Machine Learning Engineer Nanodegree

Capstone Project

Prakhar Dogra
February 22, 2018

Bank Marketing Classification

I. Definition

Project Overview

An increasing number of marketing campaigns has reduced its effect on the general public over time. Moreover, competition has led marketing managers to invest on directed marketing campaigns with a strict and rigorous selection of contacts. And such direct marketing campaigns can be enhanced using various Machine Learning techniques.

There are two main approaches for enterprises to promote products and services. Either through mass campaigns that targets general public or directed marketing, that targets a specific set of contacts [2]. Nowadays, in this today's competitive world, positive responses to mass campaigns are typically very low. And directed marketing focuses more on targets that assumable will be keener to that specific product/service, making these kinds of campaigns very attractive due to its efficiency [3]. But directed marketing has some drawbacks, for instance it may trigger a negative attitude towards banks due to the intrusion of privacy [4].

Due to internal competition and current financial crisis, there are huge pressures for banks to increase a financial asset. And to solve this particular issue, a popular strategy is to offer attractive long-term deposit applications with good interest rates, in particular by using directed marketing campaigns. Also, the same drivers are pressing for a reduction in costs and time. Thus, there is a need for an improvement in efficiency: lesser contacts should be done, but an approximately number of successes (clients subscribing the deposit) should be kept.

Given the interest in this domain, there are several works that use Machine Learning to improve bank marketing campaigns [2] [5] [6]. These works often use a classification Machine Learning approach, where the goal is to build a predictive model that can label a data item into one of many classes (e.g. “yes”, “no”). Several Machine Learning algorithms can be used for classifying marketing contacts, each one with its own purposes and capabilities. Examples of popular Machine Learning techniques are: Naïve Bayes (NB) [7], Decision Trees (DT) [8] and Support Vector Machines (SVM) [9].

Problem Statement

This project utilizes different types of Machine Learning algorithms, using the Bank Marketing dataset [1], to check if the client has subscribed for a term deposit depending on various bank marketing attributes like age, type of job, education level, if the client has a housing loan or not, last date of contact, etc.

Logistic Regression has been chosen as the benchmark model for this project and will be compared to the following classifiers:

- Decision Tree Classifier
- Random Forest Classifier
- AdaBoost Classifier
- Gaussian Naïve Bayes Classifier
- Multinomial Naïve Bayes Classifier
- Multi-Layer Perceptron
- Support Vector Machine

Feature engineering has been done that also includes feature selection (using Chi-Square Test), transformation (using Principal Component Analysis) and elimination (using Recursive Feature Elimination), to see if there is any improvement over the classifiers (classifiers are trained separately for different feature sets obtained). Finally, I compared all the above-mentioned classifiers with the benchmark classifier (Logistic Regression classifier). Moreover, the dataset has two version (old and new) so I have used both to see if the classifiers behave the same way.

Metrics

The most common used metric to measure and compare classifiers is the accuracy.

But in the dataset there are 88.3% examples of “no” and 11.7% examples of “yes”. So due to the nature of the dataset, accuracy is not a good measure as the dataset is

unbalanced. So, for the purpose of this project F1-score and Area under ROC score has been used as the metric for evaluation and comparison.

F1-score considers both the precision p and the recall r of the test to compute the score: p is the number of correct positive results divided by the number of all positive results returned by the classifier, and r is the number of correct positive results divided by the number of all relevant samples (all samples that should have been identified as positive). The F1 score is the harmonic average of the precision and recall, where an F1 score reaches its best value at 1 (perfect precision and recall) and worst at 0. That's the reason why it is considered a good evaluation metric when dealing with imbalanced classification because it takes into account for the false positives and false negatives.

F1-Score is calculated as:

$$F_1 = 2 \cdot \frac{\mathrm{precision} \cdot \mathrm{recall}}{\mathrm{precision} + \mathrm{recall}}.$$

where precision is true_positive/(true_positive + false_positive)
and recall is true_positive/(true_positive + false_negative)

On the other hand, Area under ROC is the are under the ROC (Receiver Operating Characteristic) curve. An area of 1 represents a perfect test; an area of 0.5 represents a worthless test. In a ROC curve the true positive rate (Sensitivity) is plotted in function of the false positive rate (100-Specificity) for different cut-off points of a parameter. Each point on the ROC curve represents a sensitivity/specificity pair corresponding to a particular decision threshold. The area under the ROC curve (AUC) is a measure of how well a parameter can distinguish between two classes.

II. Analysis

Data Exploration

The data used for this project is the Bank Marketing Dataset [1]. It is related with direct marketing campaigns (phone calls) of a Portuguese banking institution. The marketing campaigns were based on phone calls. Often, more than one contact to the same client was required, in order to access if the product (bank term deposit) would be ('yes') or not ('no') subscribed.

Following are the two datasets that have been used for this project:

1. ***bank-full.csv***

    a. This dataset has 45211 entries, each with 17 inputs, ordered by date (older version of this dataset with less inputs). First 16 inputs for each entry are attributes and 17th input is the class to be predicted.
    b. Out of 16 attributes, 7 attributes (age, balance, day, duration, campaign, pdays and previous) are numeric. Rest are categorical (including binary attributes).

Following are the attributes of this dataset (with brief description):

bank client data:

1. age (numeric)
2. job: type of job
   (categorical: "admin.", "unknown", "unemployed", "management", "housemaid", "entrepreneur", "student", "blue-collar", "self-employed", "retired", "technician", "services")
3. marital: marital status
   (categorical: "married", "divorced", "single";
   note: "divorced" means divorced or widowed)
4. education (categorical: "unknown","secondary","primary","tertiary")
5. default: has credit in default? (binary: "yes","no")
6. balance: average yearly balance, in euros (numeric)
7. housing: has housing loan? (binary: "yes","no")
8. loan: has personal loan? (binary: "yes","no")

related with the last contact of the current campaign:

9. contact: contact communication type
   (categorical: "unknown", "telephone", "cellular")
10. day: last contact day of the month (numeric)
11. month: last contact month of year
    (categorical: "jan", "feb", "mar", ..., "nov", "dec")
12. duration: last contact duration, in seconds (numeric)

other attributes:

13. campaign: number of contacts performed during this campaign and for this client (numeric, includes last contact)

14. pdays: number of days that passed by after the client was last contacted from a previous campaign (numeric, -1 means client was not previously contacted)
15. previous: number of contacts performed before this campaign and for this client (numeric)
16. poutcome: outcome of the previous marketing campaign (categorical: "unknown", "other", "failure", "success")

Output variable (desired target):

17. has the client subscribed a term deposit? (binary: "yes","no")

2. *bank-additional-full.csv*

a. This dataset has 41188 entries, each with 20 inputs, ordered by date (from May 2008 to November 2010)
b. First 20 inputs for each entry are attributes and 21st input is the class to be predicted.
c. Out of 20 attributes, 10 attributes (age, duration, campaign, pdays, previous, emp.var.rate, cons.price.idx, cons.conf.idx, euribor3m, nr.employed) are numeric. Rest are categorical (there are no binary attributes in this dataset).

Following are the attributes of this dataset (with brief description):

bank client data:

1. age (numeric)
2. job: type of job (categorical: "admin.", "blue-collar", "entrepreneur", "housemaid", "management", "retired", "self-employed", "services", "student", "technician", "unemployed", "unknown")
3. marital: marital status (categorical: "married", "divorced", "single", “unkown”; note: "divorced" means divorced or widowed)
4. education (categorical: "basic.4y", "basic.6y", "basic.9y", "high.school", "illiterate", "professional.course", "university.degree", "unknown")
5. default: has credit in default? (binary: "yes","no", “unkown”)
6. housing: has housing loan? (binary: "yes","no")
7. loan: has personal loan? (binary: "yes","no")

related with the last contact of the current campaign:

8. contact: contact communication type
   (categorical: "unknown", "telephone", "cellular")
9. month: last contact month of year
   (categorical: "jan", "feb", "mar", ..., "nov", "dec")
10. day_of_week: last contact day of the week
    (categorical: "mon", "tue", "wed", "thu", "fri")
11. duration: last contact duration, in seconds (numeric)

other attributes:

12. campaign: number of contacts performed during this campaign and for this client (numeric, includes last contact)
13. pdays: number of days that passed by after the client was last contacted from a previous campaign (numeric, -1 means client was not previously contacted)
14. previous: number of contacts performed before this campaign and for this client (numeric)
15. poutcome: outcome of the previous marketing campaign
    (categorical: "failure", "nonexistent", "success")

social and economic context attributes:

16. emp.var.rate: employment variation rate - quarterly indicator (numeric)
17. cons.price.idx: consumer price index - monthly indicator (numeric)
18. cons.conf.idx: consumer confidence index - monthly indicator (numeric)
19. euribor3m: euribor 3 month rate - daily indicator (numeric)
20. nr.employed: number of employees - quarterly indicator (numeric)

Output variable (desired target):

21. has the client subscribed a term deposit? (binary: "yes", "no")

Also, as specified above in the Metrics section, the data is unbalanced as there are 88.3% examples of “no” and 11.7% examples of “yes”.

Exploratory Visualization

Firstly, the number of examples belonging to each class are plotted:

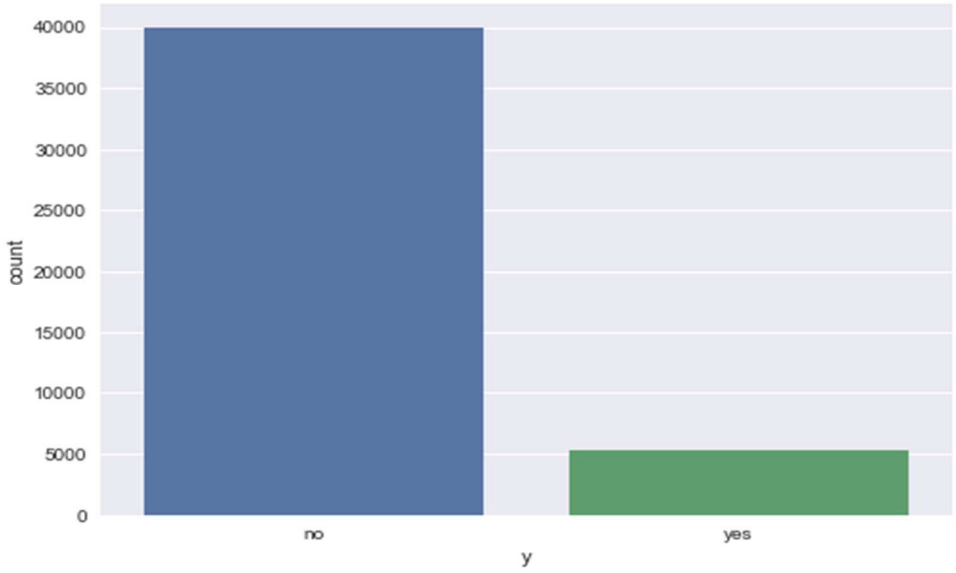

Figure 1: Displaying the count of "no" and "yes" in the dataset.

Some histograms were also plotted for some features:

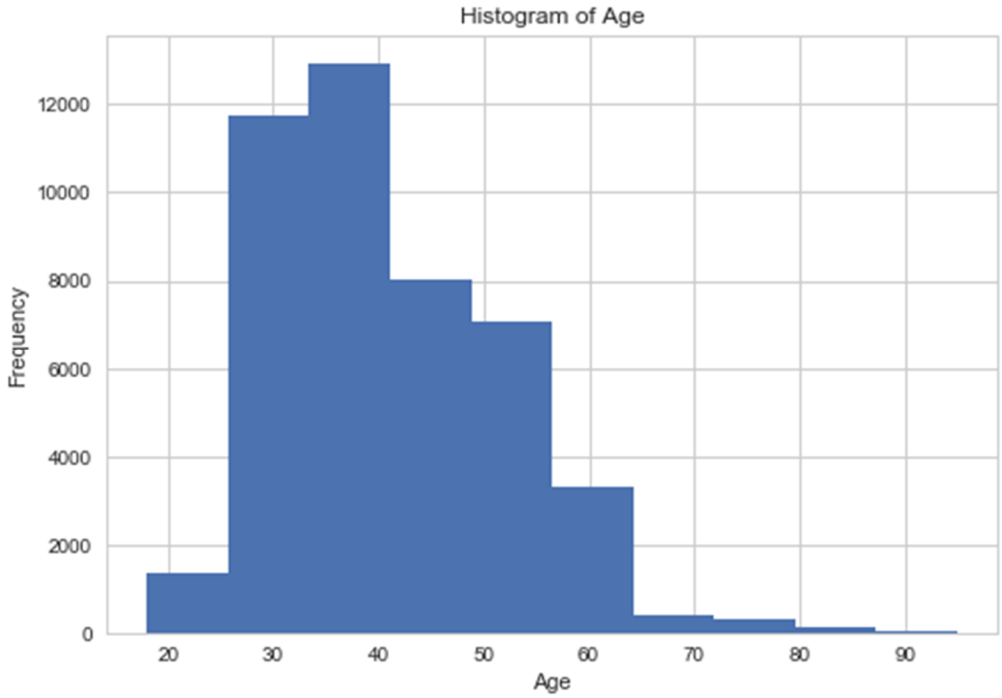

Figure 2: Histogram of "Age" feature.

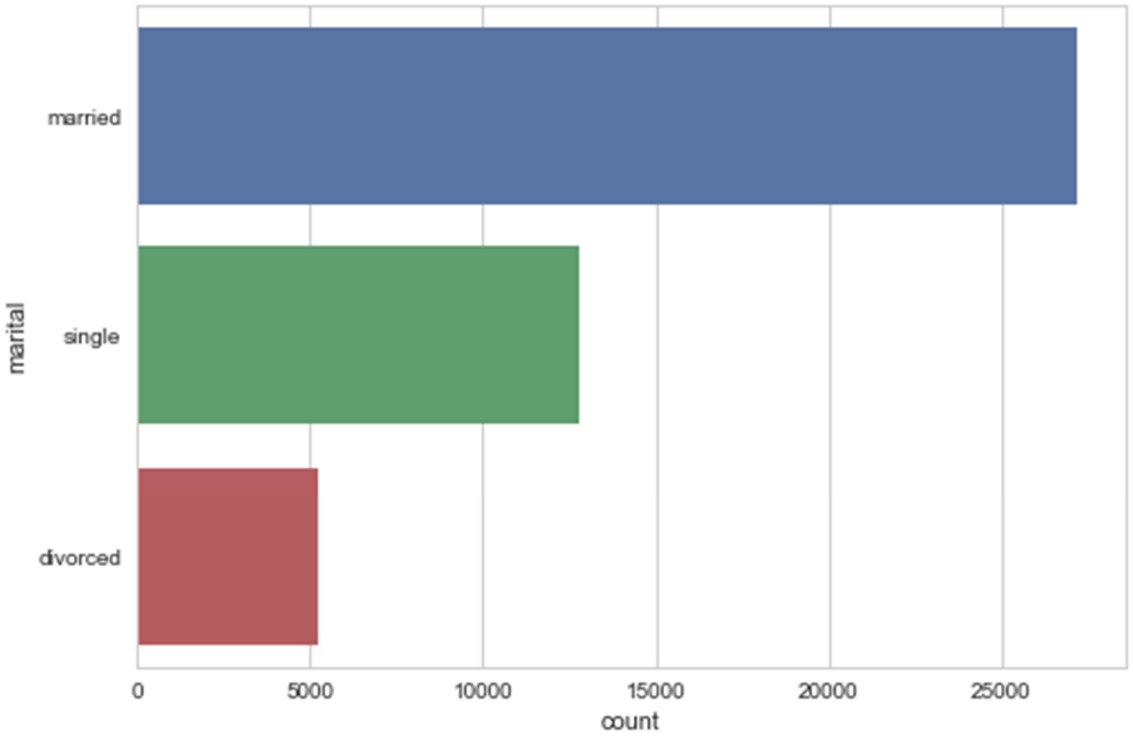

Figure 3: Histogram of "marital" feature.

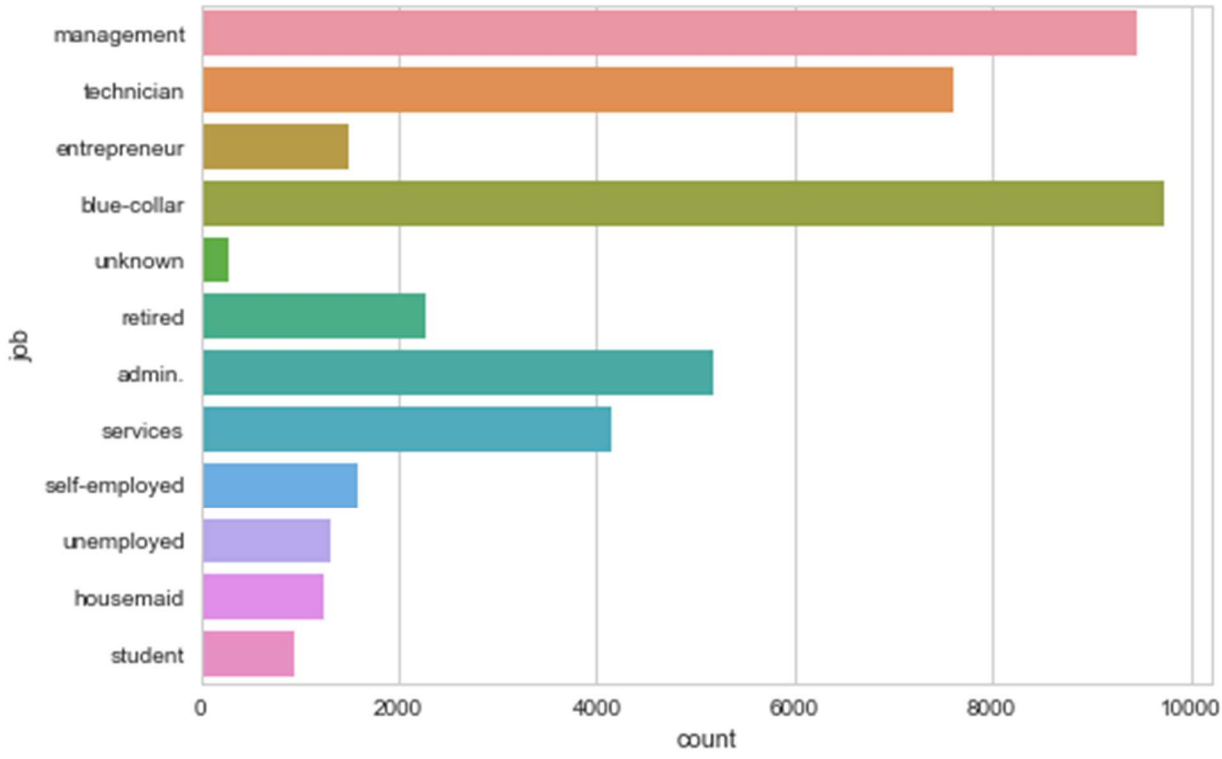

Figure 4: Histogram of "job" feature.

Some more data visualization was done on both datasets (refer to both jupyter notebooks to see)

Then after doing feature engineering, correlation matrix was calculated to find out those features that are highly correlated to each other. A heatmap (figure shown below) was plotted to identify those candidate feature pairs.

Feature-pair (pdays - previous) is highly positively correlated. Therefore, we can remove feature "pdays".

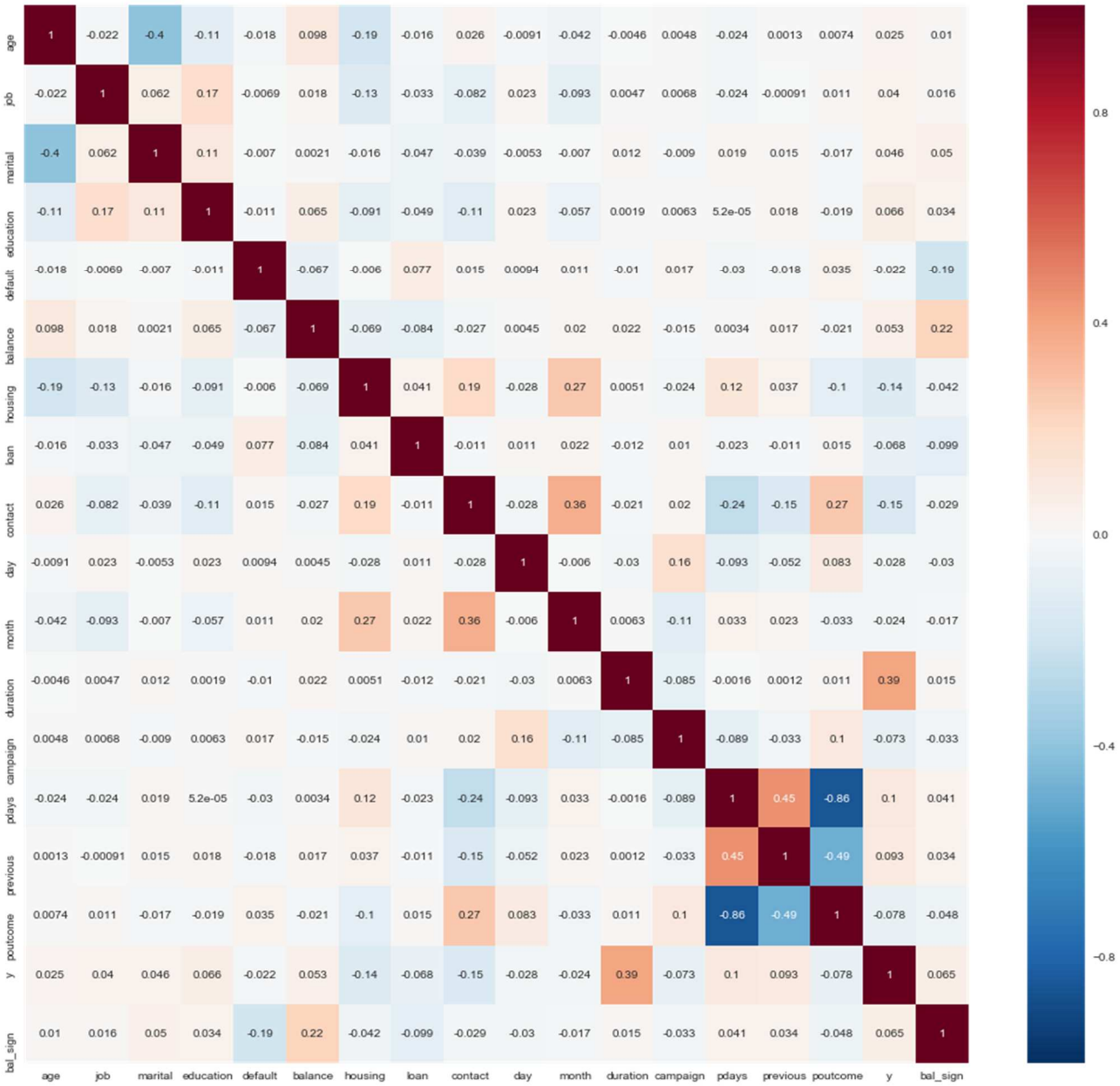

Figure 5: Heatmap of the correlation matrix of first dataset

Similarly, for second dataset, after doing feature engineering, correlation matrix was calculated to find features that are highly correlated to each other. A heatmap (figure shown below) was plotted to see those candidate feature pairs.

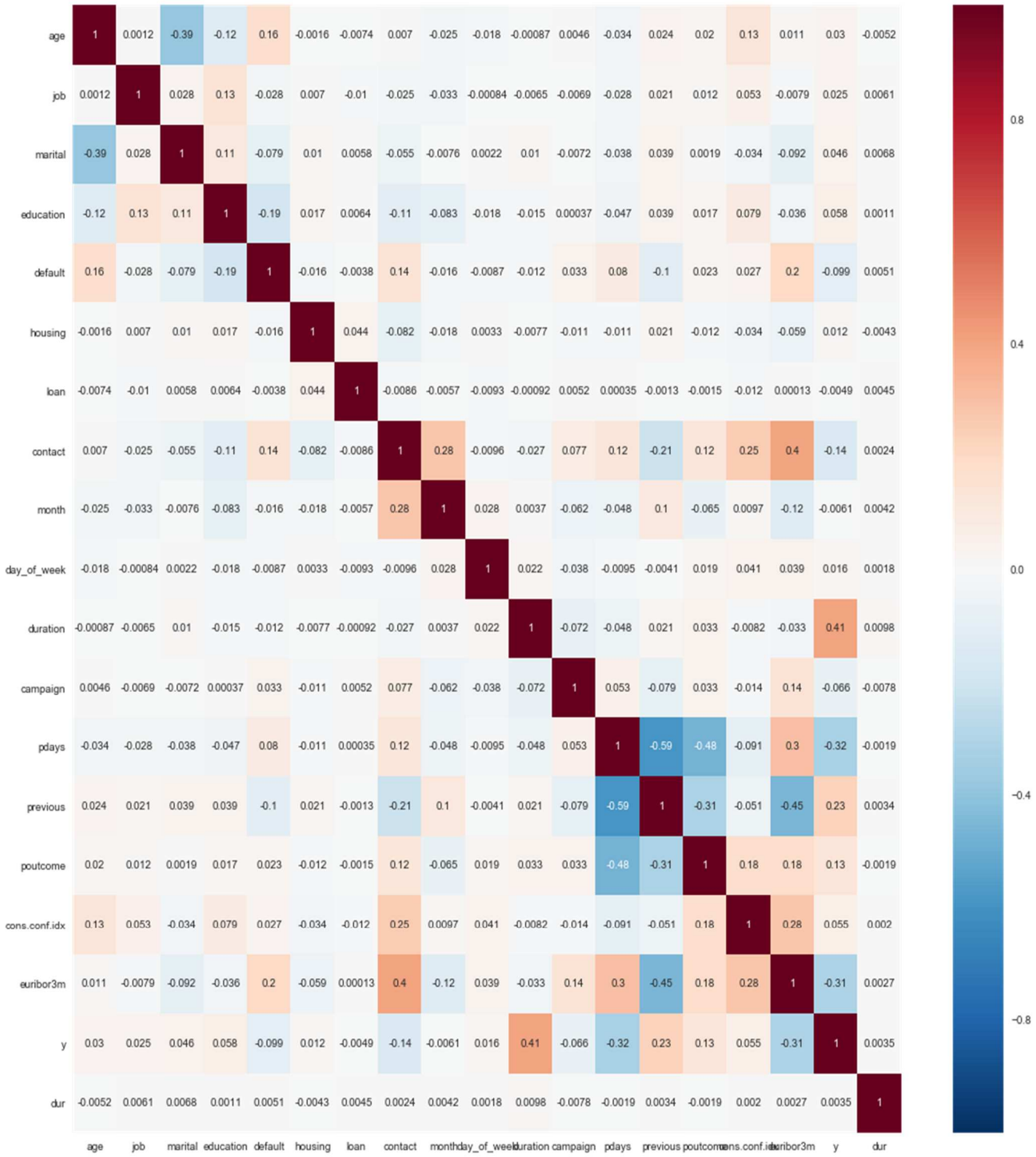

Figure 6: Heatmap of the correlation matrix for second dataset

Feature-pairs (emp.var.rate - cons.price.idx), (emp.var.rate - euribor3m), (emp.var.rate - nr.employed), (nr.employed - euribor3m) and (cons.price.idx - euribor3m) are highly positively correlated. Therefore we can remove "emp.var.rate", "cons.price.idx", "nr.employed"

Algorithms and Techniques

For this project following classifiers have been used:

- Logistic Regression (Benchmark model) (discussed in next section)
- Decision Tree Classifier:
    - Decision Trees have several real-world applications. One of them can be to find out the disease of the patient by asking them about the medical history and observing symptoms. Doctor simply has to ask several questions or do some tests to find out the disease.
    - Decision Trees automatically finds the most important attributes (feature selection) from the original set of attributes since the top most nodes on which the tree is split are essentially the most important variables within the dataset. Decision trees require relatively little effort from users for data preparation since each node is independent of the other in terms of the magnitude (value) of the attribute. Decision Trees are also very good at finding non-linear relationships.
    - Decision Trees can be extremely sensitive to small perturbations in the data: a slight change can result in a drastically different tree. They can easily overfit and hence aren't smooth.
    - Decision Trees is a good candidate for the problem since we aren't sure of the data is linearly separable just by looking at the data. Moreover, about half of the features are not numeric (even before one-hot encoding) so it is better to use Decision Trees since it works good with mixed data.
- Random Forest Classifier:
    - Random Forest can have a lot of real-world applications like email spam filtering where certain keywords can be used identify if the email is spam or not.
    - Random Forest inherits most of the advantages of a Decision Tree. Moreover, there is a reduction in overfitting since several trees are averaged and hence there is a significantly lower risk of overfitting. And there is less variance due to the use of multiple trees.
    - Random Forests are slow to create since there are number of Decision Trees are required to be created. Moreover, results of learning are

incomprehensible. Compared to a single decision tree, or to a set of rules, they don't give you a lot of insight.
    - Random Forest is a good candidate for the problem since we aren't sure of the data is linearly separable just by looking at the data. Moreover, half of the features are not numeric (even before one-hot encoding) so it is better to use Decision Trees since it works fine with mixed data.
    - The data is slightly unbalanced and Random Forests are known to deal really well with imbalanced data.
- AdaBoost Classifier:
    - An AdaBoost classifier is a meta-estimator that begins by fitting a classifier on the original dataset and then fits additional copies of the classifier on the same dataset but where the weights of incorrectly classified instances are adjusted such that subsequent classifiers focus more on difficult cases.
    - Logistic Regression and Decision Tree Classifiers have been used as base estimators for AdaBoost classification purposes.
- Naïve Bayes Classifier:
    - When assumption of independence holds, a Naive Bayes classifier performs better compare to other models like logistic regression and less training data is required.
    - It is easy and fast to predict class of test data set.
    - Gaussian Naïve Bayes and Multinomial Naïve Bayes are two variants that have been used in the project. (Bernoulli Naïve Bayes requires binary inputs)
- Multi-Layer Perceptron:
    - This classifier implements a simple neural. And neural networks are function approximators which prove to be extremely useful for classification purposes.
    - A Multi-Layer Perceptron consists of at least three layers (one or more hidden layers) of nodes. Except for the input nodes, each node is a neuron that uses a nonlinear activation function. It utilizes backpropagation for training. Multiple layers and non-linear activation of a Multi-Layer Perceptron, distinguishes it from a linear perceptron. Hence, it can distinguish data that is not linearly separable.
    - Multi-Layer Perceptron learns by updating the connection weights after each piece of data is processed, based on the value of error in the output when compared to the expected result.

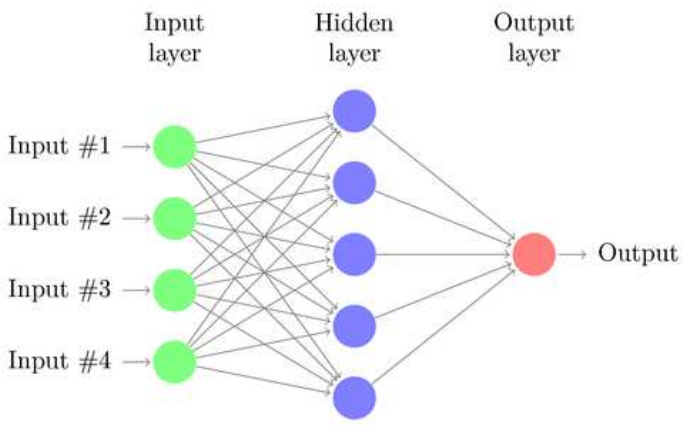

Figure 7: A simple Multi-Layer Perceptron with 1 hidden layer

- Support Vector Machine:
  - Support Vector Machines have many real-world applications such as text categorization, image classification, etc. They are capable to produce decision boundaries (linear or non-linear).
  - Support Vector Machine has many advantages. They have regularization parameter that helps avoid over-fitting. It uses the kernel trick and tries to maximize the margin to get better results for test data.
  - Support Vector Machines have disadvantages as well. choosing appropriately hyper parameters of the SVM that will allow for sufficient generalization performance. Moreover, kernel models can be quite sensitive to over-fitting the model selection criterion (choosing the appropriate kernel function can be tricky).
  - Support Vector Machines is a suitable candidate because it can find non-linear decision boundaries for classification. Moreover, they work well when dealing with nominal (binary) attributes.

For each classifier (except Naïve Bayes Classifiers because they calculate the prior automatically) the hyperparameter *class_weight = "balanced"* has been used to make sure that class imbalance is accounted for in every classification task. Moreover, the hyperparameter *stratify* has been used during train-test-split to make sure that the equal percentage of each class is divided among training and testing data. Similarly, stratified K-fold approach (using sklearn's *StratifiedKFold* function) has been used to ensure the same for cross validation.

Benchmark

A benchmark result has been provided for the given classification problem by Moro and Cotez [10] in their research paper.

Classifier	**Naïve Bayes**	**Decision Trees**	**SVM**
Area under ROC	0.87	0.868	0.938

Table 1: benchmark results from research paper by Moro and Cotez [10]

For the purpose of this project, Logistic Regression has been used as the benchmark model to compare the accuracy of other classifiers against it. Reason for choosing Logistic Regression is the simplicity of the model. It has the advantage of fitting models that tend to be easily understood by humans, while also providing good predictions in classification tasks.

III. Methodology

Data Preprocessing

Following are the two datasets that have been used for this project with a detailed description of feature engineering process:

1. ***bank-full.csv***

    a. Using the numeric attribute, "*balance*", a new binary attribute was created that depicts the sign of *balance*. If *balance* is negative then binary attribute is false otherwise it is positive. The binary attribute was added because upon normalization of the numerical attributes it will fall in the range of 0 and 1 and lose information of *balance* being negative or positive.
    b. Numeric attributes are normalized using sklearn's *MinMaxScaler* function.
    c. Binary attributes are converted to either 0 or 1 using sklearn's *LabelEncoder* function.
    d. Categorical attributes (non-binary) are converted to multiple features (depending on the number of categories for that attribute) using pandas *get_dummies* function. (this function basically applies one hot encoding to the attributes and replaces them with a new set of one hot encoded features).

2. *bank-additional-full.csv*

a. Using the numeric attribute, duration, a new binary attribute was created because as per the documentation of the dataset , if duration is 0 then the class is "no" so the new binary attribute is false for that case otherwise it is true. I added this binary attribute because upon normalization of the numerical attribute, it will fall in the range of 0 and 1 and might lose information duration being zero and non-zero.
b. Numeric attributes are normalized using sklearn's *MinMaxScaler* function.
c. Categorical attributes (non-binary) are converted to multiple features (depending on the number of categories for that attribute) using pandas *get_dummies* function (this function basically applies one hot encoding to the attributes and replaces them with a new set of one hot encoded features).

Also, as specified above in the Metrics section, the data is unbalanced as there are 88.3% examples of "no" and 11.7% examples of "yes". This issue was take care of by using *class_weight = "balanced"* parameter in all of the classifiers (except Naïve Bayes Classifiers because they calculated the prior automatically).

Following approaches were applied separately to achieve different sets of features and each classifier (except in case of Recursive Feature Elimination) was trained on these feature sets:

FEATURE SELECTION:

Chi-Square test was also applied using sklearn's *SelectKBest*. Although this resulted in reducing the F1-Score for all the classifiers.

FEATURE TRANSFORMATION:

Principal Component Analysis was used for feature transformation (or dimensionality reduction). This also resulted in reducing the F1-Score for all the classifiers

FEATURE ELIMINATION:

Recursive Feature Elimination approach was used for feature elimination using sklearn's *RFE* function. It removed half of the features. This approach turned out to be a bit better than other approaches as it didn't reduce the scores by much.

Implementation

For implementation, all the dependencies were loaded in the beginning that were required for Data preprocessing, classification and metric evaluation. Since there were two datasets (old and new versions), both were used in different jupyter notebook, mainly because the first notebook was getting too large.

For data preprocessing (as discussed above), *pandas* library was used to store the data in the data-frame and apply transformations (including On Hot Encoding). Then, correlation was checked among the features by calculating the correlation matrix and it was found that a feature pair had a high negative correlation (in case of first dataset) and 5 feature pairs has positive correlation (in case of second dataset) so those features were removed. Finally, the *preprocessing* sub-library of *sklearn* was used for normalization and Label Encoding.

Then for each classifier (mentioned in Algorithms and Techniques section), sklearn's *GridSearchCV* function was used to apply cross validation on different combinations of parameters.

For each classifier, initially used a huge number combinations of hyper parameters were used and it was discovered that certain parameters were redundant (they always gave same results) so some of them were removed in the final version of the implementation, so it would be easier for the reviewer to check them all.

As discussed earlier, F1-Score and Area under Curve was as the evaluation metric. Results were stored in a common data-frame to later compare those results. Later, Accuracy was the added as a metric to show how misleading it can be. (See the case of Gaussian and Multionomial Naïve Bayes). Finally, all the classifiers were compared with each other on the basis of F1-Score and Area under Curve. It was made sure that the test data is used only when the best parameters for the classifier were obtained from GridSearchCV.

Chi Square test, Principal Component Analysis and Recursive Feature Elimination were applied separately to the original set of features to see if there was any change in the metrics.

For the Chi Square Test, sklearn's *SelectKBest* function was used to select the top 25 features because Chi Square score dropped to almost zero after the first 25 features. (Shown in the figure below)

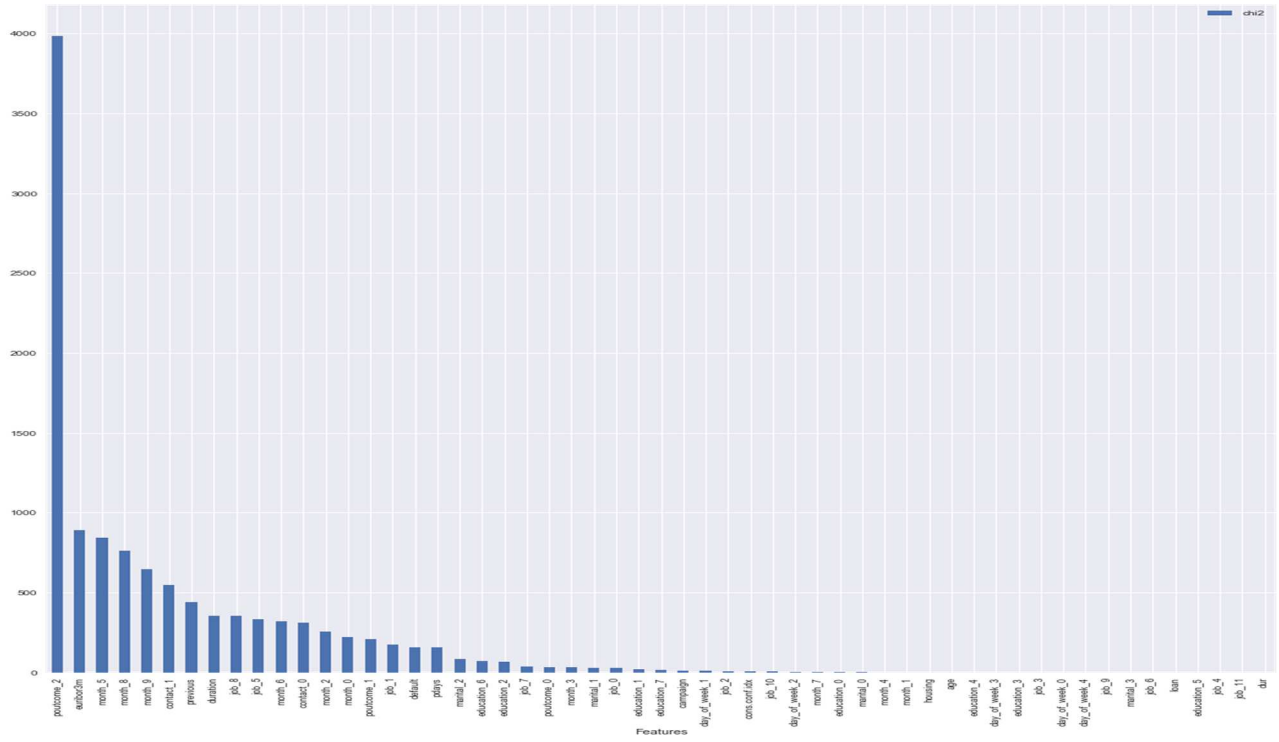

Figure 8: Plot of ChiSquare scores

Above plot is for the second dataset as the number of features are lesser to plot against so it is easier for the reader to look at the plot.

For Principal Component Analysis sklearn's *PCA* function was used to get 30 principal components because the first 30 components amount to a cumulative of 0.95 variance. (Shown in the figure below)

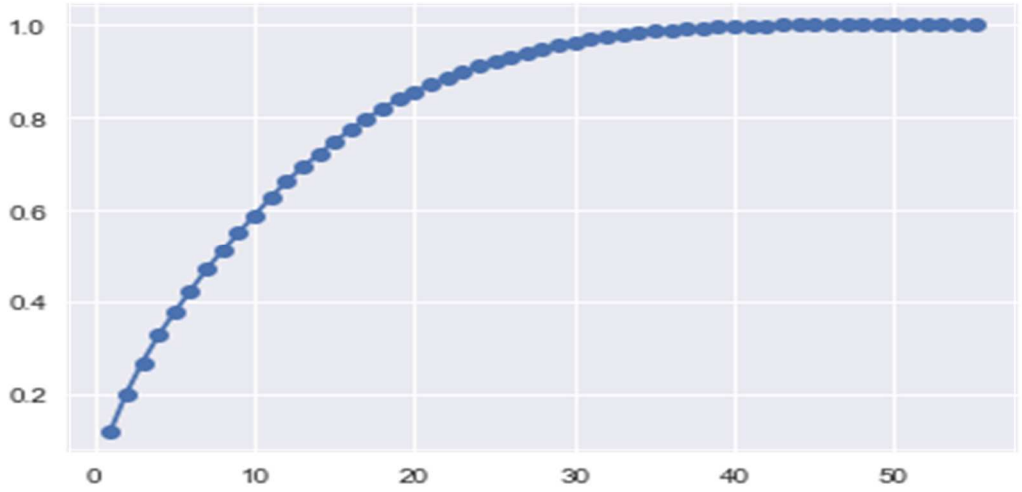

Figure 9: Plot of cumulative variances for the principal components

And for Recursive Feature Elimination, sklearn's *RFE* function was used to recursively eliminate half of the features (by default).

As for the challenges, finding the right sets of parameters for every classifier turned out to be difficult as some classifiers (except Gaussian Naïve Bayes since it didn't require any parameters) took way too much time to fit, for example Support Vector Machine, AdaBoost Classifier and Multi-Layer Perceptron.

Refinement

Initially, one hot encoding was applied to all the categorical attributes (including binary attributes) and later attributes were simply normalized. Then all the classifiers were looped over and *GridSearchCV* was applied on each classifier with a very small set of parameters (very few parameters each with 2-3 values). For example,

Logistic Regression was provided with just one hyper parameter ("max_iter": [50, 200, 500]) in the *GridSearchCV*'s *param_grid*.

Decent results were obtained in the beginning. During this process it was realized that the *stratify* hyperparameter in sklearn's *train_test_split* function should also be used.

After a lot of testing, it was realized that it would be much more efficient if each classifier was tested one by one with a larger set of parameters. And this approach turned out to be much better than changing the set of parameters only after the complete loop executed. For example,

Logistic Regression was provided now being provided with following hyperparameters in the GridSearchCV's param_grid.

```
params = [
        {
            "penalty" : ['l2'],
            "C" : [0.01, 0.1, 1.0, 10.0],
            "solver" : ["sag"],
            "max_iter" : [100, 200]
        },
        {
            "penalty" : ['l1'],
            "C" : [0.01, 0.1, 1.0, 10.0],
            "solver" : ["saga"],
            "max_iter" : [100, 200]
        }
]
```

This approach got slightly better scores as it was able to use more hyperparameters.

Following are the initial and final results:

Classifier	Accuracy	F1-Score	AUC
SVM-RBF	0.840761	0.551122	0.841258
MLPClassifier	0.898596	0.439829	0.657229
MultinomialNB	0.887648	0.367372	0.624149
GaussianNB	0.866969	0.438113	0.684424
DecisionTreeClassifier	0.814995	0.501638	0.809310
Logistic Regression	0.899259	0.411879	0.640645
AdaBoost	0.901802	0.459196	0.666075
Random Forest	0.887427	0.480612	0.696824

Figure 10: Screenshot of Initial Results

Following parameter grid was used to obtain the Initial Results:

```
{"GaussianNB":{},
 "Random Forest": {'n_estimators': [10, 20, 50], 'max_features': ['sqrt', 'log2'], "criterion": ["entropy", "gini"]}
 "AdaBoost": {"learning_rate": [0.1, 0.3, 1, 3], "n_estimators" : [50, 100, 500]},
 "Logistic Regression": {"max_iter": [50, 200, 500]},
 "DecisionTreeClassifier": {"max_depth": [5, 15, 25], "criterion": ["entropy", "gini"]},
 "MultinomialNB": {"alpha" : [0, 1, 2]},
 "MLPClassifier":{"hidden_layer_sizes" : [(10, 10, 10), (30, 30, 30)], "activation": ['logistic', 'relu'],
                  "solver" : ['sgd', 'adam']},
 "SVM-RBF": {"kernel": ["poly", "rbf"], "C":[0.1, 1, 5]}
}
```

Figure 11: Screenshot of parameter grid used to obtain initial results

Classifier	F1-Score	AUC	Accuracy
Support Vector Machine	0.564281	0.859661	0.847779
Multi Layer Perceptron	0.576651	0.744386	0.912843
Multinomial Naive Bayes	0.383728	0.713640	0.766448
Gaussian Naive Bayes	0.420401	0.726068	0.806871
AdaBoost	0.505605	0.715294	0.892935
Random Forest	0.652889	0.839851	0.910294
Decision Tree	0.577554	0.817894	0.878004
Logistic Regression	0.584649	0.860548	0.862709

Figure 12: Screenshot of Final Results

Following parameter grids were used to obtain the Final Results along with their corresponding best parameters:

Logistic Regression

```
params = [
            {
                "penalty" : ['l2'],
                "C" : [0.01, 0.1, 1.0, 10.0],
                "solver" : ["sag"],
                "max_iter" : [100, 200]
            },
            {
                "penalty" : ['l1'],
                "C" : [0.01, 0.1, 1.0, 10.0],
                "solver" : ["saga"],
                "max_iter" : [100, 200]
            }
]
```

Figure 13: Screenshot of parameter grid of Logistic Regression

Best parameters: {'C': 1.0, 'max_iter': 200, 'penalty': 'l1', 'solver': 'saga'}

Decision Tree Classifier

```
params = {
            "criterion" : ["gini", "entropy"],
            "splitter" : ["best", "random"],
            "max_depth" : [25, 50, 75, None],
            "min_samples_split" : [3, 6],
            "min_samples_leaf" : [2, 4],
        }
```

Figure 14: Screenshot of parameter grid of Decision Tree Classifier

Best parameters: {'criterion': 'gini', 'max_depth': 40, 'min_samples_leaf': 4, 'min_samples_split': 3, 'splitter': 'best'}

Random Forest Classifier

```
params = {
            "n_estimators" : [100, 200],
            "criterion" : ["gini", "entropy"],
            "max_depth" : [25, 50, 75],
            "min_samples_split" : [3, 6],
            "min_samples_leaf" : [2, 4]
        }
```

Figure 15: Screenshot of parameter grid of Random Forest Classifier

Best parameters: {'criterion': 'entropy', 'max_depth': 40, 'min_samples_leaf': 2, 'min_samples_split': 6, 'n_estimators': 200}

AdaBoost Classifier

```
params = {
           "base_estimator": [DecisionTreeClassifier(class_weight = "balanced"),
                              LogisticRegression(n_jobs = -1, class_weight = 'balanced', penalty = 'l2', solver = 'sag')],
           "n_estimators" : [500, 1000],
           "learning_rate" : [0.01, 0.1, 1.0]
       }
```

Figure 16: Screenshot of parameter grid of AdaBoost Classifier

Best parameters: {'base_estimator': DecisionTreeClassifier(class_weight='balanced', criterion='gini', max_depth=None, max_features=None, max_leaf_nodes=None,min_impurity_decrease=0.0, min_impurity_split=None, min_samples_leaf=1, min_samples_split=2, min_weight_fraction_leaf=0.0, presort=False, random_state=None, splitter='best'), 'n_estimators': 1000, 'learning_rate': 1.0}

Multinomial Naive Bayes

```
params = {
            "alpha" : [0.0, 0.1, 0.5, 1.0],
            "fit_prior" : [True, False]
          }
```

Figure 17: Screenshot of parameter grid of Multinomial Naïve Bayes

Best parameters: {'alpha': 0.1, 'fit_prior': False}

Multi Layer Perceptron

```
params = [
            {
                "hidden_layer_sizes" : [(50, ), (50, 50), (50, 50, 50)],
                "solver" : ["adam", "lbfgs"],
                "alpha" : [0.0001, 0.001],
                "max_iter" : [100, 200]
            },
            {
                "hidden_layer_sizes" : [(50, ), (50, 50), (50, 50, 50)],
                "solver" : ["sgd"],
                "alpha" : [0.0001, 0.001],
                "learning_rate" : ["constant", "invscaling", "adaptive"],
                "max_iter" : [100, 200]
            }
]
```

Figure 18: Screenshot of parameter grid of Multi-Layer Perceptron

Best parameters: {'hidden_layer_sizes': (50, 50), 'solver': 'lbfgs', 'alpha': 0.0001, 'max_iter': 200}

Support Vector Machine

```
params = [
            {
                "C" : [0.01, 0.1, 1.0],
                "kernel" : ["poly"],
                "degree" : [2, 3, 4],
                "gamma" : [0.001, "auto", 0.1]
            },
            {
                "C" : [0.01, 0.1, 1.0],
                "kernel" : ["rbf", "sigmoid"],
                "gamma" : [0.001, "auto", 0.1]
            },
            {
                "C" : [0.01, 0.1, 1.0],
                "kernel" : ["linear"]
            }
        ]
```

Figure 19: Screenshot of parameter grid of Support Vector Machine

Best parameters: {'C': 1.0, 'degree': 2, 'gamma': 0.2, 'kernel': 'poly'}

Initially the F1-scores were pretty low but after a lot of fine tuning, tiny change in feature engineering and using more hyper parameters gave a lot better result.

IV. Results

Model Evaluation and Validation

After numerous tests on different sets of features (feature sets produced by feature selection, transformation and elimination), it was found that Random Forest Classifier is the most accurate model (see figure 5: Final Results). It outperforms all the classifiers.

Classifier	**Training F1-Score**	**Testing F1-score**
Support Vector Machine	0.5826396851437704	0.564281
Multi Layer Perceptron	0.57192544273230639	0.576651
Multinomial Naive Bayes	0.39393712706220074	0.383728
Gaussian Naive Bayes	0.4333456282447059	0.420401
AdaBoost	0.51081107455780972	0.505605
Random Forest	0.64672100335837801	0.652889
Decision Tree	0.57732955773180039	0.577554
Logistic Regression	0.59012566364638652	0.584649

Table 2: F1-scores on Training (Validation) dataset and Test set

For validation, sklearn's *GridSearchCV* function was used to perform cross validation to get the best parameters.

It was also observed that upon feature selection process, Random Forest Classifier wasn't affected much which establishes the fact that Random Forest Classifier is robust.

Feature Sets	Training F1-Score	Testing F1-Score
Original Features	0.64672100335837801	0.652889
After Feature Selection (ChiSquare Test)	0.63742906374203101	0.638456

Table 3: Training and Testing F1-Scores for Random Forest Classifiers

Finally, the best model selected (Random Forest Classifier) is validated by training and testing model with multiple different random states

Random State	F1-Score	AUC	Accuracy
9.0	0.654240	0.842939	0.909930
8.0	0.655399	0.841604	0.910901
7.0	0.653700	0.837644	0.911386
6.0	0.652869	0.839449	0.910415
5.0	0.650140	0.839235	0.909201
4.0	0.658287	0.845838	0.910901
3.0	0.650260	0.836088	0.910294
2.0	0.654100	0.839723	0.910901
1.0	0.654801	0.841869	0.910537
0.0	0.660422	0.845513	0.911993

Figure 20: Screenshot of Test set scores when best model is trained using different random states

	count	mean	std
F1-Score	10.0	0.654422	0.003175
AUC	10.0	0.840990	0.003183
Accuracy	10.0	0.910646	0.000771

Figure 21: Screenshot of mean and standard deviation of Test set scores

From the above screenshot it is clear that there isn't much variance in scores.

Justification

As can be seen from the section discussed above and the Benchmark section, the AUC score of our best model isn't able to outperform the benchmark model but does significantly well against the Logistic Regression Model (proposed benchmark model) and other classifiers that were used for this project like Gaussian and Multionmial Naïve Bayes, Decision Tree Classifier, AdaBoost Classifier, Support Vector Machine and MultiLayer Perceptron.

V. Conclusion

Free-Form Visualization

For each dataset, a bar plot has been created to compare the results of all the classifiers. Although, feature selection, transformation and elimination were applied but they didn't approve the F1-scores and the Area under ROC. So I plotted the results for classification on original set of features.

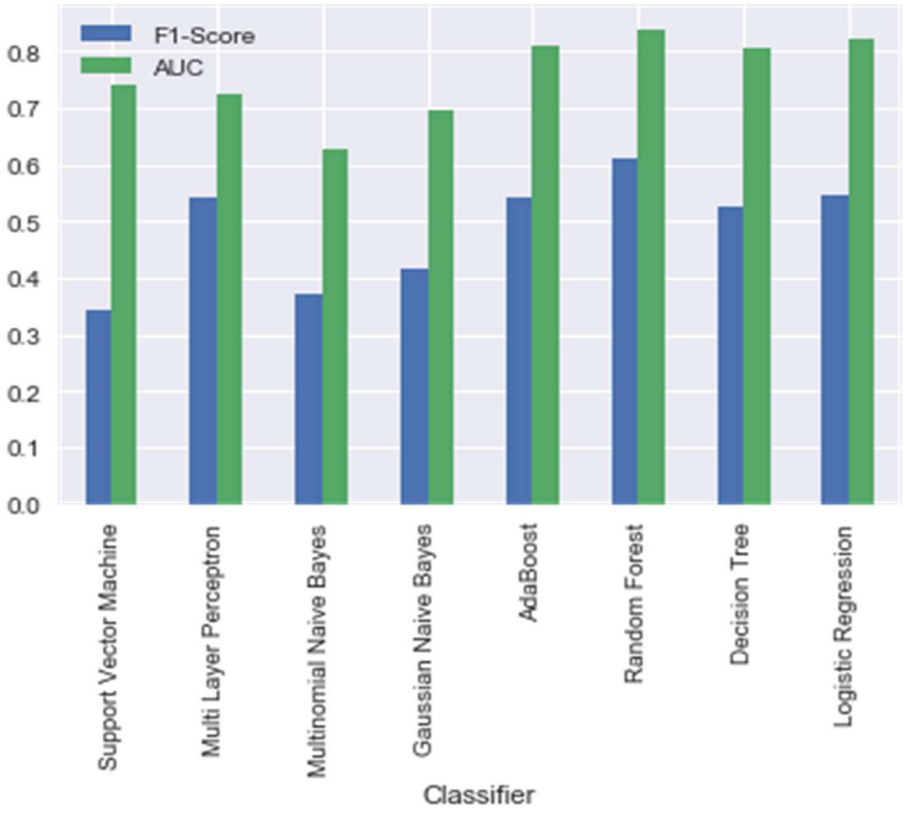

Figure 19: Results on first dataset (bank.csv)

Reflection

This project took a Bank Marketing classification problem and applied multiple classification and feature selection/transformation techniques to solve the problem at hand. Grid Search Cross Validation was also used during the course of this project for all classifiers to find bets parameters and do K-fold cross validation.

After completing this project, I have realized the importance of feature engineering and repeated testing in order to find best parameters. I have understood how hard it is to get an increase of 1% in scores. Over the course of project, I learnt how to do feature engineering and efficiently find best hyper parameters for classifiers.

Improvement

After concluding this project, I still believe that may be some improvements can be done in this project:

- The amount of computation for this project is huge when it comes to using *GridSearchCV* for classifiers like Support Vector Machine, Multi-Layer Perceptron and AdaBoost Classifier. As I result I wasn't able to test all of hyperparameter combinations for them because if possible it would take days to run even when using all cores (using *n_jobs = -1*). If possible I would try those many hyper parameter combinations to get best parameters for those classifiers.
- Using lesser number of features (feature selection or transformation) resulted in a decrease in scores for all the classifiers. I think that maybe more features can be engineered from the given data. For example, continuous attribute "*age*" can be converted into many categorical attributes by taking age slots (say of 10 years).
- Apart from using hyperparameter *class_weight = "balanced",* maybe we can use SMOTE (Synthetic Minority Over-sampling Technique) sampling. As the name suggests, this technique creates synthetic examples using the dataset.
- Anomaly/Novelty Detection Algorithms like Isolation Forest, One Class SVM, etc can be used in this case since the data is unbalanced.
- A Ensemble of all classifiers used in the project can be made which chooses the class that has the votes from majority of the classifiers.

References

[1] http://archive.ics.uci.edu/ml/datasets/Bank+Marketing

[2] Ling, X. and Li, C., 1998. "Data Mining for Direct Marketing: Problems and Solutions". In Proceedings of the 4th KDD conference, AAAI Press, 73–79.

[3] Ou, C., Liu, C., Huang, J. and Zhong, N. 2003. “On Data Mining for Direct Marketing”. In Proceedings of the 9th RSFDGrC conference, 2639, 491–498.

[4] Page, C. and Luding, Y., 2003. “Bank manager’s direct marketing dilemmas – customer’s attitudes and purchase intention”. International Journal of Bank Marketing 21, No.3, 147–163.

[5] Hu, X. 2005, “A data mining approach for retailing bank customer attrition analysis”, Applied Intelligence 22(1):47- 60

[6] Li, W., Wu, X., Sun, Y. and Zhang, Q., 2010. “Credit Card Customer Segmentation and Target Marketing Based on Data Mining”, In Proceedings of International Conference on Computational Intelligence and Security, 73-76.

[7] Zhang, H. 2004. “The Optimality of Naïve Bayes”, In Proceedings of the 17th FLAIRS conference, AAAI Press.

[8] Aptéa, C. and Weiss, S. 1997. “Data mining with decision trees and decision rules”, Future Generation Computer Systems 13, No.2-3, 197–210.

[9] Cortes, C. and Vapnik, V. 1995. “Support Vector Networks”, Machine Learning 20, No.3, 273–297.

[10] S. Moro, P. Cortez and P. Rita. A Data-Driven Approach to Predict the Success of Bank Telemarketing. Decision Support Systems, Elsevier, 62:22-31, June 2014