IOSR Journal of Computer Engineering (IOSR-JCE)
e-ISSN: 2278-0661,p-ISSN: 2278-8727, Volume 23, Issue 4, Ser. I (Jul. –Aug. 2021), PP 34-50
www.iosrjournals.org

Hyperparameter Optimization on Classification and Regression Algorithms

Georgia Lytra[1], Argyro Mavrogiorgou[1], Athanasios Kiourtis[1], Dimosthenis Kyriazis[1]
[1](*Department of Digital Systems, University of Piraeus, Greece)*

Abstract:
Background*: Machine learning has taken the technological world by storm in recent years. Every expert that needs to cope with a specific problem, demands to develop a model that will be able to cover all the problem's needs in combination with all the available resources that come by. Along the way of this process, a lot of challenges may come up. One of these challenges refers to the selection of the most appropriate hyperparameters that should be used for the construction of the most efficient model. This process, called hyperparameter optimization, is considered to be extremely vital. Even though the created models are proved to be effective in both performance and execution time, the same models can be rendered rather than useless without the appropriate hyperparameters' selection. It is a fact that hyperparameter optimization can really help a model to shine and exploit its capabilities to the fullest. However, since every problem has its uniqueness and complexity, domain knowledge is necessary for choosing the appropriate hyperparameters in each different case. Hence, the need of implementing methodologies that automatically solve this issue is on the rise.*
Materials and Methods*: This study tries to fill in this gap, by following an experimental procedure to extract information regarding the appropriate hyperparameters on various supervised (classification and regression) learning models. Various datasets with diverse features and characteristics are exploited, which could assist the successful automation of machine learning processes, whereas already existing optimization frameworks are fully utilized.*
Results*: The conducted study resulted into the extraction of both reliable and generalized results that cover a variety of diverse machine learning problems deriving from various sectors.*
Conclusion: *The study led to doubts on hyperparameter optimization as a practice that should occur in all cases of development of machine learning models. There are some factors that appear to affect the whole process, whereas the same set of factors is what should help the user decide whether performing hyperparameter optimization worths its trade-offs or not.*
Key Word*: Hyperparameters, Hyperparameter optimization, Hyperparameter tuning, Supervised learning, Classification, Regression, Machine learning, HyperOpt, Scikit-learn.*

Date of Submission: 10-07-2021 Date of Acceptance: 26-07-2021

I. Introduction

In a 2017 article of the Economist [1], data was compared with what oil was in the 18th century and was characterized as the most valuable resource of the 21st century. 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 [2]. Indeed, we live in an era where data is produced in tremendous amounts every day, from social media platforms to Internet of Things (IoT) sensors to applications that assist each of us in our everyday lives. This 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 (AI) comes along with Machine Learning (ML) practices to make an impact and change the way things work in various fields [3].

Machine Learning is involved in many applications where data is present. Most of the times, 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 models, usually there is an optimal solution to it, indicating the importance of the selection of the most suitable algorithm in a ML process [4]. Another key role is occupied by the algorithms' hyperparameters, which are also affected by the different scenarios and the data that accompany them [5]. Thus, both the selection of the ML algorithm and the selection of the algorithms' hyperparameters have to be taken carefully into con-sideration.

In this manuscript, we attempt to extract information from various experiments regarding the hyperparameters of several ML models, concentrating mainly into the supervised ones. To be more specific, to

achieve that, we have studied various classification and regression models and conducted several experiments to spot any differences or similarities in the way that their hyperparameters affect their developed models. The procedure that we followed involves a very popular hyperparameter tuning framework, HyperOpt [6]. The latter uses bayesian reasoning to construct an algorithm that is called Tree of Parzen Estimators, which has proven to be very effective on the tasks of hyperparameter optimization, especially compared with other hyperparameter search algorithms [7]. Into this context, the conducted experiments were a combination of widely used classification and regression algorithms, along with a variety of popular datasets, each one of them belonging to a different sector. This allowed us on the one hand not to 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 both reliable and generalized results that cover a variety of diverse problems deriving from various sectors. On top of these, in order to evaluate the results produced by the HyperOpt framework, these were compared with the corresponding results produced by the sklearn framework; HyperOpt was constructing the algorithms' models based upon their hyperparameters, whereas sklearn was constructing the algorithms' models based upon their default parameters, thus showcasing their individual differentiations.

The rest of the manuscript is organized as follows. Section II provides all the necessary background that the reader must know in order to understand the conducted experimental study. Section III includes all the datasets that are used in the experimental study along with a description of the respective features of each dataset, followed by a detailed explanation of the experimental phase and its components. Section IV discusses all the captured results, whilst Section V provides some conclusion based on the conducted work along with some future steps.

II. Material And Methods

The aim of this work is to investigate the performance of the hyperparameter optimization procedure, based on various datasets that are produced by diverse sectors. The flow of the conducted experimental study is illustrated in Figure no 1, consisting of 2 sequential phases. The first phase includes the initial implementation of the chosen ML algorithm with its default parameters upon the chosen dataset, whereas the second phase includes the iterative implementation of the chosen ML algorithm with its extracted hyperparameters upon the chosen dataset. In short, in the first phase the preprocessing of the data takes place, followed by the split of the input dataset into the train and the test sets. In sequel, the chosen algorithm (accompanied with its default parameters) is applied upon the train set to build its corresponding model, which is then evaluated based upon the test set. Based on the captured results, the produced model is then evaluated. Sequentially, the hyperparameter optimization (i.e., tuning) occurs, exploiting already existing frameworks in order to extract the best set of hyperparameters of the ML algorithm, which eventually will produce the most efficient ML model. Hence, since two (2) different models are produced, one from the first phase and one from the second phase, those models are compared to each other, always verifying the better performance of the model produced after the execution of the hyperparameter optimization, as it is demonstrated in the Results section.

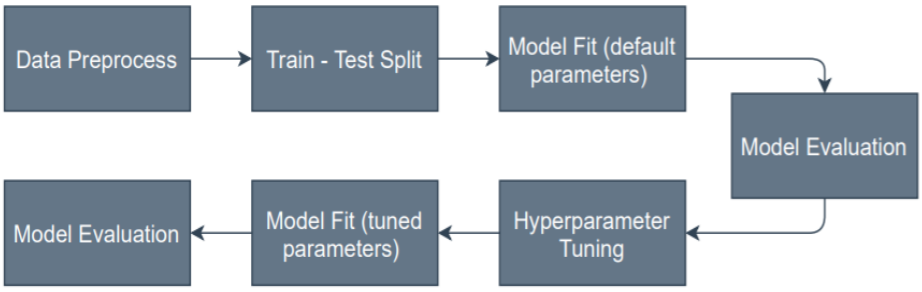


Figure no 1. Experimental study applied procedure.

Data Preprocessing

In the first step of the experimental study the data preprocessing takes place right before the separation of each dataset into train and test sets, in the context of either classification or regression. Data preparation is a critical step before training any ML model and therefore the techniques that are used have to be carefully chosen. It can seriously impact both the performance of the models and the effectiveness of the attempts to avoid the so very dreadful overfitting. All the datasets that are exploited in this work undergo the same preparation techniques to ensure as much as possible unified results, implementing just a few required exceptions, as it will be depicted in the Results section. Especially with regards to the datasets that are used for classification, a big issue is the balance of the classes. Unbalanced classes can very easily lead to overfitting of the model. The solution to this problem is either to upsample the cases of the lesser class or to undersample the cases of the majority class. In the context of this study, the latter is implemented in order to generate 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 is to split the data into two (2) subsets, the training dataset and the test dataset. To this context, in all the cases the ratio that was applied was a split of 70/30 correspondingly. The rationale behind this is that on the one hand to fit the produced model on the training data and thus having a different portion of the original dataset so as to make the appropriate predictions, and on the other hand to evaluate the performance of the produced model with the appropriate metrics that fit the classification and regression tasks respectively.

Model Selection

In sequel, irrespective of the ML problem that has to be solved, a variety of supervised models are applied upon the chosen dataset. Generally speaking, 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 [8]. It includes algorithms for performing either classification or regression analyses and predictions, such as Linear Regression for regression problems, Random Forest for classification and regression problems, or Support Vector Machines for both classification and regression problems. To the context of the current experimental study, for the classification problems the classifiers of Random Forest, Support Vector Machine, K Nearest Neighbor, Logistic Regression, Multinomial Naive Bayes are exploited, whereas for the regression problems the regressors of Ridge and Lasso are analyzed.

Random Forest

Random Forest (RF) is an ensemble classifier. It represents a set of many classifiers, in this case many binary decision trees, in order to combine the decision of each classifier with the scope to classify new examples. According to [9], assuming a training set drawn randomly for the distribution of the random vector Y, X and given an ensemble of classifiers $h_1(x)$, $h_2(x)$, … , $h_K(x)$, 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:

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

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. To this context, the definition for the generalization error is defined as:

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

where the subscripts Y, X indicate that the probability is over the Y, X space. The pseudocode for the RF model is presented in Figure no 2 [10].

```
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
```

Figure no 2. RF model pseudocode.

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 [11]:

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

where w is a normalization vector and C is the penalty parameter of the error term, which is an important hyperparameter of all SVM models. The pseudocode for the SVM model is illustrated in Figure no 3 [11].

```
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 ← 1 to N_in by 1 do
    F = 0
    for j ← 1 to N_sv by 1 do
        dist = 0
        for k ← 1 to N_ft by 1 do
            dist += (SV[j].feature[k] - IN[i].feature[k])^2
        end
        κ = exp(-γ × dist)
        F += SV[j].α* × κ
    end
    F = F + b*
end
```

Figure no 3. SVM model pseudocode.

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, $i = (1,2,\dots,n)$, 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 KNN of x [12]. The step required to implement the KNN algorithm is outlined in the pseudocode in Figure no 4 [13].

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

Figure no 4. KNN model pseudocode.

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 [14]:

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

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 Figure no 5 [15].

Input:
- *Training algorithm L (logistic regression)*
- *Sample matrix X*
- *Labels vector* $y = [1, \ldots K]$
- *Initial regresor parameters vector* θ_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 θ_i

Output:

θ_i *Parameters vector for each regresor*

Figure no 5. LR model pseudocode.

Multinomial Naive Bayes

Multinomial Naive Bayes (MNB) is designed for multinomially distributed data based on the Naive Bayes algorithm [16]. As explained in [12], 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} $(i = 0,1,2,\ldots,n)$ 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 $a \leqslant 1$, it is called Lidstone smoothing [17]. The pseudocode of the model is shown in Figure no 6 [18].

Input: training set T, hold-out set H, initial number of components k_0, and convergence thresholds δ_{EM} and δ_{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 δ_{EM} over the last iteration.
  $k \leftarrow 2 \times k$
until $\log P(H|M)$ fails to improve by ratio δ_{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}.

Figure no 6. MNB model pseudocode.

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 [19], [20]:

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

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 [21].

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 [22], [20]. The objective function takes the form as below:

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

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 [23].

Hyperparameter Optimization

As soon as the ML model is constructed, the hyperparameter optimization occurs. Generally speaking, 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 SVM or k in KNN), 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 LR or support vectors in SVM). The process of selecting the values for these hyperparameters is called hyperparameter tuning or hyperparameter optimization [24]. This process results in the selection of the optimum hyperparameters for a ML algorithm, given a set of data. Its practice is crucial before any application of the ML 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 [25]. The following are the most used ones.

Manual Search

Manual search is the act of manually selecting a model's 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 the manual testing of various sets of parameters until the best one is found. An essential factor is that the problem under investigation should not be very complex for the manual selection of hyperparameters to be feasible.

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.

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.

Bayesian Optimization

Bayesian Optimization, also known as 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 [26]. 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.

To the context of the current experimental study, to achieve hyperparameter optimization, two (2) diverse already existing frameworks are exploited, namely Scikit Learn and the HyperOpt.

Scikit Learn, or sklearn, is a Python library for ML and statistical modeling including classification, regression and clustering. This package focuses on making easier the use of ML 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 [27].

HyperOpt is an open-source Python library for hyperparameter search. One of its core characteristics is that it uses a form of SMBO, the Tree of Parzen Estimators (TPE) [28]. 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 $f: X \to \Re^N$that $f(x)$ will exceed (negatively) some threshold y [7]:

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

The basic idea of TPE is that, unlike what a Gaussian 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 (2) densities [7]:

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

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

There are four (4) parts that we need to focus on when using HyperOpt [6]:

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

Based on the abovementioned, HyperOpt works as follows:

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

III. Result

Datasets Description

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 diverse ML techniques for drug discovery, clinical trial research or epidemic outbreak prediction. Other applications involve computer vision or time series forecasting, which are utilized by many companies that offer their services to the public. To perform a wide performance evaluation of the experimental study, and successfully achieve the concept of hyperparameters optimization, a variety of datasets from different sectors and fields have been chosen in order both to examine different cases, and to include datasets with diverse features. Each of them is quite popular and therefore did not require extensive processing. This fact fitted along with the current work, since the main purpose of the study was not to emphasize in difficult preprocessing techniques but to focus on the results that come after. Table no 1 summarizes the datasets used in this study, where for each dataset, 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.

Table no 1: Summary of used datasets.

Dataset	Columns	Rows	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

The Iris Dataset involves predicting the flower species, based on given measurements of iris flowers [29]. The rows of the table represent an iris flower, including its species and dimensions of its botanical parts, sepal and petal, in centimeters. More specifically it includes information for sepal length in cm, sepal width in cm, petal length in cm, petal width in cm, and the class (Iris Setosa, Iris Versicolour, Iris Virginica).

The Breast Cancer Winsconsin dataset is a well-known dataset for breast cancer diagnosis systems [30]. 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 an image, including information for the ID number, Diagnosis (M = malignant, B = benign), radius (mean of distances from center to points on the perimeter), texture (standard deviation of gray-scale values), perimeter, area, smoothness (local variation in radius lengths), compactness (perimeter^2 / area - 1.0), concavity (severity of concave portions of the contour), concave points (number of concave portions of the contour), symmetry, fractal dimension ("coastline approximation" - 1) of cell nucleus.

The Titanic dataset contains data for the real Titanic passengers [31], where each row represents one passenger. 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.

The Wine Quality dataset involves predicting the quality of wines on a scale given chemical measures of each wine [32], including information for the wines' fixed acidity, volatile acidity, citric acid, residual sugar, chlorides, free sulfur dioxide, total sulfur dioxide, density, pH, sulphates, alcohol and quality (score between 0 and 10).

The Banknote Authentication dataset involves predicting whether a given banknote is authentic given a number of measures taken from a photograph [33]. Data was 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. In more detail, the dataset includes information for the variance of Wavelet Transformed image, the skewness of Wavelet Transformed image, the kurtosis of Wavelet Transformed image, the entropy of image, and the class (0 for authentic, 1 for inauthentic).

The MNIST dataset (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 [34]. 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.

The Melanoma 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 Class 1 for Melanoma, Class 2 for Dysplastic Nevus, and Class 3 for Non Dysplastic Nevus.

The Boston House Pricing 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 [35]. More specifically, the dataset contains information for CRIM (Per capita crime rate by town), ZN (Proportion of residential land zoned for lots over 25,000 sq. ft), INDUS (Proportion of non-retail business across per town), CHAS (Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)), NOX (Nitric oxide concentration (parts per 10 million)), RM (Average number of rooms per dwelling), AGE (Proportion of owner-occupied units built prior to 1940), DIS (Weighted distances to five Boston employment centers), RAD (Index of accessibility to radial highways), TAX (Full-value property tax rate per $10,000), PTRATIO (Pupil-teacher ratio by town), B (1000 (Bk - 0.63)², where Bk is the proportion of [people of African American descent] by town), LSTAT (Percentage of lower status of the population), MEDV (Median value of owner-occupied homes in $1000s).

The Abalone dataset's objective is predicting the age of the abalone from physical measurements [36]. 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. In deeper detail, the dataset contains information for Sex (M, F, and I (infant)), Length (Longest shell measurement), Diameter (perpendicular to length), Height (with meat in shell), Whole weight (whole abalone), Shucked weight (weight of meat), Viscera weight (gut weight (after bleeding)), Shell weight (after being dried), Rings (+1.5 gives the age in years).

Environment Setup

All the experiments were conducted using Python 3.5 [37] 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) [38], HyperOpt (version 0.2.5) [39] and also pandas (version 1.1.3) [40] and numpy (version 1.19.2) [41] for some basic preprocessing before the models were applied.

Experimental Results

The process of the experimental phase that 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 mentioned in the Material And Methods section, was applied in the chosen data, thus covering diverse combinations. At this stage, the default values for the parameters that sklearn assigns in every model were applied. Since the goal is to compare the hyperparameters optimization results between both frameworks mentioned in the Material And Methods section (i.e. sklearn, HyperOpt), we move on to evaluate the performance of the diverse models on these values with the respective metrics. Afterwards, the proposed hyperparameters are fitted in the same model that was fitted previously and the evaluation step takes place so as to produce the final results and compare them between the two (2) frameworks.

Data Preprocessing

To begin with, during the preprocessing phase, as mentioned in the Material And Methods section, a big issue in the classification tasks is the balance of the classes, where unbalanced classes can lead to overfitting of the model. To address this, in the experiments the undersample of the majority class took place in order 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. Moreover, since these datasets 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.

The next step was to split the data into the training dataset and the test dataset, having a split ratio of 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 test datasets 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 used in the rest of the cases, we applied the aforementioned.

Sequentially, with the aim to avoid data leakage, scaling of the data is implemented after the split strategy. The data is scaled on a range of 0 to 1 and this proved to be useful, not only for dodging the overfitting but also for improving the 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 HyperOpt. In many cases, it run the process much faster, whereas before it was executed in a significant amount of time.

Afterwards, the implementation of HyperOpt and sklearn took place. 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.

Metrics

Regarding the metric functions, three (3) 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 score, 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. 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.

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 no 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.

Table no 2: Accuracy of each experiment in both classification and regression models.

Dataset	Sklearn	HyperOpt	Execution Time (HPO)
Classification (Metric: Accuracy)			
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
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	42.8s
LR	0.977	0.988	5min 6s
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	55.8s
LR	0.764	0.764	5min 33s
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	57s
LR	0.554	0.571	5min 3s
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	31s
LR	0.976	0.991	5min
MNIST			
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	50.9s
LR	0.967	0.972	6min 21s
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)			
Boston House Pricing			
Linear	4.64	4.64	1min 5s
Lasso	5.15	4.64	7min 11s
Ridge	4.64	4.71	8min 43s
Adalone			
Linear	2.57	2.57	58.7s
Lasso	2.57	2.57	31min 51s
Ridge	2.57	2.57	32min 42s

Based on the results of Table no 2, it is observed that in RF, 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.

Hyperparameters

Tables no 3 and no 4 contains the results of the values of the experimental phase for every hyperparameter with regards to the classification and the regression tasks respectively. The first column includes

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 the possible values that the parameter could take, along with the type of the parameter. To this end, it should be noted the following:

- n_jobs was set to -1 to use all processors
- random_state was set to a random number (i.e. 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

Table no 3: Parameters summary from both frameworks on classification tasks.

Parameters of Algorithm	Sklearn (default)	HPO Iris	HPO Breast Cancer	HPO Titanic	HPO Wine Quality	HPO Banknote	HPO MNIST	HPO Melanoma	Search Space
RF (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]
KNN (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
SVM (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]
precomputed	-	-	-	-	-	-	-	-	-
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
MNB (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]
LR (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]
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	0.00 00	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 no 4: Parameters summary from both frameworks on regression tasks.

Parameters of Algorithm	Sklearn (default)	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]

IV. Discussion

Classification Results

Based on the results of Table no 3, it is observed that SVM Wine, Banknote and MNIST have the greatest number of observations compared to the other datasets, whereas all of them have similar c values, with a 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.

As for the results of KNN, one would expect that the number of neighbors (i.e., the most important parameter of the KNN algorithm) suggested by the hyperparameter tuning would match the number of instances that each dataset contained. However, in the examples that were examined in this experimental study, this definitely was not the case. To be more concrete, the dataset with the most observations was assigned the lowest value of neighbors (i.e. 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 the features in a dataset affects the selection of a distance metric. More accurately, the datasets with fewer features tend to be assigned with the option of the Euclidean distance, whereas datasets with a greater number of 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 NB 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 2 out of a total of 3 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 RF, one of the most important hyperparameters is the number of the estimators, which is actually the number of the trees in the forest. This appears to be highly correlated with the number of the independent variables in the dataset that 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 one 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, where 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.

In the example of LR for the Titanic and Banknote datasets, liblinear is a good choice of a solver, but for the Wine Quality dataset it is not the most appropriate 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.

On top of all the aforementioned, Hyperopt provides its own visualization module. Through this module it is easy to inspect and verify the results of Hyperopt's trials object. Figure no 7 outlines the different cases that were examined with regards to KNN, where each plotted point represents the best score achieved during each iteration. To this end, it should be noted that the corresponding results were produced for all the other classification algorithms (i.e. Support Vector Machines, Random Forest, Naive Bayes, and Logistic Regression), contributing into the effective inspection of the produced results.

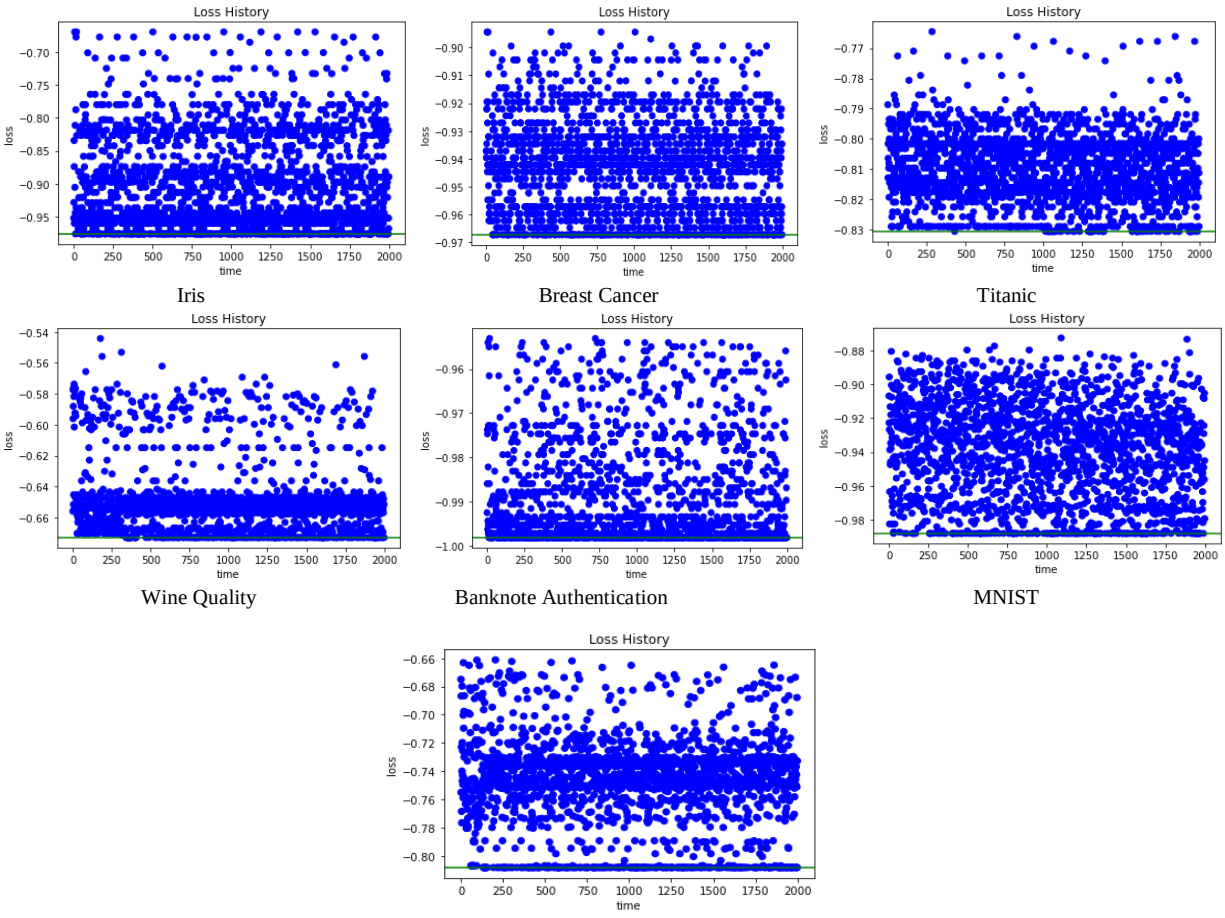


Figure no 7. KNN visualized results.

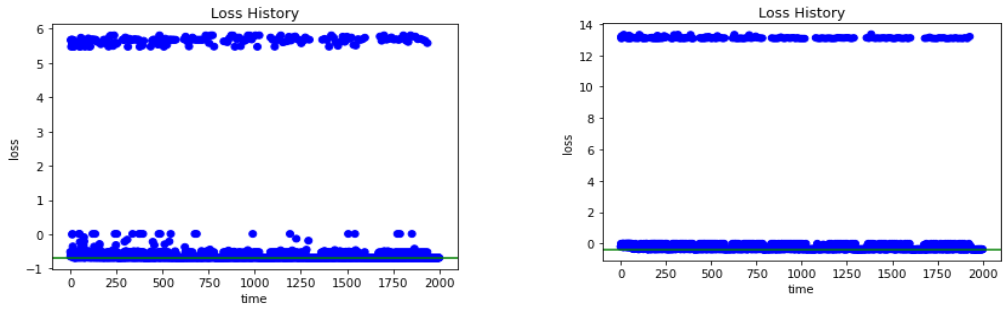
Regression Results

Based on the results of Table no 4, 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 4 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 the 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 the 3 algorithms and the 2 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 0 - if was equal to 0 it would be equivalent to ordinary least squares method that 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.

As in the case of the classification experiments, Hyperopt provides its own visualization module. Through this module it is easy to inspect and verify the results of Hyperopt's trials object. Figure no 8 outlines the different cases that were examined with regards to Lasso, where each plotted point represents the best score achieved during each iteration. To this end, it should be noted that the corresponding results were produced for all the other regression algorithms (i.e. Ridge), contributing into the effective inspection of the produced results.



Boston House Adalone

Figure no 8. Lasso visualized results.

V. Conclusion

In this manuscript we investigated 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 all cases of development of ML 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.

Into this context, 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 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 RF 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, SVM and LR showed positive results with respect to the performance, with a relatively small amount of execution time, thereby indicating that hyperparameter optimization would be worth to be applied in such cases. Furthermore, we should comment the results of the experiments related to the Melanoma dataset. All the produced results 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 additional 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) 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.

Acknowledgement

This research has been co-financed by the European Union and Greek national funds through the Operational Program Competitiveness, Entrepreneurship and Innovation, under the call RESEARCH – CREATE – INNOVATE (project code: DIASTEMA - T2EDK-04612).

References

[1]. The world's most valuable resource is no longer oil, but data. Available online: https://www.economist.com/leaders/2017/05/06/the-worlds-most-valuable-resource-is-no-longer-oil-but-data
[2]. Data is everywhere and it powers everything we do!. Available online: https://www.kdnuggets.com/2020/08/data-everywhere-powers-everything.html
[3]. Machine learning. Available online: https://searchenterpriseai.techtarget.com/definition/machine-learning-ML
[4]. Sala, R., Zambetti, M., Pirola, F., & Pinto, R. (2018). How to select a suitable machine learning algorithm: a feature-based, scope-oriented selection framework. In 23rd Summer School" Francesco Turco"-Industrial Systems Engineering 2018 (Vol. 2018, pp. 87-93). AIDI-Italian Association of Industrial Operations Professors.
[5]. Wistuba, M., Schilling, N., & Schmidt-Thieme, L. (2016, October). Hyperparameter optimization machines. In 2016 IEEE International Conference on Data Science and Advanced Analytics (DSAA) (pp. 41-50). IEEE.
[6]. Bergstra, J., Yamins, D., & Cox, D. D. (2013, June). Hyperopt: A python library for optimizing the hyperparameters of machine learning algorithms. In Proceedings of the 12th Python in science conference (Vol. 13, pp. 20). Citeseer.
[7]. Bergstra, J., Bardenet, R., Bengio, Y., & Kégl, B. (2011, December). Algorithms for hyper-parameter optimization. In 25th annual conference on neural information processing systems (NIPS 2011) (Vol. 24). Neural Information Processing Systems Foundation.
[8]. Caruana, R., & Niculescu-Mizil, A. (2006, June). An empirical comparison of supervised learning algorithms. In Proceedings of the 23rd international conference on Machine learning (pp. 161-168).
[9]. Breiman, L. (2001) 'Random Forests'. Machine Learning 45, 5–32.
[10]. Sirikulviriya, N., & Sinthupinyo, S. (2011, May). Integration of rules from a random forest. In International Conference on Information and Electronics Engineering (Vol. 6, pp. 194-198).
[11]. Lim, C., Lee, S. R., & Chang, J. H. (2012). Efficient implementation of an SVM-based speech/music classifier by enhancing temporal locality in support vector references. IEEE Transactions on Consumer Electronics, 58(3), 898-904.
[12]. Yang, L., & Shami, A. (2020). On hyperparameter optimization of machine learning algorithms: Theory and practice. Neurocomputing, 415, 295-316.
[13]. Tay, B., Hyun, J. K., & Oh, S. (2014). A machine learning approach for specification of spinal cord injuries using fractional anisotropy values obtained from diffusion tensor images. Computational and mathematical methods in medicine, 2014.
[14]. Hosmer Jr, et al. (2013). Applied logistic regression (Vol. 398). John Wiley & Sons.
[15]. Angel, L., Viola, J., Vega, M., & Restrepo, R. (2016, August). Sterilization process stages estimation for an autoclave using logistic regression models. In 2016 XXI Symposium on Signal Processing, Images and Artificial Vision (STSIVA) (pp. 1-5). IEEE.
[16]. Naïve Bayes Algorithm: Everything you need to know. Available online: https://www.kdnuggets.com/2020/06/naive-bayes-algorithm-everything.html
[17]. Kibriya, A. M., et al. (2004, December). Multinomial naive bayes for text categorization revisited. In Australasian Joint Conference on Artificial Intelligence (pp. 488-499). Springer, Berlin, Heidelberg.
[18]. Lowd, D., & Domingos, P. (2005, August). Naive Bayes models for probability estimation. In Proceedings of the 22nd international conference on Machine learning (pp. 529-536).
[19]. Hoerl, A. E., & Kennard, R. W. (1970). Ridge regression: Biased estimation for nonorthogonal problems. Technometrics, 12(1), 55-67.
[20]. Cui, Z., & Gong, G. (2018). The effect of machine learning regression algorithms and sample size on individualized behavioral prediction with functional connectivity features. Neuroimage, 178, 622-637.
[21]. Zou, H., & Hastie, T. (2005). Regularization and variable selection via the elastic net. Journal of the royal statistical society: series B (statistical methodology), 67(2), 301-320.
[22]. Tibshirani, R. (1996). Regression shrinkage and selection via the lasso. Journal of the Royal Statistical Society: Series B (Methodological), 58(1), 267-288.
[23]. Park, T., & Casella, G. (2008). The bayesian lasso. Journal of the American Statistical Association, 103(482), 681-686.
[24]. Feurer, M., & Hutter, F. (2019). Hyperparameter optimization. In Automated Machine Learning (pp. 3-33). Springer, Cham.
[25]. Bergstra, J., Bardenet, R., Bengio, Y., & Kégl, B. (2011, December). Algorithms for hyper-parameter optimization. In 25th annual conference on neural information processing systems (NIPS 2011) (Vol. 24). Neural Information Processing Systems Foundation.
[26]. Tanay Agrawal (2020). Hyperparameter Optimization in Machine Learning: Make Your Machine Learning and Deep Learning Models More Efficient.
[27]. F Pedregosa, et al. (2011). Scikit-learn: Machine learning in Python. the Journal of machine Learning research, 12, 2825-2830.
[28]. Bergstra, J., et al. (2013). Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. In International conference on machine learning, 115-123).
[29]. UCI Machine Learning Repository. Available online: https://archive.ics.uci.edu/ml/datasets/iris
[30]. UCI Machine Learning Repository. Available online: https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+(Diagnostic)
[31]. Titanic - Machine Learning from Disaster. Available online: https://www.kaggle.com/c/titanic
[32]. Cortez, P., Cerdeira, A., Almeida, F., Matos, T., & Reis, J. (2009). Modeling wine preferences by data mining from physicochemical properties. Decision support systems, 47(4), 547-553.
[33]. UCI Machine Learning Repository. Available online: https://archive.ics.uci.edu/ml/datasets/banknote+authentication
[34]. MNIST handwritten digit database. Available online: http://yann.lecun.com/exdb/mnist/
[35]. Harrison Jr, D., & Rubinfeld, D. L. (1978). Hedonic housing prices and the demand for clean air. Journal of environmental economics and management, 5(1), 81-102.
[36]. Nash, W. J., et al. (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, 48, p411.
[37]. Python. Available online: https://www.python.org/
[38]. Scikit-learn. Available online: https://scikit-learn.org/stable/
[39]. Hyperopt: Distributed Asynchronous Hyperparameter Optimization. Available online: http://hyperopt.github.io/hyperopt/
[40]. Pandas. Available online: https://pandas.pydata.org/
[41]. NumPy. Available online: https://numpy.org/