AI Approach Algorithm Selection Report

1 Introduction

1.1 Meta Information

Assignment: Practical 4

Submission Date: April 24th, 2023

ID: 220011642

Word Count: 1968

1.2 Checklist of Implemented Parts

· Part 1: Completed, fully working

· Part 2 Basic: Completed, fully working

· Part 2 Advanced: Completed, fully working

· Part 3 Extension 1: Completed, fully working

· Part 3 Extension 2: Completed, fully working

1.3 Compiling and Running Instructions

- The execution of the program is designed as the spec required.
- To run the code, simply navigate under /src and execute the `install.sh` to first set up the environment and install the following Python libraries:

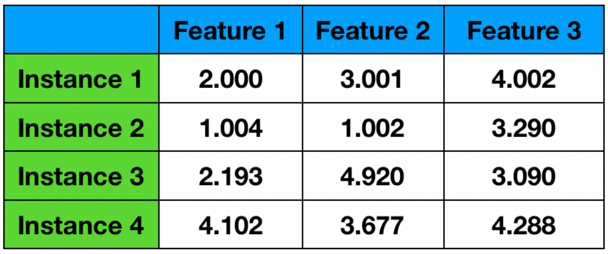
`numpy, torch, matplotlib, skorch, scikit-learn, joblib`

- Then, execute the `run.sh` and all three parts will run in order automatically

2 Design & Implementation

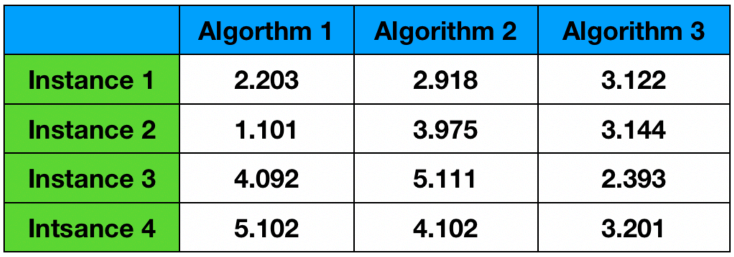
2.2 Problem Definition

This practical aims to tackle a simplified version of Algorithm Selection problem by training different artificial neural networks. We have a dataset that contains the feature vectors of many instances, and another dataset containing the costs of different algorithms on solving all the instances. The rows in the dataset represent instances and the columns represent the costs of different algorithms on solving each instance (Figure 1&2). The task is to create a classifier using neural network to predict and select the algorithm with the lowest cost when given a new instance. I first approach the task as a regression problem, then as classification with costs taken into account. After the grounds are set up, more advanced pairwise cost-sensitive classification and Random Forest models are experimented as well. The results and performance will be compared at the end.



	Feature 1	Feature 2	Feature 3
Instance 1	2.000	3.001	4.002
Instance 2	1.004	1.002	3.290
Instance 3	2.193	4.920	3.090
Instance 4	4.102	3.677	4.288

Figure 1. Demonstration of instance dataset



	Algorithm 1	Algorithm 2	Algorithm 3
Instance 1	2.203	2.918	3.122
Instance 2	1.101	3.975	3.144
Instance 3	4.092	5.111	2.393
Intsance 4	5.102	4.102	3.201

Figure 2. Demonstration of performance dataset

2.3 System Architecture

Apart from files for setting up the environment, the key architecture is as follows (Figure 3). The data are already split into “test” and “train” in the /data folder, each containing one file for instance features and another for algorithms’ performance. Inside /scripts folder, the train.py has all the implemented parts which use the neural network in model.py. The hyper-parameter tuning methods for different approaches are separated into individual files and can be called when necessary. After training, the model states will be saved under /models folder, which can be loaded later by evaluate.py to further test the models' performance on the testing set.

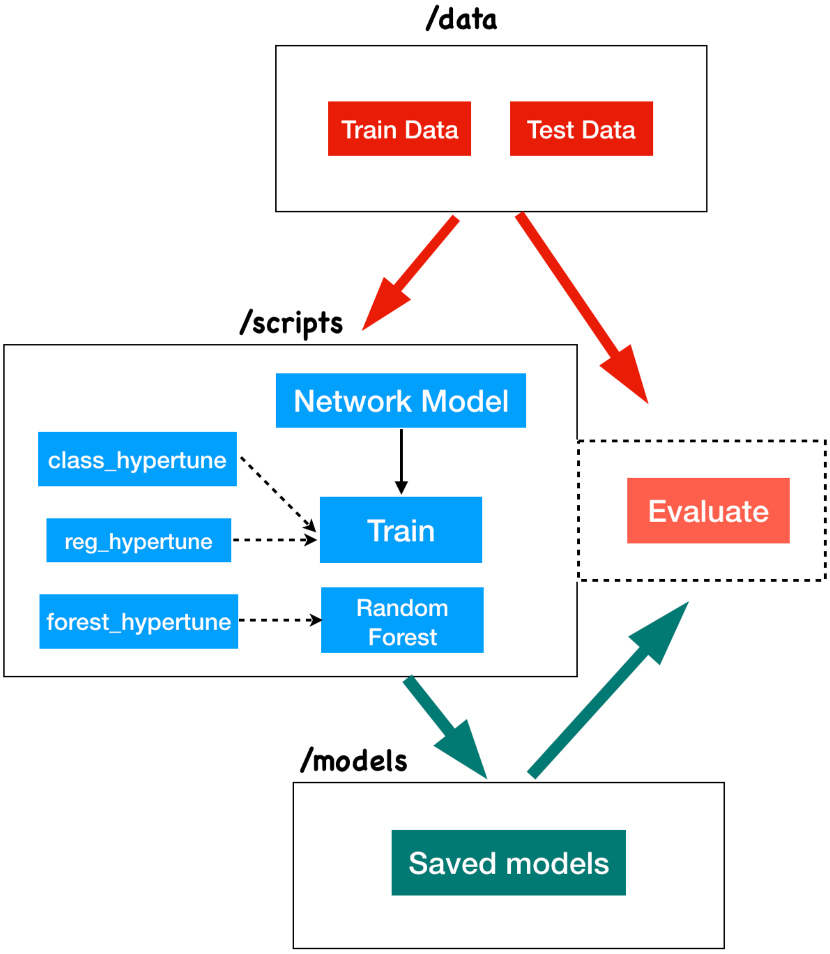


Figure 3. My System Architecture

2.4 Key Elements of Implementation

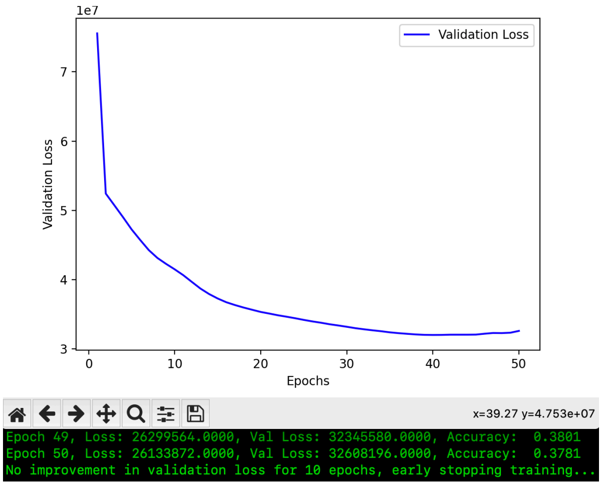
Regression & Classification:

The key implementation difference between the regression and classification approach for both neutral network and random forest ought to be mentioned. For **neutral network**, both approaches use the same network model but the loss function decides which approach it is. Regression uses **MSE** as the loss function because the goal is to predict the cost of each algorithm on each instance and try to minimize the differences to the true values. Ergo, the output would be the predicted costs. To calculate the accuracy, argmin(axis=1) is majorly used. Argmin(axis=1) returns the indices of the lowest value for each row, this exactly represents the algorithms with the lowest costs in the instance order. We do this on both train vs prediction and validation vs prediction_validation to compare the indices and discover how many predictions were made correctly, which is the accuracy. For classification however, the goal is really to output the probability of the algorithms being the best algorithm on each instance, instead of actual costs. It makes sense to add a softmax layer at the end of the network model, however, **CrossEntropyLoss** is already a combination of LogSoftmax and NLLLoss. Therefore, we can safely use the same model as regression, and a mere change of loss function should suffice. The argument passed into the loss function should also be the true labels (indices) of the best algorithm instead of raw costs. This indicates that the prediction output will be probabilities of being the best algorithm, so to get the indices for comparison, we need argmax(axis=1) instead, as they have the highest chances of being the best on different instances.

The same logic applies to **Random Forest** as well, using sci-kit learn's regressor class, we pass in the original costs for it to better learn and predict, then use argmin(axis=1) to calculate accuracy (the recommended MSE() cannot be used as getting the costs right isn't the objective). For the classifier class, we pass in the true labels again and can directly use the accuracy() method to get the percentage number of correct predictions

Early Stopping Mechanism:

An early stopping mechanism (Figure 4) is implemented during neural network training. The system will monitor the progress of loss in validation set, if the validation loss has not decreased for 10 epochs, the training will be stopped to avoid overfitting.



```
Epoch 49, Loss: 26299564.0000, Val Loss: 32345580.0000, Accuracy:  0.3801
Epoch 50, Loss: 26133872.0000, Val Loss: 32608196.0000, Accuracy:  0.3781
No improvement in validation loss for 10 epochs, early stopping training...
```

Figure 4. Demo of Early Stopping

Hyperparameter Tuning:

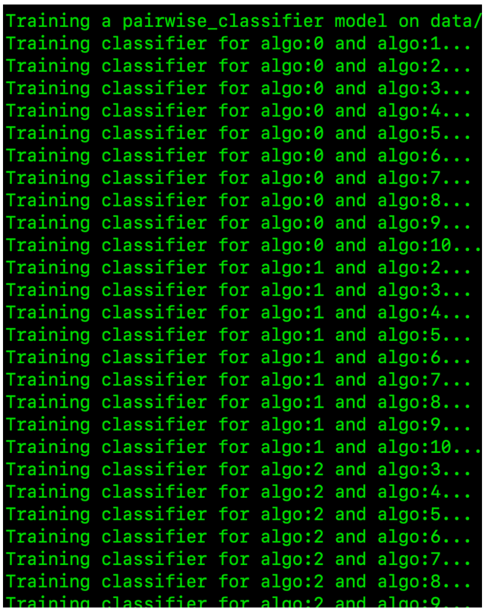
Hyper-parameter tuning is done for each approach prior to training and acquiring the results. For the two neural networks, I utilized the RandomSearchCV from sklearn library and tested many combinations on 4 hyper-parameters: learning rate, optimizer weight decay, model hidden size and training batch size. The process fitted 5 folds for each of 20 candidates, totalling 100 fits, and the best parameters were then used during actual training. The same parameters for Part 2 classification are used for pairwise classifier as well as their essences are the same and they use the same network model. For Random Forest model, I utilized the GridSearchCV from sklearn library and tested combinations on 5 core hyper-parameters for random forests: n_estimators, max_depth, min_samples_split, min_samples_leaf and bootstrap. They are used in actual training later on (Note that in the current version of the code, hyper tuning calls are commented out and the best parameters are directly used already).

SBS-VBS Gap:

Since SBS and VBS always remain the same value due to their definition, the only thing to calculate is the average predicted best cost on all instances. I first use the trained models to make predictions and retrieve the predicted indices of the best algorithms. Then apply those indices onto the original test dataset to get the actual costs of the predicted algorithms. Lastly, calculate the mean and plug this average cost into the formula for SBS-VBS gap.

Details in Pairwise Classifier:

To realize the pairwise approach, `itertools.combinations()` is used to first get unique combinations of pair algorithm indices, then iterate over the combinations to train each classifier which will be stored in a list together with the two indices involved (Figure 5). The operation for this classifier is different since the objective is only to find the better candidate between the two. Ergo, the output dimension of the neural network is strictly 2, and a binary label array is first constructed to indicate which of the two in each instance has the lower cost. This is the key for training the classifier, it is passed as the y_true argument into the model. The regret term is calculated the same way, by getting the mean of the differences between predicted algorithms and true best algorithms, only in this case the predicted will be one or the other. During testing, the array that stores the indices together with the corresponding classifiers will be iterated upon to gradually collect votes on the better candidates. Lastly, we use argmax(axis=1) to get the final predicted best algorithms that have the highest votes.



```
Training a pairwise_classifier model on data/
Training classifier for algo:0 and algo:1...
Training classifier for algo:0 and algo:2...
Training classifier for algo:0 and algo:3...
Training classifier for algo:0 and algo:4...
Training classifier for algo:0 and algo:5...
Training classifier for algo:0 and algo:6...
Training classifier for algo:0 and algo:7...
Training classifier for algo:0 and algo:8...
Training classifier for algo:0 and algo:9...
Training classifier for algo:0 and algo:10...
Training classifier for algo:1 and algo:2...
Training classifier for algo:1 and algo:3...
Training classifier for algo:1 and algo:4...
Training classifier for algo:1 and algo:5...
Training classifier for algo:1 and algo:6...
Training classifier for algo:1 and algo:7...
Training classifier for algo:1 and algo:8...
Training classifier for algo:1 and algo:9...
Training classifier for algo:1 and algo:10...
Training classifier for algo:2 and algo:3...
Training classifier for algo:2 and algo:4...
Training classifier for algo:2 and algo:5...
Training classifier for algo:2 and algo:6...
Training classifier for algo:2 and algo:7...
Training classifier for algo:2 and algo:8...
Training classifier for algo:2 and algo:9...
```

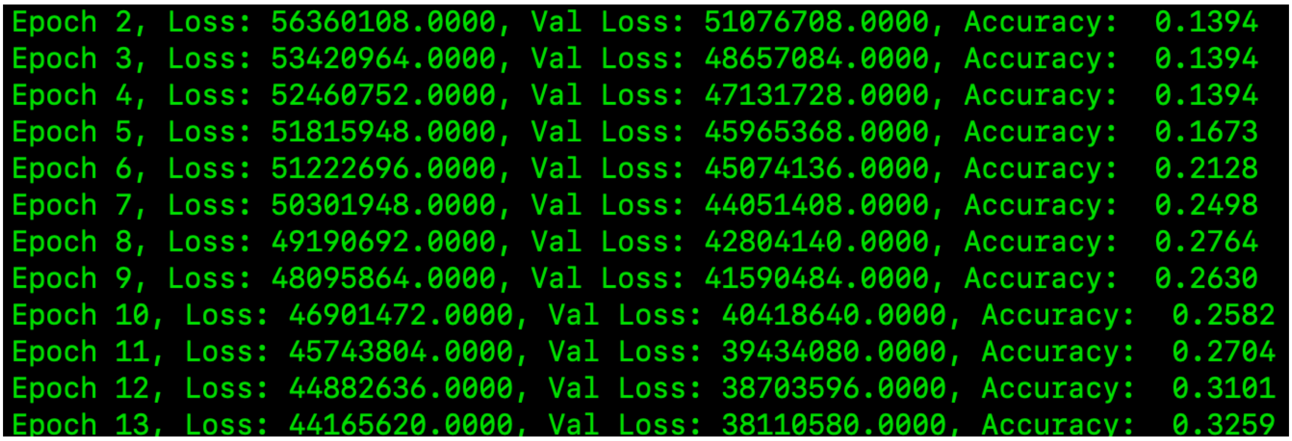
Figure 5. Training on all combinations of pair algorithms

3 Testing

Since there is no provided test for this practical, testing for system correctness is done manually.

3.1 Learning Process

The correctness of neural network's learning process can be justified by the gradually improving accuracy and decreasing validation loss. This pattern is shown in all first three parts (Figure 6).

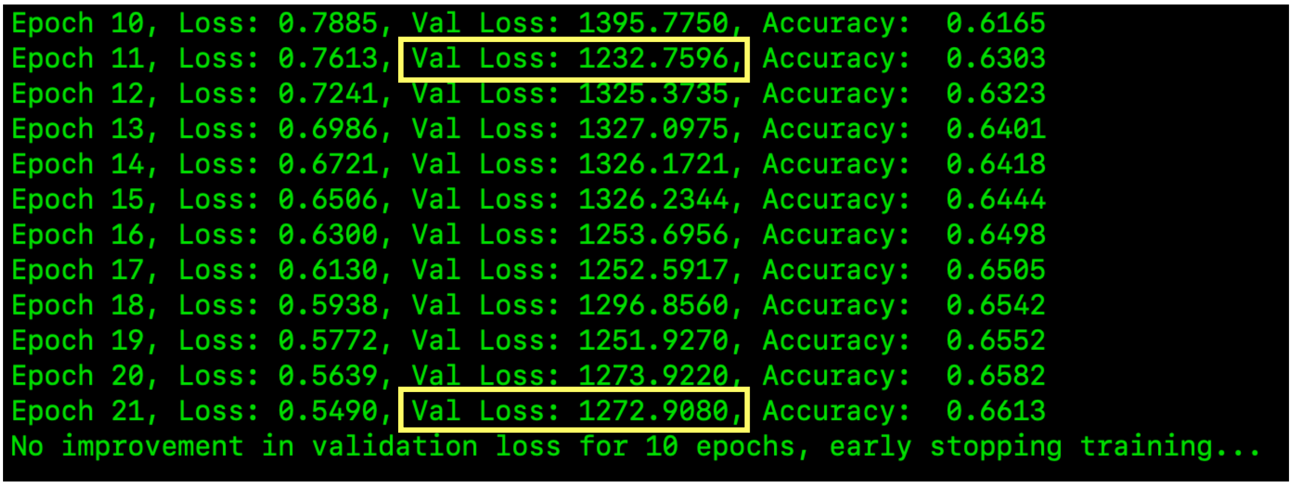


```
Epoch 2, Loss: 56360108.0000, Val Loss: 51076708.0000, Accuracy:  0.1394
Epoch 3, Loss: 53420964.0000, Val Loss: 48657084.0000, Accuracy:  0.1394
Epoch 4, Loss: 52460752.0000, Val Loss: 47131728.0000, Accuracy:  0.1394
Epoch 5, Loss: 51815948.0000, Val Loss: 45965368.0000, Accuracy:  0.1673
Epoch 6, Loss: 51222696.0000, Val Loss: 45074136.0000, Accuracy:  0.2128
Epoch 7, Loss: 50301948.0000, Val Loss: 44051408.0000, Accuracy:  0.2498
Epoch 8, Loss: 49190692.0000, Val Loss: 42804140.0000, Accuracy:  0.2764
Epoch 9, Loss: 48095864.0000, Val Loss: 41590484.0000, Accuracy:  0.2630
Epoch 10, Loss: 46901472.0000, Val Loss: 40418640.0000, Accuracy:  0.2582
Epoch 11, Loss: 45743804.0000, Val Loss: 39434080.0000, Accuracy:  0.2704
Epoch 12, Loss: 44882636.0000, Val Loss: 38703596.0000, Accuracy:  0.3101
Epoch 13, Loss: 44165620.0000, Val Loss: 38110580.0000, Accuracy:  0.3259
```

Figure 6. Example demonstration of network training

3.2 Early Stopping

The system is set to interrupt training when validation loss ceased to decrease for a consecutive 10-epoch block, this is to avoid overfitting the training dataset. As shown in the demonstration, the validation loss reached the lowest at the 11th epoch, and hence early stopped after the 21st epoch (Figure 7).

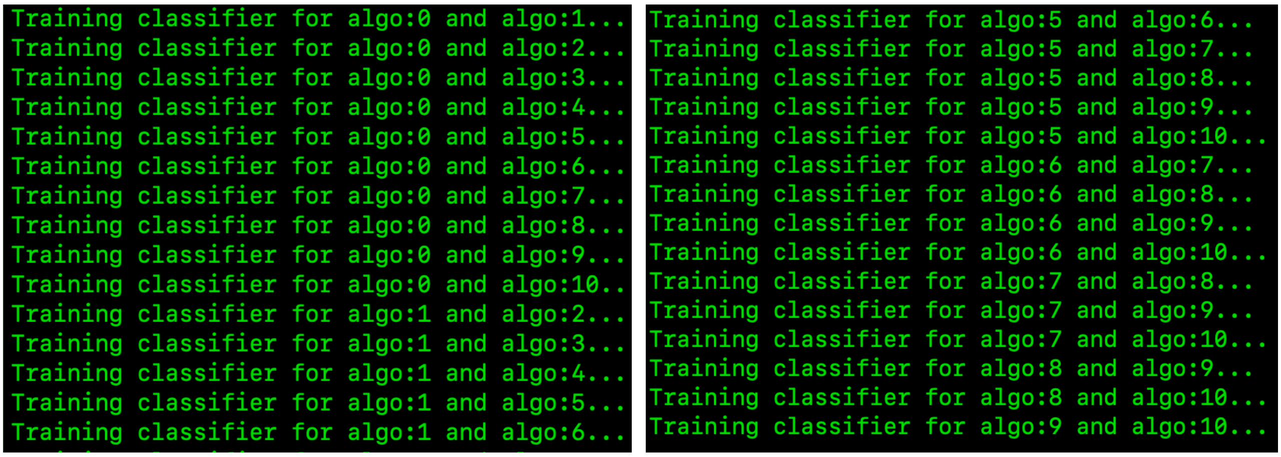


```
Epoch 10, Loss: 0.7885, Val Loss: 1395.7750, Accuracy:  0.6165
Epoch 11, Loss: 0.7613, Val Loss: 1232.7596, Accuracy:  0.6303
Epoch 12, Loss: 0.7241, Val Loss: 1325.3735, Accuracy:  0.6323
Epoch 13, Loss: 0.6986, Val Loss: 1327.0975, Accuracy:  0.6401
Epoch 14, Loss: 0.6721, Val Loss: 1326.1721, Accuracy:  0.6418
Epoch 15, Loss: 0.6506, Val Loss: 1326.2344, Accuracy:  0.6444
Epoch 16, Loss: 0.6300, Val Loss: 1253.6956, Accuracy:  0.6498
Epoch 17, Loss: 0.6130, Val Loss: 1252.5917, Accuracy:  0.6505
Epoch 18, Loss: 0.5938, Val Loss: 1296.8560, Accuracy:  0.6542
Epoch 19, Loss: 0.5772, Val Loss: 1251.9270, Accuracy:  0.6552
Epoch 20, Loss: 0.5639, Val Loss: 1273.9220, Accuracy:  0.6582
Epoch 21, Loss: 0.5490, Val Loss: 1272.9080, Accuracy:  0.6613
No improvement in validation loss for 10 epochs, early stopping training...
```

Figure 7. Example demonstration of early stopping

3.3 Pairwise Classifier

The idea of pairwise classifier approach is to train many classifiers that only focus on two algorithms at a time. Therefore, making sure all combinations of pair-algorithms are included is essential, the following run with print statement justifies that no classifier is left out or repeated (Figure 8).



```
Training classifier for algo:0 and algo:1...
Training classifier for algo:0 and algo:2...
Training classifier for algo:0 and algo:3...
Training classifier for algo:0 and algo:4...
Training classifier for algo:0 and algo:5...
Training classifier for algo:0 and algo:6...
Training classifier for algo:0 and algo:7...
Training classifier for algo:0 and algo:8...
Training classifier for algo:0 and algo:9...
Training classifier for algo:0 and algo:10..
Training classifier for algo:1 and algo:2...
Training classifier for algo:1 and algo:3...
Training classifier for algo:1 and algo:4...
Training classifier for algo:1 and algo:5...
Training classifier for algo:1 and algo:6...
```

```
Training classifier for algo:5 and algo:6...
Training classifier for algo:5 and algo:7...
Training classifier for algo:5 and algo:8...
Training classifier for algo:5 and algo:9...
Training classifier for algo:5 and algo:10...
Training classifier for algo:6 and algo:7...
Training classifier for algo:6 and algo:8...
Training classifier for algo:6 and algo:9...
Training classifier for algo:6 and algo:10...
Training classifier for algo:7 and algo:8...
Training classifier for algo:7 and algo:9...
Training classifier for algo:7 and algo:10...
Training classifier for algo:8 and algo:9...
Training classifier for algo:8 and algo:10...
Training classifier for algo:9 and algo:10...
```

Figure 8. Example run for pairwise method from start to end

4 Evaluations and Conclusion

4.1 Training Performance

The models’ performance during training varied greatly, below is the graph (Figure 9) generated by matplotlib during one run to showcase the accuracies on validation and training set. In general, using regression neural network will usually complete all the epochs without early stopping and achieve lower training accuracy around 40%. The classification approach is able to get a 70% accuracy with interruption at earlier epochs.

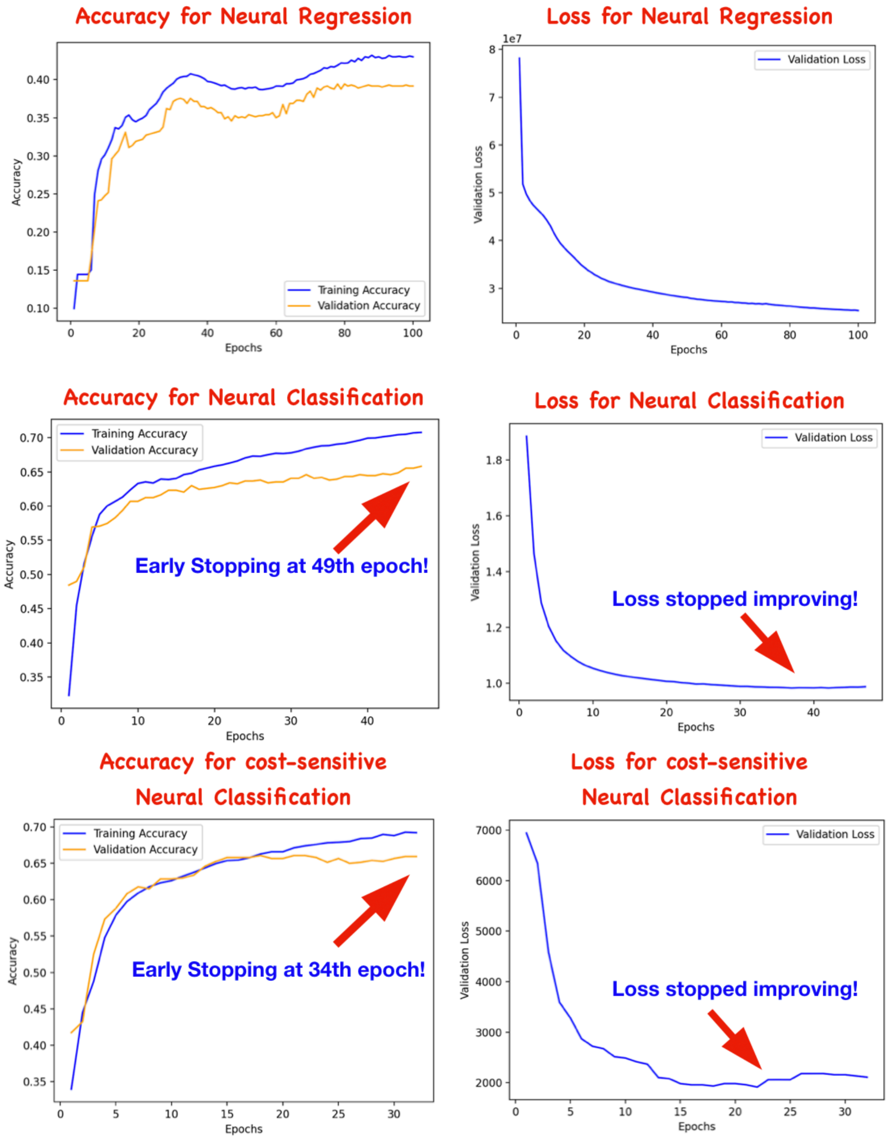


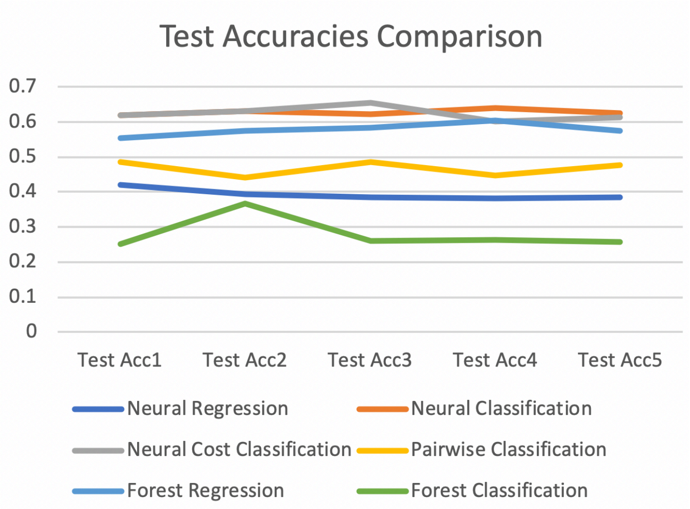
Figure 9. Plotted graphs of training performance for neural network models

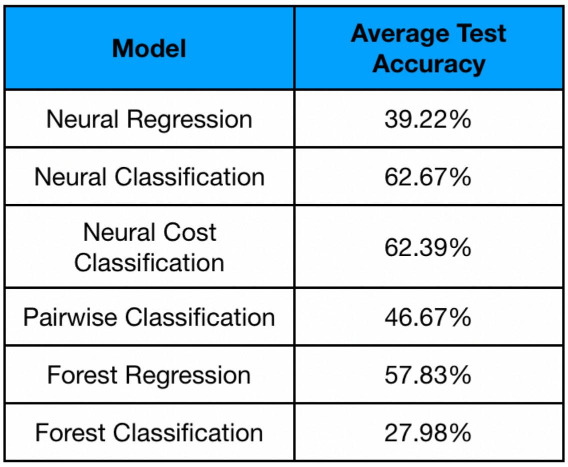
The same disparity applies to random forest models as well where the regressor achieves around 60% and classifier around 86%. This indicates that in regards to learning to select the best algorithms, approaching the problem as classification demonstrates faster execution and more promising results while training.

4.2 Testing Performance

Criterion 1. Testing Accuracy

The comparison of testing accuracies from different approaches are visualized in the graph and table below with 5 recorded runs for generalization.





Model	Average Test Accuracy
Neural Regression	39.22%
Neural Classification	62.67%
Neural Cost Classification	62.39%
Pairwise Classification	46.67%
Forest Regression	57.83%
Forest Classification	27.98%

From the rollup, random forest classification seemed to have overfitted the training data and cannot compete with neural classification. Both regression models performed with consistency and random forest showed better testing results. However, pairwise classifiers' performance remained in the middle.

Criterion 2. Average Cost & SBS-VBS Gap

The true aim is to minimize these two metrics. The performance of average costs across all instances are shown below (Figure 10). Contrary to "accuracy", neural regression showed the best competence on minimizing the average cost of the selected best algorithms. This suggests that while the predicted best algorithms could be wrong, they're always only "slightly off" as the model would most likely pick the second or third best. Therefore, the model learns to predict the costs very well. All three neural classification approaches did well on this mission, it is observed that incorporating "regret" into neural classification training can decently improve the model's performance. Dividing the labor into multiple binary classifiers also started to show true competence in this aspect. Lastly, both random forest models paled in comparison to neural networks.

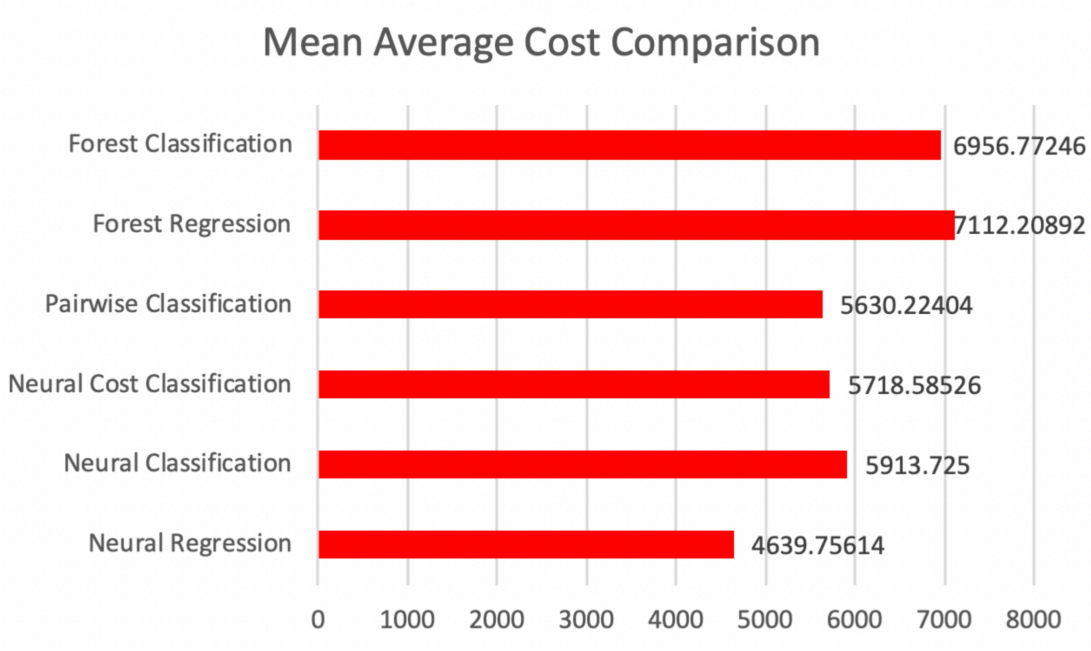


Figure 10. Predicted average costs across all instances calculated over 5 runs

The performance of SBS-VBS Gaps are summarized below (Figure 11). Similarly, neural regression showed extremely satisfying SBS-VBS Gap that's close to 0, followed by neural classification approaches. Pairwise classification also managed to minimize this metric better than conventional neural classification. Random forests continued to "lose" on this task, although the performance were still acceptable.

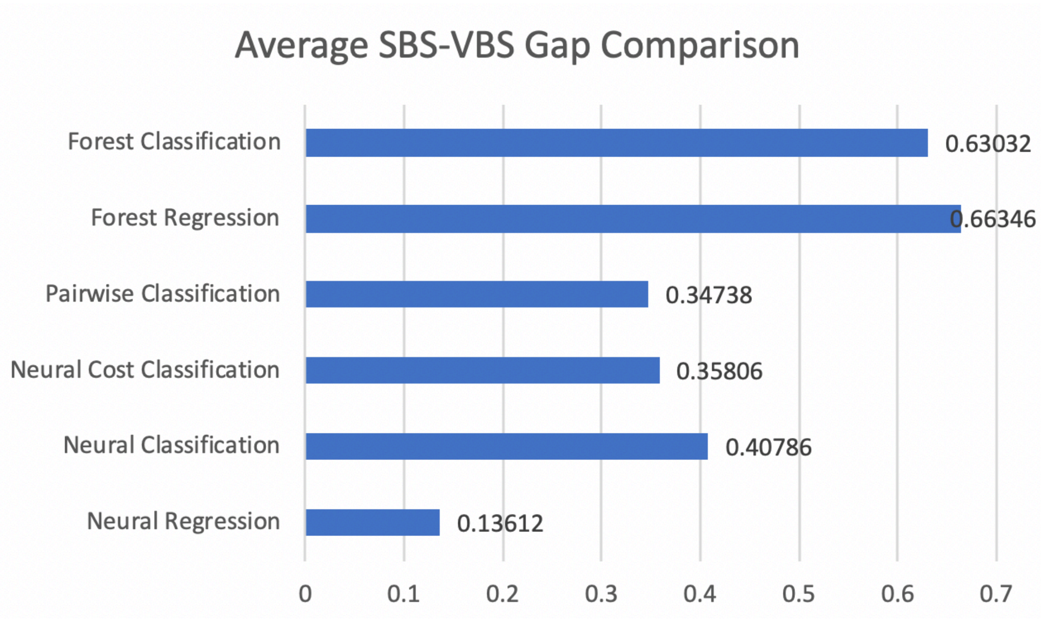


Figure 11. Average SBS-VBS gaps calculated over 5 runs

Conclusion:

To conclude, the neural regression approach tackled the AS problem more effectively than classification even when its "accuracy" was low. It successfully learned to predict the best/almost-best algorithms. This could be because that the actual data and the predicted data are of the same nature, and hence contained more information that helped the learning. Classification on the other hand, was trained using "true labels" (less information) which lead to slightly "worse" performance. To counter that, taking costs/regret into account added more information to the training and hence demonstrated improvement. This effect is more evident on pairwise classification as the classifiers only focus on two algorithms at a time and it helped them to master the learning for more informed predictions. Lastly, both random forest models showed satisfactory accuracies but were left far behind by neural network models on core metrics.

5 Bibliography

Patrick Loeber (2020) “Softmax and Cross Entropy”. *Youtube*. https://www.youtube.com/watch?v=7q7E91pHoW4

Rendyk (2021) “Tuning the Hyper-parameters and Layers of Neural Network Deep Learning”. *Analytics Vidhya*. https://www.analyticsvidhya.com/blog/2021/05/tuning-the-hyperparameters-and-layers-of-neural-network-deep-learning/

Will Koehrsen (2018) “Hyperparameter Tuning the Random Forest in Python”. *Towards Data Science*. https://towardsdatascience.com/hyperparameter-tuning-the-random-forest-in-python-using-scikit-learn-28d2aa77dd74

Lin Xu (2012) “Improved algorithm selection based on cost-sensitive classification models”. *Proceedings of SAT Challenge*, pages 57–58, 2012.

Nguyen Dang (2023) “Algorithm Selection”. *University of St Andrews Student Resource*. https://studres.cs.st-andrews.ac.uk/CS5011/Lectures/L16-Learning-3-4.pdf

Nguyen Dang (2023)“Introduction to PyTorch”. *University of St Andrews Student Resource*. https://colab.research.google.com/drive/112dka-dUiY1L_D_OsFnJ3blKlQpsZa0z?usp=sharing

Scikit-learn Developers (2023) “Metrics and scoring: quantifying the quality of predictions”. *Scikit Learn*. https://scikit-learn.org/stable/modules/model_evaluation.html