Introduction:

In machine learning, there are many algorithms to choose from. What algorithm to choose can be filtered analysing the problem and the data. Oftentimes, after the filtration process, we are still left with multiple algorithms to select from. Therefore, whenever feasible, training data on different algorithms and tuning each parameter of the algorithm is necessary. In this report, we dive into the practice of using and tuning different models. The report will go through each step involved in solving machine learning problem. First, we will define a problem. Second, we will tweak our data to better fit our problem. Third, we will define and tun the models. Lastly, we will compare and evaluate models.

The 3 models to be used in the report are:

1. XGBoost
2. Logistic Regression
3. Neural network

Problem Formulation:

The problem presented in the report is Jane Street Market Prediction[1] competition from Kaggle. We will tweak and simplify the problem. The competition involves in predicting whether a trade will be profitable or not given the input. The evaluation in the competition is using a utility function but for this experiment, we will tweak our evaluation.

Data used to train/validation/test the model is stored in the file name train.csv. The data contains date, weight, multiple resp (return) columns, and multiple feature columns. Each row represents a trade and different resp values represent the different returns. For simplicity, we will ignore the multiple resp value and weights. We will use one of the resp column to evaluate if the trade should be taken or not. If the resp is below 0, we will label 0, and 1 if resp above 1. The goal of our model will be to predict if a given trade should be 0 or 1. In other words, this will be a binary classification problem.

In the end we will compare which model makes more money or in other words which one can predict profitable trade more while minimizing losing trades. Whichever model does it the best will be classified as a better model.

Preparing the data:

Before initializing the models, we need to clean and prepare our data. We will first clear all the zero weights. Second, we will replace all the missing features with an arbitrary number. We will then extract our features and binary labels.

[1] Jane Street Market Prediction | Kaggle

Since there are ~2 millions of data entries, we will split it in 55% train – 20% Validation – 25% Test.

Finally, now that our data is ready, we can start designing models.

Approaches:

The three models mentioned in the introduction will be trained along with a baseline model in this section. For more accurate comparison, we will make all our models use the same loss function and same evaluation metrics. For loss function we will use binary cross-entropy/Log-Loss function. Log-Loss is taking a negative log of the predicted probability of the true result. For evaluation metric, we will use Area Under Curve. As seen in the figure, AUC is an area under the curve that models False-Positive vs True-Positive rate. We want the area to be high as possible as higher area means better predictions. More information on the loss function and AUC can be found under Appendix.

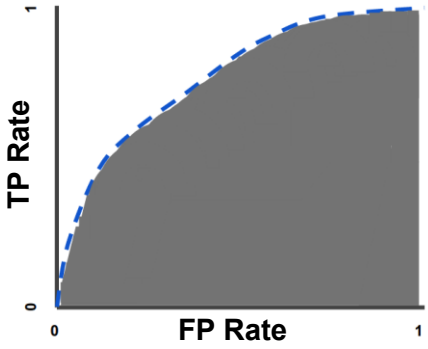


Figure 1: AUC curve

$$H_p(q) = -\frac{1}{N}\sum_{i=1}^{N} y_i \cdot log(p(y_i)) + (1 - y_i) \cdot log(1 - p(y_i))$$

Figure 2: Log-Loss function

Baseline:

For baseline we will utilize scikit-learn's dummy classifier. For classification we will be using the most frequent classifier. The most frequent classifier will always predict the most frequent label in the training set.

We do not need to tune/optimize our dummy model as it is meant to be bad model.

	precision	recall	f1-score	support
0.0	0.00	0.00	0.00	245475
1.0	0.50	1.00	0.67	249847
accuracy			0.50	495322
macro avg	0.25	0.50	0.34	495322
weighted avg	0.25	0.50	0.34	495322

Accuracy: 0.5044132907482405

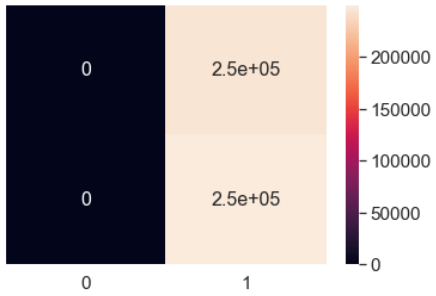


Figure 3: Classification report and confusion matrix

Xtreme Gradient Boosting:

```
print(optimal_param.best_params_)
```

```
{'learning_rate': 0.01, 'max_depth': 15, 'n_estimators': 1200}
```

Figure 4: Optimal parameter generated

The second model we are going try is one of Kaggle's favourite learning algorithm. XGBoost is known for its speed and its predictive power. XGBoost is based on a decision tree algorithm combined with other optimizing algorithms such as Parallel Learning to achieve the speed and power.

There are numerous parameters to play around with in XGBoost. We will be optimizing XGBoost by optimizing 3 hyperparameters. Each of the parameter is explained below.

- Learning rate: The rate at which the algorithm updates. Range: [0.01, 0.05, 0.005]
- N_esimators: Number of trees. Range: [600, 1000, 1200]
- Max_depth: Maximum depth of the decision tree. Higher means more complex but our model will become too powerful and will overfit. Range: [6, 11, 15]

To prevent overfitting, we will put sub_sample to 0.70 and colsample_bytree to 0.50. Sub_sample 0.70 will randomly select 0.70 of data every boosting iteration. This helps bring more randomness in the data. Also, colsample_bytree 0.50 will drop 50% of the columns when training which introduces even more randomness.

To select the best combination of parameters we will be using the grid search cross-validation from sklearn.

Figure 4 shows the validation AUC while training the model with the parameters from figure 3.

We took the model and tested against the test data. We can see the classification report and confusion matrix in figure 6 and 7. We will explain and compare them under discussion.

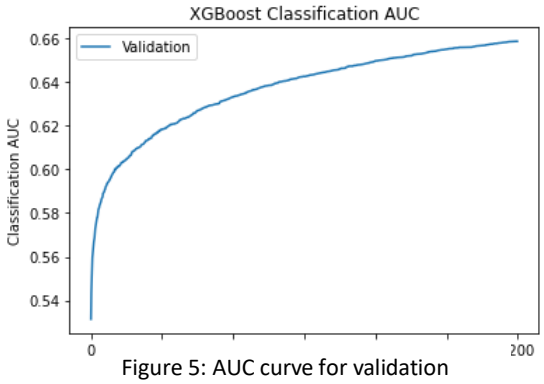


Figure 5: AUC curve for validation

	precision	recall	f1-score	support
0.0	0.61	0.58	0.59	245475
1.0	0.60	0.63	0.62	249847
accuracy			0.60	495322
macro avg	0.60	0.60	0.60	495322
weighted avg	0.60	0.60	0.60	495322

Accuracy: 0.6039445047867852

Figure 6: Classification report of XGBoost

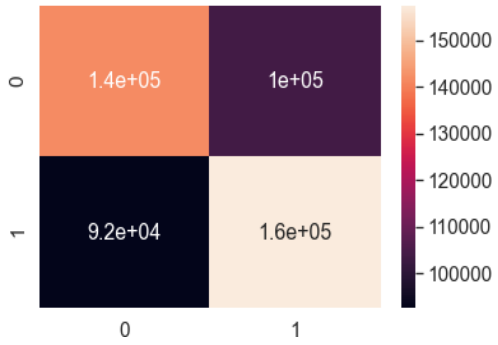


Figure 7: Confusion Matrix

The breakdown from figure 6:

True Negative: 1.4

False positive: 1.0

False Negative: 0.92

True positive: 1.6

Logistic regression:

Third model we will implement is a simple logistic regression model. We will be using Keras from TensorFlow to implement it. For logistic regression we will be optimizing two hyperparameters. One is the epochs, and another is the learning rate. Running grid search on full data would take a whole day, so we will be tuning our parameter with only 100,000 data entries.

```
print(grid.best_params_)
```

```
{'epochs': 700, 'learning_rate': 0.001}
```

Figure 8: Optimal parameter generated

For optimization algorithm, we will be using Adam's optimizer as it has shown to be one of the better optimizers from time to time.

Figure 9: Training loss and AUC validation

	precision	recall	f1-score	support
0.0	0.51	0.40	0.45	245475
1.0	0.52	0.63	0.57	249847
accuracy			0.52	495322
macro avg	0.52	0.51	0.51	495322
weighted avg	0.52	0.52	0.51	495322

Accuracy: 0.5156060098279504

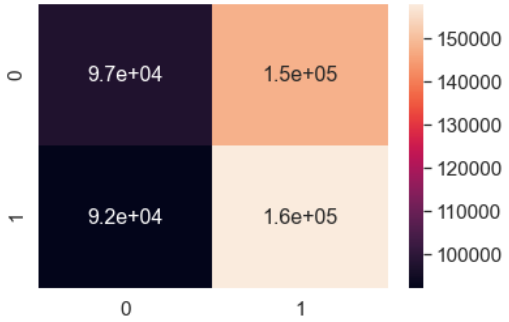


Figure 10: Classification report and confusion matrix

Our accuracy for logistic regression is 52% and breakdown for each category is listed below:

True Negative: 0.97

False positive: 1.5

False Negative: 0.92

True positive: 1.6

2-layer neural network:

In this model we try to solve our problem using a neural network. We will be using 2 hidden layers to compute an output. For the hidden layers' activation function, we decided to go with swish. A popular choice for activation function is ReLU, but according to the research conducted by Ramachandran et. Al.[2], swish outperformed ReLU. Therefore, we will be conducting our experiment with Swish.

The formula for swish is $x\sigma(\beta x)$ where beta is a constant or trainable constant. For the experiment, we will use the swish function provided by TensorFlow where the beta is set to 1.

Like the previous model, we will be using Adam's optimization. For hyper parameter tuning, we will be tuning the number of units in the hidden later, and the learning algorithm.

In neural network, with many data entries, we oftentimes run into a problem of overfitting. To prevent overfitting, we will be using a dropout technique. Dropout is dropping a portion of units that feed into the next layer. A paper published in 2014[3] prove dropout a successful strategy in preventing overfitting.

We will be using a fixed dropout rate of 50%. Ideally, you want to tune this parameter but due to the time demand of tuning, we will be using a fixed rate.

To further decrease grid search time, we reduced the number of data entries to 100000. The optimal parameter for hidden units came out to be 64 and the training rate 0.001. We used it to train and predict on test data. Below figures display the finding.

[2] Searching for Activation Functions (arxiv.org)

[3] srivastava14a.pdf (jmlr.org)

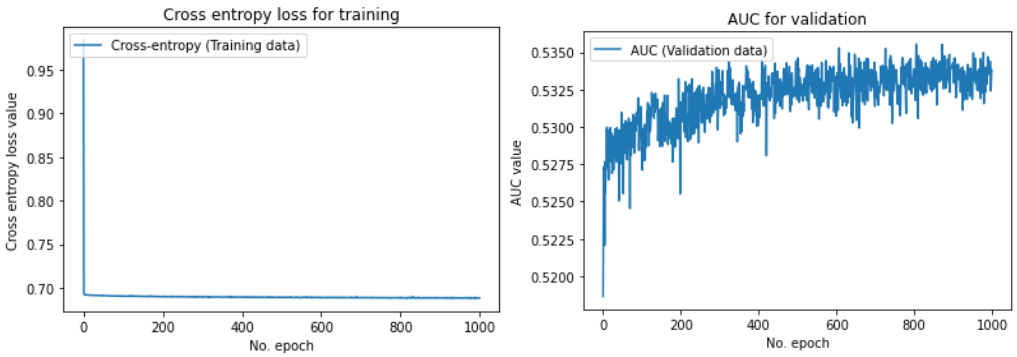
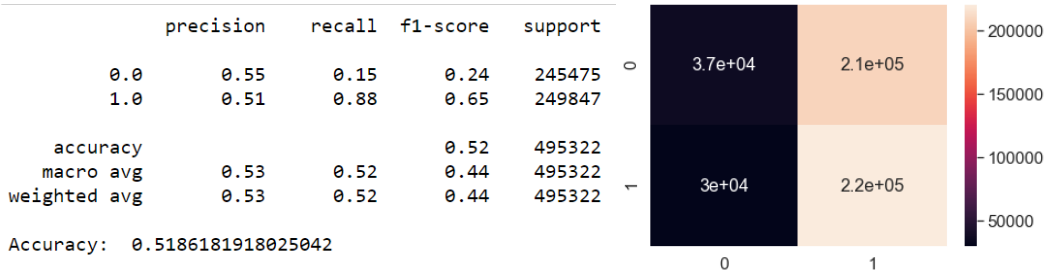


Figure 11: Training loss and AUC validation



	precision	recall	f1-score	support
0.0	0.55	0.15	0.24	245475
1.0	0.51	0.88	0.65	249847
accuracy			0.52	495322
macro avg	0.53	0.52	0.44	495322
weighted avg	0.53	0.52	0.44	495322

Accuracy: 0.5186181918025042

Figure 12: Classification report and confusion matrix

Our accuracy for logistic regression is 52% and breakdown for each category is listed below:

True Negative: 0.37

False positive: 2.1

False Negative: 0.3

True positive: 2.2

Discussion:

Comparing to the baseline model, all models did better. Accuracy of baseline is 0.50, XGBoost is 0.60, logistic regression and neural network is 0.52. We notice that accuracy of logistic regression and neural network is only slightly better. If all our model gave accuracy close to the baseline, we would need to reconsider the data or the problem. Since our XGBoost did better than baseline, we can move forward for additional comparison to select the final candidate.

If we compare 3 models amongst each other, we see that XGBoost has the highest accuracy. Since we are only executing the trade when the model predicts 1, we should look at the precision for each model. Precision will tell us the accuracy of positive prediction. Knowing the accuracy of positive prediction is crucial because the only time we are risking money is when the model predicts 1. Precision will tell us how often the trade we execute is profitable.

$$Precision = \frac{TP}{TP + FP}$$

Looking at the classification report for each model, we find XGBoost has precision of 0.60; logistic regression has precision of 0.52 and neural network has precision of 0.51.

Comparing the accuracy and precision, XGBoost has an edge over the other two model.

Third edge of XGBoost comes from its time to train. Training on XGBoost only took about 10 minutes, where our other two model took more an hour to train. This is a huge advantage for XGBoost over the two.

Conclusion and Future work:

The three models used were compared against the baseline. Our results indicated that the accuracy of logistic regression and neural network is 2% better than baseline. However, XGBoost produced an accuracy which is 10% higher than baseline. Comparing the three with each other we found XGBoost has an edge over the other two model in terms of accuracy, recall, and training time. It is evident that XGBoost is the best suitable solution to our problem.

To further improve our results, there are couple of things we can do. Firstly, we can factor in the weights and various type of return values. Secondly, we can do further exploratory data analysis to get to know our data more closely. In addition, we can perform various algorithm such as Boruta to help us filter our irrelevant features. All three points can help us obtain more relevant and accurate dataset. In terms of models, we can perform more in-depth hyperparameter tuning. Instead of tuning hyperparameter on a subset of data, we can perform tuning on the whole dataset. To make tuning faster, we can even consider random search instead of grid search.

Appendix:

Cross Entropy Loss: Understanding binary cross-entropy / log loss: a visual explanation | by Daniel Godoy | Towards Data Science

AUC curve: Classification: ROC Curve and AUC | Machine Learning Crash Course (google.com)