BOSS AI Notre Dame Analytics Challenge

Jack Lambert

Preparing, Exploring, and Defining the Data

The given data was nicely structured and easy to manipulate. While using Python, my first step in preparation was breaking the datetime column out into components. These components included individual columns for year, month, day of the month, day of the week, hour, and minute. After the datetime was expanded into these columns I looked for any additional insight that could be gained from the total credit (sum_amount) and average credit per user (avg_amount_per_user). From these columns, I generated a feature representing the ratio of total credit to the average credit per user (sum_to_avg_ratio). While the value wasn't identical to the total number of users at a given time stamp, it was directly proportional to the number of users at a given time stamp.

Once the datetime column was broken up I began to explore the relationships between the given variables and different datetime measurements. As expected – given the time series format – there was variation in total credit and average credit per user by different months, days, and hours. Examples of these varying relationships are shown below in Figure 1.

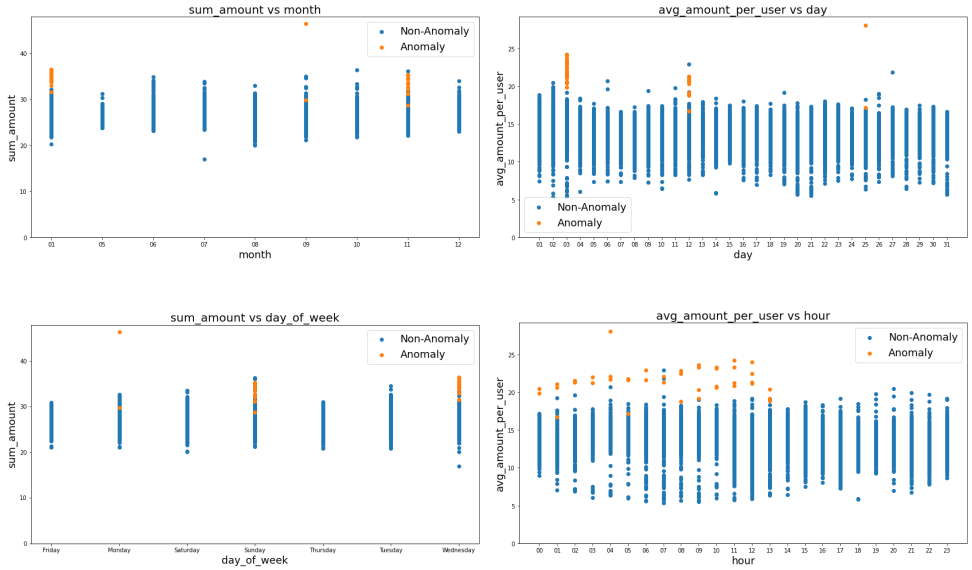

Figure 1. Example Relationships Displaying Time Series Variation of Total Credit and Average Credit per User

I then observed the two given variables over the entire timespan of the dataset. Doing so showed that both the total credit and average credit per user varied over time. These relationships are shown below in Figure 2.

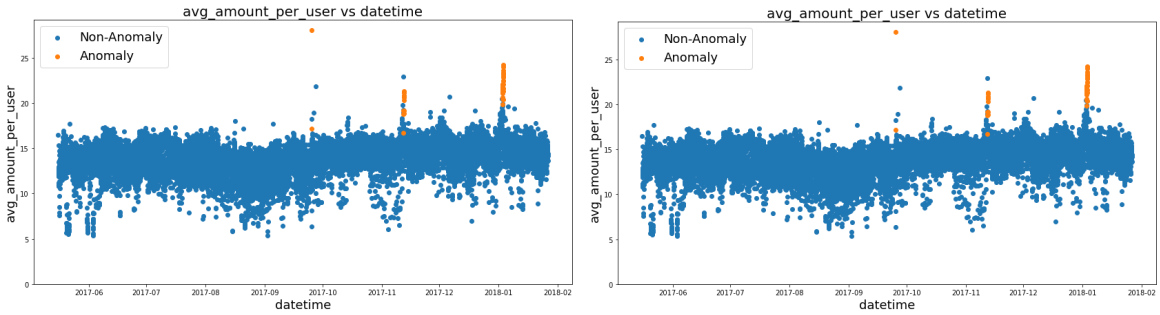

Figure 2. Total Credit and Average Credit per User Over Total Time

From this point, I then explored the relationships between the two given variables and the one I created. Graphing total credit and average credit per user relationship further emphasizes what can be seen in Figures 1 and 2: all anomalies occurred with high levels of total credit and average credit per user. Contextualizing these values with the ratio of the total credit to average credit per user adds further insight, showing a distinct relationship between the total number of users and the total credit. These relationships are shown below in Figure 3.

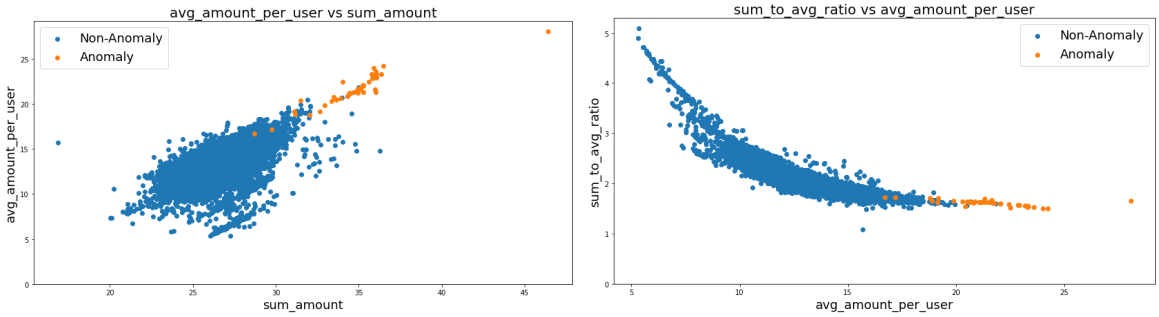

Figure 3. Relationships Between Given Features and Created Feature

While these trends are quite strong, there is still significant overlap between non-anomalies and anomalies. It would not be possible to simply draw a line between values to determine whether or not an anomaly has occurred.

This overlap between non-anomalies and anomalies shows the importance of the context in which the data point was collected. The first step in representing the features contextually was to take a rolling average for the values. After much experimentation, I determined that a rolling average of the last week of values (336 data points) for a given feature was optimal for success. In addition to a rolling average, a rolling standard deviation was also taken to account for differing variances in different timespans.

The next step in accounting for the seasonality of the different features was to take an average measurement for each data point based on the month, day of month, day of week, and hour averages for the different features to that point in time. A standard deviation was also taken to account for the differing variances.

Once these new features were created, an anomaly could now be predicted based on its expected values by month, day of the month, day of the week, hour and rolling average from the past week.

Building, Testing, and Evaluating the Model

Once all of the features were created, I split the data into training and testing sets. A particular problem in splitting the data was the significant imbalance in non-anomalies and anomalies. Anomalies only made up 41 of the 12,274 total data points, so a traditional train-test split of 0.7/0.3 or 0.8/0.2 would result in less than 15 anomaly cases being predicted. The results from such a model would not be robust enough to justify use of this model outside of this data. To address this issue, I created synthetic data points with SMOTE to train the model with. Once this synthetic data was created I utilized a traditional 0.7/0.3 training/testing split on the data.

I implemented an Extreme Gradient Boosting algorithm to predict these anomalies with the generated training and test splits. When generating a model I was quite wary of overfitting the model to the data, since all of the anomalies essentially occurred in three clusters as shown in Figures 1 and 2. To

account for the possibility of overfitting I compared the performance of a model on the training and testing set to make sure the results were quite similar. When valuing the performance of a model, I prioritized the F1 and recall the most. While accuracy and precision are valuable, working with ecommerce data makes Type ll error the highest importance; failing to identify an anomaly when one exists is costly due to allocation of resources for traffic that is fraudulent. The model that best balanced similarity of results in the training and testing set while maintaining a high F1 and recall was found by lowering the learning rate to 0.01 and subsample to 0.2. The results of this model are shown below in Figure 4.

Results	
Accuracy	0.9955
Precision	0.9834
Recall	0.9673
F1	0.9752

Confusion Matrix		
	No Anomaly	Anomaly
No Anomaly	3664	6
Anomaly	12	355

Figure 4. Optimum Model Performance

Implementation in BOSS AI and Next Steps

There was very little data cleaning to do in BOSS AI, as I simply imported the dataset I had created with the techniques explained above. The only transformation I implemented was dropping all rows with null values and encoding the target label to be a 1/0 instead of True/False.

I was able to recreate the scatterplots I created above much easier in the BOSS AI platform, and even explore some 3-dimensional scatter plots more in depth. I also found the histograms interesting to analyze the distributions of different features.

The process of implementing a model on the platform was incredibly simple. I very much preferred the interface with sliders to alter the hyperparameters to the traditional Python method. Additionally, the model generated by BOSS AI performed even better than the one I implemented in Python. These results are shown below in Figure 5.

Results	
Accuracy	0.9997
Precision	1.000
Recall	0.9997
F1	0.9998

Confusion Matrix		
	No Anomaly	Anomaly
No Anomaly	3631	0
Anomaly	1	14

Figure 5. BOSS AI Optimum Model Performance

While there were some slow loading times and glitchy animations, overall the functionality of BOSS AI was great. The platform was very easy to use after a very short amount of time familiarizing myself with it.

After my work understanding and transforming the data to create a model, I believe this model could enter production. Despite the high imbalance in anomalies, I believe the synthetic data points created a model that would handle new data quite well. Additionally, the features that predict the anomaly withstand any new seasonality by factoring into account previous averages by month, day of the month, day of the week, hour, and in the most recent week.