Politecnico di Torino

Mathematics in Machine Learning

Adult Income Dataset

Fabio Tatti - s282383

October 2022

1 Introduction

The Adult Income Dataset was created by the US Census Bureau in order to predict if the income of a given person is above or under 50 thousand dollars per year given some features, which is called a binary classification task. The dataset has 48842 instances, with 6 continuous, 8 nominal attributes. In our case we will proceed by first describing the dataset itself, then we will explore the data in order to understand better how to properly use it. We will clean the data and then prepare it before testing various models given the chosen metrics. The dataset is available to the public at the link: `https://www.kaggle.com/datasets/wenruliu/adult-income-dataset` and also: `https://archive.ics.uci.edu/ml/datasets/Adult/`.

	age	workclass	fnlwgt	education	educational-num	marital-status	occupation	relationship	race	gender	capital-gain	capital-loss	hours-per-week	native-country	income
0	25	Private	226802	11th	7	Never-married	Machine-op-inspct	Own-child	Black	Male	0	0	40	United-States	<=50K
1	38	Private	89814	HS-grad	9	Married-civ-spouse	Farming-fishing	Husband	White	Male	0	0	50	United-States	<=50K
2	28	Local-gov	336951	Assoc-acdm	12	Married-civ-spouse	Protective-serv	Husband	White	Male	0	0	40	United-States	>50K
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
48839	58	Private	151910	HS-grad	9	Widowed	Adm-clerical	Unmarried	White	Female	0	0	40	United-States	<=50K
48840	22	Private	201490	HS-grad	9	Never-married	Adm-clerical	Own-child	White	Male	0	0	20	United-States	<=50K
48841	52	Self-emp-inc	287927	HS-grad	9	Married-civ-spouse	Exec-managerial	Wife	White	Female	15024	0	40	United-States	>50K

Figure 1: Sample of the Adult Income Dataset

2 Dataset Description

The dataset contains 2 different classes for the target feature *Income*: **>50K** and **<=50K**. The other features are:

- *age*: integer values
- *workclass*: categorical class with 8 unique possible values
- *fnlwgt*: integer values
- *education*: categorical class with 16 unique possible values
- *educational-num*: continuous integer values describing the previous feature
- *marital-status*: categorical class with 7 unique possible values
- *occupation*: categorical class with 14 unique possible values
- *relationship*: categorical class with 6 unique possible values
- *race*: categorical class with 5 unique possible values
- *sex*: categorical class with 2 unique possible values
- *capital-gain*: integer values
- *capital-loss*: integer values
- *hours-per-week*: integer values
- *native-country*: categorical class with 41 unique possible values

We already know of the presence of *null* values represented as *?* character.

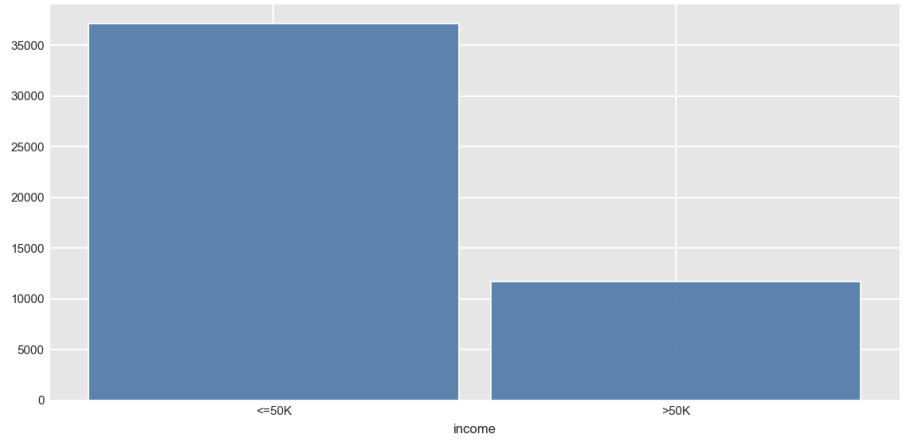

Figure 2: Balance of the *Income* class

3 Data Exploration and Processing

3.1 Null Values

We first clean the data by handling the null values. This problem affects the 7% of the data: we will handle it by substituting the missing value with the most frequent value of the given feature. This will help us, respect to drop the missing value's row, to preserve the quantity of data with a compromise on the quality. By performing it on the dataset we know that this problem was affecting only the *workclass*, *occupation* and *native-country* column features of the dataset.

3.2 Balance of classes

We will explore the balance of the target variable. As we can notice in Figure 2, the *Income* class is unbalanced. This represents a bias in our database that can affect the model's behaviour, but also is due to the reality of the world from where the samples are taken: the majority of the people do not have an annual salary which is above 50 thousand. The fact that the $>50K$ class is almost one third of the $<50K$ is the proof that we have an unbalanced dataset. This issue will be handled in our pipeline, since some of the models can be sensitive to the unbalance of the target class. We also noticed that for the feature *native-country* we do have a distribution with a main value that is *United-States*, just followed by *Mexico*, which is really smaller than the first one. The rest of the values have a cardinality close to 0: for this reason we do aggregate all those values under a new value *Not-US-MX*.

3.3 Correlation

We exploit the correlation in such a way to understand the linear relationship among two different features. In case of an high level of correlation it means that two features are carrying similar information: the higher the correlation, the more similar the behaviour of the two features will be. A positive correlation among A and B means that if feature A increases also feature B will do it. Vice versa in case of a negative one. To better investigate the data we will create a correlation map, as the one showed in Figure 3, which is based on the Pearson Correlation coefficient between two random variables (or vectors)

$$\rho = \frac{Cov(X,Y)}{\sqrt{Var(X) * Var(Y)}} \tag{1}$$

with $-1 \leq \rho \leq 1$ and X and Y as random vectors. The $Cov(X,Y)$ is the covariance among X and Y, while the $Var(X)$ is the variance of the random variable X. Since is a symmetric matrix, the diagonal has only the maximum value 1, because of course for every random variable we get in the equation 1 that the denominator is the same as the nominator. As we can notice in the Figure 3, some features do seem to affect the target class *Income* more than others. All of them, such as *age*, *educational-num* or *hours-per-week* do relate to it with a weakly positive correlation. Not only, we can also see that some variables do relate each other with even more strong relationships, such as the case of *education* and *educational-num*, which reflects the fact that the two variables express the same feature. For that reason we decide to drop the categorical variable *education*. This comparison was possible since the concept of education has indeed and order that not all the categorical features do have. That is why in the correlation the only categorical feature present is the education, which was firstly factorized in order to get a number for each different value of the feature.

4 Data Preparation

We now proceed with the preparation of the data for feeding then the various Machine Learning models.

4.1 Hash Encoding

Since we are dealing with a lot of categorical values we now proceed by exploiting an Hash Encoder, given the fact that the encoding with a label one would introduce an order that is biased since not present in the reality. Not only, by means of the One Hot Encoding we would get an explosion of the dimensionality of the data. The Hashing procedure is the transformation of arbitrary size input into the form of a fixed-size value. Hashing is a one-way process, meaning that we cannot generate back the input from the hash representation. Like the One-Hot Encoding the Hash Encoder will represent categorical features by means of

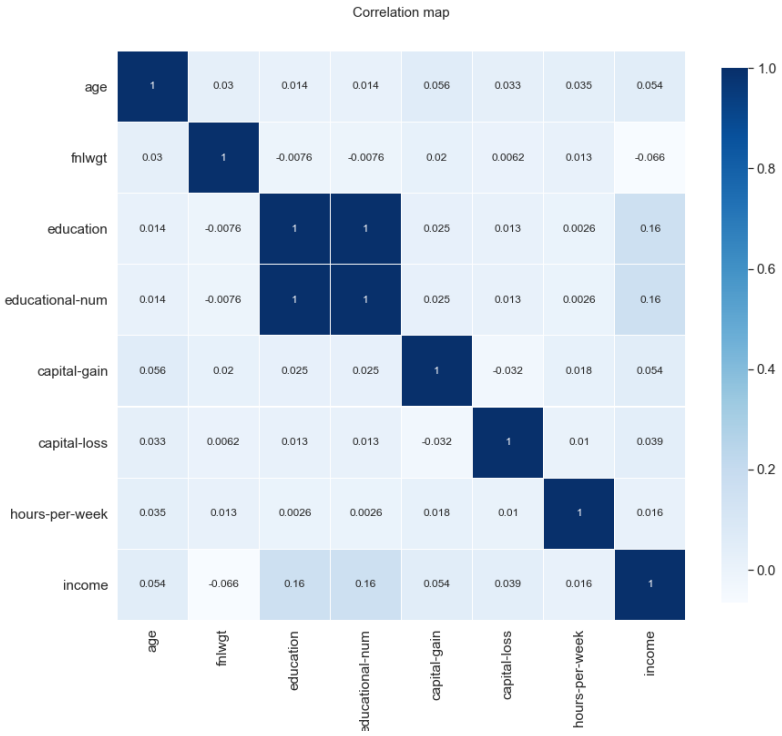

Figure 3: Correlation map of the Adult Income Dataset

new dimensions. We can also set a number for those: we set it to 32, meaning that we are going to use 32 new features to describe the already present data. This number was specifically suggested for features with high number of unique values in the documentation of the Python package providing the encoder. We decided to leverage on this Encoder respect the One-Hot Encoder in order to avoid the explosion of the features, since the creation of a column for each unique value of the various categorical values would result in a way bigger matrix respect the one that we are currently dealing with.

4.2 Standardization

Many machine learning algorithms perform better when numerical input variables are scaled to a standard range. Algorithms which are using weighted sum of the input, like linear regression or using distance measures, like k-nearest neighbors or SVM can be example to this situation. Standardization (or z-score normalization) is a scaling technique where the values are centered around the mean with a unit standard deviation. This means that the mean of the attribute becomes zero and the resultant distribution has a unit standard deviation. The formula for the standardization is:

$$X_{standardized} = \frac{X - \mu}{\sigma} \tag{2}$$

where μ is the mean value of the given feature and σ is its standard deviation. During this procedure the feature values are rescaled so that they have the properties of a standard normal distribution with $\mu = 0$ and $\sigma = 1$. Also, this step is important for applying PCA, because it calculates a new projection of the dataset, and the new axis are based on the standard deviation of the variables. So, a variable with a high standard deviation will have a higher weight for calculation of axis than a variable with a low standard deviation. If we normalize the data, all variables have the same standard deviation, thus all variables have the same weight and PCA calculates relevant axis. The two most popular technique for scaling numerical data prior to modeling are normalization and standardization.

4.3 Principal Component Analysis - PCA

In some cases, we need to implement dimensionality reduction. Dimensionality reduction removes redundant or highly correlated features, also reduces the noise in the data.All of those contribute to the interpretability of the model. One of the main and most used examples of dimensionality reduction techniques is Principal Component Analysis, which is generally referred shortly, as PCA. The objective of Principal Component Analysis is to reduce the dimensionality of the data while preserving the maximal possible variance. Principal components are vectors that define a new coordinate system in which the first axis goes in the direction of the highest variance in the data. The second axis is orthogonal to the first one and goes in the direction of the second highest variance. If our

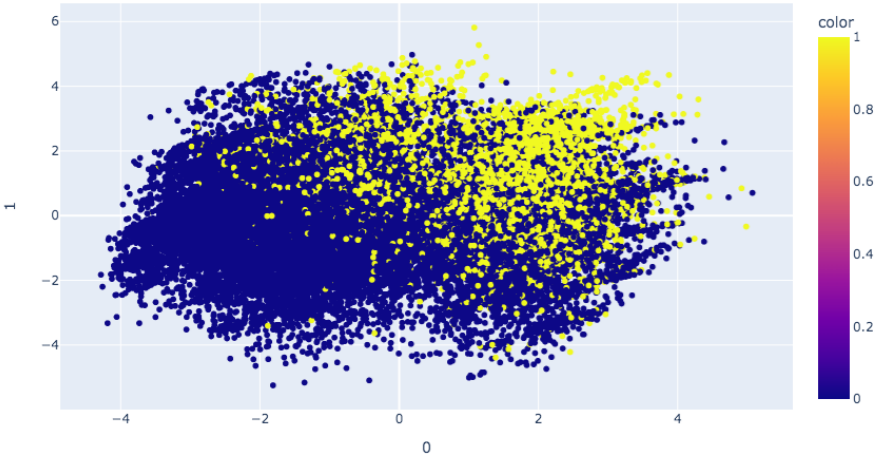

Figure 4: First two Principal Components of the dataset

data is three-dimensional, the third axis will be orthogonal to both the first and the second axes and go in the direction of the third highest variance, and so on. In Figure 4, we see example of two-dimensional dataset. The PCA can be seen as an optimization problem where

$$argmin \sum_{i=1}^{m} ||x_i - UWx_i||_2^2 \tag{3}$$

$$s.t. W \in \mathbb{R}^{n,d}, U \in \mathbb{R}^{d,n}$$

The two matrices W and U do act like a compression/encoding matrix and as decoding, rispectively. The aim is to minimize the difference between the output and the input one for each sample in the data. In this example the PCA behaves as an autoencoder, that tries to compress down the size of the data and to then reconstruct after, by minimizing the difference among the original and reconstructed.

The PCA allow us also to visualize, as showed in Figure 5, the cumulative Explained Variance of the principal components: this will help us in order to choose how many components select in order to still explain the bigger amount of the variance of the data, while decreasing the size of the input. From the graph we can see that with 32 principal components we can describe more than 99% of the whole variance of the data, allowing us also to remove the noise from it.

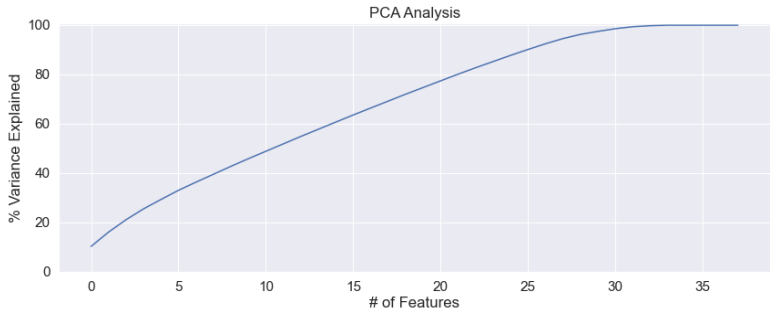

Figure 5: Explained Variance given the cumulative principal components

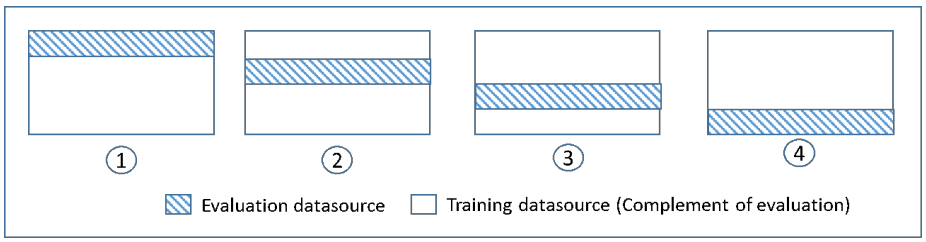

Figure 6: Cross Validation

5 Settings and Metrics

5.1 Cross Validation

Cross-validation is a statistical method used to estimate the skill of machine learning models. It is commonly used in applied machine learning to compare and select a model for a given predictive modeling problem. The method is developed to increase the security of classification in which we randomly split our training set into several subsets (k folds) of the same size. As we can see in Figure 6 we take the first fold as validation set and the rest folds as training set. Then we continue this process with second fold as validation and the rest as training set, etc. This is done until the last fold is used as validation and the rest as training set. At the end, we average the values of score to get the final value. Unlike Hold-out method which only splits dataset into training and test set, cross validation is more computationally time demanding task but at the same time less dependent on test set like its counterpart. So, this makes it more robust to overfitting problem with respect to Hold-out method. In our case we did choose to use $K = 5$ folds for the input data.

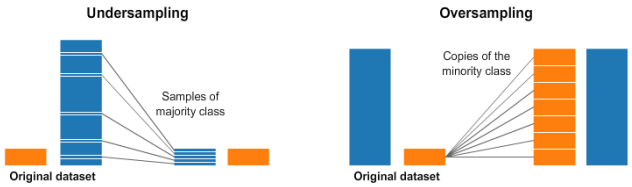

Figure 7: Differences among undersampling and oversampling

5.2 SMOTE

We mentioned before that our dataset is unbalanced: to overcome this issue we will exploit the Synthetic Minority Oversampling Technique (SMOTE). This technique handles the imbalanced dataset by oversampling the minority class, in our case $>50K$. The technique belongs to the data augmentation class, that synthetizes new examples for the model, more proficient respect the more simple techniques that just duplicate the minority class. SMOTE works by selecting examples that are close in the feature space, drawing a line between the examples in the feature space and drawing a new sample at a point along that line. Specifically, a random example from the minority class is first chosen. Then k of the nearest neighbors for that example are found (typically k=5). A randomly selected neighbor is chosen and a synthetic example is created at a randomly selected point between the two examples in feature space. Since our dataset is small we chose to use an oversampling technique in order to increase the size of the minority class rather then decrease the majority as shown in Figure 7.

5.3 Metrics

In order to assess the performance of the various models on the test partition of the dataset we need to define different metrics. We use some of the most widely used metrics and tools such as Confusion matrix, Accuracy, Recall, F1 score etc. to understand what is going on generally.

- **Confusion Matrix**: showed in Figure 8, a table that summarizes how successful the classification model is at predicting examples belonging to various classes. One axis of the confusion matrix is the label that the model predicted, the other axis is the actual label. All the other metrics we use are based on this matrix.

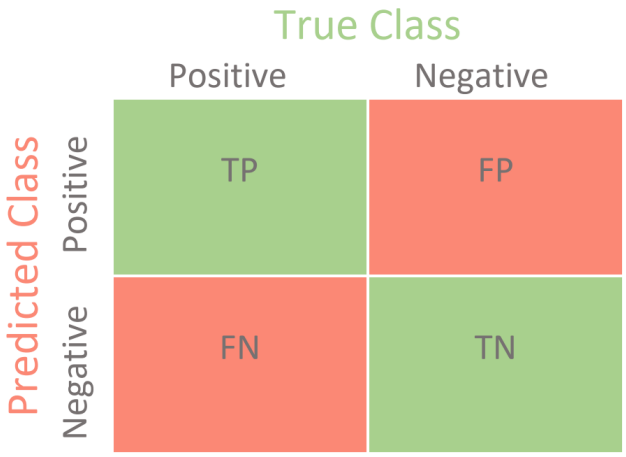

Figure 8: Confusion matrix

  - TP: True Positive. Both predicted and actual values are positive.
  - FN: False Negative. Predicted value is negative, actual value is positive.
  - FP: False Positive. Predicted value is positive, actual value is negative.
  - TN: True Negative. Both predicted and actual values are negative.

- **Accuracy**: given by the number of correctly classified examples divided by the total number of classified examples.

$$a = \frac{TP + TN}{TP + FP + FN + TN} \tag{4}$$

- **Precision**: the ratio of correct positive predicted values respect to the total number of predicted positive.

$$p = \frac{TP}{TP + FP} \tag{5}$$

- **Recall**: the ratio of correct positive predictions to the overall number of positive examples in the dataset.

$$r = \frac{TP}{TP + FN} \tag{6}$$

- **F1 score**: harmonic mean of Precision and Recall.

$$F1 = \frac{2rp}{r + p} \tag{7}$$

6 Model Selection

6.1 Perceptron

The perceptron is one of the most simple linear binary classifiers, it works, as shown in Figure 9, in three main steps:

- All the inputs x are multiplied with their weights w. Let's call it k.
- Add all the multiplied values and call them Weighted Sum.
- Apply that weighted sum to the correct Activation Function.

One of the main assumptions, but also drawbacks, that this algorithm works on is that the data should be linearly separable, that is, there should exist a linear classifier that separates the data with zero training error. The perceptron is a single layer classifier, one of the most simple ANN, with a forward stage and a backward one. We did perform the Cross Validation Gridsearch in order to find the best performing set of parameters for our model, the best performing ones are:

- 'alpha': 0.0002 . Constant that multiplies the regularization term if regularization is used.
- 'max_iter': 1400 . The maximum number of epochs over the training data.
- 'penalty': 'l1' . Regularization term to be used.
- 'random_state': 42
- 'shuffle': True . Whether or not the training data should be shuffled after each epoch.
- 'warm_start': False . When set to True, reuse the solution of the previous call to fit as initialization, otherwise, just erase the previous solution.

In Figure 10 we can see the Confusion matrix of the model tuned with the best performing parameters over the test set.Through confusion matrix we can observe the number of correct and incorrect predictions which are summarized with count values and broken down by each class. In the table 1 we can see the performances expressed in the previously mentioned metrics.

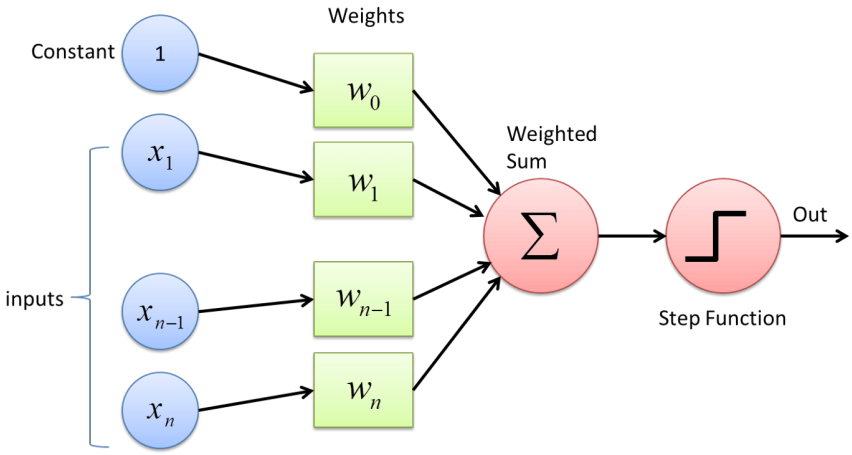

Figure 9: Perceptron scheme

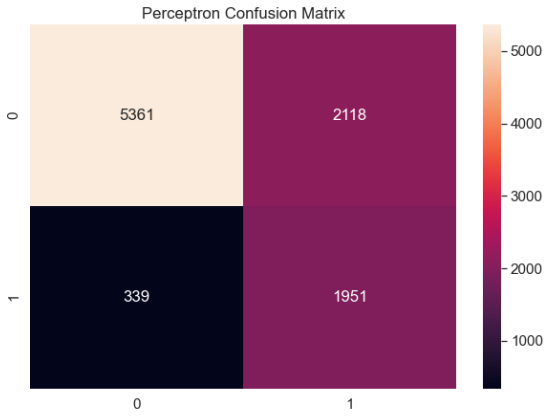

Figure 10: Confusion Matrix of the Perceptron

Perceptron Results	
Accuracy	0.74849
Precision	0.47948
Recall	0.85196
F1 Score	0.61362

Table 1: Perceptron results

6.2 Linear Support Vector Machines

The objective of the Support Vector Machine algorithm is to find a hyperplane in an N-dimensional space (where N is the number of features) that distinctly classifies the data points. To separate two classes of data point, there are many possible hyperplanes that can be chosen as it can be seen in Figure 11. Our objective is to find a plane that has the maximum margin. A margin of a hyperplane with respect to a training set is defined to be the minimal distance between a point in the Training Set and the hyperplane itself. Maximizing the margin distance provides some reinforcement so that future data points can be classified with more confidence. A characteristic of the SVM is to exploit the Kernel Trick, in order to better divide data that result to be not linearly separable in the original domain. Since map the data into a higher dimensional space would result computationally expensive we need a way to operate directly on the original feature space. We did use the GridSearch in order to find the best performing parameters, which resulted to be:

- 'C': 1. It is regularization parameters
- 'degree': 2 . Degree of the polynomial kernel function ('poly'). Ignored by all other kernels.
- 'gamma': 'auto' . Kernel coefficient for 'rbf', 'poly' and 'sigmoid'.If 'auto', uses $\frac{1}{n_{features}}$.
- 'kernel': 'rbf'. Specifies the kernel type to be used in the algorithm.
- 'random_state': 42

The best performing kernel is the *rbf*, the radial basis function kernel, also called Gaussian because of the similarity with it. The function of the kernel is

$$K(X_1, X_2) = exp(-\frac{||X_1 - X_2||^2}{2\sigma^2}) \tag{8}$$

where σ is our hyperparameter and variance, and the numerator $||X_1 - X_2||$ is the Euclidean Distance, or L-2 Norm, among two datapoints. The maximum value that the RBF kernel can be is 1 and occurs when distance is 0 which is when the points are the same, i.e. $X_1 = X_2$. In the Figure 12 we can see the Confusion Matrix of the SVM model trained with the best performing parameters and performing on the test set. In the table 2 we can see the numerical results expressed in the metrics.

6.3 Logistic Regression

The logistic regression takes the name from the logistic function, showed in Figure 13, at the core of the algorithm.

$$logistic(x) = \frac{1}{1 + exp(-x)} \tag{9}$$

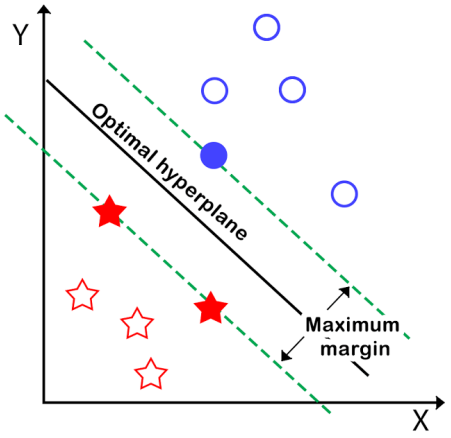

Figure 11: Concept of SVM

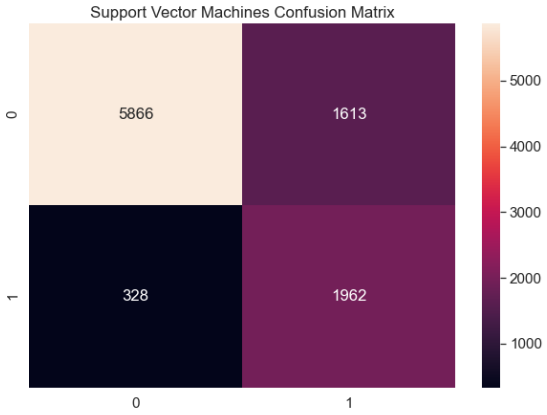

Figure 12: Confusion Matrix of the SVM

SVM Results	
Accuracy	0.80131
Precision	0.55552
Recall	0.85677
F1 Score	0.54881

Table 2: Linear Support Vector Machines results

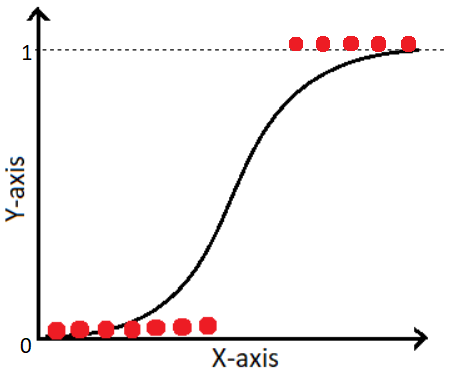

Figure 13: Logistic Function

Logistic Regression Results	
Accuracy	0.80111
Precision	0.55013
Recall	0.83144
F1 Score	0.66215

Table 3: Logistic Regression results

Similarly to the linear regression, we want to use it to assess the probability of an event occurring, in our case if a person earns more money than a given value. Since the result of this process is a probability it will be bounded between 0 and 1.

The best performing hyperparameters given by the GridSearch are:

- 'C': 1.5 . Inverse of regularization strength.
- 'penalty': 'l2' . Specify the norm of the penalty.
- 'random_state': 42
- 'solver': 'lbfgs' . Algorithm to use in the optimization problem.

The Confusion Matrix is showed in Figure 14 and the table with the result is the 3.

6.4 Random Forest

The Random Forest is an ensemble learning method used for the classification that leverages on a multitude of decision trees to perform the task. An ensemble method is a technique that combines the predictions from multiple machine learning algorithms together to make more accurate predictions than any individual model. A simplified example is showed in Figure 15. In the case of

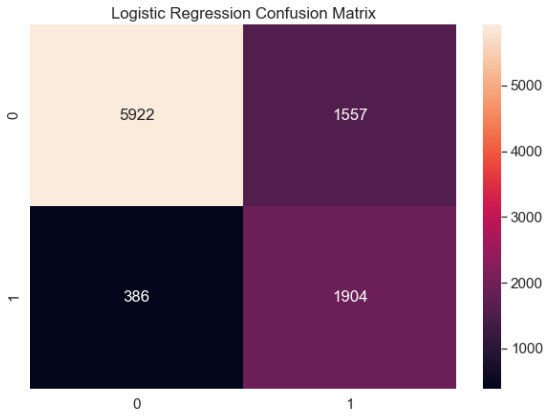

Figure 14: Confusion Matrix of the Logistic Regression

the classification the procedure collects the votes of each decision tree; then the majority voted class is assigned to the new data point. A strong point respect to a single decision tree is that the random forest averages over different tree each one trained over a different part of the data, with the goal of reducing the variance. Random Forest is based on the idea of Bagging (Bootstrap Aggregation) which consists of creating many "copies" of training data (each copy is slightly different from another) and then apply the weak learner (decision tree in our case) to each copy to obtain multiple weak models and then combine them. Single decision trees are sensitive to the specific data on which they are trained. If the training data is changed (e.g. a tree is trained on a subset of the training data) the resulting decision tree can be quite different and in turn the predictions can be quite different. Bagging is the application of the Bootstrap procedure to a high-variance machine learning algorithm. When bagging with decision trees, we are less concerned about individual trees overfitting the training data. For this reason and for efficiency, the individual decision trees are grown deep. In our code we did implement the model provided by the sklearn package for Python which allow us to set the number of trees, the criterion used to measure the quality of a split in the tree, the maximum number of features considered when looking for the best split and the minimum number of samples required to split an internal node. The best performing parameters are the following:

- 'criterion': 'gini' . The function to measure the quality of a split.
- 'max_features': 'sqrt'. The number of features to consider when looking for the best split, in this case $sqrt(n_{features})$.
- 'min_samples_split': 3 . The minimum number of samples required to split

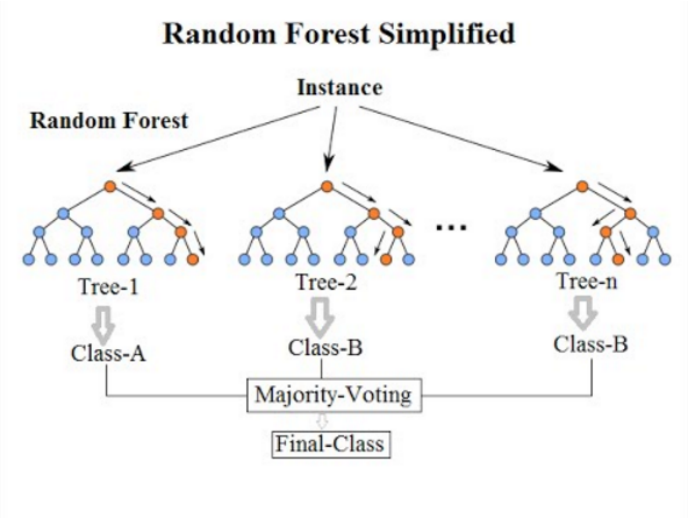

Figure 15: Conceptual example of the Random Forest

Random Forest Results	
Accuracy	0.82926
Precision	0.61727
Recall	0.71485
F1 Score	0.66248

Table 4: Random Forest results

an internal node.

- 'n_estimators': 200 . The number of trees in the forest.
- 'random_state': 42

We will just explain the Gini criterion, used to create the tree by calculating the impurity of the split.

$$Gini(t) = 1 - \sum_{i=1}^{j} P(i|t)^2 \tag{10}$$

Where the j is the number of classes in the label, and the P represents the ratio of class at the i^{th} node. The results of the best performing model over the test set are showed in the Confusion Matrix in the Figure 16. The metrics are expressed in the Table 4.

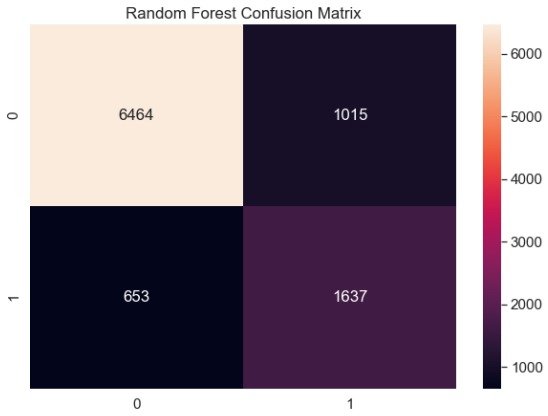

Figure 16: Confusion Matrix of the Random Forest

6.5 K-Nearest Neighbor

One of the most popular classifier that relies on distance to perform the classification task: it is a clustering unsupervised technique. We want to group different data points al together in order to determine their class. In this case the algorithm retains all the training point in order to use them for comparison respect to the unseen ones by means of the distance. This in order to assess the similarity between two different data point: the closer they are the more similar are. In our case the classifier provided by sklearn can have different ways to compute the distance among two data points, such as *ball_tree* or *kd_tree*. The *auto* value will attempt to decide automatically which is the most appropriate algorithm based on the features of the input data points in the training phase. As showed in Figure 17, when computing the distance in order to classify the unseen data point, the algorithm will select the K closest data points to the unseen one in order to classify it. Then we can use different weights to assess the importance of each neighbor in the voting process for the classification task, which leverages on the distance: the closer the neighbor is, the more important can be its vote. But we can also use uniform values in the neighborhood, which gives the same importance to all the neighbors, no matter the distance. The algorithm is sensible to the size of the input data: this is an effect of the curse of dimensionality, because the higher the number of dimensions the harder it is to define the concept of distance among two point into that space. The best performing parameters after the GridSearch are :

- 'algorithm': 'ball_tree' . Algorithm used to compute the nearest neighbors.
- 'n_neighbors': 4 . Number of neighbors to use by default for kneighbors queries.

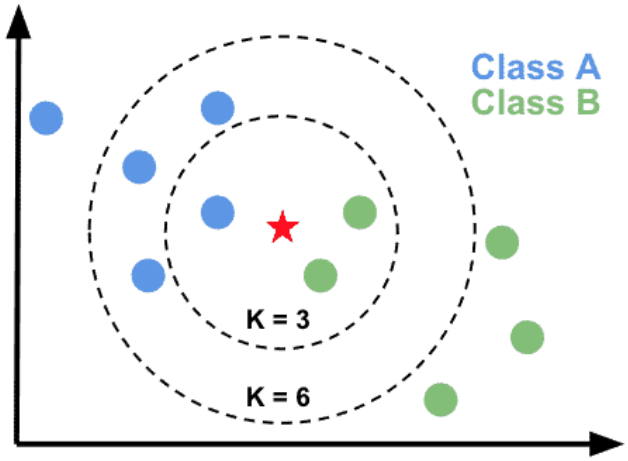

Figure 17: K-Nearest Neighbors visual example

KNN Results	
Accuracy	0.80387
Precision	0.56775
Recall	0.68428
F1 Score	0.62059

Table 5: K-Nearest Neighbors results

- 'weights': 'uniform' . Weight function used in prediction: 'uniform', all points in each neighborhood are weighted equally.

The results of the classifiers are showed in the Confusion Matrix of Figure 18 and the metrics are expressed in the Table 5.

7 Conclusion

As we can see in the summary Table 6, not all the classifiers did perform really well in the classification task. The worst result is given by the Perceptron, but was not surprise, due to the simple structure of this model. The rest of the models perform with a similar level of accuracy; although, the best performing one is the Random Forest, with the higher accuracy, precision and F1 score. On the other hand the best recall belongs to the SVM, which still has comparable accuracy respect to the Random Forest. Considering the worst one, the Perceptron, this algorithm does not take into consideration the quality of the hyperplane and converges to the first one it finds. This is a problem solved by the SVM, which indeed performs better than the other.

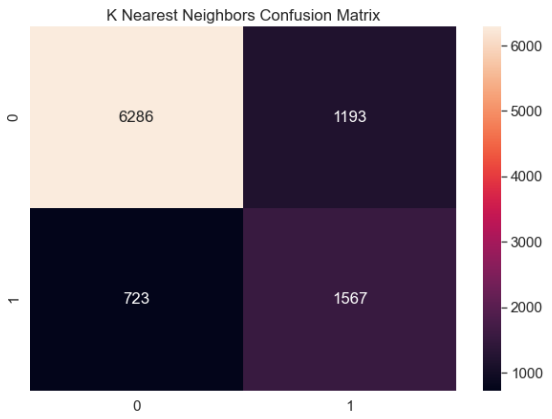

Figure 18: Confusion Matrix of the K-Nearest Neighbors

Model	Accuracy	Precision	Recall	F1 Score
Perceptron	0.74849	0.47948	0.85196	0.61362
SVM	0.80131	0.55552	**0.85677**	0.54881
Logistic Regression	0.80111	0.55013	0.83114	0.66215
Random Forest	**0.82926**	**0.61727**	0.71485	**0.66248**
KNN	0.80387	0.56775	0.68428	0.62059

Table 6: Summary results