Yang Liu
3323643

Assignment

COMP6245
yl58n21@soton.ac.uk

1 Lab 1 - 5

I confirm that I have completed all five labs and uploaded reports. I also confirm that I didn't receive "Incomplete" in my feedback of Lab 1-2.

2 Non-Negative Matrix Factorization

2.1 Complete the code and verify

According to D.Lee and S.Seung's paper, W and H could be updated by equations below, where i is the iteration time.

$$H^{i+1} = H^i \frac{(W^i)^T * V}{(W^i)^T * W^i * H^i}$$

$$W^{i+1} = W^i \frac{V * (H^i)^T}{W^i * H^i * (H^i)^T}$$

The length of row and column of the matrix we use to decompose is 40 and 10 respectively. For each iteration, we use $||V - WH||$ to compute error, and for each rank $r = [1, 9]$, we compute the corresponding minimum error. **Figure 1** shows the convergence and minimum error of NMF with different rank r on a synthetic dataset. And from **Figure 1** we could find that the errors didn't converge when r=6 after 100000 iterations, which is supposed to be a fail.

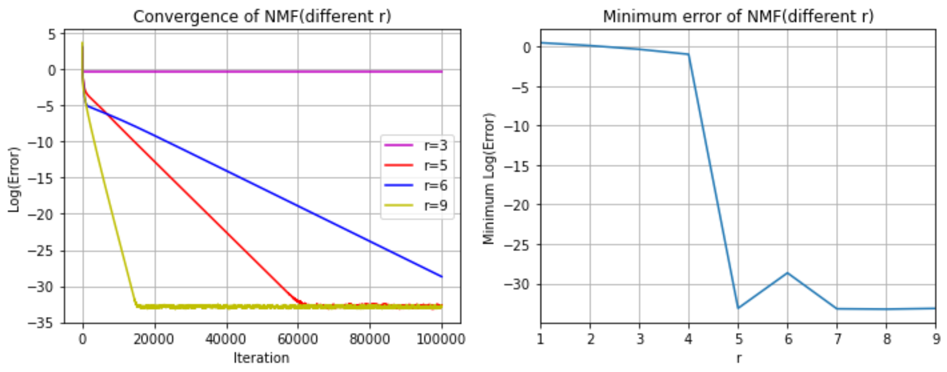


Figure 1: Convergence and minimum error of NMF

2.2 Comparison

By using NMF from sklearn.decomposition, we could easily compare our implementation results with sklearn. With the same matrix V that is used to decompose and same iteration 100000, **Figure 2** shows the result by using sklearn package. It is clear that our error is lower than sklearn thus our factorization result is better than sklearn.

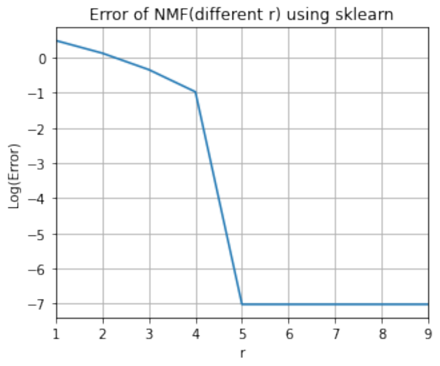


Figure 2: error of NMF using Sklearn

2.3 FTSE100

Figure 3 is plotted in order to observe the FTSE100 dataset.

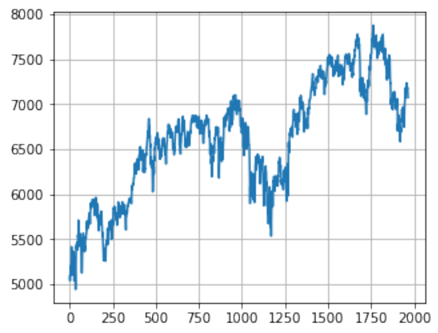


Figure 3: FTSE100

First we choose rank $r = 10$, and split FTSE100 and other 95 stocks price into training set and test set, where training set is first half of FTSE100, test set is second half of FTSE100. Then we set the $iteration = 15000$ and apply NMF to decompose the training set, finally use $||V - WH||$ to compute error of each iteration. **Figure 4** shows the convergence of error at each iteration.

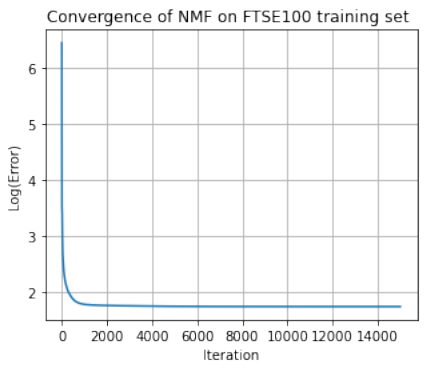


Figure 4: Convergence

Then we get the factors of interest which is W. After that we compute the most correlated factor with training set in W, combined with training set of stocks price, we use Lasso regression to find the 10 most correlative stocks and their weights. Then we use $CumulativeReturn = \frac{CurrentPrice - OriginalPrice}{OriginalPrice}$ to compute the cumulative return of investing FTSE100 and 10 stocks we selected in the test set. **Figure 5** shows the cumulative return we get by investing FTSE100 index and selected 10 stocks in the test set. It could be seen that the moving trends of two cumulative returns are roughly same, however the cumulative return we get from investing selected stocks is nearly always more than FTSE100 index. And in order to compare the result of different ranks we choose, we compute the cumulative returns when rank $r = 30$ and $r = 50$. **Figure 6** and **Figure 7** show cumulative returns when ranks are different.

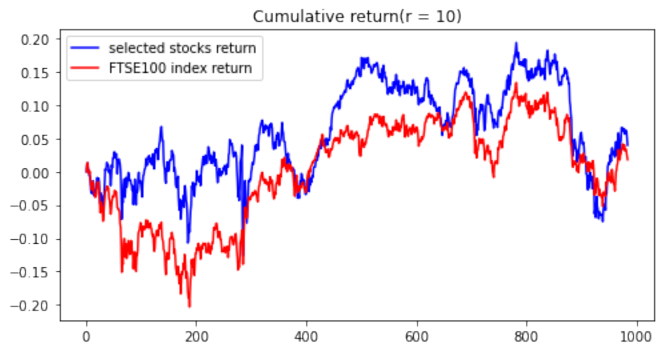


Figure 5: Cumulative return

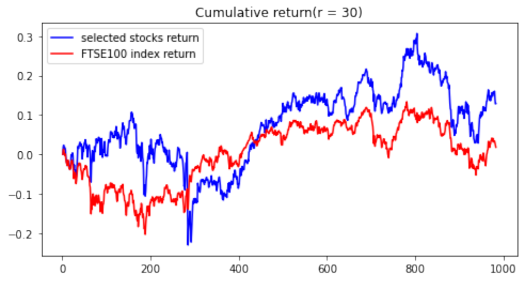


Figure 6: Cumulative return when r = 30

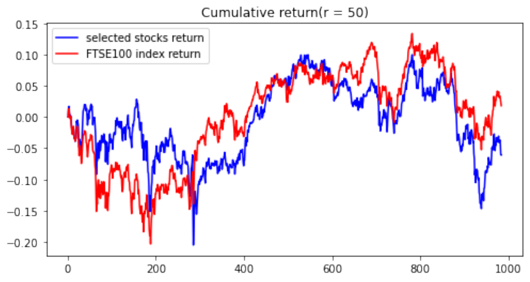


Figure 7: Cumulative return when r = 50

Finally we randomly choose equally weighted basket of 10 stocks to invest, and compute their cumulative return.**Figure 8** shows 3 different cumulative returns by investing FTSE100 index, 10 most correlative stocks and randomly selected stocks. In this case, cumulative return of randomly selected stocks is better than our selected stocks.

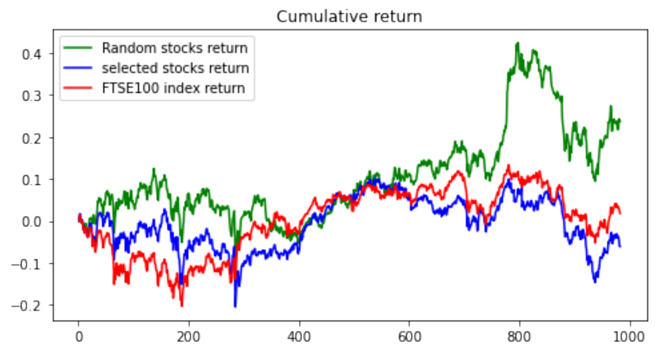


Figure 8: 3 different Cumulative returns

According to the results we get, NMF is a good way to find stocks that could best track the index, however it might not be a good way to get a better revenue, even in most of the cases, equally weighted 10 randomly selected stocks could do a better job than it.

3 Multi-Layer Perceptron

According to the code in the appendix, datasets of two different classification problems are generated by Gaussian distribution, one is relatively easy to classify while another is more difficult to classify because of the overlap. **Figure 9** shows the distributions of dataset of two problems.

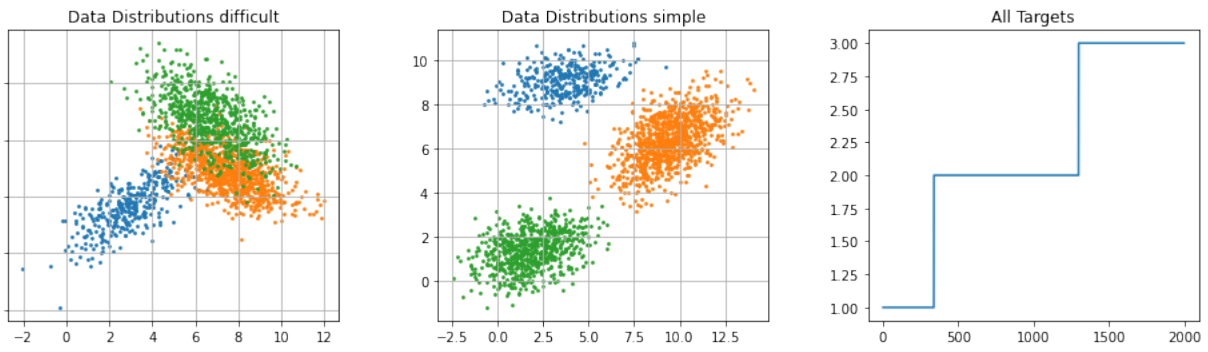


Figure 9: 2 different data distributions

Then we split the data into training set and test set by using $KFold$ from sklearn to implement ten-fold cross validation. Meanwhile, a Bayesian classifier($GaussianNB$ from *sklearn.naive_bayes*) and two MLP classifier(one contains 5 nodes in 1 hidden layer, another contains 200 nodes in 1 hidden layer) are implemented to compare their performance on two datasets generated previously. **Figure 10** shows their accuracy on the overlapping dataset, **Figure 11** shows their accuracy on the relatively far apart dataset. It is clear that all the classifiers achieve high accuracy on the dataset that classes are far apart while relatively worse performance are witnessed on the overlapping dataset. Among them all, MLP classifier with more nodes(MLP_test2 and MLP_train2) got the best accuracy on both datasets. The Naive Bayesian classifier performs well on the far apart dataset but not ideal on the overlapping dataset.

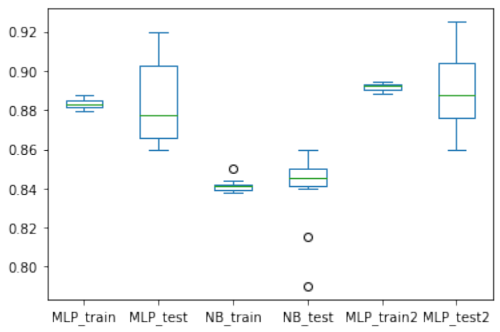


Figure 10: Accuracy(overlapping dataset)

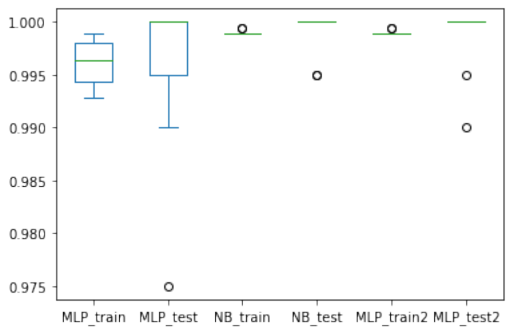


Figure 11: Accuracy(far apart dataset)

And we plot the class boundaries from the Bayesian classifier and two different MLP classifiers. **Figure 12** and **Figure 13** show class boundaries of two of partitions from two different datasets. When dealing with dataset that is not easy to classify due to overlap, class boundaries of three different classifiers are quite different. The class boundary of simpler classifier(with fewer nodes) is more rigid compared with the complex one(with more nodes) thus might lose some fitting ability. While NB classifier relies on the point of intersection in the middle, so the performance could be limited when dealing with a complex dataset.

For a simple dataset with nearly no overlap, three classifiers could easily find a decision boundary to classify three different classes with very few errors. And simpler MLP classifier and complex one could both find a class boundary that looks identical.

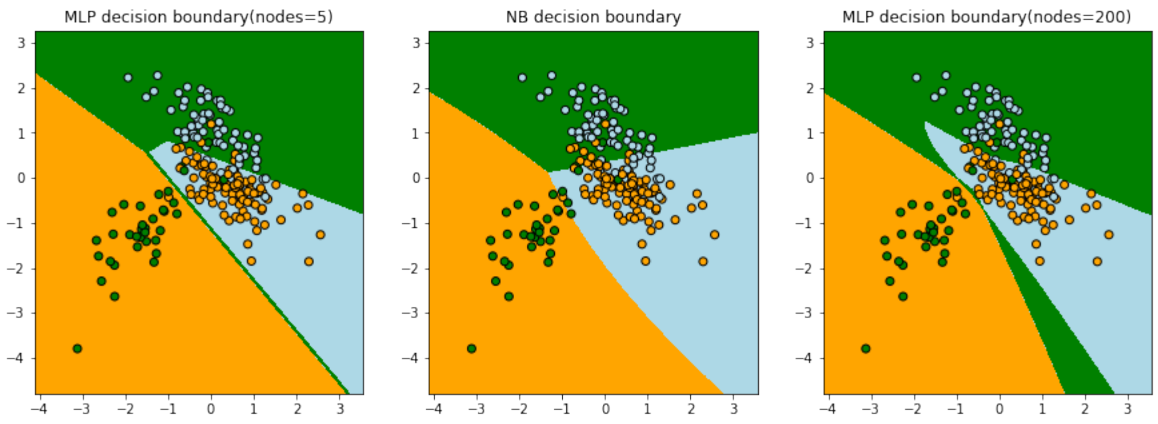


Figure 12: Class boundaries(overlapping dataset)

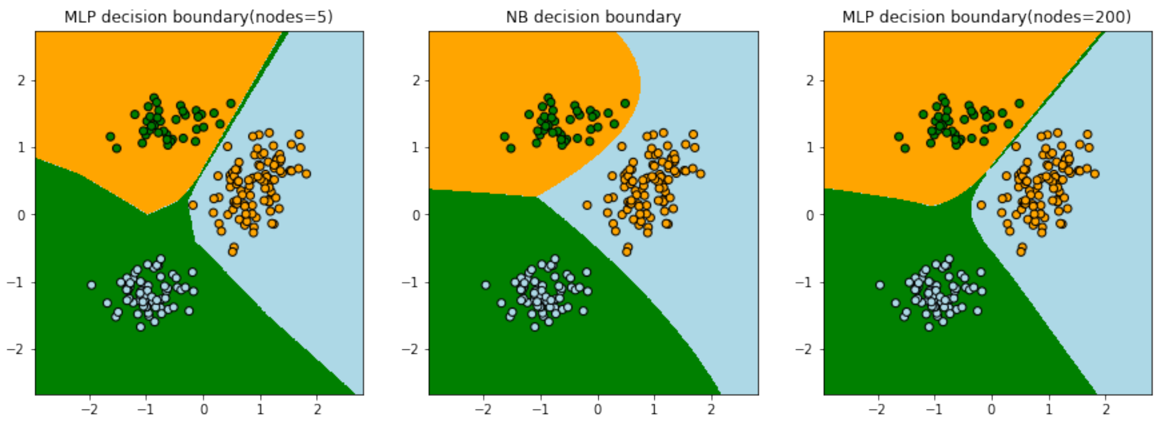


Figure 13: Class boundaries(far apart dataset)

In order to improve the performance of the model, we adjust some parameters to see the impact on the model.

When we are training a neural network, we care about the generalization performance of the model, which is the model not only performs well on training set but also on test test and in practice. Overfitting could be a problem when the model is trained in overmuch iterations, when the performance on training set is getting better, the error on test set is actually getting higher.

Early stopping is used to stop the training process when the performance of the model on a validation set starts to decline thus to prevent overfitting problem. **Figure 14** compares loss of the two different classifiers with and without early stopping on training set, and **Figure 15** shows validation scores of MLP classifier with early stopping in each iteration. The accuracy of models with and without early stopping on training set are 0.8799 and 0.8905 respectively, while on test set, the numbers are 0.87 and 0.8675. The result shows us that early stop does improve a little bit performance of model on the test set.

Meanwhile we find that after around 60 iterations, the loss declines much slower than before 50 iterations, which means it took more than twice as much time to train but ended up with no benefit, so early stopping is also a time saver.

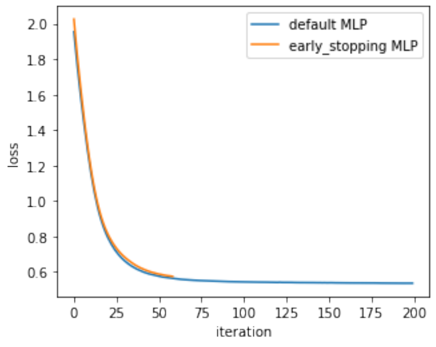


Figure 14: Error convergence

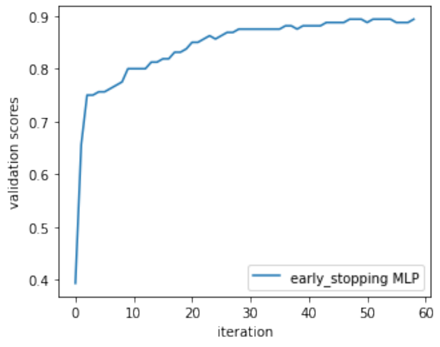
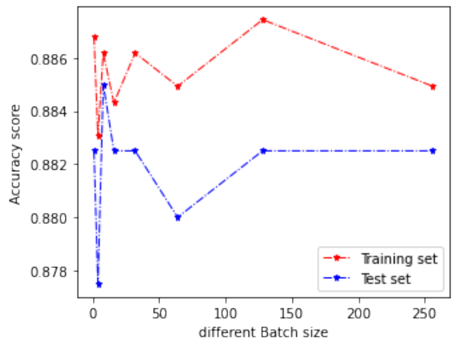


Figure 15: Validation scores

The batch size and learning rate are extremely important hyper parameters in neural network. At present, most deep learning models use gradient descent algorithm, and the weight of the model is updated by $W_{t+1} = W_t - \mu\frac{1}{n}\Sigma\nabla l(x, w_t)$, where μ is learning rate and n is batch size. So the batch size and learning rate directly influence the update of the weight, from this point of view they are the most important parameters. We adjust the initial learning rate from 0.1 to 0.0001 and batch size from 1 to 256 in order to find the better parameter values that maximise the performance on test set. **Figure 16** shows the accuracy scores of choosing different batch sizes, and **Figure 17** shows the accuracy scores of choosing different initial learning rate.



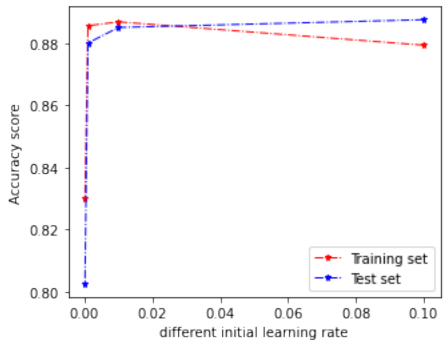


Figure 16: Accuracy on different batch size

Figure 17: Accuracy on different init learning rate

Finally, by changing the parameters of early stopping, learning rate and batch size, we trained a model that has the best performance on test set. When batch size is 32, initial learning rate is 0.001 and early stopping is on, the accuracy score is 0.89 while the default MLP classifier is just 0.88. **Figure 18** shows the convergence of error of each iteration.

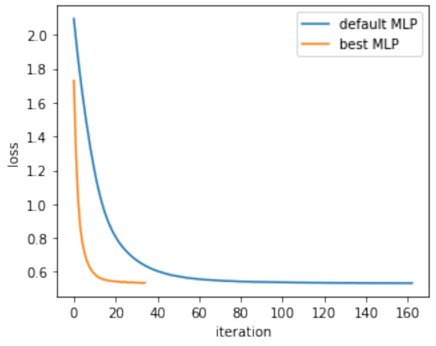


Figure 18: Accuracy