Capstone 2 - Final Report

5th September 2018

Problem and Client

Santander Group wants to provide their current customers and new customers services not found in other banking providers. The first step in achieving this level of service, Santander Group wants to **determine the amount or value of a customer's transaction.** This will enable them to personalize their services to current and new customers. Customers today are more likely to do business with a company if they provide personalized service; this applies to all sectors of business.

Data Wrangling and Exploratory Data Analysis (EDA)

The data for this project has been provided by the Santander Group in the form a Kaggle competition (https://www.kaggle.com/c/santander-value-prediction-challenge). Santander Group has provided 3 csv files; 1) a sample submission file, 2) a training set file and, 3) a test set file. The training and test files are an anonymized dataset containing numeric feature variables, the numeric target column, and a string ID column.

To understand the size of the train and test data sets the shape of the data frames was explored.

```
print("Train Shape : ", train_df.shape)
print("Test Shape : ", test_df.shape)
```

```
Train Shape :  (4459, 4993)
Test Shape :  (49342, 4992)
```

As shown above the test data set is roughly 10 times larger than the training set. The test set shows a very large number of rows and each set has over 4000 columns as well. Next, to understand the what data looks like we can review the first few rows of the training set.

```
train_df.head()
```

	ID	target	48df886f9	0deb4b6a8	34b15f335	a8cb14b00	2f0771a37	30347e683	d08d1fbe3	6ee66e115
0	000d6aaf2	38000000.0	0.0	0	0.0	0	0	0	0	0
1	000fbd867	600000.0	0.0	0	0.0	0	0	0	0	0
2	0027d6b71	10000000.0	0.0	0	0.0	0	0	0	0	0
3	0028cbf45	2000000.0	0.0	0	0.0	0	0	0	0	0
4	002a68644	14400000.0	0.0	0	0.0	0	0	0	0	0

Looking at the first few rows of the training set shows an ID column and feature columns with anonymized naming conventions. The target column is showing large values which would make sense if we are trying to determine a value for each ID or customer. It's also interesting to see a lot of zero values in the data. It appears that for each ID there could only be a few greater than zero values in the 4000 plus columns.

To get a better idea of the values in the feature columns we can look at some summary statistics.

```
train_df.describe()
```

	target	48df886f9	0deb4b6a8	34b15f335	a8cb14b00	2f0771a37	30347e683	d08d1fbe3
count	4.459000e+03	4.459000e+03	4.459000e+03	4.459000e+03	4.459000e+03	4.459000e+03	4.459000e+03	4.459000e+03
mean	5.944923e+06	1.465493e+04	1.390895e+03	2.672245e+04	4.530164e+03	2.640996e+04	3.070811e+04	1.686522e+04
std	8.234312e+06	3.893298e+05	6.428302e+04	5.699652e+05	2.359124e+05	1.514730e+06	5.770590e+05	7.512756e+05
min	3.000000e+04	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00
25%	6.000000e+05	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00
50%	2.260000e+06	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00
75%	8.000000e+06	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00
max	4.000000e+07	2.000000e+07	4.000000e+06	2.000000e+07	1.480000e+07	1.000000e+08	2.070800e+07	4.000000e+07

The summary statistics confirm an earlier assumption, the feature columns are made up of primarily zero values and a few large values. We also need to verify there no feature columns missing data or NA values. The following code checks if there are any missing values in the training set.

```
# missing values check
missing = train_df.isnull().sum(axis=0).reset_index()
missing.columns = ['column_name', 'missing_total']
missing = missing[missing['missing_total']>0]
missing = missing.sort_values(by='missing_total')
missing
```

	column_name	missing_total

The output table shows there are now missing values in the feature columns.

Now we need to determine if there are any feature columns with all zero values. If so, these columns will need to be removed from the training set because they do not provide any value to the model and would only increase the computation time. The code below shows how it determined if any feature columns were all zero values.

```
unique_df = train_df.nunique().reset_index()
unique_df.head()
```

	index	0
0	ID	4459
1	target	1413
2	48df886f9	32
3	0deb4b6a8	5
4	34b15f335	29

```
unique_df.columns = ['col_name', 'unique_count']
constant_col_df = unique_df[unique_df['unique_count']==1]
constant_col_df.shape
```

```
(256, 2)
```

To determine the columns with only zero values the unique numbers in each column were counted. Columns with 1 unique number are zero and those will be removed. 256 columns were identified with constant values. A quick sanity check was run looking at the sum of columns with unique values of 1 and 32.

```
constant_col_df.head()
```

	col_name	unique_count
28	d5308d8bc	1
35	c330f1a67	1
38	eeac16933	1
59	7df8788e8	1
70	5b91580ee	1

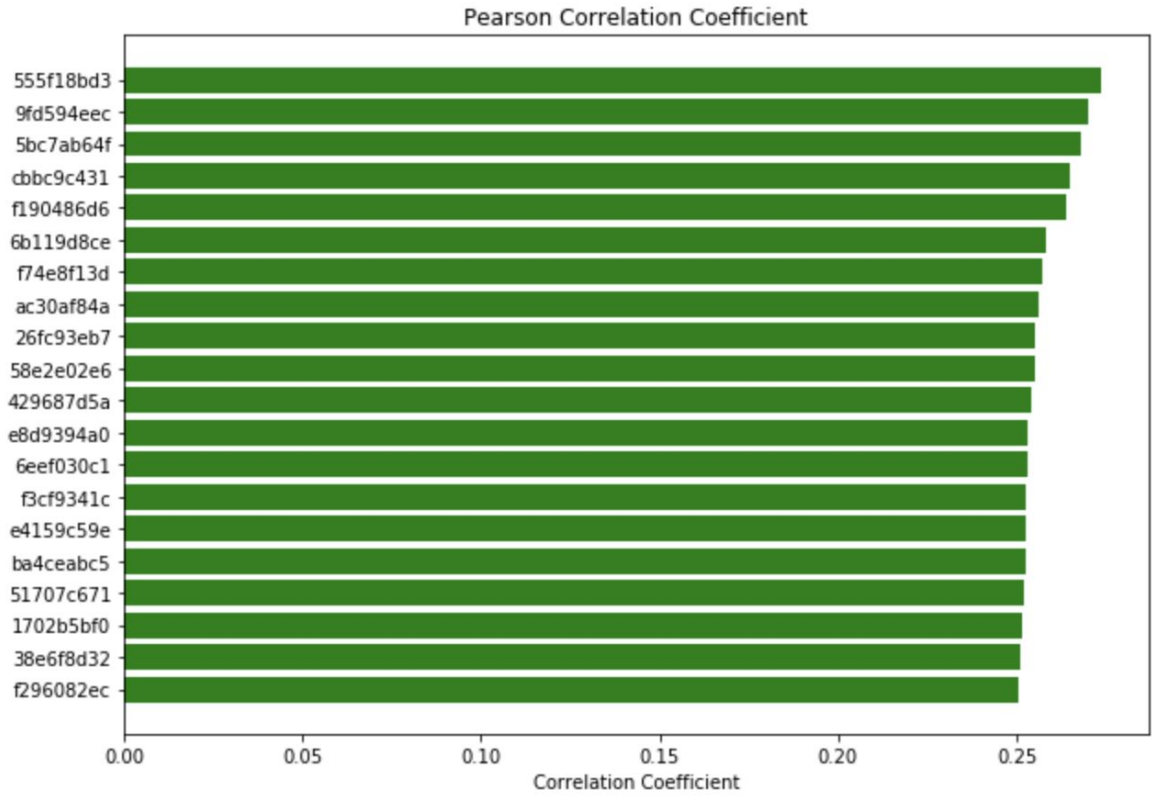
```
print("Column: d5308d8bc, with unique count 1 =", train_df['d5308d8bc'].sum())
print("Column: 5b91580ee, with unique count 1 =", train_df['5b91580ee'].sum())
print("Column: 48df886f9, with unique count 6 =", train_df['48df886f9'].sum())
```

```
Column: d5308d8bc, with unique count 1 = 0
Column: 5b91580ee, with unique count 1 = 0
Column: 48df886f9, with unique count 6 = 65346333.31999999
```

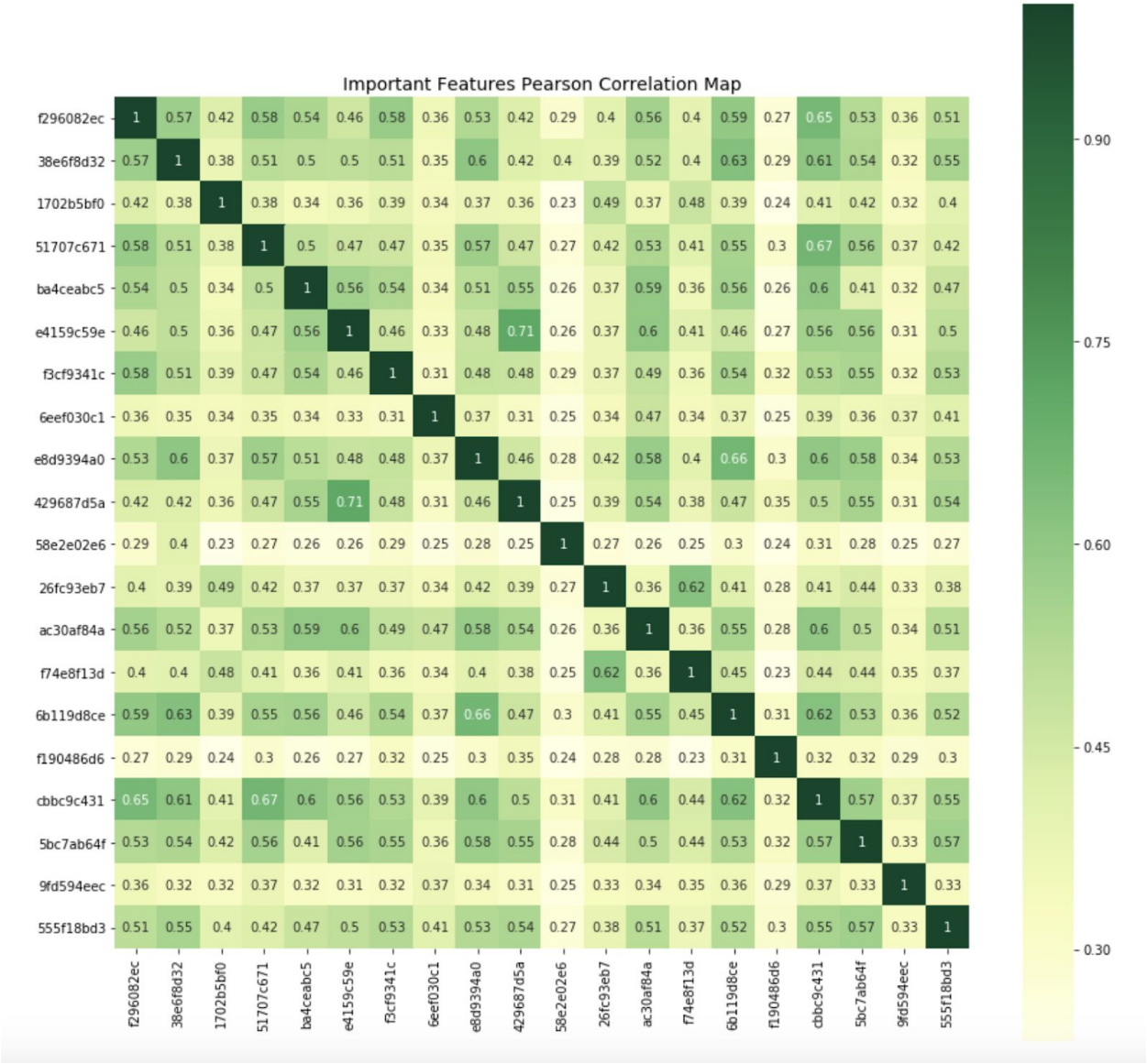
The 256 columns identified as having no values were removed from the training set and test set.

Feature Correlation

The data set has a large number of feature columns > 4000, we can use correlation to understand which features are correlated to the target. This will allow us to identify and use the highly correlated variables in our models. The Pearson correlation method will be used to determine the correlation coefficients. The first chart is a list of the correlation coefficients with an absolute correlation greater than 0.25; 20 feature columns were returned.



Next, we will use the 20 correlation coefficients to create a heat map to find importance.



The correlation map shows 2 features with Pearson correlation higher than 0.7 with each other. Not a lot of highly correlated features are being seen. We can also use feature importance from our machine learning models to verify our correlation is returning similar results.

Feature Engineering

Since we are not experts in the financial industry it is difficult to know what features could be created relevant to the models. We do know that the values and zeros in the data are important to the data set. We can create features by summing the values and zeros in each row. We can also run aggregations on the rows looking at Mean, Median, Mode, Max, Variance and Standard Deviation. These will be easy features to add to give the models more data to learn from. The code for the 3 features is shown below.

```
# Sum Zeros in each row

def add_sum_zeros(train, test, features):
    flist = [x for x in train.columns if not x in ['ID','target']]
    if 'SumZeros' in features:
        train.insert(1, 'SumZeros', (train[flist] == 0).astype(int).sum(axis=1))
        test.insert(1, 'SumZeros', (test[flist] == 0).astype(int).sum(axis=1))
    flist = [x for x in train.columns if not x in ['ID','target']]

    return train, test

train_X, test_X = add_sum_zeros(train_X, test_X, ['SumZeros'])
```

```
# Sum Values in each row

def add_sum_values(train, test, features):
    flist = [x for x in train.columns if not x in ['ID','target']]
    if 'SumValues' in features:
        train.insert(1, 'SumValues', (train[flist] != 0).astype(int).sum(axis=1))
        test.insert(1, 'SumValues', (test[flist] != 0).astype(int).sum(axis=1))
    flist = [x for x in train.columns if not x in ['ID','target']]

    return train, test

train_X, test_X = add_sum_values(train_X, test_X, ['SumValues'])
```

```
# Compute aggregates

def add_other_agg(train, test, features):
    flist = [x for x in train.columns if not x in ['ID','target','SumZeros','SumValues']]
    if 'OtherAgg' in features:
        train['Mean']   = train[flist].mean(axis=1)
        train['Median'] = train[flist].median(axis=1)
        train['Mode']   = train[flist].mode(axis=1)
        train['Max']    = train[flist].max(axis=1)
        train['Var']    = train[flist].var(axis=1)
        train['Std']    = train[flist].std(axis=1)

        test['Mean']   = test[flist].mean(axis=1)
        test['Median'] = test[flist].median(axis=1)
        test['Mode']   = test[flist].mode(axis=1)
        test['Max']    = test[flist].max(axis=1)
        test['Var']    = test[flist].var(axis=1)
        test['Std']    = test[flist].std(axis=1)
    flist = [x for x in train.columns if not x in ['ID','target','SumZeros','SumValues']]

    return train, test

train_X, test_X = add_other_agg(train_X, test_X, ['OtherAgg'])
```

Now that we have created some additional features we can move to developing the models.

Machine Learning

The goal of the project is to **determine the amount or value of a customer's transaction.** We will be using regression models to determine the values for the customers. The two models we will be using are GradientBoostingRegressor and RandomForestRegressor. Ensemble methods are good learners, work well on large datasets and are accurate classifiers. Since this is a Kaggle competition we have a test set which we will use for the final prediction which is submitted for scoring on the leaderboard. To test our models the training set is split into training and test sets (shown below).

```
train_X = train_df_1.drop(['ID','target'], axis=1)
train_y = np.log1p(train_df_1['target'].values)
test_X = test_df.drop(constant_col_df.col_name.tolist() + ['ID'], axis=1)
```

```
# create train and test sets from training data
X_train, X_test, y_train, y_test = train_test_split(train_X, train_y, test_size=0.30, random_state=28)
```

To get a baseline for our models we run the algorithms with the default parameters. In the table below are the baseline results for the two models. The result was calculated using the Root Mean Squared Log Error which is the scoring method for the competition. The scoring formula is shown below. Using GradientBoostingRegressor and RandomForestRegressor we fit the training data to

our models and then predict the results using the training and test data (code for the baseline models is located on Github: Report Code)

```
# define scoring metric

def rmsle(y_pred, y_test) :
    assert len(y_test) == len(y_pred)
    return np.sqrt(np.mean((np.log1p(y_pred) - np.log1p(y_test))**2))
```

Method	Training Score	Test Score	Difference
Gradient Boosting	1.2345	1.3695	0.135
Random Forest	0.6338	1.3921	0.7583

The RMSLE score that is ideal is 0 that means there is no error obviously that is impossible, therefore a score as close to that is ideal. In the table looking at the results we have a training score and test score. The test score used here is the test set we created from in our test train split. We want to see the difference between the training and test sets. There is a difference between both models but the larger difference is on Random Forest. This indicates that overfitting is definitely taking place on Random Forest and slightly on Gradient Boosting. With both of these models there are key parameters that can be changed to change the performance and results. To do this hyperparameter tuning is required and this is done using Grid Search Cross Validation (GridSearchCV). GridSearchCV takes a range of defined parameters and runs validations on the training data to determine the best parameters. The code for the parameter tuning of both models and the results of the best estimator is shown below.

```
# gradient boosting tuning
param_grid_gbr = {'n_estimators': [300, 500, 700, 1000],
                  'learning_rate': [0.1, 0.05, 0.02,  0.01],
                  'max_depth': [4, 6],
                  'min_samples_leaf': [3, 5, 9, 17],
                  'max_features': [1.0, 0.3, 0.1]
                 }
gbr_gs = GradientBoostingRegressor(random_state=20)

gbr_gridcv = GridSearchCV(estimator=gbr_gs,
                          param_grid=param_grid_gbr,
                          cv=3,
                          refit=True,
                          scoring=rmsle_score,
                          verbose=1,
                          n_jobs=-1)
```

```
GradientBoostingRegressor(alpha=0.9, criterion='friedman_mse', init=None,
             learning_rate=0.01, loss='ls', max_depth=6, max_features=0.1,
             max_leaf_nodes=None, min_impurity_decrease=0.0,
             min_impurity_split=None, min_samples_leaf=3,
             min_samples_split=2, min_weight_fraction_leaf=0.0,
             n_estimators=1000, presort='auto', random_state=20,
             subsample=1.0, verbose=0, warm_start=False))
```

```
# random forest tuning
param_grid_rf = {'bootstrap': [True],
                 'max_depth': [100, 110, 120, 130],
                 'max_features': [2, 3],
                 'min_samples_leaf': [3, 4, 5],
                 'min_samples_split': [8, 10, 12],
                 'n_estimators': [700, 800, 1000, 1100]
                }

rf_gs = RandomForestRegressor()

rf_gridcv = GridSearchCV(estimator=rf_gs,
                         param_grid=param_grid_rf,
                         cv=3,
                         scoring=rmsle_score,
                         refit=True,
                         n_jobs=-1,
                         verbose=2)
```

```
GradientBoostingRegressor(alpha=0.9, criterion='friedman_mse', init=None,
             learning_rate=0.01, loss='ls', max_depth=6, max_features=0.1,
             max_leaf_nodes=None, min_impurity_decrease=0.0,
             min_impurity_split=None, min_samples_leaf=3,
             min_samples_split=2, min_weight_fraction_leaf=0.0,
             n_estimators=1000, presort='auto', random_state=20,
             subsample=1.0, verbose=0, warm_start=False))
```

Using the tuned parameters a new score is calculated on the test set (results shown in the table).

Method	Training Score	Test Score	Tuned Score
Gradient Boosting	1.2345	1.3695	1.3464
Random Forest	0.6338	1.3921	1.5600

The tuned score results did not show an improvement over the baseline scores. Factors affecting this could be: overfitting, small sample size, more feature engineering and possibly performing PCA due to the large number of feature columns. A good indicator will be to use the baseline and tuned models on the final test set and submit to the competition. The test submission data set had over 10x as many rows as the training set. There is a chance the models will perform better with more data. Submission csv files were created containing an 'ID' column and a 'Target' column which is a value for each customer. In the table there are two leaderboard scores: 1) public leaderboard and 2) private leaderboard (which is calculated with approximately 51% of the test data).

Method	Public Leaderboard	Private Leaderboard
Gradient Boosting Baseline	1.49286	1.45798
Random Forest Baseline	1.54007	1.52041
Gradient Boosting Tuned	1.50086	1.46789
Random Forest Tuned	1.69940	1.67530

The results show that Gradient Boosting was our best performing model. The Random Forest scores show that there still appears to be overfitting going on. The Gradient Boosting scores are very close and does not appear to be any overfitting.

Conclusion and Next Steps

Based on the results from the competition scoring Gradient Boosting was the best performing model. The Random Forest model showed definite signs of overfitting which led to the higher scores. There are a number of areas to explore further that would surely improve the scores for both models:

1. Reduce overfitting by creating more features or performing PCA to reduce the number of features to only the significant ones
2. Improve feature engineering, the features we created were minimal and more impactful features could be created
3. Improve parameter tuning by tuning fewer parameters, using important features to tune or providing more data to model

References

1. https://www.kaggle.com/c/santander-value-prediction-challenge
2. https://github.com/jpeterson28/DSCT_Capstone2