Spark Machine Learning Report

LINK SOUTH 222

ziwei zhang 460151894

zihua guo 470349775

sheng yuan 430058318

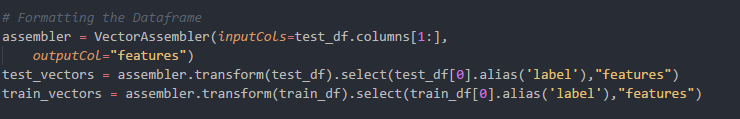
# Design

## Architecture

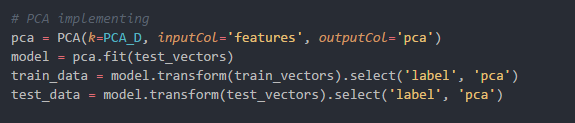
In this program, computations are mainly based on spark Dataframe rather than RDD. The input csv is read via spark csv reader and it generate a Dataframe with label and 784 columns features.



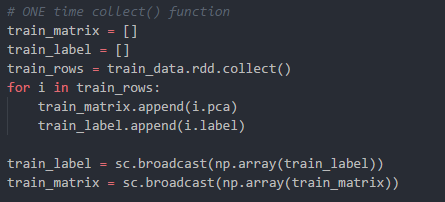
Then these dataframes will go through a VectorAssember to format themselves.



Next step the dimension of features will be reduced with the help of PCA function in order to accelerate the running process.

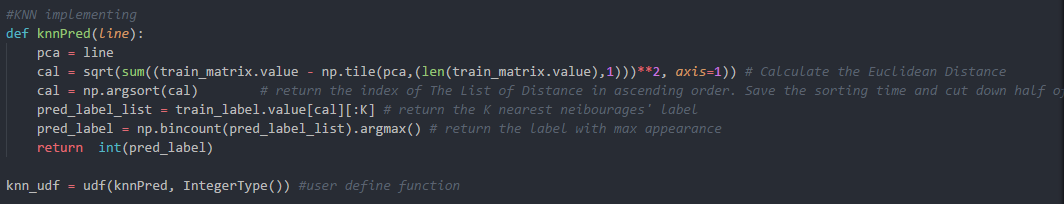


Before the KNN, we extract the pca\_feature and label of the training data and compute them into an array and broadcast to each node.



Next is the KNN computation. The computation formula is simple:

Then we sort the matrix by row and return the first K label. The label that appear the most is the predicted label of the test point.



The following step is to calculate the confusion matrix and the classification report.

Basically, all the functions are achieved without error.

## Optimization

However, during those time, Code has improved several times.

### Numpy Matrix

When I first compute the KNN, I was calculating them row by row. In this case, it would cost over 24 hours to finish all the steps which is normally inefficient. Using matrix for the computation apparently saving lots time since it would only need to map 10000 times instead of 600 million. It helps me cut down the time from over 24hours to less than 3 minutes.

### Argsort

Sorting is always the time-consuming step. I used to sort a two-dimension array with label in one column and distance in the others. With the help of Argsort, I only need to sort the distance list and it will return the index of list in ascending order. Then, we only need to use the index to map the label list and it will return the label in ascending distance order. In this way, it used to cost 0.02s for each output and now it cost 0.0075s for each, which is 3 time faster.

### Repartition



During computing, I find out that the collect() and saveAsTextFile() methods cost most of the computation time. What’s more, they are not able to work very parallelly with only 5 tasks running at the same time. With the help of repartition() function, tasks can be split into small subset and are able to run as parallel as possible. For this part, I refer to apache document and find out that the default size of each task is already defined. In this case, our size of input for these two methods can only be split into 5 tasks by default. With defining the repartition number, it can be split as many as we wish. However, I recommend setting it same as executors \* cores since it will be able to run exactly the same number of tasks at the same time as the total cores number. I’ve try up to 32 repartitions with corresponding 32 total cores. However, the KNN computation time is 36s which is longer than 22s with only 16 repartitions. In conclusion, this factor should be defined based on situations.

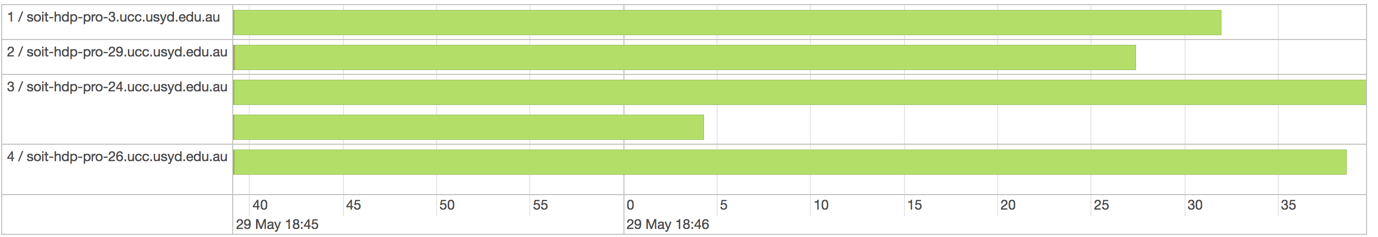


Figure1: 5 partition of computations

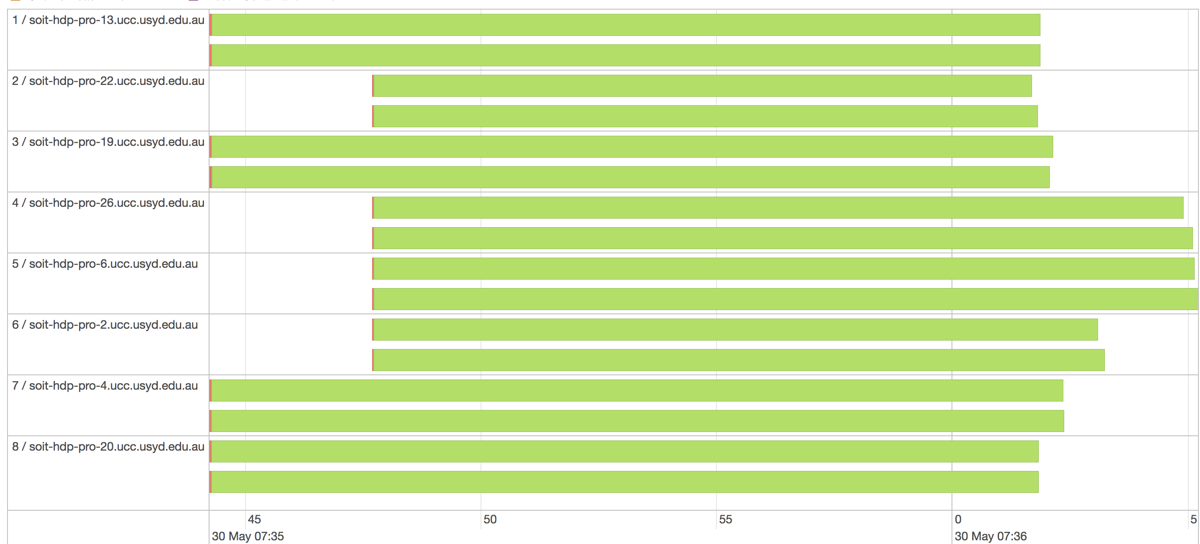


Figure2: 16 Repartition of computations

### Broadcast

It is a built-in function of spark context for broadcasting the values to different nodes. In this case, node will save time acquiring context each time from the master node. However, in different circumstance, result would be opposite. When I run the program with K=5 and PCA=50, the one with broadcast is 20s slower than the one without. For program with K=10 and PCA=100, with the help of broadcasting, it cut down over 70s overall. In result, it probably depends on the frequency of asking the context from master node. If the frequency is high enough and the broadcasting is able to help them cut down the asking frequency. On the contrast, if the frequency is low, the broadcasting time may be overwhelming the overall asking time. Therefore, with larger input, using broadcasting will become necessary.

## Sample Result

The following sample parameters: 8 executors, 2 cores, PCA=50, D=5

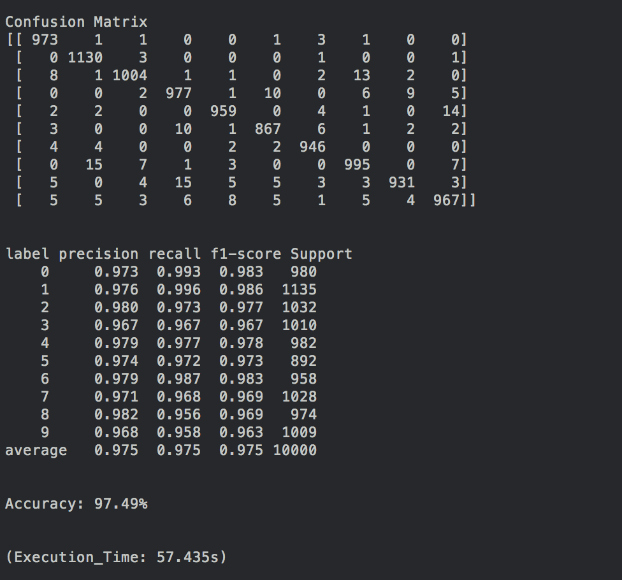
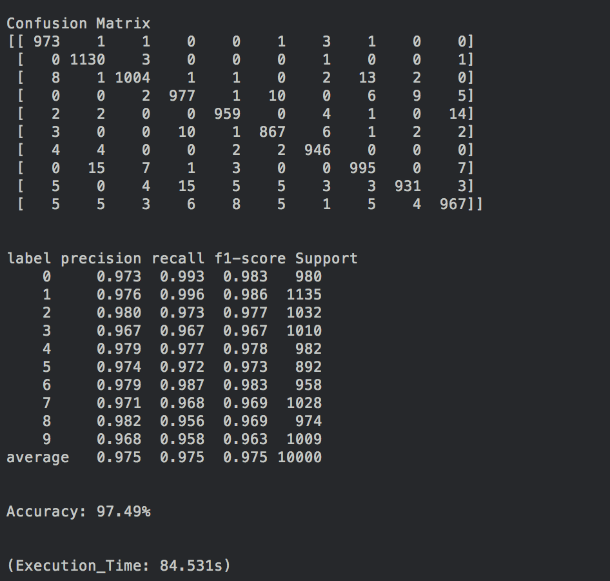
 

Figure3: KNN only Figure4: KNN & produce predict label

# Performance analysis

## Environment

For the performance analysis, the following environment is used:

OS: Linux

OS Architecture: amd64

Java version: 1.8.0\_40

Scala version: 2.11.8

Driver memory: 1 GB

Executor memory: 1 GB

## Analysis

For the variables, we decided to use d = [50, 100], k = [5, 10] and (executors, cores) = [(2, 4), (4, 8), (8, 16)]. The reason for choosing these species are to ensures enough differences from one test to another. All these tests are running on the same environments. However, error might appear as the cluster is crowded.

First, we will be looking at the difference in execution time in relation of the number of cores. For this comparison, groups with d = 50 and k = 5 are used.

Figure 1: Execution time in relation to numbers of executors and cores.

From figure 1, we can see that the execution time decrease as the number of executors and cores increases. However, the decrease rate drops from 22.808 seconds/ (executors, cores) to 6.393 seconds/ (executors, cores). The main reason for the decrease in rate of reduction is that as the number of executors increases, the reduction in running time for each executor will finally be outweighed by the shuffle I/O cost as indicated in the figure 2 below.

Figure 2: Shuffle read and write cost in relation to numbers of executors and cores.

As we can see, the read cost is rising along with the number of executors and cores, and the execution time needed for shuffling increases.

For differences related to reduce dimension and K nearest neighbour number, we choose groups with 2 executors and 4 cores. The differences are show in figure 3 and figure 4.

Figure 3: Execution time in relation to K nearest neighbour number and reduce dimension.

Figure 4: Shuffle I/O cost in relation to K nearest neighbour number and reduce dimension.

According to figure 3, it’s safe to say that the influence brought by a reduce dimension is significant. While increasing reduce dimension by 50, the execution time increases by 382.88 seconds. On the other hand, increases in K nearest neighbour number has minor effects in execution time.

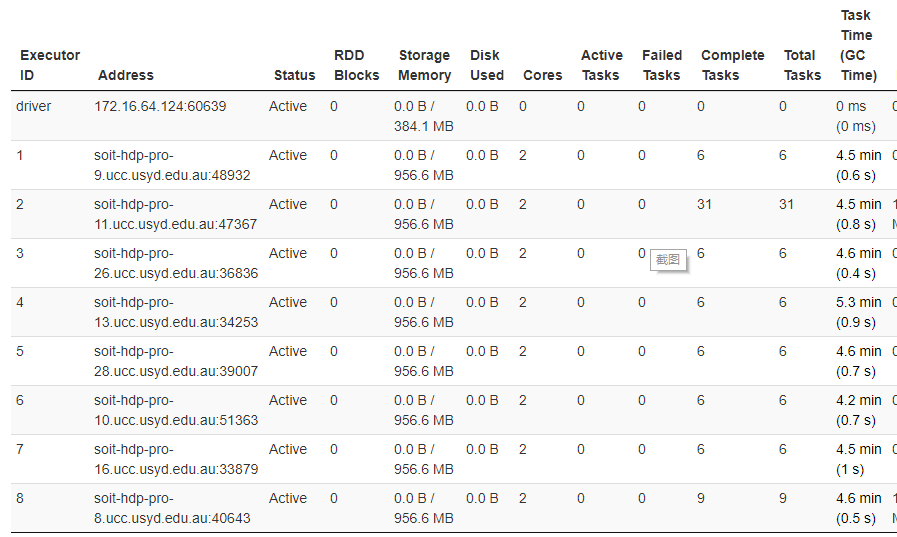
From figure 4, conclusion can be draw that increase in reduce dimension also increase shuffle I/O costs significantly while changes in K nearest neighbour number has no effect in shuffle I/O cost.

We also found that reduce dimension has a significant effect in the duration of “collect” operation as shown in figure 5.

Figure 5: Operation time for collect and save as file in relation to K nearest neighbour number and reduce dimension

From the chart, we can learn that while the reduce dimension increases, operation time for collect and save as file increases by factor of 4. Besides, the operation time for tree aggregation also increases with the reduce dimension. These two operations might be the main reason for significant increases in execution time.

During the analysis regarding the executors, we discovered that spark is capable of balancing task time amount multiple nodes as shown in the picture below:



# Classifier Exploration

## Classifier

### Random forest classifier

Random forest is one of the most popular classification algorithm to build predictive model, which was introduced by Ho in1995 and developed by Breiman in 2001 (Benyamin, 2012). In case of called random forest, this classifier is an integrated model with a number of decision trees to generate the forest. Therefore, before the explanation of the random forest algorithm, it is necessary to introduce the decision trees.

Decision trees is a top-down approach and its tree structure is constructed by different types of nodes. Firstly, the process starts with the root node and it generates a binary split based on the attributes. The process is executed until it meets the terminal criteria. Finally, the end node will provide the predicted outcome, or the predicted class. The node splitting is the important part of the decision trees, which can be achieved by implementing methods such as the Gini impurity (Polamuri, 2017).

The random forest classifier separates the dataset randomly into K numbers of subset. Each subset creates a decision tree. Hence, for each test data, it has a K possible outcome corresponding to each decision trees. To make a prediction, the class with the highest number is selected. In general cases, the higher number of trees will provide a higher accuracy results (Eulogio, 2017). The diagram of random forest classifier is shown in the Figure 3-1.

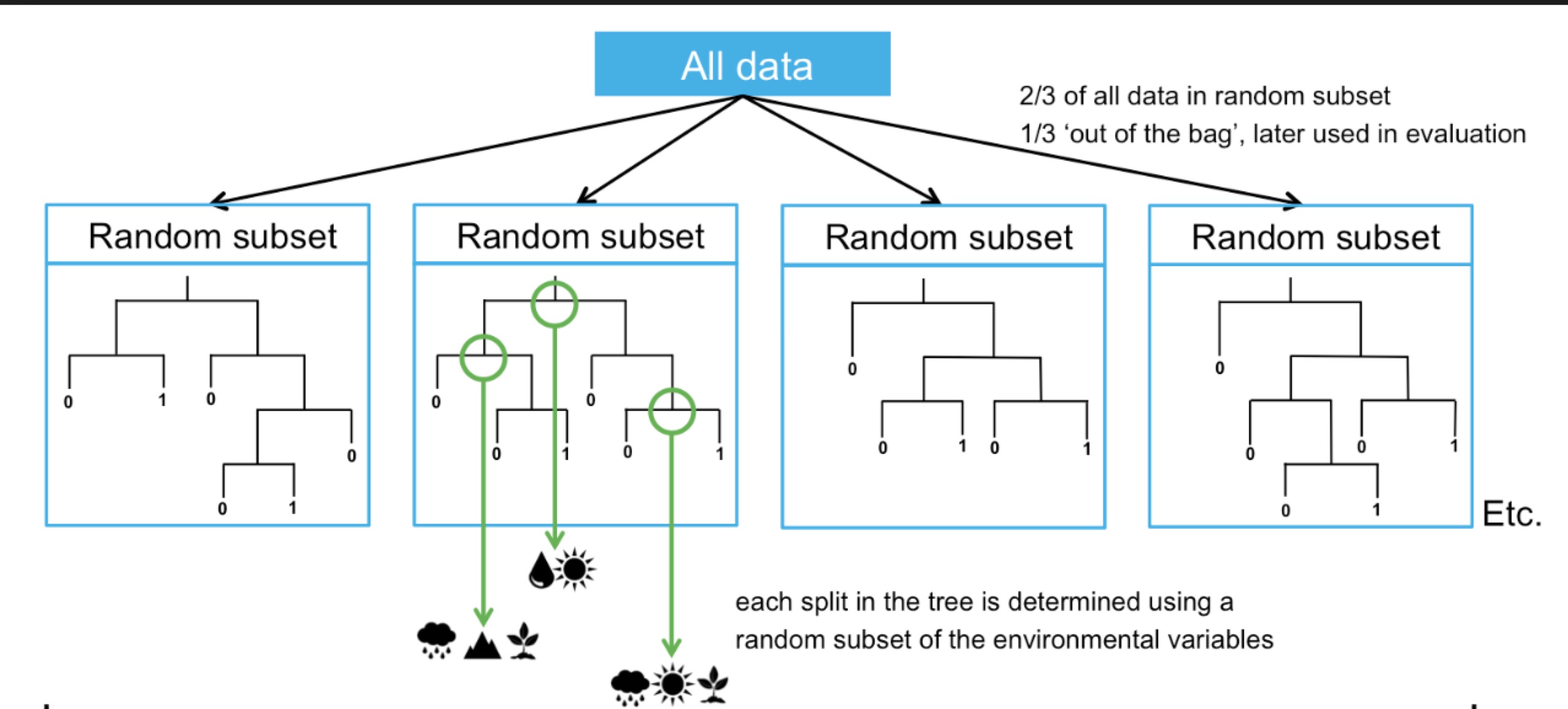


Figure 3-1 Random forest classifier diagram

### Multi-layer perceptron classifier (neural network)

The multi-layer perceptron classifier is a particular type of neural network. As called neural network, the basic components in the classifier is the neuron, which can provide a summation function to compute the output based on the external input sources. The weights are assigned to each input respectively. The activation function, such as Sigmoid, tanh or ReLU, can be implemented in the neuron (the data science blog, 2016).

The neural network shown in the Figure contains 3 types of layers, input layer, hidden layer, and output layer. For the multi-layer perceptron classifier, it consists of several hidden layers. The input layer has a bias node and external inputs nodes, which depend on the input dataset. The Hidden layer, where the activation functions are applied, passes the calculated output to the next layer. The output layer presents the predicted outcomes after similar computation. The diagram of multi-layer perceptron is shown in the Figure 3-2.

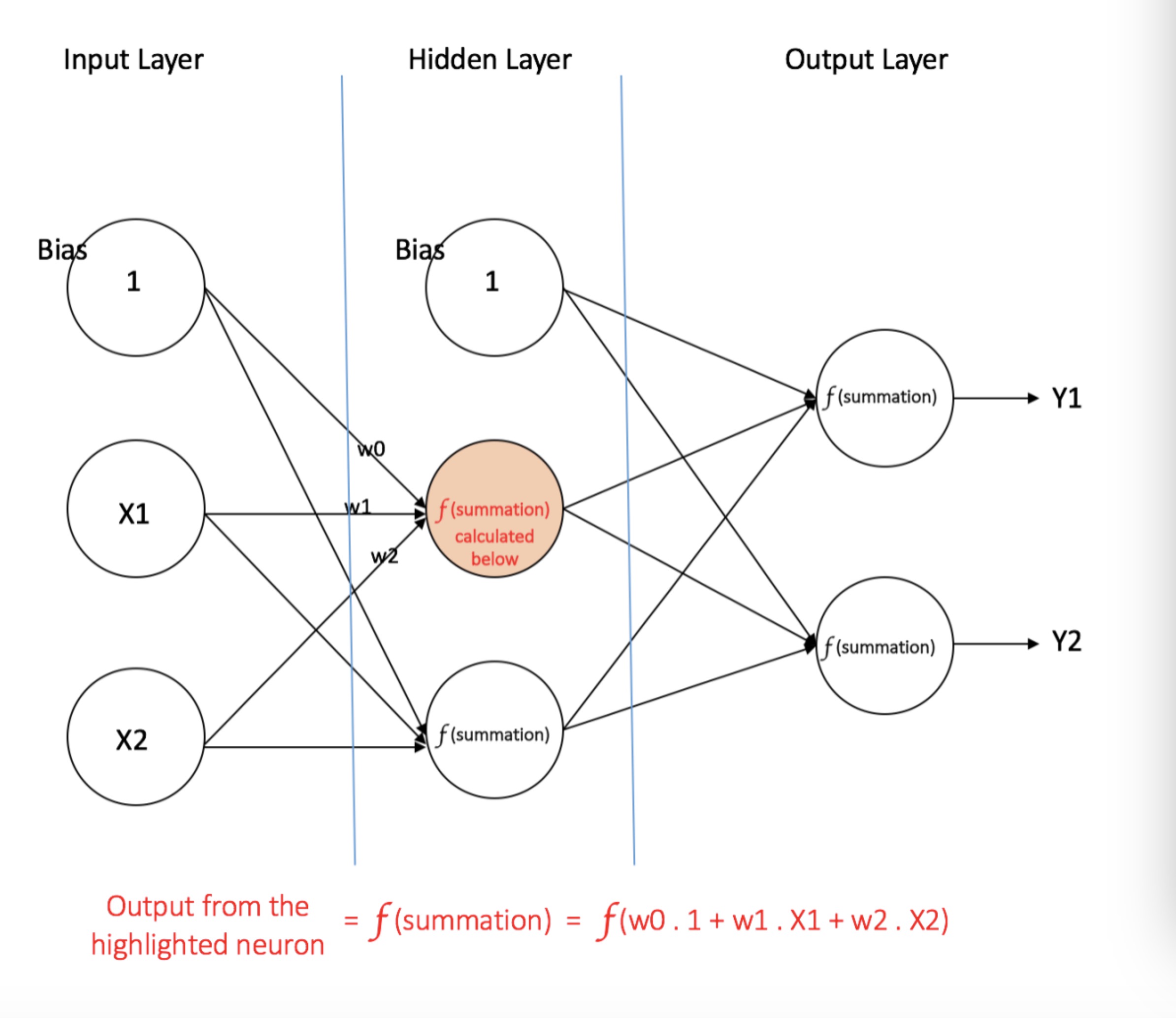


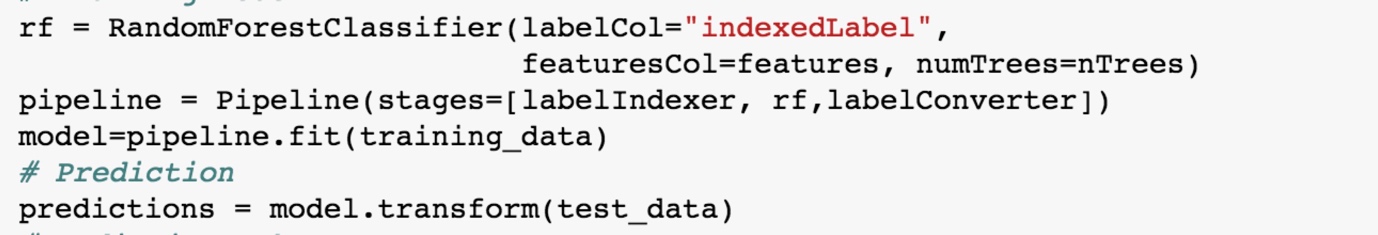
Figure 3-2 Multi-layer perceptron diagram

## Implementation

In this part, the impact on the classifiers’ performance are analysed by changing the parameter setting of each classifier. The accuracy in prediction, execution time and I/O cost will be presented in the analysis and the reduced dimension is set to 50 for both classifier.

The pyspark’s machine learning library is used in this part. The implementation for these two classifiers is similar. The first step is defining the classifier and then fitting the training dataset to train this model. Finally, the trained model is used to predict the possible outcome for the given test dataset. For random forest classifier, only the number of trees is analysed. For MLP classifier, we are interested in classifier with only one hidden layer and the impact of changing its size. The input layer is 50, output layer is 10, blocksize is 128 and maximum iteration is 100. The sample code for these two classifiers are shown in the Figure 3-3.

Generally, the StringIndexer and the VectorIndexer aer used in the pipeline, which can encode a string column of labels and features to a column of label and features indices based on the frequency. Because of the numeric representation, the result can be more understandable. However, in this part, it will not improve the performance because the labels are already represented in number. Moreover, Encoding the features will even take more time to run. Therefore, only the labels are encoded in this part.



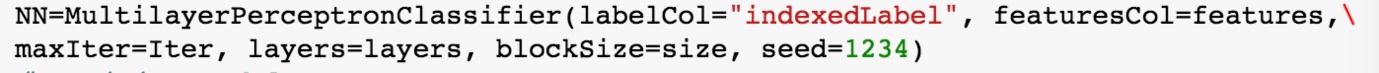


Figure 3-3 Sample code for the implementation of RF and MLP classifier

## Analysis

### Random forest classifier

For accuracy and running time analysis, the number of Trees is selected in the range from 10 to 100 by step 10. The results are shown in the Table 3-1 and Figure 3-4. For the I/O cost, the screenshots of 10, 40 and 100 number of trees are provided in the Figure 3-5.

According to the statistics, in the range from 10 to 40, the accuracy is positive correlation with the number of trees. When the number is larger than 40, the accuracy remains stable at around 81 percentage. The reasons are that when number of decision trees is increasing, more conditions are considered, which results in a better prediction. However, after it reaches to its limitation, the accuracy is not increasing any more.

For the execution time, it is obvious that the time drops dramatically from 10 to 20 and then increases steadily. Compared to the large number of trees, dataset is spilt into serval large subset to create decision trees with a large size when the small number is chosen, which leads to greater time consumption of prediction. As the number of trees becoming bigger, the size of decision trees is decreasing, and the time is reducing as well. However, when more possible results should be considered, the process becomes much more complicated, which results in time increasing.

For the I/O cost, when the number of trees is growing from 10 to 100, the shuffle write increases from the 11MB to 64MB. In the meantime, the shuffle read never change, which is 0 B. Hence, the number of trees will increase the I/O cost.

In conclusion, to select an optimal number of trees can consider the accuracy, running time as well as I/O cost. In our analysis, 40 is a suitable choice in this case.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Number of trees | 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 | 100 |
| Accuracy | 0.7767 | 0.7826 | 0.799 | 0.8119 | 0.8105 | 0.814 | 0.8124 | 0.8116 | 0.8158 | 0.8106 |
| Running time (s) | 19.22 | 12.47 | 12.09 | 12.91 | 13.3 | 14.43 | 16.86 | 16.22 | 16.36 | 17.68 |

Table 3-1 Results for RF classifier

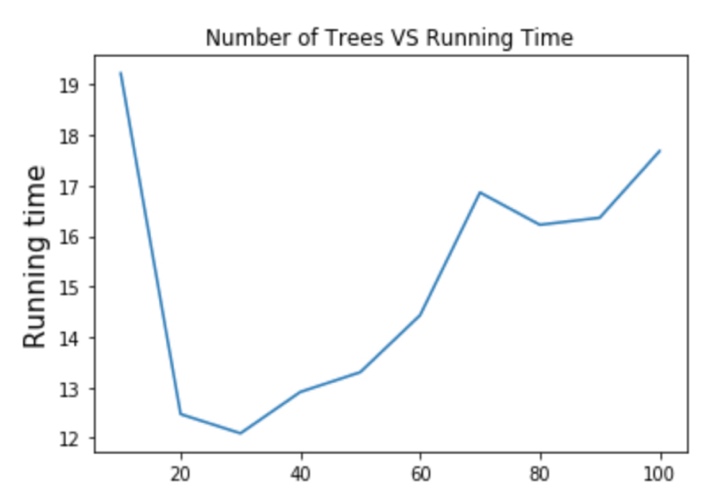
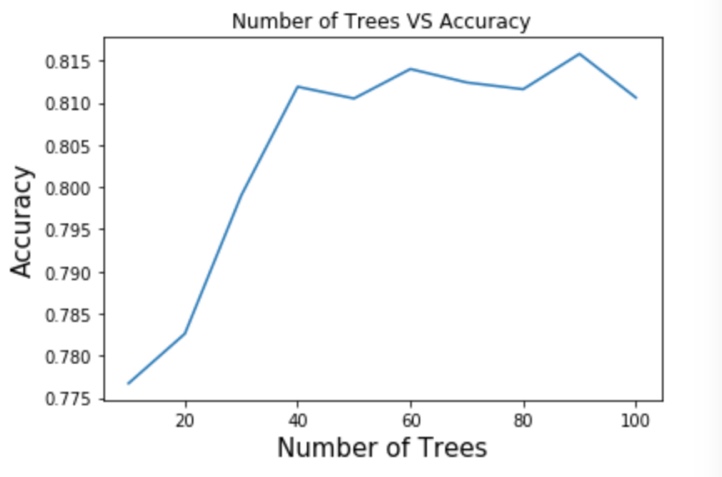
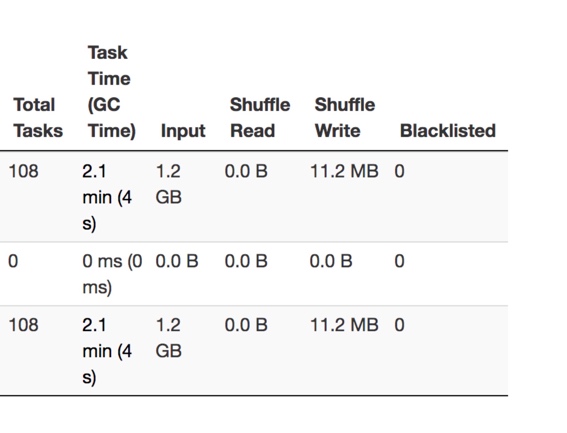
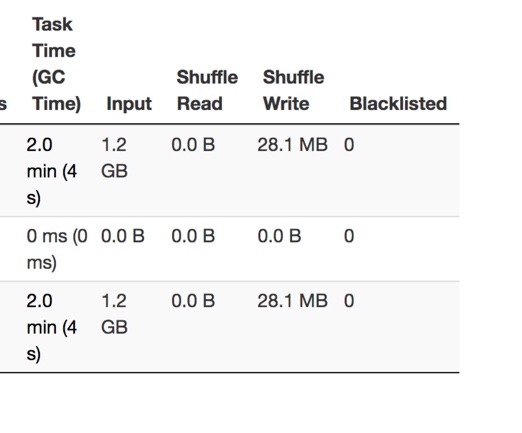


Figure 3-4 Accuracy and Execution time for RF classifier



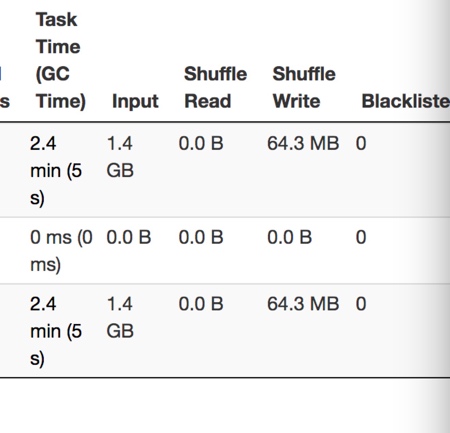


Figure 3-5 Shuffle Read/Write of RF classifier (10,40 and100 number of trees from left to right respectively)

### Multi-layer perceptron classifier

As mentioned in implementation section, the analysis in this part is changing the hidden layer size from 50 to 100 by the step of 10. The accuracy and running time results are shown in the Table 3-2 and Figure 3-6. For the I/O cost, the screenshots of 50, 75 and 100 hidden layer sizes are provided in the Figure3-7.

According to the statistics, in the range from 50 to 80, the accuracy is positive correlation with the size of hidden layer. When the size is larger than 80, the accuracy remains stable at around 93.75 percentage. The reasons are that when size of hidden layer is increasing, more neurons are in the network and more conditions are considered, which results in a better prediction. However, after it reaches to its limitation, the accuracy is not increasing any more.

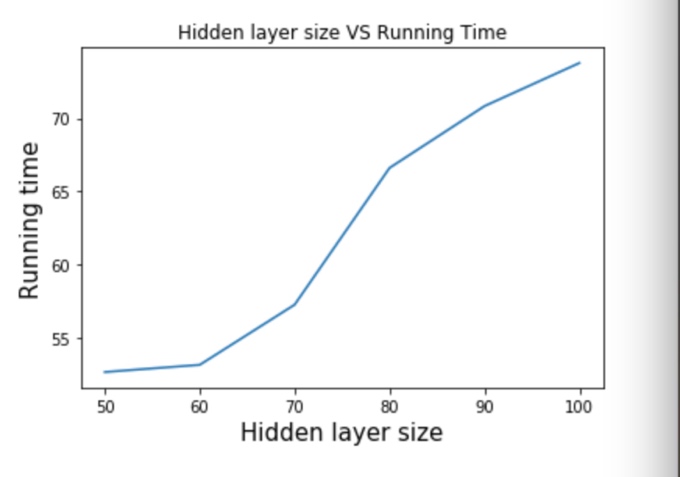
For the execution time, it is obvious the running time is increasing with the growth of hidden layer size. As the number of neurons becoming bigger, the process involves more computations, which results in time increasing.

For the I/O cost, when the size is growing from 50 to 100, the shuffle write remain stable at around 2.4 KB. In the meantime, the shuffle read never change, which is 0 B. Hence, the hidden layer size has a slight effect to the I/O cost.

In conclusion, to select an optimal size can consider the accuracy, running time as well as I/O cost. In our analysis, 80 is a suitable choice in this case.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Hidden layer size | 50 | 60 | 70 | 80 | 90 | 100 |
| Accuracy | 0.9213 | 0.9309 | 0.9302 | 0.9379 | 0.9377 | 0.9379 |
| Running time (s) | 52.63 | 53.12 | 57.24 | 66.6 | 70.83 | 73.78 |

Table 3-2 Results for MLP classifier



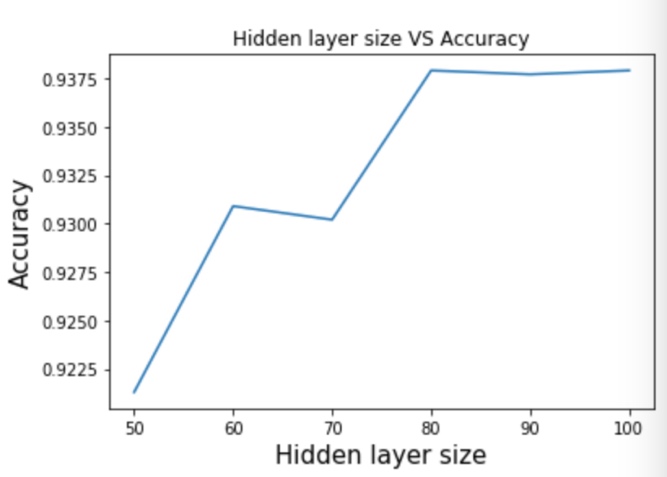


Figure 3-6 Accuracy and Execution time for MLP classifier

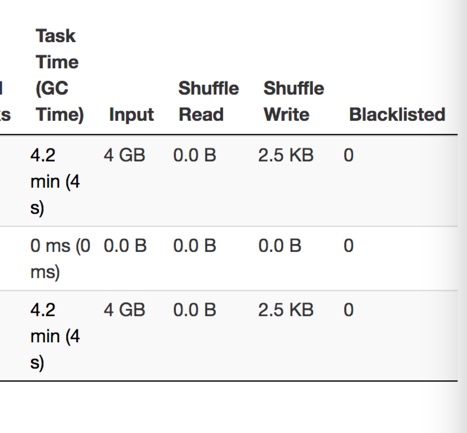
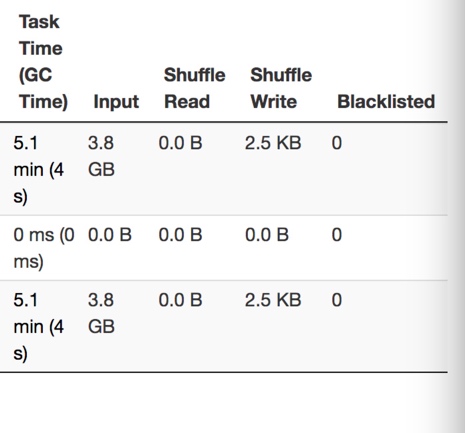
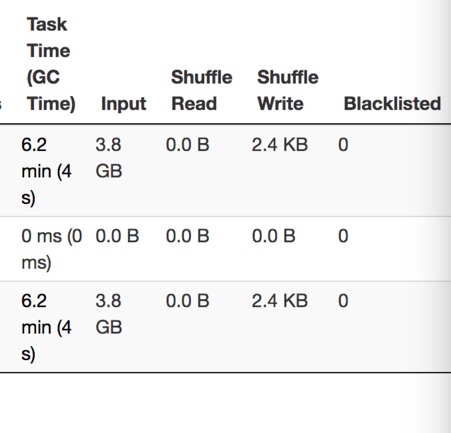


Figure 3-5 Shuffle Read/Write of MLP classifier (50, 75 and100 sizes of hidden layer from left to right respectively)

## Discussion

For this dataset, compared with the random forest classifier, the MLP classifier has a better performance in accuracy, but the execution takes more time. However, there are a variety of factors can affect the performance of MLP classifier, which are not analysed in this report, such as the maximum iteration, block size and so on. Therefore, it is hard and unreasonable to decide which classifier is better without giving any situations.

# Reference

*Benyamin, D. (2012). A Gentle Introduction to Random Forests, Ensembles, and Performance Metrics in a Commercial System. [online] Bing.com. Available at: http://www.bing.com/cr?IG=70006CA78D16421E87D8EEDB2DE67C09&CID=3D34AC24625068B23E09A7DA63AD69E7&rd=1&h=b5m98MAbg2aNXNhMnBGl3l0f\_SpMlCtg5TJ2qmUmK1M&v=1&r=http%3a%2f%2fblog.citizennet.com%2fblog%2f2012%2f11%2f10%2frandom-forests-ensembles-and-performance-metrics&p=DevEx.LB.1,5504.1 [Accessed 29 May 2018].*

*Polamuri, S. (2017). How the random forest algorithm works in machine learning. [online] Dataaspirant. Available at: http://dataaspirant.com/2017/05/22/random-forest-algorithm-machine-learing/ [Accessed 29 May 2018].*

*Eulogio, R. (2017). Introduction to Random Forests. [online] Datascience.com. Available at: https://www.datascience.com/resources/notebooks/random-forest-intro [Accessed 29 May 2018].*

*the data science blog. (2016). A Quick Introduction to Neural Networks. [online] Available at: https://ujjwalkarn.me/2016/08/09/quick-intro-neural-networks/ [Accessed 29 May 2018].*