# Team A - Final Project Report Quantitative Trading Models - Will Jane Street Hire Us?

Authors: A0230632U Chua Peng Shaun, A0228402N Tan Chuan Xin, A0230521Y Teo Chen Ning

Emails: e0698539@u.nus.edu, e0673235@u.nus.edu, [e0697782@u.nus.edu](mailto:e0697782@u.nus.edu)

### **1. Abstract**

Buy low, sell high - is it that simple? Quantitative trading has rapidly risen in popularity in recent years, and a large part of this surge is due to advances in machine learning that allow split-second computation on enormous datasets to be possible. Quantitative trading uses various numbers related to a security, passes it through a decision model, and arrives at buy/pass/sell decisions to maximize profits. This project aims to create a predictive model to maximize profits in the stock market. Here, we aim to compare the effectiveness of traditional machine learning models against that of deep learning models to see which has the best results. In particular, we are interested to discover if the average investor will be able to create a viable machine learning model to be competitive in the space of quantitative trading.

Our findings indicate that the average person will have trouble creating and training quantitative trading machine learning models that can compete with the market due to limitations in computational power and time to train these models. Nonetheless, one can still easily create a model that outperforms random guessing.

### **2. Introduction**

**Kaggle Competition**

We have selected the [**Jane Street Market Prediction**](https://www.kaggle.com/c/jane-street-market-prediction/)[1]competition dataset from Kaggle as our project.

**Motivation**

Quantitative trading is a highly lucrative industry, but in reality, it is not easy to buy low and sell high. With an influx of retail investors, markets are more volatile than ever, and even seasoned traders cannot generate profits reliably. These days, mathematical and statistical models, machine learning and artificial intelligence are employed to make rapid trading decisions that seek to beat the stock market and provide investment returns in the millions of dollars.

Predictive machine learning models based on data are important to ensure consistent gains even in a volatile and unpredictable environment, by identifying hidden market patterns, cycles and seasonalities that humans cannot. As such, we seek to explore the use of machine learning to predict buy/pass actions on securities, and experimenting with various traditional and deep learning models to see how good it can become in generating profits.

**Personal Motivations**

Quantitative trading jobs easily pay 15k-20k USD, and we have interest in picking up some experience in this field.

**Statement of the Problem/Task**

We aim to, using machine learning techniques, create a quantitative trading model that maximises profits by correctly selecting buy/pass actions. In the scope of the Kaggle dataset, we aim for our model to achieve the perfect utility (profitability) score.

Building up to our goal, we aim to explore the following secondary goals:

* To reduce the feature space of our dataset through data analysis and dimensionality reduction to discover a subset of features that represent the most salient aspects of the data
* To explore if deep learning techniques outperform traditional machine learning in creating a quantitative trading model

### **3. Related Work**

Hiransha M et al [2] attempts to use artificial neural networks (ANNs) to predict financial data. They note that it is difficult to predict accurately for individual stocks as they are affected by market volatility and other external factors. However, they note that deep learning models tend to perform better than linear models as they are able to predict deviations from linearity and short term swings.

There have been many other attempts at using deep learning at predicting the markets, such as those from Yetis et al [3] and Jabin S [4], but little have compared deep learning models directly with traditional learning models to determine which is superior in predicting the markets. As such, we would like to conduct an experiment on both kinds of models and evaluate them on the same dataset.

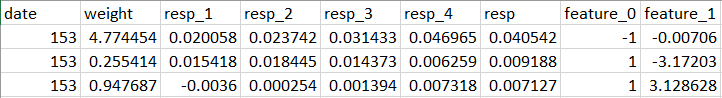
An exploration of the Kaggle competition page [5] clearly shows a heavy bias towards employing neural networks and deep learning models from libraries such as Tensorflow or Pytorch. However, none of the submissions investigate and demonstrate the difference between deep learning and traditional machine learning. While it may seem intuitive that neural networks are predisposed towards solving quantitative trading problems (Jingtao et al) [6] due to the high dimensional complexity of the input features, we find it worthwhile to still carry out an investigation into traditional machine learning techniques to explore all available options.

### **4. Method**

This report shall only address the final methodology employed, following feedback from the interim presentation. We center our approach around the two groups of models - traditional machine learning models, and deep learning models.

**Step 1: Data Inspection**

Basic data inspection and exploration was first carried out to inspect the data provided by the Jane Street Market Prediction competition dataset.

**

*Fig 1: Some columns of our dataset.*

Data was observed to consist of:

**date:** Date of investment transaction

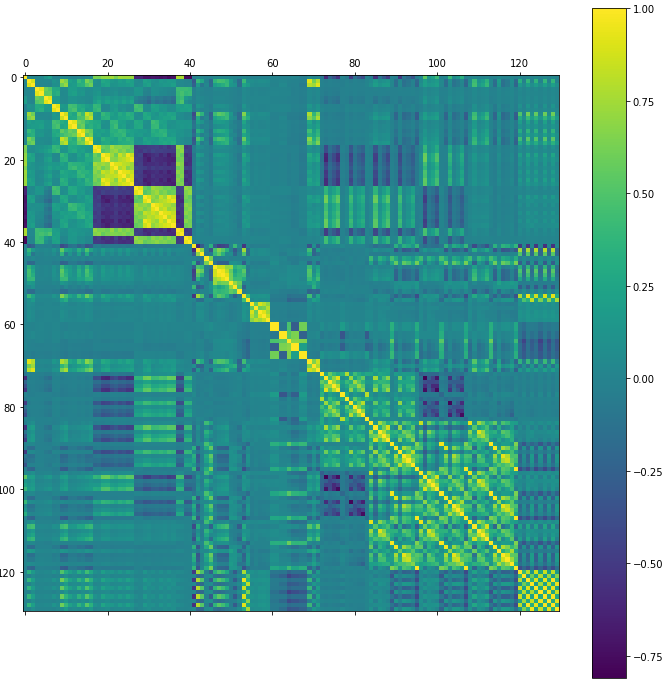
**weight:** Portfolio weight of an investment

**resp:** Return on investment

**resp1-resp4:** Alternative ROIs not used for evaluation

**features 0-129:** Anonymized features of that particular investment which will be the input variables in training

Of particular note is the large number of features, which are anonymized. Beyond mild correlation among some features, we could not derive more value from them through a qualitative analysis due to the redacted features. Conventionally, screening of input features is necessary in order to employ domain knowledge to remove unnecessary and potentially detrimental features from the dataset.



*Fig 2: Correlation matrix of our dataset’s features*

We observe from the correlation matrix regions of high correlation (bright yellow, dark blue, exclude diagonal). This prompts us to explore the use of feature elimination through Principal Component Analysis to remove unwanted dimensions from our training data. This will reduce computational costs subsequently as we train with fewer features.

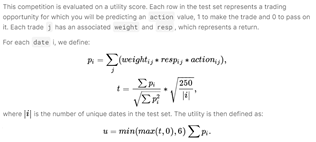
**Step 2: Data Preparation**

Data cleaning was then carried out. Given the large dataset of 6GB, there will still be sufficient data even after removing NA values, thus we avoid introducing noise by not filling in the NA values with values. Rows with weight=0 were removed as well. Lastly, an ‘action’ column was introduced to represent the buy/pass action (binary classification) for each transaction by returning 1 for a buy action if weight\*resp is positive, else 0.

As the original total dataset is 6gb large, we sliced the dataset to the following : 1gb total (700mb train set and 300mb test set). Lastly, we save the respective slices into a train and test csv for subsequent learning. Since this is a time series dataset, we ensured that the test set occurred at least a few days after the train set to avoid temporal elements bleeding through the train to the test set.

**Step 3: Create Evaluation Metric**

The evaluation metric provided by the competition is as follows:



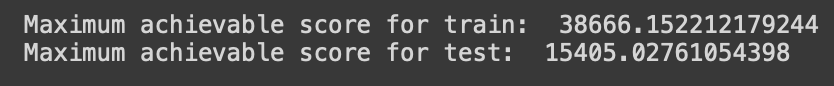
*Fig 3: Utility Score - competition’s evaluation metric*

The main metric for evaluating each machine learning model will thus be the utility score, calculated by a function of weight \* resp \* action. It is the significance of the opportunity multiplied by the returns of that opportunity and whether you took that action. A pass action (0) would result in all utility for that opportunity being lost.

It is worth noting that utility score is more important than accuracy - a model that only predicts all pass actions is highly accurate but would achieve a utility score of 0, indicative of 0 profits. Hence, to maximise profits, we aim to maximise our utility score.

**Step 4: Setting Benchmark**

To benchmark our model performances, we also calculated the maximum attainable utility score for both train and test.



*Fig 4: Maximum achievable scores*

To determine if a model was actually better than just not

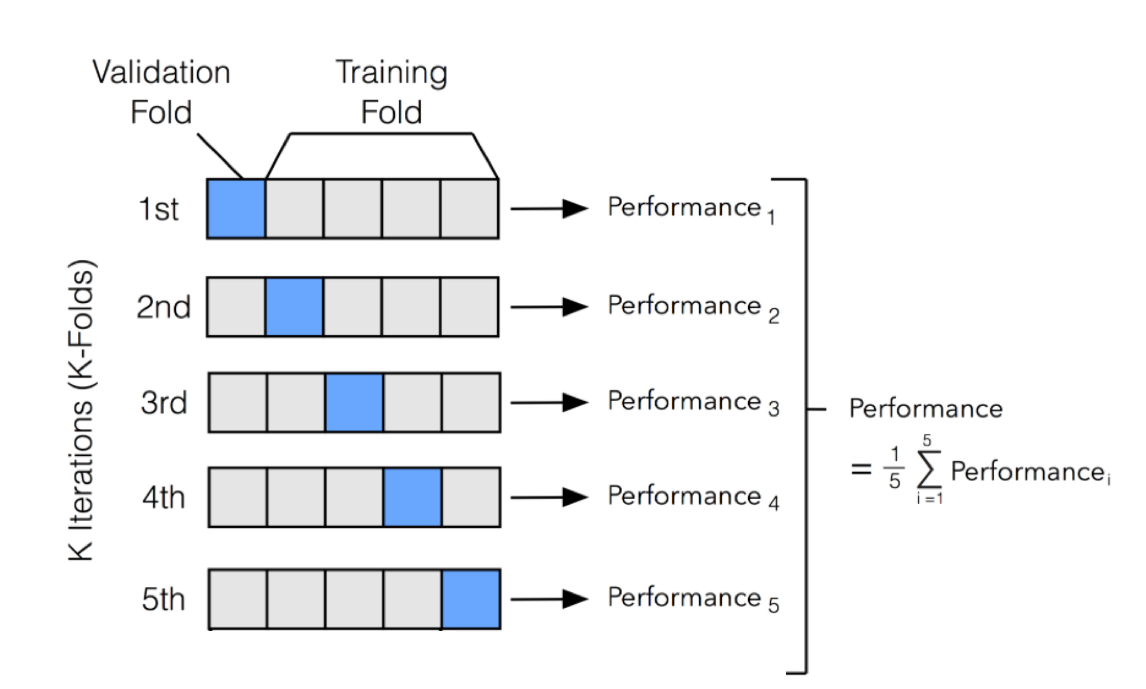
using a model at all, we further set up a model that simply has a 50-50 random chance of choosing to buy or pass a trade. The scores will be illustrated later.

**Step 5: Employing Traditional Machine Learning**

We experimented with various traditional machine learning models perform binary classification (buy or pass)

* Linear Regression (LR)
* Naive Bayes (NB)
* Decision Trees Classifier (DT)
* Random Forest Classifier (RF)
* Support Vector Machines (SVM)

K-fold cross validation was also implemented (k=10, a typically accepted value from literature reviews) to access the various model performances as it returns an average across 10 training folds for a more consistent result. More importantly, given the various hyperparameters available for each model, k-fold cross validation allows us to perform hyperparameter tuning.



*Fig 5: Illustration of K-Fold Cross Validation*

*Diagram Credit:* [*ethan8181 @ Github*](http://ethen8181.github.io/machine-learning/model_selection/model_selection.html)

For each machine learning model, we varied the relevant hyperparameters iteratively, logging the utility score produced on the *cross validation* set.

Models were first trained and evaluated without employing PCA on the dataset. We subsequently repeated the experiments by applying PCA on the dataset and varying the number of components to be reduced to. This allowed us to determine if PCA was important and if so, determine the most important number of salient features.

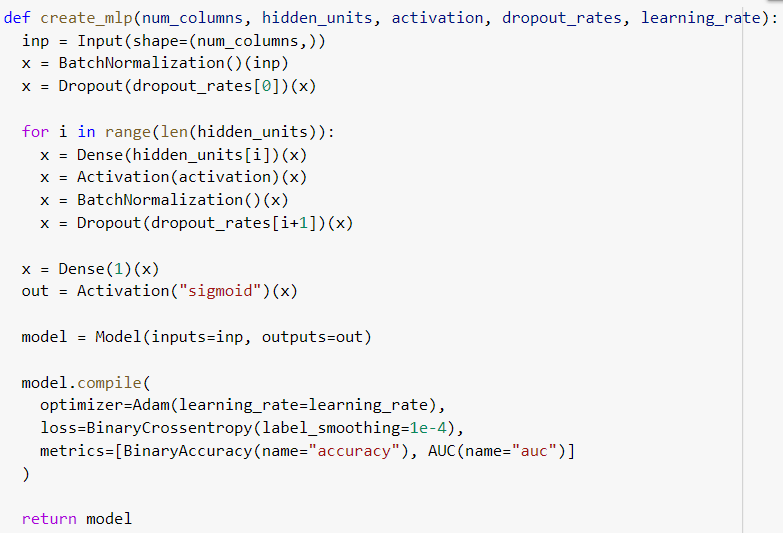
|  |  |
| --- | --- |
| **Model** | **Hyperparameter** |
| PCA | n\_components = [70,90,110,130] |
| LR | solver = [liblinear, sag, saga]  iterations = [100,1000,10000] |
| NB | var\_smoothing=[1, 1x10-3, 1x10-6, 1x10-9, 1x10-12] |
| DT | max\_depth = [2,4,8,16,32,64]  splitter = [best, random]  max\_feature = [None, sqrt, log2] |
| RF | n\_estimators = [10,30,50]  max\_depth = [2,4,8,16]  max\_feature = [None, sqrt, log2] |
| SVM | iteration = [100, 500, 1000, 10000] |

*Table 1: Hyperparameters swept for Traditional Machine Learning Models*

**Step 6: Employing Deep Learning**

We also experimented with various neural networks and hyperparameter tuning. There are countless deep learning models and architectures that we can create to perform binary classification. Due to a lack of time and resources to exhaustively sweep all potential models, we decided to employ a simple Multi Layer Perceptron [7] model as learnt in the course and focus instead on understanding and performing hyperparameter tuning [8]. We implemented a simple MLP with 4 hidden layers, and optimized its hyperparameters.

PCA was also not implemented with our neural networks given our findings from traditional machine learning which will be discussed further in evaluations. Also, neural networks are well suited to handling and learning from high-dimensional data, hence the use of PCA may not be necessary.



*Fig 6: Implementation of MLP neural network*

We introduce two versions of our MLP

* MLP1 - as pictured in Fig 6
* MLP2 - no dropout applied on input layer

The following table details the hyperparameters swept:

|  |  |
| --- | --- |
| **Models** | MLP1, MLP2 |
| **Epochs** | 50, 100, 400 |
| **Batch Sizes** | 256, 512, 1024, 2048 |
| **Hidden Units** | [50, 100, 200, 100], [100, 200, 400, 200], [200, 400, 800, 400], [400, 800, 1600, 800] |
| **Dropout Rates** | [0, 0, 0, 0, 0], [0.2, 0.2, 0.2, 0.2, 0.2],  [0.4, 0.4, 0.4, 0.4, 0.4] for MLP1,  [0, 0, 0, 0], [0.2, 0.2, 0.2, 0.2],  [0.4, 0.4, 0.4, 0.4] for MLP2 |
| **Activations** | tanh, relu, sigmoid |
| **Learning Rate** | 1x10-2, 1x10-4 1x1064 |

*Table 2: Hyperparameters swept for Deep Learning Models*

10-fold cross validation was implemented on the deep learning models in the same manner with traditional machine learning models to ensure that we have an identical basis of comparison for scores across the two classes of models.

### **5. Training and Analysis**

**5A. Baseline - RNG model**

To set a baseline for comparison, we employed RNG (random number generator). The RNG model randomly chooses a value of 0 or 1 as the classification for a given training data instance.

The majority of traders are akin to random guessers. RNG simulates someone making decisions with a 50-50 chance of winning and losing. In general, over the past few decades, markets have been on an uptrend, hence a random decision maker will also stand to gain over time (with the assumption of similar wins and losses). As such, a model should at least aim to beat a random decision maker.

Our RNG model, over a 100 random sets of predictions, returns an average utility score of **662.6082534** and an accuracy of **0.500279454**.

**5B. Traditional Machine Learning Models**

Observations for hyperparameters sweep for TML are as follows:

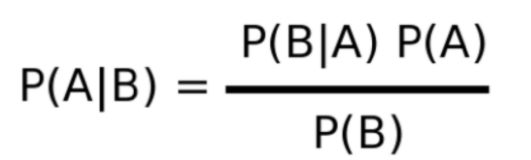
**Logistic Regression**

Logistic regression is a simple model for a binary dependent variable, classifying training instances into a positive and negative class based on a logit function and threshold value.

No particular trend was observed and difference in performances between models of different hyperparameters was minimal. This observation was consistent with and without PCA. No conclusion can be drawn for pca\_components.

**Naive Bayes**

Utilizes the Bayes Theorem, a probabilistic approach towards classification.



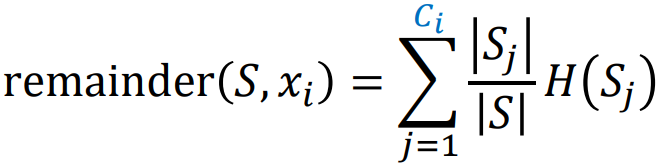
*Fig 7: Naive Bayes Algorithm*

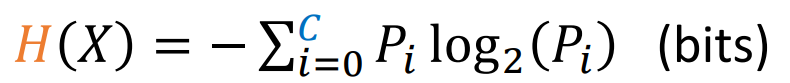
Utility scores increase with increasing var\_smoothing although performance improvements were negligible. This observation was consistent with and without PCA. Additionally, with PCA implemented, it was observed that pca\_components = 110 performs the best.

**Decision Tree**

Essentially a tree of conditions that training instances are passed through, with each condition designed to split the training data in a way that maximizes the information gain. This uses the concept of entropy, loosely the amount of information.







*Fig 8: Information Gain and Entropy Algorithm*

We observed a trend of increasing utility score with max\_depth and the hyperparameters splitter = best and max\_feature = None performed the best. This observation was consistent with and without PCA. No conclusion can be drawn for pca\_components.

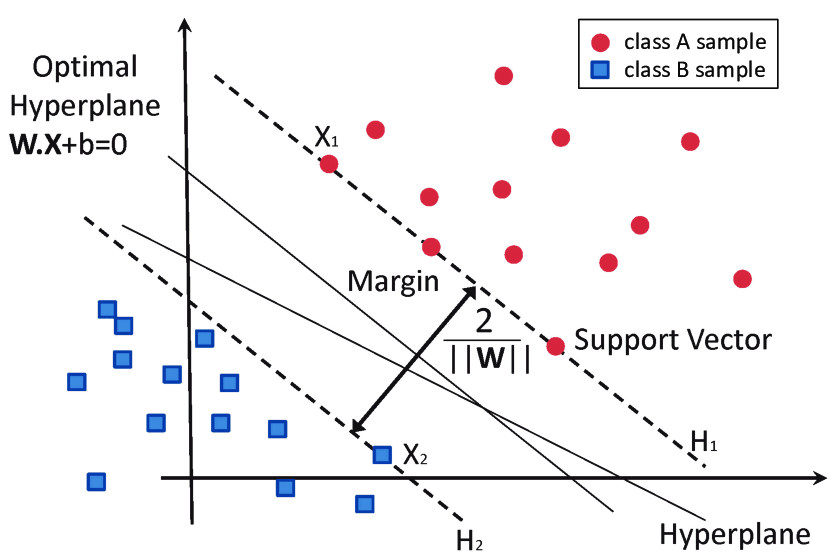
**Random Forest**

Creating a ‘forest’ of decision trees, and combining bagging with decision trees in order to achieve a more diverse model where decisions are made through voting on consensus across different decision tree models.

We observed that random forest with max\_depth = 16 performs the best. Utility scores generally increase with max\_depth. This observation was consistent with and without PCA. No conclusion can be drawn for pca\_components. However, random forest performs worse with PCA implemented.

**Support Vector Machines**

SVM separates the classes by finding a hyperplane. Support vectors are those closest to the hyperplane that will change the hyperplane if removed. SVM optimizes the margin of separation between support vectors, and points falling on either side of the hyperplane will all belong to one class.



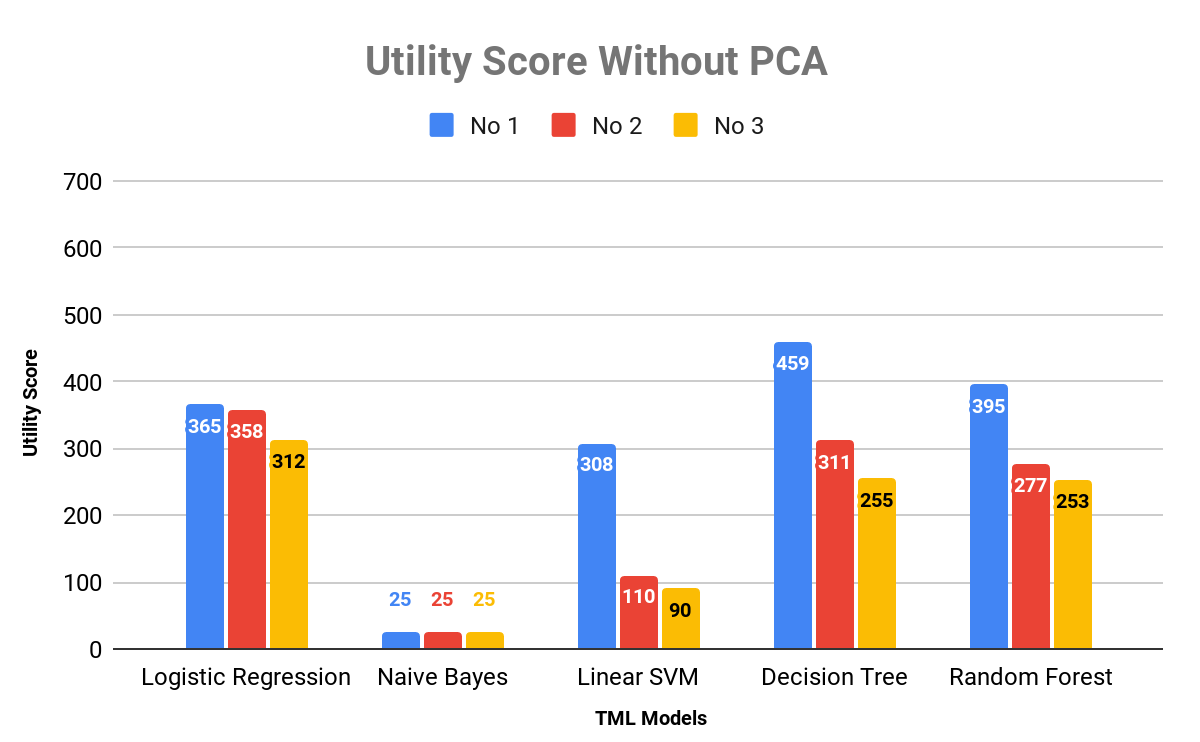
*Fig 9: Illustration of SVM*

*Diagram Credit:* [*ResearchGate*](https://www.researchgate.net/figure/Classification-of-data-by-support-vector-machine-SVM_fig8_304611323)

We observe a trend of increasing utility scores with increasing iterations. This observation was consistent with and without PCA. No conclusion can be drawn for pca\_components.

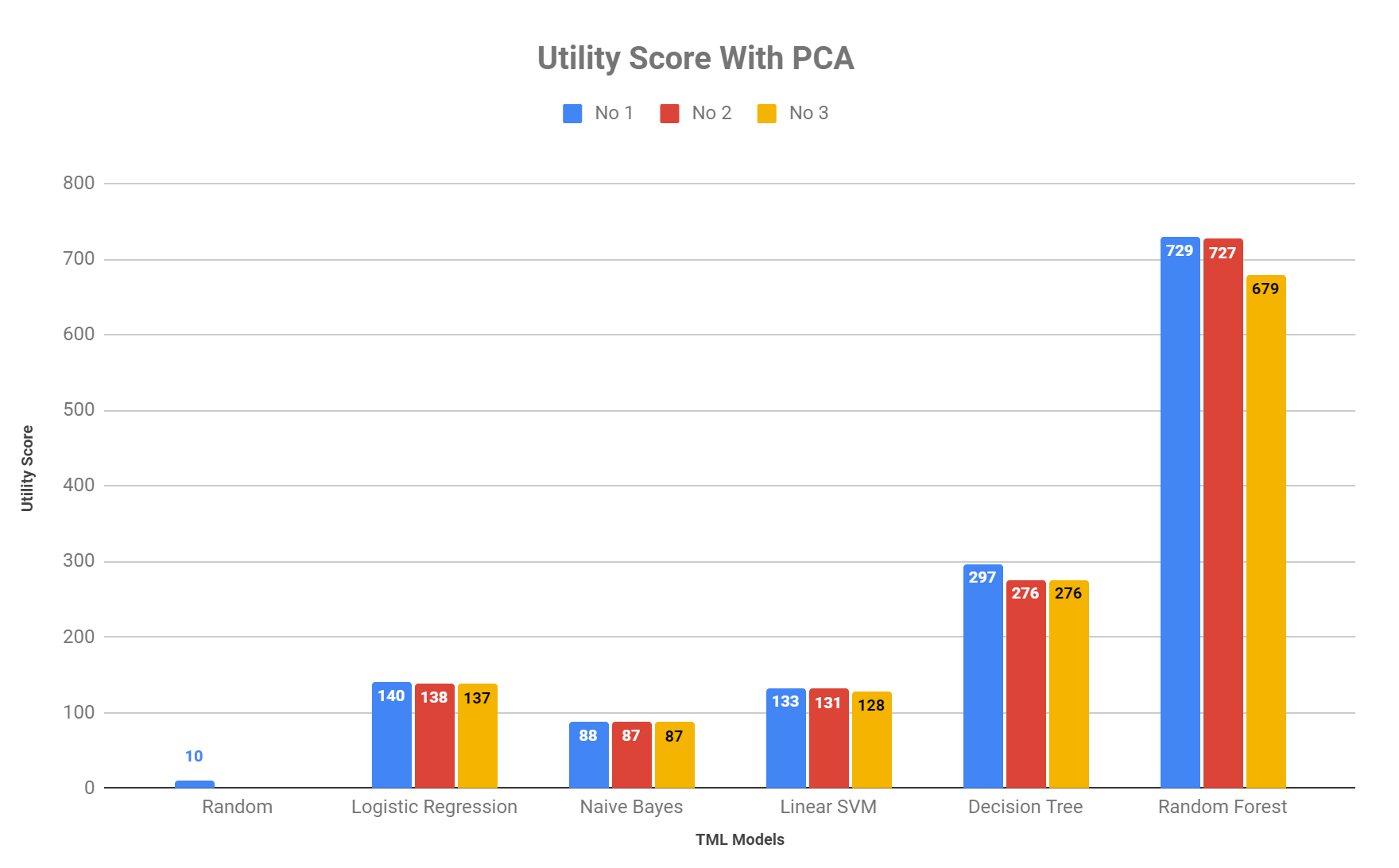
**Top TML Models**

For brevity, we will only analyze the top three models for every model type. For traditional machine learning, our results are as follows:



*Fig 10: TML utility scores without PCA*

Without PCA, Random Forest performed the best on the test set, resulting in an almost 1700 utility score for the top 3 hyperparameter sets. Of course, out of the theoretical maximum of 15405, this does not appear to get significant profits, but it does do better than the RNG mode (utility score of 10), which helps to confirm the theory that the markets can be modelled.

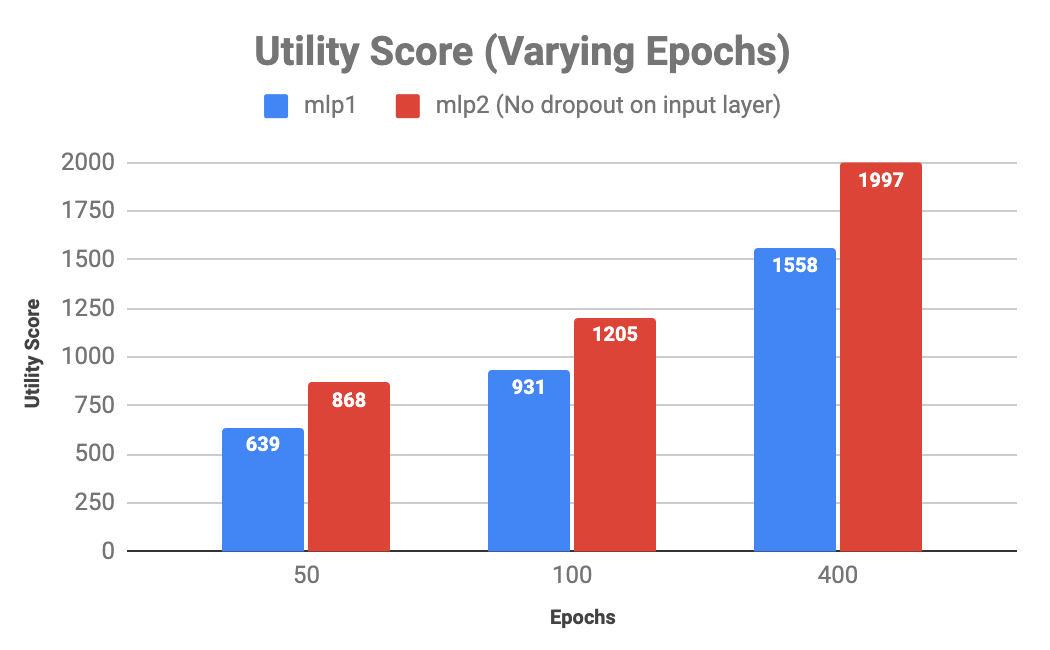


*Fig 11: TML utility scores with PCA*

We tried training the traditional learning models without PCA. Strangely, the models trained after PCA performed worse, with a max utility score of only around 700. This could mean that the features are individually important towards the action decided.

**5C. Deep Learning Models**

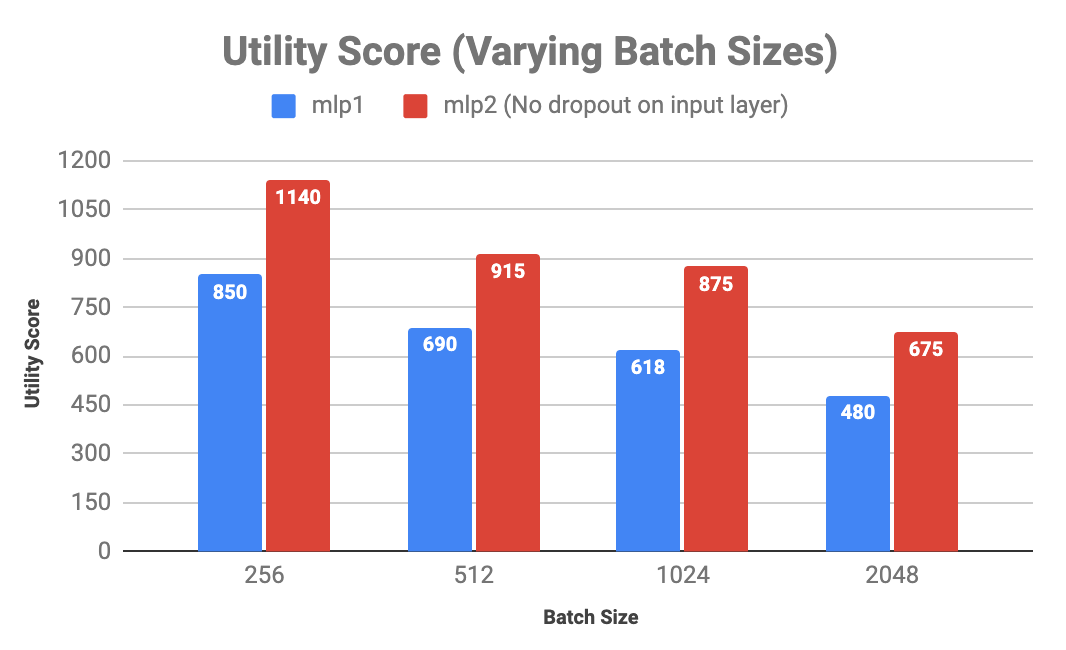
We tuned the hyperparameters sequentially for the models MLP1, MLP2, starting with the number of epochs.



*Fig 12: MLP tuning of epochs*

Theory - More epochs allow the model to achieve convergence, and therefore reach its maximum learning ability. Scores are expected to increase with epochs.

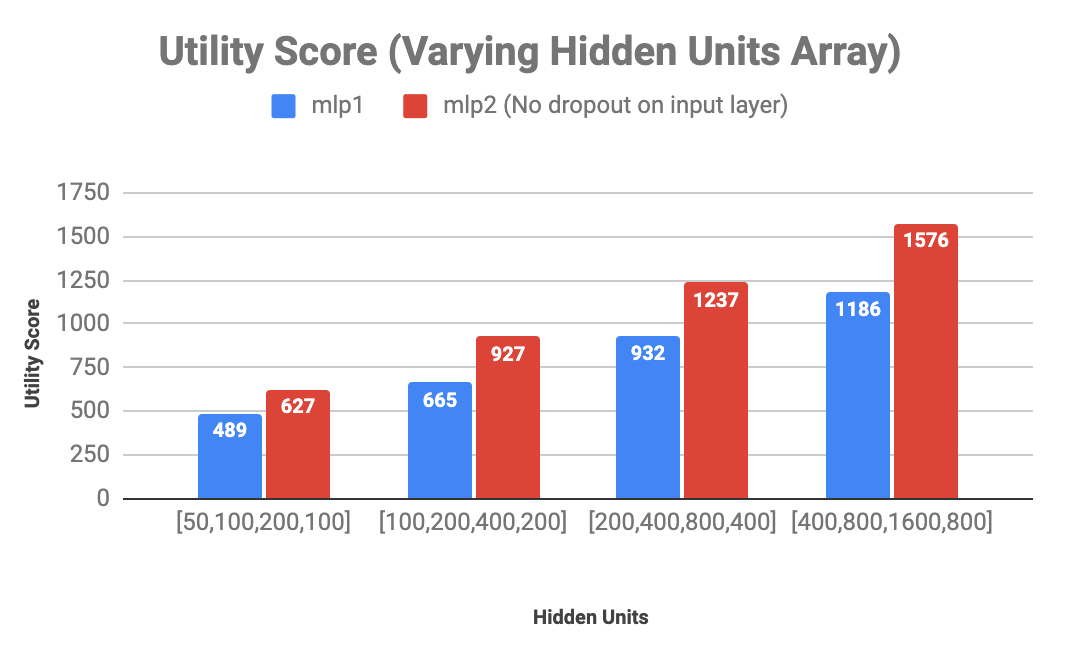
We observe that our model experienced *increasing* utility scores as the number of epochs *increases.* It is worth noting that convergence was not achieved even at 400 epochs. In interest of time, the models were trained on 50 epochs instead. However, we will aim to train till convergence for the test set.



*Fig 13: MLP tuning of batch sizes*

Theory - Small batch sizes [9] are expected to provide a small regularizing effect due to the noise in frequently updating gradients. However, they are comparatively slower to train due to more frequent updates to gradients.

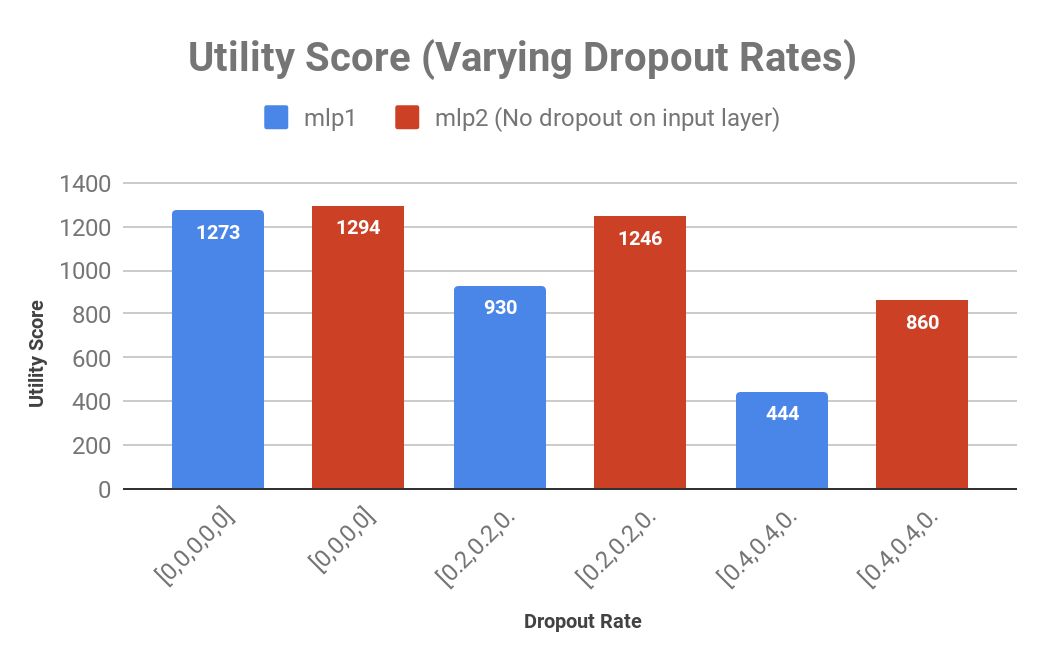
We observe that our model experienced decreasing utility scores as the batch size *increases*. Therefore, we stuck to a small batch size of 256 as our batch size for the test set.



*Fig 14: MLP tuning of hidden units array*

Theory - The number of hidden units in a MLP affects its ability to learn new features. [11] The more units, the more features will be learned. However, the risk of overfitting also increases.

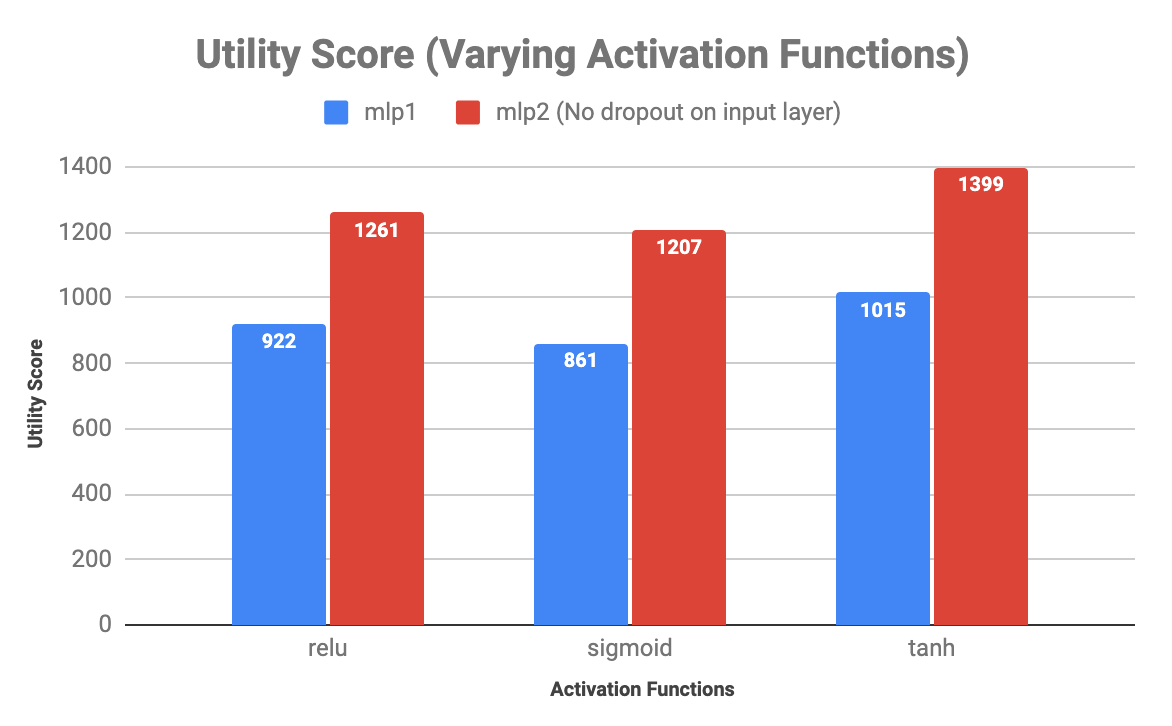
We observe that our model experiences *increasing* utility scores as the size of hidden units in the hidden units array increases. Therefore we selected the hidden unit array [400,800,1600,800] for the test set.



*Fig 15: MLP tuning of dropout rates*

Theory - Dropout randomly kills nodes within a layer and weights learned at those neurons become invalid and no longer contribute to the model. [10] This improves regularisation as it makes the training process noisy, forcing nodes within a layer to probabilistically take on responsibility for the inputs. However, too high a dropout rate will affect the ability of the model to learn.

We observe that our model experiences *decreasing* utility scores with *increasing*  dropout rates. Furthermore, the decrease in utility score is lower for MLP2, indicating that utility score is higher without dropout in the input layer. Therefore, we selected the dropout rate of [0.2,0.2,0.2,0.2] for the test set.



*Fig 16: MLP tuning of activation functions*

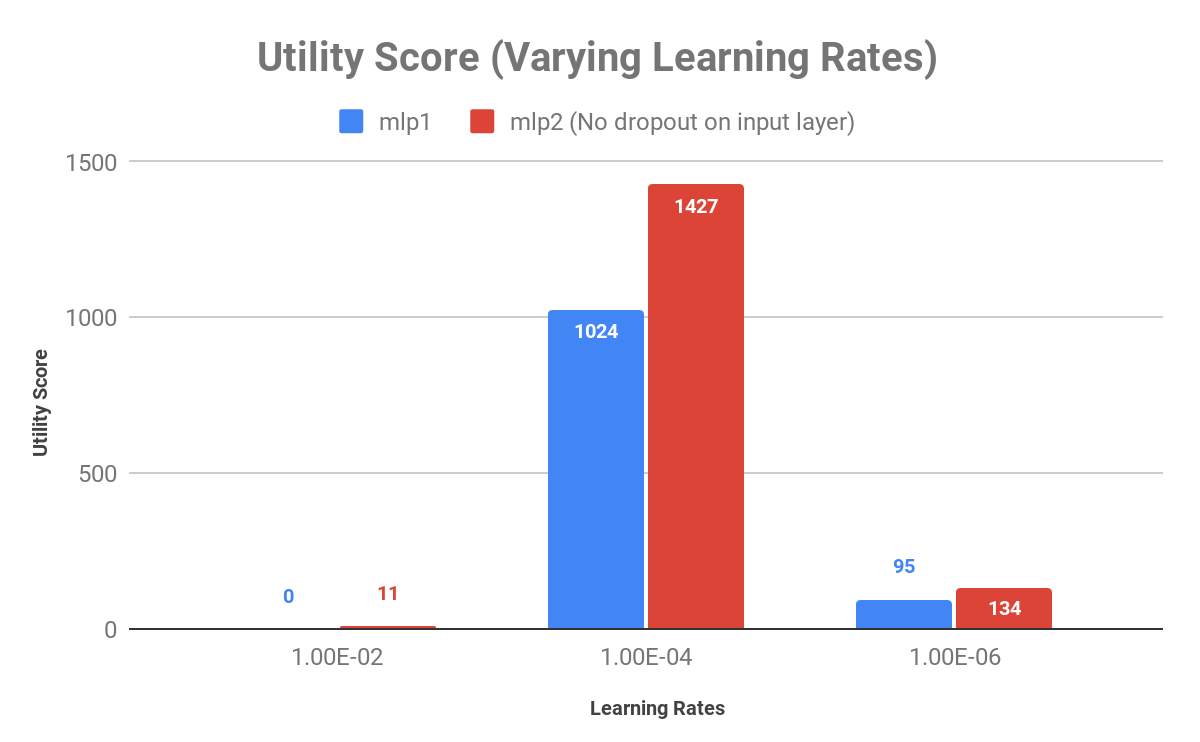
Theory - Different activations provide non-linearity. [12]

ReLU: Avoids vanishing gradient, piecewise function from 0 to 1

Sigmoid: ranges from 0 to 1

Tanh: ranges from -1 to 1

We observe that our model returns the highest utility score when tanh is used without much time tradeoffs. Therefore, we will use tanh as the activation function for the test set.



*Fig 17: MLP tuning of learning rates*

Theory: The optimal learning rate allows us to find the local minimum through gradient descent in the shortest time. Low learning rates increase time required for convergence significantly, high learning rates cause erratic learning that might never converge. [13]

We discovered that for rates higher or lower than 1x10-4, the utility score diminishes significantly. Therefore, we used a learning rate of 1x10-4for future models.

### **6. Final Evaluation**

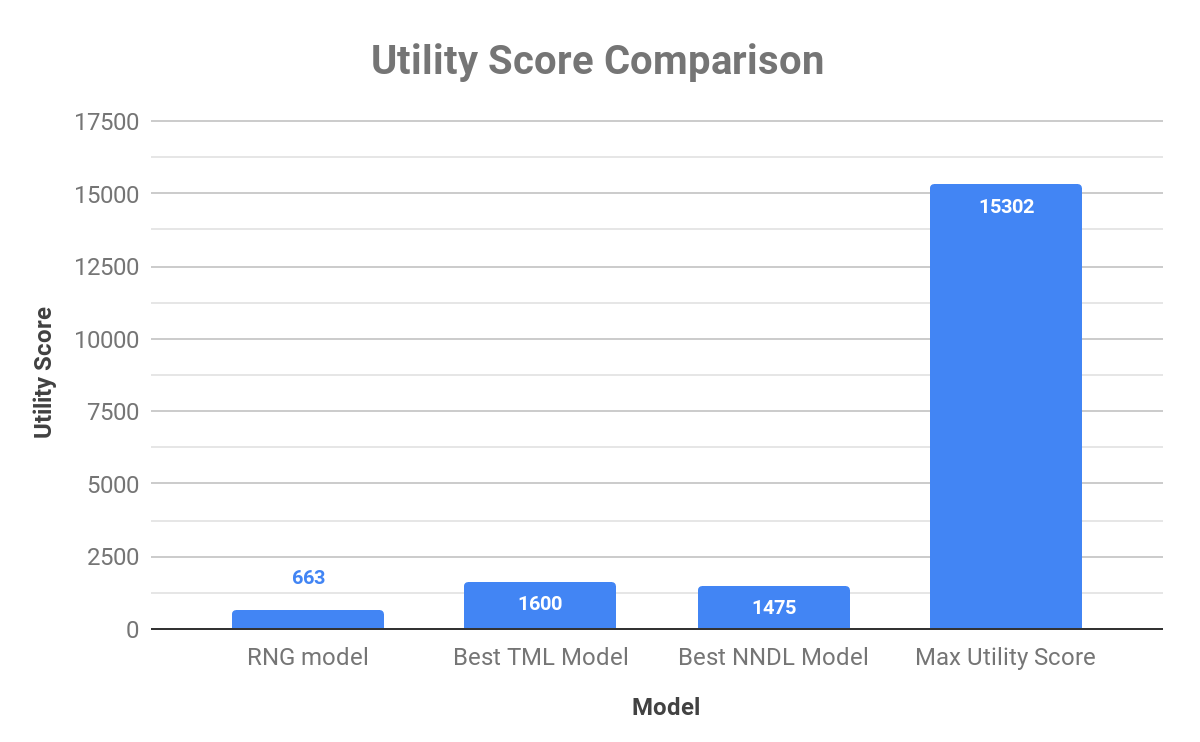
Based on cross validation scores, our highest scoring traditional machine learning model is Random Forest without PCA, with model parameters:

* n\_estimators = **50**
* max\_depth = **16**
* max\_feature = **None**

Our expected highest scoring deep learning model will be the combination model from our experiments:

* Model = MLP2 (no dropout on input layer)
* Batch Size = **256**
* Hidden Units = **[400,800,1600,800]**
* Dropout Rates = **[0.2,0.2,0.2,0.2]**
* Activation Function = **tanh**
* Learning Rate = **1e-4**

We retrain these models without cross validation on the full training data, and perform an evaluation on the unseen test set.



*Fig 18: Performance of best models on test set*

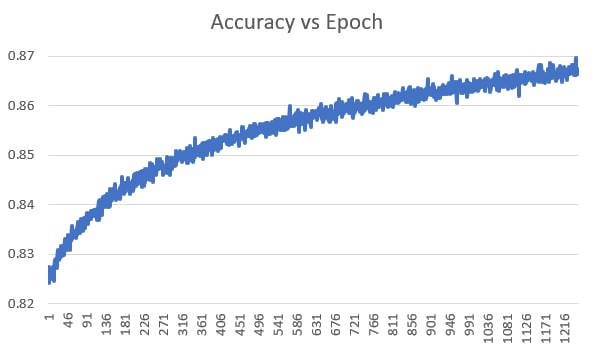
|  |  |
| --- | --- |
| **Best Model** | **Utility Score** |
| RNG model | 662.6082534 |
| TML model (RF no PCA) | 1599.608341 |
| NNDL model (MLP2) | 1474.707667 |
| Maximum achievable (test) | 15302 |

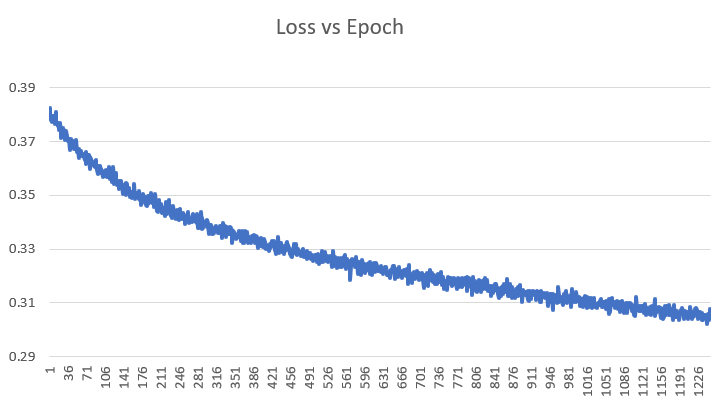
*Table 3: Performance of all models*

From Table 3, our models were unable to produce a utility score close to the maximum score. While both TML and NNDL outperformed the RNG model, NNDL failed to outperform TML.

### **7. Discussion**

For intensive classification problems with a lot of features (especially anonymized), conventional wisdom and practice heavily advocates for the use of neural networks. However, in our case, traditional machine learning models outperformed our deep learning model. A possible explanation for this would be the lack of convergence in our deep learning model. When we plot out the loss for our best NNDL model, we notice that we have yet to achieve convergence.





*Fig 19: Accuracy and Loss vs epoch for best NNDL model*

**Failure to reach convergence**

We exhausted our computing resources and time on our platform of choice, Google Colab. However it was prohibitively expensive to run on our local machines due to the lack of dedicated GPUs to accelerate the deep learning process. As we were unable to fully train the model, the failure to converge results in possibly inaccurate results, which could mean our deep learning model did not perform to its full potential, and thus cannot be compared directly to the traditional machine learning models.

Furthermore, due to the lengthy training time required (400 epochs, 4 hours), we could not train our models till convergence when stepping through our hyperparameter tuning for NNDL. Therefore, our results and conclusions regarding the various optimal hyperparameters may not be fully reliable as well.

**Exhaustive search impossible**

Our space of hyperparameters searched were typical values obtained from various machine learning related articles. It is prohibitively expensive to perform exhaustive searches over the entire possible hyperparameter space, and hence the optimal values may not have been considered at all. This further contributes to the reason why our NNDL model performed worse than our TML model.

**PCA not helpful**

We expected PCA to help in reducing the dimensionality of data, but it had the adverse effect in this case. Usually, with high dimensionality, the “curse of dimensionality” occurs if there are a lot of features but not enough data, resulting in overfitting. In this case however, we had more than enough data. As such, training on the full feature set gave better results in our case. There is no way for us to evaluate the validity of this PCA conclusion due to the anonymized feature set preventing us from drawing qualitative conclusions.

### **8. Conclusion**

While we have indeed been able to create a quantitative trading model that is able to return substantial profits by correctly selecting buy/pass actions, we realise that it is very difficult to maximise profits ideally to return the perfect utility score.

Market volatility is subject to many real world factors like news, earning reports etc. All of these can be difficult to model since they are external factors and are akin to stochastic noise. Nevertheless, our model is able to confidently beat the RNG baseline and rake in reliable profits over time, which signifies the possibility of an automated trading system.

In particular, we managed to show to a certain extent that the majority of traditional machine learning models do not perform as well as deep learning ones, and given enough time and training for convergence, deep learning ones should go on to outperform the best traditional models.

**Shortcomings**

While we theorized that deep learning would perform much better than traditional machine learning techniques, we were unable to achieve convergence for our deep learning models even after running for many hours - leading to timeout on Google Colab. Resource availability is a big factor when trying to perform deep learning.

**Future Work**

Despite the challenges, we managed to develop a model able to return a utility score of 10% of the maximum utility score. In comparison, the top ranked submissions for the competition returned approximately 30% of the maximum utility score. Given more time to reach convergence and more computing power as well as exploration of other techniques like ensembles, we foresee a much better performance.

### **9. Acknowledgements**

We would like to thank Assoc. Prof Stefan Winkler and TA Calvin Chen Xingzhu for the feedback and guidance throughout this project.

### 

### **10. References**

[[1]](https://www.kaggle.com/c/jane-street-market-prediction/overview) Jane Street Market Prediction Competition (2021), Kaggle.

[[2]](https://pdf.sciencedirectassets.com/280203/1-s2.0-S1877050918X00088/1-s2.0-S1877050918307828/main.pdf?X-Amz-Security-Token=IQoJb3JpZ2luX2VjEFYaCXVzLWVhc3QtMSJHMEUCIGNVMmMx3pXyJWGv8tJAGUWAO1qpPwIC%2FHbRWpvxP0udAiEAn8I7c4prhCI2RDwuKquOlgzXyPjXOk0Iq2yBN7DTKaUqvQMIv%2F%2F%2F%2F%2F%2F%2F%2F%2F%2F%2FARADGgwwNTkwMDM1NDY4NjUiDD%2B2gTgwQQniE%2FkGASqRA0WOQgtTg%2FFlt4Vc2UURmhANzktHKXZWaYE2K%2BCpfu8pmKPkDkPLCnjNmdoVp0lwxpldwRT3d28Ed8Nmr%2FcGAiz6QI7ER6g9LooAgIenwjbU8YYjT0d1m2F%2FWNVwpQdTa13LFe9Ldje%2Bu0Td0CUN4gFQcpusPI6Mw1fY46jcYKRMlSIIfJASfVIJgs97Rz8WwwyTowz5%2B%2FaDgUhFglTGYnohzZH0EfnmWeKVs5acOvgFcpN%2B76WwYD1TrLtfhYJ7UoYk0Ye9ODaPxFpqiptZXu3D%2BkR8bPiDn4FiGQBqPAHMCvxy3vB6eCsoMK9scVJ3%2FJKAA3Ed0A%2B0zObxnuaGKnegat%2BPL9wTm2dS4vQFG1tsTRq0YLakmWX1f2uibX%2FtC944CscwZZJtbXEJlxHtY8F8s2e3L4laOARHW5ERnHA4jRhp4BWEtl5dWahzsHqAnTcn6CgsJ99YK511NquE9cxJugYfWF668et433eqhxk%2F%2FQZ0QU6MICbCuntM5IFkBVqfHXdEdKxDlxjROnvmtOw1MKKAhoQGOusBAXuwRFG%2BtNyMW%2BhSxmxo1RSjrBq%2F%2B%2FR2tvNff51GDh4IezHExGjRRHUZOhaeg02bQtsJtPukWkD3Xg2VlJLn0gbXPjY0R2c1HNrXeGhKsLjG0huX%2B16BwbH4wP3EWV3I7lJJoZPOLtq2vHIDZaW6gAXCyDslzDmg%2FKCJddPnG1hr%2BIaV3RDyJU1nS2gmKjdIAJGPyCUB9ODJhh0pB3QMtO9n5%2BmwnsaRwqNVgye7o8xI%2B%2BJqxKkVHBnOVqSgHAuJC6FFrUKruMRt4oIA70JHanDFoH%2BN7qg4wUp2RZdBn5%2FKyaFhSrHmp%2FAdHg%3D%3D&X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Date=20210422T145052Z&X-Amz-SignedHeaders=host&X-Amz-Expires=300&X-Amz-Credential=ASIAQ3PHCVTYX52WUGNT%2F20210422%2Fus-east-1%2Fs3%2Faws4_request&X-Amz-Signature=afef7a9991d94d5cbbeb8825a3a189ea391e0f0285b98bc156cd4609d0dc5aed&hash=cc702f05f147690818fd3e48aae9a9631d35a1d52a86b967788c83f93c164f23&host=68042c943591013ac2b2430a89b270f6af2c76d8dfd086a07176afe7c76c2c61&pii=S1877050918307828&tid=spdf-34f544be-580d-448f-8ccf-cbb033861ae7&sid=0aeebfb88c43b747242a49b83bdabb2e422fgxrqb&type=client) Hiransha Ma, Gopalakrishnan E.Ab , Vijay Krishna Menonab, Soman K.P (2018) *NSE Stock Market Prediction Using Deep-Learning Models*, ScienceDirect.

[[3]](https://ieeexplore.ieee.org/document/6936118) Yetis Y., Kaplan H., and Jamshidi M. (2014). *Stock market prediction by using artificial neural networks,* In World Automation Congress (WAC) :718-722.

[[4]](https://www.researchgate.net/publication/268152880_Stock_Market_Prediction_using_Feed-forward_Artificial_Neural_Network) Jabin S. (2014). ”Stock market prediction using feed-forward artificial neural network”. growth 99 (9).

[[5]](https://www.kaggle.com/c/jane-street-market-prediction/code) Jane Street Market Prediction Code Submissions (2021).

[[6]](http://www2.cs.uregina.ca/~jtyao/Papers/guide_iconip01.pdf) Yao, Jingtao & North, Palmerston & Tan, Chew Lim. (2002). Guidelines for Financial Forecasting with Neural Networks, ResearchGate

[[7]](https://machinelearningmastery.com/build-multi-layer-perceptron-neural-network-models-keras/) Brownlee, J. (2016, May 19) *How To Build Multi-Layer Perceptron Neural Network Models with Keras.* Machine Learning Mastery.

[[8]](https://datascience.stackexchange.com/questions/36049/how-to-adjust-the-hyperparameters-of-mlp-classifier-to-get-more-perfect-performa) *How to adjust the hyperparameters of MLP classifier to get more perfect performance*. Data Science Stack Exchange. (2018, July 23)

[[9]](https://machinelearningmastery.com/how-to-control-the-speed-and-stability-of-training-neural-networks-with-gradient-descent-batch-size/) Brownlee, J. (2020, August 27). *How to Control the Stability of Training Neural Networks With the Batch Size*. Machine Learning Mastery.

[[10]](https://machinelearningmastery.com/dropout-for-regularizing-deep-neural-networks/) Brownlee, J. (2018, December 3) *A Gentle Introduction to Dropout for Regularizing Deep Neural Networks.* Machine Learning Mastery.

[[11]](https://medium.com/swlh/hidden-units-in-neural-networks-b6a79b299a52) Batsuuri, J. (2020, May 7). *Hidden Units in Neural Networks*. Medium.

[[12]](https://machinelearningmastery.com/choose-an-activation-function-for-deep-learning/#:~:text=for%20Output%20Layers-,Activation%20Functions,a%20layer%20of%20the%20network.) Brownlee, J. (2021, January 8) *How to Choose an Activation Function for Deep Learning.* Machine Learning Mastery.

[[13]](https://machinelearningmastery.com/learning-rate-for-deep-learning-neural-networks/) Brownlee, J. (2019, January 3) *How to Configure the Learning Rate When Training Deep Learning Neural Networks.* Machine Learning Mastery.

### **11. Appendix**

### NA