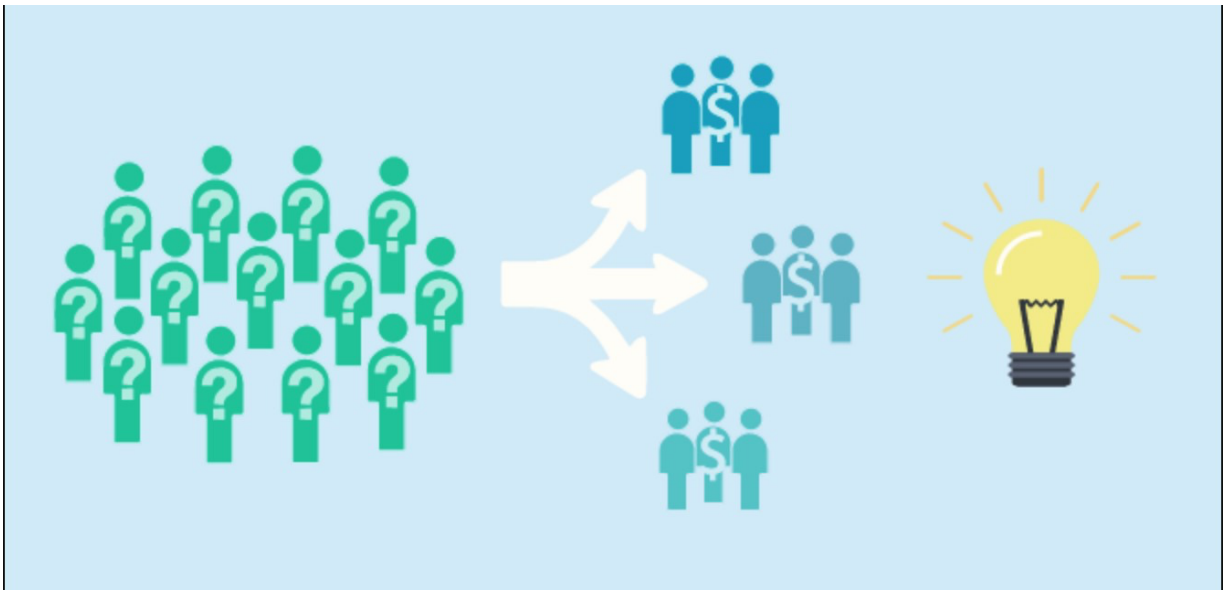


Customer Transaction Prediction

Featuring: Santander Bank

Jake Zegeer & Anh Nguyen
CS-484-DL1
Professor Lin
30 April 2021

Santander Bank: Future Customer Transaction Prediction

Kaggle Project Overview (datasets in Kaggle link, datasets were too large for submission): https://www.kaggle.com/c/santander-customer-transaction-prediction/overview

Github (currently private for honor code reasons): https://github.com/jzegeer/CS484

Introduction

After analyzing the various different types of kaggle competitions, we both settled on Santander Banks, Customer Transaction Prediction competition. This specific competition caught our attention since I had a higher difficulty and because there were multiple approaches to it. The goal of this project was to predict customers' future transactions. The dataset included 200 feature variables, target values consisting of "1"'s and "0"'s, a means of identification codes and 200,000 total datasets. We frequently ran test runs with about 1/10 of this data to improve computational speeds and to better efficiency. Below is a screenshot of the first five ID's from the training data.

	ID_code	target	var_0	var_1	...	var_196	var_197	var_198	var_199
0	train_0	0	8.9255	-6.7863	...	7.8784	8.5635	12.7803	-1.0914
1	train_1	0	11.5006	-4.1473	...	8.1267	8.7889	18.3560	1.9518
2	train_2	0	8.6093	-2.7457	...	-6.5213	8.2675	14.7222	0.3965
3	train_3	0	11.0604	-2.1518	...	-2.9275	10.2922	17.9697	-8.9996
4	train_4	0	9.8369	-1.4834	...	3.9267	9.5031	17.9974	-8.8104

Approach

Our approach was to first research which type of models would fit best with this dataset. Other competitors used linear-based models since we were dealing with binary target values. From here we decided to use three models at least to test this and explore different ways of solving this problem. The two that came to mind first were Gausssian Navíe Bayes and Logistic Regression. Both of which use a linear model. We also decided to use a Random Decision Tree to see if this theory of if a linear based model could be better than non-linear. Once we had decided the three models we were going to implement later on, we began creating pseudo code to map out our approach.

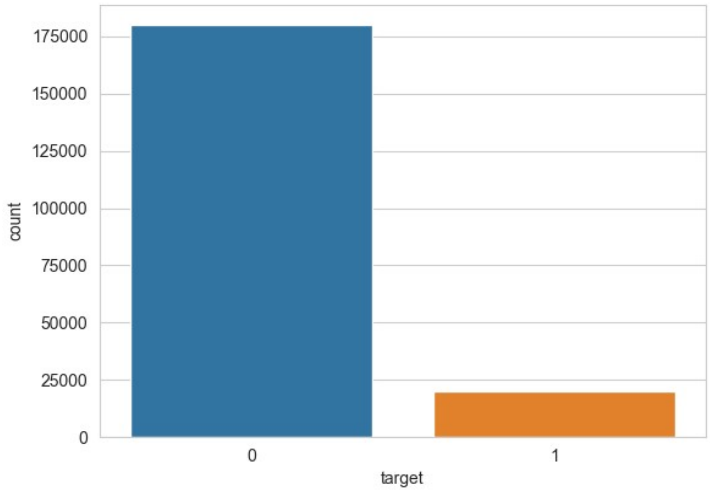
We used the three "sklearn" imports for the three models. Next we had to decide how we were going to deal with preprocessing and cross-validation methods. Again we went with "sklearn" libraries to speed up this process as we were trying to find which model would be best applicable to the given problem from Santander Bank on future Customer Transaction predictions. After mapping out the approach, we came to the conclusion that we would use the following "sklearn"

imports: “cross_val_score”, “CalibratedClassifierCV”, and “MinMaxScaler”. Two of which we had never used or heard of. “CalibratedClassifierCV” is used to optimize probabilities for a probability prediction. We found this could be useful after implementing each model to crossvalidate the data with each partition of training datasets. “MinMaxScaler” we thought could be useful in the preprocessing stage as it could quickly scale down the training set and fit this result with our prediction model. We believe this could be useful in the TSNE computation as it would keep the same distribution and be more efficient time wise.

Moving further, we found creating visuals concurrently as we moved forward would not only help our grade but also visualize what our code was doing. To do this we found using both ”matplotlib.pyplot”, “seaborn”, and “scikitplot” could be of use. “Matplotlib.pyplot” is used to plot normal figures such as showing the given target range of data from the training/test sets. “Seaborn” could be used to further illustrate similar plots as “matlibplot.pyplot” but with further details, such as for a normal distribution chart. Finally, we found “scikitplot” could help show what was happening for plots such as the ROC(Receiver Operating Characteristic Curve) for visually seeing the difference percentage wise of test to training data.

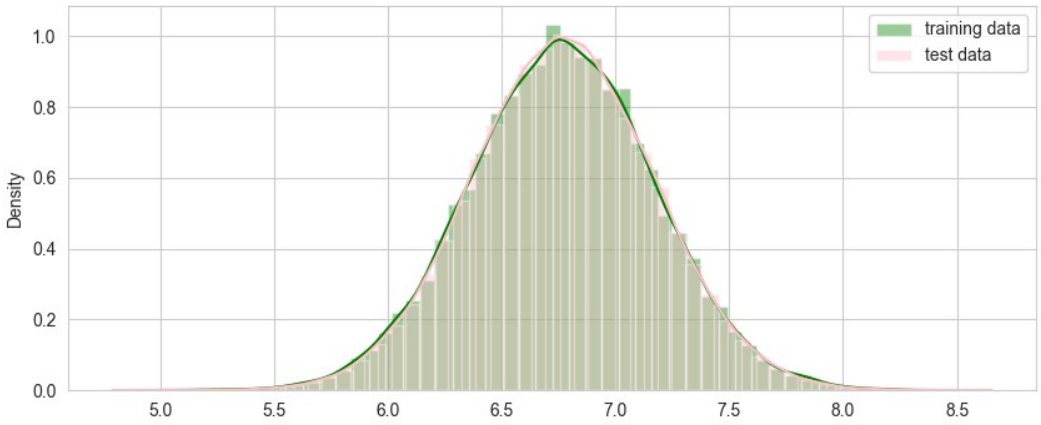
Pre-Implementation of models:

Once we saw how the datasets were formatted, we decided to analyze this data further by plotting the target values. Since the provided data did not have a target value, we had to calculate this. The following graph depicts the target distribution. There are 179,902 total “0”’s and 20,098 “1”’s. The “Blue/0” labeled data indicates customers that did not make a transaction. While the “Orange/1” labeled data indicates the customers that did make a transaction. This illustrated to us that there were far many more target values of “0”’s than “1”’s.



At this point, we decided to cut the data by 1/10 (200,000 datasets to 20,000) to help efficiently use our time. This problem had a few special cases. First, the negative data (customers not making transactions) greatly ratioed the positive data (customers making a transaction).

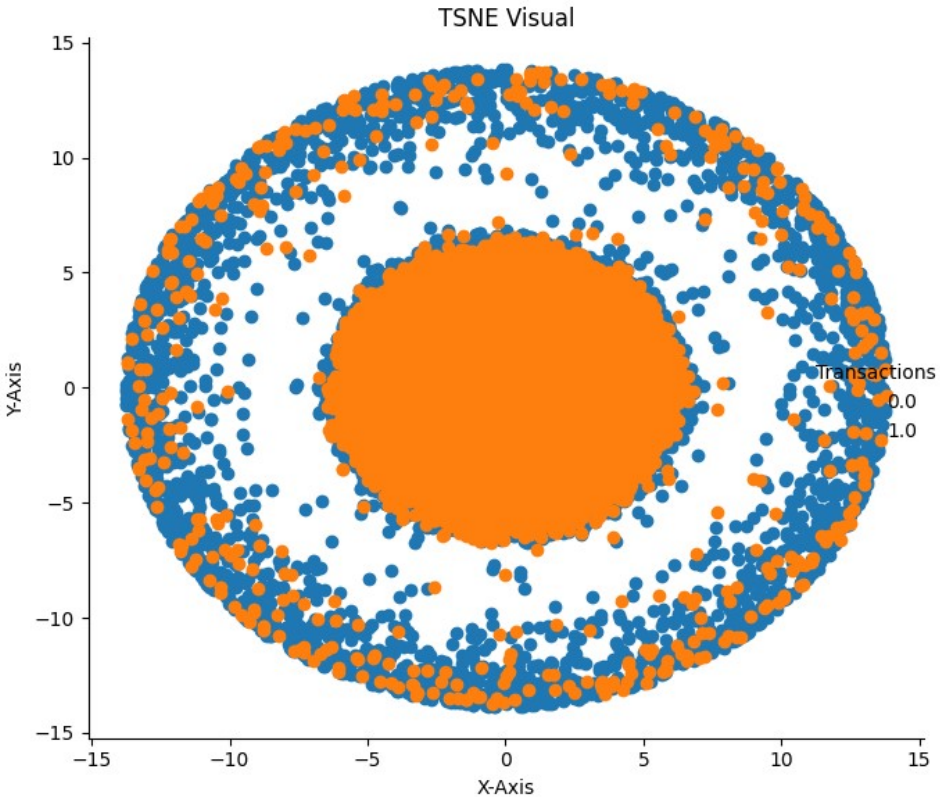
Next, we decided to verify how the distribution of training data looked like for the rows. As seen below, there is a bit of skewed data between 6.5 and 7.0 value areas. The “lilly green” bars represent the training data attributes while the pick represents the test data attributes. There are several ways to visualize the data for the given data set but the plot we wanted to analyze was for the rows of target values.



Since this distribution graph was not completely clear of how this data was in correlation with each other at this point, we decided to use the TSNE library. We decided this because TSNE was helpful for use in previous assignments for this course and thought that since the data sets were so large, that TSNE could be of use.

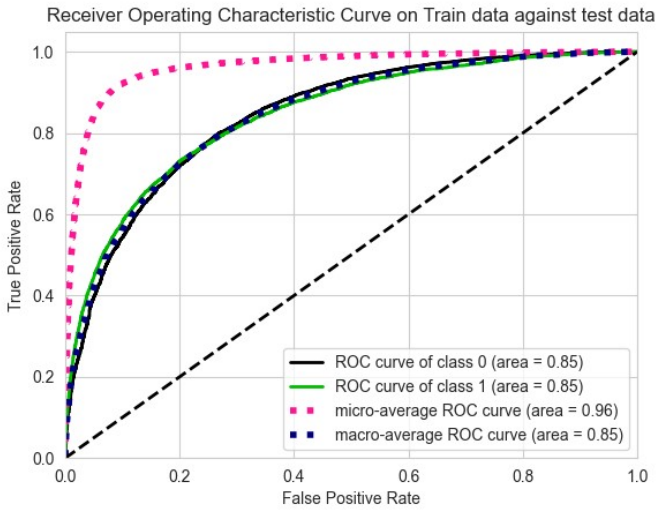
As a means for visualizing the data further, we used the import TSNE to try and help see how the target values reside with the entire training data set. TSNE (t-distributed stochastic Neighbor Embedding) library explained: TSNE works by making projections based on a computed fit. Basically, by taking a high-dimensional data set and reducing it to a low-dimensional graph. Based on this first fit, TSNE tries to compare two entities to find which fit is preferred. Because of this, it is known as a reduction algorithm. Going deeper, the first step TSNE does is scaling all these data points onto a 2-Dimensional graph (as seen above). This is done in a similar manner as other distance formulas, but it takes place on a 2-dimensional, meaning it only cares about the distance on a normal distribution chart. For example, say the average grade in CS-484 is 80 percent and this is a centroid. The other picked centroids are 70 and 90 percent. If a point is 82, this would reside to the class average this this is closest to the normal distribution of 80 percent. Next comes the “t” distribution which acts similarly to a normal distribution. TSNE uses “t” distribution to avoid the clusters all clumping up and becoming inaccurate clusters.

As seen below, the data shows that the customers making transactions mostly clump together (Orange plots) and the customers not making transactions clump together (Blue plots).



Implementation of models:

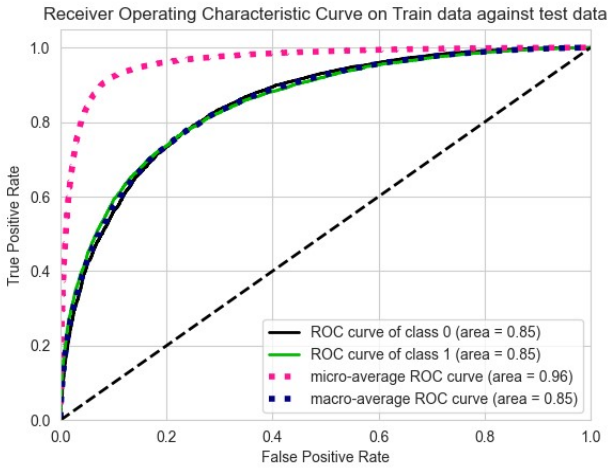
With this being known we then decided to move forward and use a Random Decision Tree for our model. For this process we needed to first get rid of the data attribute labels (“target” and “ID_code”). Once this was complete, we then had to split the training data and test data into two subsets making random partitions for the data. Next, we plugged these partitions of training data into the Random Forest model. We chose this model first because of the simplicity aspect compared to other models. Random Forest works by making several different decision trees and then merging these decision trees into one. This makes for higher scoring accuracy and better prediction stability. The model is non-linear which we thought might help give a higher score due to the high scattered data sets. Below is the Receiver Operating Characteristic (ROC) Curve for our Random Decision Tree. The black solid line is the Customers who did not make a transaction and the green solid line is the Customers who did make a transaction.



Below is the ROC Curve accuracy score we received for the Random Decision Tree:

```
ROC Score for training on test data : 0.841356436964704
```

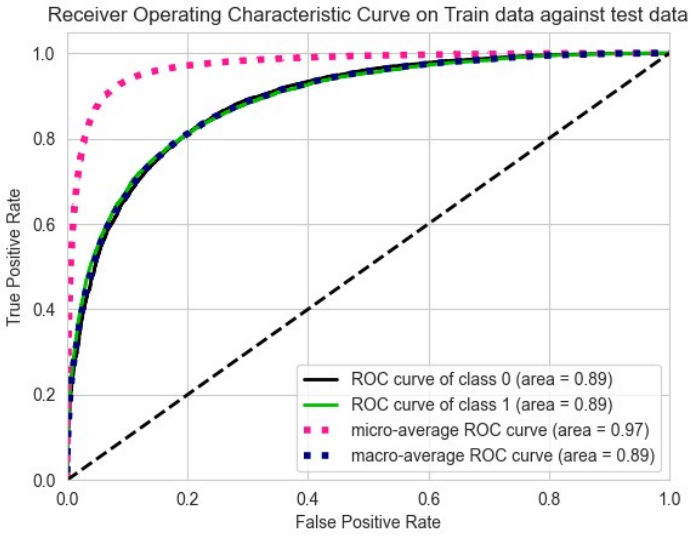
After our results for our Random Decision Tree, we decided to try logistic regression to see if we could achieve better results. Logistic regression is a linear based model which differs from the previous approach. We thought this model would fit in our project well since it is a great way to classify binary based problems. This model works by estimating the given target values in an equation such as (1/(1+e^(-target_value))) and then transforming this output into a plot. Although we believed this model to be the best out of the three models we used, it ranked second (ahead of the Random Tree Classification). Below is our ROC Curve for our Logistical Regression model. The black solid line is the Customers who did not make a transaction and the green solid line is the Customers who did make a transaction.



Below is the ROC Curve score for the Logistic Regression model:

```
ROC Score for training on test data : 0.8450533878688824
```

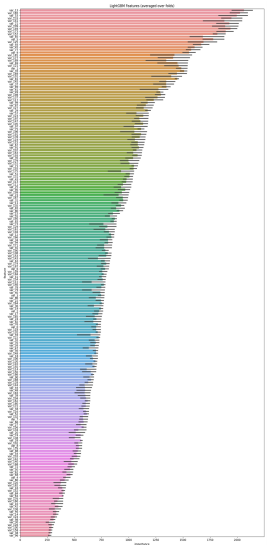
Finally, we tried our final model of Gaussian Naïve Bayes (GNB). We thought that GNB would be a good model to try on the given dataset since it is known to work well with continuous data sets. We also thought GNB would improve our predictions since it is a linear based model and our score improved while using a linear based model. GNB first takes prior probabilities to determine a starting point. Once this is complete, this value is multiplied by the target values. These values are placed on a plot, giving the visual of a standard distribution chart. From here, the predictions are already made. Since we only dealt with binary numbers, we also found that underflow could not possibly happen since there are only two possible target values this model works with. Below are our ROC Curve results:



We received our best ROC Curve score with GNB, presented below is our score:

```
ROC Score for training on test data : 0.8884236612933112
```

After additional research of how other competitors received higher scores, we found that the highest scores were achieved by using a library called "LightGBM". This library works based on "tree-based learning". LightGBM greatly improves computation power and is very efficient compared to other models. Rather than a regular tree that creates leafs horizontally, the leaves are grown vertically. After doing further research, we discovered that this library is extremely popular, especially in Hackathon competitions. We did attempt to run this model, however, found that this model took over two hours to run with the full dataset. We deleted our code but thought we'd include our plot of what this model looks like. Below is the attributes from the training data:



To conclude our findings, we found that out of the three models we used, Gaussian Naive Bayes fit the best; then following Logistic Regression and then Random Decision Tree. GNB not only favored the best score but also computed the quickest. We believe this is the case because GNB only needs to take an initial estimate for a mean value and then categorize this data on this plot, creating a standard deviation visual. While Logistic Regression might've had a shot given the best optimal value, based off the cross-validation method we used, this model fell short just behind GNB by a few percentage points (84.5% to 88.8%). And the Random Decision Tree came in last since we believe the other two models are a more optimal approach for this problem since the tree is a broader approach while compared to a GNB or Logistic regression based model.

References:

Model Type research:

"Sklearn.naive_bayes.GaussianNB¶." *Scikit*,

scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.GaussianNB.html.

"Sklearn.linear_model.LogisticRegression¶." *Scikit*,

scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.ht

ml.

"Sklearn.ensemble.RandomForestClassifier¶." *Scikit*, scikit-

learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html.

Sklearn libraries researched:

Sklearn.calibration.calibratedclassifiercv¶. (n.d.). Retrieved April 30, 2021, from https://scikit-learn.org/stable/modules/generated/sklearn.calibration.CalibratedClassifierCV.html

Sklearn.model_selection.cross_val_score¶. (n.d.). Retrieved April 30, 2021, from

https://scikit-

learn.org/stable/modules/generated/sklearn.model_selection.cross_val_score.html **Kaggle**

research:

Jerifate. (2021, March 31). Santander customer transaction prediction [lgbm]. Retrieved April 30,

2021, from https://www.kaggle.com/jerifate/santander-customer-transaction-prediction-lgbm

Xuanzhihuang. (2021, January 05). Santander customer transaction prediction - lgbm. Retrieved ,

April 30, 2021, from https://www.kaggle.com/xuanzhihuang/santander-customer-transaction-prediction-lgbm

Arozrl. (2020, December 16). Accelerated training using snap ml. Retrieved April 30, 2021, from https://www.kaggle.com/arozrl/accelerated-training-using-snap-ml **LightGBM**

Research:

Dwivedi, R. (n.d.). What is lightgbm algorithm, how to use it? Retrieved April 30, 2021, from https://www.analyticssteps.com/blogs/what-light-gbm-algorithm-how-use-it