Machine Learning Engineer Nanodegree
Capstone Project Report – Santander Customer Transaction Prediction

Natu Lauchande
June, 2019

Table of Contents

I. Definition

Project Overview

Financial services are at the cornerstone of the modern society opportunity value chain for individuals and business. Santander is a bank that thrives to know their customers better to serve them correctly. Part of providing the customers with the right financial choices is to be able to know and predict their desires [1].

The topic of this project is to predict if bank customers will make a specific transaction in the future based on a set of anonymous features. Predicting customer propensity/suitability for transaction or willingness is paramount on bringing inclusion and low cost financial services for underprivileged communities [6].

Predicting if a customer for instance will make a specific transaction might help the financial institution provision resources if it makes business sense. A special important application of predicting transaction is in the context of financial fraud. Having a performant transaction fraud detection system can lower financial risk for institutions [7].

Being able to predict what transaction is more likely a banking user would make can also enable more ways to connect with customers: using appropriate digital channels, for example SMS and mobile, apps enables engagement with customers who have previously been unreachable by and therefore invisible to more traditional environments.

The fact that the current project focus on anonymized data (no identification of the variable names and basically numerical data) opens the possibility of using the outcomes of this project in diverse contexts aligned with data privacy.

Problem Statement

This project will be approached as a supervised learning classification problem. The first strategy used to solve this project is to find ways to decrease the imbalance of the classes.

The problem statement of this project is based on the Kaggle competition :
https://www.kaggle.com/c/santander-customer-transaction-prediction

The goal of this project is to solve the problem of predicting whether a customer will make a given unidentified transaction given a set of features and historical data. The core of the project is to identify which customers will make a specific transaction in the future, irrespective of the amount of money transacted [1].

The solution for this project will consists of the following:

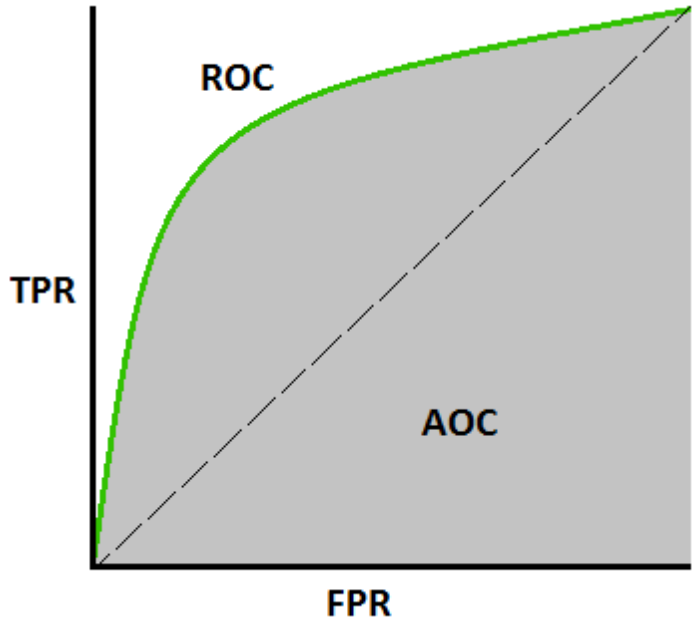
- An exploratory analysis of the training dataset provided by the Santander competition.

- A predictive solution to the problem.
- A notebook and reporting depicting the solution process to the problem.

Metrics

In terms of metrics to objectively compare the different models the Area Under Curve (AUC) of the Receiver Operator characteristics is the chosen metric a comparison of how well a binary classifier distinguishes between the two classes at play in the specific problem.

- AUC ROC curve metric as suggested by the official competition. [2]



Source : https://towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5

True Positive Rate : $\frac{Number\ of\ True\ Positives}{Number\ of\ True\ Positives + Number\ of\ False\ Negatives}$

False Positive Rate : $\frac{Number\ of\ False\ Positives}{Name\ of\ False\ Positives + Number\ of\ True\ Positives}$

The area under the curve is able to capture the quality measuring the quality of a scoring function. The best possible ROC curve has the area 1 the closer your ROC curve area is to 1 the better is the classifier. The maximum uncertainty classifier would yield an area of 0.5 and would be the diagonal line of the graph of the figure above [3].

Given that the current problem is a binary classifier the AUC ROC metric will allow us to compare correctly the quality of the different classifiers.

II. Analysis

Data Exploration

The raw data consists of 200000 rows and 202 columns. It contains one categorical variable for the target (0,1) and 1 string variable for the ID code and the remaining of the variables are continuous.

	ID_code	target	var_0	var_1	var_2	var_3	var_4	var_5	var_6	var_7	...	var_190	var_191	var_192	var_193	var_194	var_195	var_196	var_197	v
0	train_0	0	8.9255	-6.7863	11.9081	5.0930	11.4607	-9.2834	5.1187	18.6266	...	4.4354	3.9642	3.1364	1.6910	18.5227	-2.3978	7.8784	8.5635	
1	train_1	0	11.5006	-4.1473	13.8588	5.3890	12.3622	7.0433	5.6208	16.5338	...	7.6421	7.7214	2.5837	10.9516	15.4305	2.0339	8.1267	8.7889	
2	train_2	0	8.6093	-2.7457	12.0805	7.8928	10.5825	-9.0837	6.9427	14.6155	...	2.9057	9.7905	1.6704	1.6858	21.6042	3.1417	-6.5213	8.2675	
3	train_3	0	11.0604	-2.1518	8.9522	7.1957	12.5846	-1.8361	5.8428	14.9250	...	4.4666	4.7433	0.7178	1.4214	23.0347	-1.2706	-2.9275	10.2922	
4	train_4	0	9.8369	-1.4834	12.8746	6.6375	12.2772	2.4486	5.9405	19.2514	...	-1.4905	9.5214	-0.1508	9.1942	13.2876	-1.5121	3.9267	9.5031	

5 rows × 202 columns

It can be seen form the table description below that some of the standard deviations below are very high. This dataset can definitely benefit from some normalization down the line.

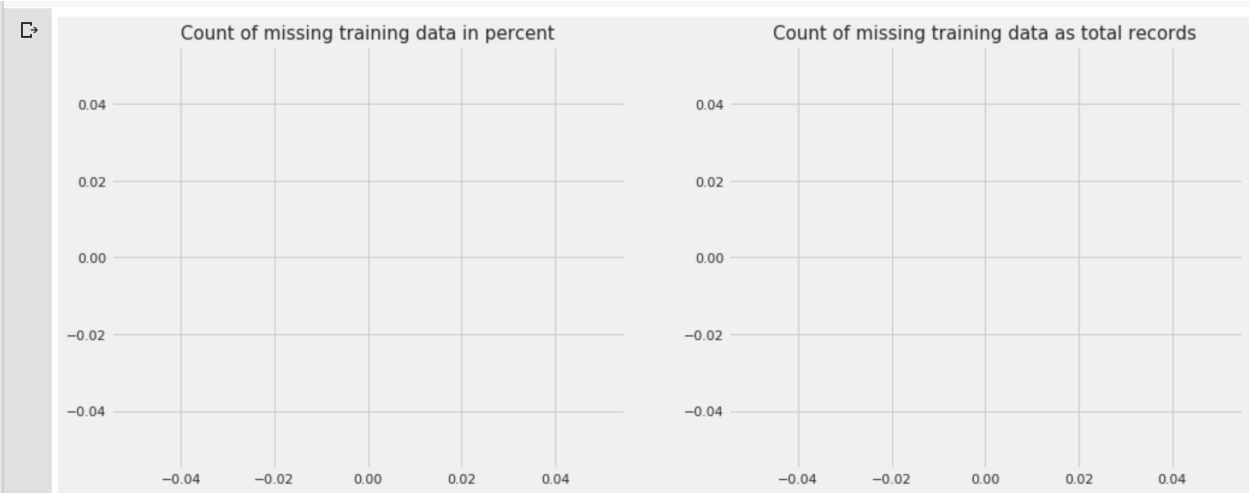
	target	var_0	var_1	var_2	var_3	var_4	var_5	var_6	var_7	var_8	...	var_
count	200000.000000	200000.000000	200000.000000	200000.000000	200000.000000	200000.000000	200000.000000	200000.000000	200000.000000	200000.000000	...	200000.000
mean	0.100490	10.679914	-1.627622	10.715192	6.796529	11.078333	-5.065317	5.408949	16.545850	0.284162	...	3.234
std	0.300653	3.040051	4.050044	2.640894	2.043319	1.623150	7.863267	0.866607	3.418076	3.332634	...	4.559
min	0.000000	0.408400	-15.043400	2.117100	-0.040200	5.074800	-32.562600	2.347300	5.349700	-10.505500	...	-14.093
25%	0.000000	8.453850	-4.740025	8.722475	5.254075	9.883175	-11.200350	4.767700	13.943800	-2.317800	...	-0.058
50%	0.000000	10.524750	-1.608050	10.580000	6.825000	11.108250	-4.833150	5.385100	16.456800	0.393700	...	3.203
75%	0.000000	12.758200	1.358625	12.516700	8.324100	12.261125	0.924800	6.003000	19.102900	2.937900	...	6.406
max	1.000000	20.315000	10.376800	19.353000	13.188300	16.671400	17.251600	8.447700	27.691800	10.151300	...	18.440

8 rows × 201 columns

Exploratory Visualizations

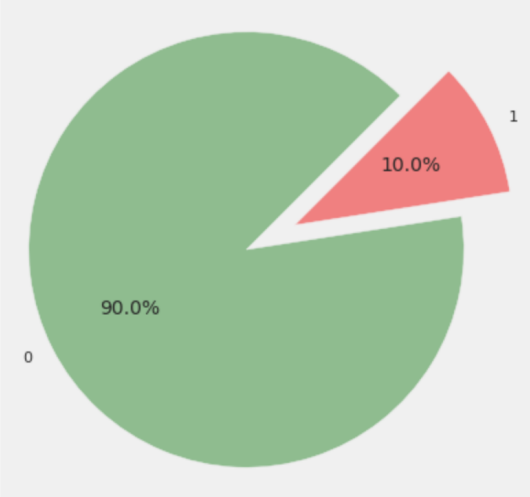
Missing data visualisation

The graph below displaying basically no missing data in the training dataset. What is perhaps good news in terms of less work during data preparation to impute missing variables.



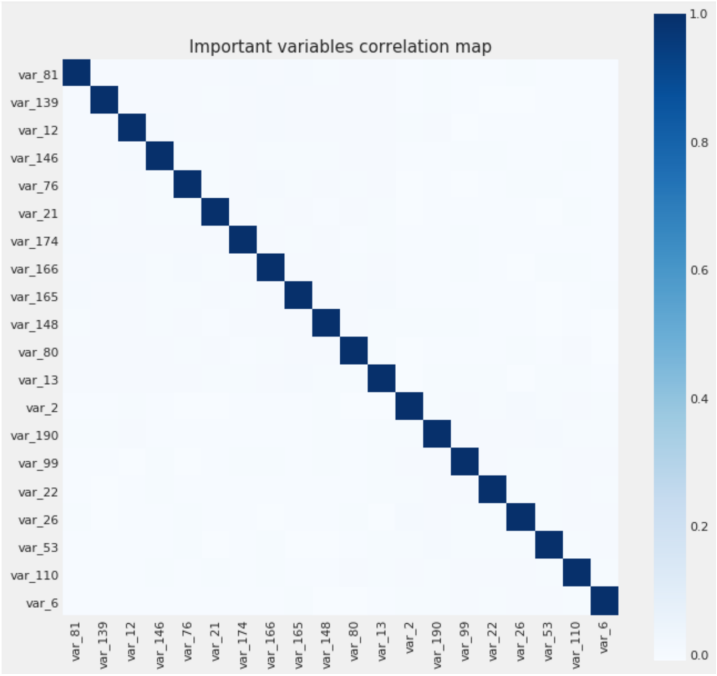
Target description

This section describes the target variable and its distribution. From the above data we clearly have a category imbalance problem in here. A ration of 1:9 between a target of taking a transaction and not taking a transaction.



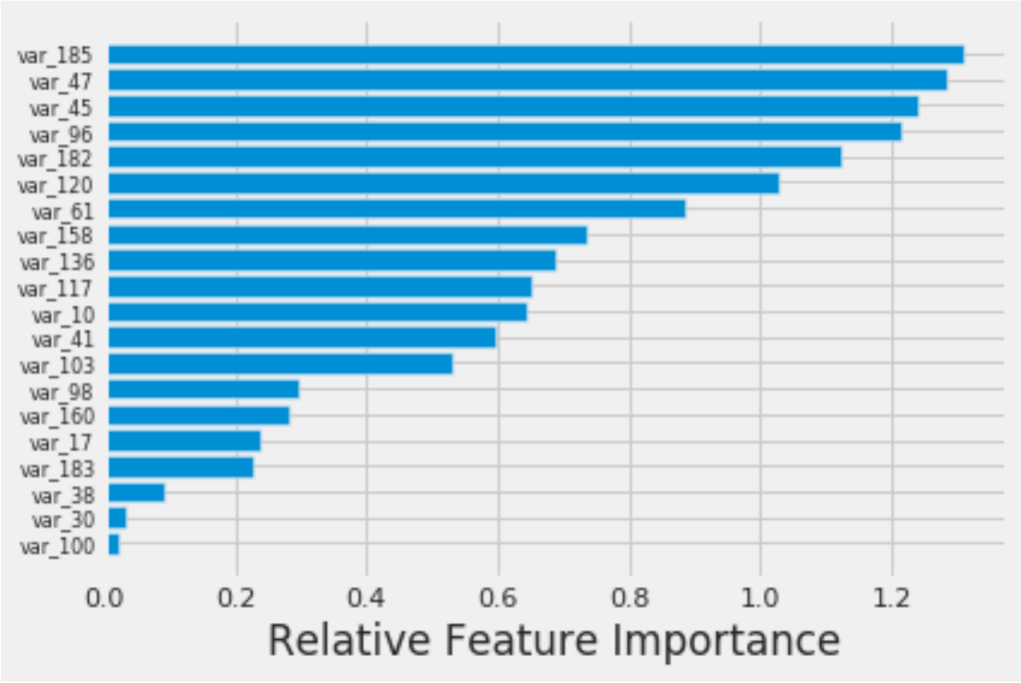
Features Correlations

The correlation analysis helps us understanding how 2 variables are related. It's clear from the figure below that there is very little correlation between the variables in the dataset.

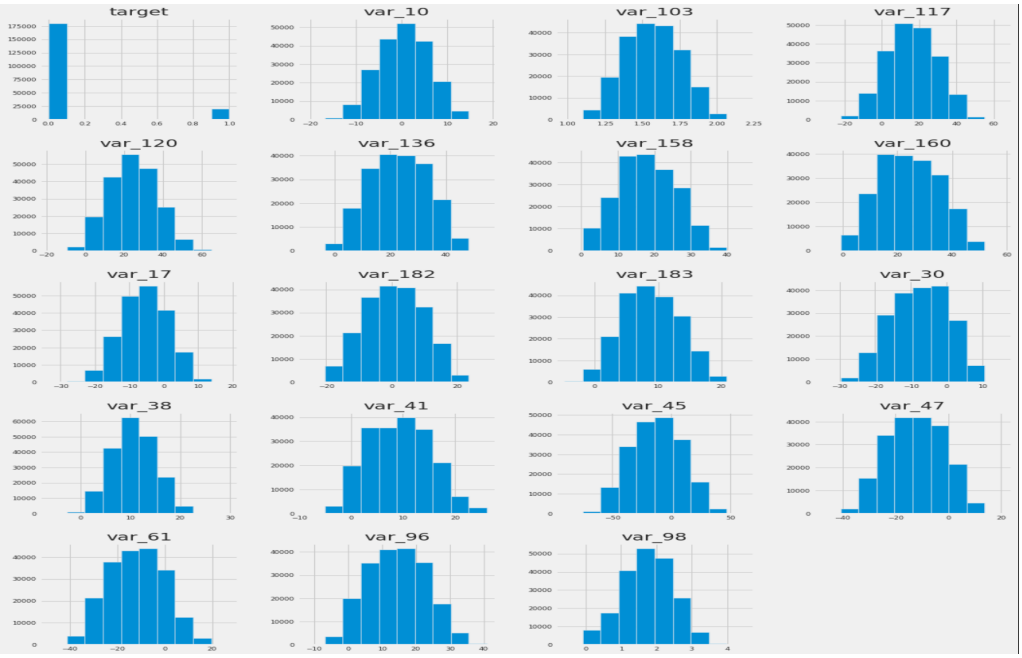


Top Features and Target Histogram

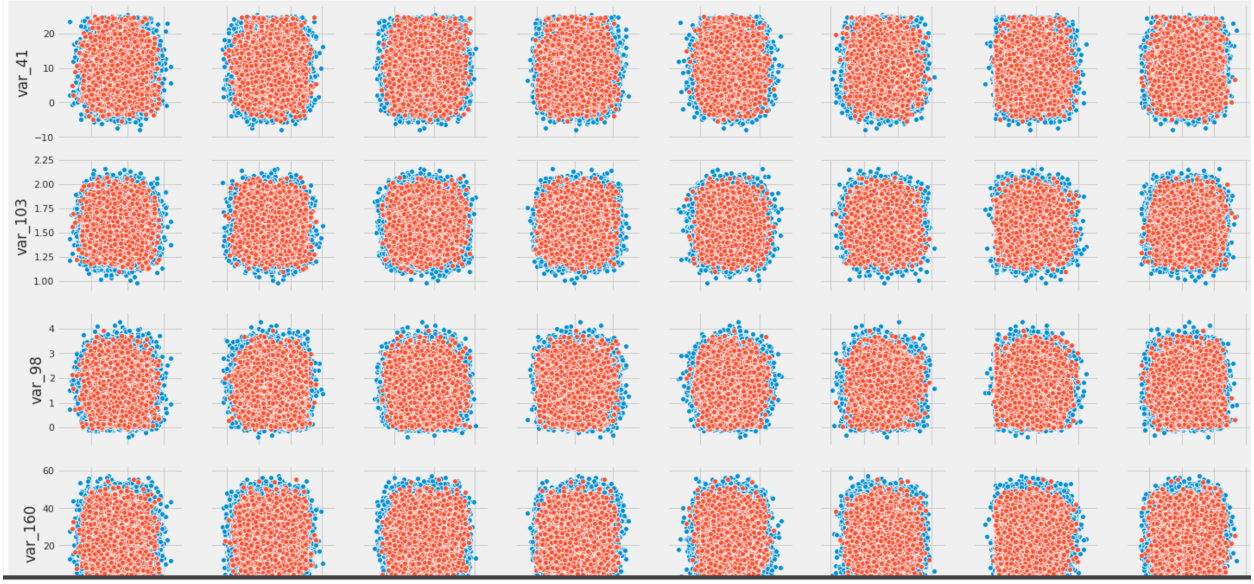
In the benchmark section a simple logistic regression was run and and a feature importance list was extracted. A limited set of relevant features would allow us to have a look at different properties of the dataset.



On the distribution graph above becomes very clear an underlying normal distribution of the important variables this. The target imbalance is definitely super distinct from the top 20 most important features.



Another important visualization is the relation between the restricted set of relevant variables and the target pairwise:



No significant conclusion can be driven from the insights provided from the visualizations above. Definitely confirming the low correlation between the variables showing just the distribution between binary target variable.

Algorithms and Techniques

As noticed before the training dataset suffers from class imbalance that needs to be addressed during model iteration and taken into consideration by mitigating using the most appropriate strategies (example: Smote, sampling, penalties, etc.) [8].

The general approach on solving the problem of this project is the following:

1. Apply standard data pre-processing:

    - Data normalization scaling
    - Category imbalance mitigation
    - Feature selection based on importance ranking

2. Run a baseline classifier pipeline on each stage of the data processing to detect improvements. Having a pipeline of classifiers ready to use facilitates the execution of the process and to denote improvements from the different standard techniques.
3. Select from step 2 the most promising classifier and execute further tuning.

The classifier pipeline consists of the following algorithms:

Logistic Regression

Logistic regression is one of the most classical machine learning algorithms. It's a regression technique that instead of using a linear function to fit the training point it uses a sigmoid function. [3]

Among main strengths of the logistic regression are the following:

- Interpretability (the sklearn implementation provides feature importance that can be used for feature selection)
- Fast to run
- Natural fit for binary classification problems

Naive Bayes

Naive Bayes is a simple probabilistic classifier. In its simplistic form is an implementation of the Bayes Theorem with the feature set with the "naive" assumption that features are not correlated. It's very common as a baseline for text classification problems. [10]

The main motive behind the choice for this particular problem is the fact that it's extremely fast and the low correlation on the feature set detected during Data Exploration.

Ensemble techniques are based on a combination of multiple weak learners that can be used for classification or regression. [11]

A common type of ensembles are the ones based on tree methods (example: Gradient Boosting, Random Forests), presenting a combinations of trees that in particular are performant in one component of the target distribution.

Ensemble techniques are a very popular approach to a lot of popular tabular/transactional data problems with significant success across industries.

Random Forest

The random forest algorithm basically uses the mode or mean of the different trees identified by decision trees learned by random selections of the training data [12].

The equation above represents the situation when we are averaging the data.

XGBoost

It's a technique based on Gradient Boosting that combines weak classifiers in sequence where each classifier is created where the data of mispredictions in the training data is augmented. The implementation is relatively fast and simple to use.

Catboost

Catboost is a new generation of boosting frameworks that uses some new insights and has out of the optimization tuning and optimization. [13].

The main reason to choose this technique as one of the candidates for this project was to evaluate also the capabilities of this new framework.

Neural Networks

A classical technique recently popularized by the industry success of Deep Learning techniques for perception data (audio, video, speech) and natural language problems.

Defined as collection of connected units or nodes called artificial neurons, each connection can transmit a signal from one artificial neuron to another. Each neuron receives a signal from an external source (another neuron or input data) and is able to learns an expected classification in a

supervised learning setting. An artificial neuron mimics the working of a biophysical neuron with inputs and outputs. [3]

The two techniques described below (Data Scaling, and Handling imbalanced datasets) are basically data processing techniques executed to improve the performance of machine learning algorithms on the training data and will be applied after a given baseline.

Data Scaling

Machine learning algorithms and techniques generally assume out of the box the input being of standard normal distributed data. The scaling of data facilitates optimization techniques and improves learning on standardized data. Sklearn provides a simple method that will be used in this project: StandardScaler [15]

Handling imbalanced datasets

Given the fact that our initial dataset is highly unbalanced some techniques must be used to improve the accuracy rate of the dataset.

Most of the inspiration on the techniques used in this project involved looking at the following project : https://www.kaggle.com/rafjaa/resampling-strategies-for-imbalanced-datasets

Benchmark

The benchmark for this classification is based on a simple out of the box Logistic Regression binary classification over the data implemented with the intention of improving the model with insights gained over the execution of the project.

Another possibility of a benchmark could be the Kaggle Competition Leaderboard in itself, after a careful evaluation of some of the submissions it became clear to me that most of the solutions on the Kaggle projects are based on deep expertise and specialized competition specific knowledge (stacking, over fitting and feature engineering). In line with broadening personal knowledge in Data Science and Machine Learning there is a deliberate choice on this project for a simpler benchmark that will allow to improve on knowledge gained over the Nanodegree.

Benchmark model will the out of the box vanilla logistic regression outlined below:

```
from sklearn.linear_model import LogisticRegression
```

```
X=train[features]
y=train[target]

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=0)
X_train.head()

clf = LogisticRegression(random_state=0, solver='lbfgs', multi_class='multinomial').fit(X_train,
y_train)

y_pred = clf.predict(X_test)
y_probas = clf.predict_proba(X_test)

from sklearn.metrics import roc_auc_score
roc_auc_score(y_test, y_pred)
```

The initial roc_auc_score was: 0.6089

A standard 70/30 ratio was chosen to split between testing and training data. This split will and training data ratio is used throughout the execute the project.

To calculate the benchmark metrics sklearn metric package was used.

III. Methodology

This section contains a more detailed description and execution documentation of the approach to tackle this particular problem. The summary of this approach is basically data pre-processing tuning followed by an algorithmic refined depicted by the training classifiers pipeline and the further refinement processes.

Data Preprocessing

For data normalization the Sklearn ScaledScaler library was used run a transform over the data pre-selected. As previously mentioned scaling was precisely used given the fact that some of the initial standard deviations results were high for some of the columns >3 in some cases.

```
X_scaled = pd.DataFrame(StandardScaler().fit_transform(X))

X_train_scaled, X_test_scaled, y_train_scaled, y_test_scaled = train_test_split(X_scaled, y,
test_size=0.3, random_state=0)
```

New train and tests datasets were created with the scaled data to be used during refinement and decision around the best classifier.

With regard to training dataset imbalance further analysis was executed over this particular problem to detect the minority class. The package SMOTE was used to handle category imbalanced data.

```
print("Before OverSampling, counts of label '1': {}".format(sum(y_train_scaled==1)))
print("Before OverSampling, counts of label '0': {} \n".format(sum(y_train_scaled==0)))

sm = SMOTE(random_state=2)
X_train_res, y_train_res = sm.fit_sample(X_train_scaled, y_train_scaled.ravel())

print('After OverSampling, the shape of train_X: {}'.format(X_train_res.shape))
print('After OverSampling, the shape of train_y: {} \n'.format(y_train_res.shape))

print("After OverSampling, counts of label '1': {}".format(sum(y_train_res==1)))
print("After OverSampling, counts of label '0': {}".format(sum(y_train_res==0)))
```

```
Output :

Before OverSampling, counts of label '1': 13984
Before OverSampling, counts of label '0': 126016

After OverSampling, the shape of train_X: (252032, 200)
After OverSampling, the shape of train_y: (252032,)

After OverSampling, counts of label '1': 126016
After OverSampling, counts of label '0': 126016
```

Training and classifier pipeline

Since there was a need to run multiple classifier. Decided to investigate flexible approaches to run multiple classifiers in an easy manner and manageable manner. A promising approach after analyzing a couple was using the suggestion available in this Kaggle kernel https://www.kaggle.com/jeffd23/10-classifier-showdown-in-scikit-learn .

The following decisions were made in order to create a scalable and streamlined approach for experiments:

1. Use the standard sklearn interface that includes fit/predict for all the algorithms
2. Modified the code in order to use the minimal amount of code and not duplicate prediction and metrics code
3. Use GPU's capability whenever possible in order to speed up turnaround to get results.

The function *run tabular prediction pipeline* contains a list of classifiers as outlined in the algorithms section and a standard way to obtain the AUC ROC metric (the chosen metric for the project) and allow us to run during each step of the refinement process to understand the improvement of each technique over the pipeline of classifiers.

```
def run_tabular_prediction_pipeline(X_train, X_test, y_train,  y_test):

 nn_batch_num = int((X_train.shape)[0]/100) # Magic guess number
 input_dim_num = X_train.shape[1]

 #define closure for NN callback
 def baseline_model_nn():
  # create model
  model = Sequential()
  model.add(Dense(8, input_dim=input_dim_num, activation='relu'))
  model.add(Dense(2, activation='softmax'))
```

```
    # Compile model
    model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
    return model

  classifiers = [
      RandomForestClassifier(),
      LogisticRegression(random_state=0, solver='lbfgs', multi_class='multinomial'),
      GaussianNB(),
      XGBClassifier(objective='binary:logistic', tree_method='gpu_hist'),
      CatBoostClassifier(task_type = "GPU", verbose=False),
      KerasClassifier(build_fn=baseline_model_nn, epochs=100, batch_size=nn_batch_num,
verbose=0)
  ]
  # Logging for Visual Comparison
  log_cols=['Classifier', "AUC ROC"]
  log = pd.DataFrame(columns=log_cols)

  for clf in classifiers:
      try:
        clf.fit(X_train, y_train)
        train_predictions = clf.predict(X_test)
        roc_auc = roc_auc_score(y_test, train_predictions)
      except:
        #To handle failure situations
        roc_auc = None
      name = clf.__class__.__name__
       log_entry = pd.DataFrame([[name, roc_auc]], columns=log_cols)
      log = log.append(log_entry)

  return log
```

Baseline**: **Run classifiers pipeline without any data processing

This experiment involves running the classifier pipeline over all the data and collecting the required metric . Further improvements to the solution will be documented in the Refinement section.

	Classifier	AUC ROC
0	RandomForestClassifier	0.506549
0	LogisticRegression	0.608911
0	GaussianNB	0.670789
0	XGBClassifier	0.508230
0	CatBoostClassifier	0.645972
0	KerasClassifier	0.606637

Refinement

Step 1: Add data scale normalization

In this section we run the data over the normalized data with the training datasets produced by SKLearns standard scaler.

	Classifier	AUC ROC
0	RandomForestClassifier	0.507121
0	LogisticRegression	0.628388
0	GaussianNB	0.670871
0	XGBClassifier	0.508230
0	CatBoostClassifier	0.646054
0	KerasClassifier	0.634216

Step 2: **Class imbalance mitigation with Minority Oversampling**

In this section we run the classifiers pipeline strategy of target class imbalance by oversampling the minority class.

	Classifier	AUC ROC
0	RandomForestClassifier	0.529920
0	LogisticRegression	0.774191
0	GaussianNB	0.525620
0	XGBClassifier	NaN
0	CatBoostClassifier	0.658729
0	KerasClassifier	0.749112

The XGB classifier had an issue when handling the augment data format, special provisioning would have to be conducted to fix this problem. It was decided to remove from the Step 2 iteration. CatBoost classifier is by itself a representative of Gradient Boosting family of algorithms.

Step 3: Tune and improve on most promising classifier candidate

Most best candidate from a metrics perspective is Logistic Regression basesd classifier with significant gains over baseline and ahead in the metric perspective from the second one the KerasClassifier. Since the data preprocessing perspective was heavily explored on the previous refinement steps at this stage we will revert to Hyper parameter tuning of the Logistic Regression solution.

The parameter tuning consisted of exploring the solution space of the following elements :

C - Regularization parameter solver - The method used to solved the regression equation

Solver – The solver selected to resolve the linear regression problem.

```
grid = { 'C': np.power(10.0, np.arange(-10, 10)) ,
        'solver': ['newton-cg','lbfgs'],


      }
```

```
clf = LogisticRegression(penalty='l2', random_state=777, max_iter=100, tol=10)
gs = GridSearchCV(clf, grid, scoring='roc_auc', cv=10)
gs.fit(X_train_res, y_train_res)
```

Following the training process we will execute the best classifier over the test data in order to capture the needed metrics for this project. The listing above presents the different tested and chosen parameters. The AUC ROC for this case was : AUC ROC: 0.774630059890547

The hyper parameter tuning process yielded a tiny improvement over Step 2 but nonetheless the cross validations process brings into table a more robust classifier that generalizes better.

IV. Results

Model Evaluation and Validation

The results section of this project was intertwined in the previous section being the main results of this project the following:

1. Comparing multiple Machine algorithm algorithms for the Santander Customer Transaction prediction.

2. Descriptive analysis of the training data of the competition.

3. A winning algorithm from the analysis of the project.

In summary we were able to improve from the benchmark in terms of the chosen metric in excess of 20%, what is by itself a good result.

Overall, the best classifier came from House Grid Search 1. The F_1 score on the validation set for the refined or optimized model on the house data was 0.770, which does a much better job at predicting EV houses than the benchmarks which were 0.000, 0.367, 0.487 for all 0, alternating 01, and all 1, respectively.

The standard deviation for the grid training time, best training cross-validation F1-Score, and F1-Score on the validation data were 18.865, 0.007, and 0.015, respectively. The training time was most sensitive to the parameter grid and random seed. The training cross-validation F1-score was very consistent. The F1-Score on the validation data also showed to be sensitive to the model. The training cross-validation score and the final validation score are close, indicating that the model at the house level is not doing too much overfitting. Interestingly, a couple of House Grid Searches showed the validation F1-Score to be higher than the training cross-validation F1-Score. This may be caused by chance or may be the result of shuffle-splitting.

Model evaluation and validation happened on Step 3 of refinement using GridSearch with a 10 fold cross validation aproach.

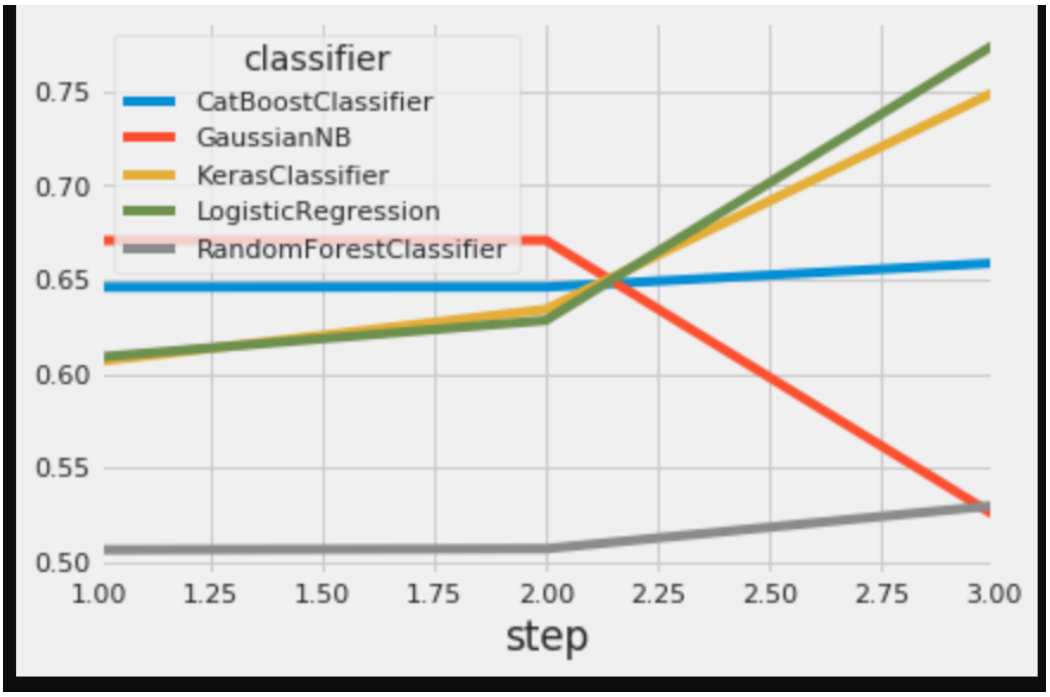
The final model will definitely be the last Logistic Regression approach on Step 3 given the fact that it's much simpler and widely understood algorithm and can provide an easy and interpretable model for the future users of the solution depicted on this notebook . The Neural Network could have been tweaked a bit more but the Logistic Regression approach was favoured given it's simplicity .

V Conclusion

This was definitely a very rewarding project from knowledge acquisition perspective. I had the opportunity to familiarize a bit more with the very rewarding world of Data Science competitions and learn with the community and available knowledge, certainly a source of info that will consider in my future projects.

Comparing the different models performance was at the central point of making a decision around the best model to execute this project.

It becomes clear from the visualisation below that Logistic Regression and KerasClassifier from a metrics AUC ROC perspective are the ones that gain the most with the processing techniques used in this project.



Reflection

Main points of reflection:

- This was a major independent undertanking at personal level in the Data Science world, very distinct experience than the rest of the projects of the course where guidance was readily available on the problem description.
- An important take away of this project is that simpler methods can outperform almost out the box the more advanced and modern techniques, for example : boosting and deep learning.

- Found this project particularly challenging and rewarding at same time given the fact that i had to participate for the first time in a Kaggle competition.
- During the execution of the project noticed that had to make important tradeoffs of some of the initial ideas on the proposal. For example : the focus on the software tools, deeper investigation of deep learning methods and couple of other approaches that didn't prove to be very relevant to the conclusion of the project.
- One significant learnign for me was on a couple of the algorithms was the ability to use GPU computing environment provided by Goolge Collab.

Improvements

Possible improvements of this project are the following :

- More advanced feature engineering.
- Using a pipelining that facilitates experiment management similar to Kubeflow or MLFlow.
- Implement a software solution that allow the seamless execution of this pipeline.
- More extensive hyperparameter tuning over the logistic regression classifier.
- Tweak and explore a bit more the Deep Learning approach.
- Experiment with automatic machine learning (example: AutoKeras).

VI. References

- [1] - Kaggle - (https://www.kaggle.com/c/santander-customer-transaction-prediction)
- [2] - Wikipedia Matthews correlation coefficient - (https://en.wikipedia.org/wiki/Matthews_correlation_coefficient)
- [3] - Steven S. Skiena - The Data Science Design Manual, Springer, 2014
- [4] - Databricks MLFlow - https://mlflow.org/, 2019
- [5] - Kubeflow - https://www.kubeflow.org/, 2019
- [6] - Zhang. E. et al - Financial Forecasting and Analysis for Low-Wage Workers - arXiv:1806.05362v3 , 2018
- [7] - Zheng. X et al - FinBrain: When Finance Meets AI 2.0* -Front Inform Technol Electron Eng , 2010 https://arxiv.org/pdf/1808.08497.pdf ,
- [8] - Brownlee J - https://machinelearningmastery.com/tactics-to-combat-imbalanced-classes-in-your-machine-learning-dataset/ - 2019
- [9] - https://www.kaggle.com/jeffd23/10-classifier-showdown-in-scikit-learn , 2018
- [10] Wikipedia Naive Bayes classifier - https://en.wikipedia.org/wiki/Naive_Bayes_classifier
- [11] Wikipedia Ensemble Learning - https://en.wikipedia.org/wiki/Ensemble_learning
- [12] Wikipedia Random Forest - https://en.wikipedia.org/wiki/Random_forest

- [13] Catboost - https://catboost.ai/
- [14] XGBoost - https://en.wikipedia.org/wiki/XGBoost
- [15] https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html
- [16] Imbalanced learn package documentation - https://imbalanced-learn.readthedocs.io