Assignment report, group number 50

Antonio Mone, Dimitrios Ieronymakis, Maria Ieronymaki

Leiden Institute of Advanced Computer Science, The Netherlands

Abstract. This document contains the report of the work that has been done regarding the assignment of the Introduction to Machine Learning course. The work has been carried out on the analysis of a dataset about bike rentals in a large European city.

1 Introduction

The goal of this report is to show the results of our work that focused on the analysis of a dataset about bike rentals in a large European city, using some of the algorithms discussed during the lectures. The main objective of this work was to predict the amount of bikes rented (by subscribers to the service and by non-subscribers) based on the other features in the dataset. Firstly, an accurate analysis of the data set provided is carried out. Secondly, a description of the preprocessing applied to the data set is explained, with different techniques, in order to have a more comprehensive view of the data set. This is followed by a formulation of the problem and a description of the performance metrics considered during this work. Subsequently, in the Experiments section [5], a considerate amount of experiments has been carried out on the data set, with different supervised and unsupervised methods that have been applied in order to experiment, study and analyse the differences between the techniques applied. Finally, a description of the method that proved to have the highest accuracy is given, with a related analysis of the reasons why the selected method is the one that performs best.

2 Data Set

The first step in the analysis of the dataset has been the identification and consideration of the variables, their distribution and how they are represented. The dataset is composed by the 15 following variables: instant, date, season, year, month, hour, weekday, weather, temperature, feeling_temperature, humidity, windspeed, Subscribed, Non-subscribed, Total. From a first scan of the content of the dataset it is possible to see that the information is represented with different datatypes (String, Float and Integer). Since the dataset has been imported into a *pandas.DataFrame* object, it is possible to use the $pandas.DataFrama.isna().sum()$ function, that shows the number of missing values for each variable. In this dataset no value is missing. For the same reason we can also use the $pandas.DataFrame.describe()$ function, that shows other statistics metrics about our data, like the mean for each variable and also the standard deviation and the minimum and maximum value that each variable assumes in the dataset. The next step is to have a graphical representation of distribution patterns in our data, taking into considerations the variables temperature, humidity, windspeed, Total, Subscribed and Non-subscribed, as showed in Fig. 1.

The variables temperature, humidity and windspeed are continuous variables, while Total, Subscribed and Non-subscribed are discrete variables. On the X-axis of each graph there is the single

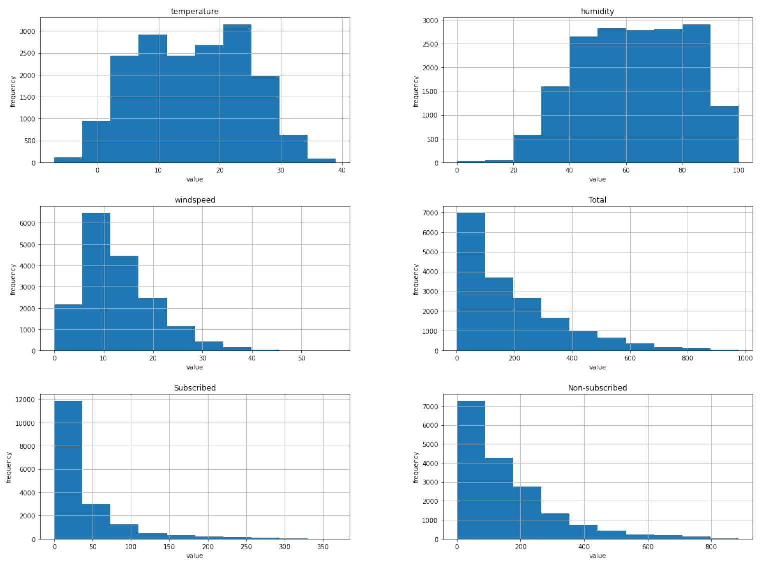


Fig. 1: Distribution patterns

value of the variable, while on the Y-axis there is the number of occurrences of that value in the dataset. As we can see from the graphs in Fig. 1 the temperature variables can be identified as a Standard Normal Distribution, the humidity can be identified as a Beta Distribution, while the windspeed variable can be identified as a Gamma Distribution. On the other side the Total, Subscribed and Non-Subscribed variables can be identified as Poisson Distributions. [17]
Looking instead at the correlations between the variables in relation with the Total, Subscribed and Non-Subscribed variables it is clear how Non-Subscribed and Total are very strongly correlated, the variables Total and Subscribed are strongly correlated. Interesting to notice how the windspeed variable is weakly correlated with Total, Subscribed and Non-Subscribed. The humidity variable instead shows negative weak correlation values.
Also, as showed in Fig. 2 there are a lot of outliers in our target variables which we can scale by using the natural logarithm. As far as windspeed is concerned, we will simply drop the feature, as it contains too many outliers.

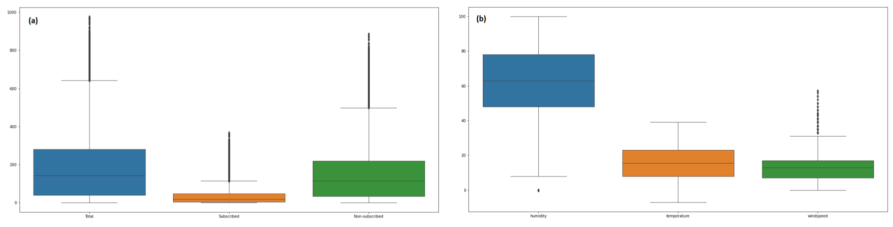


Fig. 2: Boxplot of: (a) Total, Subscribed and Non-Subscribed, (b) Humidity, Temperature and Windspeed.

An intresting note can be made after grouping the data by weekday and by month (Figures 4 and 3) it's noticeable that subscribed people tend to rent bikes on weekends. A possible explanation for this might be that subscribed people are probably residents of the city and tend to use their subscriptions when not at work, which is usually during the weekends. On the other side, Non-subscribed people,who might be visitors or tourists of the city, tend to rent bikes during weekdays. As visitors are usually on holiday for a whole week or more it makes sense that most rentals happen during the weekdays as they tend to return to their home country/city during weekends, to get prepared for next week's obligations like work or university.
Also, in general we see that the number of bike rents ends to increase during spring and summer months which is to be expected.

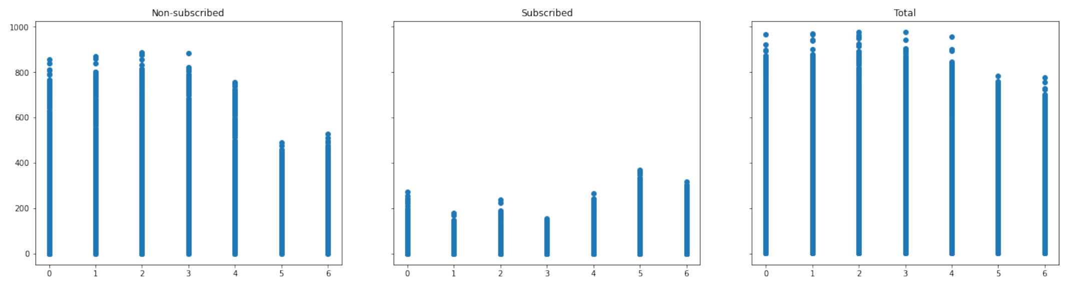


Fig. 3: Number of bikes rented on each day of the week (0-4 weekdays, 5-6 weekend)

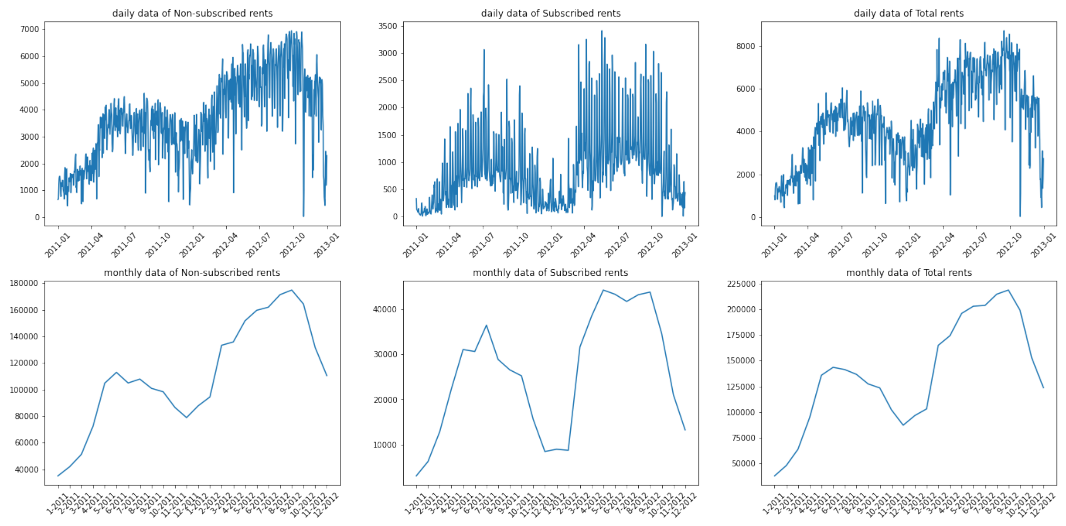


Fig. 4: Plots that represent the number of rented bikes per day and per month for the Non-Subscribed, Subscribed and Total data.

3 Preprocessing

After the analysis of the dataset it is recognizable that some pre-processing techniques needs to be applied in order to make the data fully usable. For this part of the assignment three methods have been tested: a label encoding approach, a mixed one hot encoding and labelling approach and a full one hot encoding approach.
The label encoding approach consists in preprocessing all categorical variables with labels. This includes all the date features like month or weekday, and the weather and season features. The mixed one hot encoding and labelling approach consists into preprocessing the month, the weekday, and the year columns by making them cyclical (with sine and cosine functions), and use a one hot encoding approach only for the season and weather values. We believe that cyclical values may yield to better results. The full one hot encoding approach converts not only the categorical variables but also the date related ones into dummy variables. We acknowledge the fact that this approach increases the dimensionality of our dataset, as new columns are created for each unique element of the variable considered (e.g., 12 new columns are only created for the month variable!), which consequently causes an increase on the complexity of the model.
We note that one column for each variable one hot encoded is dropped to avoid creating multicollinearity; a condition in which columns are very highly correlated with each other. It is very important to do this, because if present in the data, the statistical conclusions may not be reliable. [15]
All the preprocessing approaches have been tested on Linear Regression models in order to evaluate and select the best preprocessing strategy. The best performing model will then dictate which preprocessing approach we will use to evaluate the following models.

4 Problem Formulation and Performance Metrics

The problem of predicting the number of bikes rented in a large city is a classic case of supervised learning as the scope is to train our model to map inputs to outputs based on known but limited set of input-output pairs (called training data) [14]. In our case, the inputs are the values that we retrieve from the data set and the output is the number of total bikes rented.
Linear Regression is one of the first and most common approach to tackle this kind of situation, since it is capable of making accurate predictions because it allows us to identify the relations between our inputs and output variables. Specifically, the objective of our case is to predict the count of bike rental based on the season, climate and other environmental conditions which usually have an impact on someone's decision whether to rent a bike or not.
But Linear Regression is not the only method for addressing this kind problem. Other ways to tackle this problem, that are also going to be experimented during this work, are Logistic Regression, Decision Tree Regressor, Perceptron, Support Vector Machines, Principal Component Analysis, Clustering, Random Forest and Multilayer Perceptron. In order to implement all these methods we will make use of the Scikit-Learn library of Python, and in particular we will use firstly the Linear Regression model [6] for the prediction of the total number of bike rentals, while later a multiple binary classification version of this problem be evaluated and addressed with Logistic Regression (using the One-Vs-One and the One-Vs-All approaches) in Section 5.3. Furthermore we will use a Decision Tree Regressor [12] (explained in more detail in Section 5.2) to predict the number of bikes rented by subscribers and non-subscribers. The problem of

predicting the number of bikes rented by subscribers will also be addressed with the Support Vector Machine method, in Section 5.5, and with the Random Forest method, in Section 5.8.The data will be also clustered using hierarchical clustering in Section 5.7, using all features (except the number of rented bikes). Principal Component Analysis will also be used, in order to reduce the dimensionality of the starting dataset to 2 (using all the features except the number of rented bikes), and the results of the application of PCA to the dataset will be used together with the clustering method. First a limited amount of clusters will be projected on the 2D PCA space, then the clustering method will be applied directly to the PCA-transformed data. Also, the Perceptron method will be used to predict the weather situation based on other some other features in the data set, considering only a subset selection of features and it will also be evaluated as a binary classification problem, with a One-Vs-One approach. It is very important before making any assumptions to verify the performance of our model, and we are going to test it with a few typical measures used to evaluate regression and classification models:
- Mean Square Error (MSE) which gives an idea of how much error the system typically makes in its predictions, with a higher weight for large errors [14].
- R^2 score which measures the strength of the relationship between our model and the dependent variable on a convenient 0 – 100% scale [13].
- $F1$ score, which represents the weighted average of $Precision$ and $Recall$.
$Precision$ represents the ratio of correctly predicted observations to the total predicted observations while $Recall$, on the other hand, looks at the performance from a different perspective, representing the measurement that specifies how many observations that should have been predicted to a given label were actually predicted with that label. [19] [18]

5 Experiments

This section reports all the experiments conducted on various well-known Machine Learning models and the results obtained.

5.1 Linear Regression

Label, full one hot and mixed encoding

The first approach that has been tested is label encoding of all categorical variables. Then, the mixed one hot encoding and labelling method, and finally, the full one hot encoding method. All three methods have been tested on different linear regression models on the dataset that has been split in training and test set (in different sizes, in order to test also how the number of elements in the training set influences the performance of the model) and for each one of these models the MSE and the R^2 values have been calculated and plotted, as shown in Figure 5.

The values on the X-axis represent the percentages of the sizes of the test sets (and for each test set size the training set size is equal to $1-$ the size of the test set) that have been experimented. As shown in Table 1, the size of the test data does not influence much the R^2 score and MSE values.

In the experimentation of the Full One Hot Encoding method, using the $pandas.get\_dummies()$ function the variables weather, season, weekday, month, hour variables have been encoded. After the encoding of these variables again different linear regression models have been created, in order to notice how the performances vary based on the sizes of the training set. Once more for each one of the models based on the different sample sizes the MSE and the R^2 values have been calculated and plotted, as shown in Figure 6.

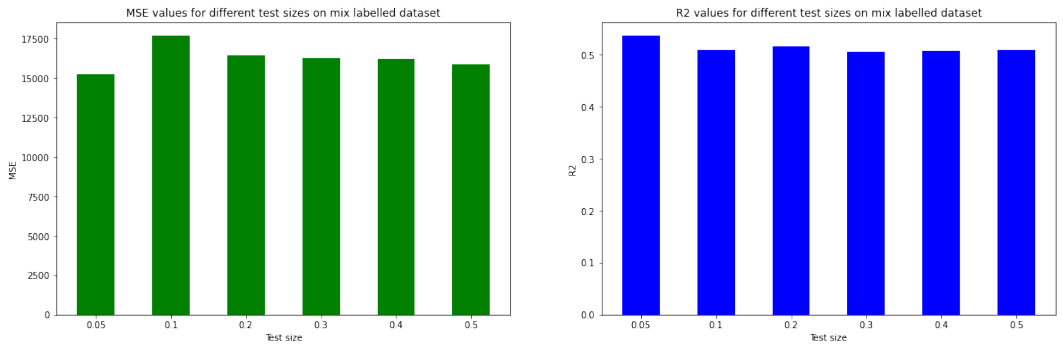


Fig. 5: MSE and R^2 values for different training and test sets sizes with a mixed one hot encoding and labelling approach

Test Sample Size	R^2 **Scores**	**MSE Values**
0.05	0.536	15246.17
0.10	0.509	17686.67
0.20	0.515	16422.61
0.30	0.504	16248.64
0.40	0.508	16227.81
0.50	0.508	15891.78

Table 1: R^2 scores and MSE values for each test sample size of the Mixed One Hot and Labelling model

As shown in Table 2 the full one hot encoded model performs a lot better than the previous one, which concludes the experimentation on data preprocessing strategies.

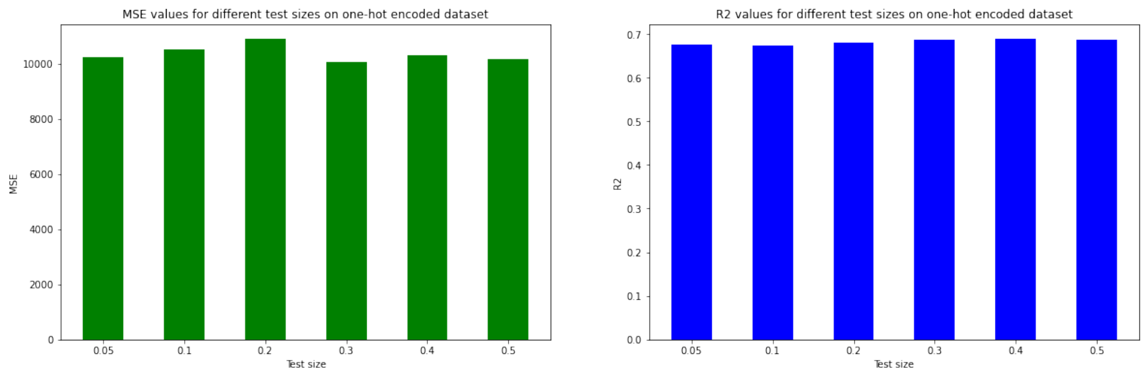


Fig. 6: MSE and R^2 values for different training and test sets sizes with a Full One Hot Encoding approach

Test Sample Size	R^2 Scores	MSE Values
0.05	0.676	10231.56
0.10	0.672	10519.21
0.20	0.680	10888.50
0.30	0.687	10063.96
0.40	0.688	10301.42
0.50	0.687	10152.83

Table 2: R^2 scores and MSE values for each output transformation of the Full One Hot Encoded model

Target transformation

As the part of preprocessing is done, we proceed with the target transformation. In particular, we transform our target values in order to see how our model performs, focusing only on the fully one hot encoded dataset since it yields better R^2 results.
Several transformation techniques have been utilized to do so: the natural logarithm, the square root, and the square on the Total column.
We notice that the accuracy of our model drastically improves if we consider the natural logarithm of the target value. However, it degrades if we take the squared value of the target. The square root also gives some good results, concluding to the fact that using transformations that scale down our target leads to a better performance of our model (see Table. 3).
This large impact on the R-squared metric is based on the down scaling of our real output. If we take a look at the R-squared metric formula, $R^2 = 1 - \frac{SSres}{SStot}$, where $SSres$ represents the residual sum of squares and $SStot$ represents the total sum of squares (proportional to the variance of the data), if the output values are smaller, the mean of the output decreases which leads to an increase of the Total sum of squares. As a consequence, the relation between the Sum of squares of residuals and the Total sum of squares becomes smaller and the R^2 score improves [13].

Further experiments have been conducted on our data of our one hot encoded model, like for instance, scaling the temperature, humidity and windspeed values between 0 and 1. However, this did not improve the R^2 score of ourLinear Regression model (R^2 = 0.6546).

Transformation	R^2 **Scores**	**MSE Values**
Natural Logarithm	0.829	3.735588e-01
Square Root	0.789	9.533646e+00
Power of 2	0.514	7.028751e+09

Table 3: R^2 scores and MSE values for each test sample size of the Full One Hot Encoded model

5.2 Decision Tree Regressor

In the next experiment we created a Decision Tree Regressor (importing *DecisionTreeRegressor* from *sklearn.tree*) in order to predict the number of bikes rented by Subscribers. The parameters of the *sklearn.tree.DecisionTreeRegressor* class are criterion, splitter, max_depth, min_samples_split, min_samples_leaf, min_weight_fraction_leaf, max_features, random_state, max_leaf_nodes, min_impurity_decrease, ccp_alpha [12]. Using only the default values of the parameters, the Decision Tree Regressor has a R^2 score of 0.609 for the Subscribed users. Analyzing each one of these parameters we identified the parameters that manage the splitting of a node as the most important ones (*min_samples_split* before the others), and since in the description of the *min_samples_leaf* parameters it is stated that this parameter "may have the effect of smoothing the model, especially in regression" [12], we have decided to analyze how the min_samples_splits and the min_sample_leaf parameters affect the R^2 score metric. To do so, 1000 possible configuration have been tested for each one of these two parameters, with $min\_samples\_split$ going from 0.00001 to 0.1 and $min\_samples\_leaf$ going from 0.00001 to 0.1.

Using the default values for the $min\_samples\_leaf$ parameter (equal to 1), assigning the value of 0.00001 to the $min\_samples\_split$ and fitting the tree on the transformed target train set, yields the best R^2 score for this tree, which is equal to 0.70 for the Subscribed rented bikes. On the other hand, assigning the default value to the $min\_samples\_split$ and the value 0.00001 $min\_samples\_leaf$, yields an R^2 score of 0.64. Overall, the $min\_samples\_split$ property is the one that can achieve higher scores for other parameter default values, so we can regard it as one of the most important hyperparameters to tune.

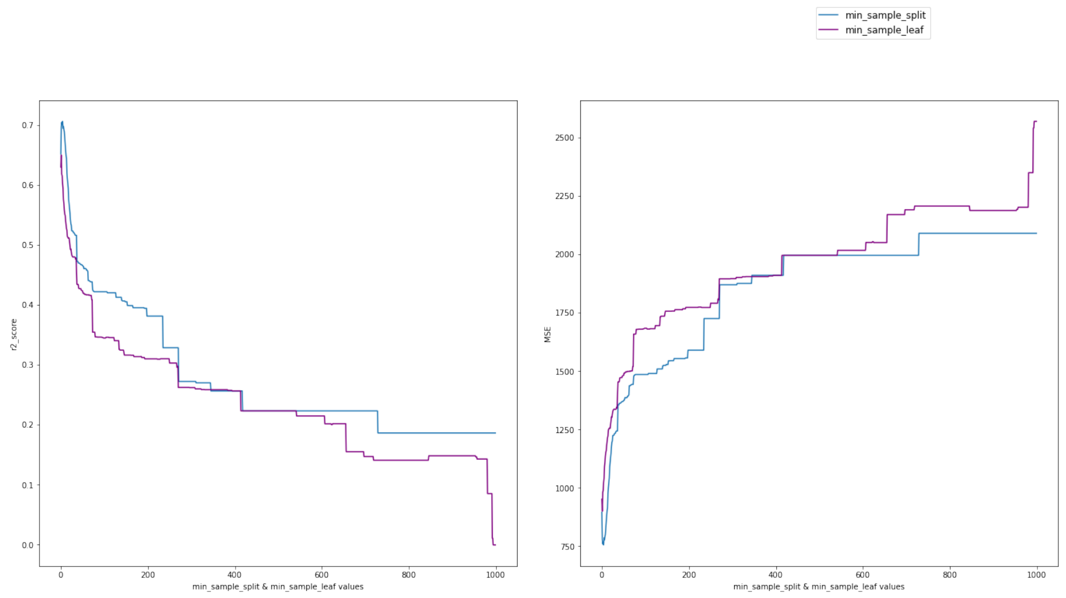


Fig. 7: R^2 and MSE scores for each $min\_samples\_split$ and $min\_samples\_leaf$ values.

Finally, we run a Gridsearch in order to find the best hyperparameters to predict Subscribed and Non-subscribed rents (scaled with the natural logarithm since it yields better results). As it turns out, the hyperparameters that achieve the best scores are different for the two target variables. The parameters utilized to create a Decision Tree Regressor to predict the Non-subscribed rents is significally more complex of the one used to predict Subscribed rents. The final R^2 scores we are able to achieve with the best found parameters are 0.75 for Subscribed rents and 0.82 for Non-subscribed rents.

5.3 Logistic Regression

The Logistic Regression model is a method used when the dependent variable (target) is categorical. In this case, assuming that the variable to predict is the total number of rented bicycles, we transform our problem into a multiple binary classification problem by dividing the data of the column into 5 different classes of equal range. Two implementations that are usually used in multiclass classification are the One vs.One and the One vs. All methods.

In order to implement them in our code, we used the OneVsOneClassifier of Scikit-Learn [8] and the Logistic Regression [7] and set the multi_class parameter to OVR.

One vs All

The One vs. All model trains a binary classifier for each class of the target. Specifically, since we created 5 classes, the One vs. All generates 5 binary classifier models. The results obtained are represented through the classification report shown below:

Class	Precision	Recall	F1-score	Support
0	0.86	0.92	0.89	3157
1	0.57	0.64	0.60	1306
2	0.41	0.22	0.28	527
3	0.55	0.24	0.34	169
4	0.83	0.35	0.49	55
accuracy			0.75	5214
micro avg	0.64	0.47	0.52	5214
weighted avg	0.73	0.75	0.73	5214

Table 4: Classification report for One vs. All

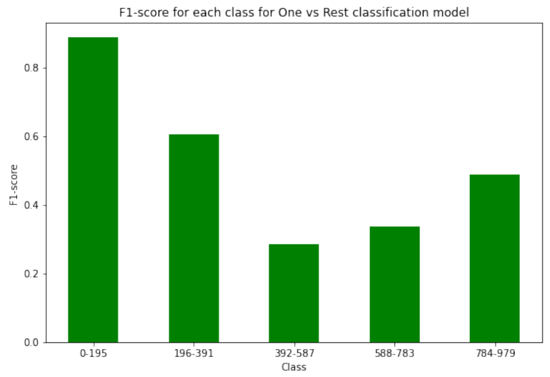
One vs One

On the other hand, the One vs One method, builds a binary classifier for every pair of the target classes. Specifically, for an N-class instances output, $\frac{N\cdot(N-1)}{2}$ binary classifier models have to be generated. In our case, the One vs. One method generates 10 models. Again, the results obtained are represented through the classification report shown below:

Class	Precision	Recall	F1-score	Support
0	0.88	0.91	0.89	3157
1	0.64	0.66	0.65	1306
2	0.56	0.45	0.50	527
3	0.47	0.39	0.43	169
4	0.67	0.51	0.58	55
accuracy			0.78	5214
micro avg	0.64	0.58	0.61	5214
weighted avg	0.77	0.78	0.77	5214

Table 5: Classification report for One vs. One

Another representation of the results is shown in Fig. 8. the height of each bar represents the F1 score in relation to each class for the One vs. One and One vs. All methods. The classes are: 0-195, 196-391, 392-587, 588-783, 784-979.



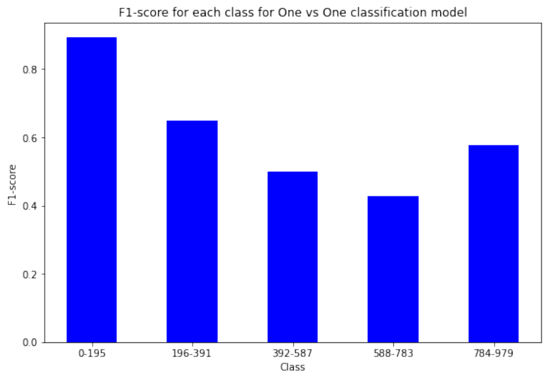


Fig. 8: On the left, the F1 scores for the One vs. Rest model in relation to the five classes. On the right, the F1 scores for the One vs. One model in relation to the five classes.

The results obtained indicate that both methods perform satisfactorily, with the one vs one method performing slightly better, obtaining an overall F1 score of 0.78 compared to the 0.75 of the One vs All method. It should be noted that class 0 has an F1 score significantly higher than the other classes. This can be attributed to the uneven distribution of the data in our respective classes, depicting the presence of a long-tail distribution. For instance, the 'support' column in the classification report, shows that class 0 which indicates the number of rented bikes between 0 and 195 has more than 3000 data, while class 4 has 51. It would be expected that the latter's f1-score would be the worst, due to the limited amount of data present, but this is not the case. However, classes 1, 2 and 3 illustrate a decreasing performance, proportional to their respective sample sizes and the long-tail distribution. In general, we can conclude that the One vs all, although it is faster as far as computation is concerned, the One vs One method is better because it less prone to the creation of an imbalance in the dataset, even if the data is not distributed evenly among the existing classes.

5.4 Perceptron

Inspired by the concept and structure of the biological neuron, the perceptron is one of the fundamental algorithms of the Neural Networks field. The perceptron model is a single layer neural network and it is used for binary classification. This model is a very important one because, assuming separability, it has been proven that it will always converge. Resembling the structure of a biologican neuron, that take the inputs from its dendrites, in the same fashion the perceptron receives some inputs that are weighted (with weights that are initialized randomly and that are updated during training). All these weighted inputs are elaborated by a net input function (or transfer function), usually the sum of the inputs times the related weights, and the output of this net function represents the input of an activation function, the part of the perceptron that actually discerns if the input is an element of a class or the other. The most commonly used are the step function, the sign function and also the sigmoid function. At the end of every iteration during the training, the weights of the network are updated, backpropagating the error between the prediction and the true value through the network, allowing the weights to adjust and be more accurate.

This model was used to predict the weather situation based on other some other features in the dataset. The classes present in the Weather column are 4: Clear or partly cloudy, Mist, Light rain and Heavy rain. For this reason we used the One vs One method on the standardized dataset to transform this task from a multiclass classification into a binary classification problem.

Feature subset selection

The feature subset selection was performed by applying the *SelectKBest* method of Scikit-Learn [5] which selects the features according to the k highest scores. The scores were calculated with the *mutual_info_classif* of Scikit-Learn [4] which estimates the mutual information for a discrete target variable. The mutual information measures the dependency between the variables. It is equal to zero if and only if two random variables are independent, and higher values mean higher dependency. The performance of the model was tested not only on the different number of selected features - different k values - but also on different random states. This was done because it was previously noted that the results obtained depended heavily on the selected random seed. The F1 scores obtained are therefore an average of the F1 scores calculated for 30 different random seeds. From the results in Figure 9 we deduce that approximately 25 features of the dataset are enough to have a model with a satisfactory performance.

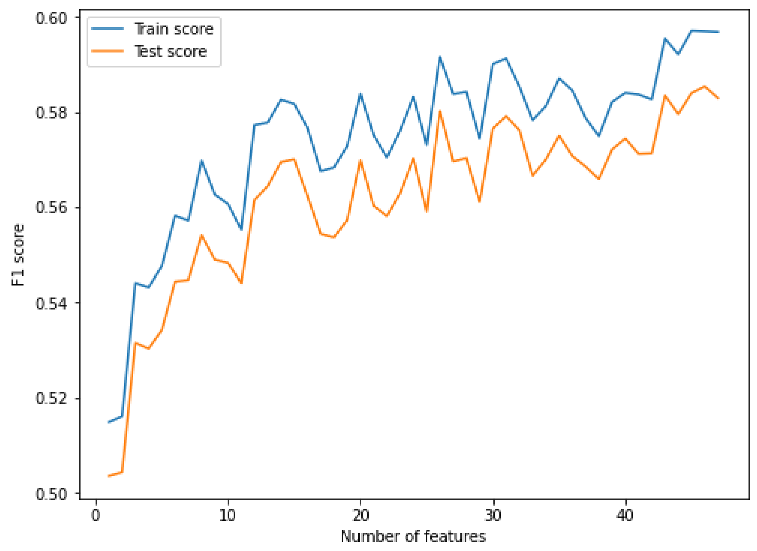


Fig. 9: F1 scores in relation to the different number of features.

Learning Rate

After deciding the number of features to select, we proceeded with the evaluation of the perceptron's performance based on different values of the learning rate. As far as the general model is concerned, where all the classes of the weather column were considered, the learning rates evaluated were 25 and fell within a range from 0.0005 to 0.1.

The impact that the learning rate had on the performance of the model was also evaluated on a binary classification model between only two classes. The 'Heavy rain' class was not taken into consideration as only 3 records were present in the database. Among the three remaining, the two were selected randomly. The One vs One method was applied to a new target column made up of binary values of the 'Mist' and 'Clear or partly cloudy' classes. The learning rates evaluated were 25 and fell within a range from 0.0005 to 1.

Once again, due to randomness, the F1 scores were evaluated not only based on different learning rate values, but also with respect to several random seeds. This means that, the F1 scores shown in Fig.10 correspond to the average values calculated on 30 different seeds for each learning rate.

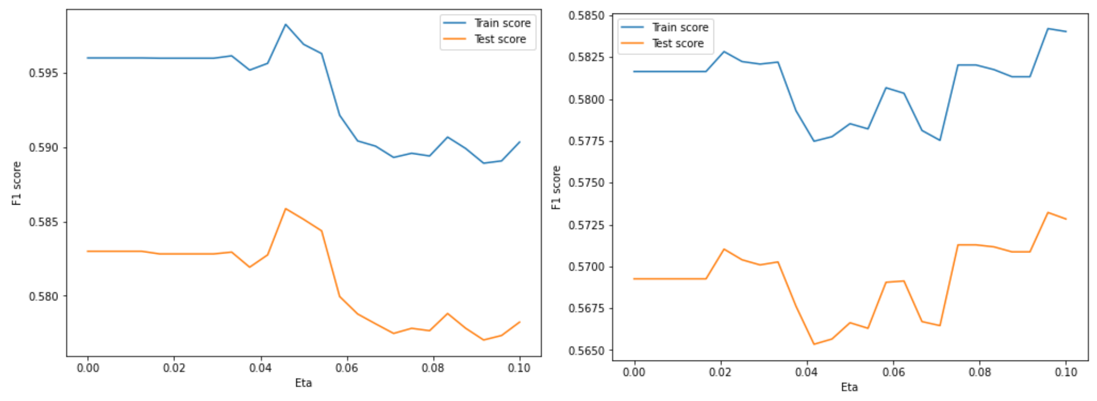


Fig. 10: F1 scores in relation to the learning rates. On the left the general One vs One method and on the right the binary classification between Mist and Clear or partly cloudy classes.

The two plots represent the average trend of the F1 score with respect to different learning rates. As far as the general model is concerned (Fig.10,left), the maximum performance of approximately 0.6 on the train set and 0.587 on the test set was reached with a learning rate of 0.05. As for the One vs One model in which only two of the four classes of the weather column (Fig.10,right), the maximum F1 score reached corresponds to 0.574 on the test set and 0.585 on the train set for a learning rate of 0.1. For a value greater than 0.2, the model's F1 score begins to worsen. Overall, based on the performances obtained both in the train and test set, we can conclude that the Perceptron model is not adequate to classify the weather condition. The poor scores obtained on the train set makes us deduce that the problem is not linearly separable and that therefore this algorithm does not find a perfect separating hyperplane. Predicting the weather is much more complicated than we thought and therefore most likely a more complicated model such as a Multilayer Perceptron classifier with more hidden layers would be able to better classify the different classes.

5.5 Support Vector Machine Regressor

Based on the concepts of linear halfspace, hyperplane and margin of an hyperplane, Support Vectore Machine (SVM) is a supervised algorithm mostly used for classification problems. The main aim of this algorithm is to find the hyperplane that creates the biggest margin between classes, in order to have a clear separation on the set even if the data points have noise. It does this considering the support vectors, the data points that are closest to the data points of the other class. The found hyperplane is the one that has the biggest margin from both classes. Other than being capable of separating classes quite nicely on a n-dimensional space (where n represents the number of features), this method has also a very good tolerance for outliers. In case of overlapping classes this method is still capable of maximing the margin, but has a slack variable that allows a few points to be on the wrong side of the margin. Furthermore, this algorithm is capable of addressing also problems that on the n-dimensional space do not have a linear solution. It does this by mapping via a kernel the considered data from a n-dimensional space to a space with a higher dimensionality, with the assumption that in a higher dimensionality space the classifier is going to be linear, where in the n-dimensional is nonlinear.

In order to implement the SVR model on our code we used use Scikit-Learn's LinearSVR class [11] and set the parameter epsilon to the default value of 0.1.

A crucial step before proceeding with the creation of our model that led us having decent results was to standardize our data with $StandarScaler()$ [10]. This was important because for models such as Support Vector Machines, where the distance between features is taken into consideration, distances regarding non-scaled and scaled data could lead in the generation of totally different models.

The type of kernels selected in order to identify the impact on the SVR model performance were: Linear, Polynomial and Rbf. The Figure below (Figure 11) represents the score achieved by the model based on the different kernel selected. From it we can obviously deduce that the linear kernel, as expected, leads to a poor model performance. As we mentioned in the Perceptron section 5.4, this comes from the fact that the problem is not linearly separable. On the other hand, the SVR model with either a polynomial or rbf kernel performs in a decent way, reaching an R^2 score sligthly less than 0.8, with rbf being the best among the other two.

5.6 Principal Component Analysis

Principal Component Analysis (PCA) can be considered as a technique for both dimensionality reduction and variance maximization. PCA is widely used to reduce the dimensionality of large datasets, while minimizing information (*'variability'*) loss. A way to do that is to select the axis that preserves the maximum amount of variance, since maximizing the variance minimizes the squared error. What this consists in is finding variables that are linearly dependant on the variables of the original dataset, uncorrelated with each other and that maximize the variance [16].

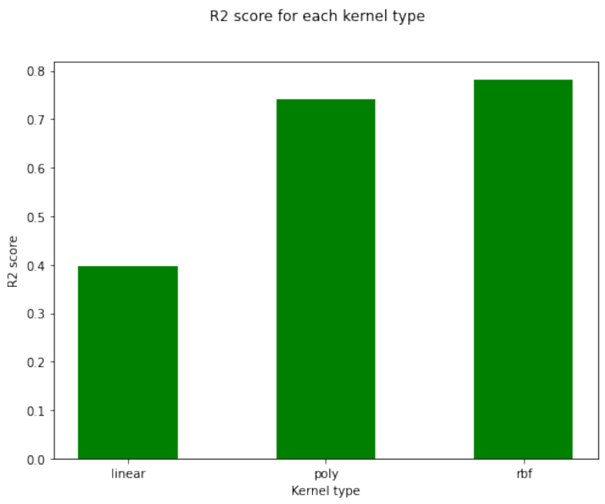


Fig. 11: R^2 score for the SVR model and the three kernels: Linear, Polynomial, Rbf

PCA finds the axis that accounts for the largest amount of variance in the set, since they are the Principal Components (PC) of the set. For each principal component PCA finds the unit vector that points in the direction of the PC [14]. We used PCA analysis on the one hot encoded dataframe and the label encoded one. When keeping the whole set of features, we notice that the one hot encoded dataset produces interesting results (see Figure 12). The graphs resembles clouds of points that are very well grouped together by their features, especially when looking at the graphs with the data colored by season and month. Applying PCA to the label encoding dataset produces more unsatisfactory results, as it can be seen in Figure 13. The observed clustering has significant overlap with no clear separation in the hyperplane and the usage of the same coloring for all the classes further enhances this observation, since there is no separation in our dimensional space. In addition, we removed iteratively each one of the features to evaluate their importance, by applying the PCA and comparing the results. Analysis of the received PCA graphs (see Figure 14), indicates that the most important feature is the *season*. The removal of this feature has worsened the clustering and minimized the separation of the classes in our hyperplane.

5.7 Clustering

Clustering is an unsupervised learning method based on similarity. It can be considered the natural analogy of classification, but with the lack of known labels. It is very useful in situation in which, looking at the data for the first time, we are looking for subgroups or patterns. The concept of similarity is fundamental in a situation in which there are no labels, since a similarity

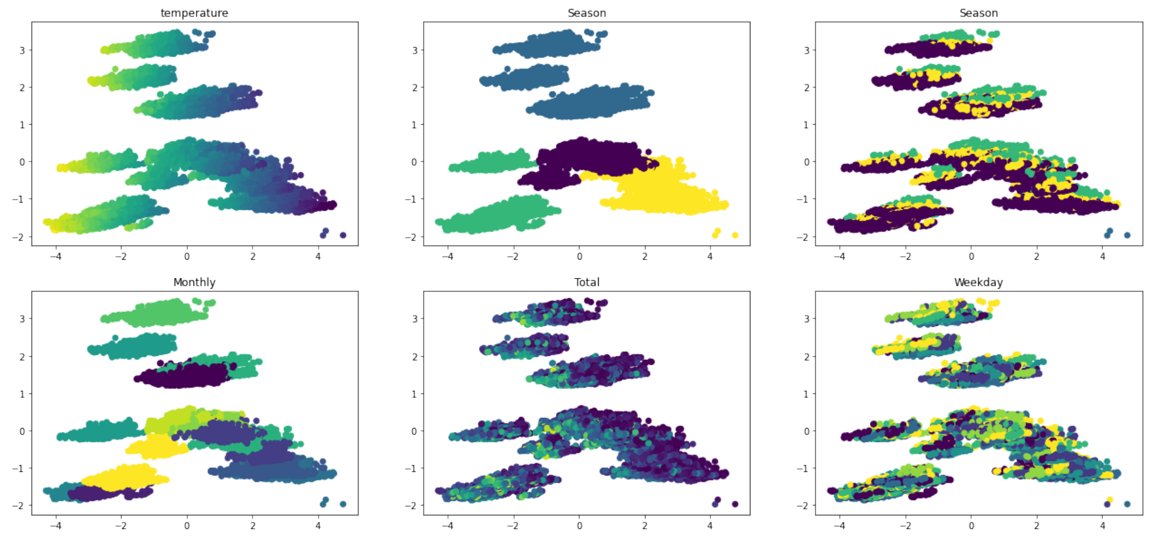


Fig. 12: PCA on the full one hot encoded dataset

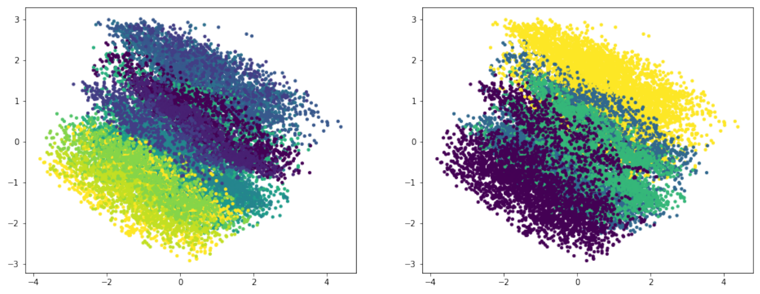


Fig. 13: PCA on the full label encoded dataset

between observations is the basis on which the observations are going to belong to the same class or not. The aim is to have similar data points grouped together and different groups of data points divided from each other. *Hierarchicalclustering* is a specific category of clustering algorithms that creates clusters by merging or splitting items successively. It is called *Hierarchical* because the clusters are going to be divided in levels, where at the first level there is one cluster of all observations and at the lowest level each cluster will contain one observation. In order to implement hierarchical clustering we utilized the *AgglomerativeClustering* class of Scikit-Learn [2]. This algorithm performs the hierarchical clustering starting by clustering each item in its individual cluster and consecutively merging together similar clusters.
Before applying hierarchical clustering we created a dendogram in order to get an approximate value of number of clusters to use later. The dendogram shows us that there are three main clusters that are considered (as seen in Figure 15).

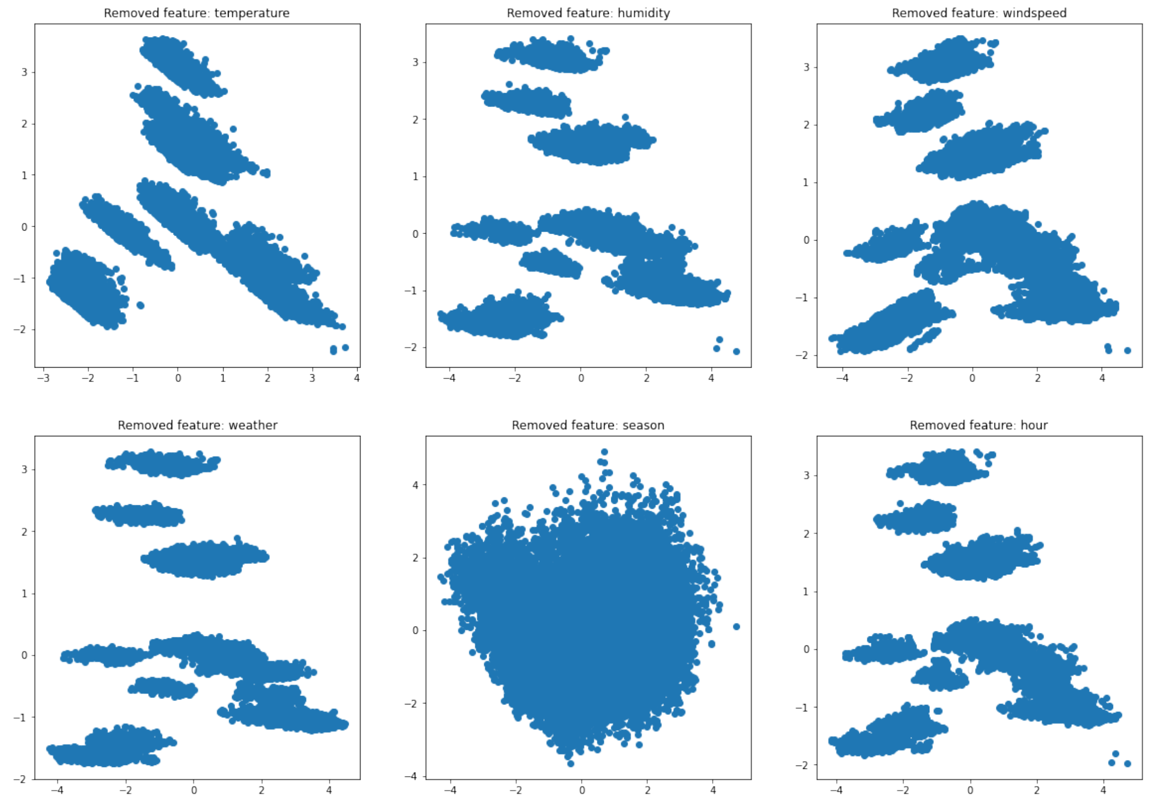


Fig. 14: PCA with feature subselection on the one hot encoded dataset

Applying then the *AgglomerativeClustering* on our label encoded dataset and projecting it on the 2D PCA transformed label encoded features, we can see that the clusters are still overlapping a lot between each other, as seen in Figure 16.

On the other hand, when applying clustering directly on the 2D PCA transformed data of the one hot encoding dataset, the clustering becomes a lot clearer and less noisy as seen in Figure 17. Again the one hot encoded dataset works better for clustering our features and for transforming them wth PCA, although there is a high computational toll for increasing our feature size.

5.8 Random Forest

Random Forest (RF) is an ensemble learning method, it is composed by a large number of decision trees and takes advantage of the fact that decision trees have low bias and a high variance, which

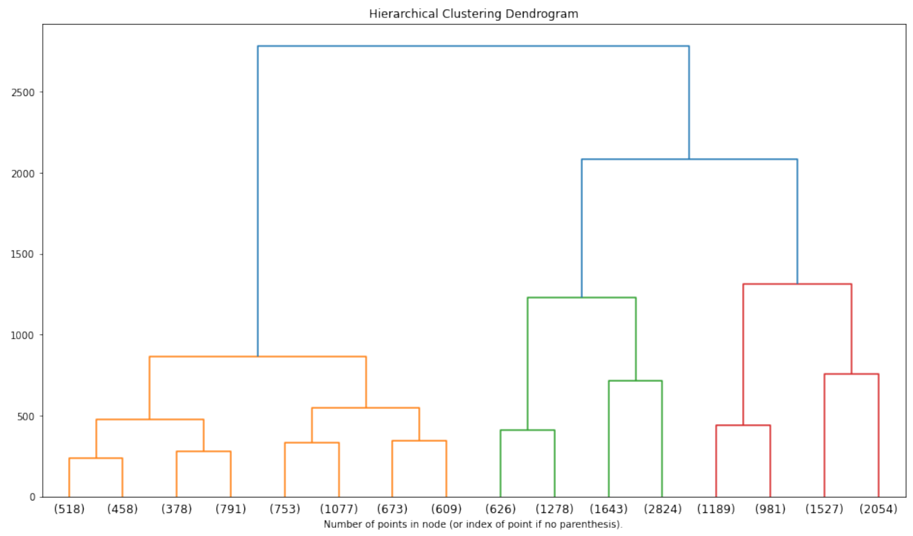


Fig. 15: Hierarchical clustering dendogram

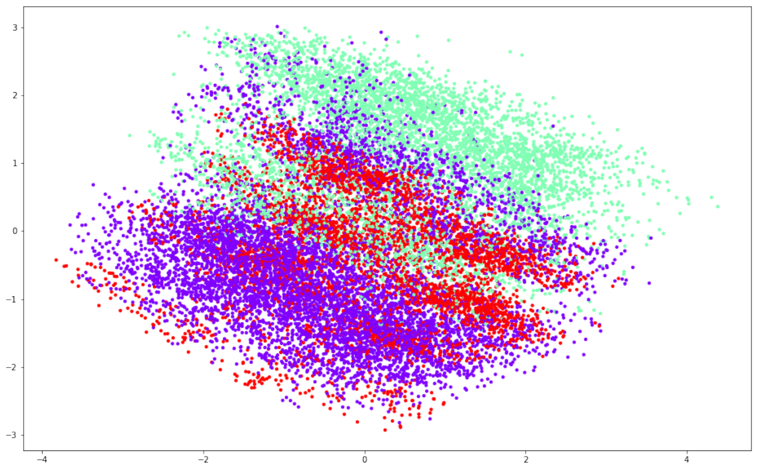


Fig. 16: Hierarchical clustering on the whole label encoded dataset, projected on the 2D PCA space.

means that they do not make any assumption about the target function, but at the same time, they are influenced by the data. Since bagging (bootstrapping aggregating) works well for learners with low bias and high variance, the decision trees represent the perfect choice for an ensemble method. When creating ensembles of learners, individual models should be uncorrelated in order to increase the error-correcting capability of the model. This is due to the fact that the low correlated trees will diverge in their individual errors, and therefore even if a small subset of the trees present an error in one direction, it can be cancelled out by the remaining ones. Trees and forests have the same bias, the actual improvement in using Random Forests that can be found is

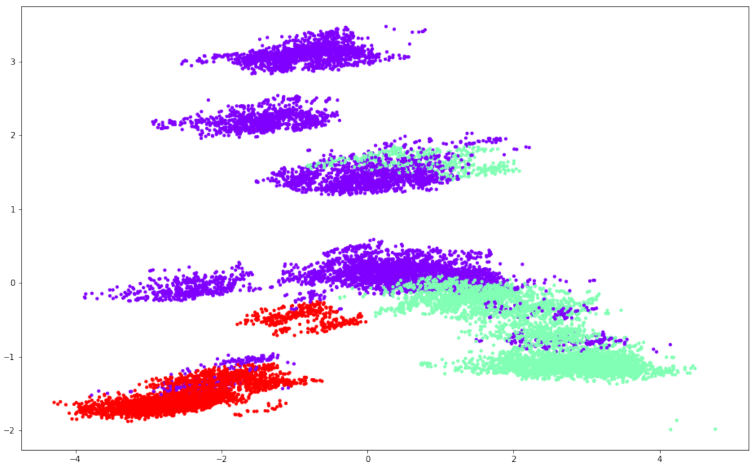


Fig. 17: Hierarchical clustering on the PCA tranformed features, projected on the 2D PCA space.

in variance reduction in a trade-off system between variance and bias in which the increase in bias is cushioned by the variance reduction. Furthermore, overfitting can be controlled by the number of trees in a RF, while diversity within the random forest is controlled by the number of features considered by each tree and by the number of samples.

Hyperparameter Tuning The parameters and ranges of the values chosen for the hyperparameter optimization of the Random Forest model [3] are aproximately the same as those of the Decision Tree precisely to be able to subsequently compare their structure. The only difference is due to the fact that the Random forest is an ensemble of individual trees and therefore it was necessary to find the optimal value of the number of estimators needed. The hyperparameter tuning in this case was performed with Random Search. When building a Random Forest Regressor the chosen parameters are crucial for the model's performance because finding a valid combination can lead to control diversity, correlation and overfitting. The number of estimators is very important because as we increase the number of decision trees the model is able to generalize better. We should always take into consideration that increasing this number means higher computational time, and we should take in mind that the gain after a certain value of decision trees does not improve. Moreover, by tuning the maximum depth the minimum number of samples required to split an internal node and the minimum number of samples required to be at a leaf node we can decrease the chance of overfitting and increase the model's R^2 score. The table below reports the parameters selected for tuning and the ranges of values in which the parameters were tested with Random Search.

Parameter	**Range**
Number of estimators	10-150
Max depth	5-200
Min samples split	0.0001-0.5
Min samples leaf	0.0001-0.5

Table 6: Ranges for the hyperparameter tuning of the Random Forest's parameters with Random Search

Implementation and Results

For the evaluation of the random forest model, the dataset was split into a train test and validation set. Furthermore, a cross validation of 5 splits was performed during the random forest. Among all the Random forest's parameters, we set the oob_score to True to then be able to compare it with the score obtained on the validation set and on the test set calculated in relation to the best estimator.

The best parameter settings for the Random Forest model are: $max\_depth : 26, min\_samples_leaf : 0.0036, min\_samples_split : 0.0116, n\_estimators : 68$

With the best parameter settings we fit a new model and calculate the predictions on validation and test set. The results obtained are reported in Table **??**tab:rf$_s cores below$.

As expected, the Out-Of-Bag given by the forest itself, produces a very similar score estimate, because the number of estimators used is the optimal one found by applying the Random Search. This is explained by the fact that the Out-of-Bag technique is used to measure the prediction error of random forests utilizing bagging, which uses subsampling with replacement to create training samples for the model. Generally, each decision tree in the random forest is trained only on the 2/3 of the data, however, if the number of estimators utilized is high enough, the model sees more data during training and hence achieves a higher score.

Score	
Out-of-bag	0.7967
R^2 validation	0.79
R^2 test	0.7968

Table 7: Comparison between the Out-of-Bag score and the R^2 score metrics on the validation and test sets.

Structure comparison with Decision Tree

The main difference between the decision tree and the random forest structure is the max depth which represents the depth of the individual tree and each tree in the forest. Generally, the deeper the tree, more information about the data is captured. Overall, the random forest performs better than the decision tree although the depth of each tree, 26, is lower than the depth, 100, of the individual one. Obviously, it is a result that we expected since the random forest combines multiple decision trees and generally is more suitable for situations when we have a large dataset, although the interpretability of it can be much harder.

5.9 Multilayer Perceptron

After all the models implemented, the only implementation missing is the Multilayer Perceptron (MLP). The difference between the latter model and the Perceptron created in Section 5.4, is that in this case, we have more than one layer. The Perceptron is very useful for classifying data sets that are linearly separable, but cannot classify dataset that are not linearly separable. The MLP, on the other hand, with its more deep and complex structure equipped with one input layer, one or more hidden layers and an output layer, is able to classify non linearly separable datasets [1]. The training of an MLP consists in adjusting weights and bias of the model in order to minimize the error, and this is done by backpropagation. For the MLP implementation in our code, we used the MLPRegressor method imported from Scikit-Learn [9].

The activation function for each node has been set to a constant valu ReLU, which is among the most popular functions in the field of Neural Networks and outputs the input directly if it is positive, otherwise, it outputs zero. The solver for weight optimization has been set to a constant value as well, without having to tune it through a hyperparameter optimization. The combination of the ReLu function with the Adam solver is one of the most famous combinations and guarantees a more than valid model performance, especially for rather large databases. Furthermore, a fraction of 0.1 was dedicated to the validation set during the training of the neural network.

Hyperparameter tuning

Among all the possible parameters that can be tuned, the parameter which it was necessary to apply a hyperparameter optimization was the *hidden_layer_sizes*, which determines not only the number of hidden layers and therefore the depth of the model, but also the number of neurons for each hidden layer. The number of hidden layers was set to two and it was not necessary to experiment with a greater number, not only because we didn't want to increase the computational time of our algorithm, but also because the results showed that the model performed satisfactorily. This means that our model is composed of 4 layers in total, considering the input and output layers. In order to find the optimal combination and reach the highest possible R^2 score, we performed a Random Search and tested several values between 2 and 200 which correspond to the number of neurons for each hidden layer.

From the results obtained from the random search we can deduct that the optimal number of neurons for the fist hidden layer is 165 and 77 for the second. The performance metric used was the R^2 score and the model reached a score on train set of 0.9359 and 0.9282 on test set.

6 Conclusion and future work

In conclusion, the analysis conducted on the dataset provides a clear idea on how to approach regression and classification problems. In general, we understood that data preprocessing has a fundamental role when trying to analyze and predict features of data. Specifically, it is noticeable how one hot encoding categorical values performs better than label encoding, although it

significantly increases the dimension of our features. On the other hand, label encoding is not great when the features have cyclical relations in their data. In that case it is better to use a cyclical encoding with sine and cosine.
Moreover, transforming the target has proven very valuable since the R^2 score of our model has increased remarkably. The reason for this is that the natural logarithm punishes high outlier values present in our target variables, by scaling them logarithmically.
In general, all the models implemented obtained satisfactory scores, with the exception of the single perceptron which was the one with the worst performance. We can conclude that the classification models despite the limited number of data in certain classes, such as the 'Heavy Rain' class of the Weather column of which only three records are present in the dataset, perform decently. Overall, the best model obtained was the Multilayer Perceptron Regressor which achieved a very high coefficient of determination on the test set. Our future work includes a deeper hyperparameter tuning for the random forest and decision tree models in order to achieve higher scores. It would be also interest to see how an MLP would perfom with the weather classification problem.

References

1. Multilayer perceptron definition, https://deepai.org/machine-learning-glossary-and-terms/multilayer-perceptron
2. sklearn.cluster.agglomerativeclustering, https://scikit-learn.org/stable/modules/generated/sklearn.cluster.AgglomerativeClustering.html
3. sklearn.ensemble.randomforestclassifier, https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html
4. sklearn.feature_selection.mutual_info_classif, https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.mutual_info_classif.html
5. sklearn.feature_selection.selectkbest, https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.SelectKBest.html
6. sklearn.linear_model.linearregression, https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html
7. sklearn.linear_model.logisticregression, https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html
8. sklearn.multiclass.onevsoneclassifier, https://scikit-learn.org/stable/modules/generated/sklearn.multiclass.OneVsOneClassifier.html
9. sklearn.neural_network.mlpregressor, https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPRegressor.html
10. sklearn.preprocessing.standardscaler, https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html
11. sklearn.svm.linearsvr, https://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVR.html
12. sklearn.tree.decisiontreeregressor https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeRegressor.html
13. Coefficient of determination (Oct 2021), https://en.wikipedia.org/wiki/Coefficient_of_determination
14. Géron, A.: Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: concepts, tools, and techniques to build intelligent systems. OReilly (2020)
15. Hayes, A.: Multicollinearity (Aug 2021), https://www.investopedia.com/terms/m/multicollinearity.asp
16. Jolliffe, I.T., Cadima, J.: Principal component analysis: a review and recent developments (2016). https://doi.org/https://doi.org/10.1098/rsta.2015.0202, https://royalsocietypublishing.org/doi/10.1098/rsta.2015.0202
17. Parashar, V.: Statistical distributions! Becoming Human: Artificial Intelligence Magazine https://becominghuman.ai/statistical-distributions-533260f370f2

18. Solutions, E.: Accuracy, precision, recall f1 score: Interpretation of performance measures (2016), https://blog.exsilio.com/all/accuracy-precision-recall-f1-score-interpretation-of-performance-measures/
19. Studio, I.W.K.: Analyzing machine-learning model performance, https://cloud.ibm.com/docs/knowledge-studio?topic=knowledge-studio-evaluate-ml