Credit Card Fraud Detection with XGB

Supervised Learning Capstone

Evan Bicher
10/31/18

Goal

- The goal is to create a model which can correctly predict if a transaction on a credit card was fraudulent or not.

For this model I will mainly be focusing on reducing Type II errors. These are fraudulent transactions that were not marked as fraud. From a business standpoint, a credit company would benefit more from catching fraudulent activity even if it comes at the cost of incorrectly flagging some normal transactions and forcing the customer to call the company and verify a purchase.

Flagging Fraud (Type II) > Incorrectly Flagging Valid Transaction (Type I)

https://github.com/ebicher/Data-Science/blob/master/SupervisedLearningCapstone

The Data

- Data is from: https://www.kaggle.com/mlg-ulb/creditcardfraud
- The dataset contains credit card transactions in September 2013 by European cardholders.
- Consists of 284,807 rows and 31 columns (30 features)
- Due to confidentiality issues, they couldn’t release original feature names or background information
- V(1-28) features were transformed using PCA

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 284807 entries, 0 to 284806
Data columns (total 31 columns):
Time      284807 non-null float64
V1        284807 non-null float64
V2        284807 non-null float64
V3        284807 non-null float64
V4        284807 non-null float64
V5        284807 non-null float64
V6        284807 non-null float64
V7        284807 non-null float64
V8        284807 non-null float64
V9        284807 non-null float64
V10       284807 non-null float64
V11       284807 non-null float64
V12       284807 non-null float64
V13       284807 non-null float64
V14       284807 non-null float64
V15       284807 non-null float64
V16       284807 non-null float64
V17       284807 non-null float64
V18       284807 non-null float64
V19       284807 non-null float64
V20       284807 non-null float64
V21       284807 non-null float64
V22       284807 non-null float64
V23       284807 non-null float64
V24       284807 non-null float64
V25       284807 non-null float64
V26       284807 non-null float64
V27       284807 non-null float64
V28       284807 non-null float64
Amount    284807 non-null float64
Class     284807 non-null int64
dtypes: float64(30), int64(1)
memory usage: 67.4 MB
```

Reduce Memory Usage

- All of the data was float64 datatype

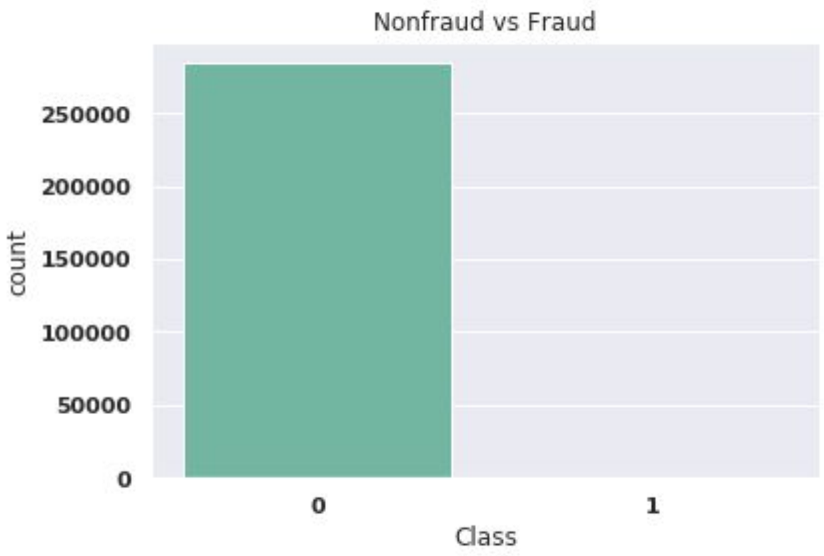
- Used a method to switch the datatype of the features so that it would take up less memory

- Allows for me to use my GPU for modeling which will lead to lower runtime for model training

```
Column:  V27
dtype before:  float64
dtype after:  float32
******************************
******************************
Column:  V28
dtype before:  float64
dtype after:  float32
******************************
******************************
Column:  Amount
dtype before:  float64
dtype after:  float32
******************************
******************************
Column:  Class
dtype before:  int64
dtype after:  uint8
******************************
___MEMORY USAGE AFTER COMPLETION:___
Memory usage is:  32.86526393890381  MB
This is  48.79038058214116 % of the initial size
```

Bias

- The data is overwhelmingly one-sided with most of the transactions being not fraud and a small sliver being actual fraud
- This needs to be corrected before building the model. There are two ways to do it.
- Undersampling
- Upsampling (Oversampling)
- Let’s try both!




```
Fraud Percentage: 0.1727%
```

```
0    284315
1       492
Name: Class,
```

Which model to use? RFC or XGB

Undersampling

	Type_I	Type_II	Type	Model
0	0.000000	0.005780	Train	RFC
1	0.010145	0.069364	Test	RFC
2	0.000000	0.005780	Train	XGB
3	0.001449	0.046243	Test	XGB

You can see that XBG is 33% better on the test for Type II errors

Upsampling

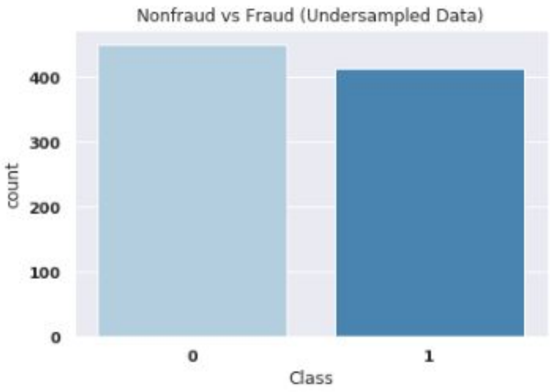
	Type_I	Type_II	Type	Model
0	0.000002	0.000000	Train	RFC
1	0.000000	0.105691	Test	RFC
2	0.002841	0.004065	Train	XGB
3	0.006218	0.060976	Test	XGB

You can see that XBG is 42% better on the test for Type II errors

Undersampling

Undersampling

- Took all the fraudulent transactions
- Took a random set of normal transactions that is equal to 110% amount of fraud transactions
- Made new smaller balanced dataset.
- Removed 170 data pieces for a dedicated test set
- 415 fraud & 450 normal in train/test set




```
#Get number of fraud cases and their indices
num_fraud = len(df[df.Class == 1])
fraud_index = np.array(df[df.Class==1].index)

#Get all normal indices
normal_index = df[df.Class==0].index

#Select number of normal indices that are equal to number of fraud cases
rand_norm_ind = np.random.choice(normal_index, int(num_fraud*1.1), replace = False)
rand_norm_ind = np.array(rand_norm_ind)

#Create equal dataframe
equal_ind = np.concatenate([fraud_index,rand_norm_ind])
un_df = df.iloc[equal_ind, :]

# Take some data for dedicated test set at the end
dedtest_index = un_df.index
ded_test = np.random.choice(dedtest_index, 170, replace = False)
ded_test = np.array(ded_test)
test_df = df.iloc[ded_test, :]
```

Base XGB Model

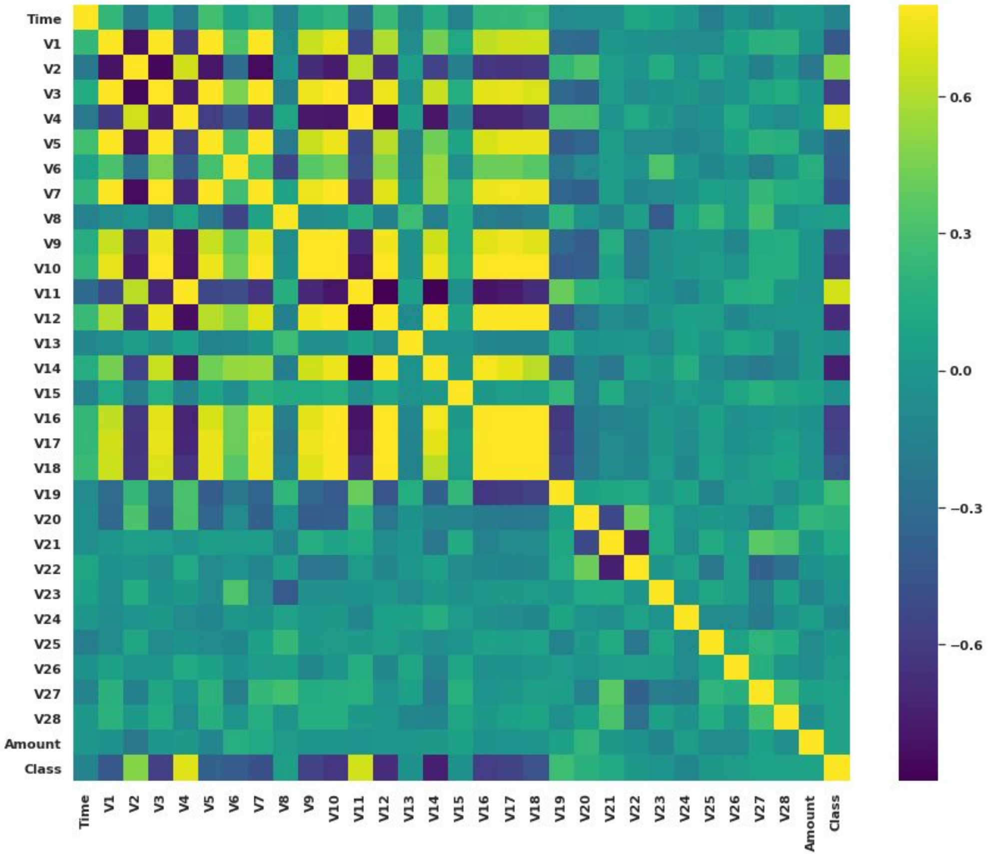
- Took out 20% of the balanced data as a validation set.

- Trained a model to see what we get in the beginning so we can see the improvement from feature selection and model tuning.

```
***********************************
********** Under Sampled ***********
***********************************

Training set accuracy:
Percent Type I errors:  0.00000000
Percent Type II errors: 0.00144928
Score: 0.99855072

Test set accuracy:
Percent Type I errors: 0.00578035
Percent Type II errors: 0.04624277
Score: 0.94797688

***********************************
* Type I errors are normal purchases that have been flagged.
* Type II errors are frauds that are not caught!
```

Correlation Matrix/ Feature Selection

- Removed any features with over 95% correlation with another feature
- V17 was removed



Model with V17 removed

```
***********************************
********** Under Sampled ***********
***********************************

Training set accuracy:
Percent Type I errors:  0.00000000
Percent Type II errors: 0.00144928
Score: 0.99855072

Test set accuracy:
Percent Type I errors: 0.00578035
Percent Type II errors: 0.04624277
Score: 0.94797688

***********************************
* Type I errors are normal purchases that have been flagged.
* Type II errors are frauds that are not caught!
```

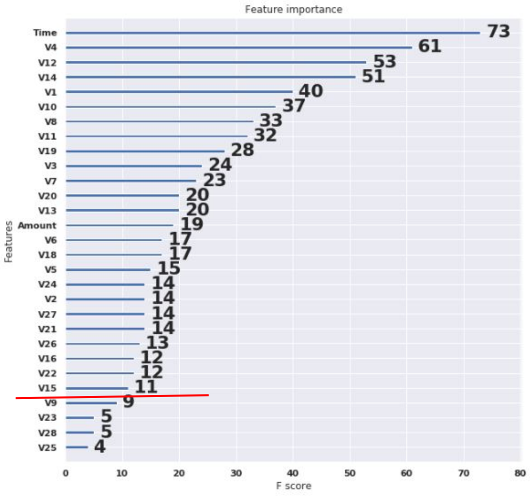
No Change

```
**********************************
*********** Exclude V17 ***********
**********************************

Training set accuracy:
Percent Type I errors:  0.00000000
Percent Type II errors: 0.00144928
Score: 0.99855072

Test set accuracy:
Percent Type I errors: 0.00578035
Percent Type II errors: 0.04624277
Score: 0.94797688

**********************************
* Type I errors are normal purchases that have been flagged.
* Type II errors are frauds that are not caught!
```

Feature Selection / Feature Importance

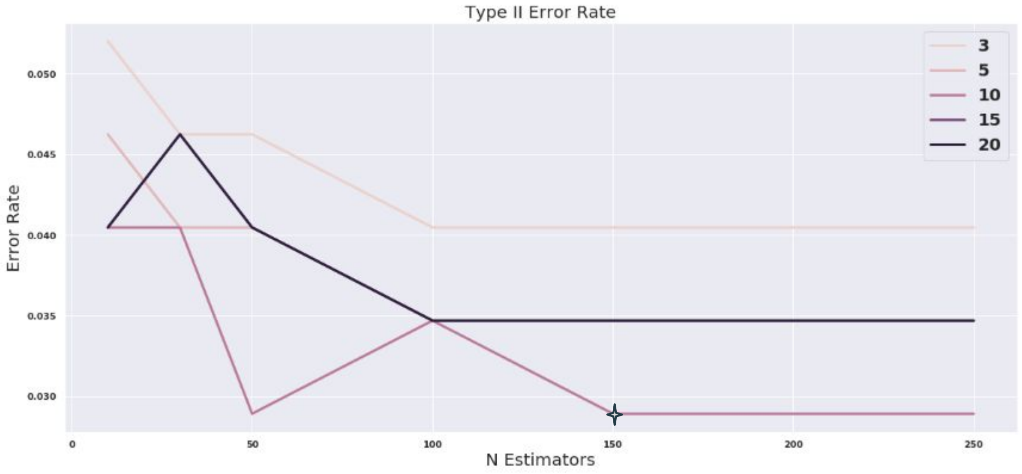
Looked at how the model viewed each feature in terms of importance. Then built a series of models by adding important features one by one.

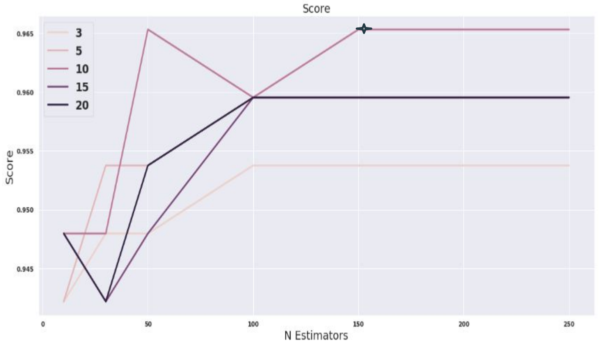
Removed: ['V25', 'V23', 'V28', 'V9', 'V17']

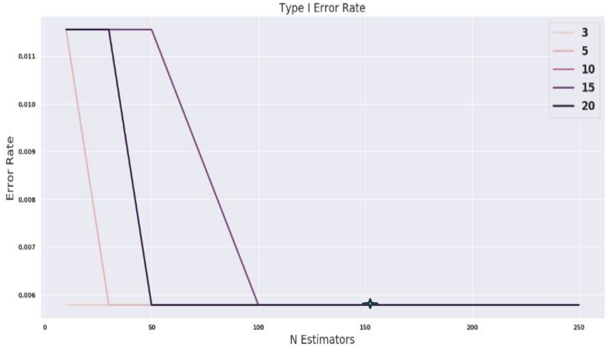


Parameter Tuning

Created models to decide optimal Max Depth and N Estimators parameters







Optimal parameters:

Max Depth: 10 N Estimators: 150

Undersample Results

```
**********************************
********** Under Sampled ***********
**********************************

Training set accuracy:
Percent Type I errors:  0.00000000
Percent Type II errors: 0.00144928
Score: 0.99855072
```

Validation

```
~~Test~~ set accuracy:
Percent Type I errors: 0.00578035
Percent Type II errors: 0.04624277
Score: 0.94797688

**********************************
* Type I errors are normal purchases that have been flagged.
* Type II errors are frauds that are not caught!
```

```
**********************************

Train set accuracy:
Percent Type I errors:  0.000000000000000
Percent Type II errors: 0.000000000000000

Validation set accuracy:
Percent Type I errors: 0.005780346820809
Percent Type II errors: 0.028901734104046

Full set accuracy:
Percent Type I errors:  0.035150119203531
Percent Type II errors: 0.000035111496557

Test set accuracy:
Percent Type I errors:  0.029411764705882
Percent Type II errors: 0.029411764705882

**********************************
* Type I errors are normal purchases that have been flagged.
* Type II errors are frauds that are not caught!
```

Undersample Error Tables

Dedicated Test

col_0	0	1	All
Class			
0	86	5	91
1	5	74	79
All	91	79	170

Full Data

col_0	0	1	All
Class			
0	274304	10011	284315
1	10	482	492
All	274314	10493	284807

Training Data

col_0	0	1	All
Class			
0	363	0	363
1	0	327	327
All	363	327	690

Validation Set

col_0	0	1	All
Class			
0	86	1	87
1	5	81	86
All	91	82	173

Upsampling

Bias

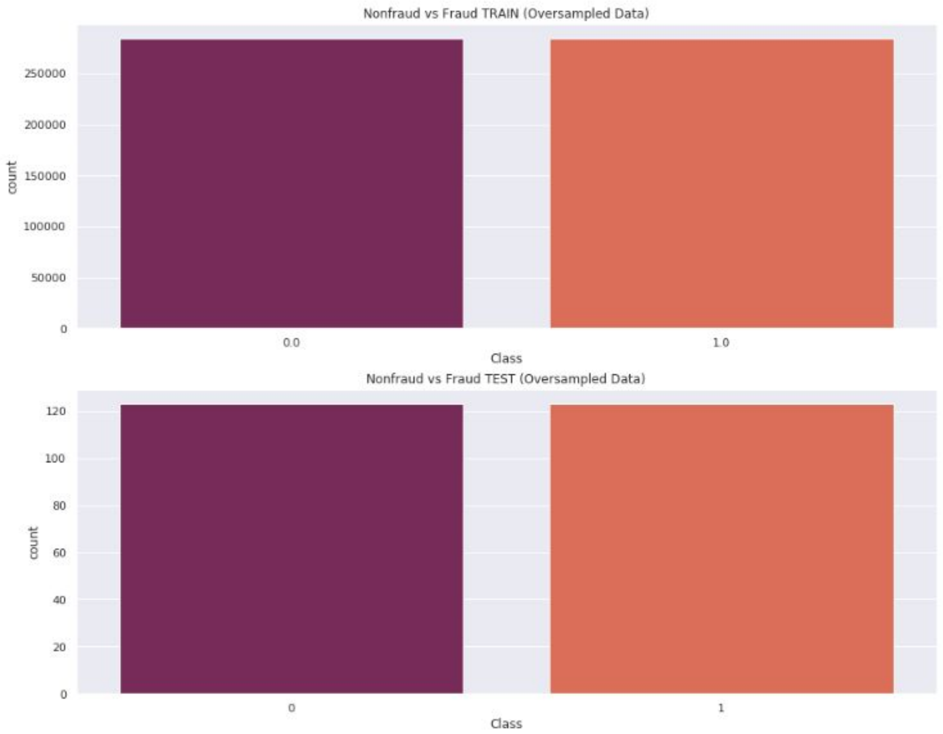
- The data is overwhelmingly one sided with most of the transactions being not fraud and a small sliver being actual fraud
- This needs to be corrected before building the model. There are two ways to do it.
- Undersampling
- Upsampling (Oversampling)
- Let’s try both!


```
Fraud Percentage: 0.1727%
```

```
0    284315
1       492
Name: Class,
```

Upsampling

- Took out a test set equal to 20% fraud + an equal amount of normal transactions
- Multiple ways to upsample
- Used SMOTE to create new “fraud” data
- 284192 Fraud / 284192 Normal



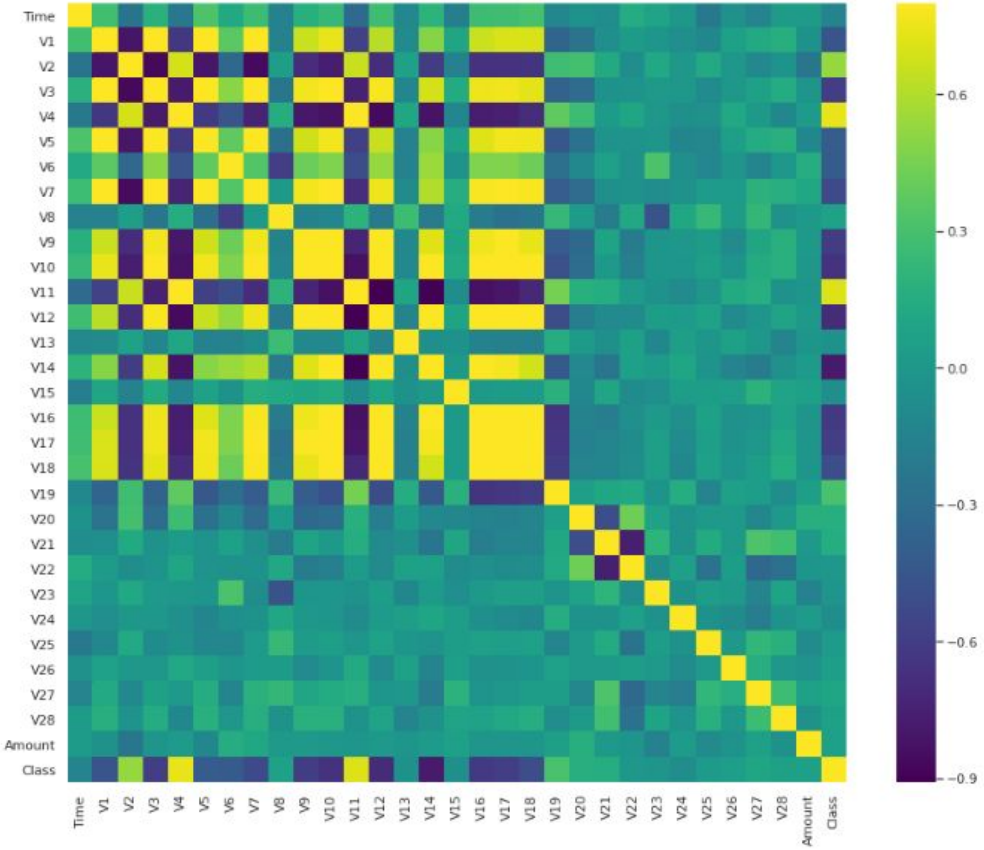
Base XGB Model

- Trained a model to see what we get in the beginning so we can see the improvement from feature selection and model tuning.

```
**********************************
********** Over Sampled ***********
**********************************

Training set accuracy:
Percent Type I errors:  0.00284139
Percent Type II errors: 0.00621763
Score: 0.99094098

Test set accuracy:
Percent Type I errors: 0.00406504
Percent Type II errors: 0.06097561
Score: 0.93495935

**********************************
* Type I errors are normal purchases that have been flagged.
* Type II errors are frauds that are not caught!
```

Correlation Matrix/ Feature Selection 1

- Removed any features with over 95% correlation with another feature
- V17 was removed



Model with V17 removed

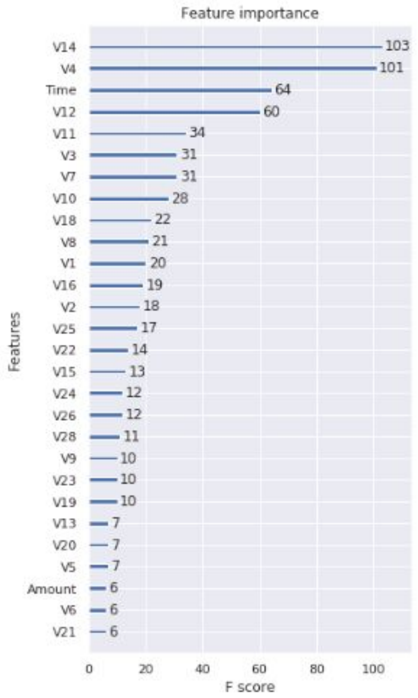
```
**********************************
********** Over Sampled ***********
**********************************

Training set accuracy:
Percent Type I errors:  0.00284139
Percent Type II errors: 0.00621763
Score: 0.99094098

Test set accuracy:
Percent Type I errors: 0.00406504
Percent Type II errors: 0.06097561
Score: 0.93495935

**********************************
* Type I errors are normal purchases that have been flagged.
* Type II errors are frauds that are not caught!
```

Improvement

```
**********************************
********** V17 Excluded ***********
**********************************

Training set accuracy:
Percent Type I errors:  0.00337272
Percent Type II errors: 0.00601530
Score: 0.99061198

Test set accuracy:
Percent Type I errors: 0.00813008
Percent Type II errors: 0.05284553
Score: 0.93902439

**********************************
* Type I errors are normal purchases that have been flagged.
* Type II errors are frauds that are not caught!
```

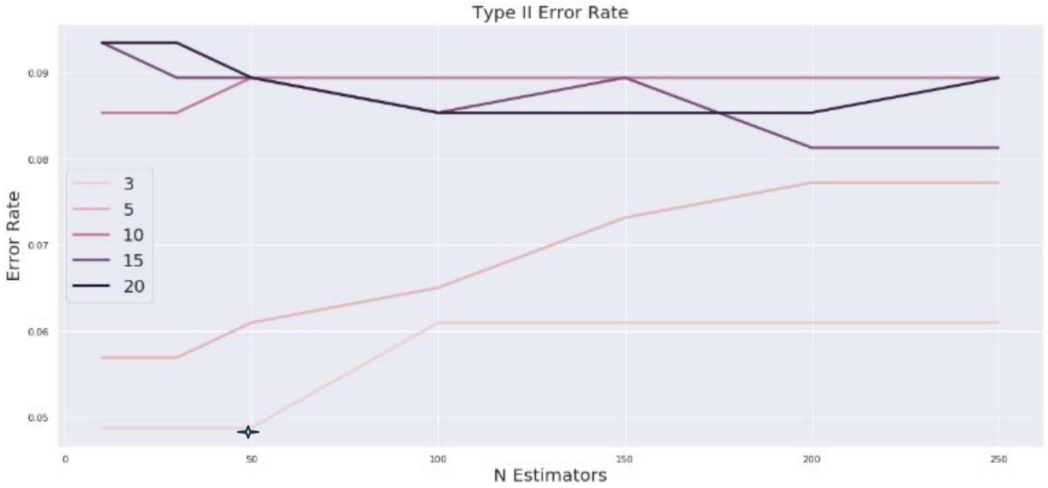
Feature Selection / Feature Importance

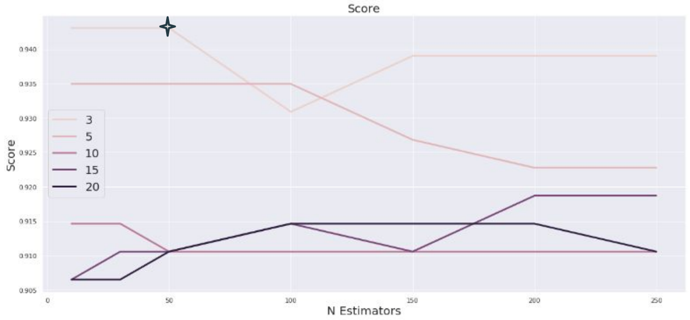
Looked at how the model viewed each feature in terms of importance. Then built a series of models by adding important features one by one.

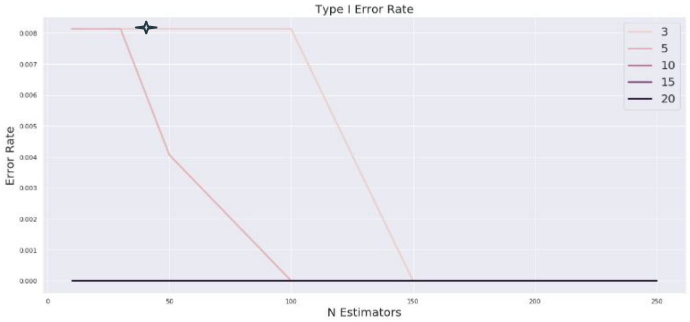


Parameter Tuning

Created models to decide optimal Max Depth and N Estimators parameters







Optimal parameters:

Max Depth: 3 N Estimators: 50

Upsample Results

```
**********************************
********** Over Sampled **********
**********************************

Training set accuracy:
Percent Type I errors:  0.00284139
Percent Type II errors: 0.00621763
Score: 0.99094098

Test set accuracy:
Percent Type I errors: 0.00406504
Percent Type II errors: 0.06097561
Score: 0.93495935

**********************************
* Type I errors are normal purchases that have been flagged.
* Type II errors are frauds that are not caught!
```

```
**********************************

Train set accuracy:
Percent Type I errors:  0.006036412003153
Percent Type II errors: 0.015806215516271

Test set accuracy:
Percent Type I errors:  0.008130081300813
Percent Type II errors: 0.048780487804878

Full set accuracy:
Percent Type I errors:  0.012053776768127
Percent Type II errors: 0.000154490584852

**********************************
* Type I errors are normal purchases that have been flagged.
* Type II errors are frauds that are not caught!
```

Upsample Error Tables

Full Data

col_0	0.0	1.0	All
Class			
0	280882	3433	284315
1	44	448	492
All	280926	3881	284807

Test Set

col_0	0.0	1.0	All
Class			
0	121	2	123
1	12	111	123
All	133	113	246

Training Data

col_0	0.0	1.0	All
Class			
0.0	280761	3431	284192
1.0	8984	275208	284192
All	289745	278639	568384

Comparing them

Undersample

Better at finding Type II errors

```
**********************************

Train set accuracy:
Percent Type I errors:  0.0000000000000000
Percent Type II errors: 0.0000000000000000

Validation set accuracy:
Percent Type I errors: 0.005780346820809
Percent Type II errors: 0.028901734104046

Full set accuracy:
Percent Type I errors:  0.035150119203531
Percent Type II errors: 0.000035111496557

Test set accuracy:
Percent Type I errors:  0.029411764705882
Percent Type II errors: 0.029411764705882

**********************************
* Type I errors are normal purchases that have been flagged.
* Type II errors are frauds that are not caught!
```

Upsample

Better at finding Type I errors

```
**********************************

Train set accuracy:
Percent Type I errors:  0.006036412003153
Percent Type II errors: 0.015806215516271

Test set accuracy:
Percent Type I errors:  0.008130081300813
Percent Type II errors: 0.048780487804878

Full set accuracy:
Percent Type I errors:  0.012053776768127
Percent Type II errors: 0.000154490584852

**********************************
* Type I errors are normal purchases that have been flagged.
* Type II errors are frauds that are not caught!
```

Improvements

- It would be amazing to have more data to work with.
- Could have a validation set for the upsampling model
- Could use these models to predict the probability of each case, then create a feature for each row that was the probability and then create a new model that would incorporate both of their benefits
- Try a different technique of feature elimination, for instance, not removing them by their importance but by another factor.

Fin
(Questions/Comments)