# **Project Report for**

# **Machine Learning Group Project**

# 

## **Master’s in Data Science and Advanced Analytics at NOVA IMS, Lisbon**

## **Group Information**

Group:

MAA\_202021\_50

Group members:

Md. Shawkatul Islam Aziz

Md. Tahir Hossain

Philipp Metzger

Ali Sabbir

Henrique Vaz

Link to GitHub repository:

<https://github.com/ph1001/Group-Project-Machine-Learning-Group-50>

**Abstract**

((Summarise the introduction))

This project’s goal is to implement machine learning methods and algorithms for classifying individuals as having an income below or above average. For this, with a training dataset that was provided, multiple methods are implemented and tested.

((Summarise the methodology))

In order to complete the task, an exploratory analysis of the training dataset is conducted, several transformations are applied to certain variables, the existence of missing values is assessed, the discriminatory power of categorical features is analysed, the categorical variables are encoded, an outlier detection is performed, the features are scaled, and an appropriate subset of the resulting features is selected for further processing. Then, several machine learning algorithms are tested for their fitness for this particular application. This is done by first finding optimal or near optimal parameter values by using Grid Search and then comparing the algorithms using these best parameters to each other. In the end, a small number of algorithms is selected and combined in an ensemble classifier. After comparing the performance of the ensemble classifier using three different classifiers types, a support vector classifier is selected as the meta classifier.

((Summarise the Results))

((Summarise the Conclusions))

**I. Introduction**

The goal of this project is to implement Machine Learning algorithms that given the suitable input data are capable of predicting if an individual has an income lower or higher than the average income in a group of citizens.

For this, a dataset of 22400 observations serves as training data. This dataset includes amongst others the variables ‘Birthday’, ‘Marital Status’ and ‘Education Level’ and the binary target variable ‘Income’. A full list of the variables contained in the dataset will be presented in Chapter III. The input data and target vector are used to train several predictive models of which one is chosen in the end to be the best suited model for the task.

The project is related to the Kaggle competition ‘Newland’ in which several groups of students of the course Data Science and Advanced Analytics at the Lisbon based university NOVA IMS compete in a contest that is embedded in the fictitious scenario of the colonisation of a newly discovered habitable planet. In this Kaggle competition, each group uploads a vector of predictions computed with their best model, based on the input data of a test dataset provided in the project materials. The vector of predictions serves as the quality measure to assess which group designed and implemented the best predictive model.

**II. Background**

In this chapter, the theoretical background of the techniques or algorithms who have not been explored during the practical classes but are applied in this project are explained.

Support Vector Classification:

“Support vector machine” (SVM) belongs to the supervised machine learning algorithms. It can be used for regression and classification. The latter is the case in this project. Thus, the algorithm used here is called support vector classifier. The goal of support vector machines is to define a hyperplane in the n-dimensional space containing the training data points[[1]](#footnote-1), that separates the different classes in the target variable[[2]](#footnote-2). In case of a linearly separable dataset, meaning that a hyperplane can be defined such that if all observations from one class are on one side and all observations from the other class are on the other side of the hyperplane, the hyperplane simply represents the border that separates the two classes and that has the maximum distance to each and every point in the dataset (measured individually), which means that only the distances to the points on each side that are closest to the hyperplane are relevant. These closest points are referred to as the “support vectors” since they support the position of the hyperplane. Because the area around the hyperplane could be imagined as a channel or street separating the dataset, this is also called the “widest street approach”. In case of a non-linearly separable dataset, a penalty can be added during the calculation of the optimal solution, such that points that are “on the wrong side” of the hyperplane are penalized in a way that is adequate for the application. This is called the “Soft Margin” approach (see chapter 7.1.1 in [3]). In applications, where the dataset contains patterns where a hyperplane is not able to create a satisfactory division, such as when a ring of points of one class surrounds a cluster of points of the other class, the n-dimensional space of the datapoints can be mapped to a higher-dimensional space in order to achieve a better separability. This is called the “Kernel Trick” (see chapter 6 in [3]).

Random Oversampling:

When one group of a categorical or binary target variable has more observations than the others (in the categorical case) or the other (in the binary case), it can be sensible to use random oversampling to harmonise the number of observations associated to each category of the variable. In random oversampling this is achieved by duplicating observations that are associated to the category or the categories that originally have less observations associated to them. The observations to duplicate are selected randomly and with repetition. The goal of random oversampling can be that after the process, all categories have the same number of observations associated with them. Or in the binary case, it is also be an option to define a factor, by which the numbers of observations of the two classes differ from each other such that n\_small = a \* n\_large, ∈ IR, a ∈ (0,1], where n\_small is the number of observations associated with the class that originally had less observations and n\_large is the number of observations associated with the other class. Random oversampling is described in Chapter 5 of [1].

**III. Methodology** ((check the order of everything in the notebook))

**III.1 Materials and Software**

The materials used in this project are the training dataset and the test dataset provided. The original training dataset consists of 22400 observations and 15 variables (including the target variable). These variables are shown in Table 1, which is originated in the PDF file ‘Project Presentation.pdf’ which serves for defining the fictitious background and the goal of the project and the kaggle competition.

|  |  |
| --- | --- |
| **Variable** | **Description** |
| Citizen\_ID | Unique identifier of the citizen |
| Name | Name of the citizen (First name and surname) |
| Birthday | The date of Birth |
| Native Continent | Home continent of the citizen on planet Earth |
| Marital Status | The marital status of the citizen |
| Lives with | The household environment of the citizen |
| Base Area | The neighborhood of the citizen in Newland |
| Education Level | The education level of the citizen |
| Years of Education | The number of years of education of the citizen |
| Employment Sector | The employment sector of the citizen |
| Role | The job role of the citizen |
| Working Hours per week | The number of working hours per week of the citizen |
| Money Received | The money payed to the elements of Group B |
| Ticket Price | The money received by the elements of Group C |
| Income | The dependent variable (Where 1 is Income higher than the average and 0 Income Lower or equal to the average) |

Table : The original dataset’s variables adopted from “Project Presentation.pdf”

The software that is used in order to complete the project is Python and more precisely Anaconda and Jupyter Notebook. In the latter, the code for this project is created.

In the following part, the steps conducted in our Jupyter notebook are described.

**III.2 Loading of Data, Data Exploration and Data Pre-processing**

First, all packages and libraries that are used are imported. These include, amongst others, libraries from the standard Python library such as ‘os’ and packages from ‘sklearn’ which are used for the predictive models but also the library ‘xgboost’ which we used for feature selection, but not for classification.

In the second step the data described above is loaded into main memory and a first data exploration is conducted through which it becomes clear, that the training dataset contains more observations with the target variable ‘Income’ equal to 0 than observations with the target variable ‘Income’ equal to 1, meaning that in the training dataset, there are more observations that are classified as having an income lower than the average.[[3]](#footnote-3) This serves as the motivation for applying Random Oversampling in our application.

In the following step the variable ‘Birthday’ is transformed to obtain the age of each individual. As the unit for this newly created variable *days* is chosen, and the old variable is replaced by the new one.

In the next step, the existence of cells containing empty strings, spaces or NaN values[[4]](#footnote-4) is assessed.

In a further exploratory analysis, the discriminatory power of the categorical features in the training dataset is assessed by plotting bar charts for each category in which the affiliation the either target value is represented in either grey or green.

The following step’s purpose is to one-hot encode most of these categorical features in order to make it possible to process them in the algorithms used later on. Only the feature ‘Education Level’ is not one hot encoded. For this one, a different encoding method is designed and applied, which assigns a numerical rank to each original value of this variable.

Next, outlier detection is performed. For this, the minima and maxima of features that could potentially have outliers are computed and an assessment is made on whether or not these values are realistic. The result of this assessment is described in chapter IV.

The following step is the procedure of feature scaling. For this, a standard scaler is used, which standardises every feature by removing its mean and scaling it to unit variance [2].

**III.3 Feature Selection**

Next, as a first feature selection step, correlations between the metric variables (and the target variable ‘Income’) as well as between all variables (and the target variable ‘Income’) are assessed to decide, which features are contributing only redundant information to the dataset. The outcome of this analysis is described in more detail in chapter IV; for now, it is relevant to know that some features are discarded and a list of features ‘features\_to\_keep\_1’ is defined, which contains the features that are kept after the correlation analysis. The variables of the resulting training dataset are presented in Table 2.

|  |  |
| --- | --- |
| **Variable** | **Description** |
| Age\_days\_rel\_to\_2020 | Age of the citizen relative to the most recent day the code has been run |
| Marital\_Status\_Married | 1: Citizen is married, 0 otherwise |
| Marital\_Status\_Single | 1: Citizen is single, 0 otherwise |
| Marital\_Status\_Divorced | 1: Citizen is divorced, 0 otherwise |
| Lives\_with\_Children | 1: Citizen lives with children, 0 otherwise |
| Lives\_with\_Husband | 1: Citizen lives with husband, 0 otherwise |
| Lives\_with\_Alone | 1: Citizen lives alone, 0 otherwise |
| Lives\_with\_Other Family | 1: Citizen lives with other family, 0 otherwise |
| Role\_Management | 1: Citizen’s role is in management, 0 otherwise |
| Role\_Other services | 1: Citizen’s role is in “other services”, 0 otherwise |
| Role\_? | 1: Citizen’s role is unknown, 0 otherwise |
| Role\_Administratives | 1: Citizen’s role is of administrative nature, 0 otherwise |
| Role\_Cleaners & Handlers | 1: Citizen’s role is described as “Cleaners & Handlers”, 0 otherwise |
| Role\_Professor | 1: Citizen is a professor, 0 otherwise |
| Working Hours per week | The number of working hours per week of the citizen |
| Money Received | The money payed to the elements of Group B |
| Ticket Price | The money received by the elements of Group C |
| Education\_Level\_Classified | The education level of the citizen in respect to the classification implemented in this project |
| Employment\_Sector\_Self-Employed (Company) | 1: Self-employed (Company), 0 otherwise |
| Employment\_Sector\_Private Sector - Services | 1: Employed in private sector, 0 otherwise |
| Income | The dependent variable (Where 1 is Income higher than the average and 0 Income Lower or equal to the average) |

Table : ‘features\_to\_keep\_1’ - The training dataset’s variables after the pre-processing steps

In a further feature selection step, RFE, a Ridge classifier and XGBoost are applied in order to identify the most important features. The result of this further limitation of the feature space is saved in the list ‘features\_to\_keep\_2’.

III.3.xx Verification of feature selection for MLP and AdaBoost classifier

This part contains an analysis whose goal it is to verify whether or not the feature selection defined in the list ‘features\_to\_keep\_2’ is an adequate choice for applying it in combination with an MLP[[5]](#footnote-5) and with an AdaBoost[[6]](#footnote-6) classifier. For this purpose, two steps are done:

1. The two algorithms were applied with four different subsets of the training dataset. As a quality measuring technique, k-fold cross validation with k = 10 was used.
2. Starting from ‘features\_to\_keep\_2’, features are randomly removed, one feature at a time and the validation score is visualised in a multiline plot.

In the next section, the process of selecting the appropriate models and their parameters is described.

**III.4 Model Selection and Parameter Tuning**

The purpose of the following part of the code is to assess the fitness of different models for dealing with the application at hand. Grid Search[[7]](#footnote-7) and k-fold cross validation with k = 10 are used to find optimal or near optimal combinations of parameter values for each algorithm. After this, the best models are selected based on their accuracy. This selection is also conducted using k-fold cross validation with k = 10. The models that are considered in this part are MLP classifier, AdaBoost, Gradient Boosting Classification[[8]](#footnote-8), Random Forest Classification[[9]](#footnote-9), Support Vector Classification[[10]](#footnote-10) and K-nearest Neighbours Classification[[11]](#footnote-11).

**IV. Results**

**IV.1 Results in Data Exploration and Data Pre-processing**

**IV.1.i Assessment of discriminatory power**

For the categorical variable ‘Base Area‘, most cases are in one category and very few observations are in the other categories. Due to the small sample sizes in the low cardinality categories, it is probable that the variation between the categories is only of random nature and doesn’t represent any type of pattern. The visualization for this analysis is shown in Figure 1.

Chart

Description automatically generated

Figure : Discriminatory power of ‘Base Area’

Following the same logic, the categorical feature ‘Native Continent’ was also discarded. Due to the low variation in the categories this one feature was considered as not relevant for the classification.

Chart, waterfall chart

Description automatically generated

Figure 2: Discriminatory power of ‘Native Continent’

As for the remaining categorical features, they were kept and one-hot encoded to be later used in the correlation assessment.

* Having removed those columns, we advance to the correlations between features

1. Between originally metric features there were no redundancies except for ‘Years of Education’ and ‘Education\_Level\_Classified’ which was expected. Both features have roughly the same correlation to the target variable ‘Income’. We kept the 2nd one because it is more informative than just ‘Years of Education’.
2. After it we advance to a bigger correlation matrix. This one is computed using all the remaining features ((state which ones exactly)), both metric and the encoded ones. We used pearson and spearman correlations. Two limits were stablished in both:
   1. In order to find redundancies, we want to retrieve only result with correlations higher than 0.3 and evaluate those.
   2. On the other hand, to find the most correlated features with the target and for that matter the limited is stablished in 0.08. Most features with a lower correlation to the target than 0.08 are removed.

As a result of this extensive matrices there are a few relations that are worth pointing out due to redundancy matters:

1. ‘Role\_?’ and ‘Employement\_Sector\_?’ have a correlation of 1. In this case ‘Employement\_Sector\_?’ is removed for two reasons. Firstly ‘Role\_?’ has higher correlation to our target variable. Not only that correlation is higher, but the second feature’s correlation is even lower than the limit established, 0.08.

Besides these relations, as said before, also features with correlation lower than 0.08 were also eliminated. After this step the process goes on with a much smaller list of features, which is ‘features\_to\_keep\_1’. ((Consider putting the feature table here))

This list of features is taken through other features selection techniques.

The first is the Recursive Feature Elimination. In this technique the model chosen to be used was a Gradient Boosting Classifier knowing that this one is a convenient model to the purposes of the problem, would make sense to have techniques that suit the models to be used. The result of the RFE was stored in a data frame indicating which features are relevant for our model and which are not. The value ‘True’ indicates it is relevant, and ‘False’ represents the opposite.

Table

Description automatically generated

Figure 3: Resulting DataFrame from RFE

Following the RFE, feature importance was assessed using a Ridge Classifier. The output of this process is a chart comparing all features’ importance.

Chart

Description automatically generated

Figure 4: Resulting chart from Ridge Classifier

From the chart we see that all originally metric features are considered important. Sitting right next to those, ‘Marital\_Status\_Married’ is the most important according to this techquine, ‘Role\_Management’ is also one to take into account as are ‘Role\_Professor’ and ‘Lives\_with\_Other\_Family’.

Next technique used, for feature selection, was also feature importance, but this time with XGBoost Classifier.

Table

Description automatically generated with medium confidence

Figure 5: Resulting chart from XGBoost Classifier

According to this technique, what is most important to point out is the high importance of originally metric features. Apart from those, the others don’t seem to have much to add to the model.

Once the 3 techniques are complete, the goal is to come up with a new list of features that provides us with a balanced result between the different techniques. Considering this we are only eliminating ‘Role\_?’. This one feature is the only one that is not revealed as important by any of the techniques. All the others have relative importance and can bring versatility to our model.

As a result of this techniques, we end up with ‘features\_to\_keep\_2’.

(Add the list here)

**IV.1.xx Results for verification of feature selection for MLP and AdaBoost classifier**

For the step described in III.3.xx point 1, the first subset of features for which the models’ performance is evaluated is defined by the list ‘features\_certainly\_to\_keep’ which consists of the features ‘Age\_days\_rel\_to\_2020’, ‘Working Hours per week’, ‘Money Received’, ‘Ticket Price’, ‘Education\_Level\_Classified’. These features are the five metric features but with the substitution of ‘Years of Education’ by 'Education\_Level\_Classified’. These features are regarded to be the most essential features of the dataset and are supposed to serve as a base case in this step of the analysis. The result for this feature set for the MLP and the AdaBoost classifiers are shown in table xx.

|  |  |  |  |
| --- | --- | --- | --- |
|  | Training score | Validation score | Iterations |
| MLP classifier | 0.83 | 0.828 | 71.9 |
| AdaBoost classifier | 0.846 | 0.844 | - |

Table xx: Results for base case (‘features\_certainly\_to\_keep’)

The seconds subset of features is ‘features\_to\_keep\_2’. For this subset the results are shown in table xx.

|  |  |  |  |
| --- | --- | --- | --- |
|  | Training score | Validation score | Iterations |
| MLP classifier | 0.859 | 0.853 | 71.9 |
| AdaBoost classifier | 0.867 | 0.865 | - |

Table xx: Results for ‘features\_to\_keep\_2’

The third subset of features is ‘features\_to\_keep\_1’. For this subset the results are shown in table xx.

|  |  |  |  |
| --- | --- | --- | --- |
|  | Training score | Validation score | Iterations |
| MLP classifier | 0.862 | 0.855 | 71.9 |
| AdaBoost classifier | 0.867 | 0.864 | - |

Table xx: Results for ‘features\_to\_keep\_1’

The last subset is the whole set of preprocessed training data which is stored in the variable ‘train\_data\_scaled’. For this, the results are shown in table xx.

|  |  |  |  |
| --- | --- | --- | --- |
|  | Training score | Validation score | Iterations |
| MLP classifier | 0.874 | 0.851 | 71.9 |
| AdaBoost classifier | 0.872 | 0.867 | - |

Table xx: Results for all data (‘train\_data\_scaled’)

Looking at these results it can be stated that the features defined by ‘features\_to\_keep\_2’ are a good choice for both models. The validation scores for ‘features\_to\_keep\_2’ and ‘features\_to\_keep\_1’ are approximately the same whilst the difference in the training and validation scores is lower for ‘features\_to\_keep\_2’ than for ‘features\_to\_keep\_1’. Also, the scores of the base case are lower than for ‘features\_to\_keep\_2’, which was expected since the base case has significantly less features. Using the whole data is not favourable, since it leads to overfitting for both models.

For the step described in III.3.xx point 2, which is the analysis of the validation score when randomly dropping one feature each iteration, starting with the features from ‘features\_to\_keep\_2’, yields the result shown in figure xx and xx, suggesting that it is probably favourable not remove any more features from ‘features\_to\_keep\_2’.

Chart, line chart

Description automatically generated

Figure xx: Visualisation of the decrease in validation accuracy for the MLP classifier

Chart, line chart

Description automatically generated

Figure xx: Visualisation of the decrease in validation accuracy for the AdaBoost classifier

**Grid Search MLP classifier**

The Grid Search conducted in order to find optimal or near optimal parameters for the MLP classifier resulted in the conclusion that the best parameters for this particular application are:

activation = 'tanh',

alpha = 4e-05,

hidden\_layer\_sizes = (10,),

learning\_rate = 'constant',

learning\_rate\_init = 0.01,

random\_state = 42, and

solver = 'adam'.

**Grid Search AdaBoost**

The best parameters found for AdaBoost are:

n\_estimators=962,

learning\_rate=0.61,

algorithm='SAMME.R', and

random\_state=42.

**Parameter tuning for Gradient Boosing classifier**

For the Gradient Boosting Classifier, as said in III.4, different parameters were manually tested:

{'n\_estimators': [50, 75, 100, 200],

'learning\_rate': [0.1, 0.2, 0.3],

'max\_features': [None, 7],

'max\_depth': [3, 4, 5],

'min\_samples\_leaf': [5, 6, 7],

'min\_samples\_split': [4, 7, 10],

'subsample': [1.0, 0.9, 0.8],

'max\_features': [5, 9, None],

'random\_state': [42]}

After testing these parameters and iteratively applying small changes the result set of parameters for the manual Gradient Boosting Classifier was:

{'learning\_rate': 0.1,

'max\_depth': 4,

'max\_features': None,

'min\_samples\_leaf': 2,

'min\_samples\_split': 10,

'n\_estimators': 150,

'random\_state': 42}

Adding to the manual testing, also a grid search was run including these parameters:

{'n\_estimators': [50, 100, 150, 300, 400],

'learning\_rate': [0.1, 0.3],

'max\_features': [None, 7],

'max\_depth': [2, 3, 5],

'min\_samples\_leaf': [1, 3, 5],

'min\_samples\_split': [4, 8, 10],

'random\_state': [42]}

The output of this search was:

{'learning\_rate': 0.1,

'max\_depth': 3,

'max\_features': 7,

'min\_samples\_leaf': 3,

'min\_samples\_split': 8,

'n\_estimators': 300,

'random\_state': 42}

When comparing both models’ (including the best parameters in the manual one) performances we got the following results:

|  | **Time** | **Train** | **Test** | **Iterations** |
| --- | --- | --- | --- | --- |
| **Manual** | 4.018+/-1.0 | 0.877+/-0.0 | 0.867+/-0.01 | nan+/-nan |
| **Grid** | 5.418+/-0.21 | 0.876+/-0.0 | 0.867+/-0.01 | nan+/-nan |

The scores of the models for the test set are exactly the same, however, the

manual seems to take less time and for that reason we chose to keep that one.

Not only the time it took was important, but also when stacking with other the results obtained using the manual were better.

Moving on into the Random Forest parameter tuning phase a similar approach was followed. The set of parameters tested was:

{'n\_estimators': [50, 100, 150, 200, 250, 300, 350, 400, 450, 500],

'max\_features': ['auto', 'sqrt'],

'max\_depth': [5, 10, 15, 20, 25, 30, 35, 40, 45, 50, None],

'min\_samples\_split': [10, 20, 30],

'min\_samples\_leaf': [6, 10, 14],

'bootstrap': [True, False]}

The set of best parameters retrieved was:

{'n\_estimators': 250,

'max\_features': 'auto'

'max\_depth': 30,

'min\_samples\_split': 10,

'min\_samples\_leaf': 6,

'bootstrap': False}

Using that configuration, the score obtained was:

|  | **Time** | **Train** | **Test** | **Iterations** |
| --- | --- | --- | --- | --- |
| **random forest** | 5.179+/-0.89 | 0.885+/-0.0 | 0.861+/-0.01 | nan+/-nan |

After evaluating the random forest, we will now look into the Support Vector Classification

(SVC) performance.

As we did for other models, we also ran a grid search in order to find the best possible set of

parameters to use in SCV. The set of parameters chosen in this case was as follows:

{'C': [1, 5, 10],

'gamma': ['auto', 'scale', 0.1, 5],

'kernel': ['rbf', 'linear', 'sigmoid']}

As a result of the grid, the best configuration within that set appears to be:

{'C': 5,

'gamma': scale,

'kernel': rbf}

The corresponding results of this model are shown below:

|  | **Time** | **Train** | **Test** | **Iterations** |
| --- | --- | --- | --- | --- |
| **scv** | 3.055+/-0.16 | 0.858+/-0.0 | 0.847+/-0.00 | nan+/-nan |

After this parameter tuning phase was completed, a high-level analysis of our models’ performances was issued. Using the best configurations found in the previous step, we take a look at the models we have so far and compare the scores. The main goal of this step is to decide which models represent better results for us. After finding them, we proceed to the stacking.

|  | **Avg. train score** | **Std. dev. train score** | **Avg. validation score** | **Std. dev. validation score** | **Avg. time** | **Std. dev. time** |
| --- | --- | --- | --- | --- | --- | --- |
| **Multilayer perceptron** | 0.8595 | 0.001137 | 0.8528 | 0.006613 | 6.12 | 1.28 |
| **Gradient Boosting Classifier** | 0.8768 | 0.000766 | 0.8671 | 0.007770 | 3.79 | 0.39 |
| **Ada Boost Classifier** | 0.8666 | 0.001012 | 0.8648 | 0.008005 | 11.40 | 1.00 |
| **Random Forest Classifier** | 0.8856 | 0.000937 | 0.8610 | 0.008026 | 5.83 | 1.32 |
| **SVC** | 0.8594 | 0.001000 | 0.8482 | 0.006426 | 18.10 | 0.50 |
| **K Neighbors** | 0.8721 | 0.001069 | 0.8379 | 0.009219 | 1.02 | 0.05 |

Table 3: General performance of the models tested

Chart

Description automatically generated with medium confidence

Fig 4: Models’ performances chart

As we see in the chart above, there are 3 models that stand out due to their higher performance when compared to the others. Those are the Gradient Boosting Classifier, the Ada Boost Classifier and the Random Forest Classifier.

Once the best models were found, the last step to take is understanding which combination of models provides us with the best results. In order to get this information, we created a for loop that runs a list containing the different combinations between the classifiers to stack.

Write about why random oversampling wasn’t used

Comparison of the three meta classifiers

**V. Discussion**

**VI. Conclusion**

**VII. REFERENCES**

[1] Fernández, Alberto & García, Salvador & Galar, Mikel & Prati, Ronaldo & Krawczyk, Bartosz & Herrera, Francisco. (2018). Learning from Imbalanced Data Sets. 10.1007/978-3-319-98074-4.

[2] <https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html>, viewed on 25. Dec. 2020 at 13:07.

[3] Bishop, Christopher. (2006). Pattern Recognition and Machine Learning. 10.1117/1.2819119.

[4] <https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html#sklearn.neural_network.MLPClassifier.score>, viewed on 25. Dec. 2020 at 16:14.

[5] <https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html>, viewed on 25. Dec. 2020 at 17:39.

[6] <https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html>, viewed on 26. Dec. 2020 at 15:17.

[7] <https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html>, viewed on 26. Dec. 2020 at 18:06

[8] <https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html>, viewed on 26. Dec. 2020 at 18:07

[9] <https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html>, viewed on 26. Dec. 2020 at 18:08

[10] <https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html>, viewed on 26. Dec. 2020 at 18:10

[11] Boehmke, Bradley; Greenwell, Brandon (2019). "Gradient Boosting". Hands-On Machine Learning with R. Chapman & Hall. ISBN 978-1-138-49568-5.

[12] <https://www.stat.berkeley.edu/~breiman/RandomForests/cc_home.htm>, viewed on 26. Dec. 2020 at 18:18

1. Or, if the “Kernel Trick” is used, a higher dimensional space of which said n-dimensional space is a subspace. [↑](#footnote-ref-1)
2. Typically, two classes as in this application, but adaptations for multiclass problems also exist (See chapter 7.1.3 in [3]). [↑](#footnote-ref-2)
3. Income = 1: 76.29 %, Income = 1: 23.71 % [↑](#footnote-ref-3)
4. NaN stands for Not a Number, meaning that the respective value is non-existent. [↑](#footnote-ref-4)
5. Multilayer Perceptron (for a general explanation see chapter 5 in [3]; for the documentation of the implementation used for this project see [4]) [↑](#footnote-ref-5)
6. AdaBoost, short for Adaptive Boosting (for a general explanation see chapter 14.3. in [3]; for the documentation of the implementation used for this project see [5]) [↑](#footnote-ref-6)
7. For the documentation of the implementation used in this project see [6] [↑](#footnote-ref-7)
8. For a general explanation see chapter 12 of [11]; for the documentation of the implementation used in this project see [7] [↑](#footnote-ref-8)
9. For a general explanation see [12]; for the documentation of the implementation used in this project see [8] [↑](#footnote-ref-9)
10. Explanation of the method in chapter II of this report; for the documentation of the implementation used in this project see [9] [↑](#footnote-ref-10)
11. For a general explanation see chapter 2.5.2 of [3]; for the documentation of the implementation used in this project see [10] [↑](#footnote-ref-11)