Bank Marketing Analysis

By: Harshita Bali, Ayesha Ali, Bemnet Desta, Evan Sant

Bank Marketing Analysis

Author1 Ayeshi Ali, DePaul University, School of Computing, aali111@depaul.edu
Author2 Beemnet Desta, DePaul University, School of Computing, bdesta@depaul.edu
Author3 Evan Sant, DePaul University, School of Computing, esant@depaul.edu
Author4 Harshita Bali, DePaul University, School of Computing, hbali@depaul.edu

ABSTRACT

We analyzed the bank marketing data that gives us insights about customer behavior and effectiveness of the campaign. The dataset shows demographic information of customers along with campaign details and response outcomes. The study identifies high-value customer segments and effective campaigns which in turn enables banks to optimize marketing strategies thereby enhancing customer acquisition. These insights contribute to data driven decision making, facilitating improved marketing outcomes in the competitive banking industry. This study proposes various machine learning models to predict whether the customer subscribes to term deposit based on the promotional phone call. Then, aim to propose the best model suitable for bank marketing. This approach allows banks to deliver relevant and timely messages, resulting in improved customer engagement and higher conversion rates.

INTRODUCTION

The Bank Marketing dataset we selected gives the information on Portuguese financial institution's direct marketing campaigns. The purpose of the project is to get people to open a term deposit which is a form of savings account with a set period and interest rate. The dataset is sourced from UCI repository and contains 23 variables and 45211 observations. The following are columns definitions:

- age: age of the customer (numeric)
- age group: what age group does the customer belong to
- eligible: if the customer is eligible for the talk or not(binary: "yes", "no")
- job: what does the customer do (categorical: "admin", "unknown", "unemployed", "management", "housemaid", "entrepreneur", "student", "blue-collar", "self-employed", "retired", "technician", "services")
- salary: salary of the customer
- marital: marital status (categorical: "married", "divorced", "single")
- education: level of education completed (categorical: "unknown", "primary", "secondary", "tertiary")
- marital-education: married or educated/ uneducated
- targeted: if the customer or being targeted or not
- default: if the customer has credit in default(binary: "yes", "no")
- balance: remaining balance in their accounts (numeric)
- housing: has housing loan or not(binary: "yes", "no")
- loan: has prior loan (personal loan) or not
- contact: contact communication type (categorical: "unknown", "telephone", "cellular")
- day: last contact day of the month (numeric)

- month: last contact month of the year (categorical: "jan", "feb", ..., "dec")
- duration: number of days last contacted (numeric)
- campaign: number of contacts performed during this marketing campaign (numeric)
- pdays: number of days passed by after the client was last contacted from a previous campaign (numeric, -1 means the client was not previously contacted)
- previous: number of contacts performed before this campaign (numeric)
- poutcome: outcome of the previous marketing campaign (categorical: "unknown", "other", "failure", "success")
- y: has the client subscribed a term deposit (binary: "yes", "no")
- response: response of the client

The goal of our project is to predict whether a client will subscribe to a term deposit (“y”) or not by looking at the type of customers that take advantage of the marketing campaign offer.

LITERATURE REVIEW

Bank Marketing plays a crucial role in attracting and retaining customers in the financial industry. With an increased rise of digital technology we have seen that banks have turned to data-driven strategies to enhance their marketing efforts. Throughout our project we have explored the bank marketing data, its campaign effectiveness and the impact of the personalized marketing approaches on the customers. It is because by integrating these insights into decision making processes, banks can enhance customer acquisition, and can improve the retention rates so as to stay in a competitive zone.

The paper titled Response Modeling in Direct Marketing: A Data Mining-Based Approach for Target Selection focuses on the application of various data mining techniques in direct marketing [1]. It discusses the use of classification algorithms, such as decision trees, logistic regression, and support vector machines, for building response models.The importance of feature selection and preprocessing steps in response modeling are emphasized. The paper also highlights techniques such as information gain, chi-square test, and correlation analysis for selecting relevant predictors. Additionally, data preprocessing methods like missing value imputation, outlier detection, and normalization are discussed to ensure data quality and model performance.
The authors show the application of response modeling techniques and demonstrate how the proposed data mining- based approach can be applied to identify potential customers for bank direct marketing campaigns.

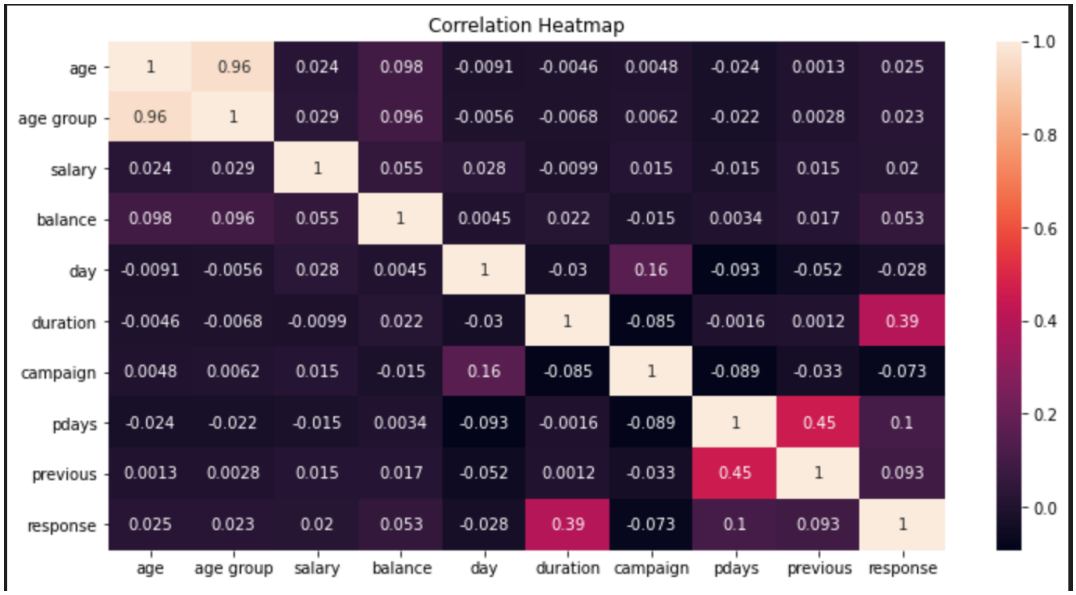
…….

HYPOTHESES/MODEL

Exploratory Data Analysis

The heatmap correlation between the features was plotted to check for multicollinearity. Age and Age Group seemed to be highly correlated with a correlation coefficient of 0.96. As a result, the age group was dropped. Salary and balance have a weak positive association with the age and age group factors. This suggests that older adults enjoy a higher wage and a better work-life balance. Another thing we can see from the heatmap is the 0.39 correlation coefficient

between duration and response variable. This implies that lengthier call durations during marketing efforts may have a better likelihood of prompting a positive response from clients.



Data Cleaning

During our data cleaning process, our data had no missing values. There were 13 categorical variables that were converted to dummy variables. Altogether, there are 69 features and 45211 instances. We dropped the variables "age group," "day," and "duration" were removed from the dataset because they did not contribute significantly to the analysis.

Logistic Regression

Logistic regression is a statistical modeling technique used to predict the probability of a binary outcome based on one or more predictor variables. In this project, we utilized sklearn's Logistic Regression class. We used it to explore our main question in addition to some extra ones that we had. We first wanted to answer our main question so we wrote the code and received a rounded up score of 0.89. While not perfect, this score still does a very good job of

```
X = df.drop(['y_yes','y_no','response'], axis=1)
y = df.response
lb = LabelBinarizer()
y = lb.fit_transform(y).ravel()
y = pd.Series(y)
X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y, random_state = 55, test_size= 0.25)
clf = LogisticRegression(max_iter = 10000, C=0.1)
clf.fit(X_train,y_train)
pred=clf.predict(X_train)
trainScore = metrics.accuracy_score(y_train, pred)
clf.fit(X_train,y_train)
pred = clf.predict(X_test)
testScore = metrics.accuracy_score(y_test, pred)
print('The train score is:',trainScore)
print('The test score is:' ,testScore)
print(recall_score(y_test, pred, average='weighted'))
```

```
The train score is: 0.886310015335614
The test score is: 0.8894098911793329
0.8894098911793329
```

predicting if someone will enroll in the program or not. We then decided to see if we could improve the model tuning the C value. We found that the score largely stays the same with 0.1 being the best value. After doing this, we then wanted to see what Logistic Regression could do with our data. We decided to see if it could predict eligibility, if a customer was targeted, if they had a loan, and their education level (primary, secondary, tertiary, and unknown). Each of these tests earned a score of 0.97, 0.82, 0.83, 0.83, 0.76, 0.78, and 0.96 respectively. While some scores are lower than we hoped, the scores are mostly solid.

Splitting the dataset

The training and test data was split in the ratio 80 -20 percentage using the sckit learn train test split package.

Decision Trees

Decision trees are versatile machine learning models that construct a hierarchical structure of decision rules to make predictions. They are widely used due to their simplicity, interpretability, and ability to handle various data types. In our case, when we ran our decision tree, we observed that our model accurately predicted all cases with a perfect accuracy score of 100% on the training set. However, when evaluating the model on the validation set the accuracy dropped to 87.1%. This suggests that the model may have a possibility of overfitting the training data.

While analyzing the classification report, we can see that for class 0 (not responding) the model achieved aPrecision of 0.93, a Recall of 0.92, and an F1-Score of 0.93. For class 1 (responding) our model achieved a Precision of 0.45, a Recall of 0.47, and an F1-Score of 0.46. This indicates that the model was able to correctly identify a high proportion of non-responding customers. However, the model struggled to accurately identify responding customers, resulting in lower precision and recall.

The overall accuracy was 87.1% which suggests that the model performed well in classifying both classes. However, the balanced accuracy of 69.7% provides a more balanced evaluation of the model's performance by taking the class imbalance into account. Our F1-Score was 0.458, and recall was 0.470, indicating that some responding customers were predicted correctly. Furthermore, we had a precision score of 0.445, which shows the proportion of expected responding customers reacting in our scenario. Finally, we had a Specificity score of 0.923, which shows the percentage of non-responding clients accurately forecasted.

```
dt = DecisionTreeClassifier(random_state=RANDOM_STATE)

dt.fit(X_train,y_train)

#Accuracy Train
pred_train = dt.predict(X_train)
print("Accuracy Score (Train): ", metrics.accuracy_score(y_true = y_train, y_pred = pred_train))

#Accuracy Val
pred = dt.predict(X_val)
print("Accuracy Score (Val): ", metrics.accuracy_score(y_true = y_val, y_pred = pred))
```

```
Accuracy Score (Train):  1.0
Accuracy Score (Val):  0.8707144437071445
```

```
print(metrics.classification_report(y_val, pred))
```

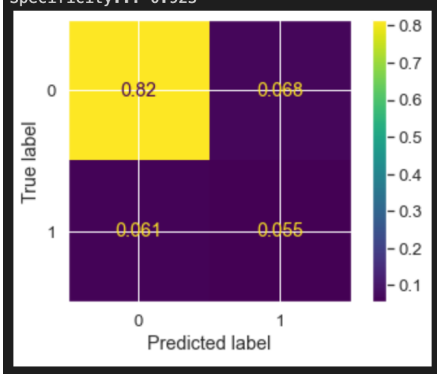
```
              precision    recall  f1-score   support

           0       0.93      0.92      0.93      7994
           1       0.45      0.47      0.46      1048

    accuracy                           0.87      9042
   macro avg       0.69      0.70      0.69      9042
weighted avg       0.87      0.87      0.87      9042
```

```
print_scores('Decision tree',dt, X_val, y_val)
```

```
Accuracy.....: 0.871
Bal. Accuracy: 0.697
F-1 Score....: 0.458
Recall.......: 0.470
Precision....: 0.445
Specificity..: 0.923
```



Decision Trees with Hyper-parameter tuning

The hypertuning operation was performed on a set of parameters min_samples_split and max_depth. min_samples_split = [5, 10, 20, 30, 40, 50],max_depth = [1,2,3,None]

On the testing set, the decision tree classifier with hyper-paramerter tuning achieved an accuracy score of 90.6%, indicating that it correctly predicted the response variable for 90.6% of the instances.

Looking at the classification report, for class 0 (not responding), the model achieved a Precision of 0.92, a Recall of 0.98, and an F1-Score of 0.95. For class 1 (responding), the model achieved a Precision of 0.68, a Recall of 0.34, and an F1-Score of 0.46. On the testing set, the model achieved an accuracy, macro-averaged F1-score, and weighted-averaged F1-score of 90.6%. These represent the model's overall success in accurately identifying instances from the testing set. These show that the model successfully identified instances that belonged to class 0, with high levels of precision, recall, and F1-score in comparison to examples that belonged to class 1.

In comparison to the model without tuning, the decision tree model with hyper-parameter tuning performed better, obtaining higher accuracy and better metrics for class 0 (not responding).

```
# Predicting on test set
gs_dt_pred = best_estimator_dt.predict(X_test)
print('Accuracy of test set: ', metrics.accuracy_score(y_test, gs_dt_pred))
```

```
Accuracy of test set:  0.9061152272475949
```

```
# computing classification report and calculating following metrics on testing set
print(metrics.classification_report(y_test, gs_dt_pred))
print('Recall : ', metrics.recall_score(y_test, gs_dt_pred, pos_label=1))
print('Specificity : ', metrics.recall_score(y_test, gs_dt_pred, pos_label=0))
print('Balanced accuracy : ', metrics.balanced_accuracy_score(y_test, gs_dt_pred) )
```

```
              precision    recall  f1-score   support

           0       0.92      0.98      0.95      8006
           1       0.68      0.34      0.46      1037

    accuracy                           0.91      9043
   macro avg       0.80      0.66      0.70      9043
weighted avg       0.89      0.91      0.89      9043

Recall :  0.3442622950819672
Specificity :  0.9788908318760929
Balanced accuracy :  0.66157656347903
```

Random Forest

Random Forest builds an ensemble of decision trees, where each tree is constructed using a random subset of the training data and a random subset of the input features. These decision trees are trained independently and make predictions based on their individual decisions .We created a random forest using a specific random state. This classifier is then trained by fitting it using X_train and y_train on the training set.

```
RandomForestClassifier
RandomForestClassifier(random_state=33)
```

During training Random Forest training model achieved an accuracy score of 1.0, indicating that the model achieved flawless accuracy on the training data. Moving on to the validation set, the random forest achieved an accuracy score of 0.89 on the validation set. This indicates that the model predicted the outcome properly on the validation data.

The classification report included a thorough assessment of the models' performance on the validation set. The macro-average F1-score of 0.70 indicates that both classes had a fair balance of precision and recall. It indicates that both positive and negative examples were accurately classified by the model in a respectable amount of time.

```
#Accuracy Train
pred_train1 = rf.predict(X_train)
print("Accuracy Score (Train): ", metrics.accuracy_score(y_true = y_train, y_pred = pred_train1))

#Accuracy Val
pred1 = rf.predict(X_val)
print("Accuracy Score (Val): ", metrics.accuracy_score(y_true = y_val, y_pred = pred1))
```

```
Accuracy Score (Train):  1.0
Accuracy Score (Val):  0.898363193983632
```

```
print(metrics.classification_report(y_val, pred1))
```

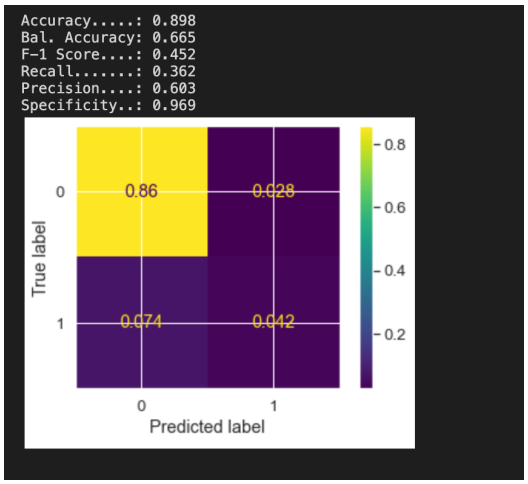
```
              precision    recall  f1-score   support

           0       0.92      0.97      0.94      7994
           1       0.60      0.36      0.45      1048

    accuracy                           0.90      9042
   macro avg       0.76      0.67      0.70      9042
weighted avg       0.88      0.90      0.89      9042
```

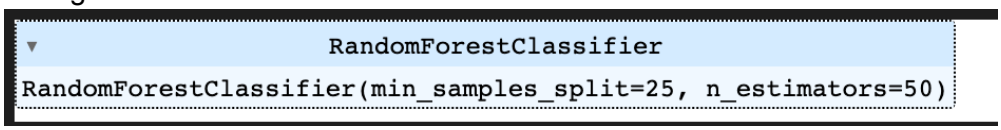
A percentage of the positive cases were correctly detected by the model, as seen by the recall (sensitivity) value of 0.362 for the positive class. The proportion of correctly predicted positive instances was 0.603, which represents the precision

for the positive class. The F1-score, which combines precision and recall into one metric, was 0.452 for the positive class. In addition, we had a balanced accuracy of 0.665.

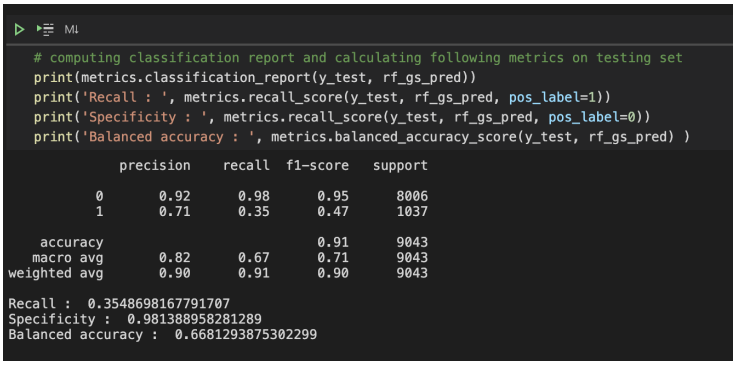


Random Forest with Hyper-parameter tuning

GridSearchCV with hyperparameter tweaking was utilized to enhance the Random Forest model's performance. The parameters that yielded the best estimator were min_samples_split=25 and n_estimators=50. The optimized model was then assessed using the testing set.



The optimized random forest model had an accuracy of 0.90. It showed that for 91.0% of the instances in the testing set, it correctly anticipated the answer variable.
The classification report shows that the model performed admirably, with a macro-average F1 score of 0.71. Class 1 recall (positive class) recall was 0.355, class 0 recall was 0.981, and balanced accuracy was 0.668.



```
# computing classification report and calculating following metrics on testing set
print(metrics.classification_report(y_test, rf_gs_pred))
print('Recall : ', metrics.recall_score(y_test, rf_gs_pred, pos_label=1))
print('Specificity : ', metrics.recall_score(y_test, rf_gs_pred, pos_label=0))
print('Balanced accuracy : ', metrics.balanced_accuracy_score(y_test, rf_gs_pred) )
```

```
              precision    recall  f1-score   support

           0       0.92      0.98      0.95      8006
           1       0.71      0.35      0.47      1037

    accuracy                           0.91      9043
   macro avg       0.82      0.67      0.71      9043
weighted avg       0.90      0.91      0.90      9043

Recall :  0.3548698167791707
Specificity :  0.981388958281289
Balanced accuracy :  0.6681293875302299
```

Overall, the random forest model demonstrated a good performance on the validation and testing sets, both in its Initial state and after hyperparameter adjustment. It was successful in accurately predicting the results of the dataset because it received high accuracy ratings and fair evaluation metrics.

We found that the optimized random forest with hyper-parameter tuning outperformed the random forest without hyper-parameter tuning when comparing the two Random Forests. This is evident when looking at the accuracy, micro-average F1-score, specificity, and ability to identify positive instances. The procedure of hyper-parameter tweaking helped us to enhance the model's functionality and produced a classification that was more accurate and balanced.

KNN

The k-Nearest Neighbors (KNN) Classifier was applied with 5 neighbors. The accuracy score for the KNN without hyper-parameter tuning was 0.8829, indicating that this classifier correctly predicts the response variable in 88.29% of cases.

```
M↓
print("Accuracy Score: ", accuracy_score(y_test, knn_pred) )
print("\n")
print("Classification Report: ")
print(metrics.classification_report(y_test, knn_pred))
```

```
Accuracy Score:  0.8828928452947031


Classification Report:
              precision    recall  f1-score   support

           0       0.91      0.97      0.94      8006
           1       0.48      0.25      0.33      1037

    accuracy                           0.88      9043
   macro avg       0.69      0.61      0.63      9043
weighted avg       0.86      0.88      0.87      9043
```

When examining the classification report, we can see that the model attained a precision of 0.91, a recall of 0.97, and an F1-score of 0.94 for class 0 (not responding). On the other hand, the model attained precision of 0.48, recall of 0.25, and F1-score of 0.33 for class 1 (responding). The weighted average F1 score ended out to be 0.87, which shows that performance across the classes was fairly balanced.

KNN with Hyper-parameter tuning

The KNN was customized by taking into account specific variables like the weight function and the number of neighbors. 3, 5, 7, and 10 were used to test the number of neighbors. 'Uniform' and 'Distance' were the two options used to investigate the weighted function. Grid search cross-validation was used to find the classifier's optimal set of hyper-parameters. The best estimator for the KNN classifier was discovered to have 10 neighbors, a "uniform" weight function, and the distance metric "manhattan" after executing a grid search classifier.

The accuracy score of 0.8893 for the KNN with hyper-parameter tuning demonstrates an improvement over the first KNN classifier. According to the classification report, our model attained a precision of 0.90, a recall of 0.99, and an F1-score of 0.94 for class 0 (not responding). Our model had a precision of 0.57, a recall of 0.14, and an F1-score of 0.22 for class 1 (responding). With a value of 0.86, the weighted average F1-score shows a modest decline when compared to the initial KNN classifier.

```
# Print Accuracy Report and Classification Report
print("Accuracy Report:", accuracy_score(y_test, knn_pred_2) )
print ("\n")
print ("Classification Report:")
print(metrics.classification_report(y_test, knn_pred_2))
```

```
Accuracy Report: 0.8893066460245493


Classification Report:
              precision    recall  f1-score   support

           0       0.90      0.99      0.94      8006
           1       0.57      0.14      0.22      1037

    accuracy                           0.89      9043
   macro avg       0.73      0.56      0.58      9043
weighted avg       0.86      0.89      0.86      9043
```

After hyper-parameter adjustment, the classifier demonstrated greater accuracy when predicting the majority of class 0 (not responding) with higher precision and recall values when compared to the two KNN models. On the other hand, class 1's performance worsened from reduced precision, recall, and an F1-score.

We can also notice that the weighted average F1-score has somewhat decreased. This shows that the hyper-parameter's tweaking did not result in an enhancement to the model's overall performance.

XGBoost

When running the XGBoost before hyper-parameter tuning the model achieved an accuracy score of 0.91 on the test set, which indicates that it correctly classified 91% of the instances. According to the classified report, the model's precision for class 0 is higher than for class 1, with a value of 0.93 versus 0.64. which suggests that it is more adept at spotting unfavorable circumstances. Class 1 recall is 46%, while class 0 recall is 97%. The model does a good job of balancing the classification performance for both classes, as seen by the balanced accuracy of 71.4%.

```
# computing classification report and calculating following metrics on testing set
print(metrics.classification_report(y_test, pred_val))
print('Recall : ', metrics.recall_score(y_test, pred_val, pos_label=1))
print('Specificity : ', metrics.recall_score(y_test, pred_val, pos_label=0))
print('Balanced accuracy : ', metrics.balanced_accuracy_score(y_test, pred_val) )
```

```
              precision    recall  f1-score   support

           0       0.93      0.97      0.95      8006
           1       0.64      0.46      0.54      1037

    accuracy                           0.91      9043
   macro avg       0.78      0.71      0.74      9043
weighted avg       0.90      0.91      0.90      9043

Recall :  0.46190935390549664
Specificity :  0.9659005745690732
Balanced accuracy :  0.7139049642372849
```

XGBoost with Hyper-parameter tuning

Grid search with 5-fold cross-validation was used to search over a parameter grid with various hyperparameter values. With the defined parameter grid and accuracy as the scoring metric, a grid search is conducted.

```
# Using k=5 folds
k= 5
params = dict(
    n_estimators= [25],
    max_leaves = [7,8],
    gamma = [0],
    max_depth = list(range(5,7))
)
params
```

```
{'n_estimators': [25], 'max_leaves': [7, 8], 'gamma': [0], 'max_depth': [5, 6]}
```

```
# Applying XGBoost classifier model
xg = XGBClassifier()
xg_gs = GridSearchCV(estimator=xg, param_grid=params, scoring='accuracy', cv=k, n_jobs=-1 )

# Fit the model
xg_gs.fit(X_train, y_train)
```

```
▸ GridSearchCV
▸ estimator: XGBClassifier
    ▸ XGBClassifier
```

```
best_estimator_xg = xg_gs.best_estimator_

# predicting on test data
xg_pred = best_estimator_xg.predict(X_test)
metrics.accuracy_score(y_test, xg_pred)
```

```
0.9082162999004755
```

```
pd.DataFrame(metrics.confusion_matrix(y_test, xg_pred))
```

	0	1
0	7765	241
1	589	448

```
# computing classification report and calculating following metrics on testing set
print(metrics.classification_report(y_test, xg_pred))
print('Recall : ', metrics.recall_score(y_test, xg_pred, pos_label=1))
print('Specificity : ', metrics.recall_score(y_test, xg_pred, pos_label=0))
print('Balanced accuracy : ', metrics.balanced_accuracy_score(y_test, xg_pred) )
```

```
              precision    recall  f1-score   support

           0       0.93      0.97      0.95      8006
           1       0.65      0.43      0.52      1037

    accuracy                           0.91      9043
   macro avg       0.79      0.70      0.73      9043
weighted avg       0.90      0.91      0.90      9043

Recall :  0.43201542912246865
Specificity :  0.969897576817387
Balanced accuracy :  0.7009565029699278
```

After hyper-parameter adjustment, the model's accuracy score on the test data was 90.8%, which is somewhat less accurate than the initial XGBoost. According to the classification report, class 0 continues to have a high precision rating of 0.93. However, the accuracy for class 1 went from 64% to 65%, with a 1% increase. With a value of 0.43, the recall for class 1 shows that the model has considerable trouble describing the positive examples. On the other hand, the recall for class 0 was 0.97. After hyper-parameter adjustment, the balanced accuracy dropped marginally to a new value of 70.1%. This shows that there was a small performance trade-off between the two classes.

Choosing the best fitted models for each of the classifiers

Based on the metrics below, XGBoost looks to be the classifier that performs the best overall, with reasonably high values for accuracy, F1-score, recall, precision, and balanced accuracy. It is critical to remember that the best classifier is determined by the specific objectives and goals of the activity at hand.

```
tableList = []
#tableList.append(svmList)
tableList.append(knnList)
tableList.append(dtList)
tableList.append(rfList)
tableList.append(xgList)

column_labels = ['classifier', 'accuracy', 'f1-score', 'specificity', 'precision', 'recall', 'balanced']
model = pd.DataFrame(tableList, columns = column_labels)
print(model)
```

```
     classifier  accuracy  f1-score  specificity  precision    recall  \
0           KNN  0.889307  0.224632     0.986385   0.570866  0.139826
1  DecisionTree  0.906115  0.456814     0.978891   0.678707  0.344262
2  RandomForest  0.908548  0.496039     0.975393   0.673841  0.392478
3       XGBoost  0.908216  0.519119     0.969898   0.650218  0.432015

   balanced
0  0.563106
1  0.661577
2  0.683936
3  0.700957
```

RESULTS

Choosing the best fitted models for each of the classifiers.

The performance of the models is evaluated with different metrics accuracy, recall , precision and Fscore. The performance analysis on each of the five learning models are carried out.

Model name	Train Accuracy	Test Accuracy	Recall
Logistic Regression	0.886310015335614	0.8894098911793329	NA
KNN		0.8828928452947031	0.25 (For class1)
Decision Tree (Before Tuning)	1.0	0.8707144437071445	0.470
Decision Tree (After Tuning)	NA	0.9061152272475949	0.3443 (For class 1)
Random Forest (Before Tuning)	1.0	0.898363193983632	0.362(For class 1)
Random Forest (After Tuning)	NA	0.9095432931549264	0.3549 (For class 1)

XGBoost	NA	0.908216	0.432015

DISCUSSION AND CONCLUSIONS

Overall, a lot of the tests scored really well. Additionally, that means that they all scored very similarly in test scores. This is a good thing because it means our models are consistent and we have a lot of solid options, but it also raises a tricky question. Which method should we recommend to the bank? Between Logistic Regression, KNN, Decision Tree, Random Forest, and XGBoost, the range of scores is only about 0.03. This means that we need to go a little more indepth to recommend a model. The next step would be to compare recall scores. From this we have a little bit more variation with a range of 0.22. The highest score was for Decision Trees prior to tuning. The lowest score was from KNN. Before picking which method we think is best, let's recap what we did in each model.

Starting with Logistic Regression, we simply dropped the response variable as well as y_yes and y_no. After that, we were able to run the regression model and calculate the score. Logistic Regression is used to predict a variable that has two outcomes. For our dataset this is the case. They can answer yes or no, so Logistic Regression here makes sense. In terms of its score it was middle of the pack.

Next, let's talk about Decision Trees. Decision Trees are used to create a hierarchy of decision rules to make predictions. They are used because they are simpler to interpret and can handle many data types. The untuned version of the Decision Tree model showed signs of overfitting so we decided to tune it. After tuning the model, the model performed much better and as such makes a strong case.

Third, we're going to look at Random Forest. This method builds an ensemble of decision trees with random chunks of the training set and of input features. It does well because it averages multiple decision trees, which makes it less susceptible to overfitting, noise, and outliers. Our base model performed well, but it is still worthwhile to tune the model to see if we can get it to perform better. After tuning, we noticed the model's performance increased slightly.

Fourth, we wanted to look at KNN. KNN is a good method to use in cases where recognition is needed. Things like speech and image identification. However, it doesn't perform great on large datasets. One might think we mean that it doesn't score well and in that situation they would be incorrect. In fact, the model scores quite well. However, the runtime of the model is exceptionally long. If the model had performed better than the others in a substantial way, then it could make sense to use it over the others. However, it only performs better than our base Decision Tree model. As such, it is the first one we can feel comfortable eliminating as the method we would recommend.

Lastly, we used XGBoost. XGBoostis used in cases that have a large number of observations and has a mixture of numerical and categorical features. Our dataset fits both of these situations and as such seems promising. When running the model we get an excellent score and can see that it does a good job of balancing classification performance. After tuning the model, the score was similar to how it was before. In some areas it decreased while in others it increased. After looking at the tradeoff we decided that the original model is better than the tuned one due to the fact that the tradeoffs were slightly more negative than positive.

So which model do we recommend? When looking at situation and performance it seems that Logistic Regression and XGBoost fit the best. Due to both models fitting the

situation well, it makes sense to choose the better performing one of the two. As such XGBoost is the model that we would recommend.

AUTHOR CONTRIBUTIONS

In our case all four authors worked together to help create the final project. We each made models, visualizations, slides in the presentations, and additions to the final report.

REFERENCES

Sadaf Hossein Javaheri, Mohammad Mehdi Sepehri, and Babak Teimourpour. Response modeling in direct marketing: A data mining based approach for target selection. In Yanchang Zhao and Yonghua Cen, editors, Data Mining Applications with R, chapter 6, pages 153–178. Elsevier, 2014.