more than 50 appearances in the dataset
category_counts = data.groupby(by="category")['category'].count()
filtered_categories = category_counts[category_counts>50]
```

In [12]:
```
data = data.loc[data["category"].isin(list(filtered_categories.index))].reset_index(dr
```

In [14]:
```
data['category'] = data['category'].apply(lambda x:x.strip())
```

In [15]:
```
# Replacing 'Neurosurgery' category with 'Neurology'
mask = data['category']=='Neurosurgery'
data.loc[mask,'category'] = 'Neurology'

# Replacing 'Nephrology' category with 'Urology'
mask = data['category']=='Nephrology'
data.loc[mask,'category'] = 'Urology'

data = data.reset_index(drop=True)
print(f'The new dataset is of shape: {data.shape}')
data['category'].value_counts()
```

In [16]:
```
#Leaving out specialty of Surgery and other overlaping categories
labels_to_drop = ['Surgery', 'General Medicine', 'Consult - History and Phy.', 'SOAP /
                  'Emergency Room Reports', 'Pain Management', 'Radiology']


data = data.loc[~data['category'].isin(labels_to_drop)]
```

In [17]:
```
plt.figure(figsize=(10,7))

ax = sns.barplot(x=data['category'].value_counts().values,
            y=data['category'].value_counts().index,
            palette="viridis")

for i in ax.containers:
    ax.bar_label(i,)

plt.title("medical_specialty - Categories counts \n",fontsize=20)
plt.show()
```

In [18]:
```
# Getting the cleaned and lemmatized text into list of tokens
data["text"] = data["text"].apply(lambda x: clean_transcription(x,stemming_method = '
data["text"]
```

In [19]:
```
target_variable_mapping, inverse_target_variable_mapping = get_target_variable_mapping
docs = data['text'].to_list()

X = docs
y = np.array(data['category'].map(target_variable_mapping))
```

Models

Deep Neural Network

In [20]:
```
X_train, X_test , y_train, y_test = train_test_split(X, y , test_size = 0.2, random_st
```

In [21]:
```
vectorizer = TfidfVectorizer(max_features=5000)
X_train = vectorizer.fit_transform(X_train).toarray()
X_test = vectorizer.transform(X_test).toarray()
print("tf-idf with",str(np.array(X_train).shape[1]),"features")
print(X_train.shape)
print(X_test.shape)
```

In [22]:
```
dropout = 0.25
node = 512 # number of nodes

# metrics = [Recall(),Precision(), AUC()]

dnn = Sequential()

dnn.add(layers.Dense(node,input_dim=5000, activation='relu',name='Input_layer'))
dnn.add(layers.Dropout(dropout, name='Dropout_layer_1'))
dnn.add(layers.Dense(node,input_dim=node, activation='relu', name='hidden_1'))
dnn.add(layers.Dropout(dropout, name='Dropout_layer_2'))
dnn.add(layers.Dense(node,input_dim=node, activation='relu', name='hidden_2'))
dnn.add(layers.Dropout(dropout, name='Dropout_layer_3'))
dnn.add(layers.Dense(node,input_dim=node, activation='relu', name='hidden_3'))
dnn.add(layers.Dropout(dropout, name='Dropout_layer_4'))
dnn.add(layers.Dense(node,input_dim=node, activation='relu', name='hidden_4'))
dnn.add(layers.Dropout(dropout, name='Dropout_layer__'))
dnn.add(layers.Dense(len(np.unique(y)), activation='softmax', name='Output_layer'))

dnn.compile(loss='sparse_categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])
dnn.summary()
```

In [23]:
```
%%time
dnn_history = dnn.fit(X_train,
                      y_train,
                      epochs=6,
                      batch_size = 100,
                      callbacks=[get_tensorboard('DNN')],
                      validation_data=(X_test,y_test))



dnn_loss, dnn_accuracy = dnn.evaluate(X_train, y_train, verbose=False)
print("Training Accuracy: {:.4f}".format(dnn_accuracy))
dnn_loss, dnn_accuracy = dnn.evaluate(X_test, y_test, verbose=False)
print("Testing Accuracy:  {:.4f}".format(dnn_accuracy))

# Making Predictions
y_pred = dnn.predict(X_test)
y_pred_classes = y_pred.argmax(axis=-1)

# Classification Report
print(classification_report(list(y_test),list(y_pred_classes)))

# Saving the classification report into xlsx file
clf_report = pd.DataFrame(classification_report(list(y_test), list(y_pred_classes), ou
clf_report = clf_report.rename(columns={'index':'label'})
clf_report_labels = clf_report.iloc[:pd.Series(y).nunique()]
clf_report_metrics = clf_report.iloc[pd.Series(y).nunique():]
clf_report_labels['label'] = clf_report_labels['label'].astype(int).map(inverse_target
clf_report = pd.concat([clf_report_labels,clf_report_metrics])
clf_report.to_excel(export_directory+'/classification_report_dnn_model.xlsx',index=Fal

# Ploting the confusion matrix of the classification
plot_confusion_matrix(list(y_test),list(y_pred_classes),labels=target_variable_mapping

plot_model(dnn, to_file=export_directory+'/dnn_model_plot.png', show_shapes=True, show
```

Convolutional Neural Network

In [24]:
```
tokenizer = Tokenizer()
tokenizer.fit_on_texts(docs)
```

In [25]:
```
vocab_size = len(tokenizer.word_index) + 1
vocab_size
```

In [28]:
```
encoded_docs = tokenizer.texts_to_sequences(docs)
```

In [27]:
```
encoded_docs_padded = pad_sequences(encoded_docs, padding='post', maxlen=100)
print(encoded_docs_padded[0, :])
```

In [29]:
```
X_train, X_test , y_train, y_test = train_test_split(encoded_docs_padded, y , test_si
```

In [30]:
```
%%time
embedding_dim = 300
cnn = Sequential(name='CNN_model')
cnn.add(layers.Embedding(vocab_size, embedding_dim, input_length=100, name='Embedding_
cnn.add(layers.Conv1D(80, 5, activation='relu', name='1-D_Convoluion'))
cnn.add(layers.GlobalMaxPooling1D(name='Global_Max_Pooling'))
cnn.add(layers.Dense(64, activation='relu', name='Dense'))
cnn.add(layers.Dropout(0.35, seed=10,name='Dropout'))
cnn.add(layers.Dense(len(np.unique(y)), activation='softmax', name= 'Output_Layer'))
cnn.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
cnn.summary()
```

In [31]:
```
%%time
cnn_history = cnn.fit(X_train,
                      y_train,
                      epochs=6,
                      batch_size = None,
                      callbacks=[EarlyStopping(monitor='val_loss',patience=3), get_ter
                      validation_data=(X_test,y_test))


cnn_loss, cnn_accuracy = cnn.evaluate(X_train, y_train, verbose=False)
print("Training Accuracy: {:.4f}".format(cnn_accuracy))
cnn_loss, cnn_accuracy = cnn.evaluate(X_test, y_test, verbose=False)
print("Testing Accuracy:  {:.4f}".format(cnn_accuracy))

# Making Predictions
y_pred = cnn.predict(X_test)
y_pred_classes = y_pred.argmax(axis=-1)

# Classification Report
print(classification_report(list(y_test),list(y_pred_classes)))

# Saving the classification report into xlsx file
clf_report = pd.DataFrame(classification_report(list(y_test), list(y_pred_classes), ou
clf_report = clf_report.rename(columns={'index':'label'})
clf_report_labels = clf_report.iloc[:pd.Series(y).nunique()]
clf_report_metrics = clf_report.iloc[pd.Series(y).nunique():]
clf_report_labels['label'] = clf_report_labels['label'].astype(int).map(inverse_target
clf_report = pd.concat([clf_report_labels,clf_report_metrics])
clf_report.to_excel(export_directory+'/classification_report_cnn_model.xlsx',index=Fal

# Ploting the confusion matrix of the classification
plot_confusion_matrix(list(y_test),list(y_pred_classes),labels=target_variable_mapping


plot_model(cnn, to_file=export_directory+'/cnn_model_plot.png', show_shapes=True, show
```

Convolutional Neural Network with Pre-trained Glovee

In [32]:
```
tokenizer = Tokenizer()
tokenizer.fit_on_texts(docs)
```

In [33]:
```
vocab_size = len(tokenizer.word_index) + 1
```

In [34]:
```
word_index = tokenizer.word_index
encoded_docs = tokenizer.texts_to_sequences(docs)
```

In [35]:
```
encoded_docs_padded = pad_sequences(encoded_docs, padding='post', maxlen=100)
print(encoded_docs_padded[0, :])
```

In [36]:
```
X_train, X_test , y_train, y_test = train_test_split(encoded_docs_padded, y , test_si
```

In [37]:
```
word_vectors={}
f = open('Glovee/glove.6B.300d.txt', encoding="utf8")
for line in tqdm(f):
    values = line.split()
    word = values[0]
    coefs = np.asarray(values[1:], dtype='float32')
    word_vectors[word] = coefs
f.close()
print('Loaded %s word vectors.' % len(word_vectors))
```

In [38]:
```
embedding_dim=300
vocabulary_size=len(word_index)+1

embedding_matrix = np.zeros((vocabulary_size, embedding_dim))


for word, i in tqdm(word_index.items()):
    if i>=vocabulary_size:
        continue
    try:
      embedding_vector = word_vectors[word]
      embedding_matrix[i] = embedding_vector
    except:
        continue
```

In [39]:
```
%%time
embedding_dim = 300
cnn_glovee = Sequential(name='CNN_with_Glovee_Embediings')
cnn_glovee.add(layers.Embedding(vocab_size, embedding_dim, input_length=100, weights=[
cnn_glovee.add(layers.Dropout(0.2, seed=10, name='Droput_1'))
cnn_glovee.add(layers.Conv1D(80, 5, activation='relu', name='1-D_Convolution_1'))
cnn_glovee.add(layers.Dropout(0.2, seed=10, name='Droput_2'))
cnn_glovee.add(layers.GlobalMaxPooling1D(name='Global_Max_Pooling'))
cnn_glovee.add(layers.Dense(32, activation='relu', name='Dense_layer_1'))
cnn_glovee.add(layers.Dropout(0.20, seed=10, name='Droput_4'))
cnn_glovee.add(layers.Dense(len(np.unique(y)), activation='softmax',name='Output_Layer
cnn_glovee.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
cnn_glovee.summary()
```

In [40]:
```
%%time
cnn_glovee_history = cnn_glovee.fit(X_train,
                                    y_train,
                                    epochs=25,
                                    batch_size = None,
                                    callbacks=[EarlyStopping(monitor='val_loss',patier
                                    validation_data=(X_test,y_test))


cnn_glovee_loss, cnn_glovee_accuracy = cnn_glovee.evaluate(X_train, y_train, verbose=F
print("Training Accuracy: {:.4f}".format(cnn_glovee_accuracy))
cnn_glovee_loss, cnn_glovee_accuracy = cnn_glovee.evaluate(X_test, y_test, verbose=Fal
print("Testing Accuracy:  {:.4f}".format(cnn_glovee_accuracy))

# Making Predictions
y_pred = cnn_glovee.predict(X_test)
y_pred_classes = y_pred.argmax(axis=-1)

# Classification Report
print(classification_report(list(y_test),list(y_pred_classes)))

# Saving the classification report into xlsx file
clf_report = pd.DataFrame(classification_report(list(y_test), list(y_pred_classes), ou
clf_report = clf_report.rename(columns={'index':'label'})
clf_report_labels = clf_report.iloc[:pd.Series(y).nunique()]
clf_report_metrics = clf_report.iloc[pd.Series(y).nunique():]
clf_report_labels['label'] = clf_report_labels['label'].astype(int).map(inverse_target
clf_report = pd.concat([clf_report_labels,clf_report_metrics])
clf_report.to_excel(export_directory+'/classification_report_cnn_Glovee_emb_model.xls

# Ploting the confusion matrix of the classification
plot_confusion_matrix(list(y_test),list(y_pred_classes),labels=target_variable_mapping


plot_model(cnn, to_file=export_directory+'/cnn_Glovee_emb_model_plot.png', show_shapes
```

In [41]:
```
fig, ax = plt.subplots(2,figsize=(15,12))


l1 = ax[0].plot(cnn_history.history['val_accuracy'], color='#580000', linewidth=3, lab
l2 = ax[0].plot(dnn_history.history['val_accuracy'], color='#297878', linewidth=3, lab
l3 = ax[0].plot(cnn_glovee_history.history['val_accuracy'], color='#edc58d', linewidth
ax[0].set_title('CNN Vs DNN Validation Accuracy over epochs \n',fontsize=20)
ax[0].set_xlabel("Epochs",fontsize=15)
ax[0].set_ylabel("Validation Accuracy",fontsize=15)
ax[0].legend([l1, l2, l3], labels= ['CNN accuracy: {0:.2f}'.format(cnn_accuracy),
                                    'DNN accuracy: {0:.2f}'.format(dnn_accuracy),
                                    'CNN with Glovee accuracy: {0:.2f}'.format(cnn_glo


l4 = ax[1].plot(cnn_history.history['val_loss'], color='#580000', linewidth=3, label='
l5 = ax[1].plot(dnn_history.history['val_loss'], color='#297878', linewidth=3, label='
l6 = ax[1].plot(cnn_glovee_history.history['val_loss'], color='#edc58d', linewidth=3,
ax[1].set_title('CNN Vs DNN Validation Loss over epochs \n',fontsize=20)
ax[1].set_xlabel("Epochs",fontsize=15)
ax[1].set_ylabel("Validation Loss",fontsize=15)
ax[1].legend([l4, l5, l6], labels= ['CNN loss: {0:.2f}'.format(cnn_loss),
                                    'DNN loss: {0:.2f}'.format(dnn_loss),
                                    'CNN with Glovee loss: {0:.2f}'.format(cnn_glovee_
plt.subplots_adjust( bottom=0.1,
                     top=0.9,
                     wspace=0.7,
                     hspace=0.7)



plt.savefig(export_directory+'/Models_Performance.png')
```

Text Generation for minority classes

In [42]:
```
# data['category'].value_counts()
```

In [43]:
```
# mask = data['category'] == 'Urology'
# urology_txt = data.loc[mask, 'text'].to_list()

# mask = data['category'] == 'Gastroenterology'
# gastroenterology_txt = data.loc[mask, 'text'].to_list()

# mask = data['category'] == 'Obstetrics / Gynecology'
# obstetrics_gynecology_txt = data.loc[mask, 'text'].to_list()

# mask = data['category'] == 'ENT - Otolaryngology'
# ent_otolaryngology_txt = data.loc[mask, 'text'].to_list()

# mask = data['category'] == 'Hematology - Oncology'
# hematology_oncology_txt = data.loc[mask, 'text'].to_list()

# mask = data['category'] == 'Ophthalmology'
# ophthalmology_txt = data.loc[mask, 'text'].to_list()

# mask = data['category'] == 'Pediatrics - Neonatal'
# pediatrics_neonatal_txt = data.loc[mask, 'text'].to_list()

# mask = data['category'] == 'Psychiatry / Psychology'
# psychiatry_psychology_txt = data.loc[mask, 'text'].to_list()
```

In [44]:
```
# category = 'Ophthalmology'
# texts = psychiatry_psychology_txt
```

In [45]:
```
# gfl = get_features_labels(15,texts)

# features = gfl['features']
# labels = gfl['labels']
# num_words = gfl['num_words']
# word_idx = gfl[`word_idx`]
# new_sequences = gfl['new_sequences']
```

In [46]:
```
# embedding_dim=300
# vocabulary_size=len(word_idx)+1

# embedding_matrix = np.zeros((num_words, embedding_dim))


# for word, i in tqdm(word_idx.items()):
#     if i>=vocabulary_size:
#         continue
#     try:
#       embedding_vector = word_vectors[word]
#       embedding_matrix[i] = embedding_vector
#     except:
#         continue
```

In [47]:
```
# X_train, X_test, y_train, y_test = split_dataset(features=features,
#                                                  labels=labels,
#                                                  num_words=num_words,
#                                                  training_size=0.7)
```

LSTM model for text generation

In [48]:
```
embedding_dim = 300
```

In [49]:
```
embedding_matrix = np.zeros((num_words, embedding_dim))

for word, i in tqdm(word_idx.items()):
    if i>=num_words:
        continue
    try:
      embedding_vector = word_vectors[word]
      embedding_matrix[i] = embedding_vector
    except:
        continue
```

In [50]:
```
# Definition of model architecture
check_point = ModelCheckpoint(filepath=os.getcwd(),monitor='val_loss', verbose=1, save
early_stoping = EarlyStopping(monitor='val_loss', mode="min", patience=2)


x = layers.Input(shape=(15,))

embedding = layers.Embedding(input_dim=num_words,
                     output_dim=embedding_matrix.shape[1],
                     weights=[embedding_matrix],
                     trainable=False)(x)

encoder = layers.Bidirectional(layers.LSTM(128, return_sequences=True, dropout=0.1, re
encoder = layers.LSTM(128, return_sequences=True, dropout=0.1, recurrent_dropout=0.1)(

decoder = layers.LSTM(128, recurrent_dropout=0.35, dropout=0.3, return_sequences=True)
decoder = layers.LSTM(128, recurrent_dropout=0.35, dropout=0.3, return_sequences=True)

attention = layers.Attention()([encoder, decoder])

decoder_concat_input = layers.Concatenate()([decoder, attention])

flatten = layers.Flatten()(decoder_concat_input)

y = layers.Dense(num_words, activation='softmax')(flatten)

model = Model(inputs=x, outputs=y)
model.compile(
    optimizer='adam',
    loss='categorical_crossentropy',
    metrics=['accuracy'])

plot_model(model, to_file=export_directory+'/lstm_text_generator.png', show_shapes=Tru
```

In [51]:
```
model.fit(X_train,
          y_train,
          epochs=1,
          callbacks=[check_point, early_stoping],
          validation_data=(X_test,y_test))
```

In [52]:
```
generated_text = []
for i in tqdm(range(250,),leave=True):

  gen_text = generate_text(model, new_sequences)
  generated_text.append(gen_text)

generated_text_df = [' '.join(i) for i in generated_text]
generated_text_df = pd.DataFrame(generated_text_df, columns = ['text'])
generated_text_df['category'] = category
generated_text_df.to_csv(f"generated_data/generated_text_glove_300{category}.csv", inc
```

Glovee - LSTM Generated Data Experiments

In [53]:
```
# Loading all the generated datasets into one
import glob

file_path='generated_data'
files = glob.glob(file_path + "/*.csv")

generated_data = pd.DataFrame()
content = []

for file_name in files:
  print(file_name)

  df = pd.read_csv(file_name, index_col=None)
  content.append(df)

generated_data = pd.concat(content).sample(frac=1).reset_index(drop=True)

generated_data['generated'] = 1
display(generated_data)
```

In [54]:
```
generated_data = generated_data.loc[generated_data['category']!='Radiology']

data_all = pd.concat([data,generated_data],axis=0)
data_all['generated'] = data_all['generated'].fillna(0)



target_variable_mapping, inverse_target_variable_mapping = get_target_variable_mapping
X = data_all['text'].to_list()
y = np.array(data_all['category'].map(target_variable_mapping))

X_generated = generated_data['text'].to_list()
y_generated = np.array(generated_data['category'].map(target_variable_mapping))
```

In [55]:
```
plt.figure(figsize=(10,7))

ax = sns.barplot(x=data_all['category'].value_counts().values,
            y=data_all['category'].value_counts().index,
            palette="viridis")

for i in ax.containers:
    ax.bar_label(i,)

plt.title("Categories counts after Text Generation\n",fontsize=20)
plt.show()
```

DNN

In [56]:
```
X_train, X_test , y_train, y_test = train_test_split(X, y , test_size = 0.25, random_s
```

In [57]:
```
vectorizer = TfidfVectorizer(max_features=2500)
X_train = vectorizer.fit_transform(X_train).toarray()
X_test = vectorizer.transform(X_test).toarray()
print("tf-idf with",str(np.array(X_train).shape[1]),"features")
print("X-train shape is: ", X_train.shape)
print("X-test shape is: ", X_test.shape)
```

In [58]:
```
vectorizer = TfidfVectorizer(max_features=2500)
X_generated_vectorized = vectorizer.fit_transform(X_generated).toarray()

print("X_generated shape is: ", X_generated_vectorized.shape)
```

In [59]:
```
X_train = np.concatenate((X_train, X_generated_vectorized), axis=0)
y_train = np.concatenate((y_train,y_generated),axis=0)

print(X_train.shape)
print(y_train.shape)
```

In [60]:
```
dropout = 0.25
node = 512 # number of nodes
# metrics = [Recall(),Precision(), AUC()]

dnn = Sequential()

dnn.add(layers.Dense(node,input_dim=X_train.shape[1], activation='relu',name='Input_la
dnn.add(layers.Dropout(dropout, name='Dropout_layer_1'))
dnn.add(layers.Dense(node,input_dim=node, activation='relu', name='hidden_1'))
dnn.add(layers.Dropout(dropout, name='Dropout_layer_2'))
dnn.add(layers.Dense(node,input_dim=node, activation='relu', name='hidden_2'))
dnn.add(layers.Dropout(dropout, name='Dropout_layer_3'))
dnn.add(layers.Dense(node,input_dim=node, activation='relu', name='hidden_3'))
dnn.add(layers.Dropout(dropout, name='Dropout_layer_4'))
dnn.add(layers.Dense(node,input_dim=node, activation='relu', name='hidden_4'))
dnn.add(layers.Dropout(dropout, name='Dropout_layer_5'))
dnn.add(layers.Dense(len(np.unique(y)), activation='softmax', name='Output_layer'))

dnn.compile(loss='sparse_categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])
dnn.summary()
```

In [61]:
```
%%time
dnn_gen_history = dnn.fit(X_train,
                          y_train,
                          epochs=2,
                          batch_size = None,
                          callbacks=[get_tensorboard('DNN_gen')],
                          validation_data=(X_test,y_test))



dnn_gen_loss, dnn_gen_accuracy = dnn.evaluate(X_train, y_train, verbose=False)
print("Training Accuracy: {:.4f}".format(dnn_gen_accuracy))
dnn_gen_loss, dnn_gen_accuracy = dnn.evaluate(X_test, y_test, verbose=False)
print("Testing Accuracy:  {:.4f}".format(dnn_gen_accuracy))

# Making Predictions
y_pred = dnn.predict(X_test)
y_pred_classes = y_pred.argmax(axis=-1)

# Classification Report
print(classification_report(list(y_test),list(y_pred_classes)))

# Saving the classification report into xlsx file
clf_report = pd.DataFrame(classification_report(list(y_test), list(y_pred_classes), ou
clf_report = clf_report.rename(columns={'index':'label'})
clf_report_labels = clf_report.iloc[:pd.Series(y).nunique()]
clf_report_metrics = clf_report.iloc[pd.Series(y).nunique():]
clf_report_labels['label'] = clf_report_labels['label'].astype(int).map(inverse_target
clf_report = pd.concat([clf_report_labels,clf_report_metrics])
clf_report.to_excel(export_directory+'/classification_report_dnn_gen_model.xlsx',index

# Ploting the confusion matrix of the classification
plot_confusion_matrix(list(y_test),list(y_pred_classes),labels=target_variable_mapping

plot_model(dnn, to_file=export_directory+'/dnn_gen_model_plot.png', show_shapes=True,
```

In [62]:
```
X_train, X_test , y_train, y_test = train_test_split(X, y , test_size = 0.25, random_s
```

In [63]:
```
tokenizer = Tokenizer()
tokenizer.fit_on_texts(X)

vocab_size = len(tokenizer.word_index) + 1
print('Vocabulary size is: ', vocab_size)
```

In [64]:
```
word_index = tokenizer.word_index
encoded_docs_train = tokenizer.texts_to_sequences(X_train)
encoded_docs_test = tokenizer.texts_to_sequences(X_test)
```

In [65]:
```
encoded_docs_padded_train = pad_sequences(encoded_docs_train, padding='post', maxlen=1
encoded_docs_padded_test = pad_sequences(encoded_docs_test, padding='post', maxlen=100
```

In [66]:
```
embedding_dim=300
vocabulary_size=len(word_index)+1

embedding_matrix = np.zeros((vocabulary_size, embedding_dim))


for word, i in tqdm(word_index.items()):
    if i>=vocabulary_size:
        continue
    try:
      embedding_vector = word_vectors[word]
      embedding_matrix[i] = embedding_vector
    except:
        continue
```

In [67]:
```
%%time
embedding_dim = 300
cnn_glovee = Sequential(name='CNN_with_Glovee_Embediings')
cnn_glovee.add(layers.Embedding(vocab_size, embedding_dim, input_length=100, weights=[
cnn_glovee.add(layers.Dropout(0.2, seed=10, name='Droput_1'))
cnn_glovee.add(layers.Conv1D(80, 5, activation='relu', name='1-D_Convolution_1'))
cnn_glovee.add(layers.Dropout(0.2, seed=10, name='Droput_2'))
cnn_glovee.add(layers.GlobalMaxPooling1D(name='Global_Max_Pooling'))
cnn_glovee.add(layers.Dense(32, activation='relu', name='Dense_layer_1'))
cnn_glovee.add(layers.Dropout(0.20, seed=10, name='Droput_4'))
cnn_glovee.add(layers.Dense(len(np.unique(y)), activation='softmax',name='Output_Layer
cnn_glovee.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
cnn_glovee.summary()
```

In [68]:
```
%%time
cnn_glovee_gen_history = cnn_glovee.fit(encoded_docs_padded_train,
                                        y_train,
                                        epochs=25,
                                        batch_size = None,
                                        callbacks=[EarlyStopping(monitor='val_loss',patier
                                        validation_data=(encoded_docs_padded_test,y_test))


cnn_glovee_gen_loss, cnn_glovee_gen_accuracy = cnn_glovee.evaluate(encoded_docs_padded
print("Training Accuracy: {:.4f}".format(cnn_glovee_gen_accuracy))
cnn_glovee_gen_loss, cnn_glovee_gen_accuracy = cnn_glovee.evaluate(encoded_docs_padded
print("Testing Accuracy:  {:.4f}".format(cnn_glovee_gen_accuracy))

# Making Predictions
y_pred = cnn_glovee.predict(encoded_docs_padded_test)
y_pred_classes = y_pred.argmax(axis=-1)

# Classification Report
print(classification_report(list(y_test),list(y_pred_classes)))

# Saving the classification report into xlsx file
clf_report = pd.DataFrame(classification_report(list(y_test), list(y_pred_classes), ou
clf_report = clf_report.rename(columns={'index':'label'})
clf_report_labels = clf_report.iloc[:pd.Series(y).nunique()]
clf_report_metrics = clf_report.iloc[pd.Series(y).nunique():]
clf_report_labels['label'] = clf_report_labels['label'].astype(int).map(inverse_target
clf_report = pd.concat([clf_report_labels,clf_report_metrics])
clf_report.to_excel(export_directory+'/classification_report_cnn_gen_Glovee_emb_model.

# Ploting the confusion matrix of the classification
plot_confusion_matrix(list(y_test),list(y_pred_classes),labels=target_variable_mapping


plot_model(cnn, to_file=export_directory+'/cnn_gen_Glovee_emb_model_plot.png', show_sh
```

In [69]:

```
fig, ax = plt.subplots(2,figsize=(15,12))


l1 = ax[0].plot(cnn_history.history['val_accuracy'], color='#580000', linewidth=3, lab
l2 = ax[0].plot(dnn_history.history['val_accuracy'], color='#297878', linewidth=3, lab
l3 = ax[0].plot(cnn_glovee_history.history['val_accuracy'], color='#edc58d', linewidth
l4 = ax[0].plot(cnn_glovee_gen_history.history['val_accuracy'], color='#ac0000', linew
l5 = ax[0].plot(dnn_gen_history.history['val_accuracy'], color='#0050ae', linewidth=3,

ax[0].set_title('CNN Vs DNN Validation Accuracy over epochs \n',fontsize=20)
ax[0].set_xlabel("Epochs",fontsize=15)
ax[0].set_ylabel("Validation Accuracy",fontsize=15)
ax[0].legend([l1, l2, l3], labels= ['CNN accuracy: {0:.2f}'.format(cnn_accuracy),
                                    'DNN accuracy: {0:.2f}'.format(dnn_accuracy),
                                    'CNN with Glovee accuracy: {0:.2f}'.format(cnn_glo
                                    'CNN with Glovee on Generated text: {0:.2f}'.forma
                                    'DNN on Generated text: {0:.2f}'.format(dnn_gen_ac

l6 = ax[1].plot(cnn_history.history['val_loss'], color='#580000', linewidth=3, label='
l7 = ax[1].plot(dnn_history.history['val_loss'], color='#297878', linewidth=3, label='
l8 = ax[1].plot(cnn_glovee_history.history['val_loss'], color='#edc58d', linewidth=3,
l9 = ax[1].plot(cnn_glovee_gen_history.history['val_loss'], color='#ac0000', linewidth
l10 = ax[1].plot(dnn_gen_history.history['val_loss'], color='#0050ae', linewidth=3, la

ax[1].set_title('CNN Vs DNN Validation Loss over epochs \n',fontsize=20)
ax[1].set_xlabel("Epochs",fontsize=15)
ax[1].set_ylabel("Validation Loss",fontsize=15)
ax[1].legend([l4, l5, l6], labels= ['CNN loss: {0:.2f}'.format(cnn_loss),
                                    'DNN loss: {0:.2f}'.format(dnn_loss),
                                    'CNN with Glovee loss: {0:.2f}'.format(cnn_glovee_
                                    'CNN with Glovee on Generated text: {0:.2f}'.forma
                                    'DNN on Generated text: {0:.2f}'.format(dnn_gen_lo

plt.subplots_adjust( bottom=0.1,
                     top=0.9,
                     wspace=0.7,
                     hspace=0.7)



plt.savefig(export_directory+'/Models_Performance_all.png')
```

8.3. Medical Image Classification Code

Imports

In [1]:
```
# !pip install visualkeras
```

In [2]:
```
import pandas as pd
import numpy as np

import random
import datetime
import cv2
import os
import glob

from sklearn.metrics import classification_report,confusion_matrix, roc_curve, roc_auc

import tensorflow as tf
from tensorflow.keras.callbacks import EarlyStopping

from keras.preprocessing.image import ImageDataGenerator
from keras.utils import array_to_img

from tensorflow.keras.applications.vgg16 import VGG16

import os
os.chdir("drive/MyDrive/Github/Medical_Text_and_Image_Classification_v1")

from functions.model_evaluation import plot_confusion_matrix
# from functions.utils import  get_tensorboard

import visualkeras
import matplotlib.pyplot as plt
from keras.utils.vis_utils import plot_model
# from ann_visualizer.visualize import ann_viz
```

Constants

In [3]:
```
log_dir = 'tensorboard_logs/'

export = True
image_augmentation = False
data = 'Covid' # 'Covid' or 'all'

if data == 'all':
  train_dir = '/content/drive/MyDrive/Github/Medical_Text_and_Image_Classification_v1/
  test_dir = '/content/drive/MyDrive/Github/Medical_Text_and_Image_Classification_v1/D

else:
  train_dir = '/content/drive/MyDrive/Github/Medical_Text_and_Image_Classification_v1/
  test_dir = '/content/drive/MyDrive/Github/Medical_Text_and_Image_Classification_v1/D

image_width = 224
```

```
image_height = 224
color_channels = 3

image_pixels = image_width*image_height
total_imputs = image_pixels*color_channels

log_dir = 'tensorboard_logs/'

# Folders
export_folder = 'Run_{}'.format(str(datetime.datetime.now()).replace(' ','_')[:19])

# Export directory
export_directory = './exports/'+export_folder

if export:
  if not os.path.exists(export_directory):
    os.makedirs(export_directory)
```

Image Data Loading

In [4]:

```
train_covid_dir =  train_dir + '/Covid'
train_normal_dir = train_dir + '/Normal'
train_pneumonia_dir =  train_dir + '/Viral Pneumonia'
test_covid_dir = test_dir + '/Covid'
test_normal_dir = test_dir + '/Normal'
test_pneumonia_dir =  test_dir + '/Viral Pneumonia'

# Loading Images
dir = train_covid_dir
data_path = os.path.join(dir,'*g')
files = glob.glob(data_path)
train_covid = []
for f1 in files:
    img = cv2.imread(f1)
    train_covid.append(img)


dir = train_normal_dir
data_path = os.path.join(dir,'*g')
files = glob.glob(data_path)
train_normal = []
for f1 in files:
    img = cv2.imread(f1)
    train_normal.append(img)


dir = test_covid_dir
data_path = os.path.join(dir,'*g')
files = glob.glob(data_path)
test_covid = []
for f1 in files:
    img = cv2.imread(f1)
    test_covid.append(img)
```

```
dir = test_normal_dir
data_path = os.path.join(dir,'*g')
files = glob.glob(data_path)
test_normal = []
for f1 in files:
    img = cv2.imread(f1)
    test_normal.append(img)


if data == 'all':

  dir = test_pneumonia_dir
  data_path = os.path.join(dir,'*g')
  files = glob.glob(data_path)
  test_pneumonia = []
  for f1 in files:
      img = cv2.imread(f1)
      test_pneumonia.append(img)


  dir = train_pneumonia_dir
  data_path = os.path.join(dir,'*g')
  files = glob.glob(data_path)
  train_pneumonia = []
  for f1 in files:
      img = cv2.imread(f1)
      train_pneumonia.append(img)

if data == 'all':

  train_all = train_normal + train_covid + train_pneumonia
  test_all = test_normal + test_covid + test_pneumonia

else:

  train_all = train_normal + train_covid
  test_all = test_normal + test_covid
```

Data Exploration

In [20]:
```
fig, axes = plt.subplots(2,3,figsize=(15,7))
plt.suptitle("Normal Chest X-rays", fontsize=18, y=0.95)

for i, ax in enumerate(axes.ravel()):

  ax = plt.subplot(2, 3, i + 1)
  ax = plt.imshow(test_normal[i])
  plt.xticks([])
  plt.yticks([])

plt.show()

fig, axes = plt.subplots(2,3,figsize=(15,7))
plt.suptitle("Covid-19 Chest X-rays", fontsize=18, y=0.95)
```

```
for i, ax in enumerate(axes.ravel()):

  ax = plt.subplot(2, 3, i + 1)
  ax = plt.imshow(train_covid[i])
  plt.xticks([])
  plt.yticks([])

plt.show()

if data == 'all':

  fig, axes = plt.subplots(2,3,figsize=(15,7))
  plt.suptitle("Viral Pneumonia Chest X-rays", fontsize=18, y=0.95)

  for i, ax in enumerate(axes.ravel()):

    ax = plt.subplot(2, 3, i + 1)
    ax = plt.imshow(train_pneumonia[i])
    plt.xticks([])
    plt.yticks([])

  plt.show()
```

Preprocessing

In [21]:
```
# Defining the target variable
y_train_normal = list(np.full(len(train_normal),0))
y_test_normal = list(np.full(len(test_normal),0))
y_train_covid = list(np.full(len(train_covid),1))
y_test_covid = list(np.full(len(test_covid),1))

if data == 'all':

  y_train_pneumonia = list(np.full(len(train_covid),2))
  y_test_pneumonia = list(np.full(len(test_covid),2))

  target_variable_mapping = {'Normal': 0,
                             'Covid-19': 1,
                             'Viral Pneumonia': 2
                             }
else:

  target_variable_mapping = {'Normal': 0,
                             'Covid-19': 1
                             }


inverse_target_variable_mapping = {value : key for key, value in target_variable_mappi
```

In [7]:
```
y_train_all = y_train_normal + y_train_covid #+ y_train_pneumonia
y_test_all = y_test_normal + y_test_covid #+ y_test_pneumonia
```

In [8]:
```
train_data = list(zip(train_all,y_train_all))
random.shuffle(train_data)
```

```
X_train_all, y_train_all = zip(*train_data)

test_data = list(zip(test_all,y_test_all))
random.shuffle(test_data)
X_test_all, y_test_all = zip(*test_data)

y_train_all = np.array(y_train_all)
y_test_all = np.array(y_test_all)
```

In [9]:
```
# Function that resizes all the images ito images of 224x224 pixels
def image_resize(image,
                 new_dim = (None, None)
                 ):

  img = cv2.resize(image,
                   dsize=new_dim,
                   interpolation=cv2.INTER_CUBIC)
  return img
```

In [10]:
```
# Resizing all the images

X_train_all_resized = [image_resize(img, new_dim=(image_width, image_height)) for img
X_train_all_resized = np.asarray(X_train_all_resized).reshape((len(X_train_all_resized

X_test_all_resized = [image_resize(img, new_dim=(image_width, image_height)) for img i
X_test_all_resized = np.asarray(X_test_all_resized).reshape((len(X_test_all_resized),

# Scaling the data
X_train_all_resized, X_test_all_resized = X_train_all_resized/255.0, X_test_all_resize
```

In [11]:
```
# Reshaping the data
X_train_all_resized_dnn = X_train_all_resized.reshape(X_train_all_resized.shape[0],tot
X_test_all_resized_dnn = X_test_all_resized.reshape(X_test_all_resized.shape[0],total_
```

In [12]:
```
X_train_all_resized_dnn.shape
```

DNN

In [13]:
```
dnn = tf.keras.models.Sequential([

    tf.keras.layers.Dense(256, input_dim=total_imputs, activation='relu'),
    tf.keras.layers.Dropout(0.10, seed=10),
    tf.keras.layers.Dense(128,activation='relu'),
    tf.keras.layers.Dropout(0.10, seed=10),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dropout(0.10, seed=10),
    tf.keras.layers.Dense(32, activation='relu'),
    tf.keras.layers.Dropout(0.10, seed=10),
    tf.keras.layers.Dense(16, activation='relu'),
    tf.keras.layers.Dropout(0.10, seed=10),
    tf.keras.layers.Dense(8, activation='relu'),
    tf.keras.layers.Dropout(0.10, seed=10),
])
```

```
if data=='all':
  dnn.add(tf.keras.layers.Dense(3, activation='softax'))

  dnn.compile(optimizer=tf.keras.optimizers.Adam(),
            loss='sparse_categorical_crossentropy',
            metrics=['accuracy'])
else:
  dnn.add(tf.keras.layers.Dense(1, activation='sigmoid'))

  dnn.compile(optimizer=tf.keras.optimizers.Adam(),
              loss='binary_crossentropy',
              metrics=['accuracy'])

dnn.summary()
```

In [14]:

```
dnn_history = dnn.fit(
    X_train_all_resized_dnn,
    y_train_all,
    epochs=10,
    validation_data=(X_test_all_resized_dnn, y_test_all)
                     )

dnn_loss, dnn_accuracy = dnn.evaluate(X_train_all_resized_dnn, y_train_all, verbose=Fa
print("Training Accuracy: {:.4f}".format(dnn_accuracy))
dnn_loss, dnn_accuracy = dnn.evaluate(X_test_all_resized_dnn, y_test_all, verbose=Fals
print("Testing Accuracy:  {:.4f}".format(dnn_accuracy))

# Making Predictions
y_pred_proba_dnn = dnn.predict(X_test_all_resized_dnn)
y_pred_classes = np.where(y_pred_proba_dnn>0.5,1,0)

# Classification Report
print(classification_report(y_test_all, y_pred_classes))

# Saving the classification report into xlsx file
clf_report = pd.DataFrame(classification_report(y_test_all, y_pred_classes, output_dic
clf_report = clf_report.rename(columns={'index':'label'})
clf_report_labels = clf_report.iloc[:pd.Series(y_test_all).nunique()]
clf_report_metrics = clf_report.iloc[pd.Series(y_test_all).nunique():]
clf_report_labels['label'] = clf_report_labels['label'].astype(int).map(inverse_target
clf_report = pd.concat([clf_report_labels,clf_report_metrics])
clf_report.to_excel(export_directory+'/classification_report_dnn_model.xlsx',index=Fal


# Ploting Roc Curve
fpr_dnn, tpr_dnn, _ = roc_curve(y_test_all,  y_pred_proba_dnn)
auc_dnn = roc_auc_score(y_test_all, y_pred_proba_dnn)

plt.figure(figsize=(12,7))
plt.grid()
plt.plot(fpr_dnn, tpr_dnn, linewidth = 3, color = '#ac0000', label='AUC = {:.3f}'.form
plt.plot(fpr_dnn, fpr_dnn, linestyle='--', linewidth = 3, color='black')
plt.text(0.58,0.3,'AUC = {:.3f}'.format(auc_dnn),  fontsize=25)
plt.ylabel('True Positive Rate', fontsize=12)
plt.xlabel('False Positive Rate', fontsize=12)
plt.title('ROC Curve DNN Model \n',fontsize = 15)
plt.savefig(export_directory + '/ROC_curve_DNN.png')
```

```
plt.show()

# Ploting the confusion matrix of the classification
plot_confusion_matrix(y_test_all,
                      y_pred_classes,
                      labels=target_variable_mapping.keys(),
                      model_name='DNN',
                      export_dir=export_directory)

plot_model(dnn, to_file=export_directory+'/dnn_model_plot.png', show_shapes=True, show
```

VGG16

In [15]:
```
base_model = VGG16(weights='imagenet', include_top=False,
                           input_shape=(image_width, image_height,color_channels))

# freeze extraction layers
base_model.trainable = False

# add custom top layers
x = base_model.output
x = tf.keras.layers.GlobalAveragePooling2D()(x)
# x = tf.keras.layers.Dense(1024 , activation='relu')(x)
# x = tf.keras.layers.Dense(512 , activation='relu')(x)
x = tf.keras.layers.Dense(256 , activation='relu')(x)
x = tf.keras.layers.Dense(128 , activation='relu')(x)
predictions = tf.keras.layers.Dense(1, activation='sigmoid')(x)
vgg16 = tf.keras.Model(inputs=base_model.input, outputs=predictions)

# confirm unfrozen layers
for layer in vgg16.layers:
    if layer.trainable==True:
        print(layer)

vgg16.compile(
  loss='binary_crossentropy',
  optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
  metrics=['accuracy']
)
```

In [16]:
```
vgg16_history = vgg16.fit(
    X_train_all_resized,
    y_train_all,
    epochs = 2,
    validation_data = (X_test_all_resized,y_test_all),
    callbacks=[EarlyStopping(monitor='val_loss', patience=2, verbose=1)])



vgg16_loss, vgg16_accuracy = vgg16.evaluate(X_train_all_resized, y_train_all, verbose=
print("Training Accuracy: {:.4f}".format(vgg16_accuracy))
vgg16_loss, vgg16_accuracy = vgg16.evaluate(X_test_all_resized, y_test_all, verbose=Fa
print("Testing Accuracy:  {:.4f}".format(vgg16_accuracy))
visualkeras.layered_view(vgg16, legend=True)
```

```
# Making predictions
y_pred_proba_vgg16 = vgg16.predict(X_test_all_resized)
y_pred_classes = np.where(y_pred_proba_vgg16>0.5,1,0).reshape(y_test_all.shape[0])

# Classification report
print(classification_report(y_test_all,
                            y_pred_classes
                            ))

# Saving the classification report into xlsx file
clf_report = pd.DataFrame(classification_report(y_test_all, y_pred_classes, output_dic
clf_report = clf_report.rename(columns={'index':'label'})
clf_report_labels = clf_report.iloc[:pd.Series(y_test_all).nunique()]
clf_report_metrics = clf_report.iloc[pd.Series(y_test_all).nunique():]
clf_report_labels['label'] = clf_report_labels['label'].astype(int).map(inverse_target
clf_report = pd.concat([clf_report_labels,clf_report_metrics])
clf_report.to_excel(export_directory+'/classification_report_vgg16_model.xlsx',index=F

# Ploting Roc Curve
fpr_vgg16, tpr_vgg16, _ = roc_curve(y_test_all,  y_pred_proba_vgg16)
auc_vgg16 = roc_auc_score(y_test_all, y_pred_proba_vgg16)

plt.figure(figsize=(12,7))
plt.grid()
plt.plot(fpr_vgg16, tpr_vgg16, linewidth = 3, color = '#297878', label='VGG16 = {:.3f}
plt.plot(fpr_vgg16,fpr_vgg16, linestyle='--', linewidth = 3, color='black')
plt.text(0.58,0.3,'AUC = {:.3f}'.format(auc_vgg16),  fontsize=25)
plt.ylabel('True Positive Rate', fontsize=12)
plt.xlabel('False Positive Rate', fontsize=12)
plt.title('ROC Curve VGG16 Model \n',fontsize = 15)
plt.savefig(export_directory + '/ROC_curve_VGG16.png')
plt.show()

# Confusion matrix
plot_confusion_matrix(y_test_all,
                      y_pred_classes,
                      labels=target_variable_mapping.keys(),
                      model_name='VGG16',
                      export_dir=export_directory)

plot_model(vgg16, to_file=export_directory+'/vgg16_model_plot.png', show_shapes=True,
visualkeras.layered_view(vgg16, legend=True, to_file=export_directory+'/VGG16_model_pl
```

In [17]:
```
fig, ax = plt.subplots(2,figsize=(15,12))


l1 = ax[0].plot(dnn_history.history['val_accuracy'], color='#ac0000', linewidth=3, lab
l2 = ax[0].plot(vgg16_history.history['val_accuracy'], color='#297878', linewidth=3, l

ax[0].set_title('DNN Vs VGG16 Validation Accuracy over epochs \n',fontsize=20)
ax[0].set_xlabel("Epochs",fontsize=15)
ax[0].set_ylabel("Validation Accuracy",fontsize=15)
ax[0].legend([l1, l2], labels= ['DNN accuracy: {0:.2f}'.format(dnn_accuracy),
                                'VGG16 accuracy: {0:.2f}'.format(vgg16_accuracy)],
             loc='lower right')

l3 = ax[1].plot(dnn_history.history['val_loss'], color='#ac0000', linewidth=3, label='
```

```
l4 = ax[1].plot(vgg16_history.history['val_loss'], color='#297878', linewidth=3, label

ax[1].set_title('DNN Vs VGG16 Validation Loss over epochs \n',fontsize=20)
ax[1].set_xlabel("Epochs",fontsize=15)
ax[1].set_ylabel("Validation Loss",fontsize=15)
ax[1].legend([l3, l4], labels= ['DNN loss: {0:.2f}'.format(dnn_loss),
                                'VGG16 loss: {0:.2f}'.format(vgg16_loss)],
            loc='upper right')

plt.subplots_adjust( bottom=0.1,
                     top=0.9,
                     wspace=0.7,
                     hspace=0.7)



plt.savefig(export_directory+'/Models_Performance_all.png')


plt.figure(figsize=(12,7))
plt.grid()
plt.plot(fpr_dnn, tpr_dnn, linewidth = 3, color = '#ac0000', label='DNN = {:.3f}'.for
plt.plot(fpr_vgg16, tpr_vgg16, linewidth = 3, color = '#297878', label='VGG16 = {:.3f}
plt.plot(fpr_vgg16,fpr_vgg16, linestyle='--', linewidth = 3, color='black')
plt.ylabel('True Positive Rate', fontsize=12)
plt.xlabel('False Positive Rate', fontsize=12)
plt.title('ROC Curves for DNN and VGG16 \n',fontsize = 15)
plt.legend()
plt.savefig(export_directory + '/ROC_curve_common.png')
plt.show()
```

9. References

[1] Ah-hwee Tan "Text Mining: The state of the art and the challenges (1999)"

[2] Xiaohua Zhou and Hyoil Han College of Information Science and Technology Drexel University Philadelphia and Isaac Chankai, Ann Prestrud and Ari Brooks

[3] Mahmoud Elbattah, Émilien Arnaud, Maxime Gignon and Gilles Dequen, he Role of Text Analytics in Healthcare: A Review of Recent Developments and Applications"

[4] Han, J., Nandan, N., & Sun, A. (2015). Did You Know? A Rule-Based Approach to Finding Similar Questions on Online Health Forums. In Proceedings of the 2015 International Conference on Healthcare Informatics, pp. 513-514). IEEE.

[5] **Martínez, P., Martínez, J. L., Segura-Bedmar, I., Moreno-Schneider, J., Luna, A., & Revert, R. (2016). Turning user generated health-related content into actionable knowledge through text analytics services. Computers in Industry, 78, 43-56.**

[6] **Chang, M., Chang, M., Reed, J. Z., Milward, D., Xu, J. J., & Cornell, W. D. (2016). Developing timely insights into comparative effectiveness research with a text-mining pipeline. Drug Discovery Today, 21(3), 473-480.**

[7] **Brown, A. D., & Marotta, T. R. (2017). A natural language processing-based model to automate MRI brain protocol selection and prioritization. Academic Radiology, 24(2), 160-166.**

[8] **Castro, S. M., Tseytlin, E., Medvedeva, O., Mitchell, K., Visweswaran, S., Bekhuis, T., & Jacobson, R. S. (2017). Automated annotation and classification of BI-RADS assessment from radiology reports. Journal of Biomedicasl Informatics, 69, 177-187.**

[9] **Pendyala, V. S., & Figueira, S. (2017). Automated medical diagnosis from clinical data. In Proceedings of the IEEE Third International Conference on Big Data Computing Service and Applications (BigDataService), pp. 185-190. IEEE.**

[10] Jelodar, H., Wang, Y., Orji, R., & Huang, H. (2020). Deep sentiment classification and topic discovery on novel coronavirus or covid-19 online discussions: NLP using lstm recurrent neural network approach. IEEE Journal of Biomedical and Health Informatics, vol. 24, no. 10, pp. 2733-2742

[11] **Tvardik, N., Kergourlay, I., Bittar, A., Segond, F., Darmoni, S., & Metzger, M. H. (2018). Accuracy of using natural language processing methods for identifying healthcare-associated infections. International Journal of Medical Informatics, 117, 96-102.**

[12] **Afzal, N., Mallipeddi, V. P., Sohn, S., Liu, H., Chaudhry, R., Scott, C. G., ... & Arruda-Olson, A. M. (2018). Natural language processing of**

clinical notes for identification of critical limb ischemia. International Journal of Medical Informatics, 111, 83-89.

[13] Sterling, N. W., Patzer, R. E., Di, M., & Schrager, J. D. (2019). Prediction of emergency department patient disposition based on natural language processing of triage notes. International Journal of Medical Informatics, 129, 184-188.

[14] Ge, S., Isah, H., Zulkernine, F., & Khan, S. (2019). A scalable framework for multilevel streaming data analytics using deep learning. In Proceedings of the IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC), Vol. 2, pp. 189-194). IEEE.

[15] Kidwai, B., & Nadesh, R. K. (2020). Design and development of diagnostic Chabot for supporting primary health care systems. Procedia Computer Science, 167, 75-84.

[16] Chen, C. H., Hsieh, J. G., Cheng, S. L., Lin, Y. L., Lin, P. H., & Jeng, J. H. (2020). Emergency department disposition prediction using a deep neural network with integrated clinical narratives and structured data. International Journal of Medical Informatics, 104146.

[17] Arnaud, E., Elbattah, M., Gignon, G. & Dequen, G. (2020). Deep learning to predict hospitalization at triage: Integration of structured data and unstructured text. In Proceedings of the 2020 IEEE International Conference on Big Data (Big Data).

[18] Mohan, Vijayarani. (2015). Preprocessing Techniques for Text Mining - An Overview.

[19] Porter, M.F. (1980), "An algorithm for suffix stripping", Program: electronic library and information systems, Vol. 14 No. 3, pp. 130-137.

[20] Fuchun Peng, Nawaaz Ahmed, Xin Li, and Yumao Lu. 2007. Context sensitive stemming for web search. In Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval (SIGIR '07). Association for Computing Machinery, New York, NY, USA, 639–646.

[21] G. Singh, B. Kumar, L. Gaur and A. Tyagi, "Comparison between Multinomial and Bernoulli Naïve Bayes for Text Classification," 2019 International Conference on Automation, Computational and Technology Management (ICACTM), 2019, pp. 593-596, doi: 10.1109/ICACTM.2019.8776800.

[22] AI 2004: Advances in Artificial Intelligence

[23] Z. Wang, X. Sun, D. Zhang and X. Li, "An Optimal SVM-Based Text Classification Algorithm," 2006 International Conference on Machine Learning and Cybernetics, 2006, pp. 1378-1381, doi: 10.1109/ICMLC.2006.258708.

[24] Goyal C. Published On May 18, 2021 and Last Modified On October 26th, 2021. Multiclass Classification Using SVM. Retrieved from

https://www.analyticsvidhya.com/blog/2021/05/multiclass-classification-using-svm/

[25] AGGARWAL, Charu C., et al. Neural networks and deep learning. Springer, 2018, 10: 978-3.

[26] Ali A. Published on December 9, 2019, and Last Modified on February 6, 2023. Building Neural Network (NN) Models in R. Retrieved from https://www.datacamp.com/tutorial/neural-network-models-r

[27] https://www.javatpoint.com/single-layer-perceptron-in-tensorflow

[28] https://en.wikipedia.org/wiki/Perceptron

[29] arXiv:1511.08458

[30] Sluijmers M. Published on July 30, 2020, and Last Modified on September 9, 2020. Convolutional neural network text classification with risk assessment. Retrieved from https://machine-learning-company.nl/en/technical/convolutional-neural-network-text-classification-with-risk-assessment-eng/

[31] Brownlee J. Published on October 9, 2017, and Last Modified on August 7, 2019. A Gentle Introduction to the Bag-of-Words Model.Retrieved from https://machinelearningmastery.com/gentle-introduction-bag-words-model/

[32] Thushan G. Published on May 5, 2019, and Last Modified on July 25, 2022. Intuitive Guide to Understanding GloVe Embeddings. Retrieved from https://towardsdatascience.com/light-on-math-ml-intuitive-guide-to-understanding-glove-embeddings-b13b4f19c010

[33] Prabhu. (2019, November 11). Understanding NLP Word Embeddings Text Vectorization. Retrieved from https://towardsdatascience.com/understanding-nlp-word-embeddings-text-vectorization-1a23744f7223

[34] Brownlee J. Published on October 9, 2017, and Last Modified on August 7, 2019. What Are Word Embeddings for Text? Retrieved from https://machinelearningmastery.com/what-are-word-embeddings/

[35] Uzila A. Published on September 12, 2022, and Last Modified on November 26, 2022. GloVe and fastText Clearly Explained: Extracting Features from Text Data. Retrieved from https://levelup.gitconnected.com/glove-and-fasttext-clearly-explained-extracting-features-from-text-data-1d227ab017b2

[36] Donges N. Published on July 29, 2021 an Last Modified on February 28, 2023. A Guide to Recurrent Neural Networks: Understanding RNN and LSTM Networks. Retrieved from https://builtin.com/data-science/recurrent-neural-networks-and-lstm

[37] Biswal A. Published on April 24, 2020 and Last Modified on February 14, 2023. Recurrent Neural Network (RNN) Tutorial: Types,

Examples, LSTM and More. Retrieved from https://www.simplilearn.com/tutorials/deep-learning-tutorial/rnn

[38] Olah C. (2015, August 27). Understanding LSTM Networks. Retrieved from https://colah.github.io/posts/2015-08-Understanding-LSTMs/

[39] Shekhar S. Published On June 14, 2021 and Last Modified On June 30, 2021. LSTM for Text Classification in Python Retrieved from https://www.analyticsvidhya.com/blog/2021/06/lstm-for-text-classification/

[40] A. Kumar, J. Kim, D. Lyndon, M. Fulham and D. Feng, "An Ensemble of Fine-Tuned Convolutional Neural Networks for Medical Image Classification," in IEEE Journal of Biomedical and Health Informatics, vol. 21, no. 1, pp. 31-40, Jan. 2017, doi: 10.1109/JBHI.2016.2635663.

[41] ARI, ALİ and HANBAY, DAVUT (2018) "Deep learning based brain tumor classification and detection system," Turkish Journal of Electrical Engineering and Computer Sciences: Vol. 26: No. 5, Article 9.

[42] Sadad, T, Rehman, A, Munir, A, et al. Brain tumor detection and multi-classification using advanced deep learning techniques. Microsc Res Tech. 2021; 84: 1296– 1308.

[43] Nawaz, M., Sewissy, A. A., & Soliman, T. H. A. (2018). Multi-class breast cancer classification using deep learning convolutional neural network. International Journal of Advanced Computer Science and Applications, 9(6) doi:https://doi.org/10.14569/IJACSA.2018.090645

[44] Wang, Xi, Hao Chen, Caixia Gan, Huangjing Lin, Qi Dou, Qitao Huang, Muyan Cai, and Pheng-Ann Heng. "Weakly supervised learning for whole slide lung cancer image classification." (2018).

[45] Robini G. Published on September we, 2021 and Last Modified on January 5, 2022. Everything you need to know about VGG16. Retrieved from https://medium.com/@mygreatlearning/everything-you-need-to-know-about-vgg16-7315defb5918

[46] Datagen (2022, November 02). Understanding VGG16: Concepts, Architecture, and Performance. Retrieved from https://datagen.tech/guides/computer-vision/vgg16/