ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ
UNIVERSITY OF PIRAEUS

University of Piraeus
School of Information Technology and Communications
Department of Digital Systems

MSc in Information Systems and Services
Area of study: Big Data and Analytics

Hyperparameter Optimization on Supervised Learning Models

Georgia Lytra
AM: ME1815

Supervisor:
Dimosthenis Kyriazis
Associate Professor

Abstract

Machine learning has taken the technological world by storm in recent years. Every practitioner needs to develop a model that meets the needs of the problem that is faced along with all the available resources that come by. A lot of challenges come up along the way of this process; one of these challenges is the selection of the most appropriate hyperparameters in the developing model. This phase, called hyperparameter optimization, is crucial since on the one hand there are models that have proven to be effective in both performance and execution time, while on the other hand these same models can be rendered rather useless without the appropriate selection of hyperparameters. In addition, hyperparameter tuning can really help a model to shine and exploit its capabilities to the fullest.

Since every problem is unique and complex, domain knowledge is required to select the appropriate hyperparameters in each case; but that is not always possible. A need is on the rise for tools that automatically solve this issue and give information and guidance to the users on how to solve the problem at hand.

This thesis follows an experimental procedure to extract information regarding the appropriate hyperparameters on various supervised learning models. We use datasets with diverse features and characteristics that could assist with the automation of machine learning processes. This is approached through already existing optimization frameworks that have been proven to achieve great results on hyperparameter tuning.

Keywords

hyperparameters, hyperparameter optimization, hyperparameter tuning, supervised learning, hyperopt, scikit-learn, bayesian optimization, tree-structured parzen estimator, classification, regression, machine learning

Table of Contents

1. Introduction

1.1. Motivation

In a 2017 article of the *Economist*, data was compared with what oil was in the 18th century and was characterized as the most valuable resource of the 21st century[1]. We are already going through the fourth industrial revolution, a data-driven revolution where traditional processes are becoming more and more automated to meet the needs of the times. Data is everywhere and they are here to stay.

Indeed, we live in an era where data is produced in tremendous amounts every day; from social media platforms to IoT sensors to apps that assist each of us in our everyday lives, all these play their part in formulating this situation. These data need not only a place to be stored but also a sufficient way to be analyzed to produce results that help answering difficult and complex questions. This is where Artificial Intelligence comes along through the Machine Learning practices to make an impact and change the way things work in various fields.

Machine Learning is involved in many applications where data are present. Most of the time, there is an algorithm that fits well within the definition of a particular problem, its requirements and its peculiarities. Even if a problem can be approached with more than one model, usually there is an optimal solution to it, so the selection of the algorithm is a very important step in the machine learning application process. Another key role is occupied by algorithms' hyperparameters, which also are affected by the different case scenarios and the data that accompany them, and they too need a way to be optimized.

1.2. Problem Statement

In this thesis, we attempt to extract information from various experiments regarding the hyperparameters of several machine learning models. To achieve that, we have dealt with

[1] https://www.economist.com/leaders/2017/05/06/the-worlds-most-valuable-resource-is-no-longer-oil-but-data

classification along with regression models and produced several experiments to spot any differences or similarities in the way that these hyperparameters work.

The procedure that we followed involves a very popular hyperparameter tuning framework, HyperOpt. Hyperopt, which was developed by James Bergsta [3], uses bayesian reasoning to construct an algorithm proposed by its creator. That algorithm is called Tree of Parzen Estimators and has proven to be very effective on the tasks of hyperparameter optimization, especially compared with other hyperparameter search algorithms [1].

The experiments that were conducted are a combination of widely used classification and regression algorithms, along with a variety of popular datasets that each belong to a different sector. This allowed us on the one hand to not get distracted by other issues that concern the process before fitting a model and focus solemnly on hyperparameter optimization, and on the other hand to extract results that concern a variety of diverse problems.

1.3. Thesis structure

Chapter 1 gives some information about the scope and goals of this thesis. Chapter 2 provides all the necessary background on the most commonly used hyperparameter optimization methods and some of the best practices on employing them. Chapter 3 includes all the datasets that are used in this thesis along with a description of the respective features of its dataset. In chapter 4 we explore all the basic information about the algorithms that are used in this work to have a better understanding of the characteristics of each model. Chapter 5 includes all the relevant information about the optimization frameworks that are found in this work, while chapter 6 includes a detailed explanation of the experimental phase and its components. Finally, in chapter 7 we provide some conclusion based on this work along with some potential next steps.

2. Hyperparameter Optimization

Hyperparameters are parameters that are not estimated from the model itself but instead are determined before the training process takes place (e.g. C in Support Vector Machines or k in K-Nearest Neighbors) and can have a significant impact on a model's results. In contrast, a parameter is an internal characteristic of the model and its value can be estimated from the data that is passed along in the model (e.g., beta coefficients in Linear Regression or Support vectors in Support Vector Machines).

The process of selecting the values for these hyperparameters is called hyperparameter tuning or hyperparameter optimization. This process results in the selection of the optimum hyperparameters for a machine learning algorithm, given a set of data. Its practice is crucial before any application of the machine learning model and can greatly affect the model's performance.

Many different methods can be deployed to determine the optimal parameters and return the best fitted model. The following are the most commonly used ones.

2.1. Manual Search

Manual search is the act of hand-picking the models hyperparameters without using any search method. This is commonly preferred when the practitioner is a domain expert and has extended knowledge on the problem along with the data that is handled. The procedure involves manually trying lots of different sets of parameters until the best one is found. An important factor is that the problem under investigation should not be very complex for the manual selection of hyperparameters to be feasible.

2.2. Grid Search

Grid search is an exhaustive search that is performed on specific parameter values of a model. In this case, a grid is predefined with all the possible values of each hyperparameter that will be tuned. After that, a model is built for all possible combinations of this grid, which is later evaluated until the best set is selected. It is more effective than manual search since it covers more possibilities, but it is computationally costly.

2.3. Random Search

Random search uses the same logic as grid search, but instead of performing a detailed search on the defined search space, it randomly chooses and evaluates sample points. A probability distribution of values is specified and a number of samples are drawn from these distributions. Then, the performance of the model is evaluated for each sample that was drawn.

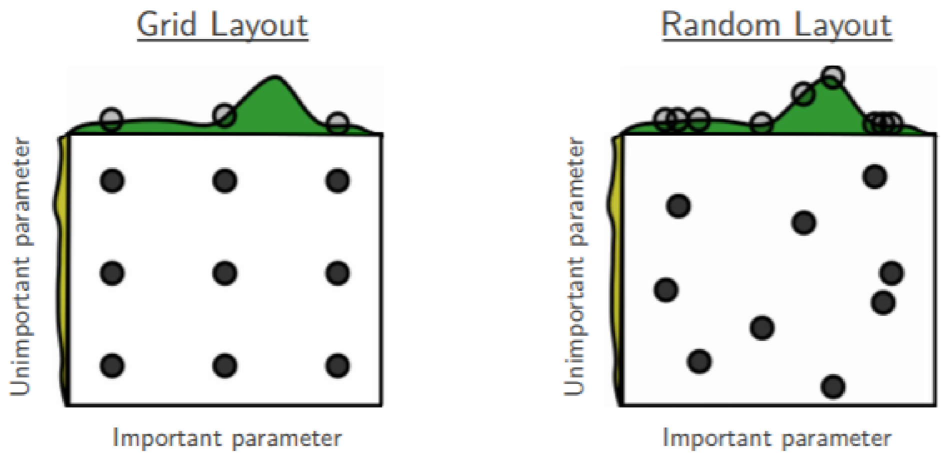
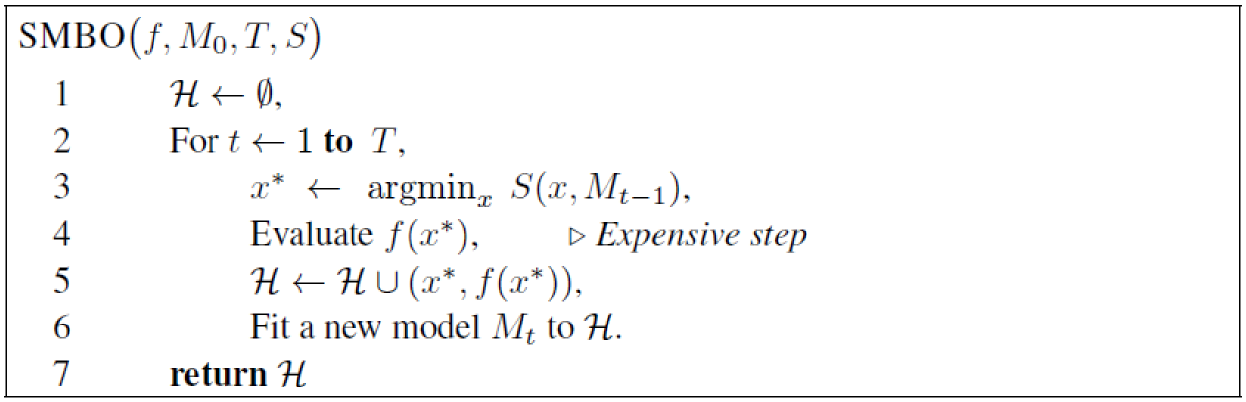


Figure 1: Example of Grid and Random Search of nine exploration sets [31]

2.4. Bayesian Optimization

Bayesian Optimization, or as it is also known Sequential Model-Based Optimization (SMBO), uses the results of the past evaluations to form a probabilistic model of the objective function and later uses this model to choose the next set of hyperparameter values. The probabilistic model is called the surrogate model and is represented by $p(x|y)$; y being the performance metric for the model and x being the hyperparameter values [18]. In hyperparameter optimization, the objective function is a function that maps the hyperparameter values to the model's chosen performance metric, either on a validation set or maybe by using cross validation.



```
SMBO($f, M_0, T, S$)
    $\mathcal{H} \leftarrow \emptyset$,
    For $t \leftarrow 1$ **to** $T$,
        $x^* \leftarrow \text{argmin}_x\ S(x, M_{t-1})$,
        Evaluate $f(x^*)$,          ▷ *Expensive step*
        $\mathcal{H} \leftarrow \mathcal{H} \cup (x^*, f(x^*))$,
        Fit a new model $M_t$ to $\mathcal{H}$.
    **return** $\mathcal{H}$
```

Listing 1: pseudo-code of Sequential Model-Based Optimization [1]

3. Datasets

A variety of datasets exists and is easily accessible through public databases. In the Financial sector, models are used for fraud detection or loan default prediction. In Pharma and Medicine, researchers use many machine learning techniques for drug discovery, clinical trial research or epidemic outbreak prediction, aiming to save thousands of lives each year. Other applications involve computer vision or time series forecasting, which are utilized by many companies that offer their services to the public.

One reasonable question is why were these specific datasets chosen for either the classification or regression tasks. The goal was to have a variety of datasets from different sectors and fields to examine different cases, but also to include datasets with diverse features. Each of them is quite popular and therefore didn't require extensive processing, which fitted along with this work as the purpose of the study was not to emphasize in difficult preprocessing techniques but to focus on the results that come after. A short description of each one that was utilized in this work follows.

Table 1, which is shown right below, summarizes the data sets used in this study. For each data set, it includes the name, the number of attributes, the number of rows, the field that the dataset belongs to and the type of task that the dataset was used for.

	Number of Columns	**Number of Rows**	**Number of classes**	**Domain**	**Task Type**
Iris	5	150	3	Flora	Classification
Breast Cancer Wisconsin	11	569	2	Medicine	Classification
Wine Quality (Red)	12	1599	10	Food / Drinks	Classification
Titanic	8	887	2	Marine	Classification
Banknote Authentication	5	1372	2	Banking	Classification
MNIST	64	1797	10	Digits	Classification
Melanoma	31	3632	3	Medicine	Classification

Boston House Pricing	14	506	-	Real Estate	Regression
Adalone	8	4177	-	Life	Regression

Table1: Summary of all datasets

3.1. Iris Flowers

The Iris Flowers Dataset involves predicting the flower species, given measurements of iris flowers. The rows of the table represent an iris flower, including its species and dimensions of its botanical parts, sepal and petal, in centimeters [8].

Variable information:

1. Sepal length in cm
2. Sepal width in cm
3. Petal length in cm
4. Petal width in cm
5. Class (Iris Setosa, Iris Versicolour, Iris Virginica)

3.2. Breast Cancer Winsconsin

This is a well-known dataset for breast cancer diagnosis systems. Features are computed from a digitized image of a fine needle aspirate of a breast mass. They describe characteristics of the cell nuclei present in the image [9].

Variable information:

1. ID number
2. Diagnosis (M = malignant, B = benign)

   Ten real-valued features are computed for each cell nucleus:

   a. radius (mean of distances from center to points on the perimeter)
   b. texture (standard deviation of gray-scale values)

c. perimeter
d. area
e. smoothness (local variation in radius lengths)
f. compactness (perimeter^2 / area - 1.0)
g. concavity (severity of concave portions of the contour)
h. concave points (number of concave portions of the contour)
i. symmetry
j. fractal dimension ("coastline approximation" - 1)

3.3. Titanic

The dataset contains data for 887 of the real Titanic passengers. Each row represents one person. It describes the survival status of individual passengers on the Titanic ship. The columns describe different attributes about the person including whether they survived, their age, their passenger-class, their sex and the fare they paid. It was created to be part of an entry level competition on the kaggle platform [10].

3.4. Wine Quality

The Wine Quality Dataset involves predicting the quality of wines on a scale given chemical measures of each wine [11].

Variable information:

1. Fixed acidity
2. Volatile acidity
3. Citric acid
4. Residual sugar
5. Chlorides
6. Free sulfur dioxide
7. Total sulfur dioxide
8. Density

9. pH
10. Sulphates
11. Alcohol
12. Quality (score between 0 and 10)

3.5. Banknote Authentication

The Banknote Dataset involves predicting whether a given banknote is authentic given a number of measures taken from a photograph. Data were extracted from images that were taken from genuine and forged banknote-like specimens. For digitization, an industrial camera usually used for print inspection was used. The final images have a size of 400x400 pixels. Due to the object lens and the distance to the investigated object, gray-scale pictures with a resolution of about 660 dpi were gained. Wavelet Transform tools were used to extract features from images [12].

Variable information:

1. Variance of Wavelet Transformed image
2. Skewness of Wavelet Transformed image
3. Kurtosis of Wavelet Transformed image
4. Entropy of image
5. Class (0 for authentic, 1 for inauthentic)

3.6. MNIST

The MNIST database (Modified National Institute of Standards and Technology database) of handwritten digits consists of a training set of 60,000 examples, and a test set of 10,000 examples. The task is to classify a given image of a handwritten digit into one of 10 classes representing integer values from 0 to 9, inclusively. Its format though is a little chaotic to use and therefore a simpler csv file was created for classification problems.

The new csv format consists of a label in the beginning, which is the actual digit that the handwritten digit is supposed to represent, and the subsequent values are the pixel values of the handwritten digit [13].

3.7 Melanoma

This dataset is a highly unbalanced dataset containing information about features regarding melanoma detection. The goal is to predict whether the instance is a melanoma or a kind of nevus. It contains 31 features and it consists of 3631 instances. The classes to choose from to predict are the following:

- Class 1: Melanoma
- Class 2: Dysplastic Nevus
- Class 3: Non Dysplastic Nevus

3.8. Boston House Pricing

This dataset contains information collected by the U.S Census Service concerning housing in the area of Boston Mass. It was obtained from the StatLib archive, and has been used extensively throughout the literature to benchmark algorithms [14].

The attributes are:

1. CRIM: Per capita crime rate by town
2. ZN: Proportion of residential land zoned for lots over 25,000 sq. ft
3. INDUS: Proportion of non-retail business acres per town
4. CHAS: Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)
5. NOX: Nitric oxide concentration (parts per 10 million)
6. RM: Average number of rooms per dwelling
7. AGE: Proportion of owner-occupied units built prior to 1940
8. DIS: Weighted distances to five Boston employment centers
9. RAD: Index of accessibility to radial highways
10. TAX: Full-value property tax rate per $10,000
11. PTRATIO: Pupil-teacher ratio by town
12. B: $1000(Bk — 0.63)^2$, where Bk is the proportion of [people of African American descent] by town
13. LSTAT: Percentage of lower status of the population
14. MEDV: Median value of owner-occupied homes in $1000s

3.9. Abalone

The dataset's objective is predicting the age of the abalone from physical measurements. The age of abalone is decided by cutting the shell through the cone, staining it, and counting the number of rings through a microscope. Other measurements are also used in this dataset to predict the age and a slight modification was applied in it regarding the missing values and scaling of data [15].

Attribute Information:

1. Sex: M, F, and I (infant)
2. Length: Longest shell measurement
3. Diameter: perpendicular to length
4. Height: with meat in shell
5. Whole weight: whole abalone
6. Shucked weight: weight of meat
7. Viscera weight: gut weight (after bleeding)
8. Shell weight: after being dried
9. Rings: +1.5 gives the age in years

4. Supervised Learning Models

Supervised Learning consists of methods and techniques that rely on prior information regarding the output values, rather than attempting to conduct the analysis without any extra knowledge or intervention. It includes algorithms for performing either classification or regression analyses and predictions. Some examples involve Linear Regression for regression problems, Random forest for classification and regression problems, or Support vector machines for both classification and regression problems.

This chapter is a brief description of all the models that were used in this work to better understand how each of them works and therefore how each of them is affected by hyperparameter tuning.

4.1 Classifiers

4.1.1 Random Forest

Random Forest (RF) is an ensemble classifier. It represents a set of many classifiers, in this case many binary decision trees, to combine the decision of each classifier with the scope to classify new examples.

According to [25], assuming a training set drawn randomly for the distribution of the random vector Y, X and given an ensemble of classifiers $h_1(x), h_2(x), \ldots, h_K(x)$, the margin function, i.e, the function that measures the extent to which the average number of votes at Y, X for the right class exceeds the average vote for any other class, is defined as:

$$mg\,(X,\,Y) \;=\; a\,u_k\,I\,(h_k(X) \,=\, Y) \,-\, max_{i\neq Y} a\,\upsilon_k\,I(h_k(x) \,=\, j)$$

Where where $I(\cdot)$ is the indicator function. The margin measures the extent to which the average number of votes at Y, X, for the right class exceeds the average vote for any other class. The lesser the margin the less confidence in the classification. In the same source ([25]) we may also find the definition for the generalization error:

$$PE^{*} = P_{X,Y}(mg(X, Y) < 0)$$

where the subscripts Y, X indicate that the probability is over the Y, X space.

The pseudo code for the Random Forest model are presented in the following figure:

```
To generate $c$ classifiers:
for $i = 1$ to $c$ do
    Randomly sample the training data $D$ with replacement to produce $D_i$
    Create a root node, $N_i$ containing $D_i$
    Call BuildTree( $N_i$ )
end for

BuildTree(N):
if $N$ contains instances of only one class then
    return
else
    Randomly select x% of the possible splitting features in $N$
    Select the feature $F$ with the highest information gain to split on
    Create f child nodes of $N$ , $N_1$,..., $N_f$ , where $F$ has $f$ possible values ( $F_1$, … , $F_f$ )
    for $i = 1$ to $f$ do
        Set the contents of $N_i$ to $D_i$ , where $D_i$ is all instances in $N$ that match
        $F_i$
        Call BuildTree( $N_i$ )
    end for
end if
```

Listing 2: Pseudo code for Random Forest model [21]

4.1.2. Support Vector Machine

Support Vector Machine (SVM) finds a hyperdimensional plane that separates distinct classes. SVM finds the hyperplane such that the margin is maximum.

SVM algorithms are based on the concept of mapping data points from low-dimensional into high-dimensional space to make them linearly separable; a hyperplane is then generated as the classification boundary to partition data points. Assuming there are n data points, the objective function of SVM is [23]:

$$arg\ min_w \left\{ \frac{1}{n} \sum_{i=1}^{n} max \left\{ 0,\ 1\ -\ y_i f(x_i) \right\} +\ Cw^T w \right\},$$

where w is a normalization vector; C is the penalty parameter of the error term, which is an important hyper-parameter of all SVM models.

The pseudocode for the SVM model is presented below:

```
Input :
        $N_{in}$ (the number of input vectors),
        $N_{sv}$ (the number of support vectors),
        $N_{ft}$ (the number of features in a support vector),
        $SV[N_{sv}]$ (support vector array),
        $IN[N_{in}]$ (input vector array),
        $b^*$ (bias)
Output :
        $F$ (decision function output)

for $i \leftarrow 1$ to $N_{in}$ by 1 do
    $F = 0$
    for $j \leftarrow 1$ to $N_{sv}$ by 1 do
        $dist = 0$
        for $k \leftarrow 1$ to $N_{ft}$ by 1 do
            $dist += (SV[j].feature[k]$ - $IN[i].feature[k])^2$
        end
        $\kappa = \exp(-\gamma \times dist)$
        $F += SV[j].\alpha^* \times \kappa$
    end
    $F = F + b^*$
end
```

Listing 3: Pseudocode of SVM model [23]

4.1.3. K Nearest Neighbor

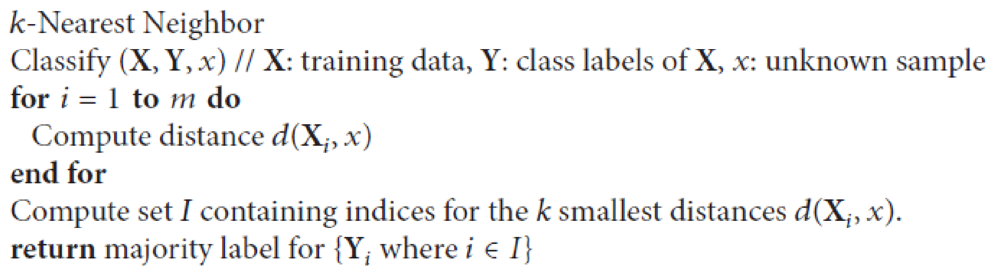
K Nearest Neighbor (KNN) finds the K samples, which is the number of nearest points, in the closest proximity to the point that is to be predicted.

Given a training set $T = \{(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)\}$, where x_i is the feature vector of an instance and $y_i \in \{c_1, c_2, \dots, c_m\}$ is the class of the instance, $i = (1, 2,..., n)$, for a test instance x, its class y can be denoted by:

$$y = arg\,max_{c_j} \sum_{x_i \in N_k(x)} I(y_i = c_i),\ i = 1, 2, \dots, m\ ,$$

where $I(x)$ is an indicator function, $I = 1$ when $y_i = c_i$, otherwise $I = 0$; $N_k(x)$the field involving the k-nearest neighbors of x [7].

The step required to implement the KNN algorithm is outlined in the pseudocode below:



```
$k$-Nearest Neighbor
Classify $(\mathbf{X}, \mathbf{Y}, x)$ // $\mathbf{X}$: training data, $\mathbf{Y}$: class labels of $\mathbf{X}$, $x$: unknown sample
for $i = 1$ to $m$ do
  Compute distance $d(\mathbf{X}_i, x)$
end for
Compute set $I$ containing indices for the $k$ smallest distances $d(\mathbf{X}_i, x)$.
return majority label for $\{\mathbf{Y}_i$ where $i \in I\}$
```

Listing 4: Pseudo Code of KNN classification [20]

4.1.4. Logistic Regression

Logistic Regression (LR) is a kind of regression employed to predict the probability of a binary output X from an input dataset.

Consider a collection of p independent variables denoted by the vector $x' = (x_1, x_2, \dots, x_p)$. Let the conditional probability that the outcome is present be denoted by $P(Y = 1|\, x) = \pi(x)$. The logit of the multiple logistic regression model is given by the equation [19]:

$$g(x) = \beta_0 + \beta_1 x_1 + ... + \beta_p x_p\ ,$$

in which case the logistic regression model is:

$$\pi(x) = \frac{e^{g(x)}}{1+e^{g(x)}}.$$

For classification one against all, the training algorithm is constructed from several binary classifiers, which use the logistic regression model shown above. The algorithm in pseudocode for one vs all is presented in:

```
Input:
    • Training algorithm L (logistic regression)
    • Sample matrix X
    • Labels vector $y = [1, \dots K]$
    • Initial regresor parameters vector $\theta_i$

Main:
    For i=1:K
        Create a new binary vector $y_i$ for each label
        where $y_i = 1$ if it belong to the label and $y_i = 0$
        if it does not belong.

        Apply L to X to find $\theta_i$
Output:
    $\theta_i$ Parameters vector for each regresor
```

Listing 5: One vs all algorithm Pseudocode [24]

4.1.5. Multinomial Naive Bayes

Multinomial Naive Bayes (MNB) is designed for multinomially distributed data based on the naive Bayes algorithm.

As explained in [7], assuming there are *n* features, and θ_{yi} is the distribution of each value of the target variable *y*, which equals the conditional probability P $(x_i | y)$ when a feature value *i* is involved in a data point belonging to the class *y*; based on the concept of relative frequency counting, θ_y can be estimated by a smoothed version of θ_{yi}:

$$\widehat{\theta}_{yi} = \frac{N_{yi} + a}{N_y + an},$$

where N_y is the sum of all N_{yi} $(i = 0, 1, 2, ..., n)$ and N_{yi} is the number of times that feature i is in a data point belonging to class y. The smoothing priors $a \geq 0$ are used for features that are not in the learning samples. When $a = 1$, it is called Laplace smoothing; when $a < 1$, it is called Lidstone smoothing [32].

The pseudo code of the model is shown below:

```
Input: training set $T$, hold-out set $H$, initial number of components $k_0$, and convergence thresholds $\delta_{EM}$ and $\delta_{Add}$.

  Initialize $M$ with one component.
  $k \leftarrow k_0$
  **repeat**
    Add $k$ new mixture components to $M$, initialized using $k$ random examples from $T$.
    Remove the $k$ initialization examples from $T$.
    **repeat**
      E-step: Fractionally assign examples in $T$ to mixture components, using $M$.
      M-step: Adjust parameters of $M$ to maximize the likelihood of this fractional assignment.
      If $\log P(H|M)$ is the highest we've seen so far, save $M$ in $M_{best}$.
    **until** $\log P(H|M)$ fails to improve by ratio $\delta_{EM}$ over the last iteration.
    $k \leftarrow 2 \times k$
  **until** $\log P(H|M)$ fails to improve by ratio $\delta_{Add}$ over the last iteration.
  Execute E-step and M-step twice more on $M_{best}$, using examples from both $H$ and $T$.
  Return $M_{best}$.
```

Listing 6: Pseudo code for Naive Bayes Estimator [22]

4.2. Regressors

4.2.1. Ridge

Ridge regression develops a model that minimizes the sum of the squared prediction error in the training data and an L2-norm regularization, i.e., the sum of the squares of regression coefficients. The function is as below [26][29]:

$$min_b \sum_{i=1}^{N} (f(x_i - y_i))^2 + \lambda \sum_{j=1}^{P} \left|\left|b_j\right|\right|^2$$

This procedure can shrink the regression coefficients, resulting in better generalizability for predicting unseen samples. In this algorithm, a regularization parameter λ is used to control the trade-off of penalties between the bias and variance. A large λ corresponds to more penalties on variance, and a small λ corresponds to more penalties on bias [27].

4.2.2. Lasso

Least Absolute Shrinkage and Selection Operator (LASSO) is another regularized version of Linear Regression. LASSO regression applies L1-norm regularization to the Ordinary Least Squares loss function, aiming to minimize the sum of the absolute value of the regression coefficients [28][29]. The objective function takes the form as below:

$$min_b \sum_{i=1}^{N} \left(f(x_i - y_i) \right)^2 + \lambda \sum_{j=1}^{P} \left| b_j \right|^2$$

This L1-norm regularization typically sets most coefficients to zero and retains one random feature among the correlated ones. Thus, LASSO regression results in a very sparse predictive model, which facilitates optimization of the predictors and reduces the model complexity [30].

5. Optimization Frameworks

5.1. Scikit Learn

Scikit Learn, or sklearn as it is widely known, is a Python library for machine learning and statistical modeling including classification, regression and clustering. This package focuses on making easier the use of machine learning to non-specialists using a general-purpose high-level language. Emphasis is put on ease of use, performance, documentation, and API consistency. For its creation sklearn utilizes other libraries such as Numpy, Scikit or Libsvm for the implementation of SVM among others [5].

5.2. Hyperopt

HyperOpt is an open-source Python library for hyperparameter search developed by James Bergstra. One of its core characteristics is that it uses a form of Sequential Model Based Optimization: the Tree of Parzen Estimators (TPE) [2].

The SMBO sequentially narrows down the search space of values using information from previous results. The TPE algorithm aims to achieve this by optimizing the criterion of Expected Improvement (EI). Expected improvement is the expectation under some model M of $f: X \rightarrow \Re^N$ that $f(x)$ will exceed (negatively) some threshold y [1]:

$$EI_{y^*}(x) := \int_{-\infty}^{\infty} max\,(y^* - y,\ 0)p_M(y|x)dy$$

The basic idea of TPE is that, unlike what a Gausian-process approach would do, instead of modeling directly the $p(y|x)$ (i.e., the surrogate model in the case of hyperparameter optimization), the strategy changes to model $p(x|y)$ and $p(y)$ separately, y being the performance metric for the model and x being the hyperparameter values. The TPE defines $p(x|y)$using two densities [1]:

$$p(x|y) = \begin{cases} l(x) & if\ y < y^* \\ g(x) & if\ y \geq y^* \end{cases}$$

where $l(x)$ is the density formed by using the observations $\{x^{(i)}\}$ such that corresponding loss $f(x^{(i)})$ was less than y^*, and $g(x)$ is the density formed by using the remaining observations.

There are four parts that we need to focus on when using hyperopt [3]:

1. Define an objective function which takes an input and returns a loss to minimize (i.e. cross validation).
2. Specify a configuration space, which is the range of input values to evaluate. A number of options are available in hyperopt for describing the distribution of these values such as:
    a. hp.choice(label, options): This is usually used for categorical parameters, it returns one of the *options*, which should be a list or tuple.
    b. hp.randint(label, upper): This returns a random integer in the range of *label* and *upper.*
    c. hp.uniform(label, low, high): It returns a uniform value between *low* and *high.*
    d. hp.normal(label, mu, sigma): This returns a real value that's normally distributed with mean *mu* and standard deviation *sigma.*
3. Define a search algorithm: the method used to construct the surrogate function and choose the next values to evaluate. Hyperopt currently supports:
    a. Random Search
    b. Tree Parzen Estimator
    c. Adaptive Tree Parzen Estimator
4. Create a "trials object" to store the results of the process.

How Hyperopt works can be summarized in the next five steps [3]:

1. Build a surrogate probability model of the objective function.
2. Locate the hyperparameters that best perform on the surrogate model .
3. Fit said hyperparameters to the objective function.
4. Update the surrogate model and include the new results.
5. Repeat steps 2 to 4 until max iterations are reached.

6. Experimental Results

The process of the experimental phase that we followed in this work is rather straightforward. As a first step, the data preprocessing took place right before the separation of each dataset into train and test sets. Following, every model that is mentioned in section 4 was applied in the data that are explained in section 3, thus covering all possible combinations. At this stage, we applied the default values for the parameters that sklearn assigns in every model. Since the goal is to compare the hyperparameters on both frameworks, we move on to evaluate the performance of the model on these values with the respective metrics. Then, hyperparameter optimization takes place using hyperopt. Afterwards, the proposed hyperparameters are fitted in the same model that was fitted previously and the evaluation step takes place so we have the results to compare the two frameworks.

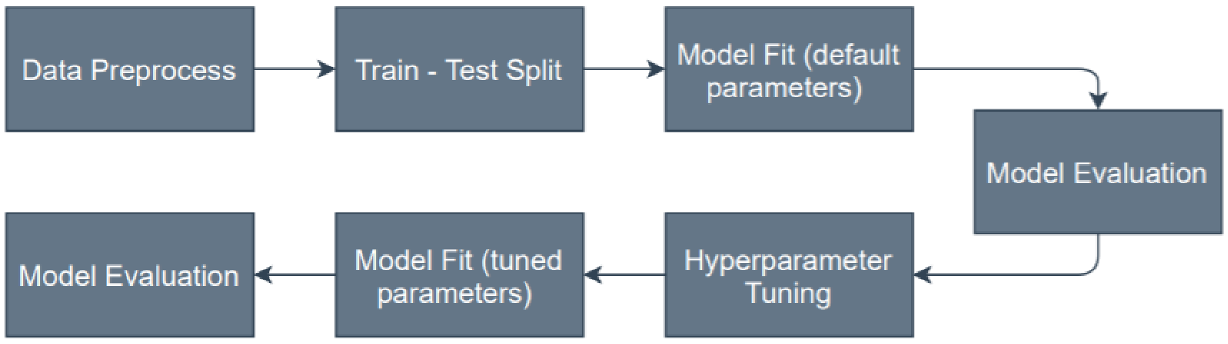


Figure 2: Process of experimental procedure

6.1. Data Preprocessing

Data preparation is a critical step before training any machine learning model and therefore the techniques used have to be chosen carefully. It can seriously impact both the performance of the models and the effectiveness of the attempts to avoid the so very dreadful overfitting. All datasets used in this work have undergone the same preparation techniques to ensure as much of a unified result as possible, with only a few necessary exceptions. This preparation includes only basic yet inevitable alterations in the datasets, since the goal is to examine the characteristics of each of them and make assumptions. Therefore, techniques such as feature selection have been avoided in order to make room for further information to look at that might affect the results.

On the datasets used for classification, a big issue was the balance of the classes. Unbalanced classes can very easily lead to overfitting of the model. The solution to this is either to upsample the cases of the lesser class, or to undersample the cases of the majority class. We chose to implement the former, to produce more data and use them on the training and test phases rather than reducing the dataset's cases, since some of the sets already contained very few data as it is.

The next step was to split our data into two subsets: the training dataset and the testing dataset. The ratio of the split is 70/30. The rationale behind this is that we want to fit our model on the training data, but we also want to have a different portion of the original dataset to make predictions on and to evaluate the performance of our model with the appropriate metrics that fit the classification and regression tasks respectively. The only unique occurrence in this step was in the Iris dataset in combination with the KNN algorithm. Here, we split the data in a 15/85 ratio between training and testing set, respectively, because when the model was fitted, the number of neighbors was greater than the number of samples in the data for some of the runs of Hyperopt. This resulted in an error; thus, to avoid this, instead of the analogy that was used in the rest of the cases, we applied the aforementioned.

With the aim to avoid data leakage, scaling of the data was implemented after the split strategy. The data were scaled on a range of 0 to 1 and this proved to be useful, not only for dodging the overfitting, but also to improve overall accuracy of the models in both frameworks. Another important result that was affected by scaling is the execution time of the hyperparameter tuning using the Hyperopt framework. In many cases, it made the process run much faster, whereas before it would be executed in a significant amount of time.

Since these data sets were chosen for their popularity and characteristics, they did not need much tidying up before we started using them. As a result, we were lucky enough to not run into any NaN values and therefore no technique was used to deal with this issue. This helped with the end result quite much, since the goal of this study would not have been affected by their existence.

We now move on to the hyperopt implementation. The first step of the hyperparameter tuning was to define the configuration space upon which the framework would choose the best value for each case. The search space was defined for each hyperparameter of every algorithm that we used separately, although the same search space was used for every experiment. The objective function was the cross validation score for both the classification and regression task, and the search algorithm we chose was the TPE algorithm option, to utilize hyperopt's capabilities to the fullest.

6.2. Metrics

Regarding the metric functions, three of the most commonly used were put into use: the accuracy score, the cross validation score and the RMSE (Root Mean Square Error) score.

The accuracy, which is the fraction of number of correct predictions that came from our model against the total number of observations, was used to compare the classification model's performance, first we measured the accuracy with the default values that sklearn assigned on each parameter and then we measured it with the values that came from the tuning of those parameters with hyperopt.

For the regression tasks we calculated the RMSE score. It is defined as the square root of the differences between the predicted variables of the model and the corresponding observed values. It was implemented to compare the results from the model that was fitted using the default parameters of sklearn against the one that was fitted with the parameters that were tuned via hyperopt.

The cross validation splits the data into groups. In our case, we choose it to be k=5 groups, which is the default value of sklearn [17]. It then proceeds to hold out a set at a time and train the model on the remaining set. In the end, it combines the results of each iteration to produce the final result. This method was used as a metric of evaluation during the hyperparameter tuning phase to find out which of the trials was the best.

6.3. Hyperopt' s Performance

The resulting accuracy for all classification and regression models under both the sklearn and hyperopt frameworks, as well as the elapsed execution time for the case of the latter, have been compiled into Table 2. Note that the execution time for the cases where sklearn was employed is not presented in this table. The reason for this is that in all these cases we used sklearn's default values for the hyperparameters. On the contrary, hyperopt effectively calculates optimized values for the hyperparameters, hence possibly incurring a non-negligible time overhead, which is, after all, of great interest within the scope of this work.

Dataset	**Sklearn**	**Hyperopt**	**Execution Time(HPO)**
Classification (Metric: Accuracy)			
1. Iris			
RF	0.978	0.956	48min 57s
KNN	0.957	0.957	7min 55s
SVM	0.978	0.978	1min 49s
NB	0.933	0.8	29.5 s
LR	0.911	0.967	5min 8s
2. Breast Cancer			
RF	0.965	0.947	48min 26s
KNN	0.965	0.959	12min 9s
SVM	0.982	0.982	1min 59s
NB	0.83	0.836	42.8 s
LR	0.977	0.988	5min 6s
3. Titanic			
RF	0.768	0.622	47min 42s
KNN	0.76	0.76	16min 1s
SVM	0.768	0.794	2min 18s

NB	0.798	0.798	55.8 s
LR	0.764	0.764	5min 33s
4. Wine Quality			
RF	0.652	0.575	46min 38s
KNN	0.565	0.648	15min 11s
SVM	0.583	0.617	10min 21s
NB	0.49	0.49	57 s
LR	0.554	0.571	5min 3s
5. Banknote Authentication			
RF	0.996	0.985	44min 1s
KNN	0.996	0.998	8min 13s
SVM	0.993	0.978	2min 20s
NB	0.679	0.679	31 s
LR	0.976	0.991	5min
6. MNIST (csv)			
RF	0.972	0.946	49min 28s
KNN	0.987	0.9833	16min 32
SVM	0.987	0.989	17min 38s
NB	0.893	0.893	50.9 s
LR	0.967	0.972	6min 21s
7. Melanoma			
RF	0.803	0.737	2h 5min 17s
KNN	0.7599	0.8162	1h 40min 1s
SVM	0.749	0.805	2h 53min 13s
NB	0.555	0.691	13min 10s
LR	0.682	0.704	1h 55min 28s

Regression (Metric: RMSE)			
1. Boston House Pricing			
Linear	4.64	4.64	1min 5s
Lasso	5.15	4.64	7min 11s
Ridge	4.64	4.71	8min 43s
2. Adalone			
Linear	2.57	2.57	58.7 s
Lasso	2.57	2.57	31min 51s
Ridge	2.57	2.57	32min 42s

Table 2: Accuracy of each experiment in both classification and regression models

From the above table we conclude to the following:

In Random forest, the execution time appears to have been significantly affected when the number of estimators in the search space was increased. More specifically, by increasing the number of searched estimators from 100 to 150, each process required around ~20 minutes more to be completed.

Another factor that appears to be affecting the execution time is the process of scaling the datasets. Between the StandardScaler and the MinMaxscaler, which were tested in this work, the latter was chosen. Apart from its performance benefits, it enables the scaling of the data into values between 0 and 1, thus allowing for frictionless compatibility with the Naive Bayes algorithm, which cannot be employed at all when negative values exist among the data. Furthermore, models that use distance metrics are sensitive to the distribution of input data. Scaling resolves this issue, hence enabling all the data to have the same influence over the distance metrics.

Moreover, by examining the number of parameters in combination with the execution time elapsed, we observe that, in the cases of the classification tasks, those two appear to be independent to each other, since the models that have the most parameters are Random Forest, SVM, and KNN. On the other hand, the number of parameters seems to be playing a crucial role in execution time for the regression tasks, since the hyperparameter optimization for Linear Regression is executed relatively fast compared to the other two regression models, which both have a significantly bigger amount of parameters.

On the Wine Quality dataset we observe that the accuracy is generally low. This could be explained by the fact that the dataset's dependent variable contains many classes which is a factor that may be tampering with the results, as well as the information available; it is after all a dataset that could be used for regression tasks too.

Two of the most performant hyperparameter optimization procedures are observed in the cases of the Wine Quality and the Titanic datasets. Both of them contain many features (12 and 8, respectively), and this could lead to the assumption that datasets with many independent variables could benefit from hyperparameter optimization.

We validate the above observation through the case of the Melanoma dataset. The dataset incorporates 31 distinct features, and is in fact the dataset with the greatest number of features among all the examined ones. Indeed, we can verify that it performed better than the rest of the datasets in all cases, with the exception of the Random Forest algorithm. In addition, we should note that the Melanoma dataset also exhibits the highest execution time in general. However, even though for the rest of the datasets the Random Forest algorithm used to be the slowest case, we observe that SVM turned out to be the slowest in this case.

6.4. Hyperparameters

Table 3 contains the results of the values of the experimental phase for every hyperparameter. The first column with values is the default values that sklearn assigns in each model in the case that none of them is specified. What follows is the values that Hyperopt assigned in each hyperparameter after the optimization process in each dataset. The last column contains the search space for every hyperparameter which consists of the set of all possible values that the parameter could take, along with the type of the parameter. We then proceed to explain our experimental results.

Classification									
Parameters	**Sklearn (default)** [17]	**HPO Iris**	**HPO Breast Cancer**	**HPO Titanic**	**HPO Wine Quality**	**HPO Banknote**	**HPO MNIST**	**HPO Melanoma**	**Search Space**
Random Forest (Number of parameters = 19)									
n_estimators	*100*	32	116	23	144	34	143	77	discreet: [1, 150]
criterion	*gini*	gini	entropy	gini	entropy	entropy	entropy	entropy	categorical: [gini, entropy]
max_depth	*None*	76	40	51	39	81	44	24	discreet: [1, 100]
min_samples _split	*2*	7	4	5	5	7	3	2	discreet: [2, 10]
min_samples _leaf	*1*	5	15	12	6	3	4	16	discreet: [1, 50]

min_weight_fraction_leaf	*0.0*	0.2016	0.2864	0.2238	0.0107	0.009	0.0008	0.0001	continuous: [0.0, 0.5]
max_features	*auto*	auto	None	sqrt	None	log2	sqrt	None	categorical: [None, auto, sqrt, log2]
max_leaf_nodes	*None*	21	61	37	26	21	28	55	discreet: [2,65]
min_impurity_decrease	*0.0*	0.1992	0.6012	0.0478	0.0003	0.0009	0.0165	0.0006	continuous: [0.0, 0.9]
min_impurity_split	*None*	-	-	-	-	-	-	-	-
bootstrap	*True*	True	True	True	True	True	True	TRue	boolean: True
oob_score	*False*	True	False	False	True	False	False	True	boolean: [True, False]
n_jobs	*None*	-1	-1	-1	-1	-1	-1	-1	discreet: -1
random_state	*None*	42	42	42	42	42	42	42	discreet: 42
verbose	*0*	0	0	0	0	0	0	0	discreet: 0
warm_start	*False*	False	True	False	False	False	False	True	boolean: [True, False]
class_weight	*None*	balanced	None	None	None	balanced_subsample	balanced_subsample	balanced	categorical: [None, balanced,balanced_subsample]
ccp_alpha	*0.0*	0.0285	0.4754	0.0714	0.0017	0.0007	0.0175	0.000	continuous:

									[0.0, 1.0]
max_samples	*None*	0.2561	0.3471	0.4696	0.5957	0.5784	0.4729	0.5235	continuous: [0.0, 1.0]
K Nearest Neighbors (Number of parameters = 8)									
n_neighbors	*5*	9	16	19	90	18	3	1	discreet: [1,100]
weights	*uniform*	uniform	distance	uniform	distance	distance	distance	distance	categorical: [uniform, distance]
algorithm	*auto*	brute	brute	brute	brute	kd_tree	kd_tree	brute	categorical: [auto, ball_tree, kd_tre, brute]
leaf_size	*30*	48	37	19	48	40	46	16	discreet: [1, 50]
p	*2*	11	2	4	12	4	3	1	discreet: [1, 15]
metric	*minkowski*	euclidean	minkowski	chebyshev	euclidean	euclidean	minkowski	euclidean	categorical: [euclidean, manhattan, chebyshev, minkowski]
metric_params	*None*	-	-	-	-	-	-	-	-
n_jobs	*None*	-1	-1	-1	-1	-1	-1	-1	discreet: -1
Support Vector Machines (Number of parameters = 16)									

C	*1.0*	6.1926	6.8039	15.3549	19.389	14.3604	16.7241	7.0052	continuous: [0.0, 20.0]
kernel	*rbf*	rbf	rbf	rbf	rbf	rbf	rbf	rbf	categorical: [linear, poly, rbf, sigmoid]
precompute d	-	-	-	-	-	-	-	-	-
degree	*3*	16	18	17	5	4	23	24	discreet: [1,30]
gamma	*scale*	scale	auto	scale	scale	scale	scale	scale	categorical: [scale, auto]
coef0	*0.0*	29.3523	26.9973	21.55	15.951	17.8193	23.9593	25.4899	continuous: [15.0, 30.0]
shrinking	*True*	False	True	False	False	False	True	False	boolean: [True, False]
probability	*False*	False	False	False	False	False	True	False	boolean: [True, False]
tol	*0.001*	0.7173	1.527	1.176	1.5451	1.3344	0.2997	0.5149	continuous: [0.0, 3.0]
cache_size	*200*	2000	2000	2000	2000	2000	2000	2000	discreet: 2000
class_weight	*None*	balanced	None	None	None	None	None	None	categorical: [None, 'balanced']
verbose	*False*	False	False	False	False	False	False	False	boolean: [True, False]
max_iter	*-1*	-1	-1	-1	-1	-1	-1	-1	discreet: -1

decision_function_shape	*ovr*	ovo	ovo	ovr	ovr	ovr	ovr	ovo	categorical: [ovo, ovr]
break_ties	*False*	-	-	-	-	-	-	-	-
random_state	*None*	42	42	42	42	42	42	42	discreet: 42
Multinomial Naive Bayes (Number of parameters = 3)									
alpha	*1.0*	1.4837	0.2123	0.2225	0.5744	0.9016	1.0476	0.2225	continuous: [0.5, 1.5]
fit_prior	*None*	-	-	-	-	-	-	-	-
class_prior	*True*	False	True	True	True	True	True	True	boolean: [True, False]
Logistic Regression (Number of parameters = 15)									
C	*1.0*	43.863	6.0898	15.5532	49.6688	18.4761	58.55	55.9101	continuous: [0.01, 100]
class_weight	*None*	balanced	balanced	balanced	balanced	balanced	balanced	balanced	category: balanced
dual	*False*	False	False	False	True	True	False	False	boolean: [True, False]
fit_intercept	*True*	True	True	True	True	True	True	False	boolean: [True, False]
intercept_scaling	*1*	1	1	1	1	1	1	1	discreet: 1
l1_ratio	*None*	None	None	1	None	1	None	None	continuous: [0, 1]
max_iter	*100*	743	1017	1235	1043	1399	169	646	discreet:

									[100, 2000]
multi_class	*auto*	auto	auto	auto	auto	auto	auto	auto	categorical: [auto, ovr, multinomial]
n_jobs	*None*	-1	-1	-1	-1	-1	-1	-1	discreet: -1
penalty	*l2*	none	l2	l1	l2	l2	l1	l2	categorical: [l1, l2, elasticnet, none]
random_state	*None*	42	42	42	42	42	42		discreet: 42
solver	*lbfgs*	newton-cg	newton-cg	liblinear	liblinear	liblinear	saga	newton-cg	categorical: [newton-cg, lbfgs, liblinear, sag, saga]
tol	*0.0001*	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	continuous: [0.00001, 0.0001]
verbose	*0*	0	0	0	0	0	0	0	discreet: 0
warm_start	*False*	False	True	True	False	True	True	True	boolean: [True, False]

Table 3: Parameters summary from both frameworks on classification task

As far as Table 3 is concerned we note the following:

- *n_jobs* was set to -1 to use all processors
- *random_state* was et to a random number (42) to not shuffle data and get different results each time
- verbose was set to 1 to show limited wordy information for the model
- *class_prior* was not specified so prior probabilities of classes would be adjusted according to the data
- *min_impurity_split* would be removed in newer version so it was not included at all
- *break_ties*: was not included since if it is set to true the decision_function_shape is equal to ovr and the number of classes is 2 and since we had a univariate result we excluded this
- *class_weight* was set to *balanced* since uses the values of y to automatically adjust weights inversely proportional to class frequencies in the input data as n_samples / (n_classes * np.bincount(y)) [17]
- *multi_class* was set to *multinomial* since in this case the multinomial loss is minimized even when data is binary
- *intercept_scaling* was set to the default value since it is useful only in certain cases
- *bootstrap* was set to *True* to use all of the training data to fit the model and not have a random variation between trees at each example

In the cases of Support Vector Machines, we observe that Wine, Banknote and Mnist, which have the greatest number of observations compared to the other datasets, all have similar *c* values, all of them range between 14 and 16. An exception to this is the Melanoma dataset. Also, the rbf kernel has been chosen to all the examples on the *kernel* parameter and *gamma* value is equal to scale in 6 out of 7 datasets so we can assume that they are not very affected by the cases that we examine.

On the experimentation with K Nearest Neighbors, one would expect the *number of neighbors* (i.e., the most important parameter of the KNN algorithm) suggested by the hyperparameter tuning to match the number of instances that each dataset contained. However, in the examples that were examined in this work, this definitely was not the case: the dataset with the most observations was assigned the lowest value of neighbors (1), whereas the number of most neighbors was assigned on the Titanic dataset, which, having 887 observations, would be probably considered medium-sized with respect to the size of the rest of the datasets that were included in this study. Another noteworthy observation is that the number of features in a dataset affects the selection of a *distance metric*: datasets with fewer features tend to be assigned with the option of the euclidean distance, whereas datasets with a greater number of features features are assigned with the Chebyshev or Minkowski distance (i.e., sklearn's default distance metric) (again though, with the exception of the Melanoma dataset).

On the Naive Bayes we also did not notice a big difference in the model's performance. Apparently, it either remained unaffected by the hyperparameter optimization, or was degraded with respect to its performance score. The number of parameters that were tuned were two out of a total of three that the model depends on. The difference was, again, insignificant, with many of the *alpha* values being around 1, which happens to be sklearn's default value, or close to 0, and the *class_prior* value being True almost in every case.

In Random Forest, one of the most important hyperparameters is the *number of estimators*, which is actually the number of trees in the forest. This appears to be highly correlated with the number of independent variables in the dataset which is something that we anticipated. Breast cancer, Wine Quality, and MNIST, all have a great number of features compared to the rest of the datasets. The value of the *estimator* parameter for each of them ranges between 100 and 144, while 150 is the max value of the search space for this particular hyperparameter. An exception to this is the Melanoma dataset. *Max features* is another parameter that is related to the number of features and the *number of estimators*, but in our cases there seems to be no pattern in this one.

In the example of Logistic Regression for the Titanic and Banknote datasets, liblinear is a good choice of a *solver*, but for the Wine Quality dataset it is not as good of a choice, since it is a multiclass classification problem and liblinear solver does not support multinomial loss. On the other hand, HyperOpt picked an optimal value for the *solver* parameter in the case of the MNIST dataset, since the saga algorithm works faster for large datasets.

Regression				
Parameters of algorithm	**Sklearn (default) [17]**	**HPO Boston House Pricing**	**HPO Adalone**	**Search Space**
Linear (Number of parameters = 4)				
copy_X	*True*	True	True	boolean : [True, False]
fit_intercept	*True*	True	True	boolean : [True, False]
n_jobs	*None*	-1	-1	discrete : -1
normalize	*False*	False	True	boolean : [True, False]
Lasso (Number of parameters = 11)				
alpha	*1.0*	0.0000	0.002	continuous: [0.0, 1.5]
copy_X	*True*	True	True	boolean: [True, False]
fit_intercept	*True*	True	True	boolean: [True, False]
max_iter	*1000*	123	2646	discrete: [100, 2000]
normalize	*False*	False	False	boolean: [True, False]
positive	*False*	False	False	boolean: [True, False]
precompute	*False*	False	True	boolean: [True, False]
random_state	*None*	42	42	discrete : 42
selection	*cyclic*	cyclic	cyclic	categorical: [cyclic, random]
tol	*0.0001*	0.0001	0.0025	continuous: [0.00001, 0.001]
warm_start	*False*	True	False	boolean: [True, False]

Ridge (Number of parameters = 8)				
alpha	*1.0*	0.0766	0.000	continuous: [0.0, 1.5]
copy_X	*True*	False	True	boolean: [True, False]
fit_intercept	*True*	True	True	boolean: [True, False]
max_iter	*None*	1803	113	discrete: [100, 2000]
normalize	*False*	True	False	boolean: [True, False]
random_state	*None*	42	42	discrete : 42
solver	*auto*	lsqr	lsqr	categorical: [auto, svd, cholesky, lsqr, sparse_cg, sag, saga]
tol	*0.001*	0.000	0.0009	continuous : [0.00001, 0.0001]

Table 4: Parameters summary from both frameworks on regression tasks

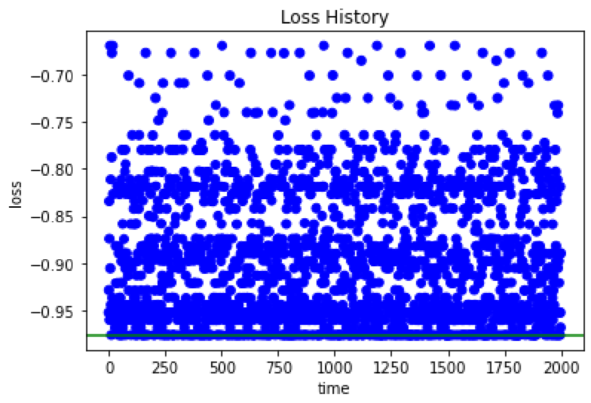
In the cases of the regression tasks, it is obvious that hyperparameter optimization did not improve significantly the performance of the models. The linear regression case has only four hyperparameters to tune. The rest of them, which incidentally also are the ones that contribute the heaviest in the final result of the model, are determined during the training process and not beforehand. Therefore, the small effect on the performance could have been foreseen. The set of parameters that was proposed by Hyperopt ended up being rather similar to the default values that sklearn uses, while the RMSE score was not affected at all. The same holds for the Abalone dataset, which had an RMSE of 2.57 in all three algorithms and two frameworks that was tested on. Since this dataset is quite large compared to the others, we could draw the conclusion that size does not affect the overall accuracy of the model. Another notable observation concerning the values of the parameters is that, in almost all cases, the alpha parameter is set to a value close to zero - if was equal to zero it would be equivalent to ordinary least squares method which is solved by Linear Regression on sklearn.

All these experiments constitute merely a subset of all the possible combinations of values that could be applied on the algorithms' hyperparameters. It is likely that a different set could produce the same result on a specific dataset combined with an algorithm. This is made rather obvious even by a quick look in the above table, where the groups of parameters of this study are displayed: we can observe that in many cases both frameworks produce results with the exact same accuracy score. The random_state variable was used in situations where a random sample was chosen in the process, with the purpose of reproducibility and also to have a more stable and unified result.

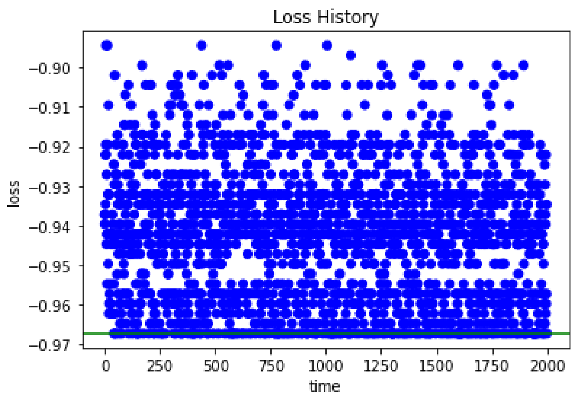
6.5. Trials

Hyperopt provides its own visualization module. Through this module it is easy to inspect the results of Hyperopt's trials object. In the figures below (where each figure corresponds to a different case examined), each plotted point represents the best score achieved during the particular iteration.

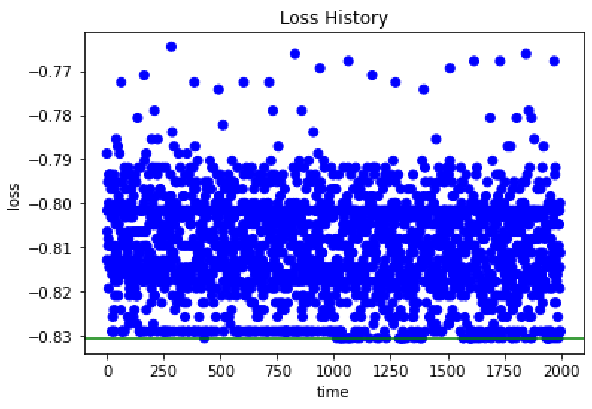
K Nearest Neighbors:



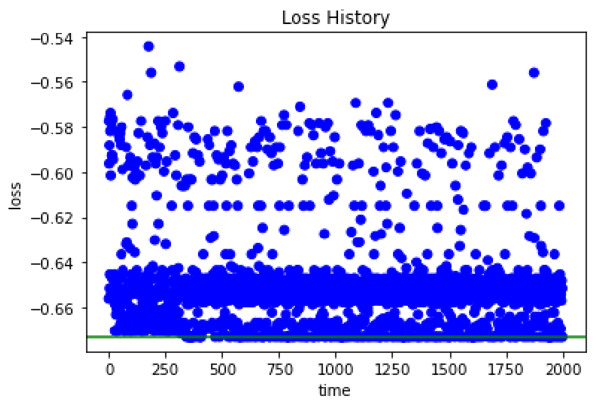
Iris



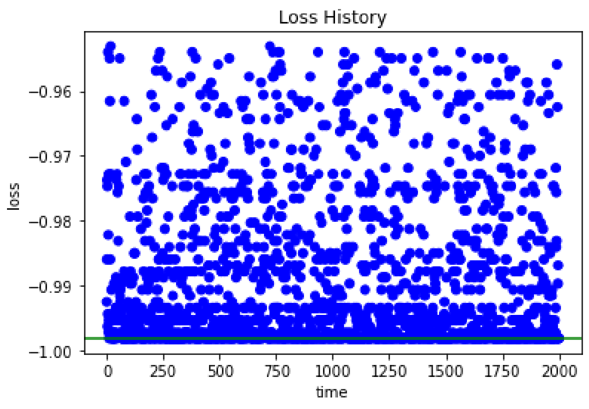
Breast Cancer



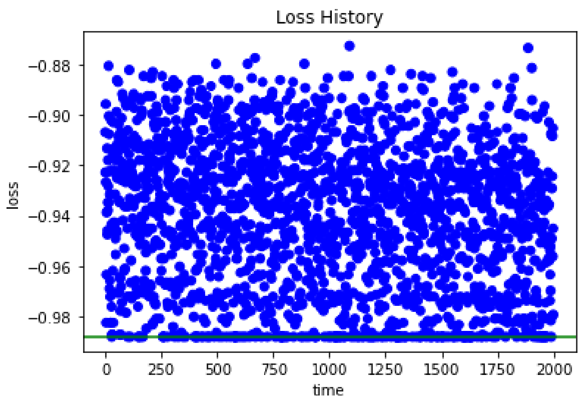
Titanic



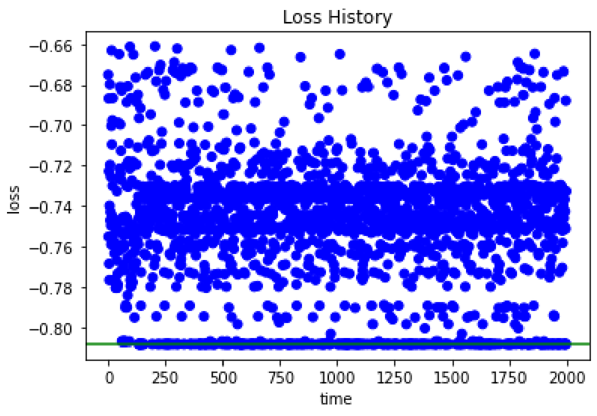
Wine Quality



Banknote Authentication

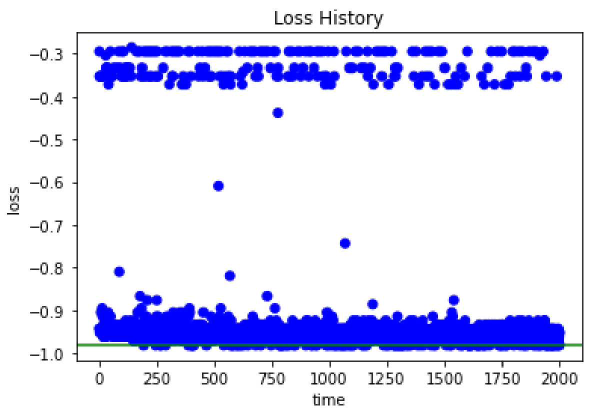


MNIST

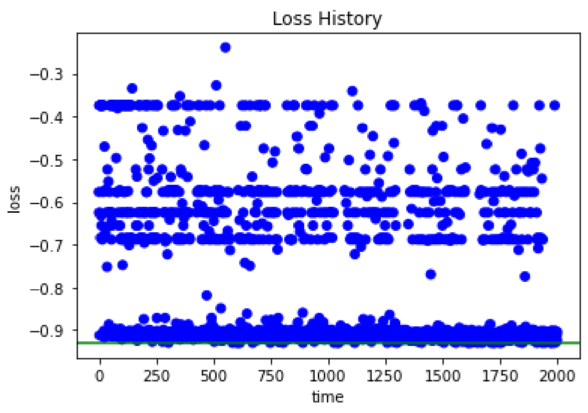


Melanoma

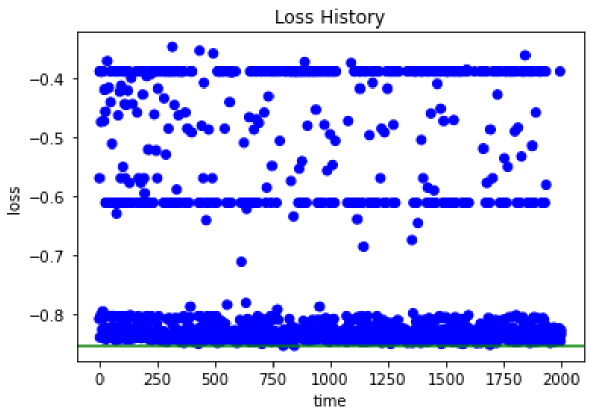
Support Vector Machines:



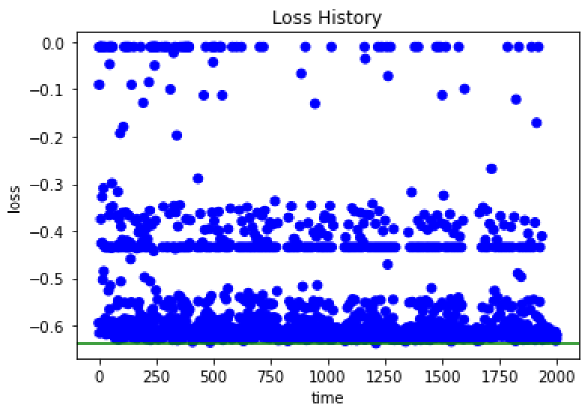
Iris



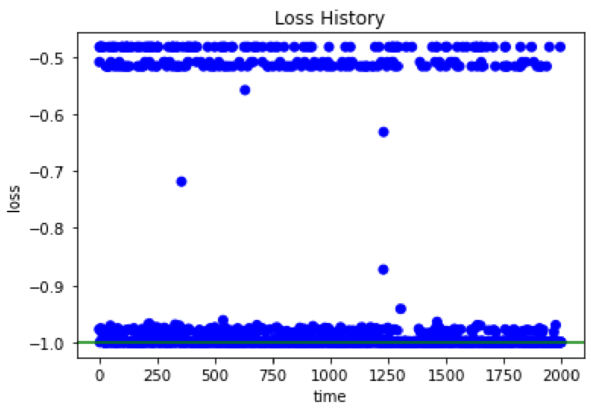
Breast Cancer



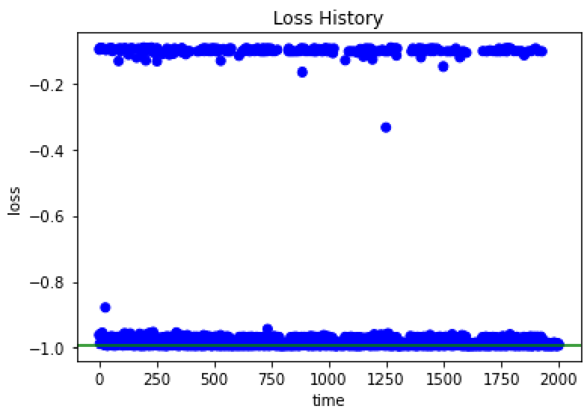
Titanic



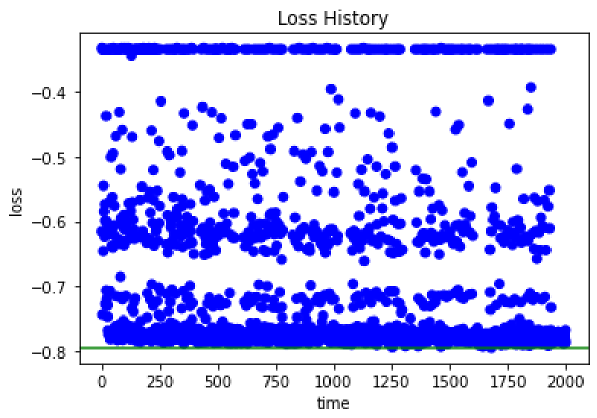
Wine Quality



Banknote Authentication

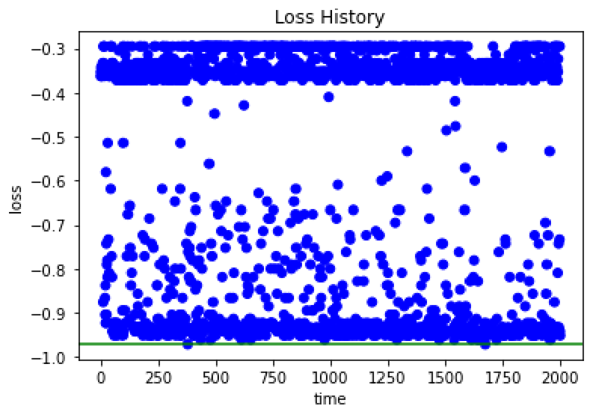


MNIST

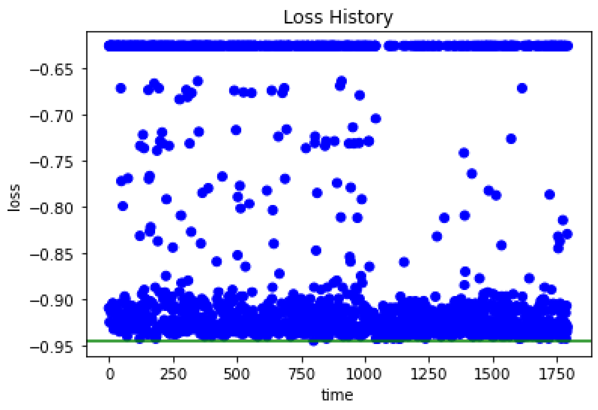


Melanoma

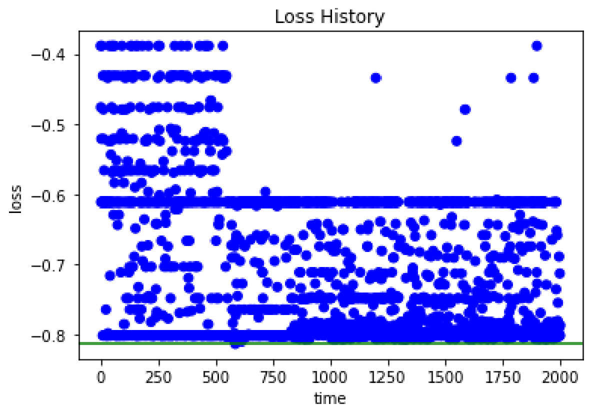
Random Forest:



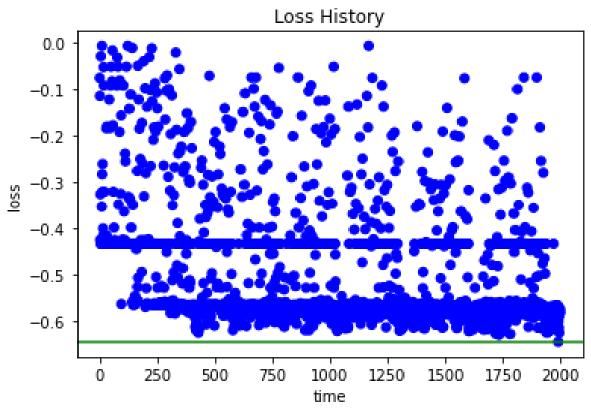
Iris



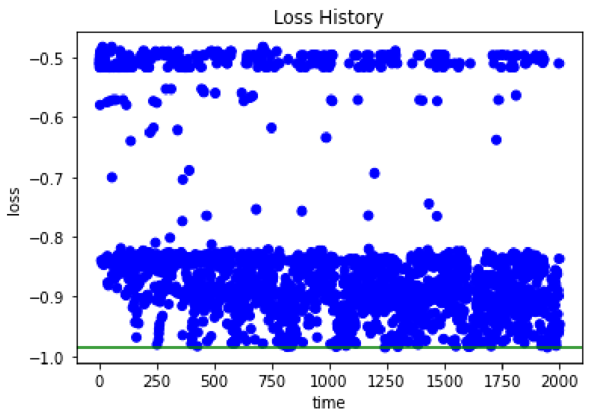
Breast Cancer



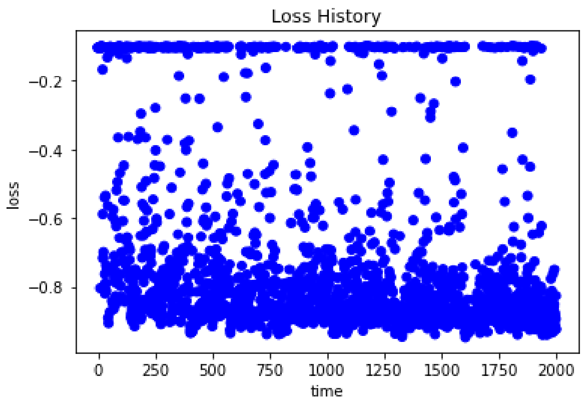
Titanic



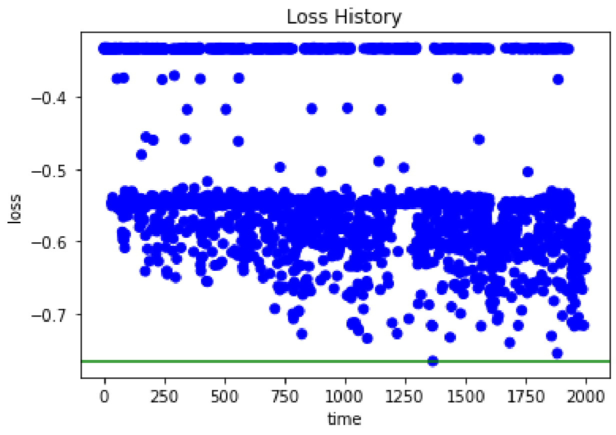
Wine Quality



Banknote Authentication

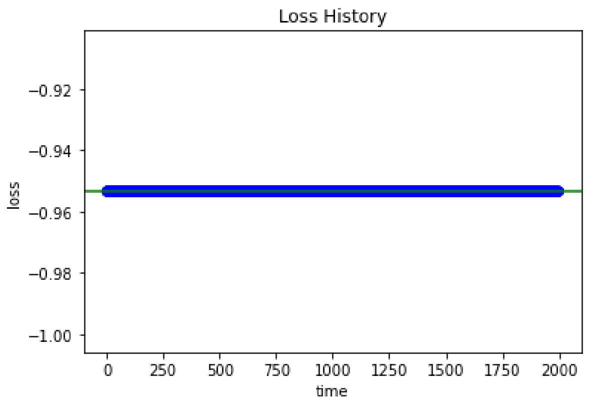


MNIST

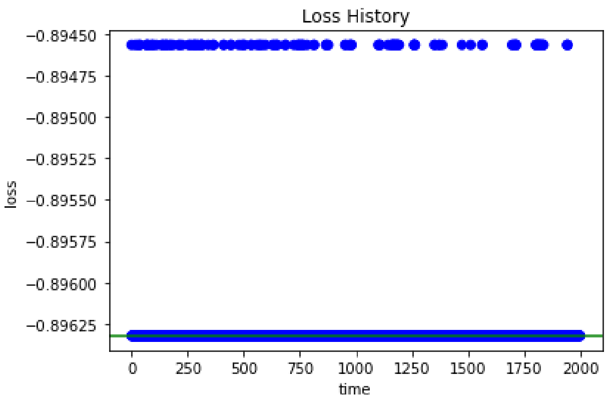


Melanoma

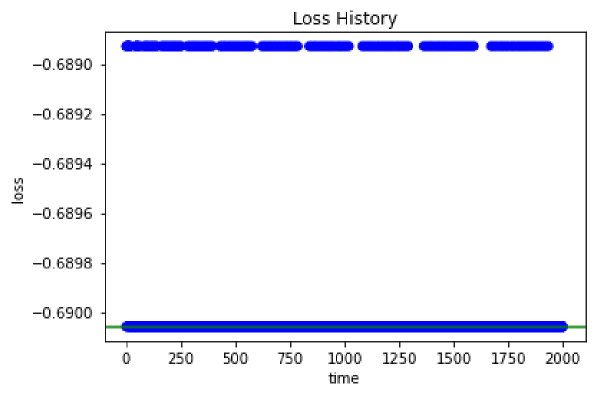
Naive Bayes:



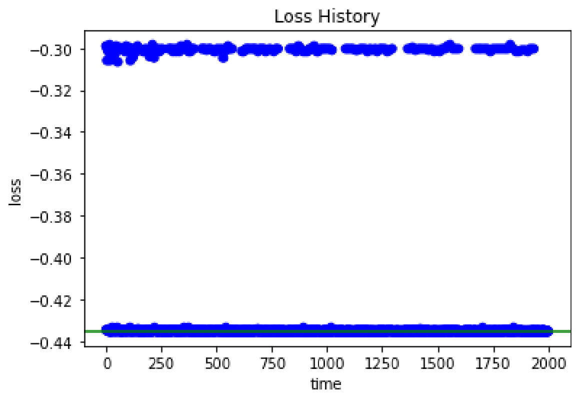
Iris



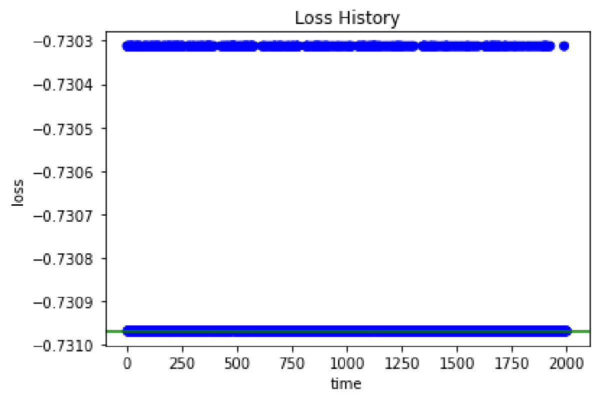
Breast Cancer



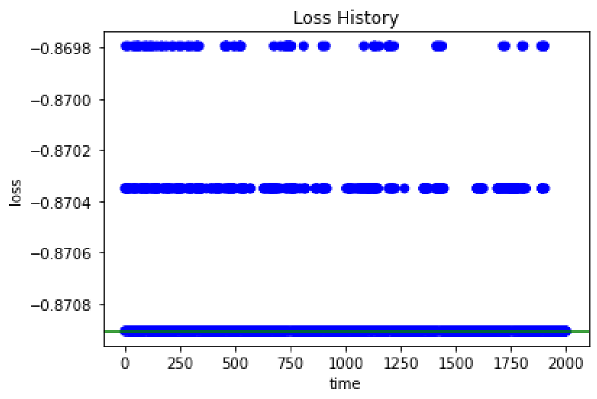
Titanic



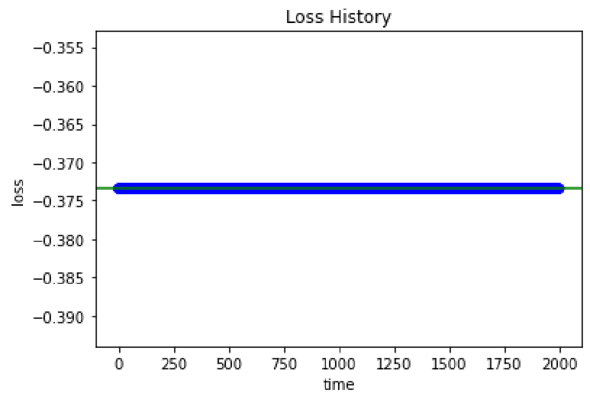
Wine Quality



Banknote Authentication

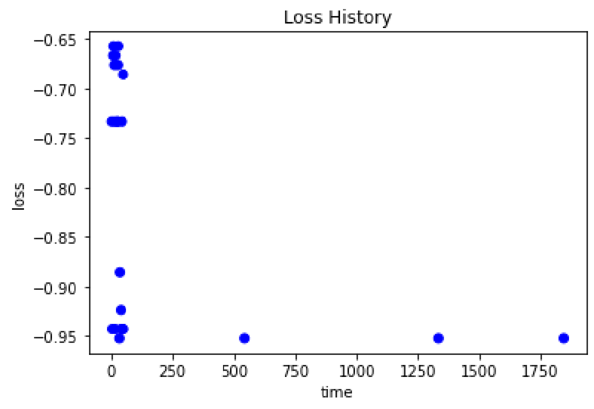


MNIST

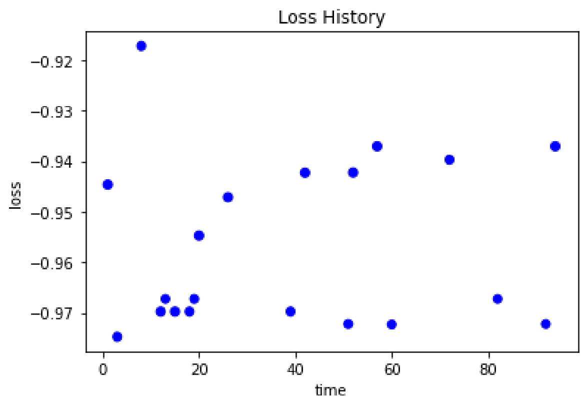


Melanoma

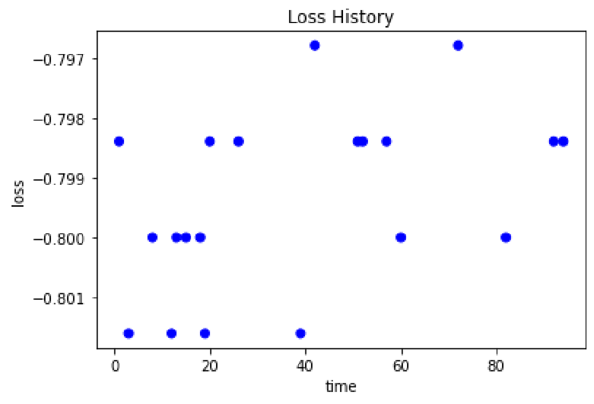
Logistic Regression:



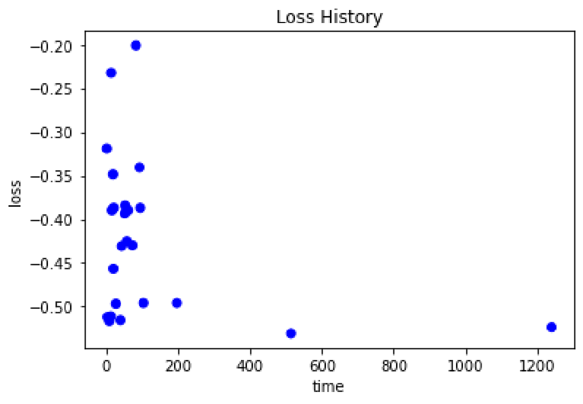
Iris



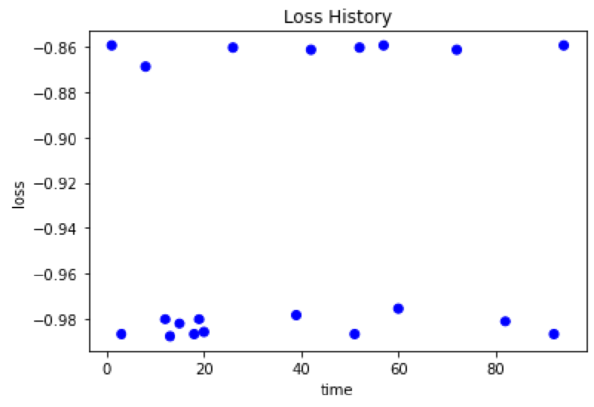
Breast Cancer



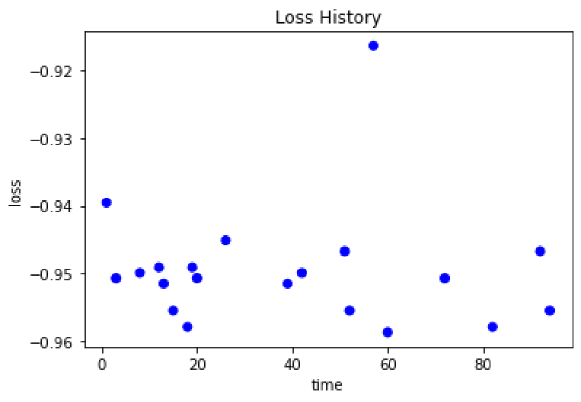
Titanic



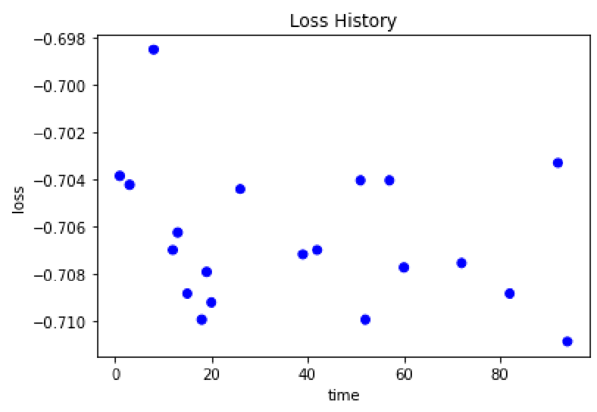
Wine Quality



Banknote Authentication

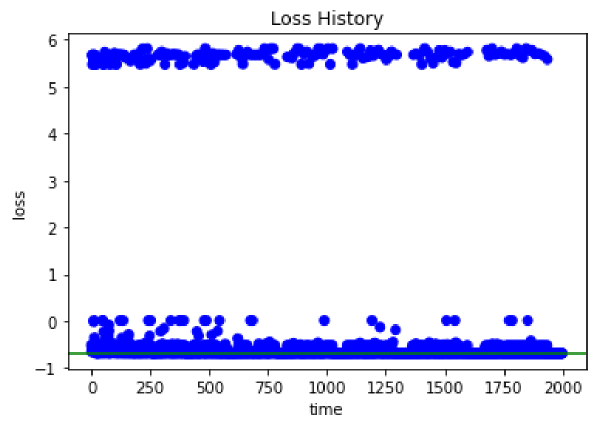


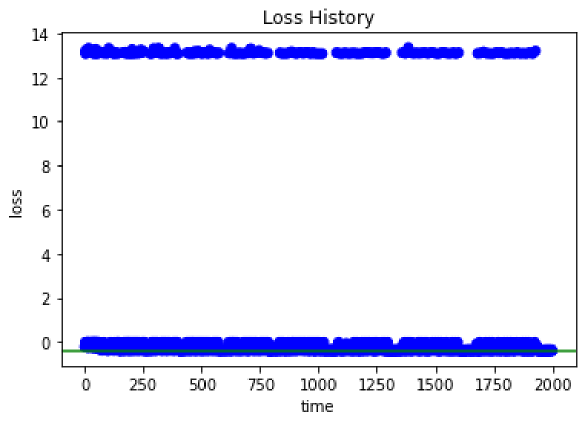
MNIST



Melanoma

Lasso:

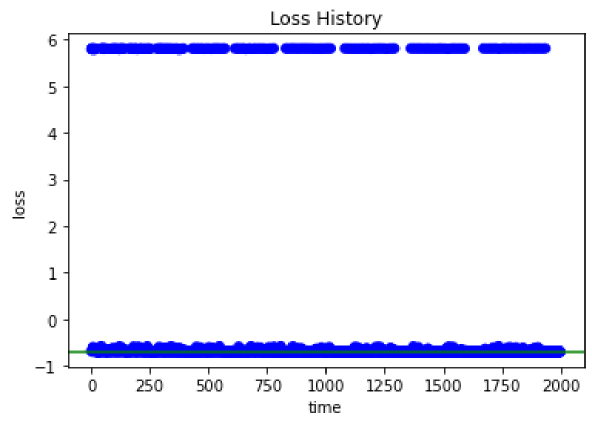


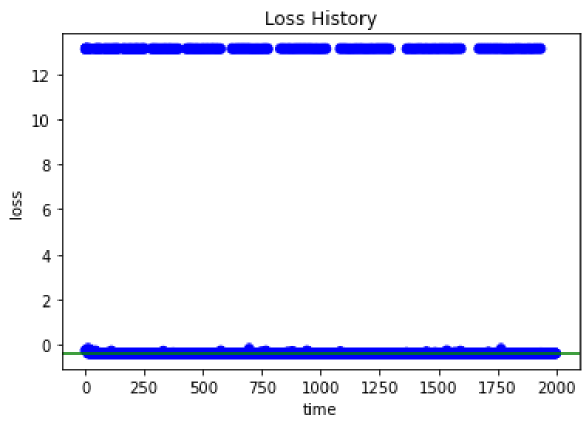


Boston House

Adalone

Ridge:





Boston House

Adalone

6.6. Setup

All experiments were conducted using Python 3.5 on a machine with a quad-core Intel i7-8550U CPU and 16 GiB of memory. The Machine Learning and Hyperparameter Optimization methods are implemented and evaluated using open-source Python libraries and frameworks including scikit-learn (version 0.23.2), hyperopt (version 0.2.5) and also pandas (version 1.1.3) and numpy (version 1.19.2) for some basic preprocessing before the models were applied.

7. Conclusions and Next Steps

In this thesis we explored how hyperparameter optimization works on supervised learning models; more specifically on regression and classification algorithms.

The conclusions that can be drawn from the evaluation results in our study lead to doubts on hyperparameter optimization as a practice that should occur in any and all cases of development of Machine Learning models.There are some factors that appear to affect the whole process. The same set of factors is what should help us decide whether performing hyperparameter optimization worths its trade-offs or not.

An important factor is the performance of the model. In the cases where there was a significant increase in the accuracy of the model, hyperparameter optimization is definitely something worth trying. However, in cases where the accuracy remained the same, or even decreased, it would probably be best to apply the model with the default parameter values.

Another factor that plays a key role in the whole process is the execution time. A noteworthy example is the Random Forest classifier experiment and its *number of estimators* hyperparameter. In this case, time was affected significantly every time the search space for this parameter was widened, albeit the corresponding accuracy did not show any satisfying results. Apart from this case, Support Vector Machines and Logistic Regression showed positive results with respect to the performance, with a relatively small amount of execution time, thereby indicating that hyperparameter optimization would worth to be applied in such cases. Furthermore, we should dedicate a moment to comment on the results of the experiments related to the Melanoma dataset: they all performed quite well in comparison with the rest of the datasets, thus contributing to the conclusion that a large number of independent variables is worth the time it takes for the hyperparameter tuning.

There is a lot of room for work to be conducted following this study. A first step would be the inclusion of more learning models (e.g., clustering, time series, reinforcement learning, etc.) along with their respective algorithms and their associated hyperparameters (i.e., XGBoost, Neural Networks, etc.) into the exploration space. Another direction for future work would be the inclusion of a greater number of additional datasets in the study. New datasets, with either similar or diverse features, could, not only validate the results of this study, but also lead to more robust evaluation results, broader conclusions and expanded sets of use cases where hyperparameter optimization can indeed make a difference.

References

[1] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kegl. (2011) 'Algorithms for hyper-parameter optimization', NIPS, 24:2546–2554.

[2] J. Bergstra, D. Yamins, and D. D. Cox. (2013) 'Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures', In Proc. ICML.

[3] J. Bergstra, D. Yamins, and D. D. Cox. (2013) 'Hyperopt: A Python library for optimizing the hyperparameters of machine learning algorithms', SciPy'13.

[4] B. Komer, J. Bergstra, and C. Eliasmith. (2014) 'Hyperopt-sklearn: automatic hyperparameter configuration for scikit-learn', ICML AutoML Workshop.

[5] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. (2011) 'Scikit-learn: Machine Learning in Python', Journal of Machine Learning Research, 12:2825–2830.

[6] Komer B., Bergstra J., Eliasmith C. (2019) 'Automated Machine Learning. The Springer Series on Challenges in Machine Learning'. Springer

[7] L. Yang and A. Shami (2020), 'On hyperparameter optimization of machine learning algorithms: Theory and practice', Neurocomputing, vol. 415, pp. 295–316.

[8] Fisher R. A. (1936). UCI Machine Learning Repository [https://archive.ics.uci.edu/ml/datasets/iris].

[9] Dr. William H. Wolberg, W. Nick Street and Olvi L. Mangasarian (1995). UCI Machine Learning Repository [https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+(Diagnostic)]. WI: University of Wisconsin, General Surgery Dept. and Computer Sciences Dept..

[10] Frank E. Harrell Jr., Thomas Cason. (2017). Titanic - Machine Learning from Disaster. Retrieved in 2020 from https://www.kaggle.com/c/titanic .

[11] P. Cortez, A. Cerdeira, F. Almeida, T. Matos and J. Reis. (2009) 'Modeling wine preferences by data mining from physicochemical properties'. In Decision Support Systems, Elsevier, 47(4):547-553.

[12] Lohweg V. and Doerksen H. (2012). UCI Machine Learning Repository [https://archive.ics.uci.edu/ml/datasets/banknote+authentication]. University of Applied Sciences, Ostwestfalen-Lippe.

[13] LeCun, Y and Cortes, C (2010) 'MNIST handwritten digit database'. Available at http://yann.lecun.com/exdb/mnist/ .

[14] Harrison, D. and Rubinfeld, D.L. (1978) `Hedonic prices and the demand for clean air', J. Environ. Economics & Management, vol.5, 81-102.

[15] Warwick J Nash, Tracy L Sellers, Simon R Talbot, Andrew J Cawthorn and Wes B Ford (1994) 'The Population Biology of Abalone (_Haliotis_ species) in Tasmania. I. Blacklip Abalone (_H. rubra_) from the North Coast and Islands of Bass Strait', Sea Fisheries Division, Technical Report No. 48 (ISSN 1034-3288).

[16] Agrawal T. (2021) 'Hyperparameter Optimization in Machine Learning'. Apress, Berkeley.

[17] https://scikit-learn.org/

[18] Tanay Agrawal (2020) 'Hyperparameter Optimization in Machine Learning: Make Your Machine Learning and Deep Learning Models More Efficient'

[19] D.W. Hosmer Jr, S. Lemeshow (2013) 'Applied logistic regression', Technomet- rics

[20] Bunheang Tay, Jung Keun Hyun, and Sejong Oh (2013) ‘A Machine Learning Approach for Specification of Spinal Cord Injuries Using Fractional Anisotropy Values Obtained from Diffusion Tensor Images’, Department of Nanobiomedical Science, Dankook University

[21] Naphaporn Sirikulviriya, Sukree Sinthupinyo (2011) Integration of Rules from a Random Forest’, Department of Computer Engineering, Chulalongkorn University, Bangkok, Thailand

[22] D. Lowd, P. Domingos (2005) ‘Naïve Bayes Models for Probability Estimation’

[23] Chungsoo Lim,, Seong-Ro Lee, Joon-Hyuk Chang (2013) ‘Efficient Implementation of an SVM-Based Speech/Music Classifier by Enhancing Temporal Locality in Support Vector References’

[24] Angel, Luis & Viola, Jairo & Vega, Mauro & Restrepo, R.. (2016). ‘Sterilization process stages estimation for an autoclave using logistic regression models’.

[25] Breiman, L. (2001) ‘Random Forests’. Machine Learning 45, 5–32

[26] Arthur E. Hoerl, Robert W. Kennard (1970) ‘Ridge Regression: Biased Estimation for Nonorthogonal Problems’, American Statistical Association and American Society for Quality.

[27] Zou, H., & Hastie, T. (2005). ‘Regularization and Variable Selection via the Elastic Net’. Journal of the Royal Statistical Society.

[28] Tibshirani, R. (1996) ‘Regression Shrinkage and Selection via the Lasso’. Journal of the Royal Statistical Society.

[29] Cui, Zaixu; Gong, Gaolang (2018) ‘The effect of machine learning regression algorithms and sample size on individualized behavioral prediction with functional connectivity features’

[30] Hui Zou, Trevor Hastie (2005) ‘Regularization and variable selection via the elastic net’

[31] James Bergstra, Yoshua Bengio (2012) ‘Random Search for Hyper-Parameter Optimization’

[32] A.M. Kibriya, E. Frank, B. Pfahringer, G. Holmes (2004) ‘Multinomial naive bayes for text categorization revisited’, Lect. Notes Artif. Intell. (Sub-series Lect. Notes Comput. Sci. 3339 (2004) 488499.