UNIVERSITY OF EXETER | BUSINESS SCHOOL

Bank Marketing Analysis

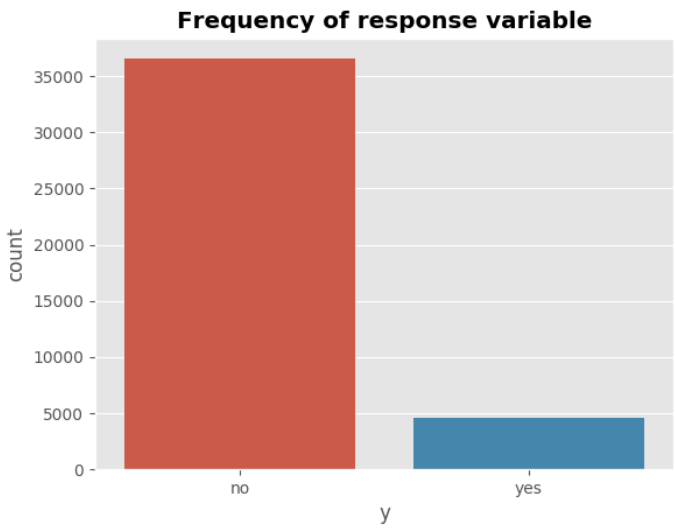
Introduction

This report seeks to analyse data from a marketing campaign of a Portuguese bank to establish whether a potential customer will deposit money or not. This is a classification problem with a binary dependent (response) variable, highlighting the event of a customer deciding or not to subscribe to the banking services. Such problem could be of great interest to professionals as it could identify the key characteristics of consumers who are likely to subscribe to such banking services, aiding in the management of budgets and resulting in potential savings from the marketing department. Furthermore, the insights given by such a study could help professionals with similar marketing strategies when predicting if a certain customer is likely to purchase a product as well as the demographics of the main user groups.

Data Used and Variables Selected

The data used in this report is retrieved from UCI Machine Learning Repository[1]. I will use *'bank-additional-full.csv'* from 'Bank Marketing Dataset' as it is the most recent and complete data available (with over 40,000 observations and 21 features) to download.

The response variable is binary (yes/no). Of the remaining 20 independent variables, 9 are categorical and will need to be transformed into dummy variables before being applied to the machine learning models. There are no null values present in the dataset, although some categorical attributes (such as marital status) are labelled as unknown– probably due to customers not being willing to share the information. The variable *pdays* has a lot of missing values as most of the customers were not contacted previously. It will be changed into a binary variable *previously_contacted*, where 0 will stand for a customer who has not been contacted previously and 1 for a customer who has been.



The initial analysis could be observed from the figure beside. As shown, the response variable is highly unbalanced, with most customers deciding not to subscribe to the banking services offered by the marketing campaign.

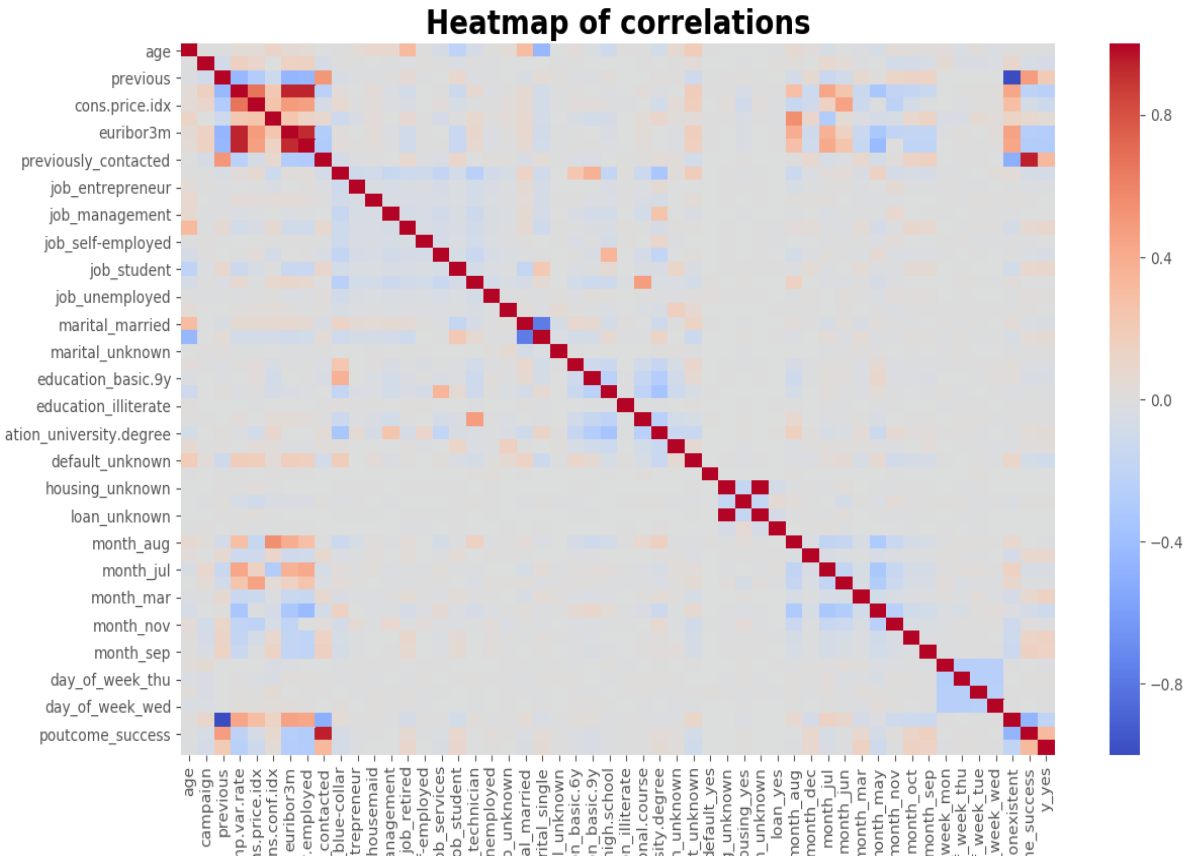
This means that normal measures of identifying the accuracy of a classification model (such as accuracy score or proportion of correct predictions) won't be very

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

effective at tackling the problem as they will lead to inflated scores due to prevalence of 'no's in the response. We will instead use roc-auc curves and auc scores, which give us information about how much the model is capable of distinguishing between its different classes.

From the analysis of the different features, we can decide to exclude the 'contact' column as we can clearly see all of the customers were successfully contacted as well as the 'duration' column. Although we can tell that the duration of each call is highly relevant to a customer deciding to subscribe to the banking services, it is information that is not available before each customer is contacted, meaning it will not aid in creating a predictive model.

When analysing the correlations of each variable we can observe the following heatmap. We can see the most important independent variables on the outcome to be the euribor 3 month rate (-0.308), the employment variation rate (-0.298), the number of contacts for this client before the campaign (0.230), whether the customer was previously contacted or not (0.325), whether the previous outcome was successful or not (0.316) and the employment rate (-0.355).

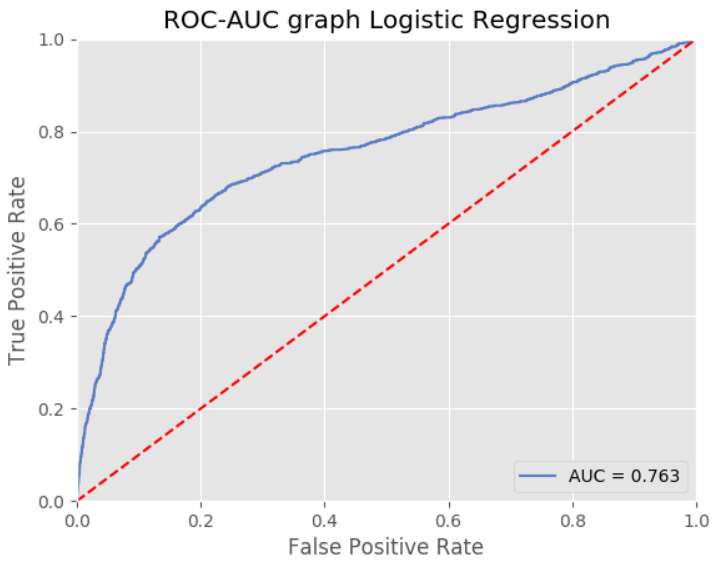


UNIVERSITY OF EXETER | BUSINESS SCHOOL

Machine Learning Models Used

Logistic Regression

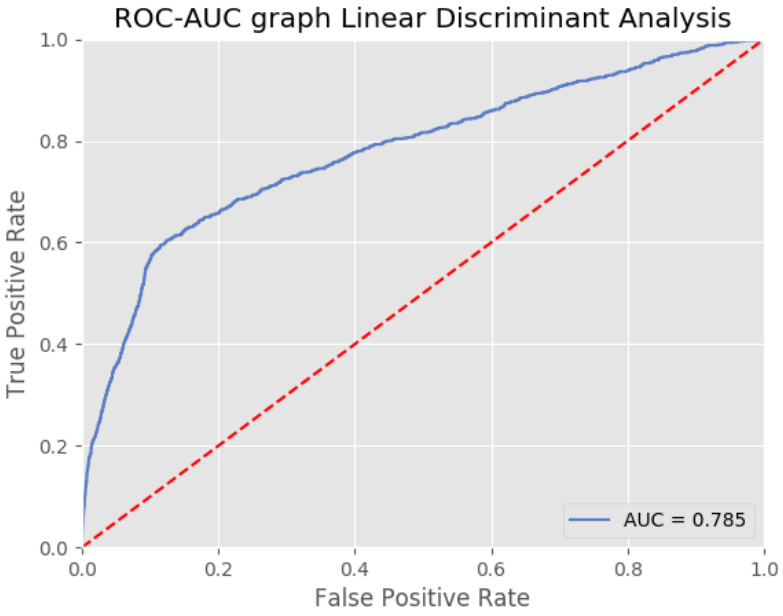
When a logistic regression model is applied on the dataset, we can observe an accuracy score of 89.4%. This score is based on a 10-fold cross validation of the training and test set and represents the percentage of correctly classified points in the test set. While this may appear very high it is deceiving as the classes of the dependent variable are highly unbalanced. This is why the results are better represented in terms of auc score of the roc-auc curve. A logistic regression produces a roc-auc score of 0.763, shown in the graph on the side.



Logistic Regression is a very strong model because if its ability of being highly interpretable and computationally efficient. It is also considered simpler as parameter tweaking doesn't take place, making this model very understandable.

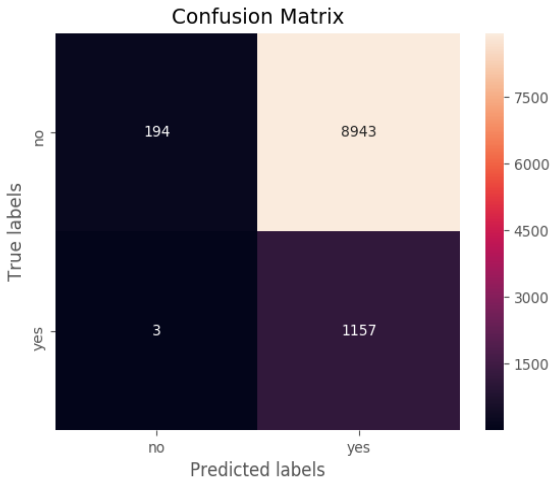
Linear Discriminant Analysis (LDA)

When the dataset is fitted with Linear Discriminant Analysis, we can observe a very similar accuracy level to Logistic Regression with 88.8% of accurately identified datapoints. The ROC-AUC score, however, is slightly higher, reaching 0.785 and is represented in the following plot. The methodology to conduct Linear Discriminant Analysis is computationally efficient and doesn't take many input parameters, meaning it is an efficient and interpretable model to run.



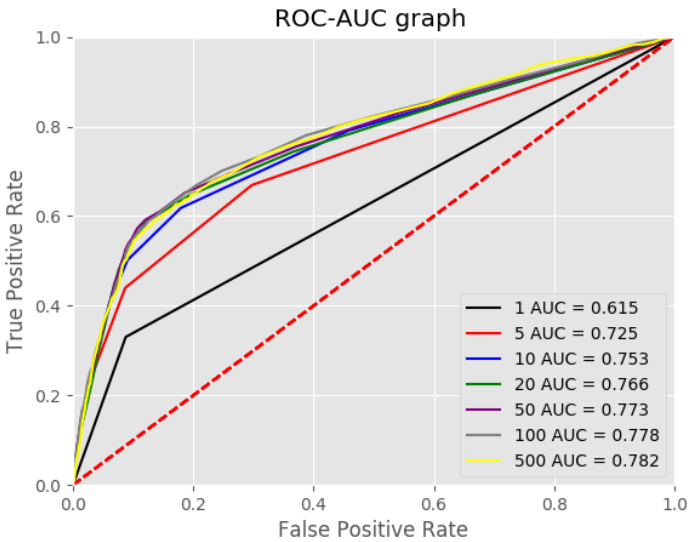
Quadratic Discriminant Analysis (QDA)

Quadratic Discriminant Analysis, differently from Linear Discriminant Analysis assumes a quadratic decision boundary instead of a linear one. When the dataset is fitted with such model, we can observe a much lower accuracy score of 13%. This because the model is predicting most of the observations to be 'yes's when instead most of them are noes. Such information can be observed in the confusion matrix of predictions. Although the ROC-AUC score is of 0.779, we will exclude this model's performance as its accuracy rate is incredibly low. This may be due to the decision boundary being quite linear or to the very unbalanced response variable. We can conclude that Quadratic Discriminant Analysis is ineffective at predicting the response variable in such dataset.



K-Nearest Neighbors

The K-Nearest Neighbors classifier was tested with values of k equal to 1, 5, 10, 20, 50, 100, 500. A K-Nearest Neighbors model with k=1 is expected to yield very poor results, this is generally due to overfitting of the data points. From such model we can observe an accuracy rate of 84.8%, which isn't so low compared to the previously tested models but is an inefficient testing measure in this case due to the disproportionality of the dependent variable (as previously mentioned).

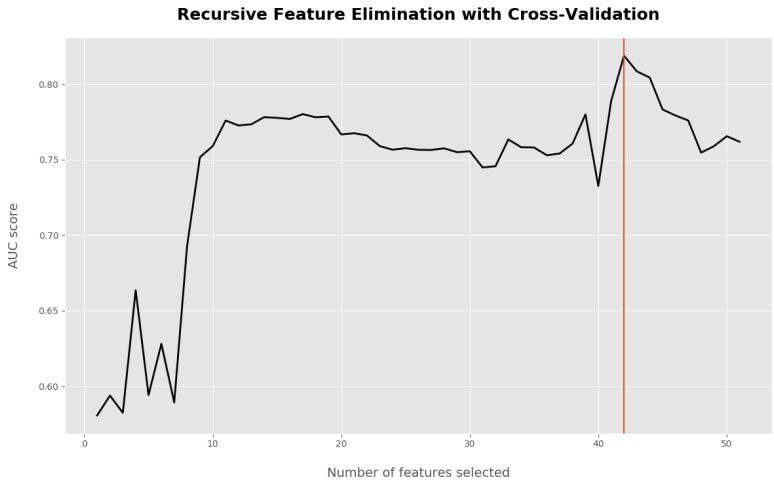


The ROC-AUC score for a K-Nearest Neighbors model with k=1 is 0.615, which is very low compared to the previous models and reaches a peak of 0.782 with k=500. This is shown in the graph beside and is still lower compared to previously tested models such as Linear Discriminant Analysis, showing KNN is not a very efficient algorithm to analyse this dataset.

UNIVERSITY OF EXETER | BUSINESS SCHOOL

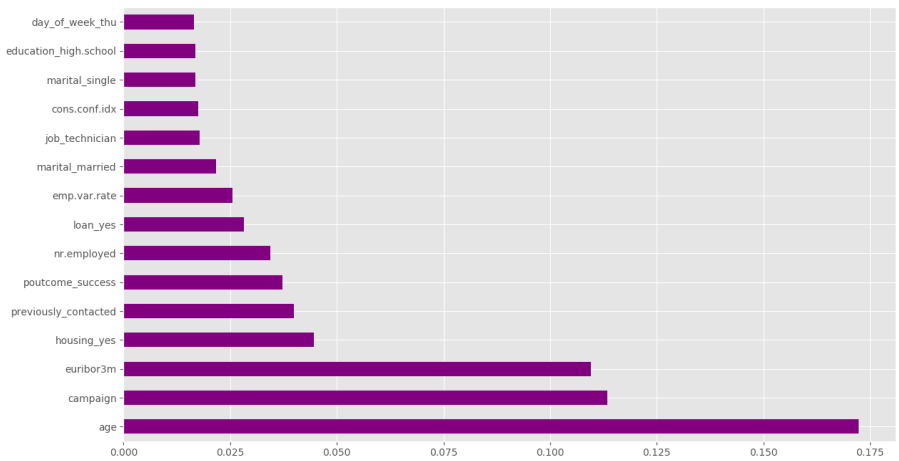
Best Subset Selection

Best Subset Selection is conducted to find the optimal number of features in a model to maximise its effectiveness. This will be conducted through the use of Recursive Feature Elimination and *feature_importance*, a function of the Extra Trees Classifier algorithm. Recursive Feature Elimination works by removing the weakest features until a specific number of features is reached, the scoring method to assess the accuracy of each model will be roc-auc and the model fitted to estimate performance will be a Logistic Regression model. We can observe the optimal number of features is 42, which leads to a roc-auc score of 0.82. This is higher than the original Logistic Regression model of 0.763, showing a reduced number of features has improved the performance of such model. The features deemed less effective by the algorithm were *age*, *campaign*, *previous*, *cons.price.idx*, *nr.employed*, *job_unknown*, *marital_unknown*, *education_illiterate* and *default_yes.* While it is more obvious for features such as *job_unknown* and *marital_unknown* to be less effective in a predictive model as they represent unknown values, variables such as *age* and *nr.employed* are interesting exclusions by the algorithm. It would be expected for age to play a big part on whether a person is more likely to open a bank account, due to the supposition of increased economic stability with an increased age.



The 'feature_importances_' is a property of the many machine learning models and it can be used to determine the most important features. We will observe results from the Extra Trees Classifier algorithm to assess the most important features in the dataset.

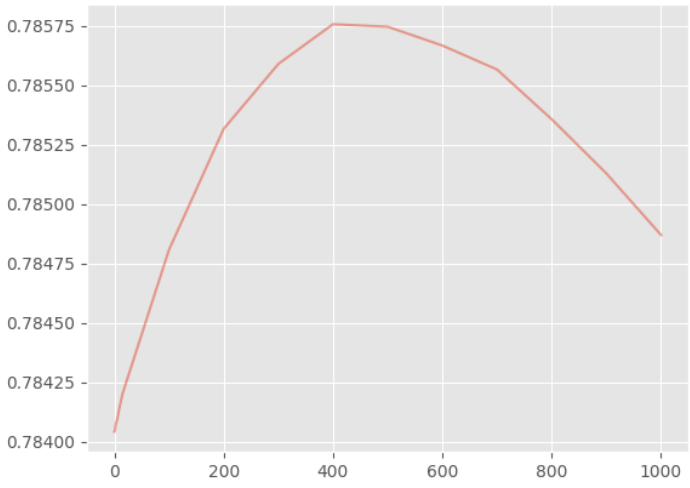
The Extra Trees Classifier establishes *age*, *campaign* and *euribor3m* to be the most important features in the dataset (contrasting the Recursive Feature Elimination results which deemed age was not an important variable). This difference may be due to feature importance being heavily based on accuracy and as previously mentioned the dataset is unbalanced and



accuracy as a measure will not perform well.

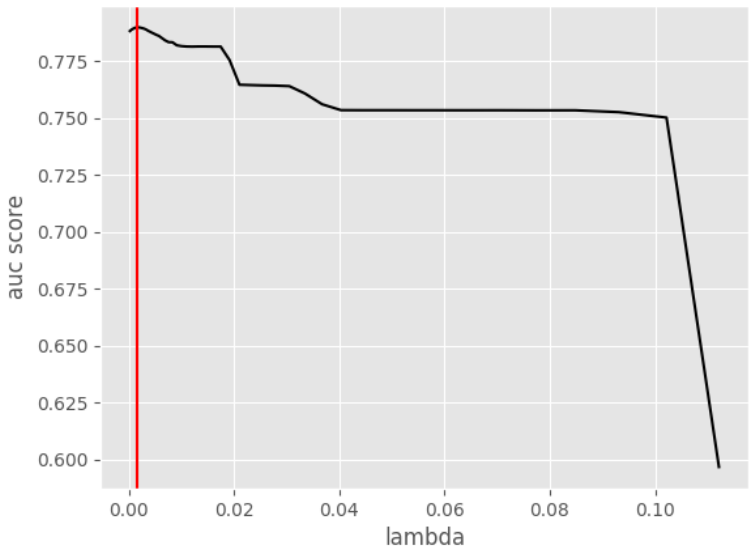
Ridge and Lasso

Ridge and Lasso are model regularisation techniques which excel in models with a large number of features (which occurs in the dataset we are analysing). Ridge performs a regression and adds a penalty equivalent to the square of the magnitude of coefficients. The model will be optimised by varying the *alpha* parameter, with an increase in *alpha* the model moves away from an ordinary least squares model (meaning a Ridge Regression with *alpha=0* is an Ordinary Least Squares Regression). When testing the model with *alpha* values between 0.001 and 1000 (with intervals of 100 between 100 and 1000) we can observe the best result from a model with *alpha=400* with an auc score of 0.786.



Lasso is another shrinkage method which is able to shrink variables to exactly zero (while Ridge was able to only shrink them close to zero), this is tuned through the lambda parameter. The dataset is fitted to the

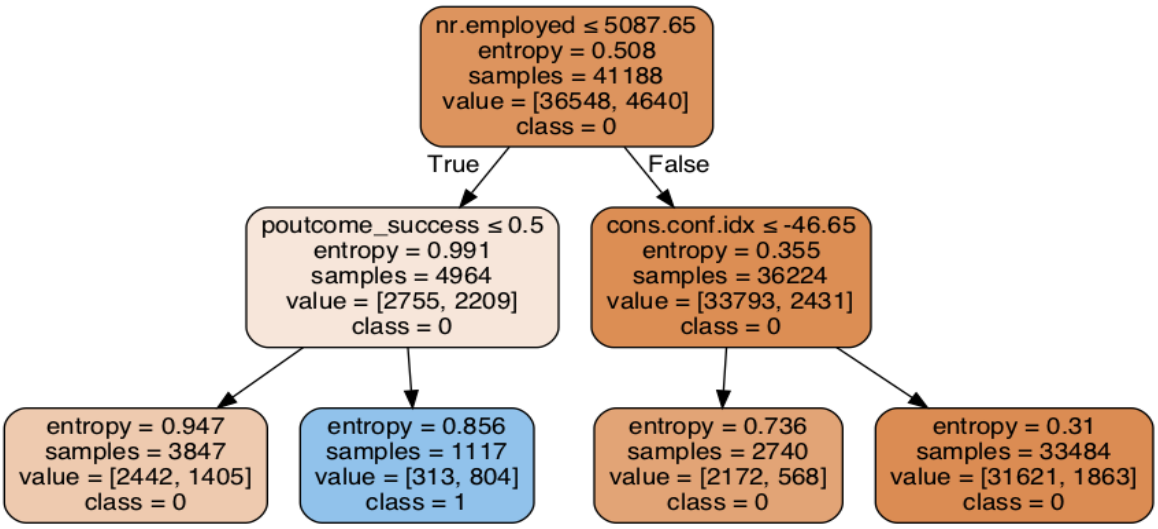
Lasso model and is tuned for the lambda parameter. By analysing the auc scores given by 100 different *lambdas* we can see the model's performance is maximised when *lambda*=0.0017, this makes the model achieve an auc score of 0.790.



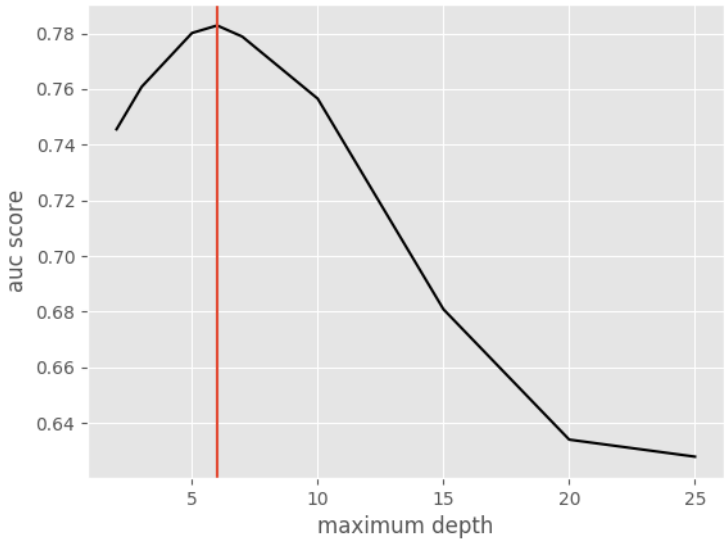
Decision Tree Classifier

Data points were fitted with a Decision Tree Classifier, where the number of branches of each tree would be optimised to result in the highest auc score (this is done by tuning the tree's max depth).

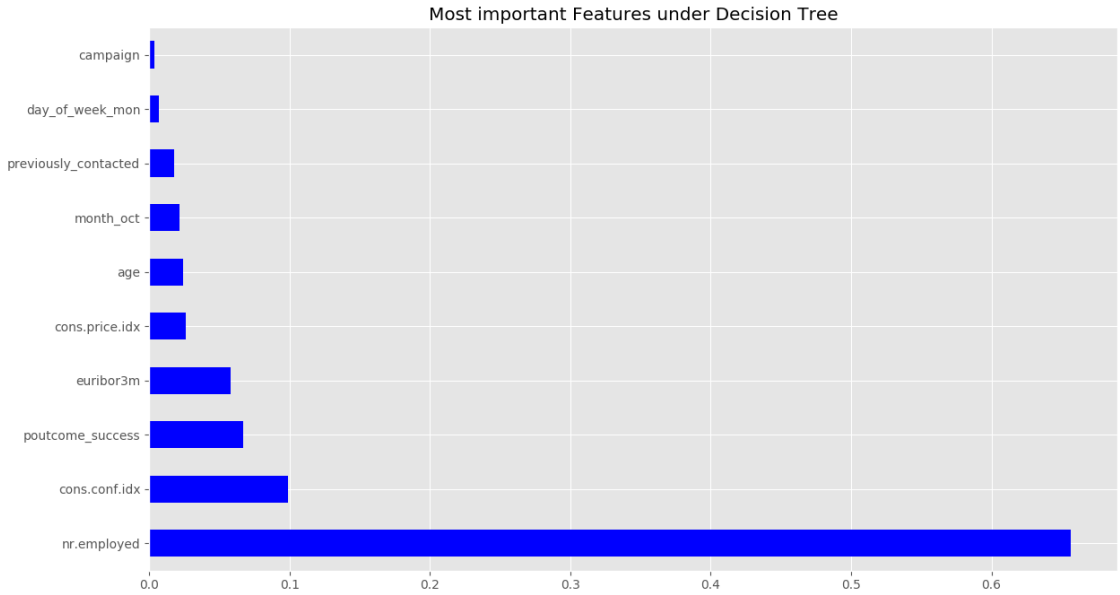
Decision Trees are very good machine learning models due to the ease at which they can be explained to third parties, often much more than other models like Ridge or Lasso. This makes them a perfect fit for the readers of such report who don't have high mathematical or statistical knowledge, as decisions taken by Decision Trees share similar characteristics to the human decision-making process. Furthermore, their graphical representation aids in understanding. The initial decision tree will be fitted with a maximum depth of 2, which results in an auc score of 0.746, lower than most of the previously described models.



Results are consequently tested with different tree depths, with the optimal one being 6 and leading to an auc score of 0.783, which can be comparable to the previously conducted models. Although decision trees are often criticised due to their less effective accuracy through pruning we were able to reach similar results to models like Ridge. From the graph we can also observe that the tree performs poorly with very large maximum depths (a tree with a maximum depth of 25 has a very low auc score of 0.628).



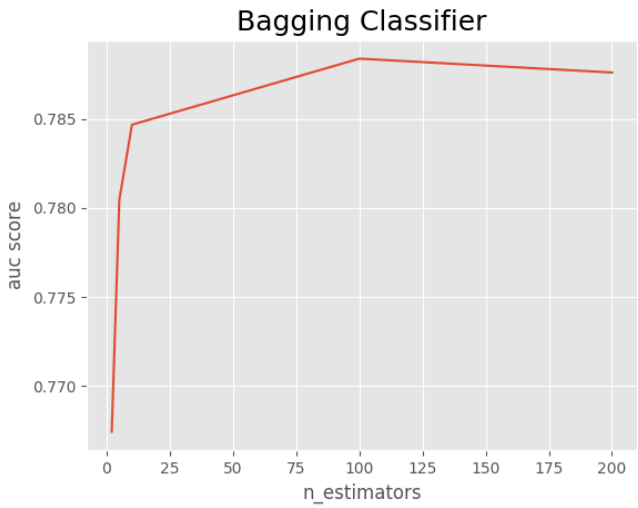
Decision Trees have the ability to rank feature importance in their algorithm, a tool which can greatly aid machine learning models and interpretability of the data. When assessing feature importance of a tree model with maximum depth of 6 (the previously established optimal choice with highest auc score), we achieve the following graph, highlighting the most important predictor variables.



Nr.employed refers to the employment rate and is expressed as a numeric quarterly indicator. As the employment rate is directly correlated to economic stability and the likelihood of a customer subscribing to a banking service, it is considered as a highly important feature in the model. Consumer confidence index (cons.conf.idx) and the European Interbank Offered Rate (euribor3m) are also strong indicators determining whether a customer is likely to utilise such banking services.

Bagging

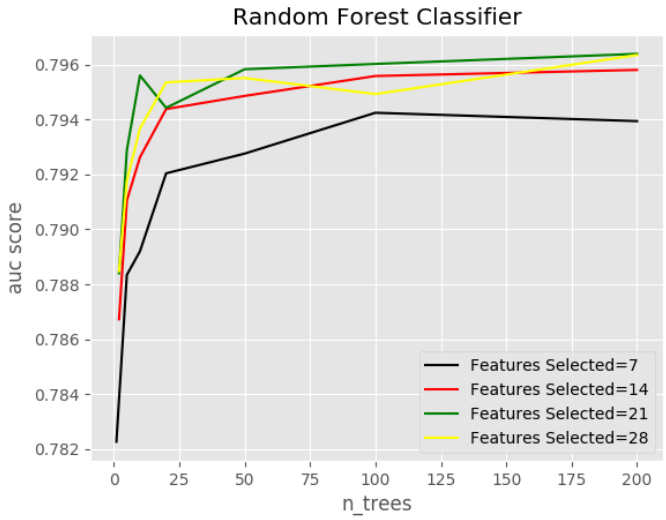
Bagging refers to Bootstrap aggregation and is used to reduce variance in a machine learning method. When applied to a Decision Tree Classifier, results are expected to show a decreased variance and a higher accuracy due to the minimisation of data points overfitting.



Auc scores will be calculated for different models, varying in the number of estimators each model has (results will be shown for estimators ranging between 2 and 200). The highest auc score is given by a Bagging Classifier with 100 estimators(trees), resulting in an auc score of 0.788. The results show the model performs slightly better than a normal Decision Tree model due to Bagging being an ensemble machine learning method which combines many trees together, however it can be criticised due to its poor interpretability. Bagging also utilises much more computational power than a normal Decision Tree because it is performing the model with a large number of estimators, each of which is a Decision Tree.

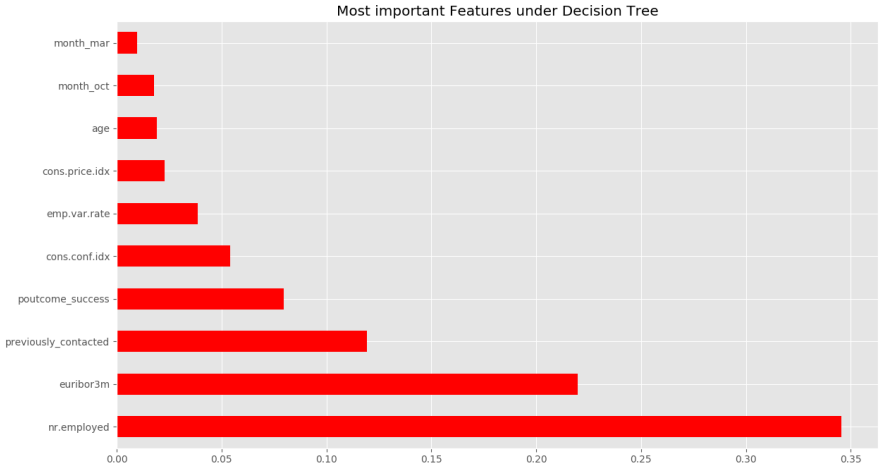
Random Forest

A Random Forest is a decision-making algorithm that consists of many Decision Tree Classifiers and is an improvement over bagging as it leads to decorrelation of the trees. This works by taking a sample out of the total number of predictors. The typical amount chosen is the square root of the total number of predictors, in this case 7 as the total number of features is 51. A Random Forest with the total number of predictors is equal in performance to a Bagging model. The graph below shows results from Random Forests with different number of prediction parameters. The highest auc score is produced is 0.796 by a model with 21 selected features (green line on the graph), contradicting the assumption that usually the optimal number of selected features is the square root of the total features. From the graph we can clearly see this is not the case, as the curve following this assumption (black curve) is the one which performs the worst.



In comparison with a Bagging Classifier, a Random Forest Classifier performs significantly better, likely due to the decorrelation of the trees and the subset of features selected. Random Forests are also great predictors as they are unaffected by outliers. However, this model could be criticised as similarly to Bagging it utilises a high amount of computational power and is considered poor in interpretability.

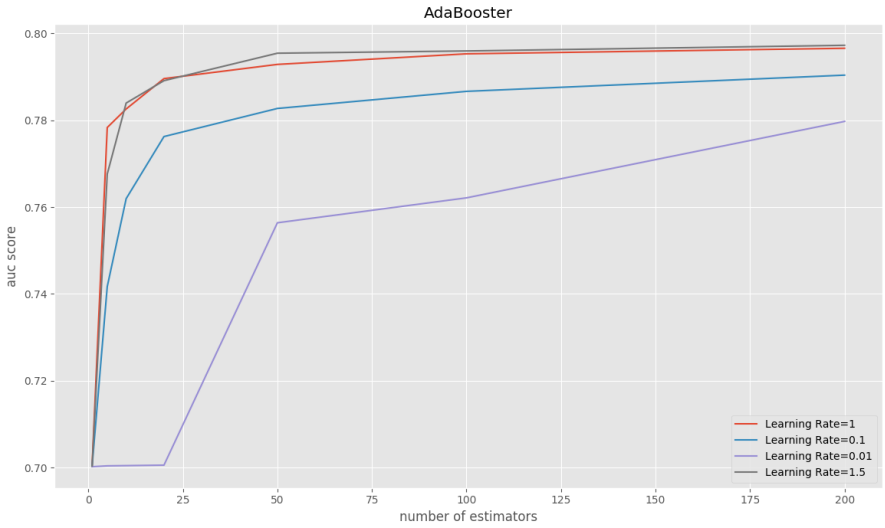
Like with a normal decision tree, we can assess the most important features the model uses for predictions. Nr.employed is again the most important feature the model is selecting, this time placing more emphasis on euribor3m and previously_contacted. Previously_contacted being selected as a key feature makes sense as customers contacted multiple times may be more likely to subscribe to the banking services advertised in the campaign. The model shows poutcome_success to be a key feature in its prediction, which also makes sense as if the campaign was previously successful with a customer it is likely to be again.



UNIVERSITY OF EXETER | BUSINESS SCHOOL

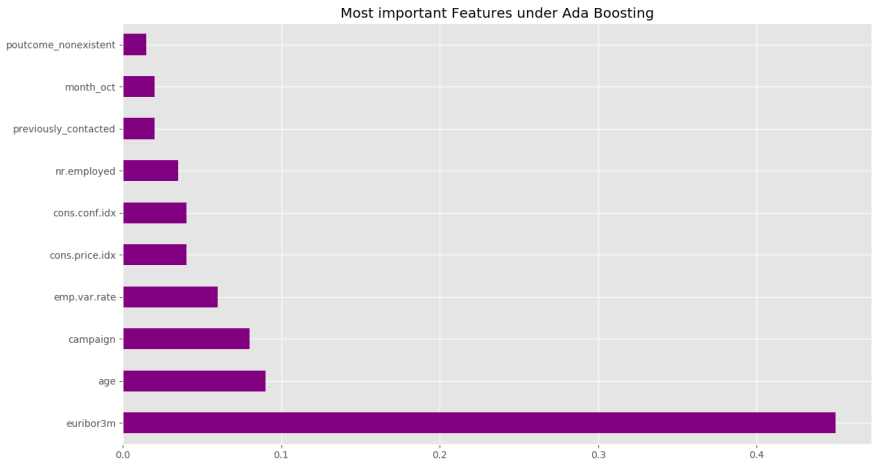
Boosting

Boosting are tree models that grow the trees slowly and each tree is grown based on information found on previous trees, their aim is to transform weak learner data points into stronger ones. The speed at which these trees grow is through the learning rate parameter (which will be tuned). The type of boosting used will be the AdaBooster algorithm (adaptive boosting algorithm), which works by assigning weightings to classifiers and data points with the objective of correctly classifying observations that would originally appear very difficult to classify. Results will be observed with different learning rates of 0.01, 0.1, 1 and 1.5 and the correspondent auc scores are plotted in the graph above.



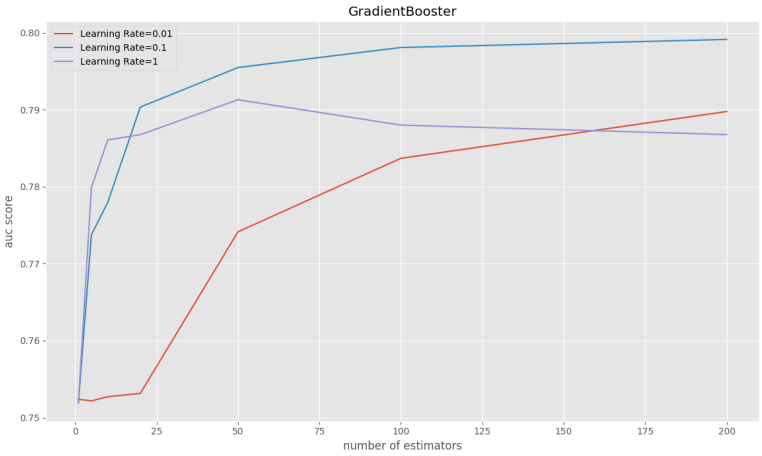
We can observe the best results are achieved with a learning rate of 1.5 (which slightly outperforms a learning rate of 1) and produce an auc score of 0.797. This is slightly below the results observed from the Random Forest algorithm.

An advantage of a boosting model is that it uses the process of ensemble learning, meaning it combines a variety of weak models into a strong one. When assessing feature importance, an AdaBooster model shows euribor3m, age and campaign. Age and campaign are two features which previously didn't appear as important in Decision Trees and Random Forests.



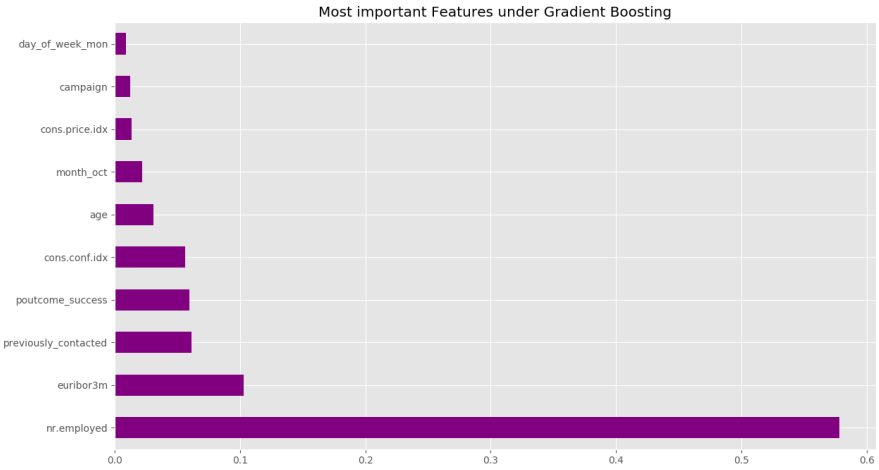
Gradient Boosting

A Gradient Boosting model is similar to an Ada Boosting one, meaning it aims to turn weak predictors into stronger ones, this time by reducing error rates sequentially (after the first model is ran the error which is remaining will be assessed and reduced by the second model and this continues for each number of models). When three different Gradient Boosting models are fitted with learning rates of 0.01, 0.1 and 1, we can observe the highest results from a model with a 0.1 learning rate and 200 estimators, reaching an auc score of 0.799, the highest so far out of all the models observed.



Feature importance under Gradient Boosting produces similar results to Random Forests and Decision Trees, deeming the most important features employment rate, euribor3m and whether the customer was previously contacted.

Boosting aims to improve its predictive ability by reducing bias while Random Forests act the opposite way with the task of variance reduction. From the results above we can say both models performed very well and the Gradient Boosting algorithm is slightly better than a Random Forest with an auc score of 0.799 compared to an auc score of 0.796.



UNIVERSITY OF EXETER | BUSINESS SCHOOL

Conclusion

From the variety of models fitted, we can observe the best results from the Random Forest Classifier, AdaBoost and GradientBoost. They produced higher auc scores than the previously examined models and were also able to identify the key prediction features. Out of all the predictor variables, we can conclude the most significant features to be the following:

- Employment Rate
- Euro InterBank Offered Rate (euribor3m)
- Previously contact with customer
- Age
- Number of contacts performed during the campaign
- Outcome of previous campaign

These identified features are greatly beneficial to a future marketing campaign as identifying key customer characteristics could lead to much higher effectiveness of the campaign and reduced costs.

The models initially fitted produced a fair predictive ability (apart from Quadratic Discriminant Analysis, which resulted in a very low predictive ability) which is beneficial as they were easier to implement and understand, as well as being computationally much more efficient. The most effective models (Random Forest and Boosting techniques) produced the best predictive ability at the cost of being not so interpretable and computationally very demanding (with cross validation and parameter tuning some of the models took 10+ minutes to train each). Another very computationally intensive task was subset selection through Recursive Feature Elimination due to each model being trained with each variable combination and with a 10-fold cross-validation the training time was high.

In conclusion, the findings were very interesting, applicable and could greatly facilitate a future marketing campaign, both from the same Portugese bank or another bank with a similar customer base. This case study could be improved by making use of deep learning and trained artificial neural networks, the predictive power of which could rival even the best performing models such as Boosting.

Appendix

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
plt.style.use('ggplot')


df = pd.read_csv('bank-additional-full.csv')
df.describe()
df.isnull().sum() #no null variables however some answered as unknown


#Data Visualisation
import seaborn as sns
sns.countplot(x='y', data=df)
plt.title('Frequency of response variable', weight='bold')


sns.countplot(x='marital', data=df)
sns.countplot(y='job', data=df)
sns.countplot(y='education', data=df)
sns.countplot(y='default', data=df) #has credit in default
sns.countplot(y='housing', data=df)
sns.countplot(y='loan', data=df)
sns.countplot(y='contact', data=df)
sns.countplot(y='month', data=df)
sns.countplot(y='day_of_week', data=df)


sns.countplot(y='pdays', data=df)
sns.countplot(y='campaign', data=df) #number of contacts performed during
the campaign
plt.hist((df['duration']),bins=100, color='#5C7DC4')
plt.hist((df['age']), bins=70, color='#5C7DC4')
plt.hist((df['emp.var.rate']), bins=5, color='#5C7DC4')
plt.hist((df['euribor3m']), bins=10, color='#5C7DC4')
```

```
#df = df.replace(to_replace='unknown', value=np.nan).dropna()

# Combine 'unknown' and 'other'

df['poutcome'] = df['poutcome'].replace(['other'] , 'unknown')

df['poutcome'].value_counts()

df.drop('contact', axis=1, inplace=True) #all participants were contacted

df.drop('duration', axis=1, inplace=True) #duration is unpredictable
beforehand


#changing pdays to previously_contacted

df['previously_contacted'] = 1

df['previously_contacted'][df['pdays']==999] = 0

df = df.drop('pdays', axis=1);


#Creating dummy variables for the series of categoricals (job, marital,
education, default, housing, loan, contact, )


df = pd.get_dummies(df,

                    columns=['job', 'marital', 'education',

                             'default', 'housing', 'loan',

                             'month', 'day_of_week',

                             'poutcome', 'y'],

                            drop_first=True) #dropping all columns but the
first one to establish as baseline


f, ax = plt.subplots(figsize=(15, 15))

sns.heatmap(df.corr(method='spearman'),

            annot=False, cmap='coolwarm') #

ax.set_title('Heatmap of correlations', weight='bold', size=20)

corr = df.corr()



X = df.drop(['y_yes'], axis=1)

y = df['y_yes']
```

```
from sklearn.linear_model import LogisticRegression
import sklearn.metrics as metrics
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    test_size=0.20,
                                                    random_state=10,

stratify=df['y_yes'].values)

#Helper Functions
def find_scores(classifier, X, y, cv_n=20, scoring='roc_auc'):
    rft = KFold(n_splits=cv_n, shuffle=True, random_state=0)
    return cross_val_score(classifier, X, y,
                           scoring=scoring,cv=rft).mean()

def create_auc_graph(classifier, X, color='blue', name=''):
    X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    test_size=0.25,
                                                    random_state=0,

stratify=df['y_yes'].values)
    classifier.fit(X_train, y_train)

    probs = classifier.predict_proba(X_test)
    preds = probs[:,1]
    fpr, tpr, threshold = metrics.roc_curve(y_test, preds)
    roc_auc = metrics.auc(fpr, tpr)

    plt.title('ROC-AUC graph ' + name)
    plt.plot(fpr, tpr, 'b', label = 'AUC = %0.3f' % score, color=color)
    plt.legend(loc = 'lower right')
```

```
    plt.plot([0, 1], [0, 1],'r--')
    plt.xlim([0, 1])
    plt.ylim([0, 1])
    plt.ylabel('True Positive Rate')
    plt.xlabel('False Positive Rate')
    plt.show()


#Logistic Regression
classifier = LogisticRegression()
score = find_scores(classifier, X, y) #76% roc_auc
accuracy_score = find_scores(classifier, X, y, scoring='accuracy')
create_auc_graph(classifier, X, color='#5C7DC4', name='Logistic
Regression')


#Linear discriminant analysis
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
clf = LinearDiscriminantAnalysis()



score = find_scores(clf, X, y, scoring='roc_auc')
accuracy_score = find_scores(clf, X, y, scoring='accuracy')
create_auc_graph(clf, X, color='#5C7DC4', name='Linear Discriminant
Analysis')



from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis
classif = QuadraticDiscriminantAnalysis()


classif.fit(X_train, y_train)
y_pred = classif.predict(X_test)


cm = confusion_matrix(y_test, y_pred)
score = find_scores(classif, X, y)
accuracy_score = find_scores(classif, X, y, scoring='accuracy')
create_auc_graph(classif, X, color='#5C7DC4')
```

```
from sklearn.metrics import plot_confusion_matrix
disp = plot_confusion_matrix(classif, X_test, y_test,
                             cmap=plt.cm.Blues)
ax= plt.subplot()
sns.heatmap(cm, annot=True,fmt='g', ax = ax)
ax.set_xlabel('Predicted labels');ax.set_ylabel('True labels');
ax.set_title('Confusion Matrix');
ax.xaxis.set_ticklabels(['no', 'yes']); ax.yaxis.set_ticklabels(['no',
'yes']);


#KNN
from sklearn.neighbors import KNeighborsClassifier


#Helper func KNN
def create_auc_graph_KNN(classifier, X,score, color='blue', name='',
k='1'):
    X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                        test_size=0.25,
                                                        random_state=0,

stratify=df['y_yes'].values)
    classifier.fit(X_train, y_train)


    probs = classifier.predict_proba(X_test)
    preds = probs[:,1]
    fpr, tpr, threshold = metrics.roc_curve(y_test, preds)
    roc_auc = metrics.auc(fpr, tpr)


    plt.title('ROC-AUC graph ' + name)
    plt.plot(fpr, tpr, 'b', label = str(k) +' AUC = %0.3f' % score,
color=color)
    plt.legend(loc = 'lower right')
    plt.plot([0, 1], [0, 1],'r--')
    plt.xlim([0, 1])
    plt.ylim([0, 1])
```

```
    plt.ylabel('True Positive Rate')
    plt.xlabel('False Positive Rate')
    plt.show()

n_neighbors = [1, 5, 10, 20, 50, 100, 500]
count = 0
colors = ['black', 'red', 'blue', 'green', 'purple', 'grey', 'yellow']
lister = list()
for i in n_neighbors:

    classifier = KNeighborsClassifier(n_neighbors=i,
                                      metric='minkowski', p=2)
    score = find_scores(classifier, X, y, scoring='accuracy')
    lister.append(score)
    create_auc_graph_KNN(classifier, X, color=colors[count], k=i,
                         score=score)
    count = count + 1



#implement logistic regression on the chosen model
#Running cross validation on the results
from sklearn.feature_selection import RFECV
from sklearn.model_selection import StratifiedKFold
rfc = LogisticRegression(random_state=0)
rfecv = RFECV(estimator=rfc, step=1, cv=20, scoring='roc_auc', verbose=1)
rfecv.fit(X, y)
rfecv.grid_scores_[41] #The optimal one

#Assigning which categories are to be removed
categories = pd.DataFrame()
categories['yes/no'] = rfecv.support_
categories['names'] = X.columns.values
categories['ranking'] = rfecv.ranking_
```

```
plt.figure(figsize=(16, 9))

plt.title('Recursive Feature Elimination with Cross-Validation',
fontsize=18, fontweight='bold', pad=20)

plt.xlabel('Number of features selected', fontsize=14, labelpad=20)

plt.ylabel('AUC score', fontsize=14, labelpad=20)

plt.plot(range(1, len(rfecv.grid_scores_) + 1), rfecv.grid_scores_,
color='black', linewidth=2)

plt.axvline(x=42)

plt.show()



#Feature importance

from sklearn.ensemble import ExtraTreesClassifier

import matplotlib.pyplot as plt

model = ExtraTreesClassifier()

model.fit(X,y)

print(model.feature_importances_) #use inbuilt class feature_importances of
tree based classifiers

#plot graph of feature importances for better visualization

feat_importances = pd.Series(model.feature_importances_, index=X.columns)

feat_importances.nlargest(15).plot(kind='barh', color='purple',
figsize=(20,10))

plt.show()




#Using ridge classifier

from sklearn.linear_model import RidgeClassifier


find_scores(classif, X, y)

create_auc_graph(classif, X, color='#5C7DC4')


score_lists = list()

alpha = [0.001, 0.1, 1, 3, 5,

         10, 15, 100, 200, 300,400, 500, 600, 700, 800,900, 1000]

for i in alpha:
```

```
    classifier = RidgeClassifier(alpha=i)
    #y_pred = classifier.predict(X_test)
    #score = roc_auc_score(y_test, y_pred)
    score = find_scores(classifier, X, y)
    score_lists.append(score)

score_df = pd.DataFrame(score_lists)
score_df.index = alpha
plt.plot(score_df.index, score_df[0], alpha=0.5) #Minimised error at
alpha=0.01



#Lasso Classifier
from glmnet import LogitNet
m = LogitNet(alpha=1, n_splits=10, n_lambda=100,
             verbose=1, scoring='roc_auc', random_state=0)
m = m.fit(X, y)

scores = pd.DataFrame()
scores['lambda'] = m.lambda_path_
scores['auc_score'] = m.cv_mean_score_

m.lambda_max_
m.lambda_max_inx_ #And we find the lambda value that maximises accuracy is
0.0026
max_score = scores[m.lambda_max_inx_]



plt.plot(scores['lambda'], scores['auc_score'], color='black')
plt.axvline(x=m.lambda_max_, color='red')
plt.xlabel('lambda')
plt.ylabel('auc score')

max_score = scores[m.lambda_max_inx_]
```

```
#Decision trees
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn import metrics

depths = [2,3,5,6,7,10,15,20,25]
scores_list = list()
for i in depths:
    clf = DecisionTreeClassifier(random_state=1, max_depth=i,
                                 criterion='entropy')
    score = find_scores(clf, X, y)
    scores_list.append(score)

scores_df = pd.DataFrame()
scores_df['max_depth'] = depths
scores_df['auc score'] = scores_list
plt.plot(scores_df['max_depth'], scores_df['auc score'],
         color='black')
plt.xlabel('maximum depth')
plt.ylabel('auc score')
plt.axvline(x=6) #the maximum score reached when max depth is 5



clf = DecisionTreeClassifier(random_state=1, max_depth=2,
                                 criterion='entropy')

find_scores(clf, X, y)

clf.fit(X, y)
#Visualising the Tree
from sklearn.externals.six import StringIO
from IPython.display import Image
```

```
from sklearn.tree import export_graphviz

import pydotplus

dot_data = StringIO()

export_graphviz(clf, out_file=dot_data,

                filled=True, rounded=True,

                special_characters=True, feature_names =
X.columns.values,class_names=['0','1'])

graph = pydotplus.graph_from_dot_data(dot_data.getvalue())

graph.write_png('tree.png')

Image(graph.create_png())


#Feature importances of the tree

clf = DecisionTreeClassifier(random_state=1, max_depth=6,

                             criterion='entropy')

clf.fit(X, y)

feat_importances = pd.Series(clf.feature_importances_, index=X.columns)

feat_importances.nlargest(10).plot(kind='barh', color='blue',
figsize=(20,10))

plt.title('Most important Features under Decision Tree')

plt.show()



#Bagging

from sklearn.ensemble import BaggingClassifier

from sklearn.tree import DecisionTreeClassifier

from sklearn.model_selection import RepeatedKFold

from sklearn.model_selection import cross_val_score



X_train, X_test, y_train, y_test = train_test_split(X, y,

                                                    test_size=0.25,

                                                    random_state=0,


stratify=df['y_yes'].values)


scores_list_bag = list()
```

```
for i in [2, 5, 10, 100, 200]:
    bag = BaggingClassifier(DecisionTreeClassifier(max_depth=6),
                            n_estimators=i, verbose=1, random_state=0)
    bag.fit(X_train, y_train)
    results = cross_val_score(bag, X_test, y_test,
                          scoring='roc_auc')
    score = results.mean()
    scores_list_bag.append(score)

scores_df_bag = pd.DataFrame()
scores_df_bag['n_trees'] = [2, 5, 10, 100, 200]
scores_df_bag['auc score'] = scores_list_bag

plt.plot(scores_df_bag['n_trees'], scores_df_bag['auc score'])
plt.xlabel('n_estimators')
plt.ylabel('auc score')
plt.title('Bagging Classifier', size=18)

#Random forest #okay so change max features to see predictability
from sklearn.ensemble import RandomForestClassifier
num_trees = [2, 5, 10, 20, 50, 100, 200]
scores_list_6 = list()
for i in num_trees:
    kfold = RepeatedKFold(n_splits=10, n_repeats=1, random_state=0)
    model = RandomForestClassifier(n_estimators=i, max_features=7,
                                   max_depth=6)
    results = cross_val_score(model, X, y, cv=kfold, scoring='roc_auc')
    score = results.mean() #89.1% accuracy
    scores_list_6.append(score)


scores_df_forest = pd.DataFrame()
scores_df_forest['n_trees'] = num_trees
scores_df_forest['scores'] = scores_list_6
```

```
#Random Forest Classifier - Features Selected= 14
from sklearn.ensemble import RandomForestClassifier
num_trees = [2, 5, 10, 20, 50, 100, 200]
scores_list_6_14 = list()
for i in num_trees:
    kfold = RepeatedKFold(n_splits=10, n_repeats=1, random_state=0)
    model = RandomForestClassifier(n_estimators=i, max_features=14,
                                   max_depth=6)
    results = cross_val_score(model, X, y, cv=kfold, scoring='roc_auc')
    score = results.mean() #89.1% accuracy
    scores_list_6_14.append(score)

scores_df_forest_14 = pd.DataFrame()
scores_df_forest_14['n_trees'] = num_trees
scores_df_forest_14['scores'] = scores_list_6_14

#Random Forest Classifier - Features Selected= 21
from sklearn.ensemble import RandomForestClassifier
num_trees = [2, 5, 10, 20, 50, 100, 200]
scores_list_6_21 = list()
for i in num_trees:
    kfold = RepeatedKFold(n_splits=10, n_repeats=1, random_state=0)
    model = RandomForestClassifier(n_estimators=i, max_features=21,
                                   max_depth=6)
    results = cross_val_score(model, X, y, cv=kfold, scoring='roc_auc')
    score = results.mean() #89.1% accuracy
    scores_list_6_21.append(score)

scores_df_forest_21 = pd.DataFrame()
scores_df_forest_21['n_trees'] = num_trees
scores_df_forest_21['scores'] = scores_list_6_21

#Random Forest Classifier - Features Selected= 28
```

```
from sklearn.ensemble import RandomForestClassifier
num_trees = [2, 5, 10, 20, 50, 100, 200]
scores_list_6_28 = list()
for i in num_trees:
    kfold = RepeatedKFold(n_splits=10, n_repeats=1, random_state=0)
    model = RandomForestClassifier(n_estimators=i, max_features=28,
                                   max_depth=6)
    results = cross_val_score(model, X, y, cv=kfold, scoring='roc_auc')
    score = results.mean() #89.1% accuracy
    scores_list_6_28.append(score)

scores_df_forest_28 = pd.DataFrame()
scores_df_forest_28['n_trees'] = num_trees
scores_df_forest_28['scores'] = scores_list_6_28

#Plotting of Random Forest Graph
plt.plot(scores_df_forest['n_trees'], scores_df_forest['scores'],
         color='black', label='Features Selected=7')
plt.plot(scores_df_forest_14['n_trees'], scores_df_forest_14['scores'],
         color='red', label='Features Selected=14')
plt.plot(scores_df_forest_21['n_trees'], scores_df_forest_21['scores'],
         color='green', label='Features Selected=21')
plt.plot(scores_df_forest_28['n_trees'], scores_df_forest_28['scores'],
         color='yellow', label='Features Selected=28')
plt.xlabel('n_trees')
plt.ylabel('auc score')
plt.title('Random Forest Classifier')
plt.legend()

#Feature importance
kfold = RepeatedKFold(n_splits=10, n_repeats=1, random_state=0)
model = RandomForestClassifier(n_estimators=200, max_features=21,
                               max_depth=6)
model.fit(X, y)
```

```
feat_importances = pd.Series(model.feature_importances_, index=X.columns)
feat_importances.nlargest(10).plot(kind='barh', color='red',
figsize=(20,10))
plt.title('Most important Features under Decision Tree')
plt.show()


#AdA Boosting - normal model with learning rate=1
from sklearn.model_selection import RepeatedKFold
from sklearn.ensemble import AdaBoostClassifier
scores_list_ada = list()
num_trees = [1, 5, 10, 20, 50, 100, 200]
for i in num_trees:
    kfold = RepeatedKFold(n_splits=10, n_repeats=1)
    model = AdaBoostClassifier(n_estimators=i,
                               learning_rate=1)
    results = cross_val_score(model, X, y, cv=kfold, scoring='roc_auc')
    score = results.mean() #89.5%
    scores_list_ada.append(score)

scores_df_ada = pd.DataFrame()
scores_df_ada['n_trees'] = num_trees
scores_df_ada['auc score'] = scores_list_ada

#Tuning the model for learning rate=0.1
scores_list_ada_01 = list()
num_trees = [1, 5, 10, 20, 50, 100, 200]
for i in num_trees:
    kfold = RepeatedKFold(n_splits=10, n_repeats=1)
    model = AdaBoostClassifier(n_estimators=i,
                               learning_rate=0.1)
    results = cross_val_score(model, X, y, cv=kfold, scoring='roc_auc')
    score = results.mean()
    scores_list_ada_01.append(score)

scores_df_ada_01 = pd.DataFrame()
```

```
scores_df_ada_01['n_trees'] = num_trees
scores_df_ada_01['auc score'] = scores_list_ada_01

#Tuning the model for learning rate=0.01
scores_list_ada_001 = list()
num_trees = [1, 5, 10, 20, 50, 100, 200]
for i in num_trees:
    kfold = RepeatedKFold(n_splits=10, n_repeats=1)
    model = AdaBoostClassifier(n_estimators=i,
                               learning_rate=0.01)
    results = cross_val_score(model, X, y, cv=kfold, scoring='roc_auc')
    score = results.mean()
    scores_list_ada_001.append(score)

scores_df_ada_001 = pd.DataFrame()
scores_df_ada_001['n_trees'] = num_trees
scores_df_ada_001['auc score'] = scores_list_ada_001

#Tuning the model for learning rate=1.5
scores_list_ada_15 = list()
num_trees = [1, 5, 10, 20, 50, 100, 200]
for i in num_trees:
    kfold = RepeatedKFold(n_splits=10, n_repeats=1)
    model = AdaBoostClassifier(n_estimators=i,
                               learning_rate=1.5)
    results = cross_val_score(model, X, y, cv=kfold, scoring='roc_auc')
    score = results.mean()
    scores_list_ada_15.append(score)

scores_df_ada_15 = pd.DataFrame()
scores_df_ada_15['n_trees'] = num_trees
scores_df_ada_15['auc score'] = scores_list_ada_15
```

```
#Plots
plt.figure(figsize=(15,10))
plt.plot(scores_df_ada['n_trees'], scores_df_ada['auc score'],
         label='Learning Rate=1')
plt.plot(scores_df_ada_01['n_trees'], scores_df_ada_01['auc score'],
         label='Learning Rate=0.1')
plt.plot(scores_df_ada_001['n_trees'], scores_df_ada_001['auc score'],
         label='Learning Rate=0.01')
plt.plot(scores_df_ada_15['n_trees'], scores_df_ada_15['auc score'],
         label='Learning Rate=1.5')
plt.xlabel('number of estimators')
plt.ylabel('auc score')
plt.title('AdaBooster')
plt.legend()


#Check Feature importances
kfold = RepeatedKFold(n_splits=10, n_repeats=1)
model = AdaBoostClassifier(n_estimators=200,
                           learning_rate=1.5)
model.fit(X, y)
feat_importances = pd.Series(model.feature_importances_, index=X.columns)
feat_importances.nlargest(10).plot(kind='barh', color='purple',
figsize=(20,10))
plt.title('Most important Features under Ada Boosting')
plt.show()


#Stochastic Gradient Boosting
from sklearn.model_selection import RepeatedKFold
from sklearn.ensemble import GradientBoostingClassifier
num_trees = 100
kfold = RepeatedKFold(n_splits=10, n_repeats=1)
model = GradientBoostingClassifier(n_estimators=num_trees,
                                   learning_rate=0.1)
results = cross_val_score(model, X, y, cv=kfold, scoring='roc_auc')
results.mean()
```

```
#Learning Rate of 0.01
scores_list_Gradboost_001 = list()
num_trees = [1, 5, 10, 20, 50, 100, 200]
for i in num_trees:
    kfold = RepeatedKFold(n_splits=10, n_repeats=1)
    model = GradientBoostingClassifier(n_estimators=i,
                                learning_rate=0.01)
    results = cross_val_score(model, X, y, cv=kfold, scoring='roc_auc')
    score = results.mean()
    scores_list_Gradboost_001.append(score)

scores_df_grad_001 = pd.DataFrame()
scores_df_grad_001['n_trees'] = num_trees
scores_df_grad_001['auc score'] = scores_list_Gradboost_001

#Learning Rate of 0.1
scores_list_Gradboost_01 = list()
num_trees = [1, 5, 10, 20, 50, 100, 200]
for i in num_trees:
    kfold = RepeatedKFold(n_splits=10, n_repeats=1)
    model = GradientBoostingClassifier(n_estimators=i,
                                learning_rate=0.1)
    results = cross_val_score(model, X, y, cv=kfold, scoring='roc_auc')
    score = results.mean()
    scores_list_Gradboost_01.append(score)

scores_df_grad_01 = pd.DataFrame()
scores_df_grad_01['n_trees'] = num_trees
scores_df_grad_01['auc score'] = scores_list_Gradboost_01

#Learning Rate of 1
scores_list_Gradboost_1 = list()
num_trees = [1, 5, 10, 20, 50, 100, 200]
```

```
for i in num_trees:
    kfold = RepeatedKFold(n_splits=10, n_repeats=1)
    model = GradientBoostingClassifier(n_estimators=i,
                                learning_rate=1)
    results = cross_val_score(model, X, y, cv=kfold, scoring='roc_auc')
    score = results.mean()
    scores_list_Gradboost_1.append(score)


scores_df_grad_1 = pd.DataFrame()
scores_df_grad_1['n_trees'] = num_trees
scores_df_grad_1['auc score'] = scores_list_Gradboost_1


#Plots for gradient boosting
plt.figure(figsize=(15,10))
plt.plot(scores_df_grad_001['n_trees'], scores_df_grad_001['auc score'],
         label='Learning Rate=0.01')
plt.plot(scores_df_grad_01['n_trees'], scores_df_grad_01['auc score'],
         label='Learning Rate=0.1')
plt.plot(scores_df_grad_1['n_trees'], scores_df_grad_1['auc score'],
         label='Learning Rate=1')
plt.xlabel('number of estimators')
plt.ylabel('auc score')
plt.title('GradientBooster')
plt.legend()


#Checking feature importances
kfold = RepeatedKFold(n_splits=10, n_repeats=1)
model = GradientBoostingClassifier(n_estimators=200,
                                learning_rate=0.1)
model.fit(X, y)
feat_importances = pd.Series(model.feature_importances_, index=X.columns)
feat_importances.nlargest(10).plot(kind='barh', color='purple',
figsize=(20,10))
plt.title('Most important Features under Gradient Boosting')
plt.show()
```


UNIVERSITY OF
EXETER
BUSINESS
SCHOOL