ITM 618 Course Project

Dante Signorella – 500745832

1 Introduction

For the course project, I was tasked with creating a classification model that predicts whether or not a customer ends up subscribing for a term deposit with a Portuguese Banking Institution based on marketing campaign data. I was provided with two sets of data: Training Data and Testing Data. The Training Data was to be used to train the model we are building, while the testing data was to be used to test the model's effectiveness.

In this report, I will talk about the attributes that contributed significantly to the target class, the different methods I attempted to use to build my model, the results of each of my models, and a recap as well as my thoughts on how my model performed.

2 Data Exploration

When exploring the data, I looked at the Information Gain (IG) for both the test and training data. What I found was that the variable with the highest IG was nr.employed, and many categorical variables such as housing, loan, and marital did not create insight into the target class, and thus, should not be included in the model. As I continued to explore the data I also found that two variables, pdays, and poutcomes, had high Information Gains but also a large amount of 'unknown' attributes as many of the rows did not feature values for these variables. As a result, they had to be removed from the data set as they could potentially lead to falsehoods in my findings.

Many of the numerical data figures also had to be cleaned due to their high amount of volatility. To do this, I decided to scale all of the data points, so that their means were zero. Doing this would allow my model to much more effectively predict target classes, as outliers do

not affect the model as much as they would have otherwise. After scaling the data, I used the summary( ) command to analyze the new variables to ensure that there would not be outliers skewing my results. The two variables of duration and campaign did just that and eliminating them from my data helped to significantly improve results.

When analyzing the target class, I found that the proportion of total outcomes that were positive (yes) was higher in the test data than in the train data. To fix this issue for my kNN model, I created a random index of 22,000 negative result rows in my training data and eliminated the excess rows from my training data. This would result in my model predicting a proper amount of target classes to be positive, as to be in line with the number of classes that are positive and negative.

3. Learning Methods

3.1 kNN

The reason I chose kNN for my first model was that in the training data, the numerical values had higher Information Gains than the categorical values in the test dataset and were not featuring any missing variables. Using only reliable numerical variables improved the reliability of my model as the training data did not feature any variables that were once N/A or unknown. In order to get some of the categorical values in the dataset to work with the kNN algorithm, I had to convert them to numerical values. For the categorical variable of the month, I converted each month to a numerical value that ran one through twelve. This resulted in a significantly better model as the month was a variable that had a high Information Gain.

When building the model and training/testing data sets, I had to create tables specifically for the target class. The reason I had to do this was that if the ‘yes’ and ‘no’ variables were in my training and testing data, the model would be heavily skewed to using those data points (the

actual target class) for predicting, and my model would predict at nearly 100% accuracy. To counteract this, the training and test data had to NOT include the target class, and instead, the target class had to be a separate data set, where my kNN model would use the training data to predict the test data, without having access to the training data's target class until seeing how variables impacted it.

The k number I decided on using for my model was 109. The reason for this high number was due to a large number of examples in my training dataset. Balancing the k number was most likely the most difficult aspect of building the predictive model as it was a lot of trial and error between choosing a k number and choosing an amount of 'no' variables to keep in my training dataset. Ultimately, the kNN classifier was the most consistent for predicting target classes, unlike other classifiers I used that were highly volatile depending on the variables being used.

3.2 Random Forest Tree

As an alternative to the kNN model I built, I decided on using a Random Forest Tree. What I found through my building of the model was that the training data was very poor at building a decision tree classifier unless heavily modified. My classifier would always predict (almost) every target class to be positive, which confused me given that only 14% of the target classes in the training data were 'yes'. I ended up having to only keep three 'yes' examples in my training data for the Random Forest Tree to predict outcomes that were proper. When building the classifier, I chose an ntree of 600, a mincriterion of 0.78, and a maxdepth of 70. I believe that these restraints on the algorithm result in more repeatable and meaningful splits in the tree and prevent over-fitting towards the training data.

In order to get the Random Forest Tree to be effective, I needed to normalize the numerical values and ensure the categorical data being used was complete. This resulted in me

using a slightly different version of training data numerical values when compared to my kNN model, as you cannot scale data sets with non-numerical variables (I had to create a normalization function and use that), and I was able to use a few more categorical variables than my kNN classifier, as I did not have to convert them to numerical values in order to get them to work with the model. I also had to use different training and testing data set as the 'Subscribed' target class was able to be included in the training data for a Random Forest classifier, unlike in kNN.

4 Evaluation

4.1 kNN

When evaluating my kNN model, I look to a few key metrics to gain insight into how accurate it is at predicting an overall number of classes, and the individual yes and no classes. The metrics and their interpretation when used to test the predictions on the Test Dataset are listed below:

- Accuracy (# of examples correctly classified / total number of examples in the dataset)
    - The accuracy of my model was 90% (10723/11917), meaning the kNN model correctly predicted the target class 90% of the time.
- Error Rate (1-Accuracy)
    - The error rate for my model was 10%, meaning the kNN incorrectly predicted the target class 10% of the time.
- Precision (True Positive / True Positive + False Positive)
    - The precision of my model was 58.18% (889 / 889+639)
- Recall (True Positive / True Positive + False Negatives)
    - The recall (completeness) of my model was 61.57% (889 / 889+555)

- F-Score
    - The F-Score for my model was 58.08%

Getting to these results was difficult, as it took a multitude of changes to my model to arrive at the proper balance of F-Score and Accuracy that I was satisfied with. To improve upon initial iterations of my model where my F-Score was low but my accuracy was high I modified my training data to have a proper balance of Positive and Negative target classes and adjusted the k number in my kNN model to provide a balanced outcome.

To get the data to work properly with the kNN algorithm, I had to normalize it. My most successful method of doing so was through scaling the data, which resulted in a mean of zero for all of the data classes, but scaled to show any variances in the data, and affect outliers proportionally. This garnered me more success than normalizing the data, with each data point having a maximum of one, and a minimum of zero. Using these methods, I was able to cut down on the False Negatives and Positives, while also maximizing the True Positives and Negatives that my model was predicting.

4.2 Random Forest Tree

When testing my random forest tree, balancing the accuracy with precision and recall was my main goal. Until I began using only three ‘yes’ examples in the training data, my random forest tree was classifying almost every single prediction as a ‘yes’. This speaks to the difficulty of this dataset, and the general unpredictability of creating predicting models. The results of my Random Forest Tree are:

- Accuracy (# of examples correctly classified / total number of examples in the dataset)
    - the accuracy of my model was 84.95% (10124/11917), meaning the Random Forest model correctly predicted the target class 84.95% of the time.

- Error Rate (1-Accuracy)
    - the error rate for my model was 15.05%, meaning the kNN incorrectly predicted the target class 15.05% of the time.
- Precision (True Positive / True Positive + False Positive)
    - the precision of my model was 43.65% (1028 / 1028 +1327)
- Recall (True Positive / True Positive + False Negatives)
    - the recall (completeness) of my model was 70.91% (1028 / 1028 +416)
- F1Score
    - The F1Score for my model was 54.04%

The Random Forest Model I built has its strengths and weaknesses when compared to the kNN classifier I built. The strengths are the capabilities of my model. As shown in the results above, the Recall of the model is extremely high, leading to a high F-Score and Accuracy. However, the precision falls well below 50%, which is simply unacceptable for a model. I believe the main issue is not the classifier itself, but the data. The data required heavy alterations to work with the random forest algorithm and is simply unreliable for a classifier.

Overall, the model is not perfect but given the challenges that the data set tasked me with tackling (volatility in data, unpredictability, missing values), I believe the 54.04% F-Score coupled with the high accuracy of it is impressive and accomplishes what the project tasked me with performing.

5 Discussion

Building these models and refining them so their predictions not only had high accuracy but recall and precision were extremely difficult. Because of the existence of incomplete data

points, gaining reliable and valuable insights into what variables could repeatedly and consistently predict the Target Class was time-consuming and, at times, frustrating. Through relying on metrics such as Information Gain, I was able to eliminate unnecessary variables and build my model off of both categorical and numerical variables that optimized the predicted outcomes.

It was also extremely interesting to see that the Random Forest model and kNN model required two training data sets that were cleaned significantly differently. For the kNN model, I had to eliminate 'no' outcomes as the high amount of those outcomes created bias in my predictions, while the Random Forest Model was not outputting (almost) any 'no' predictions until I removed all but three of the 'yes' outcomes. I believe the kNN model is more repeatable and doesn't generalize to the extent that the Random Forest model had been for the majority of its predictions.

6 Conclusion

The two models I built to predict whether or not a customer will subscribe to a term deposit succeeded. They both had F-Scores above 50%, and they both had accuracies above 85%. To get these outcomes, I had to first gain insights into the variables that were valuables to predicting the target classes, and then, clean the training data. This was ultimately the most time-consuming part of completing the projects, but ultimately was the part that rewarded me the most and helped me to build two successful models.

The kNN model I built was able to be successful after I removed variables that did not assist in predicting the target class, and I normalized the data. Normalizing the data was crucial for my model to be successful, and significantly improved performance. With an F-Score above

58%, and accuracy of 90%, my model succeeded and thrived even with a difficult and unpredictable training data set that would normally cause significant errors in predictions.

My Random Forest model is a stark contrast to the kNN model I built. The training data set had to be cleaned significantly differently than that of my kNN model, and the training data was not being successful at predicting the test data target class. After interpreting predictions that were overwhelming for the 'yes' target class, I set out to remove the significant bias in my model. I was able to do this by removing all but three 'yes' outcomes in my training data. This resulted in an accuracy that exceeded 87%, and an F-Score that was over 56%. This model was extremely successful at predicting the target class accurately but had its shortcomings in the form of repeatability and poor precision scores.

Overall, my models were extremely successful at predicting the test data target class, and performed exceptionally when using the metrics of accuracy, precision, and recall evaluating them. I believe the kNN model is the better of the two, and, as a result, it was the one I focused my time on improving as it is repeatable and consistent with its results.