Udacity Machine Learning Engineer Nanodegree: Capstone Project

Building a Data-Driven Solution to Mitigate Fraud

Author: Matt Oehler

Project Overview

Credit card fraud is a segment of identity theft and it negatively effects hundreds of thousands of people each year. One aspect of fraud detection that makes it difficult to build out effective data-based solutions is the severe class imbalance. While fraud negatively affects to many people, fraudulent transactions are only a drop in the ocean when looking at the vast sea of credit card transactions that occur on a daily, weekly, and yearly level. The focus of this project is to show several viable solutions to the issue of class imbalance in data analysis. In particular, we will show how Adaptive Synthetic Over-sampling (ADASYN) and Synthetic Minority Over-sample (SMOTE) compare to the more archaic method of Random Over-sampling. We will cover the main differences between these strategies, and then run an experiment to see how these strategies perform in a case study.

The data used for the case study in this project is a credit card fraud dataset, which was obtained from https://data-flair.training/, can be downloaded here. The dataset consists of over 280,000 transactions each with 30 features and a class label of fraud or not fraud. The class imbalance is very prevalent in this dataset as only 0.17% of the data points are labeled as fraud.

The problem that this project is designed to solve is that of fraud detection. The dataset we will be using is severely imbalanced with only 0.17% of the values being fraudulent. This severely hinders many of the most useful models in being able to detect a clear fraud signal and cleanly partition transactions as fraudulent or non-fraudulent. The ideal solution involves building a model that captures a vast majority of the fraud without blocking any of the non-fraudulent transactions (as this would lead to a very bad experience for legitimate customers). This is why precision and recall will be some of the primary metrics used to evaluate the models' performance. Potential solutions to this problem will explored using Logistic Regression and Tree Classifiers as models with the optimum solutions being determined from exploring different sampling strategies and parameter combinations. Details are expounded upon in following sections.

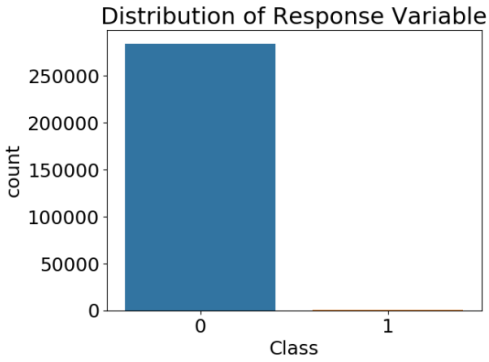
Analysis

Data Prep

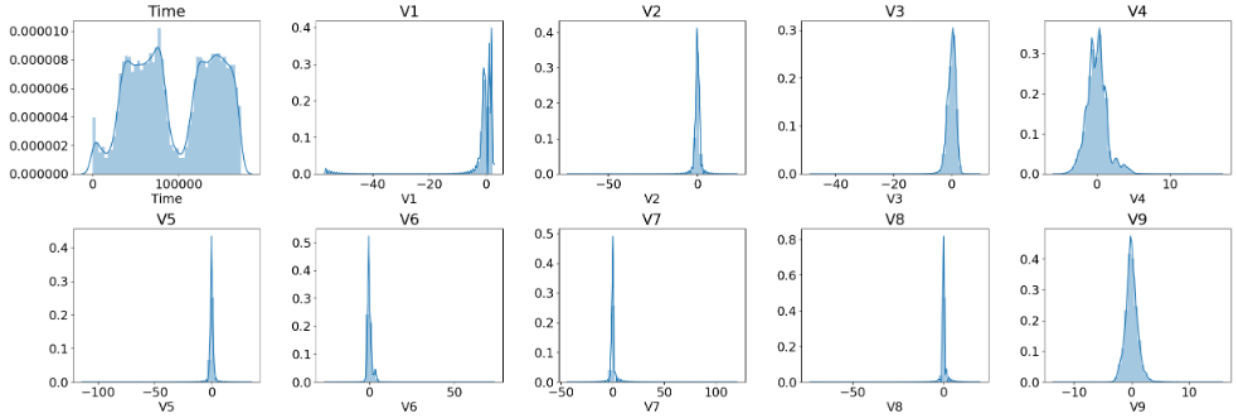
As mentioned above, the dataset used in this project contains approximately 280,000 transactions, each of which has 30 features and a class label (fraud or not fraud). Given that credit card transaction data includes personal identifiable information, the data provider anonymized the data by performing principal component analysis (PCA) on many of the features and removing most of the column names. All of the features are numeric in nature except for the class label which is simply a binary categorical feature. The feature list is specifically: Time, Amount, and "V1" - "V28". Other details such as how Time/Amount were recorded or standardized are unavailable. If any observations were missing or NULL, the data provider also handled those, but did not provide an explanation as to how. In summation, the data is quite clean from the start which will facilitate the process of exploration and model preparation.

Exploration

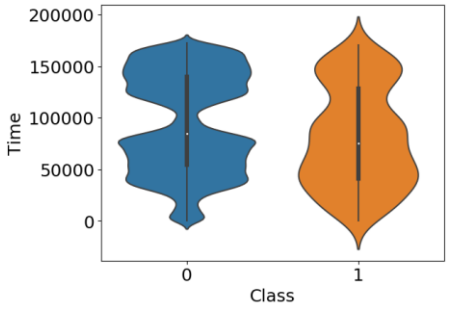
The first thing we did in exploring the data is understand the class imbalance. The 0.17% referred to earlier is composed of 492 labeled fraud instances out of exactly 284,807 total observed transactions. That isn't a lot to work with, as can be seen in the picture below. Handling this issue will be the primary focus of the Methodology section.



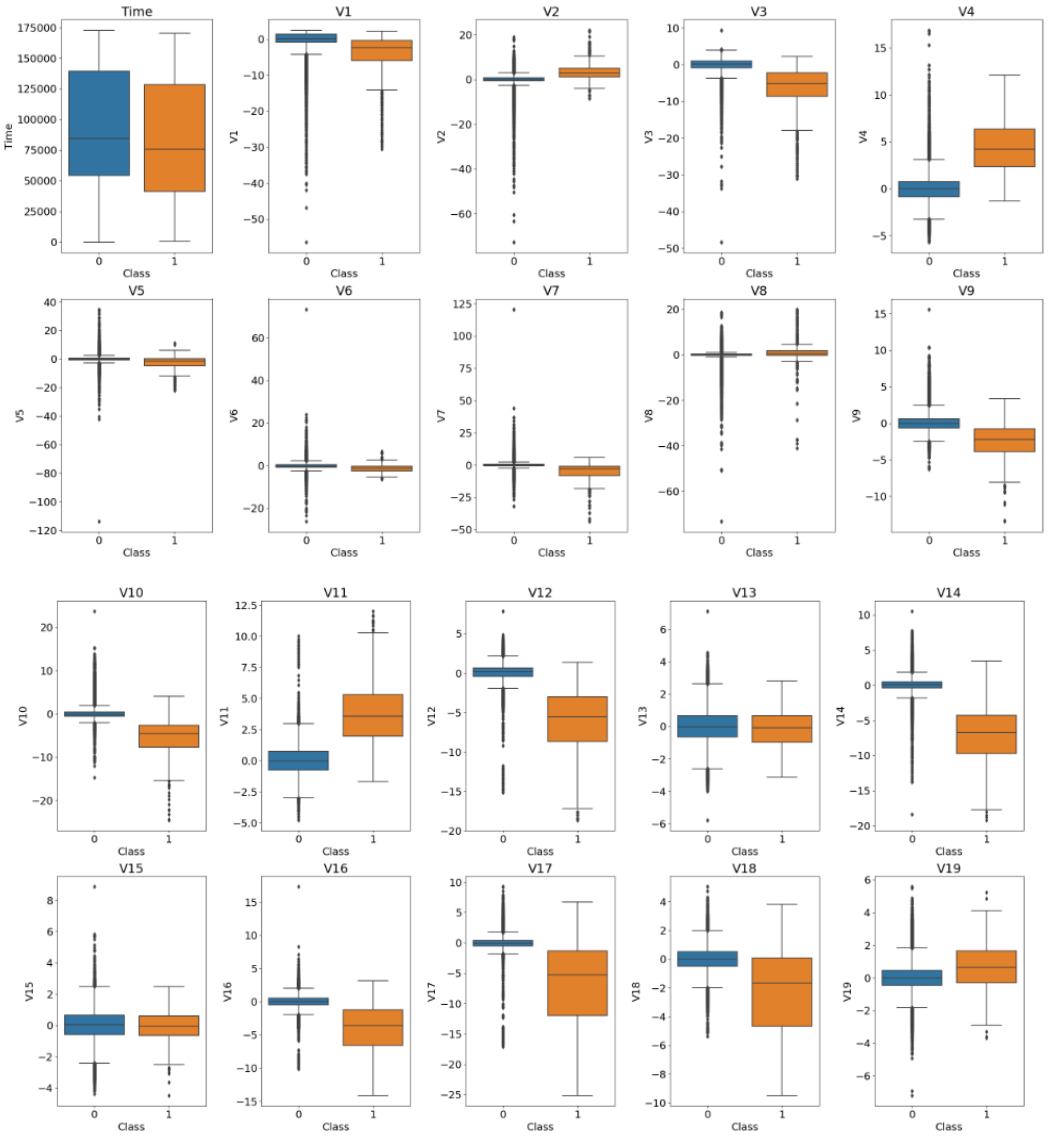
Next, I explored the distributions of each of the features. A sample of these plots is included below (the complete display of plots can be found in the development notebook associated with this project).

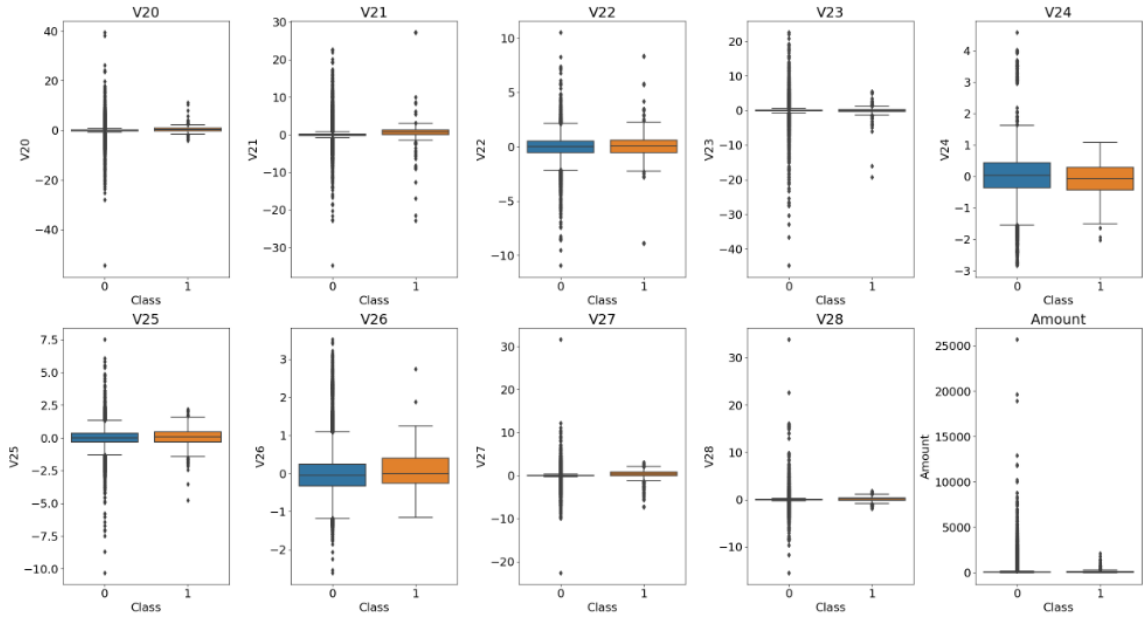


As we can see, there are some features with very skewed distributions, and some features with very low variance relative to other features. I am particularly intrigued by the bimodal distribution of the time variable. However, this makes sense given that customer activity is heaviest in the mornings/evenings when people aren't at work, and customer activity is probably very light during the late-night hours. Below I used a violin plot to compare the distribution of the Time feature between fraud and non-fraud populations. We can see that the distributions are different and that the density of fraud seems to be higher in the deepest valley of the non-fraud population. Perhaps fraudsters are more active at night while their ID theft victims are asleep?

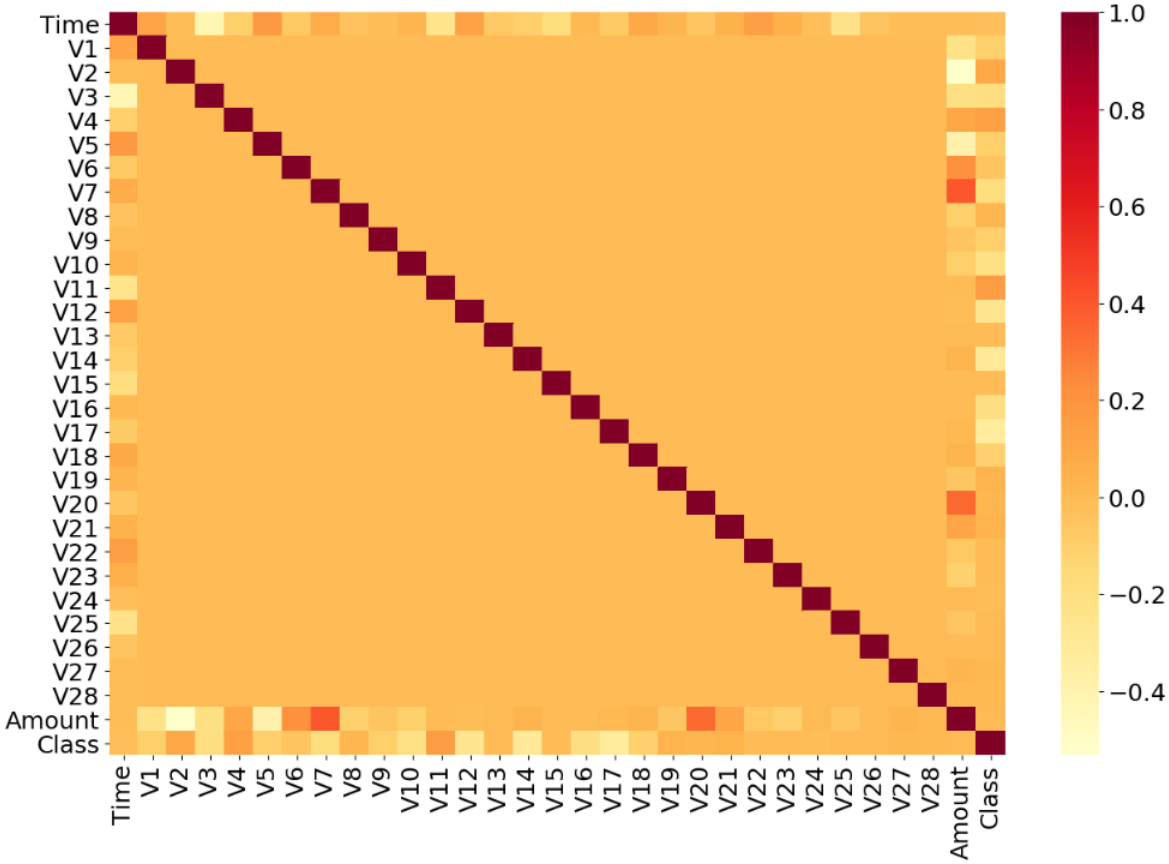


Additionally, I wanted to get a better understanding of how each of the features related with the Class label (fraud or not fraud). To do this, I made box plots for each feature since box plots are a create way to compare continuous/categorical features. I chose to include all of these plots as they provide helpful context understanding which features will be most relevant to the model. Based on the Box plots above, we can see that several features seem to have a significant relationship with the response variable. Check out the boxplots for V11, V12, V14, V17, and V18, just to name a few.





Finally, I made a heat map of the feature correlations. These maps help to identify collinearity, and can determine if any of the features should be excluded from the modeling process. Given that the data had been preprocessed using PCA, it makes sense that we don't have any features that are strongly correlated with each other. I conclude that we are good to move forward and include all features in the modeling phase of the experiment.



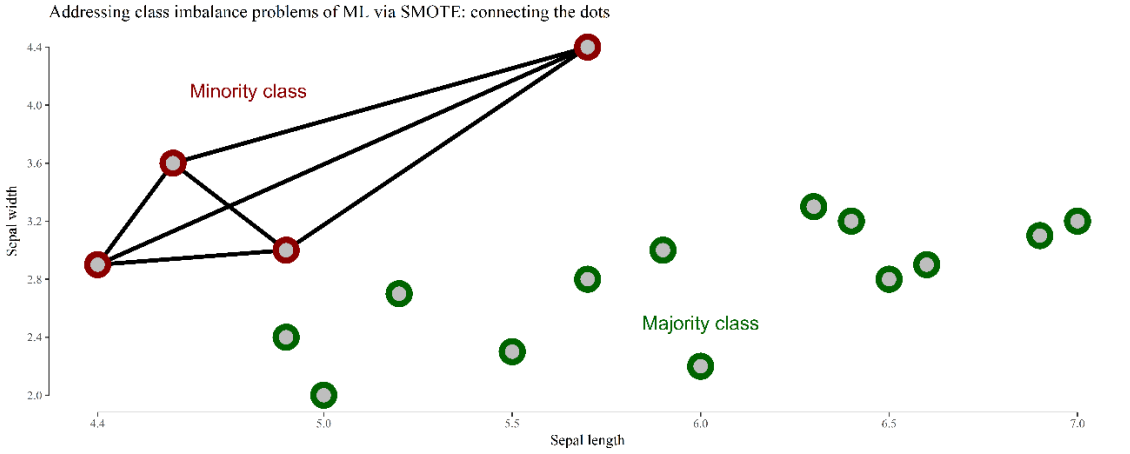
Methodology

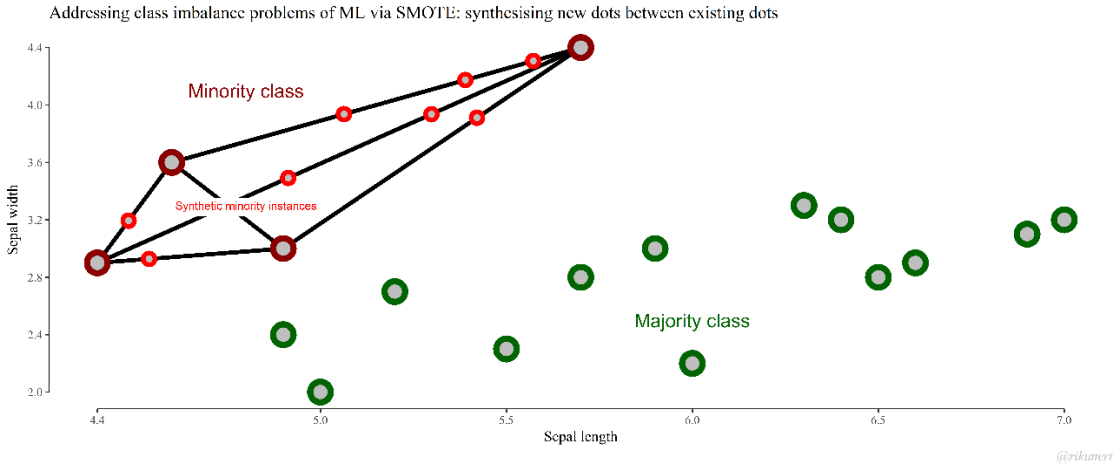
Strategies

Random Over-sampling (ROS), Synthetic Minority Oversampling Technique (SMOTE), and Adaptive Synthetic Over-sampling (ADASYN) are the three data-balancing strategies that are examined in this project. First, I will provide a brief definition of each strategy.

ROS is the simplest of the three methodologies. ROS takes the observations of the minority class and samples from them with replacement until the classes are balanced. The advantageous in that it can help models to pick up on the behavior signals of the minority class. However, since many observations are being duplicated, it can also cause models to overfit to certain behaviors or features of the minority class which is undesirable. Both ADASYN and SMOTE address the issue of duplicating observations.

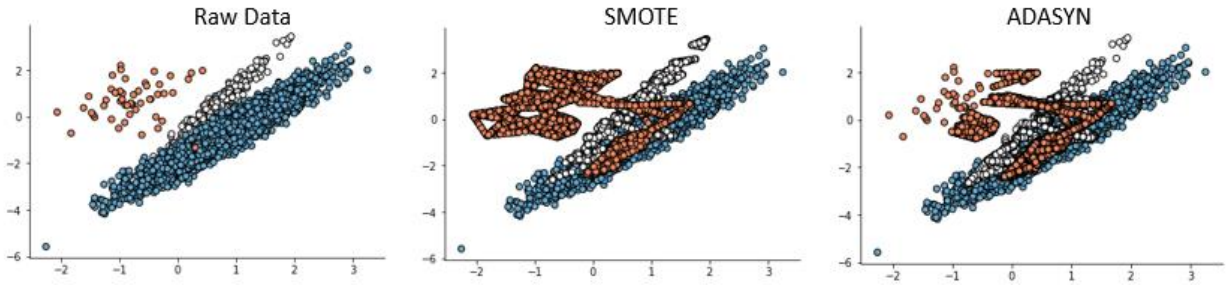
SMOTE, like ROS, is an over-sampling strategy. However, instead of duplicating observations of the minority class, it intelligently creates new or 'synthetic' data. The synthetic data are generated by using K-Nearest Neighbors on the feature vectors of the minority class. It computes the difference between the vectors, and then adds a random proportion of that difference to the original features to create synthetic data that falls within the feature space of the observed minority class observations. The pictures below (of SMOTE being applied to the classic Iris dataset) is helpful in understanding this process.





ADASYN is very similar to the SMOTE strategy in that it also incorporates the K-Nearest Neighbor algorithm in the process of generating synthetic data. However, the main difference between ADASYN and SMOTE, is that ADASYN will emphasize generating synthetic data in neighborhoods that are more dense with *majority* class observations. The idea behind this is that the neighborhoods with a higher concentration of majority class make it more difficult to classify the minority class successfully. So building more synthetic data around the harder-to-classify observations you can help to bolster the signal of those observations which should help the model to make a better decision boundary.

The figure below displays both the SMOTE and ADASYN over-sampling results on synthetically generated data. The orange class was the minority class that was being over-sampled. In the charts, it is quite easy to see that ADASYN focuses on the harder to classify points whereas SMOTE generates data uniformly across the minority class feature space.



Experiment

In this project I establish a few benchmarks to use as reference points. The primary benchmark is a logistic regression model trained on data that has been balanced with random over-sampling. Both the model and sampling strategy are less cutting edge than the methods I will be experimenting with. As such, this model and sampling strategy will provide a great primary baseline. I also, for the sake of reference, created a logistic regression model that is trained on the imbalanced data. These references will help to show the lift that data-balancing strategies bring to the table to bolster ML models in practice.

I explore a few different models to see if we can improve the results from our bench mark. After the results are examined, I will pick the top performing model and create a web app that allows users to send data to it to get predictions in real time.

I experiment with 3 different kinds of models: Logistic Regression (Baseline), Extra Trees Classifier, and Random Forest Classifier. For each model I examined the performance with multiple data balancing strategies: No Strategy (baseline), Random Over Sampling (baseline 2), ADASYN, and SMOTE.

Accuracy is not a very useful metric when dealing with imbalanced data. The primary metrics I will use to evaluate the models in this experiment are metrics that can be derived from a model's confusion matrix. These metrics primarily include precision and recall. Given that we are dealing with fraud detection, I will place more weight on precision, as we'd like to minimize the number of false positives that the model produces as they result in a negative experience for the credit card user (e.g. declining legitimate transactions).

Results

As described in the methodology section, I fit 3 different model types with 4 different strategies, resulting in 12 different combinations. The results can be seen in the table shown below.

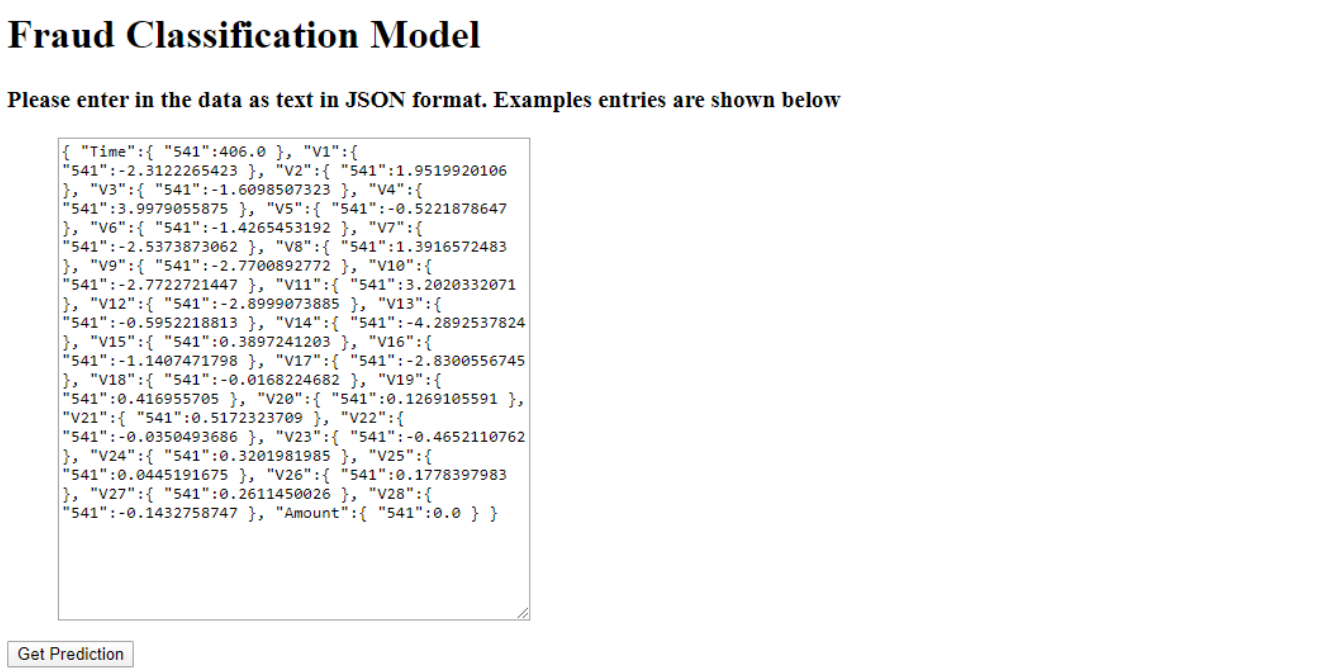
Model/Strategy	Precision	Recall		Predicted Not Fraud	Predicted Fraud
Logistic Regression - Imbalanced	0.734	0.6832	Not-Fraud	56836	25
			Fraud	32	69
Logistic Regression - ROS	0.0417	0.9109	Not-Fraud	54745	2116
			Fraud	9	92
Logistic Regression - SMOTE	0.0709	0.8911	Not-Fraud	55682	1179
			Fraud	11	90
Logistic Regression - ADASYN	0.0822	0.8911	Not-Fraud	55856	1005
			Fraud	11	90
Model/Strategy	**Precision**	**Recall**		**Predicted Not Fraud**	**Predicted Fraud**
Extra Trees - Imbalanced	0.9512	0.7723	Not-Fraud	56857	4
			Fraud	23	78
Extra Trees- ROS	0.9524	0.7921	Not-Fraud	56857	4
			Fraud	21	80
Extra Trees- SMOTE	0.8776	0.8515	Not-Fraud	56849	12
			Fraud	15	86
Extra Trees- ADASYN	0.8866	0.8515	Not-Fraud	56850	11
			Fraud	15	86
Model/Strategy	**Precision**	**Recall**		**Predicted Not Fraud**	**Predicted Fraud**
Random Forest - Imbalanced	0.9186	0.7822	Not-Fraud	56854	7
			Fraud	22	79
Random Forest - ROS	0.9222	0.8218	Not-Fraud	56854	7
			Fraud	18	83
Random Forest - SMOTE	0.87	0.8614	Not-Fraud	56848	13
			Fraud	14	87
Random Forest - ADASYN	0.86	0.8515	Not-Fraud	56847	14
			Fraud	15	86

When examining the results, I found several things that I was not expecting. First, the logistic regression model did worse when using any sort of balancing strategy. It seemed to overfit to the fraud signals and end up making predictions with a very high false-positive rate. The tree-based models performed much better overall, and the data balancing strategies (particularly ADASYN and SMOTE) really boosted the results.

I chose found that the Random Forest trained on data balanced via SMOTE was the best performing. It didn't have the highest precision, or the highest recall, but it did have a good balance of both. It was able to capture 87 out of 101 fraud instances in the training set while only capturing 13 observations as false-positives. This is the best representation of our ideal solution of a model that captures most of the fraud without ruining the customer experience for good customers.

Explanation of Web App

After selecting the best model, I wanted to create an interface for users to be able to interact with the model. This deployment step is the final step in our machine learning project. Using the 'flask' package in python I was able to build out a simple web app. On the main page (shown below) the user is prompted to input customer data in the form of a JSON string. Examples are shown to use as reference.



Fraud Classification Model

Please enter in the data as text in JSON format. Examples entries are shown below


```
{ "Time":{ "541":406.0 }, "V1":{
"541":-2.3122265423 }, "V2":{ "541":1.9519920106
}, "V3":{ "541":-1.6098507323 }, "V4":{
"541":3.9979055875 }, "V5":{ "541":-0.5221878647
}, "V6":{ "541":-1.4265453192 }, "V7":{
"541":-2.5373873062 }, "V8":{ "541":1.3916572483
}, "V9":{ "541":-2.7700892772 }, "V10":{
"541":-2.7722721447 }, "V11":{ "541":3.2020332071
}, "V12":{ "541":-2.8999073885 }, "V13":{
"541":-0.5952218813 }, "V14":{ "541":-4.2892537824
}, "V15":{ "541":0.3897241203 }, "V16":{
"541":-1.1407471798 }, "V17":{ "541":-2.8300556745
}, "V18":{ "541":-0.0168224682 }, "V19":{
"541":0.416955705 }, "V20":{ "541":0.1269105591 },
"V21":{ "541":0.5172323709 }, "V22":{
"541":-0.0350493686 }, "V23":{ "541":-0.4652110762
}, "V24":{ "541":0.3201981985 }, "V25":{
"541":0.0445191675 }, "V26":{ "541":0.1778397983
}, "V27":{ "541":0.2611450026 }, "V28":{
"541":-0.1432758747 }, "Amount":{ "541":0.0 } }
```


Get Prediction

Good:

{ "Time":{ "0":0.0 }, "V1":{ "0":-1.3598071337 }, "V2":{ "0":-0.0727811733 }, "V3":{ "0":2.536346738 }, "V4":{ "0":1.3781552243 }, "V5":{ "0":-0.3383207699 }, "V6":{ "0":0.4623877778 }, "V7":{ "0":0.2395985541 }, "V8":{ "0":0.0986979013 }, "V9":{ "0":0.3637869696 }, "V10":{ "0":0.090794172 }, "V11":{ "0":-0.5515995333 }, "V12":{ "0":-0.6178008558 }, "V13":{ "0":-0.9913898472 }, "V14":{ "0":-0.3111693537 }, "V15":{ "0":1.4681769721 }, "V16":{ "0":-0.4704005253 }, "V17":{ "0":0.2079712419 }, "V18":{ "0":0.0257905802 }, "V19":{ "0":0.4039929603 }, "V20":{ "0":0.2514120982 }, "V21":{ "0":-0.0183067779 }, "V22":{ "0":0.2778375756 }, "V23":{ "0":-0.1104739102 }, "V24":{ "0":0.0669280749 }, "V25":{ "0":0.1285393583 }, "V26":{ "0":-0.1891148439 }, "V27":{ "0":0.1335583767 }, "V28":{ "0":-0.0210530535 }, "Amount":{ "0":149.62 } }

Fraud:

{ "Time":{ "541":406.0 }, "V1":{ "541":-2.3122265423 }, "V2":{ "541":1.9519920106 }, "V3":{ "541":-1.6098507323 }, "V4":{ "541":3.9979055875 }, "V5":{ "541":-0.5221878647 }, "V6":{ "541":-1.4265453192 }, "V7":{ "541":-2.5373873062 }, "V8":{ "541":1.3916572483 }, "V9":{ "541":-2.7700892772 }, "V10":{ "541":-2.7722721447 }, "V11":{ "541":3.2020332071 }, "V12":{ "541":-2.8999073885 }, "V13":{ "541":-0.5952218813 }, "V14":{ "541":-4.2892537824 }, "V15":{ "541":0.3897241203 }, "V16":{ "541":-1.1407471798 }, "V17":{ "541":-2.8300556745 }, "V18":{ "541":-0.0168224682 }, "V19":{ "541":0.416955705 }, "V20":{ "541":0.1269105591 }, "V21":{ "541":0.5172323709 }, "V22":{ "541":-0.0350493686 }, "V23":{ "541":-0.4652110762 }, "V24":{ "541":0.3201981985 }, "V25":{ "541":0.0445191675 }, "V26":{ "541":0.1778397983 }, "V27":{ "541":0.2611450026 }, "V28":{ "541":-0.1432758747 }, "Amount":{ "541":0.0 } }

After entering the JSON data and clicking "Get Prediction" the user is taken to a results page that displays the data they entered as well as the predicted result, and the probability that it belongs to the predicted class.

JSON Data:

{ "Time":{ "541":406.0 }, "V1":{ "541":-2.3122265423 }, "V2":{ "541":1.9519920106 }, "V3":{ "541":-1.6098507323 }, "V4":{ "541":3.9979055875 }, "V5":{ "541":-0.5221878647 }, "V6":{ "541":-1.4265453192 }, "V7":{ "541":-2.5373873062 }, "V8":{ "541":1.3916572483 }, "V9":{ "541":-2.7700892772 }, "V10":{ "541":-2.7722721447 }, "V11":{ "541":3.2020332071 }, "V12":{ "541":-2.8999073885 }, "V13":{ "541":-0.5952218813 }, "V14":{ "541":-4.2892537824 }, "V15":{ "541":0.3897241203 }, "V16":{ "541":-1.1407471798 }, "V17":{ "541":-2.8300556745 }, "V18":{ "541":-0.0168224682 }, "V19":{ "541":0.416955705 }, "V20":{ "541":0.1269105591 }, "V21":{ "541":0.5172323709 }, "V22":{ "541":-0.0350493686 }, "V23":{ "541":-0.4652110762 }, "V24":{ "541":0.3201981985 }, "V25":{ "541":0.0445191675 }, "V26":{ "541":0.1778397983 }, "V27":{ "541":0.2611450026 }, "V28":{ "541":-0.1432758747 }, "Amount":{ "541":0.0 } }

Prediction: Fraud!
Probability: 0.7717
Get another prediction

It's very simple, but it accomplishes the purpose of enabling users to interact with the model.

Conclusion

The main goal of this project was to determine if the issue of class imbalance could be overcome when building out a data-driven solution to automated fraud detection. We were successful at accomplishing this goal by building a model that effectively capture a majority of fraud instances. Additionally, we deployed the model in a web app to enable users to get predictions from the model in real-time.

Acknowledgements

- http:/scholar.google.com/scholar_url?url=https://sci2s.ugr.es/keel/pdf/algorithm/congreso/2008-He-ieee.pdf&hl=en&sa=X&scisig=AAGBfm1uJ1FIWcOyYTHBq1effUALxWAmHg&nossl=1&oi=scholarr
- https://www.datasciencecentral.com/profiles/blogs/handling-imbalanced-data-sets-in-supervised-learning-using-family
- https://arxiv.org/pdf/1106.1813
- http://rikunert.com/SMOTE_explained
- https://www.kaggle.com/residentmario/oversampling-with-smote-and-adasyn