Michael Goldman
November 28, 2019

Machine Learning Engineering – Capstone Project: Detecting Instances of Credit Card Fraud

Definition:

It always starts what seems to be a normal day. You try to pay something with you credit card and it is declined. Huh, that is weird, but you don't think too much about it. You check your bank later today just to see what's up. You are locked out. Panic starts to ensue. You call your bank see what is going on. The person on the phone asks if you made a purchase for 300 Xboxes to be sent to Indonesia. You tell them no and are able to get account unlocked and the fraudulent order canceled.

In 2018 alone, more than 24.2 billion dollars were lost worldwide due to fraudulent activity and the US is the most prone to it, with 38.6% of cases residing from there.[1] The problem is only continuing to grow, so how can we stop it, or at least detect it? With this Capstone project, I am going to be working with a dataset about fraudulent activity. The dataset comes from Kaggle and is a dataset of fraudulent and non-fraudulent transactions that occurred by European Cardholders from two days in September 2013.[2] Our goal is to explore the dataset, create a few models, and see if we can predict fraudulent transactions. We will do this by creating a number of different model types, feeding in different input data (the original data, an under sampled version, and an oversampled version), and hyper tuning the parameters to see what gives us the best results. We will use a number of different models such as a logistic regression (our benchmark), SDGClassifier, Random Forest, XGBoost, a simple Neural Network, and a deeper Neural Network. We will discuss the formulas, the pros, and cons later in this report.

We will look at a number of different metrics to make sure our model is accurate, especially AUPRC. I created a scoring function that will give us a number of different metrics. It includes precision, recall, f1 score, confusion matrices, and AUPRC curves.

Precision:

$$\text{Precision} = \frac{True\ Positive}{True\ Positive + False\ Positive}$$

[3]

Precision tells us information about how precise our model is. It is used to see out of all the positive results the model predicted, how many were truly positive. This is useful for when the

[1] "Credit Card Fraud Statistics." *Shift Processing*, Shift Credit Card Processing, 2019, shiftprocessing.com/credit-card-fraud-statistics/.

[2] Machine Learning Group. "Credit Card Fraud Detection." *Kaggle*, Google LLC, 23 Mar. 2018, www.kaggle.com/mlg-ulb/creditcardfraud.

[3] Shung, Koo Ping. "Accuracy, Precision, Recall or F1?" *Medium*, Towards Data Science, 8 June 2018, towardsdatascience.com/accuracy-precision-recall-or-f1-331fb37c5cb9.

cost of false positives is high, such as spam filtering (we do not want legitimate messages to be sent to our spam folder). Although it is inconvenient for credit card transactions to be marked as fraudulent incorrectly, it is not as bad as not as having fraudulent charges go unnoticed.

Recall:

$$\text{Recall} = \frac{\text{True Positive}}{\text{True Positive}+\text{False Negative}}$$ [4]

Recall, or sensitivity, tells us information on how many positive predictions are relevant. This is used to see how complete the results are, which values the false negatives more highly. This is relevant in our project because the cost of false negatives is higher than the cost of false positives. In our model we want to be save rather than sorry. We will use this for some of our scoring metrics in our imbalanced dataset.

F1 score:

$$\text{F1} = 2 \times \frac{\text{Precision} * \text{Recall}}{\text{Precision}+\text{Recall}}$$ [5]

F1 score is a balance between recall and precision. This values type 1 and type 2 errors This can be used if false positives and false negatives both have business costs.

Confusion matrix:

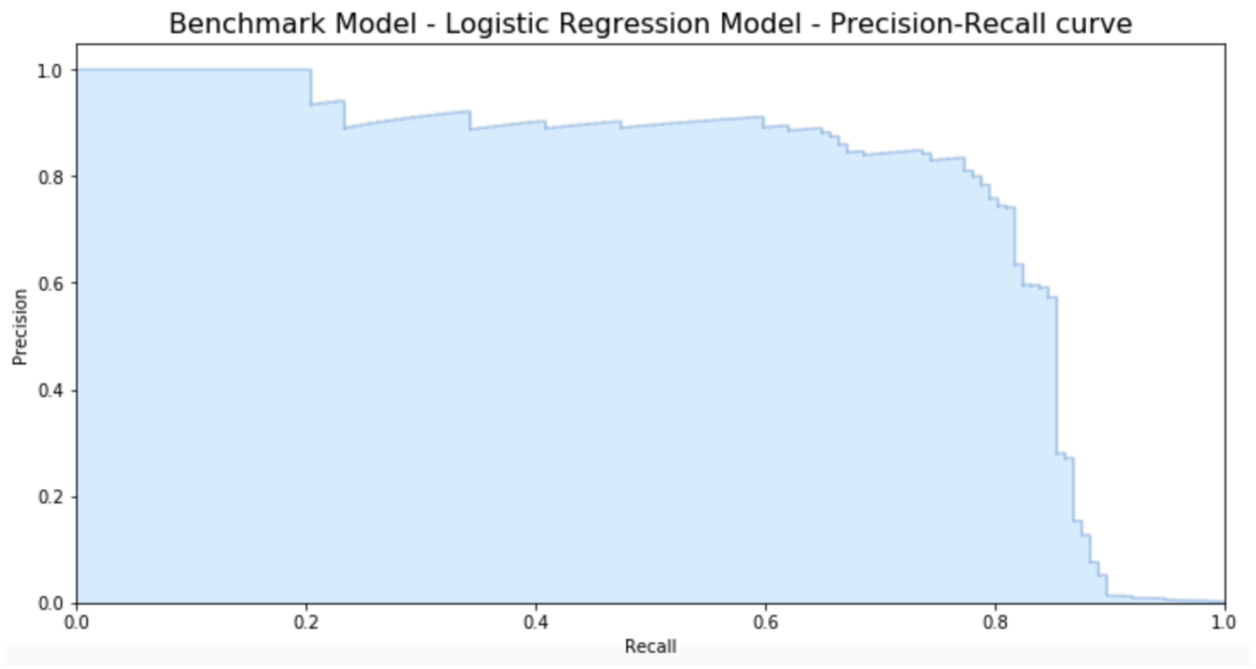
n=165	**Predicted: NO**	**Predicted: YES**
Actual: NO	50	10
Actual: YES	5	100

[6]

Confusion matrix is a good visualization to use to see how accurate our model is. It shows us the outcome of our model in comparison to what the actual results are. You can see the number of true positives, true negatives, false positives, and false negatives. This can also be used to calculate recall and precision. When class sizes are very imbalanced, like the one that we have here, it is less useful than the other metrics, but I include it for completeness.

[4] "Precision and Recall." *Wikipedia*, Wikimedia Foundation, 3 Dec. 2019, en.wikipedia.org/wiki/Precision_and_recall.
[5] "Precision and Recall." *Wikipedia*, Wikimedia Foundation, 3 Dec. 2019, en.wikipedia.org/wiki/Precision_and_recall.
[6] "Confusion Matrix." *Wikipedia*, Wikimedia Foundation, 22 Oct. 2019, en.wikipedia.org/wiki/Confusion_matrix.

AUPRC:



The main metric we will score our models. It is a visualization to show the tradeoff of our precision and recall. We will want to try and maximize the area under the curve. This can be captured by the 'average_precision' scoring method in our cross-validation function. We want our models to have high precision and recall, which is why we use this to train and score our models using this method.

We will validate our models using cross validation. Although we have information about a lot of different scoring metrics, we will default to average precision in for our ultimate decision.

Analysis:

When I first got the dataset, I explored to see what information was in it.
The dataset contains 284,807 transactions and 28 columns from the output of a PCA transformation. The dataset also contains time, which is the number of seconds that elapsed between each transaction, and amount, the transaction amount. Our target is class which indicates whether the transaction is fraudulent or not. Only .172% of transactions are fraudulent. Because the features have been anonymized through PCA, we are unable to get much information about what the numbers mean by themselves.

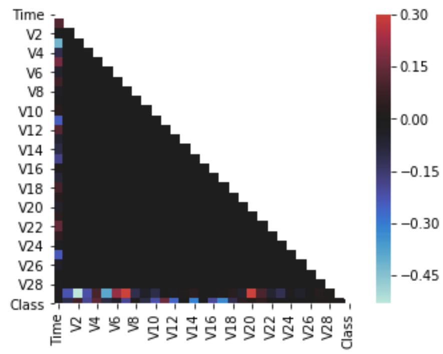
```
missing = (df.isnull().sum()*100/len(df)).to_frame(name = '% null').T
pd.concat([df.describe(), missing])
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9
count	284807.000000	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05
mean	94813.859575	1.165980e-15	3.416908e-16	-1.373150e-15	2.086869e-15	9.604066e-16	1.490107e-15	-5.556467e-16	1.177556e-16	-2.406455e-15
std	47488.145955	1.958696e+00	1.651309e+00	1.516255e+00	1.415869e+00	1.380247e+00	1.332271e+00	1.237094e+00	1.194353e+00	1.098632e+00
min	0.000000	-5.640751e+01	-7.271573e+01	-4.832559e+01	-5.683171e+00	-1.137433e+02	-2.616051e+01	-4.355724e+01	-7.321672e+01	-1.343407e+01
25%	54201.500000	-9.203734e-01	-5.985499e-01	-8.903648e-01	-8.486401e-01	-6.915971e-01	-7.682956e-01	-5.540759e-01	-2.086297e-01	-6.430976e-01
50%	84692.000000	1.810880e-02	6.548556e-02	1.798463e-01	-1.984653e-02	-5.433583e-02	-2.741871e-01	4.010308e-02	2.235804e-02	-5.142873e-02
75%	139320.500000	1.315642e+00	8.037239e-01	1.027196e+00	7.433413e-01	6.119264e-01	3.985649e-01	5.704361e-01	3.273459e-01	5.971390e-01
max	172792.000000	2.454930e+00	2.205773e+01	9.382558e+00	1.687534e+01	3.480167e+01	7.330163e+01	1.205895e+02	2.000721e+01	1.559499e+01
% null	0.000000	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00

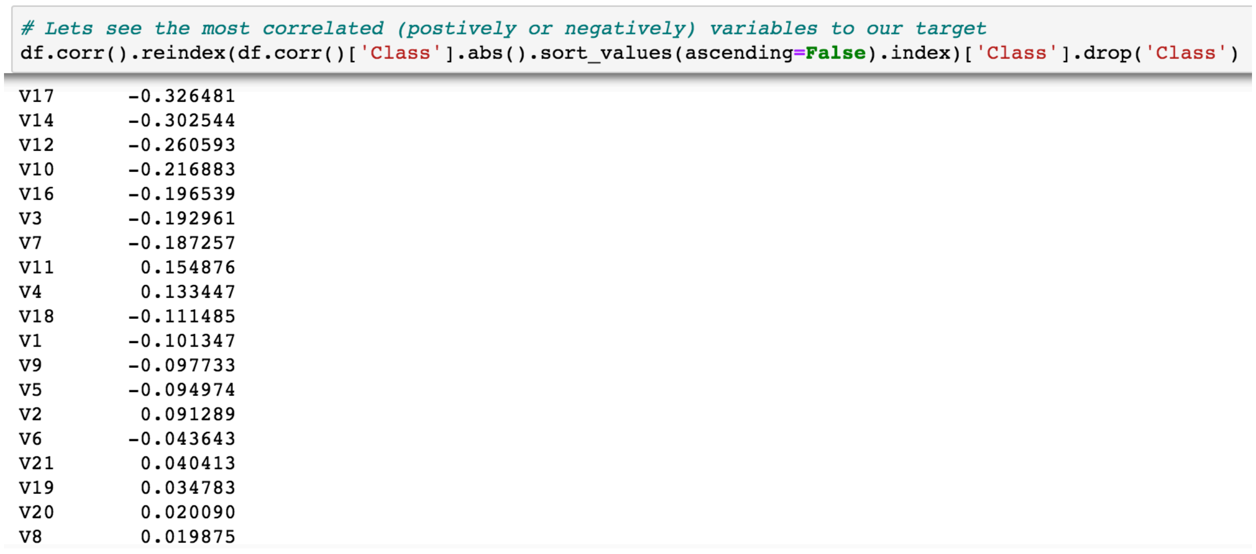
We can see that the data appears to be normalized except for Time and Amount. There are a few columns that have outliers and we will address them later. For now, we shall normalize the two columns so all the data is within similar ranges.

Although the data says it was outputs from a PCA, I wasn't sure if the V1 was the principal axis. I also wanted to see which variable had the highest (or lowest) correlation to our target.

```
corr = df.corr()
mask = np.zeros_like(corr)
mask[np.triu_indices_from(mask)] = True
ax = sns.heatmap(corr, mask=mask, center=0, vmax=.3, square=True)
```



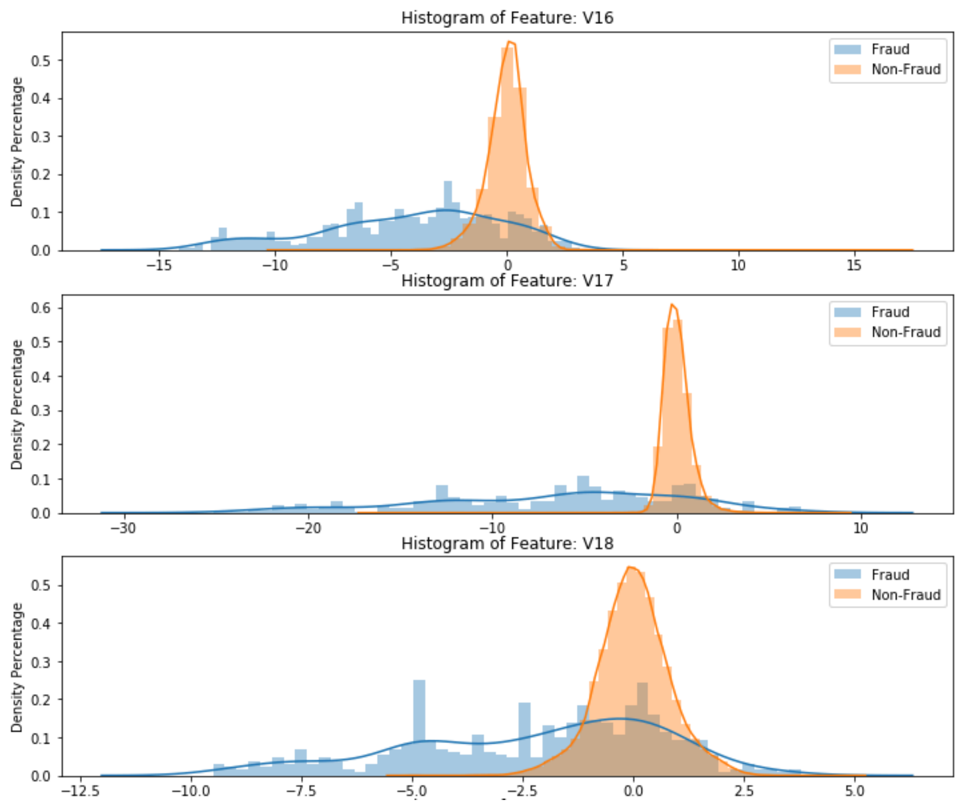
The correlation map also shows that the columns are not correlated with each other (which is known since they are outputs form PCA). This also means that we won't have issues with multicollinearity in our models. If there was any, we could have tried to combine features together to retain the information about the two columns but limit multicollinearity. Since we don't face this issue and we don't have information on the columns, we cannot do much feature engineering. That said, we should see what columns (if any) are correlated with our target variable.



```python
# Lets see the most correlated (postively or negatively) variables to our target
df.corr().reindex(df.corr()['Class'].abs().sort_values(ascending=False).index)['Class'].drop('Class')
```

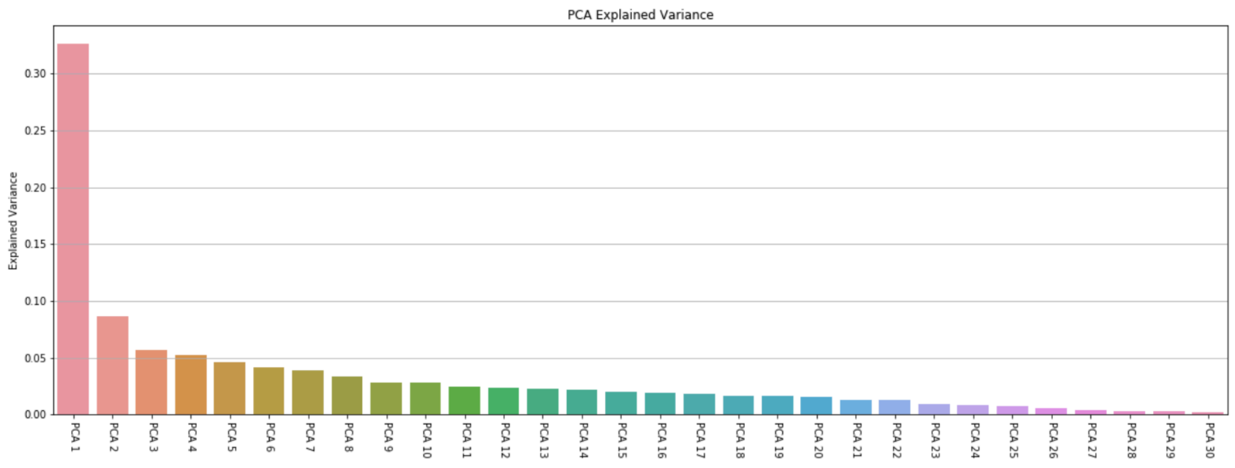
```
V17      -0.326481
V14      -0.302544
V12      -0.260593
V10      -0.216883
V16      -0.196539
V3       -0.192961
V7       -0.187257
V11       0.154876
V4        0.133447
V18      -0.111485
V1       -0.101347
V9       -0.097733
V5       -0.094974
V2        0.091289
V6       -0.043643
V21       0.040413
V19       0.034783
V20       0.020090
V8        0.019875
```

As you can see, the variables are weakly correlated. That means that the variables might not be in the same PCA output order, so the output of V1 might not be the principal axis, and that there is no strong linear relationship between any single variable and the target. That doesn't mean that there is no relationship, but there isn't a linear one at least. We can see if plotting out some of the variables can help us get more information about our columns.



There appears to be some noticeable differences between the fraud and non-fraud distribution plots. This gives us hope that the models will be able to pick up differences between the two types of transactions. These graphs also show that there can be a number of outliers that we might want to eliminate. If we eliminate all outliers (more than 3 standard deviations away from the mean), we get rid of 37,815 rows, 444 of which are fraudulent. I decided against dropping outliers from the dataset because we would lose 90% of our fraudulent cases and we do not know which columns we should drop them from. It would be a bit arbitrary to drop from some columns and not others, so I refrained from removing them. If our models were unable to predict properly, I would remove them. As a follow up to this assignment, I can try another iteration of training with non-outlier data.

One final point I wanted to see was feature selection. Could we potentially reduce the number of features in our dataset? Although we can look at the distribution plots like we did above, I wanted to see if we could put the input data in the PCA again. PCA, or principal component anlaysis, is a transformation on the data to values of linearly uncorrelated data (this explains why our correlation plot does not tell us much information). Another benefit of this is that we can see how much variance each PCA component explains. We can use this for feature selection since we can look at the number of needed PCA components to explain a sufficient amount of the data. We can also look for the 'elbow' of the curve to see where the features start to provide limited information and can be potentially cut.



As we can see, the first principal component only explains around 35% of the variance of the model. This means that we shouldn't use the elbow technique. We could use the cumulative sum of the explained variance until it reaches around 90%, cutting 11 columns. I ultimately decided against this because the dataset is already the product of PCA, so if we were to run it through PCA again, it would lose even more meaning.

There are a number of different models that we will want to implement and different ways we want to handle the imbalance of the data. The models we will use are a logistic regression, an SDG classifier, a Random Forest classifier, ax XGBoost classifier, and simple neural network classifier and a more complex neural network classifier. These are all classification models that will take the input data and predict a binary outcome, fraudulent or non-fraudulent transaction.

We will tune these non-benchmark models with Bayesian optimization to find the best parameters.

Logistic Regression (benchmark):

$$Ln\left(\frac{P}{1-P}\right) = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_k X_k$$ [7]

The simplest one classifier, is the logistic regression with the original imbalanced dataset. The reason why we are using this is because it is a basic form of binary classification. It estimates a multilinear regression from the columns and turns the regression into a sigmoid. Since it is the simplest classifier, we will use it as a benchmark. We will not tune the parameters nor will we balance the dataset.

SGD Classifier:

The next model we will use the stochastic gradient descent classifier (or SDG). This classifier implements linear models with a stochastic approximation of the gradient descent optimization, replacing the actual gradient with the estimate of the dataset. [8] We wanted to include this since it is another form of linear classification. The parameters we will tune are its alpha, max_iters, and its l1_ratio. The alpha is the constant that is multiplied by the regularization term. This means it regularizes the coefficients, but it also affects the learning rate, since the penalty is set to optimal. The learning rate is the size of each step that our model takes when moving towards the minimum of our loss function. We want to try a wide range of different learning rates to see which one gives the best predictive power. We want a high enough learning rate so that we can find the minimum, but we don't want it too large that we overshoot it. The lower the learning rate is, the higher our number of iterations has to be. For this reason, we also hyptertune the maximum number of iterations the classifier should use. The other function that we hyptertune is the l1_ratio, which is the parameter that determines if the function should be more similar to a ridge regression or a lasso regression. The difference is how the cost function is computed.

[7] Grace-Martin, Karen. "Link Functions and Errors in Logistic Regression." *The Analysis Factor*, 13 Dec. 2018, www.theanalysisfactor.com/link-functions-and-errors-in-logistic-regression/.

[8] "Stochastic Gradient Descent." *Wikipedia*, Wikimedia Foundation, 5 Dec. 2019, en.wikipedia.org/wiki/Stochastic_gradient_descent.

$$\sum_{i=1}^{n}(y_i - \sum_{j=1}^{p} x_{ij}\beta_j)^2 + \lambda \sum_{j=1}^{p} \beta_j^2$$

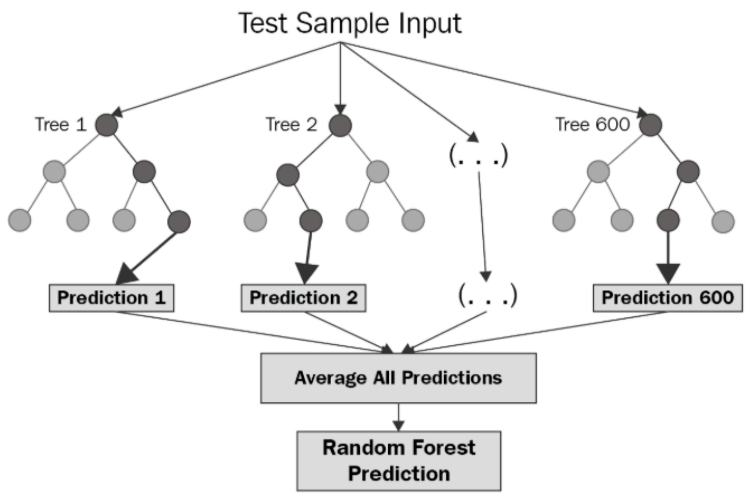
$$\sum_{i=1}^{n}(Y_i - \sum_{j=1}^{p} X_{ij}\beta_j)^2 + \lambda \sum_{j=1}^{p} |\beta_j|$$ [9]

Ridge (l1) Lasso (l2)

The lasso regression uses the absolute value of the magnitude, so when features are less important, they tend to shrink to zero. This is useful in feature selection when we have a large number of features. The ridge regression has the squared magnitude, so that the features that aren't dropped, but less important. The reason why we want to tune this parameter is to see if we should be dropping features (high l1_ratio) or if we should be keeping them (low l1_ratio). This allows us to try to get the benefits of both regressions in a single classifier.

<u>Random Forest</u>

We include the random forest because it is a tree-based model and it is unlikely to overfit the dataset. It is a bagging technique in which multiple decision trees are run in parallel. Even though each individual tree might be overfitting, we take an average of each of the trees to improve predictive accuracy and prevent overfitting. Each tree is given a number of random columns to build off of.



[10]

The parameters that we want to tune are the number of estimators, the max depth of each tree, and the minimum number of samples we need for a tree to split. We want to make sure we have a high enough number of trees that we include a lot of different combination of columns, but not too many where we could lose predictive power. N_estimators is the number

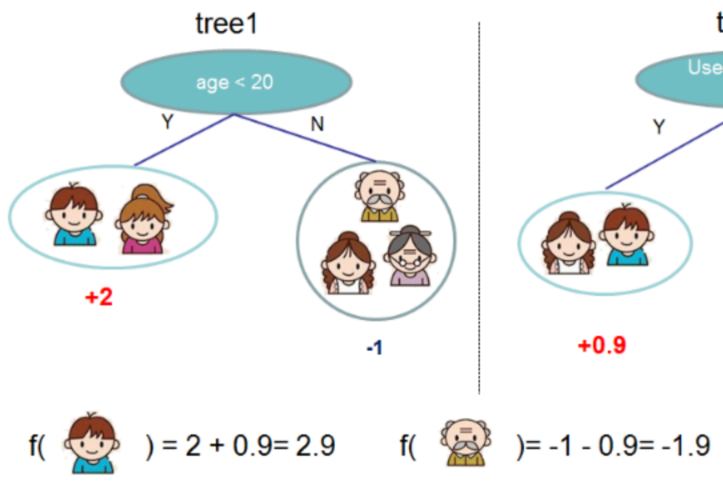
[9] Nagpal, Anuja. "L1 And L2 Regularization Methods." *Medium*, Towards Data Science, 14 Oct. 2017, towardsdatascience.com/l1-and-l2-regularization-methods-ce25e7fc831c.

[10] Chakure, Afroz. "Random Forest and Its Implementation." *Medium*, Towards Data Science, 7 July 2019, towardsdatascience.com/random-forest-and-its-implementation-71824ced454f.

of trees that are in the forest. The max depth is the maximum number of decisions that a single tree can make. The deeper the tree goes, the more precise the model can get, but it runs the risk of overfitting to the data. Finally the minimum number of samples parameter specifies how many data samples are needed to make a decision. Since we are going to be balancing our datasets, we don't need to alter the weighting of the sample leaves.

XGBoost

The other tree-based model we will be using is XGBoost. It is a boosting algorithm that predicts on the residual on the results of the previous trees. It fixes what it has predicted, by predicting on the error term, and then adds a new tree to improve. This is repeated until the



[11]

A simplified example of how XGBoost would work when predicting daily computer usage.

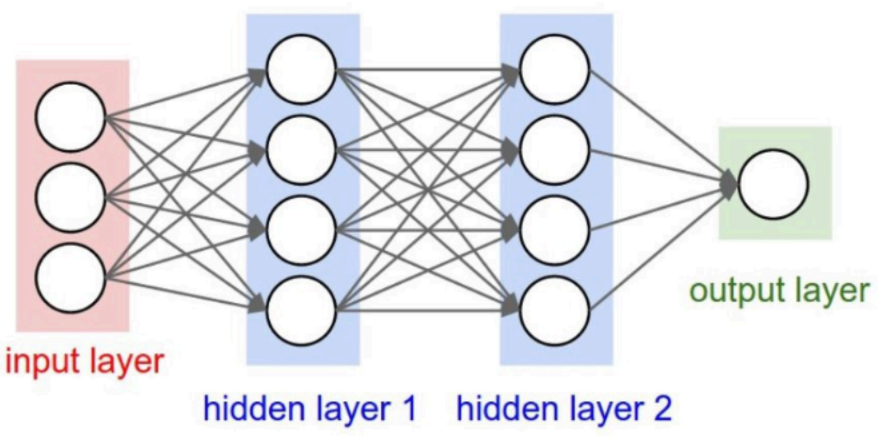
I wanted to include another tree-based model as well as a boosting technique. This model also has been used to win multiple Kaggle competitions, so I wanted to make sure to include it in my capstone. The parameters I were tuning were the number of estimators, the learning rate, the max depth, the gamma, the min_child_weight, and the colsample_bytree. We already discussed the estimators, depth, and learning function. The gamma is the regularization term in the xgboost, similar to the alpha in our SDG classifier. The min_child_weight is the minimum weight that a leaf needs to continue splitting. If the minimum is not met, the tree will no longer partition. The effect is similar to the min_samples_split in random forest. The higher the weight of the min_child_weight, the more conservative the model is.[12] Finally, the colsample_bytree is the number of columns that should be included in the subsampling. For each tree, it will select a certain percentage of columns to subsample from when creating a tree.

[11] "Introduction to Boosted Trees." *Introduction to Boosted Trees - Xgboost 1.0.0-SNAPSHOT Documentation*, xgboost.readthedocs.io/en/latest/tutorials/model.html.

[12] "XGBoost Parameters." *XGBoost Parameters - Xgboost 1.0.0-SNAPSHOT Documentation*, xgboost.readthedocs.io/en/latest/parameter.html.

Sequential Neural Networks

The final models that we will create are sequential neural networks. We will build two different types of neural networks, a simplistic neural network with only one hidden layer, and a deeper neural network, with 3 hidden layers and 2 dropouts.



[13]

A sequential neural network is an algorithm that is modeled after the human brain.[14] It is used to recognize patterns and clusters of data. We provide the model the input data with labeled data and it finds patterns and tries to then predict on what it has previously has learned. We used two different types of neural networks to see if patterns are easily found with only a single layer or if we need multiple layers to be able to predict fraud accurately. In our deeper neural network, we also add dropouts so we don't overfit our model. It will randomly select neurons to be ignored during the training phase.

Methodology

Now that we know what models we want to use, it is time for us to split the data. We want to split the data so we can prevent our model from overfitting the data. Since we want our model to work well on data it has never seen before, we split up our dataset into two different parts, the training dataset and the testing dataset. We then fit and tune our model to the training dataset and make predictions on the test dataset. If we did not do this and trained our model

[13] Allibhai, Eijaz. "Building A Deep Learning Model Using Keras." *Medium*, Towards Data Science, 21 Nov. 2018, towardsdatascience.com/building-a-deep-learning-model-using-keras-1548ca149d37.

[14] Nicholson, Chris. "A Beginner's Guide to Neural Networks and Deep Learning." *Skymind*, skymind.ai/wiki/neural-network#define.

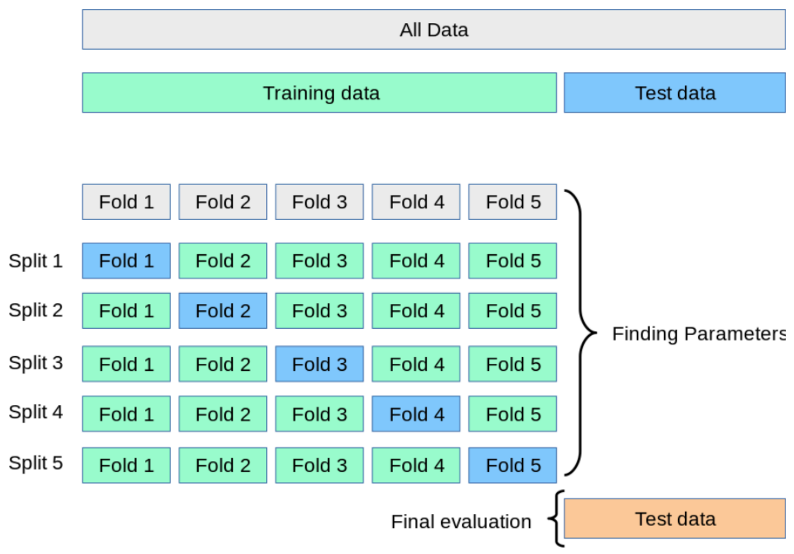
on the entire dataset, the accuracy of the model in the dataset would increase, but it would likely not be able to predict data that it hasn't seen before, since it has become overfit. Once we split, then we can start balancing the training dataset. We make sure to balance it after we split the original dataset in order to prevent information leaking into the model, defeating the purpose the train test split.

For all the models, excluding the benchmark, we will train our dataset on three different types of input, the imbalanced dataset, an under sampled dataset, and an oversampled dataset. The reason why we will want to rebalance the dataset is to give the model enough fraudulent cases it is able to accurately predict the differences between them and non-fraudulent cases. If there are not enough fraudulent cases, it might just predict that nothing is fraudulent. We will rebalance our data in two different ways. The first way is by reducing the number samples so the number of non-fraudulent cases is around 50%. This will help with our computation time, but will likely lead to a major loss of information, potentially ruining our models' predictive power. The other way we will rebalance is by oversampling the dataset using SMOTE. SMOTE synthetically creates data points by connecting that data points of the minority class and creating synthetic points along those lines. This helps us retain information about all of our minority class, but it does bias our model towards our training sample.

Once we have our three different prepared datasets, we can define our models.[15] The inputs of the models will be the parameters that we want to optimize. I made the models into functions so we can more easily score the models and optimize them with cross validation scoring and Bayesian optimization respectively.

Similarly to why we use split our dataset into training sets and testing sets, we used cross validation to score our models. What cross validation does is split up the training dataset into a certain number of folds, then trains the data on n-1 folds, using the leftover fold as validation for the model. This is to ensure that even though we split the data, we still are able to predict on potential subcategories within our training set.

[15] We discussed other preprocessing methods such as outlier removal, feature engineering, feature selection, and PCA above.

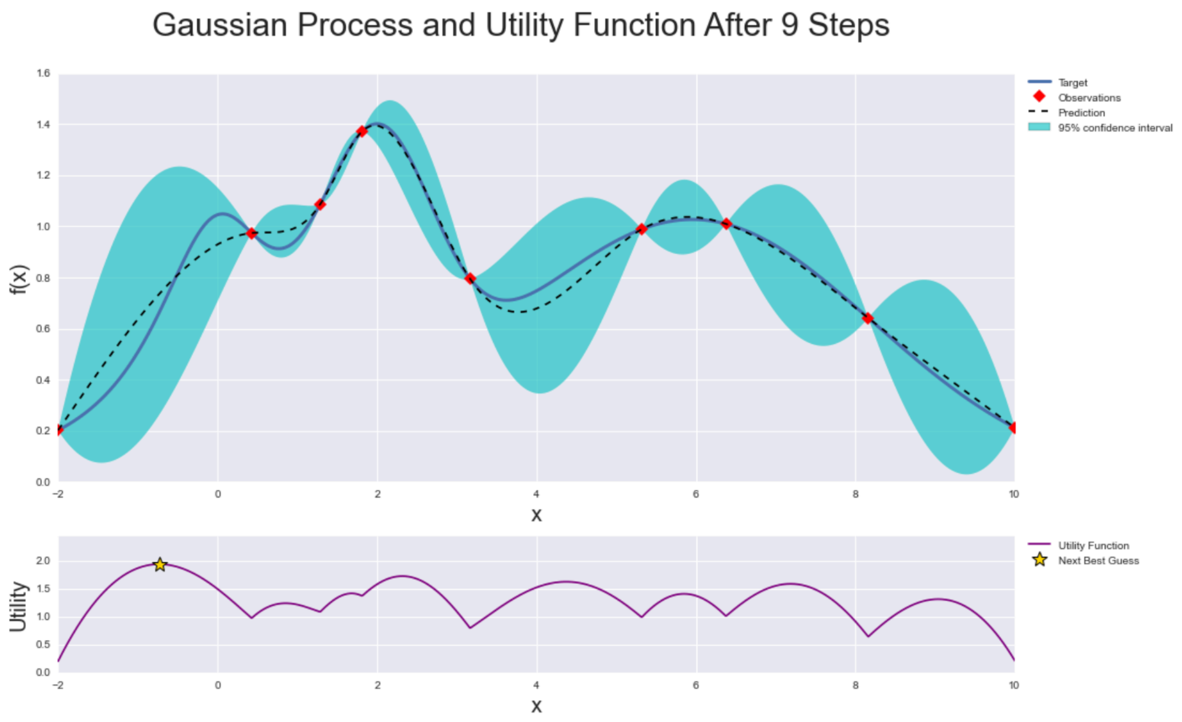


[16]

We score each of these iterations and take the average of the validation scores (which we used averaged precision), to get a final score for the model overall.

Once we have our models defined, we then optimize the hyper parameters with Bayesian optimization. What Bayesian optimization does is sample the hyperparameter space randomly and gets the confidence interval for each point of the function.

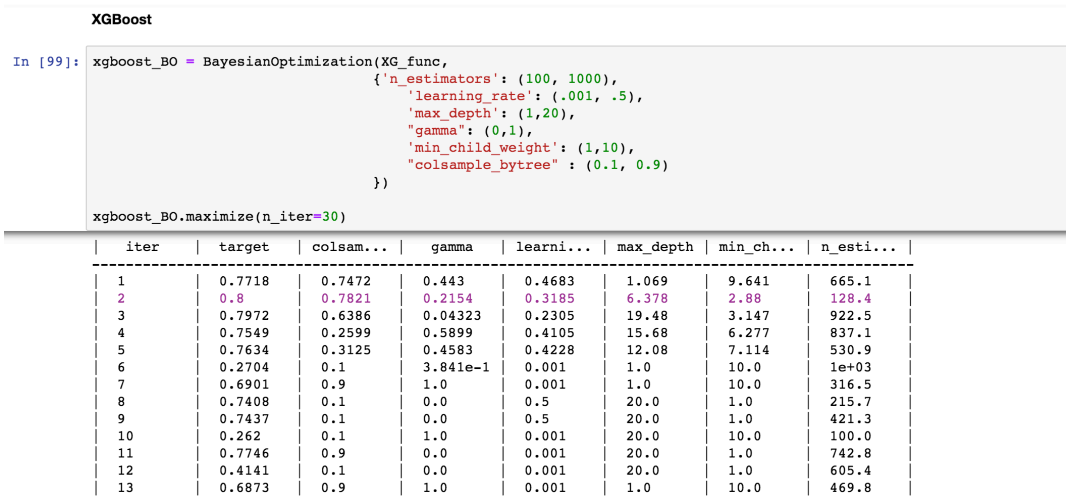
[16] "3.1. Cross-Validation: Evaluating Estimator Performance." *SciKit Learn*, scikit-learn.org/stable/modules/cross_validation.html.



17

In the beginning, it will probe places where there are large confidence intervals and ‘explore’ the function. As the functions iterate, it then starts ‘exploiting’ or probing areas where utility function is maximized. This is to try and minimize the number of steps needed to find the optimal combination of hyper parameters.

With all of this set, I put in the ranges for the hyperparameters and let the models optimize.



XGBoost

```
In [99]: xgboost_BO = BayesianOptimization(XG_func,
                                   {'n_estimators': (100, 1000),
                                       'learning_rate': (.001, .5),
                                       'max_depth': (1,20),
                                       "gamma": (0,1),
                                       'min_child_weight': (1,10),
                                       "colsample_bytree" : (0.1, 0.9)
                                   })

         xgboost_BO.maximize(n_iter=30)
```

```
|   iter    |  target   | colsam... |   gamma   | learni... | max_depth | min_ch... | n_esti... |
-------------------------------------------------------------------------------------------------
|  1        |  0.7718   |  0.7472   |  0.443    |  0.4683   |  1.069    |  9.641    |  665.1    |
|  2        |  0.8      |  0.7821   |  0.2154   |  0.3185   |  6.378    |  2.88     |  128.4    |
|  3        |  0.7972   |  0.6386   |  0.04323  |  0.2305   |  19.48    |  3.147    |  922.5    |
|  4        |  0.7549   |  0.2599   |  0.5899   |  0.4105   |  15.68    |  6.277    |  837.1    |
|  5        |  0.7634   |  0.3125   |  0.4583   |  0.4228   |  12.08    |  7.114    |  530.9    |
|  6        |  0.2704   |  0.1      |  3.841e-1 |  0.001    |  1.0      |  10.0     |  1e+03    |
|  7        |  0.6901   |  0.9      |  1.0      |  0.001    |  1.0      |  10.0     |  316.5    |
|  8        |  0.7408   |  0.1      |  0.0      |  0.5      |  20.0     |  1.0      |  215.7    |
|  9        |  0.7437   |  0.1      |  0.0      |  0.5      |  20.0     |  1.0      |  421.3    |
|  10       |  0.262    |  0.1      |  1.0      |  0.001    |  20.0     |  10.0     |  100.0    |
|  11       |  0.7746   |  0.9      |  0.0      |  0.001    |  20.0     |  1.0      |  742.8    |
|  12       |  0.4141   |  0.1      |  0.0      |  0.001    |  20.0     |  1.0      |  605.4    |
|  13       |  0.6873   |  0.9      |  1.0      |  0.001    |  1.0      |  10.0     |  469.8    |
```

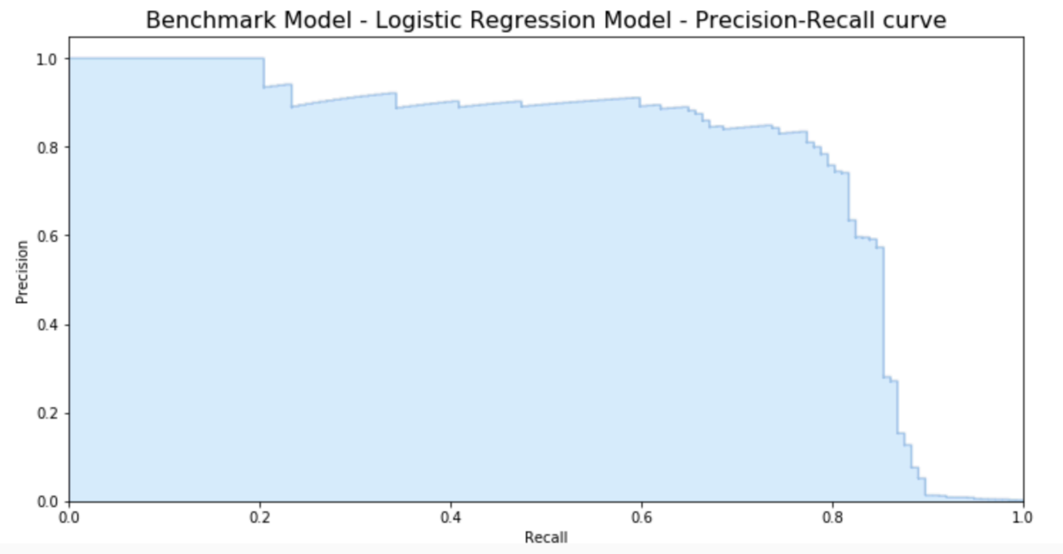
XGBoost algorithm find the optimized hyperparameters for its inputs

[17] Fmfn. “Fmfn/BayesianOptimization.” *GitHub*, 13 Nov. 2019, github.com/fmfn/BayesianOptimization.

Results and Conclusion

After multiple hours or training the models, we saw some things were working, while others were not up to par. The logistic regression that was run on the imbalanced dataset, our benchmark, had an AUPRC score of .517. The precision of the benchmark was pretty good, .907, but its recall was lacking, only .569.

```
Scoring Results for: Benchmark Model - Logistic Regression
------------------------------
Precision: 0.907
Recall: 0.569
F1 Score: 0.7
Confusion Matrix:
[[85298     8]
 [   59    78]]
AUPRC Score: 0.517
```



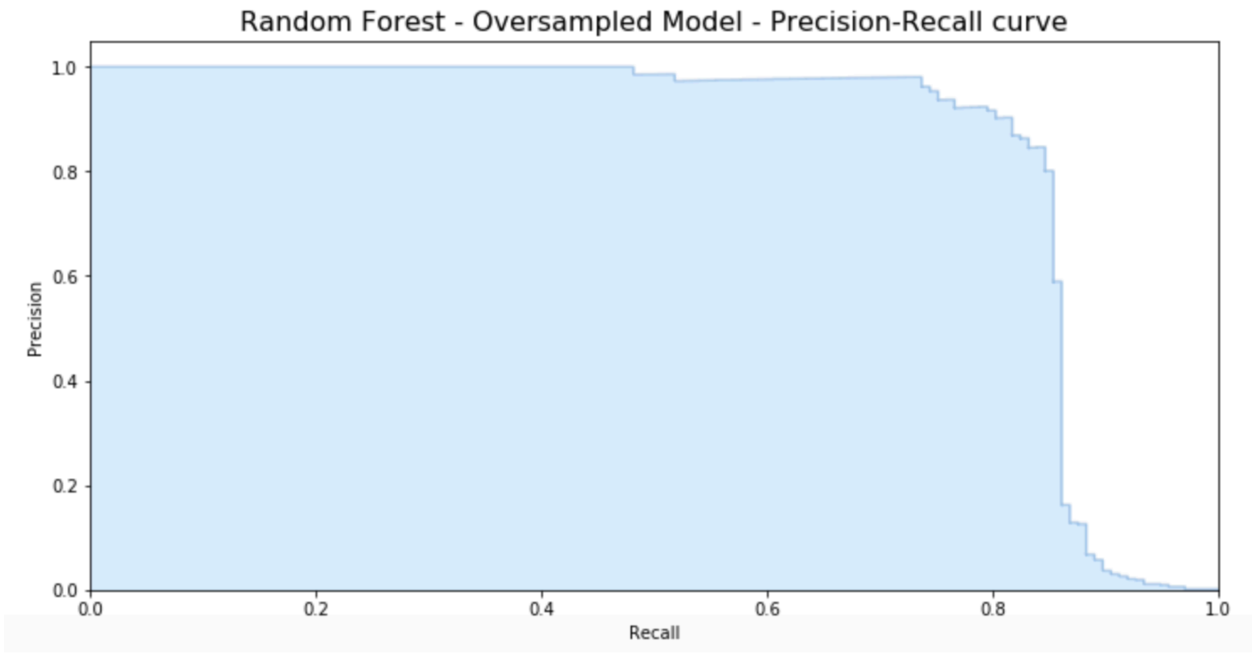
There a few notes interesting points I discovered when I was looking over the results of the models. Overall the SDG Classifier appeared the underperform when predicting on the test dataset, regardless of the input dataset. It never got a higher score than the benchmark. If I had to rerun the project, I would try tuning more of its parameters. There a large number of parameters to tune, so I tuned the parameters that I thought would have the largest impact. However, that didn't appear to be the case.

I was also a little disappointed with the performance of the neural networks. Although the performance wasn't bad, I thought it would be one of the best models to work with. I think that I would have to try using different layers in the neural network or a more complex neural network (an RNN or KNN instead of a sequential one). I also believe that if I had more time, I could try to use some form of Bayesian optimization to see which structure of neural networks work the best, rather than just fitting on two different types networks. When I reattempt this problem, I will explore what other people have done on their kernels and see how to improve my models.

Another thing I discovered was that under sampling the dataset was extremely negative in model's predictive ability. Under sampling data is a valid approach to balancing a dataset, however when the minority class is so small, there is too much information that is thrown out. The models are not able to predict properly when given such a small dataset.

The best model to predict fraud was the Random Forest model predicting on the oversampled dataset. It had a final AUPRC score of .727.

```
Scoring Results for: Random Forest - Oversampled
------------------------------
Precision: 0.889
Recall: 0.818
F1 Score: 0.852
Confusion Matrix:
[[85292    14]
 [   25   112]]
AUPRC Score: 0.727
```



The XGBoost model predicting on imbalanced data was also very good, with an AUPRC of .726 when predicting on the imbalanced dataset, which was a little bit surprising. I think because there was such a small increase in performance from the imbalanced dataset to the oversampled data, it shows that the models were able to find the patterns to determine what was fraudulent data without the synthetic results. I think in some cases, the oversampled data biased the models too heavily towards the training dataset, lowering the predictive power for some models. For the final model, I would use the Random Forest model on oversampled data since it is a bit more conservative, having a higher recall, but a lower precision than the XGBoost model. I think this is a fair reason because the cost of missing fraudulent activity is higher than the inconvenience of flagging normal transactions.

I believe that the results of the final model can be trusted and the performance will not change too much based on the input spaces because we used cross validation and made sure there was no leakage of data from the test set to the training set. Since the input data is an output of PCA, we can’t get extra data to test upon. I think that this is a somewhat good answer to the Kaggle competition of finding ways to classify fraudulent activities. It is able to have a pretty good average precision score, which is our main metric. That said, we can always improve. I think the random forest dataset can be used a new benchmark and that we can try to improve our XGBoost models and neural networks. I will work with different configurations and potentially optimize twice, once for a large range for the parameters, and a second time with a smaller range around the optimized parameters.

I found this project to be enjoyable and will definitely try to do similar Kaggle competitions in the future. I enjoy the process of exploring the data and developing different models to use. This is definitely a feasible way to answer this type of problem. If the task was for production, there would be a number of tasks I would work on. I would likely use this first attempt to see what the feature important was for each model and use that for potential feature selection. I would also try and find ways to get a more balanced dataset so we aren’t predicting with only around 500 instances of fraud. I would also try and evergreen my model, consistently training it on new data that comes in to help improve the overall results.

"3.1. Cross-Validation: Evaluating Estimator Performance." *SciKit Learn*, scikit-learn.org/stable/modules/cross_validation.html.

Allibhai, Eijaz. "Building A Deep Learning Model Using Keras." *Medium*, Towards Data Science, 21 Nov. 2018, towardsdatascience.com/building-a-deep-learning-model-using-keras-1548ca149d37.

Chakure, Afroz. "Random Forest and Its Implementation." *Medium*, Towards Data Science, 7 July 2019, towardsdatascience.com/random-forest-and-its-implementation-71824ced454f.

"Confusion Matrix." *Wikipedia*, Wikimedia Foundation, 22 Oct. 2019, en.wikipedia.org/wiki/Confusion_matrix.

"Credit Card Fraud Statistics." *Shift Processing*, Shift Credit Card Processing, 2019, shiftprocessing.com/credit-card-fraud-statistics/.

Fmfn. "Fmfn/BayesianOptimization." *GitHub*, 13 Nov. 2019, github.com/fmfn/BayesianOptimization.

Grace-Martin, Karen. "Link Functions and Errors in Logistic Regression." *The Analysis Factor*, 13 Dec. 2018, www.theanalysisfactor.com/link-functions-and-errors-in-logistic-regression/.

"Introduction to Boosted Trees." *Introduction to Boosted Trees - Xgboost 1.0.0-SNAPSHOT Documentation*, xgboost.readthedocs.io/en/latest/tutorials/model.html.

Machine Learning Group. "Credit Card Fraud Detection." *Kaggle*, Google LLC, 23 Mar. 2018, www.kaggle.com/mlg-ulb/creditcardfraud.

Nagpal, Anuja. "L1 And L2 Regularization Methods." *Medium*, Towards Data Science, 14 Oct. 2017, towardsdatascience.com/l1-and-l2-regularization-methods-ce25e7fc831c.

Nicholson, Chris. "A Beginner's Guide to Neural Networks and Deep Learning." *Skymind*, skymind.ai/wiki/neural-network#define.

"Precision and Recall." *Wikipedia*, Wikimedia Foundation, 3 Dec. 2019, en.wikipedia.org/wiki/Precision_and_recall.

Shung, Koo Ping. "Accuracy, Precision, Recall or F1?" *Medium*, Towards Data Science, 8 June 2018, towardsdatascience.com/accuracy-precision-recall-or-f1-331fb37c5cb9.

"Stochastic Gradient Descent." *Wikipedia*, Wikimedia Foundation, 5 Dec. 2019, en.wikipedia.org/wiki/Stochastic_gradient_descent.

"XGBoost Parameters." *XGBoost Parameters - Xgboost 1.0.0-SNAPSHOT Documentation*, xgboost.readthedocs.io/en/latest/parameter.html.