Predictive Modeling

Performance Metrics

In order to quantify the performance of the model on training and testing sets, we need a performance metric. However, unlike regression models, classification models have different forms to compute performance metrics. The three metrics we will compute are classification accuracy, precision, recall, and F1-score.

- **Classification Accuracy** is the number of correct predictions made as a ratio of all predictions made. This is the most common evaluation metric for classification problems, it is also the most misused. It is really only suitable when there is an equal number of observations in each class and that all predictions and prediction errors are equally important, which is not the case.
- **Classification Report**: Sci-kit learn does provide a convenience report when working on classification problems. This report displays the precision, recall, f1-score and support for each class.
    - **Precision**: the ability of a classifier not to label an instance positive that is actually negative. What percent of your predictions were correct?
    - **Recall** the ability of a classifier to find all positive instances. For each class, it is defined as the ratio of true positives to the sum of true positives and false negatives. What percentage of the positive cases did you catch?
    - **F1 Score**: The weighted harmonic mean of precision and recall such that the best score is 1.0 and the worst is 0.0. Generally speaking, F1 scores are lower than accuracy measures as they embed precision and recall into their computation. What percent of positive predictions were correct?

What to minimize?

- We know that there will be some error associated with every model that we use for predicting the true class of the target variable. This will result in False Positives and False Negatives (model classifying things incorrectly as compared to the actual class) There are no rules that say what should be minimized in situations and depends on the business needs and the context of the problem. Based on these needs we want to minimize either False Positives or False negatives.
    - **False Positives (FP)**: False positives are the cases when the actual class of the data point was 0 (False) and the predicted is 1 (True). False is

because the model has predicted incorrectly and positive because the class predicted was a positive one.
        - A network that is not an attack and the model classifying the network as an attack.
    - **False Negatives (FN)**: False negatives are the cases when the actual class of the data point was 1 (True) and the predicted is 0 (False). False is because the model predicted incorrectly and negative because the class predicted was a negative one (0).
        - A network that is an attack and the model classifying has normal.
- **Minimizing False Negatives**
    - For our network problem minimizing false negatives holds a higher priority than minimizing false positives. If that is the case, in order to capture all attack cases, we might end up making a classification when the network actually does not have an attack. This is less dangerous than the alternative.

Precision and Recall

- **Precision**: is a measure that tells us what proportion of networks we have classified as attacks, actually are attacks. The predicted positives and the network that are attacks are True Positives (TP).
- **Recall and Sensitivity**: Recall is a measure that tells us what proportion of networks actually had an attack that was classified by the algorithm as having attacks. The actual positives (networks with attacks are TP and FN) and the network classified by the model having an attack is TP.
- What to use Precision or Recall?
    - Recall gives us information about a classifier's performance with respect to false negatives (how many did we miss), while precision gives us information about its performance with respect to false positives (how much did we caught)
    - If we focus more on minimizing false negatives, we want our Recall to be as close to 100% as possible without precision being too bad.

Preprocessing

Before implementing algorithms and creating models we first need to preprocess our data. Data preprocessing is a data mining technique that involves transforming raw data into an understandable format. Our data is already in a defined data frame however, it is not in a format that a sci-kit machine learning algorithm can read, therefore we need to preprocess our data for modeling.

The first preprocessing step is to create dummy variables from our categorical variables. Machine learning classifiers can only read numeric, hence why we create dummies variables of our categorical columns. The next step was to drop redundant columns. After converting our columns to the correct data types, and dropping the columns so each data frame is of equal size we split the training and testing data frames into X sets and y sets. X sets containing the features and y sets containing our predictors. In total, we have four sets of data, X_train, y_train, X_test, and y_test. Our final step in preprocessing is standardizing our training sets.

Standardizing and normalizing is simply a case of getting all your data on the same scale. If the scales for different features are widely different, this can have a knock-on effect on the ability of the model to learn. Ensuring standardized feature values implicitly weigh all features equally in their representation.

```
#standardize features

sc = StandardScaler()

#fit the scaler to the training data and transform
X_train_std = sc.fit_transform(X_train)

#apply the scaled to the test data
X_test_std = sc.transform(X_test)
```

Baseline model- Logistic Regression

Classification tries to predict, which of a small set of classes, an observation belongs to. Our goal is to find y, a label based on knowing a feature of vector x. For this problem our goal is to predict the label (attack or normal) this is our y-variable.

For our baseline model, we will use a logistic regression model. In short, a Logistic Regression is a Machine Learning algorithm that is used for classification problems, it is a predictive analysis algorithm that is based on the concept of probability. We first import the required sci-kit learn packages and instantiate the logistic regression model. Next, we train on the training sets, then we compute the accuracy of both training sets and testing sets. Finally, we create a classification report.

```
Logistic Regression accuracy training set: 0.9338660096611745
Logistic Regression accuracy testing set: 0.8081183500947384
```

	precision	recall	f1-score	support
0	0.97	0.82	0.89	56000
1	0.92	0.99	0.95	119341
avg / total	0.94	0.93	0.93	175341

	precision	recall	f1-score	support
0	0.95	0.61	0.74	37000
1	0.75	0.97	0.85	45332
avg / total	0.84	0.81	0.80	82332

The results showcase an accuracy score of 93% for the training data and 81% for the testing data. For our classification report, we can see the difference between precision and recall on both attack and normal classified data. Our test set indicates a precision of 95% for classifying normal data, the percentage of the results which are relevant, and a recall of 61% which is the percentage of total relevant results correctly classified. Compared to our attack precision which is 75% and recall of 97%.

Depending on the problem you are trying to solve, you could either give a higher priority to maximizing precision or recall. In this case, detecting harmful attacks are our main concern. Another metric that takes into account both precision and recall is the F1-score. This is the harmonic mean of precision and recall. For our baseline model, our F1- Score is 74% for normal and 85% for an attack.

Tuning model for better results

Our next goal is to see if we can raise accuracy level, precision, and recall. I used two techniques to compute a higher score. The first technique was to tune my models' hyperparameters. Using a random search, the best parameters computed was a L2 regression, also known as a ridge, and a C value of 0.001. The results did not change, and if anything model accuracy decreased. The second technique was to handle imbalance data. Using three different techniques to handle imbalance data (using a weighted parameter on the model, upsampling imbalance data, downsampling imbalance data), I was able to marginally increase the percentages of models accuracy, precision, and recall.

Weighted

```
Logistic Regression accuracy training set: 0.9274556435745205
Logistic Regression accuracy testing set: 0.8306490793373172

              precision    recall  f1-score   support

           0       0.89      0.88      0.89     56000
           1       0.95      0.95      0.95    119341

 avg / total       0.93      0.93      0.93    175341

              precision    recall  f1-score   support

           0       0.89      0.71      0.79     37000
           1       0.80      0.93      0.86     45332

 avg / total       0.84      0.83      0.83     82332
```

Upsampling

```
Logistic Regression accuracy training set: 0.9154607385559028
Logistic Regression accuracy testing set: 0.8261064956517514
```

	precision	recall	f1-score	support
0	0.95	0.88	0.91	119341
1	0.89	0.95	0.92	119341
avg / total	0.92	0.92	0.92	238682

	precision	recall	f1-score	support
0	0.89	0.71	0.79	37000
1	0.80	0.93	0.86	45332
avg / total	0.84	0.83	0.83	82332

Downsampling

```
Logistic Regression accuracy training set: 0.9156071428571428
Logistic Regression accuracy testing set: 0.8303089928581839
```

	precision	recall	f1-score	support
0	0.94	0.88	0.91	56000
1	0.89	0.95	0.92	56000
avg / total	0.92	0.92	0.92	112000

	precision	recall	f1-score	support
0	0.89	0.71	0.79	37000
1	0.80	0.93	0.86	45332
avg / total	0.84	0.83	0.83	82332

From our baseline model and resampling techniques, our best Recall score for attacks was **97%**, from our baseline logistic model. Indicating the model successfully predicted 97% label attacks that were actually network attacks. However, our precision is 75% compared to our upsampling and downsampling methods precision's which were both 80%. My next goal is to see if I can increase the precision of attacks while maintaining a high recall percentage.

Ensemble Method

After training the logistic regression models, my next approach is to use other algorithms to see if performance can be improved. Ensemble methods is a machine learning technique that combines several base models in order to produce one optimal predictive model. We will use two ensemble methods, Random Forest and XGBoost.

Random Forest

- Random Forest model takes advantage of bagging process where it takes different training samples with replacement in order to get predictions for each observation and then averages all the predictions to obtain the estimation.

```
Random Forest accuracy Training set:  0.9948785509378868
Random Forest accuracy Test set:,  0.8659694893844435
```

RF Testing set	precision	recall	f1-score	support
0	0.94	0.75	0.83	37000
1	0.82	0.96	0.89	45332
avg / total	0.88	0.87	0.86	82332

After training both training sets and testing sets our results to show a **99%** accuracy on training data and an **86.5%** accuracy on testing data. These are great results, but because of classification, accuracy is not an ideal performance metric. With our classification report, we can see a better distribution of our predicted data. The random forest model yields a precision of **82%** for classifying attacks and **96%** for recalling attacks.

Random Forest - Tuning

We generally see a random forest as a black box which takes in input and gives out predictions. The black box has a few levers we can tune, which affects the performance of the model. Parameters in the random forest are either to increase the predictive power of the model or make it easier to train the model. We will train the parameters of a random forest in order to see if we can optimize the performance. These parameters re n_esimators, max_feattures, max_depth, min_samples_split, min_samples_leaf, and bootstrap.

The method we will use to tune or parameters is a randomized search. Random search is a technique where random combinations of hyperparameters are used to find the best solution for the built model. It tries random combinations of a range of values.

The results are:

```
{'n_estimators': 10,
 'min_samples_split': 5,
 'min_samples_leaf': 4,
 'max_features': 'auto',
 'max_depth': 10,
 'bootstrap': True}
```

```
Random Forest accuracy Training set:  0.93495531564209 17
Random Forest accuracy Test set:,  0.810268182480688
```

```
Testing Set Classification Report:              precision    recall  f1-s
core   support

          0       1.00      0.58      0.73     37000
          1       0.74      1.00      0.85     45332

avg / total       0.86      0.81      0.80     82332
```

Tuning random forest gave us the worst accuracy compared to the random forest without tuning, however, our precision for classfying normal networks and our recall of attacks are at **100%.**