Introduction

The project consists basically of three different tools of supervised learning that have been applied to conduct our classification problem. According to data set provided by a Portuguese bank with sixteen input and one output variable, we try to compute the probability that a client will subscribe a term deposit. Delivering an answer to the mentioned question, we will apply three different tools of supervised learning. The first tool is Random Forest, followed by Logistic Regression and lastly by the Neural Network (Multi-layer Perceptron). In the concluding part, the different results of the several techniques applied, are compared by using confusion matrices. Thereby, we also provide a possible interpretation of the outcome resulting of one confusion matrix which represent one applied technique. For the final part, we provide interpretability results for the Random Forest Classifier.

By conducting our analysis and, thus, programming our algorithms, we follow the assumption that machine learning and statistics, classification is the problem of identifying to which of a set of categories (sub-populations) a new observation belongs. Thereby, we divide our dataset into a training set and a test set. Based on the data of the training set which contains observations (or instances) whose category membership is known.

In this notebook we use the Bank Marketing Dataset4 to build a model to predict whether someone is going to make a deposit or not depending on some attributes. We intend to build and compare three algorithms (Random Forest, Logistic Regression, Neural Network) to detect the best model for our case. Then, we try to optimize our selected model by tuning its hyper parameters by using GridSearch. Lastly, we save the prediction result from our dataset and save our model for reusability.

After uploading our basic data set, we start with loading some basic libraries such as Pandas and NumPy for handling data frames and lists. To create and evaluate our machine learning algorithms, we import some metrics of the skicit-learn library. We also import seaborn and matplotlib library that are used for visualization purposes. The availability of our datasets permits us now to continue with the preparation process of the dataset, the so-called data pre-processing. This step ensures that our model is based on a high-quality learning dataset to be in accordance with the assumption that "a model is only as good as it's data". The data pre-processing itself consists of a series of steps which are explained below.

Data pre-processing

The first step to be taken is to load and indicate the imported dataset. For visualizing the content of the dataset, we use the pandas library which allows us to create a dataframe. The dataframe contains the “duration” column which represents the duration of the last contact between the company and the client via phone. Because of the fact that the duration of the call is not known in advance and its high influence of to the model, this component is eliminated to avoid falsification. In our code we use the df.drop function to cancel the column.

As second step we have to make sure that the distribution of our predicted column “y” is appropriate for a meaningful model. We can see that the distribution between the outcomes “y” are 40,000 for the “no” and 5300 for the “yes”. Therefore, we have an imbalanced dataset, and we need to oversample. In order not to create a bias, we will not oversample the whole dataset but only the training set, after the step that we split our data.

After that, we need to check for missing values. These missing errors may have several motives of origin, i.e., human errors. Through the isnull() function from the pandas library we can detect those missing values within the columns to avoid errors. Furthermore, by applying the sum() function, the amount of missing values can be computed. As outcome we receive the name of all columns with their respective number of missing values. In our examined case, there has not been detected a missing value which permits us to continue the procedure without taking any further action.

In the next step, we plot a pairplot of the whole dataset, compared to the outcome, to observe how the “y” outcome is related according to each column. In the dataset, we can observe that we have numerical and categorical values, so we need to separate them. Thus, we plot a histogram of the numerical data, to observe how they are distributed. Moreover, we apply a correlation matrix to see which values are highly correlated and which not.

Afterwards, we need to scale our numerical data to avoid outlier presences that may significantly affect our model. Using StandardScaler() function from the sklearn library, we can scale each column which contains numerical data. The scaling is done by using the formula below:

Z = X−US → Where Z is the final scaled value, X is the initial value, U is the mean of the data and S is the standard deviation of the data.

After scaling our data, we use the OneHotEncoder function in order to handle the categorical data. The goal is to transform non-numerical values like “yes” or “no” that describe the personal state of each person into numerical values. In our case, we use the value 1 to substitute the value “yes” while 0 is used for substituting “no”.

In the next step of the data pre-processing, we divide the data set into a training and into a test set. Basically, there is no specific rule that specifies the respective percentages of the training set and the test set. The training set just needs to be sufficiently large to ensure that the model is able to learn from the available data and is used to train the algorithm. In our case, due to a small data set (almost 50,000) we decided to choose a proportion of 0.85 for usage as training set and thus, the remaining part of 0.15 is used as test set.

Lastly, as we mentioned earlier on, we have an imbalanced dataset, so we need to apply an oversampling strategy that oversamples the minority class only in our training set and not in the initial dataset. For this reason, we import the RandomOverSampler function from the imblearn package and as a result we transform our data into an equal proportion of 33,942 “yes” and 33,942 “no”. Therefore, by plotting the histogram of “y” we can verify it.

Modelling

Having concluded the data pre-processing, we now have a good basis available for the model construction which is the actual main part of our examination. The generated data set represents the basis for all the three models we are presenting in our report. In this step we will create a baseline model for each algorithm using the default parameters set by the support of the sklearn library. After, we compare the models to see which works best for our case. For comparison, we use confusion matrices of all the three applied models. We have decided to calculate the following metrics for evaluation of the models.

We will use the 3 metrics below to evaluate models:

1. **Accuracy Score**: the proportion of true results among the total number of cases examined. It is a metric that is used in classification problems to find the percentage of accurate predictions. It is calculated by dividing the number of correct predictions by the total number of predictions.
2. **Precision Score**: It is used to calculate how much proportion of all data that was predicted positive was actually positive. In an imbalanced classification problem with two classes, precision is calculated as the number of true positives divided by the total number of true positives and false positives.
3. **Recall**: It is used to calculate how much proportion of actual positives are correctly classified. The precision is the ratio of true positives divided by the sum of true positives and false positives. Recall is related to the ability of a classifier not to label as positive a sample that is negative. The best value is 1 and the worst value is 0**.**
4. **F1 Score:** It is a measure of the accuracy of a model and is calculated from the precision and recall of the test. *(F1-Score = (2 * Precision * Recall) / (Precision + Recall)*. Because it takes into consideration both precision and recall, it is a better metric to evaluate the performance of a model compared to accuracy.

Apart from the metrics mentioned above, we also calculate and plot the confusion matrix for each algorithm. Confusion matrix is a table that is used to define the performance of a classification algorithm and visualizes and summarizes the performance of a classification algorithm. By computing it, we can have a clear picture of the exact number of the true positive, true negative, false positive and false negative samples.

In our case we ideally want to maximize the recall score of each model because in our problem we should try to predict as many actual positive as we can. Misclassification of customers who wanted to make a deposit can mean a lost opportunity and, thus, lost revenues.

Below we will define a custom function that is used to evaluate the metrics and the confusion matrix of each trained model, using the sklearn metrics library and save the score to a variable.

Constructing the three Algorithms

Random Forest

The first of our four approaches applied is the Random Forest method. This approach basically takes its final decision using multiple decision trees during the training phasis. For example, every decision tree generates an output which are collected in a so-called random forest. The most frequent outputs result as final decision. This tool might be especially useful for regression and classification issues because of its clear indication on how inputs are weighted by the model. Thus, importance of the input factors can be derived.

As very first step we import the confusion matrix and the random forest classifier from the sklearn library. The imported classifier then must be built based on two variables (X_over, y_over) from the oversampled dataset (here we could also use the train-split set because random forest handles very good imbalanced data) and by setting its random state equal to zero (max=42). In the following step, we create an evaluation model which delivers its accuracy, its precision, its recall, and the respective confusion matrix. Regarding the accuracy a value of 0.88 can be observed while its precision is about 0.49 and its recall with f1 score is 0.31 and 0.38 respectively. The confusion matrix underlines these outcomes by indicating 5845 correctly as false predicted labels and 189 correctly as true predicted labels. On the other hand, we have 613 wrongly as false and 135 as wrongly true predicted labels.

Logistic Regression

For conducting the second applied method, the logistic regression, another package from the sklearn library is imported and a corresponding variable “model” is generated. As already illustrated in the part of random forest model, the model is evaluated on the basis of two variables of the oversampled dataset and the imported logistic regression model. By printing the outcomes, its accuracy and precision overcome those of the random forest model with a value of 0.76 (accuracy), 0.27 (precision), 0.60 (recall) and 0.37 (f1 score). By plotting its respective confusion matrix, it can be observed that 4657 labels are correctly predicted as wrong and 487 labels correctly as right. Instead, the correct prediction, 315 labels are wrongly predicted as false, and 1323 labels are wrongly predicted as true. Summing up the results we have already obtained, we can see that accuracy and prediction of the logistic regression have been underperforming compared to the priorly presented random forest model.

Neural Network – MultiLayer Perceptron

As the last of the used machine learning models, we dedicate the following section to Neural Networks and particularly to multi-layer perceptron classifier.

Again, the sklearn library enables us to import the MLPClassifier function which is used to create a variable based on the dimensions of the hidden layers and the hidden units. In our presented case, for simplicity reasons, we use 3 layers — (13 input, 10 hidden and 2 output) with maximum iterations of 300. We also use as an activation function the "ReLU" (rectified linear unit), as it is very accurate for binary classification problems. This function indicates the input directly if it is positive, otherwise, the output results zero. In addition, we use as a solver for weight optimization the "adam" method which is a stochastic gradient-based optimizer. The accuracy for this algorithm is 0.78 which is more than the logistic regression but less from the random forest, the precision is almost 0.3. However, we can see that we obtained a good recall score compared to the other algorithms reaching almost 0.64 and f1 score of 0.385. These parameters can be fine-tuned later based on domain and data used to improve accuracy. For the model evaluation, again, the created mlpclassifier variable is used together with the inputs from the training dataset that permits us compute the metrics and plot the confusion matrix. We can observe that by plotting its confusion matrix, 4777 labels are correctly predicted as wrong and 513 labels correctly as right. Instead, the correct prediction, 289 labels are wrongly predicted as false, and 1203 labels are wrongly predicted as true.

Hyperparameter Tuning

For the constructing our random forest classifier, we used earlier the default parameters. However, to make the model as optimal as possible we can import the GridSearchCV function from the sklearn model selection package. This helps us to loop through predefined hyperparameters and fit our estimator (model) on our training set. Finally, we can select the optimal parameters from the listed ones and fit our model. Because of the time-consuming computation, we run it only for the Random Forest Classifier, setting some important parameters. After the running process we obtained the following outputs: max_depth=50, max_features=4, min_samples_leaf=3, min_samples_split=8, n_estimators=500. Having computed the optimal parameters, we run again the random forest classifier algorithm but this time with these parameters. Compared to the default parameters for random forest, we now obtained less accuracy =0.85 and precision = 0.42. Nonetheless, we increased our recall score significantly which is an important factor for model evaluation reaching 0.51 and the f1 score to 0.46.

Interpretability of Random Forest Classifier

For evaluating interpretability results, we decided to use the SHAP (Shapley Additive Values) library which explains small pieces of complexity in a specific machine learning model. In our case it is applied to Random Forest. The used approach explains individual predictions one at a time and, thus, we can observe contribution and importance of each feature to an individual predicted value.

Starting from installing and importing the SHAP library, first we create the Tree explainer object that can help us calculate the shap values. After that, we print the X_test data set to check the indexes of it and randomly we select the first row in order to understand how our random forest classifier makes its predictions. By calculating the shap values for the first row, we can see a force plot visualizing shap values for the features. The size of the bar indicates the magnitude of the effect of the specific feature. The feature values with blue color, tend to decrease the prediction. The sum of all feature shap values explain why the model prediction was different from baseline.

The chosen instance (3610 – first row) indicates a person with blue collar job, who is married, with primary education, with no default, with housing, without loan, with unknown contact, tried to get contacted on May, with unknown poutcome (which means there was no previous marketing campaign). The model predicted 0.13 that the client won't subscribe the term deposit where base value is 0.5. The biggest effect here is that the contact was unknown.

The next chosen instance (44323 - forth row) indicates a person with blue collar job, who is married, with primary education, without default, without housing, having a loan, having contacted by telephone (cellular) in July, with successful poutcome (which means that the previous campaign was successful). The model predicted 0.82 that the client will subscribe the term deposit which is pretty much very confident where base value is 0.5. The biggest effect here is that there was a previous campaign which was successful.

Next, we plot the summary of the shap values in our training set and we can see that the most important feature for training our random forest classifier was the poutcome_sucess which indicates if there was a previous marketing campaign or not, followed by previous (number of contacts performed before this campaign for this client) and pdays (number of days that passed by after a client was last contacted from a previous campaign.

For the Global Interpretability, we create a sample from our X_test set. By applying just for the 10 features a short display (max_display=10), we can see the importance of every feature contributing to the outcome of "yes" or "no" class. In this case we can see that the most important feature for both classes is "age", followed by "balance" and "contact_cellular". Thus, we can say that these features contribute the most in our random forest classifier for making its predictions.

Conclusion

To conclude our analysis, we compare the scores of all three techniques applied. The accuracy applied consists of the sum of the correctly wrong and the correctly true predicted that is divided through the total number of the sample

Comparison of the scores of the three algorithms used

	Random Forest (without Hyperparameter tuning)	Logistic Regression	Neural Network – Multilayer Perceptron
Accuracy Score	0.88	0.752	0.755
Precision Score	0.5	0.26	0.27
Recall Score	0.31	0.61	0.62
F1 Score	0.382	0.37	0.385

From the table above we can see that the highest Accuracy and Precision Score was that from the Random Forest (0.88 and 0.5 respectively), which means that our algorithm predicted 88% (from accuracy point of view) correctly the classification (if a client will subscribe or not), but that doesn't mean that is the best model, because our algorithm as we can see has very accurately predicted the "no" but very poorly the "yes". However, if we take into consideration the recall score (which is the number of correct positive results divided by the number of all relevant samples) the Neural Network is the most accurate.

In imbalanced datasets, the goal is to improve recall without hurting precision. These goals, however, are often conflicting, since in order to increase the TP for the minority class, the number of FP is also often increased, resulting in reduced precision. For this reason, we calculated the F1 score which takes into consideration both the precision and recall score.

Concluding, if we take into consideration the F1 score, we can see that the neural network has slightly a bit higher score (0.385) but without much difference from the Random Forest one (0.382). Because we have similar f1 score for neural network and for the random forest we can compare just the accuracy and decide which algorithm is better. By only evaluating f1 score, is not exactly precise because we don't' know what is maximized (precision or recall). Therefore, the accuracy of the Random Forest is way more higher than the neural network one (0.88 and 0.755 respectively) and thus, we can say that the Random Forest Classifier is the best model among the other two, for predicting if a client will subscribe a term deposit.