CSE514 Programming Assignment 2 Report

Name: Pingchuan Huang Student ID: 503954

Introduction

Description

In this assignment, I implemented and tested different classifiers including KNN, Decision Tree, Random Forest, Ada Boost, SVM, and ANN for **Letter Recognition**. The motivation is to find the optimal classifier for this problem, and the classifier is expected to be accurate enough to differentiate letters while maintain a acceptable speed for prediction.

Introduction to the Problem

The problem is simplified to binary classification problems, where 3 pairs of letters will be classified:

1. H and K
2. M and Y
3. U and V

Based on pure observation of the shape of each letter, I believe the first pair would be the easiest to classify as they looks most different, whereas letters from third pair would be slightly harder to classify since they looks quite similar to each other.

However, based on my test results, H and K is the hardest pair to be classified.

Discussion on Dimension Reduction

I don't think we need dimension reduction for this problem because the number of dimensions is not very high and after training turned out that the difference in time consumptions of whether dimension reduction is used are trivial. Moreover, the performance (accuracy) of models trained with 4-dimensional data degrades quite a lot (around 10%) compared to models trained with original samples. So considering the little time saved in training and predicting whereas the accuracy degradation is significant, I don't think applying dimension reduction is worthwhile.

In this assignment, time and accuracy are the main factors I considered in determining which method is good. And based on these metrics, I found the two wrapper feature selection methods, forward selection, and backward elimination, have the best performance among all 8 methods I used, they managed to reduce the training time for some models while maintaining a relatively higher accuracy.

Results

In order to speed up the training phase with no dimension reduction applied, I **standardized** the train samples using the MinMaxScaler and applied the corresponding scaling to the testing set. Since the scaling have no impact on the meaning of data, I used standardized data in dimension reduction as well.

All the images used and unused for in this section can be found at `./report/imgs/`.

KNN Classifier

Description

k-nearest neighbors algorithm (k-NN) is a supervised learning method, which take an input and finds the k nearest neighbors to the input, and vote the class based on these neighbors' label. So this algorithm is very easy to explain and takes almost no training time. However, It requires the storage of all the training data, and have to calculate the distance between the test samples and all the training data, which is very inefficient.

In this assignment, I used the algorithm from `sklearn.neighbors.KNeighborsClassifier`, and tuned 2 hyperparameters, which are `n_neighbors` and `algorithm`.

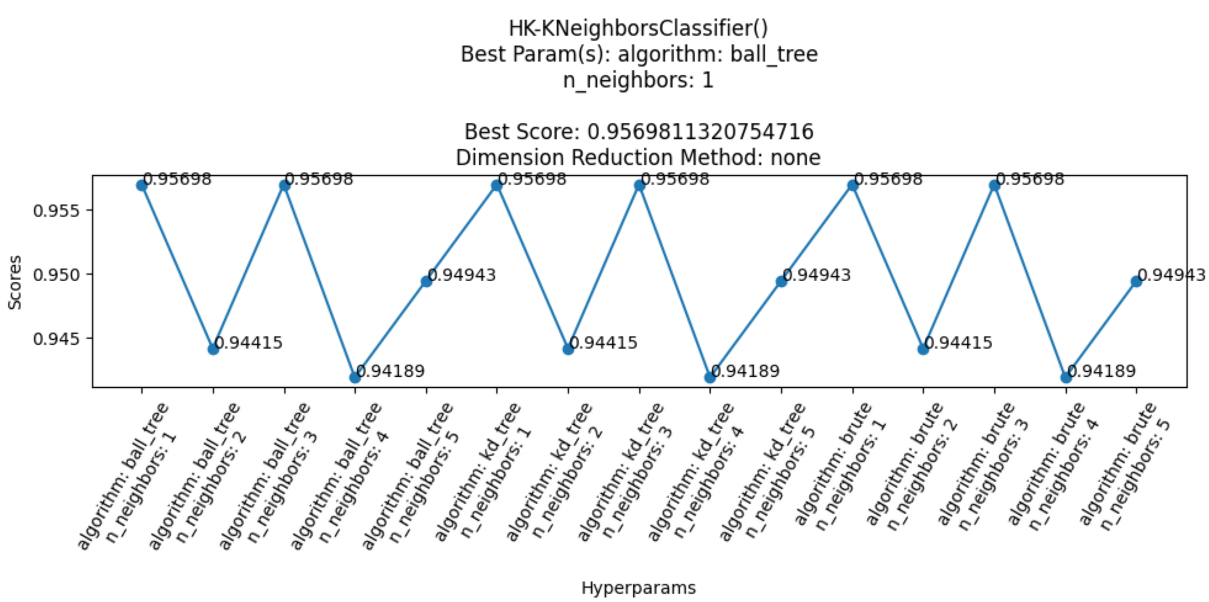
Cross Validation Results

I tuned 2 hyperparameters, which are `n_neighbors` and `algorithm`:

`n_neighbors = [1, 2, 3, 4, 5]`

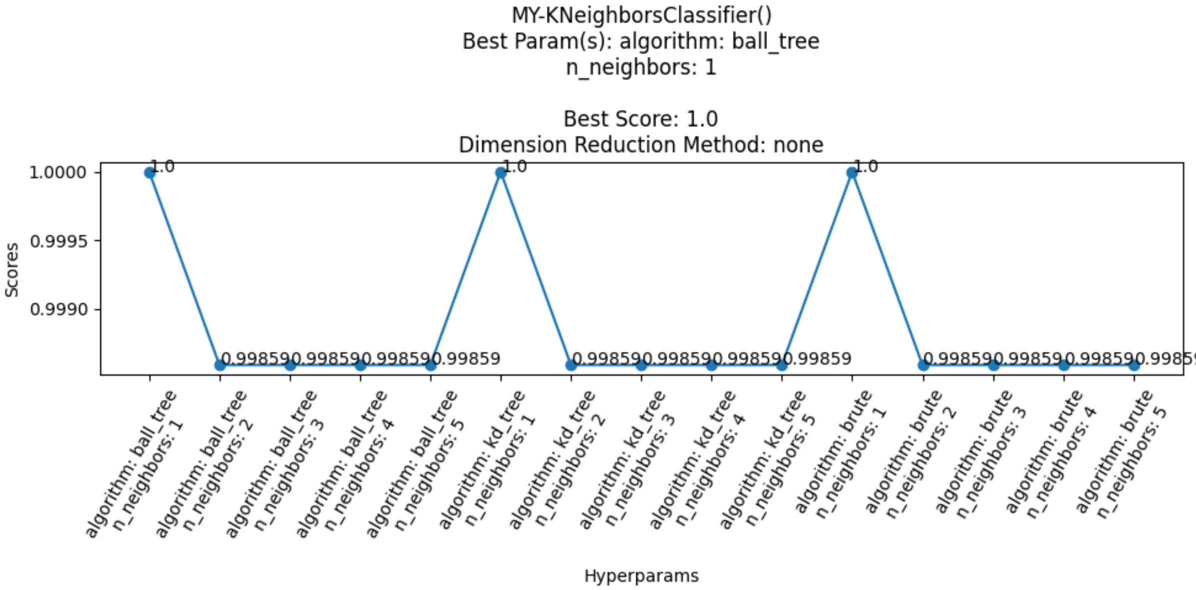
`algorithm = ['ball_tree', 'kd_tree', 'brute']`

Pair.1 - HK



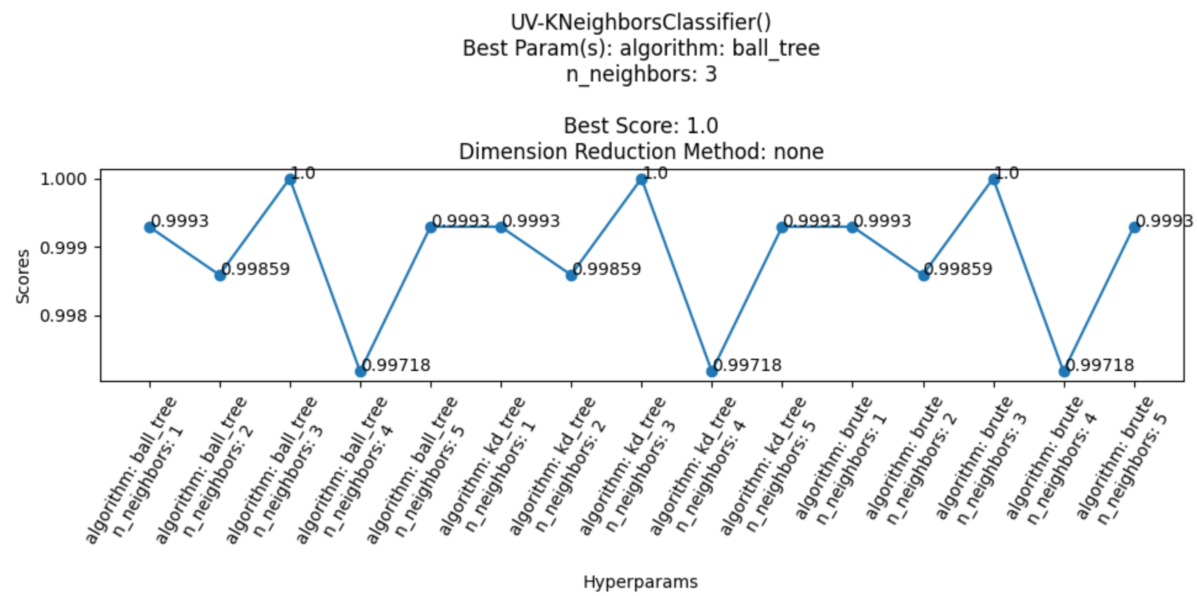
For pair 1, no matter which `algorithm` is used, the best score always comes from `n_neighbors = 1 or 3`.

Pair.2 - MY



For pair 2, regardless of `algorithm`, `n_neighbors = 1` always produce the best outcome.

Pair.3 - UV



For pair3, again only `n_neighbors` has impact on the score of cross-validation, and the best performance comes from using `n_neighbors = 3`.

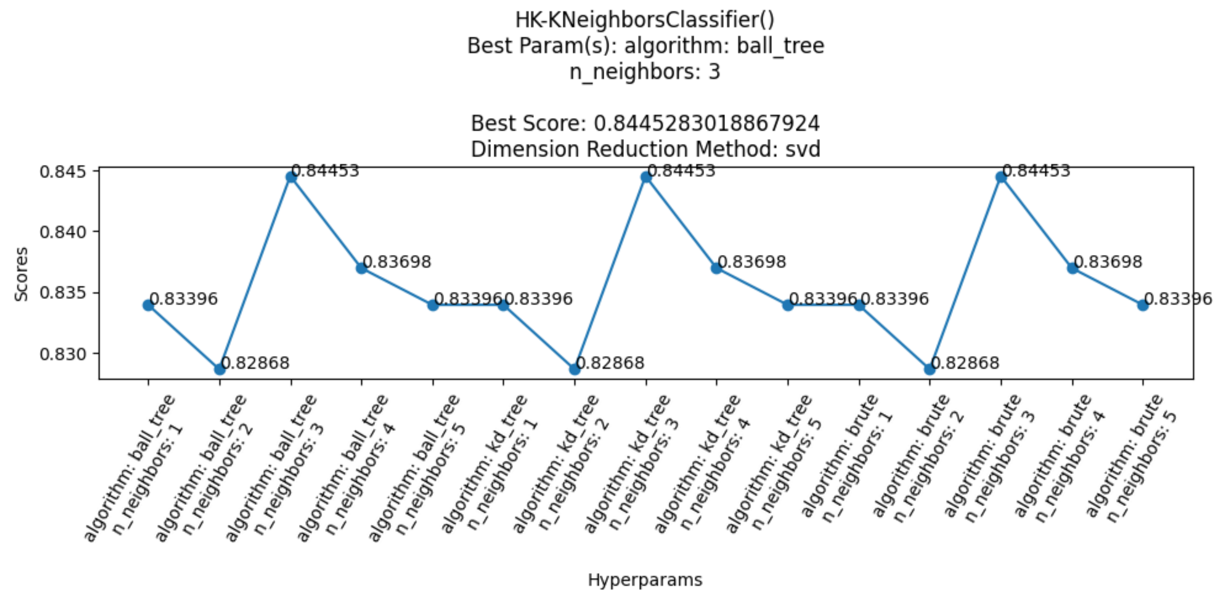
Description of the Dimension Reduction Method (SVD, Backward Feature Elimination)

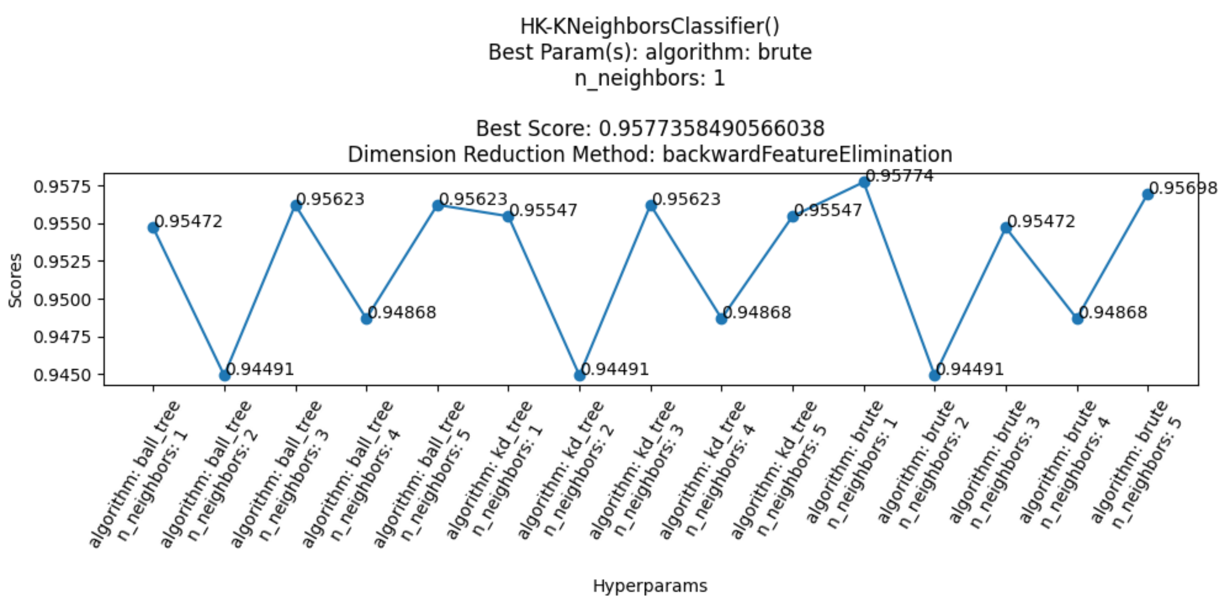
I applied 8 dimensional reduction methods (Forward Feature Selection, Backward Feature Elimination, Random Forest, Decision Tree, Lasso Regression, PCA, SVD, and NMF) to the training set and applied the same reduction to the testing set.

To reduce the length of the report (8 methods * 3 pairs = 24 images are way too many), I will only talk about 2 reduction methods, `Backward Feature Elimination` and `SVD`, in the following section. Other graphs can be accessed from the folder I submitted to GradeScope.

Cross Validation Results with Dimension Reduction

Pair.1 - HK

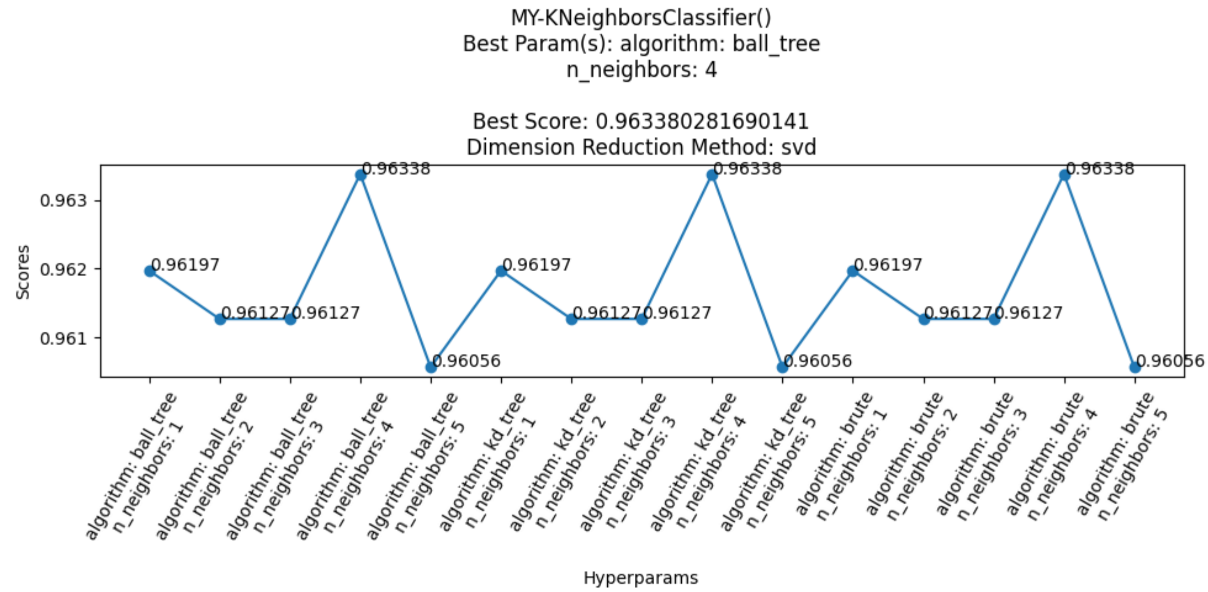


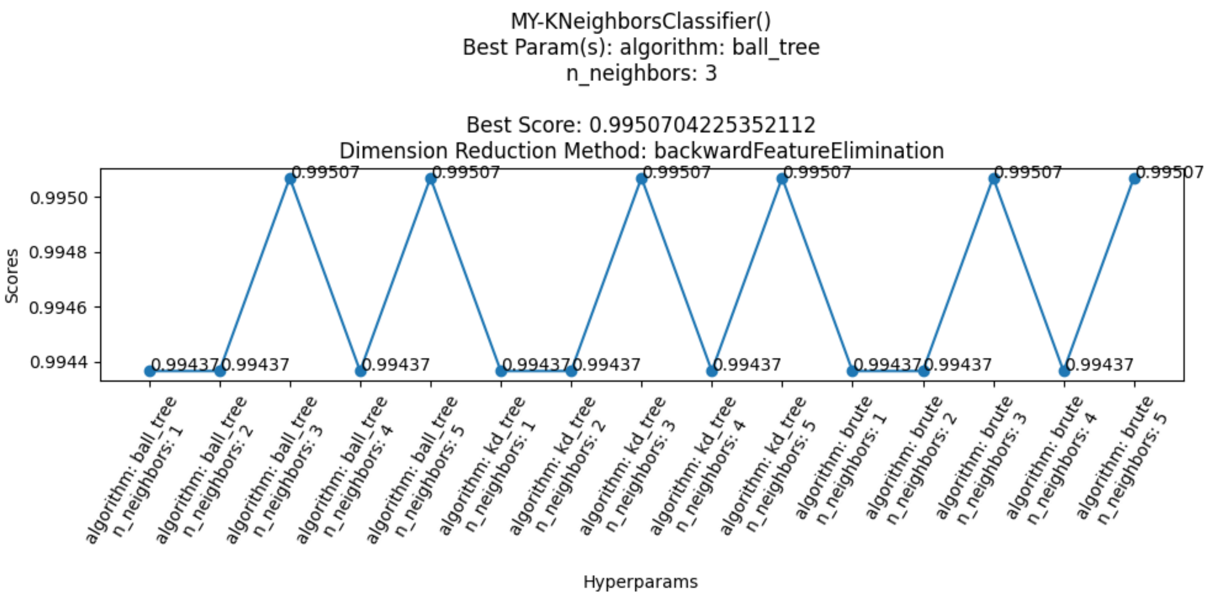


When using SVD as the dimension reduction method, the performance degrades quite a lot, and the best combination of hyperparameters are `n_neighbors = 3` no matter which `algorithm` is used.

In comparison, backward feature selection achieves relatively better performance, and the best hyperparameters are `n_neighbors = 3, algorithm = brute`.

Pair.2 - MY

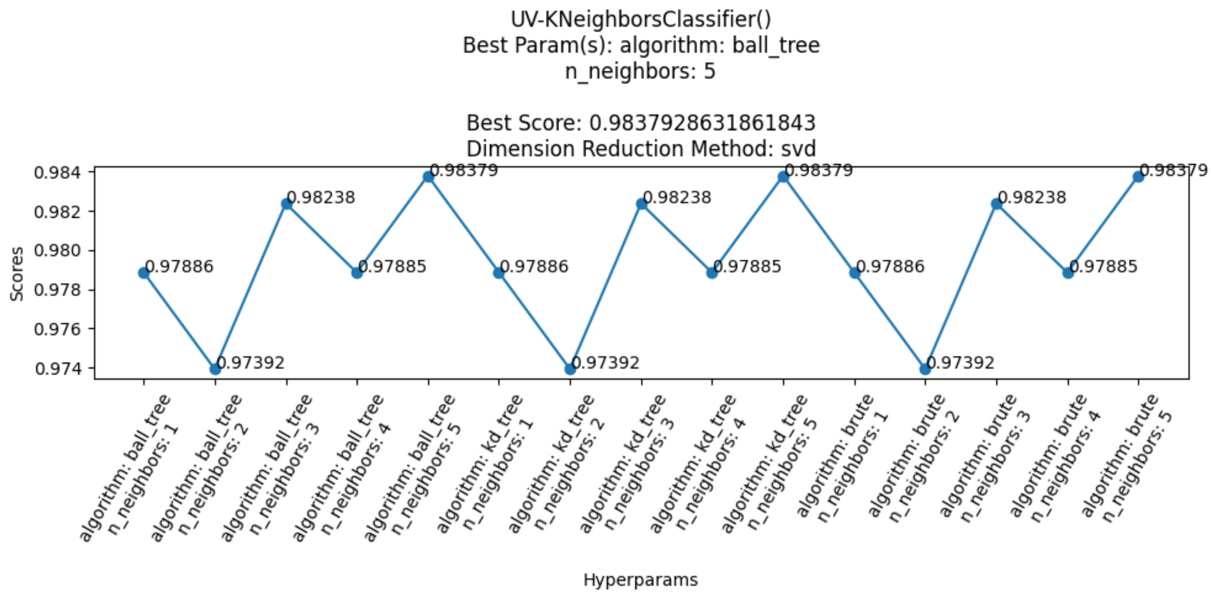


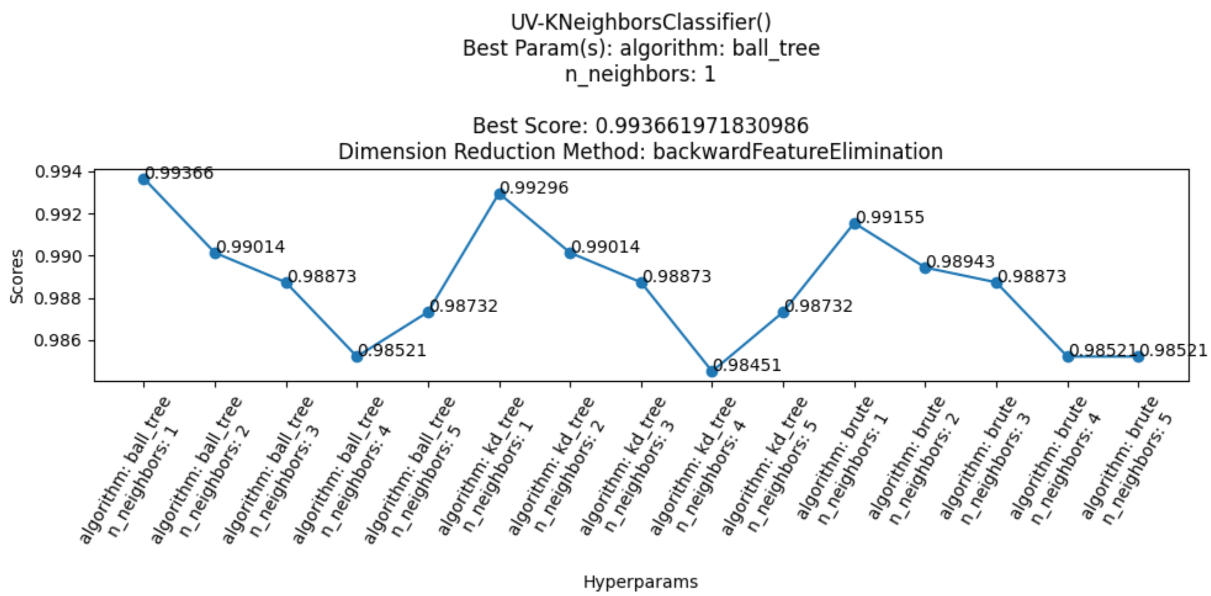


For pair 2, when using SVD as the dimension reduction method, the performance degrades but not as severe as that of HK, and the best combination of hyperparameters are `n_neighbors = 4` no matter which `algorithm` is used.

In comparison, backward feature selection achieves relatively better performance, and the best hyperparameter is `n_neighbors = 3 or 5`.

Pair.3 - UV





For pair3, again only `n_neighbors` has impact on the score of cross-validation, and the best performance comes from using `n_neighbors = 3`.

Decision Tree Classifier

Description

Decision Tree is a tree like classification model that has multiple layers and splits representing different decision makings (classifications). It is easy to apply since each chosen split can be understood and checked by a human user during the prediction process. However, it can easily overfit by splitting over and over again.

In this assignment, I used the algorithm from `sklearn.tree.DecisionTreeClassifier`, and tuned 2 hyperparameters, which are `max_depth` and `max_features`.

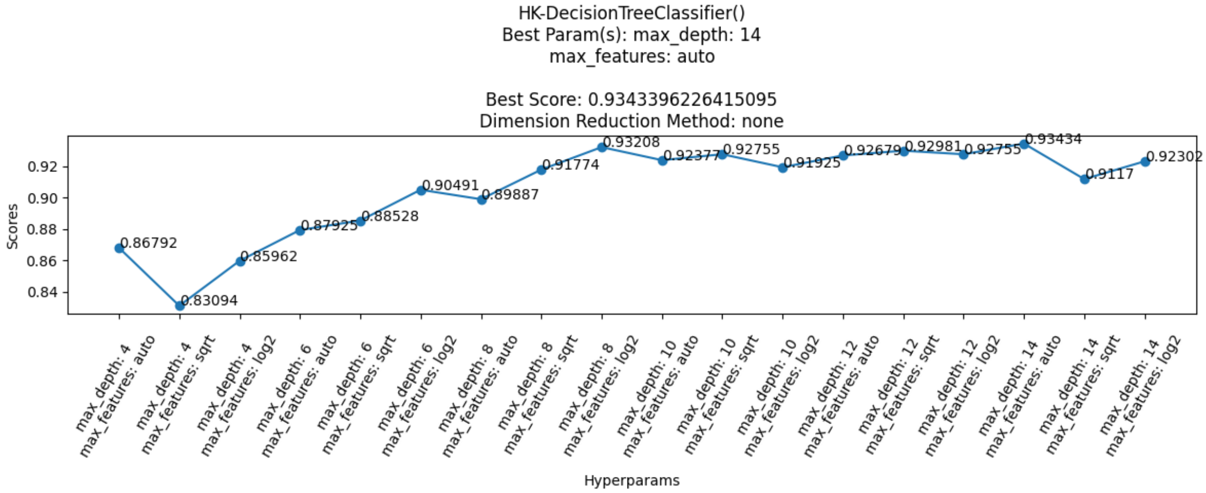
Cross Validation Results

I tuned 2 hyperparameters, which are `max_depth` and `max_features`:

`max_depth = [4, 6, 8, 10, 12, 14]`

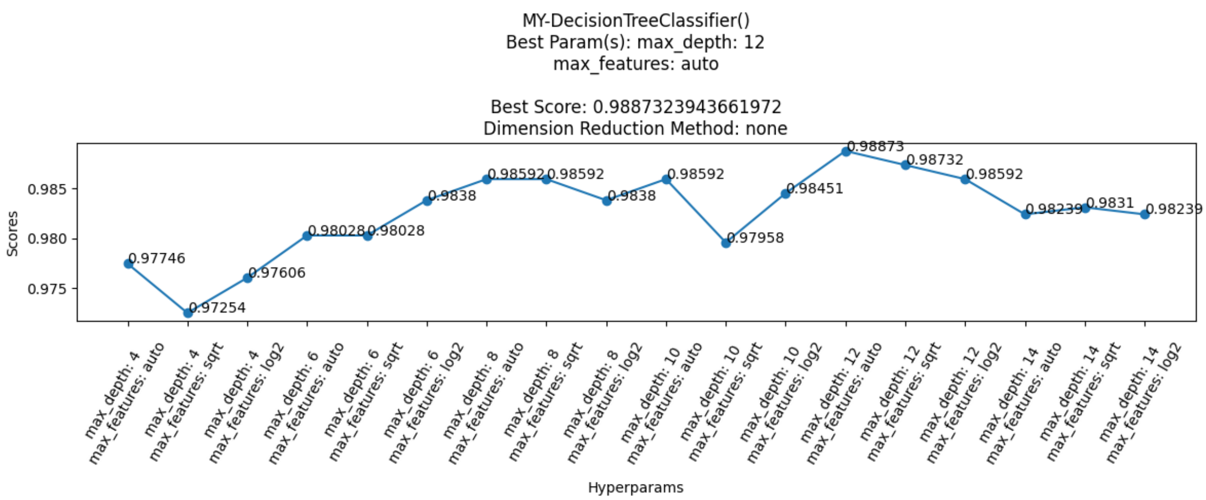
`max_features = ['auto', 'sqrt', 'log2']`

Pair.1 - HK



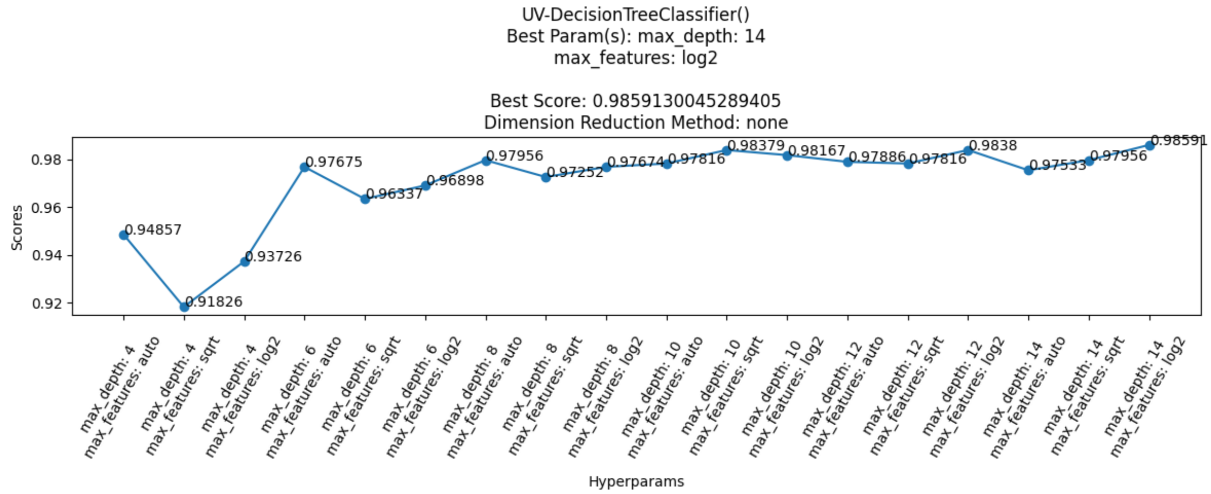
For pair 1, the score becomes consistent starting from `max_depth = 8` and it reaches the best performance when `max_depth = 14, max_features = 'auto'`.

Pair.2 - MY



For pair 2, it shares the same trend with pair 1 and reaches the best performance when `max_depth = 12, max_features = auto`.

Pair.3 - UV



For pair 3, the score becomes consistent starting from `max_depth = 6`, which is earlier than pair 1 which may reflect the fact that pair 3 is easier to classify, and it reaches its best performance when `max_depth = 14, max_features = 'log2'`.

Description of the Dimension Reduction Method (Decision Tree, Random Forest)

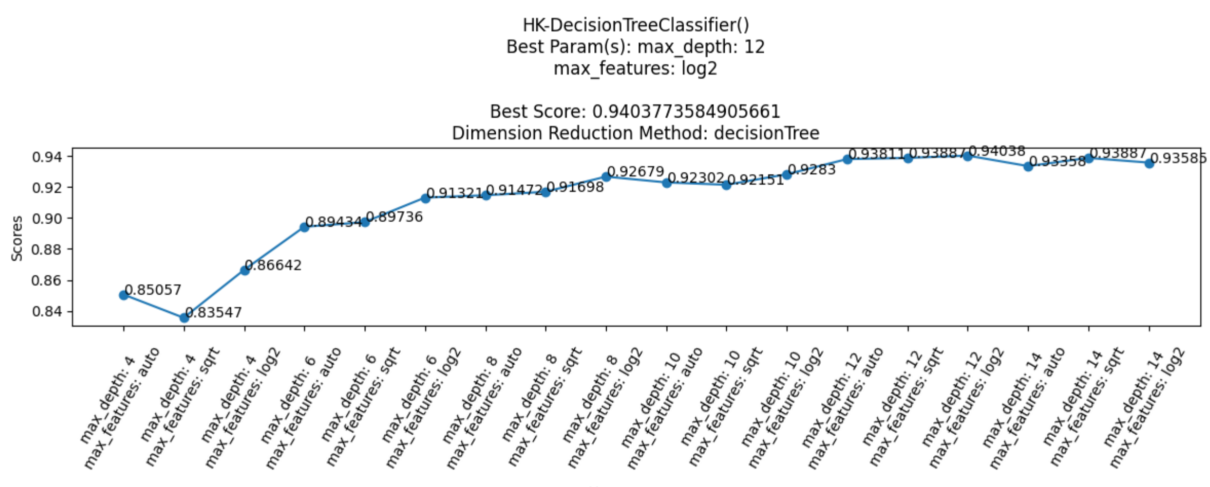
I applied 8 dimensional reduction methods (Forward Feature Selection, Backward Feature Elimination, Random Forest, Decision Tree, Lasso Regression, PCA, SVD, and NMF) to the training set and applied the same reduction to the testing set.

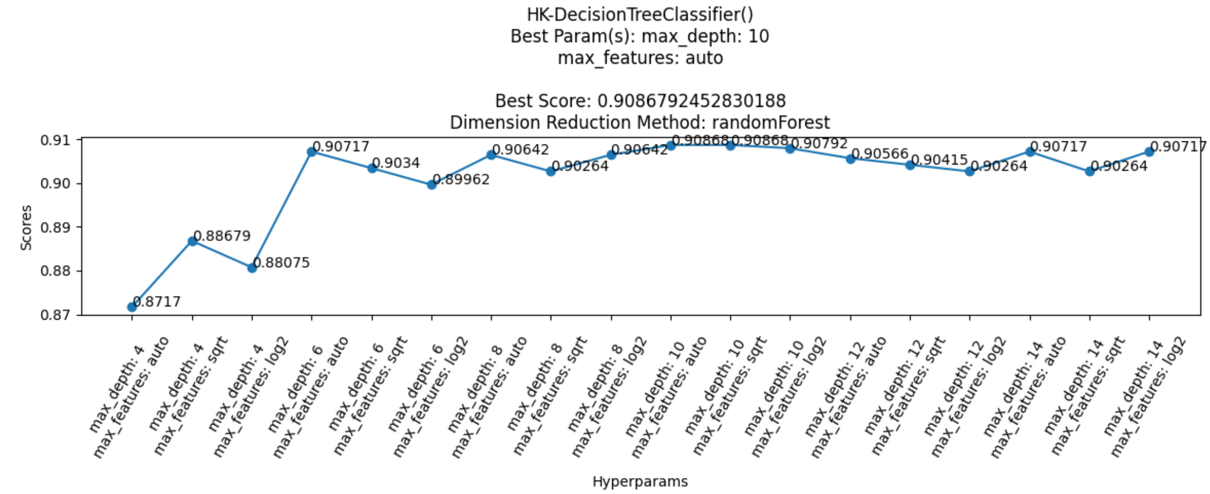
To reduce the length of the report (8 methods * 3 pairs = 24 images are way too many), I will only talk about 2 reduction methods, `Decision Tree` and `Random Forest`, in the following section. Other graphs can be accessed from the folder I submitted to GradeScope.

It is observed that after dimension reduction, this the best hyperparameters of this model requires lower max_depth.

Cross Validation Results with Dimension Reduction

Pair.1 - HK

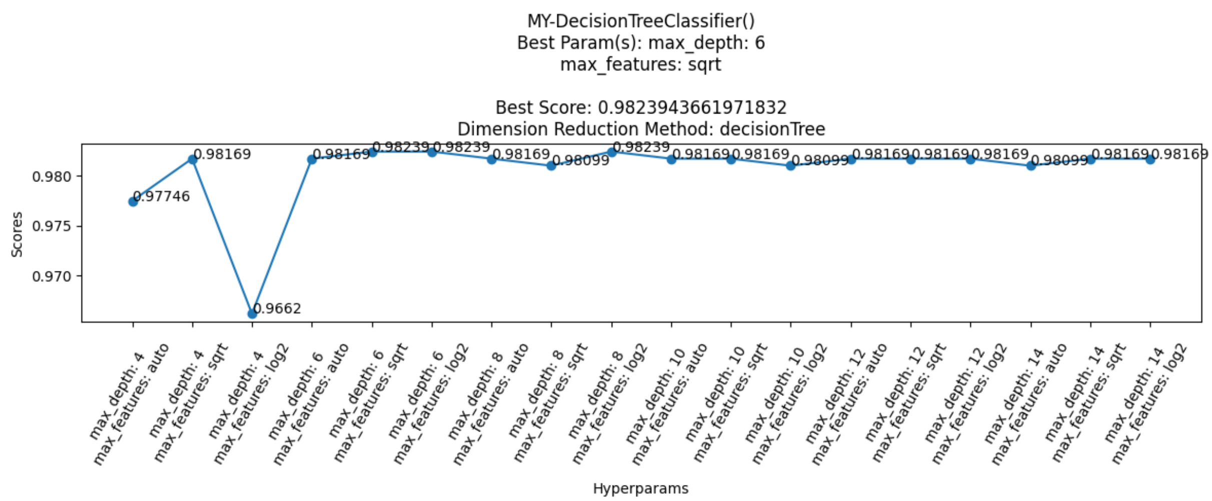


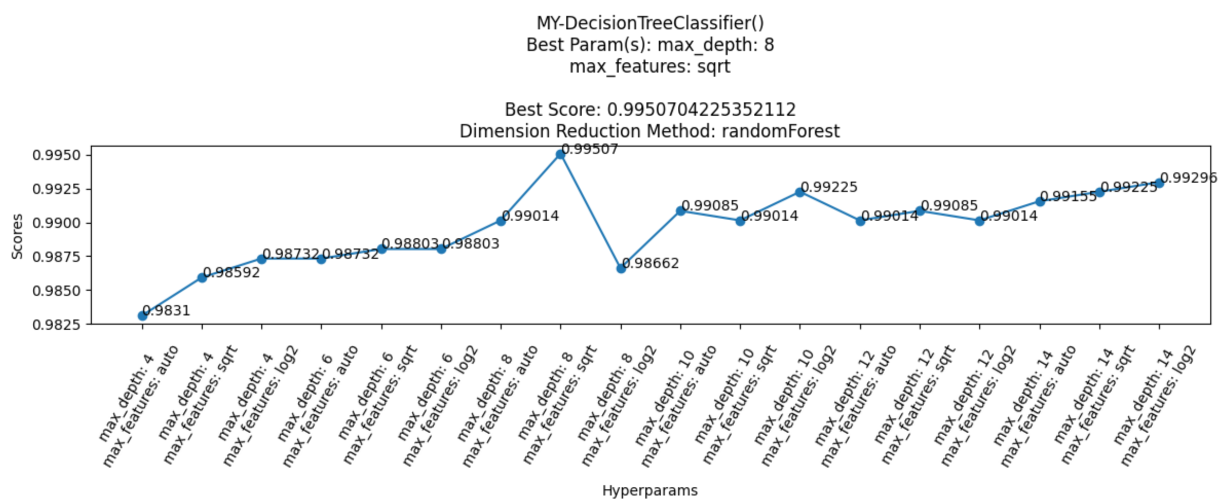


When using decision tree as the dimension reduction method, the performance actually improved by 1%, and the best combination of hyperparameters are `max_depth = 12, max_features = 'log2'`.

In comparison, random forest slightly makes the performance worse, and the best hyperparameters are `max_depth = 10, max_features = auto`.

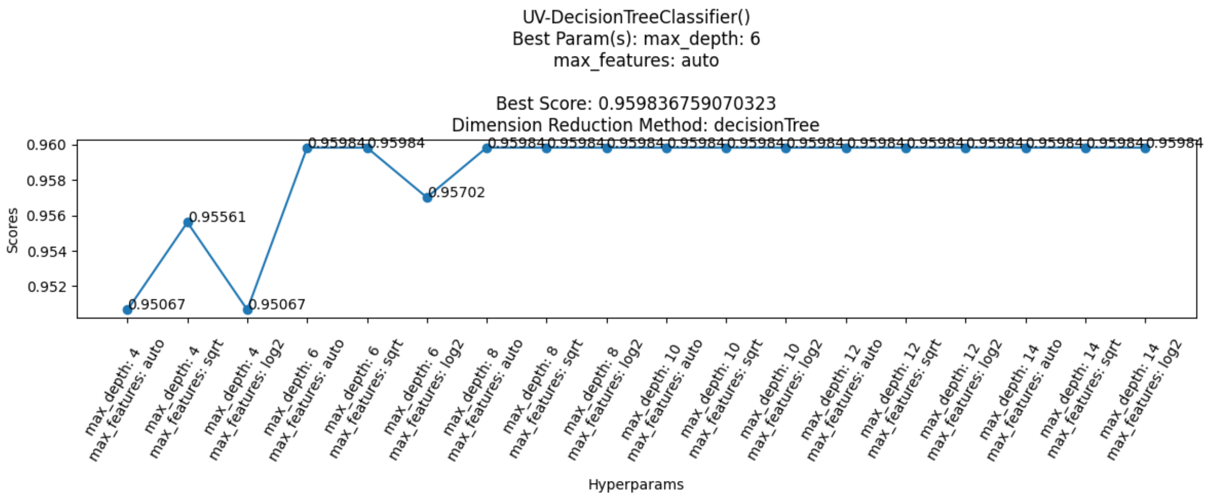
Pair.2 - MY

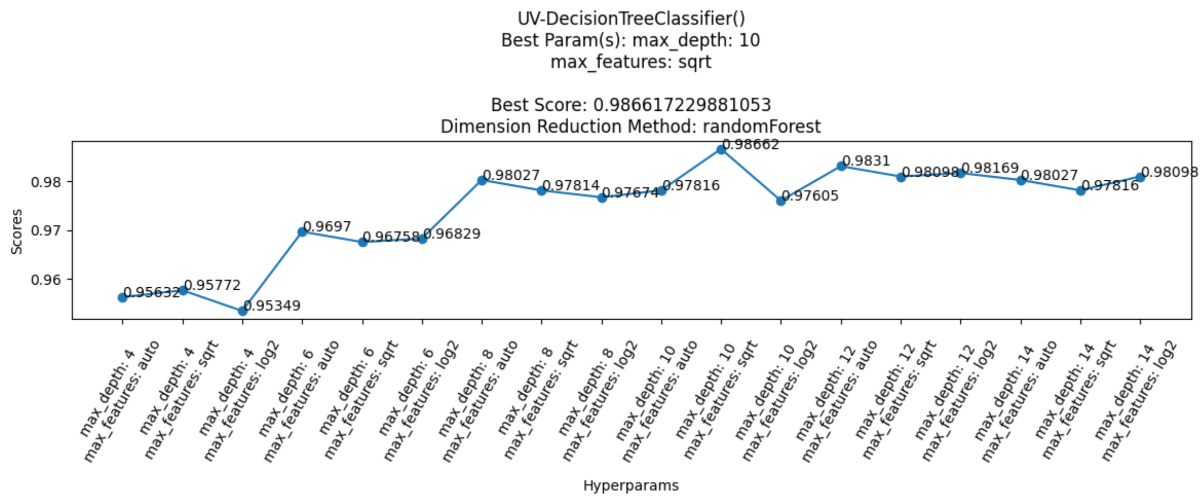




For pair 2, model trained from that processed by random forest has slightly better performance over 0.99, and both of them uses `max_features = 'sqrt'`, whereas the former uses `max_depth = 6` and the latter uses `max_depth = 8`.

Pair.3 - UV





For pair3, they uses `max_depth = 6, max_features = 'auto'` and `max_depth = 10, max_features = 'sqrt'` respectively.

Random Forest Classifier

Description

Random Forest utilzes multiple trees to train the model, so that it can have high accuracy while avoiding overfitting like a single decision tree. However, it takes longer to train than a single decision tree.

In this assignment, I used the algorithm from `sklearn.ensemble.RandomForestClassifier`, and tuned 2 hyperparameters, which are `max_depth` and `n_estimators`.

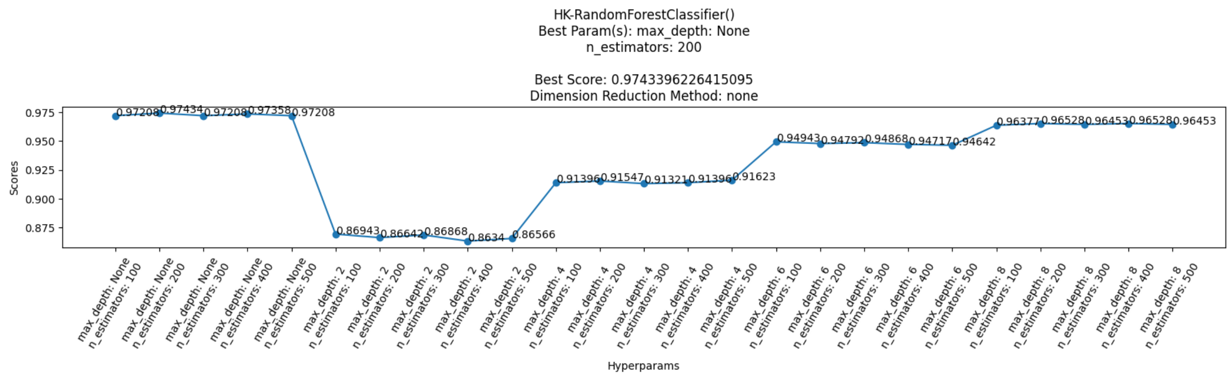
Cross Validation Results

I tuned 2 hyperparameters, which are `max_depth` and `n_estimators`:

`n_estimators = [100, 200, 300, 400, 500]`

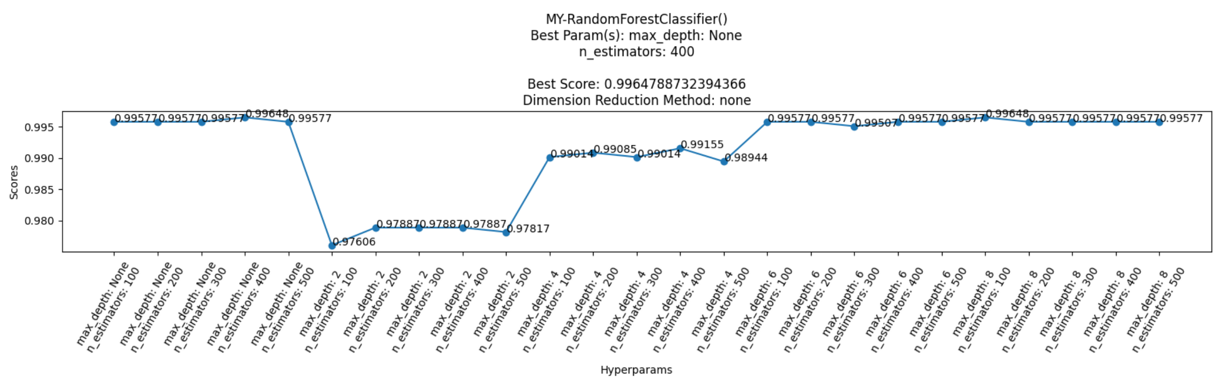
`max_depth = [None, 2, 4, 6, 8]`

Pair.1 - HK



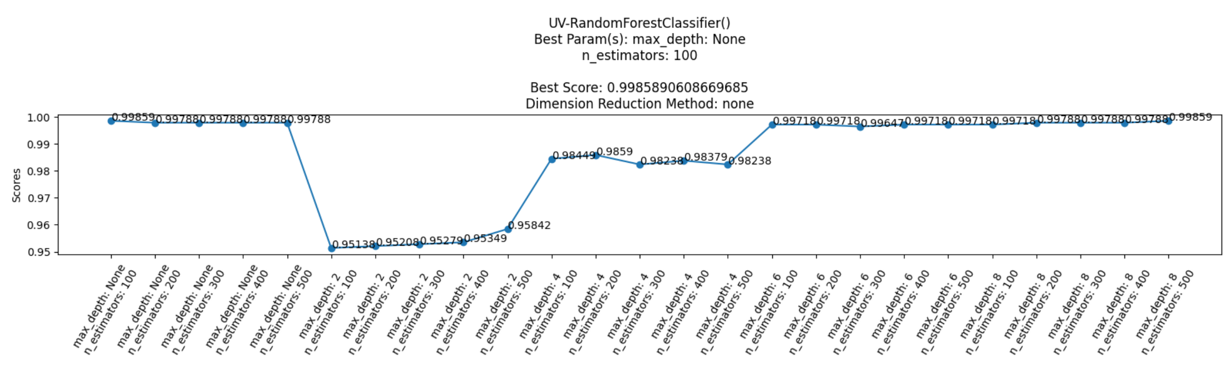
For pair 1, the tuned hyperparameters are `max_depth = None, n_estimators = 200`.

Pair.2 - MY



For pair 2, it shares the same trend with pair 1 and reaches the best performance when `max_depth = 400, n_estimators= 400`.

Pair.3 - UV



For pair 3, the best hyperparameters are `max_depth = None, n_estimators = 100`.

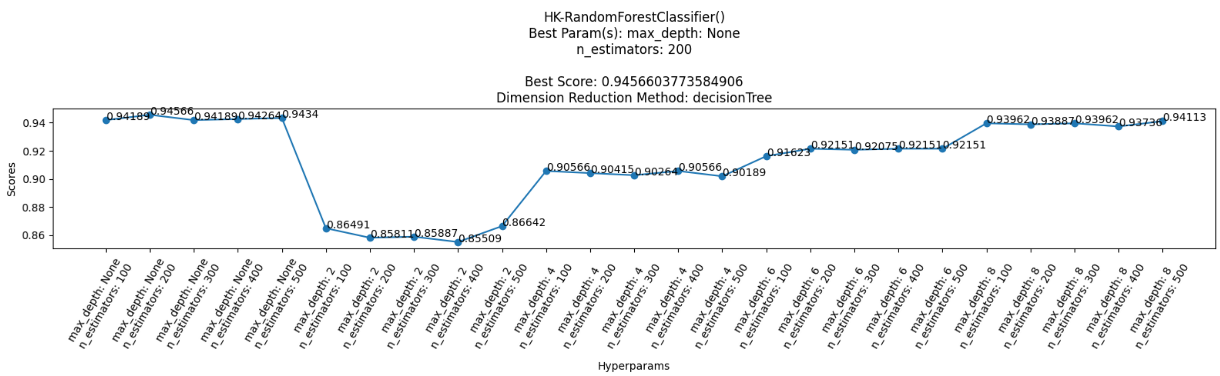
Description of the Dimension Reduction Method (Decision Tree, Forward Feature Selection)

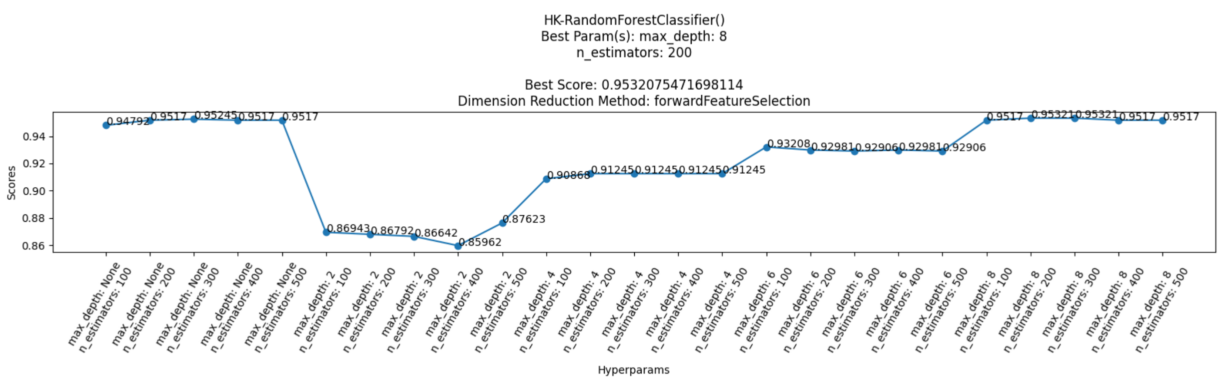
I applied 8 dimensional reduction methods (Forward Feature Selection, Backward Feature Elimination, Random Forest, Decision Tree, Lasso Regression, PCA, SVD, and NMF) to the training set and applied the same reduction to the testing set.

To reduce the length of the report (8 methods * 3 pairs = 24 images are way too many), I will only talk about 2 reduction methods, `Decision Tree` and `Forward Feature Selection`, in the following section. Other graphs can be accessed from the folder I submitted to GradeScope.

Cross Validation Results with Dimension Reduction

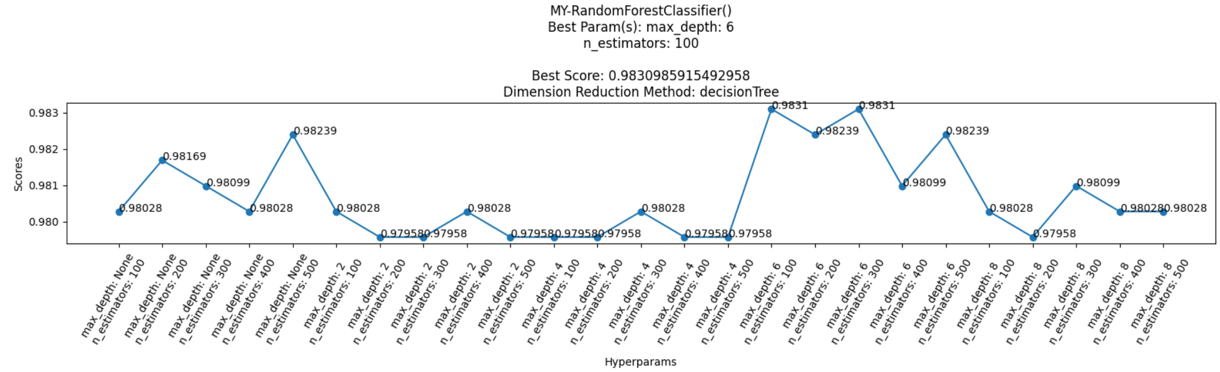
Pair.1 - HK

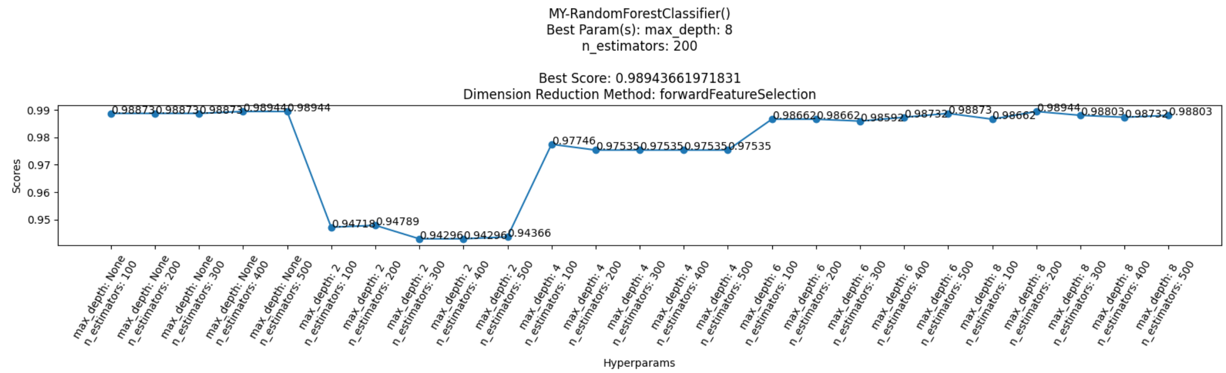




Both dimension reduction methods make the cross-validation score become lower. Their best hyperparameters are `max_depth = None, n_estimators = 200` and `max_depth = 8, n_estimators = 200` respectively.

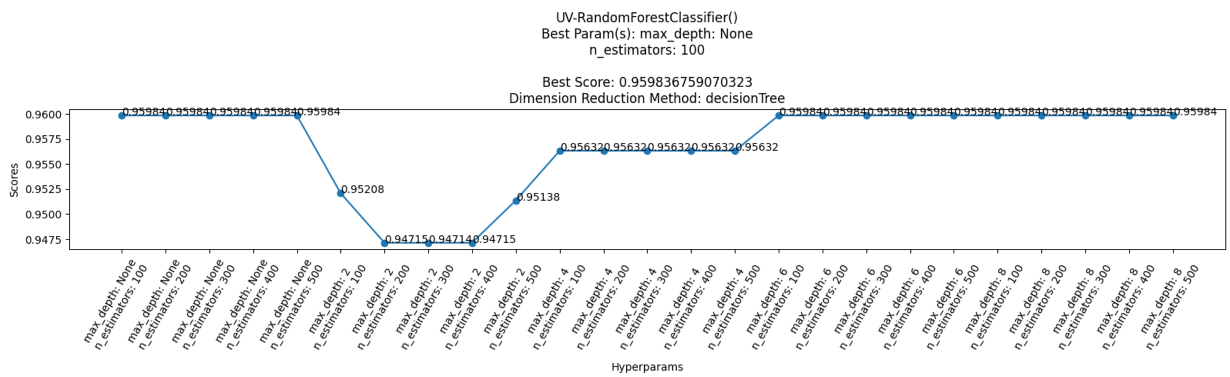
Pair.2 - MY

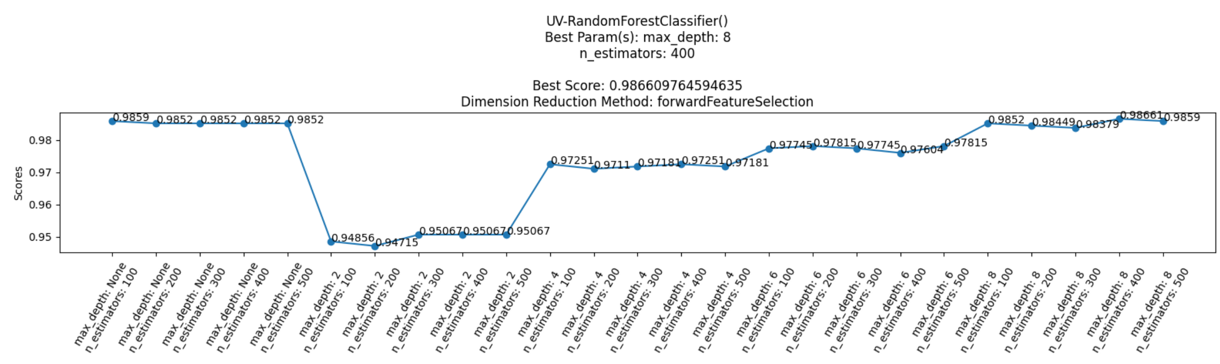




Seems pair 2 is easy to train, as they have pretty high score even with dimension reduction applied - with decision tree, the best hyperparameters are `max_depth = 6, n_estimators = 100`, and for the latter are `max_depth = 8, n_estimators = 200`.

Pair.3 - UV





For pair3, they uses `max_depth = None, n_estimators = 100` and `max_depth = 8, n_estimators = 400` respectively.

Ada Boost Classifier

Description

An AdaBoost classifier is a meta-estimator that begins by fitting a classifier on the original dataset and then fits additional copies of the classifier on the same dataset but where the weights of incorrectly classified instances are adjusted such that subsequent classifiers focus more on difficult cases. (Credit: **https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html**) And it is easier to use with less need for tweaking parameters unlike algorithms like SVM. However, AdaBoost is also extremely sensitive to Noisy data and outliers so additional processes might be required.

In this assignment, I used the algorithm from `sklearn.ensemble.AdaBoostClassifier`, and tuned 2 hyperparameters, which are `learning_rate` and `n_estimators`.

Cross Validation Results

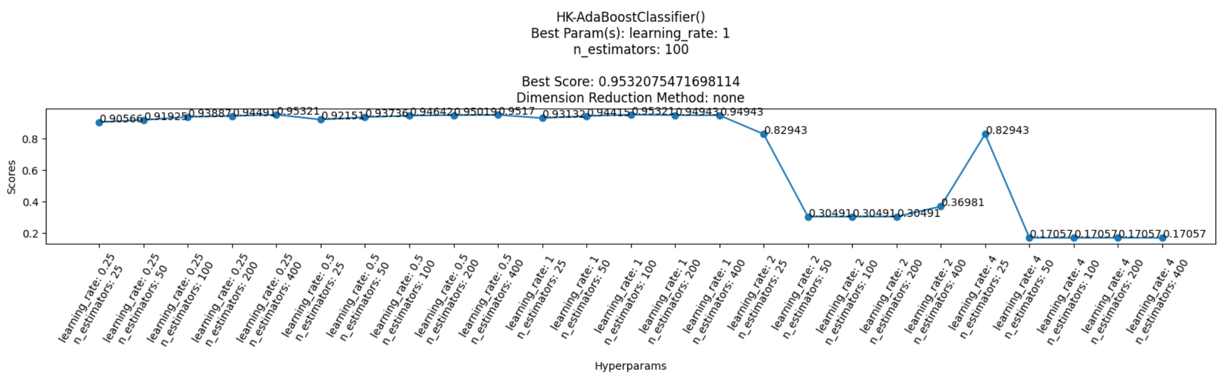
I tuned 2 hyperparameters, which are `learning_rate` and `n_estimators`:

`n_estimators = [25, 50, 100, 200, 400]`

`learning_rate = [.25, .5, 1, 2, 4]`

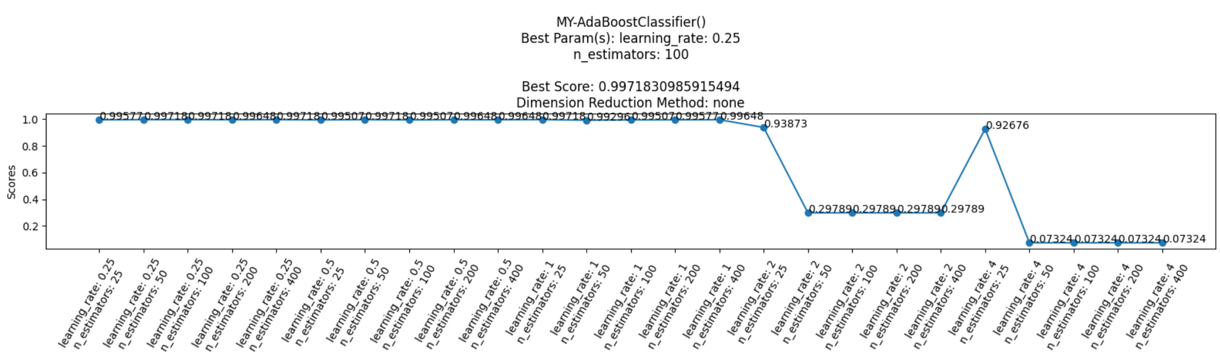
Based on the following graphs, we learnt that learning rate larger than 1 might not be a good idea.

Pair.1 - HK



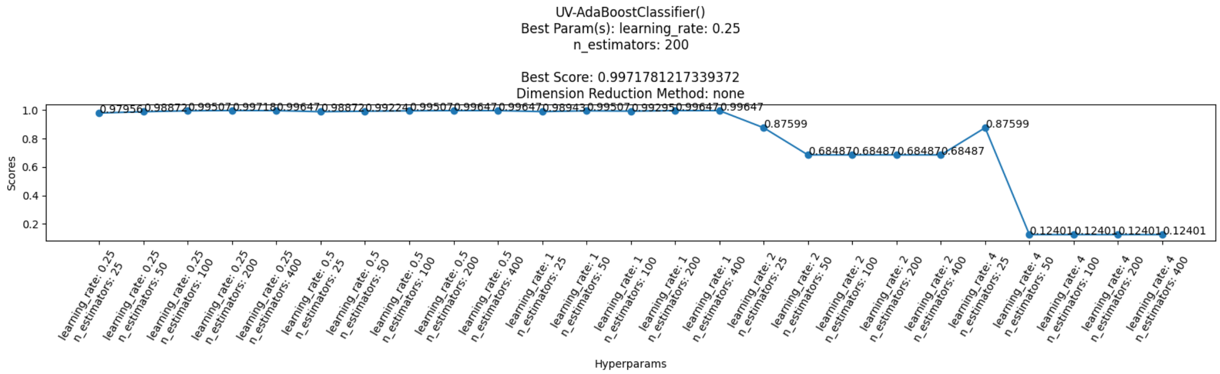
For pair 1, the best hyperparameters are `learning_rate = 1, n_estimators = 100`.

Pair.2 - MY



For pair 2, the optimal hyperparameters are `learning_rate = 0.25, n_estimators= 100`. But the score is almost the same when learning rate is less and equal than 1.

Pair.3 - UV



For pair 3, the choosen hyperparameters are `learning_rate = 0.25, n_estimators = 200`. But the score is almost the same when learning rate is less and equal than 1.

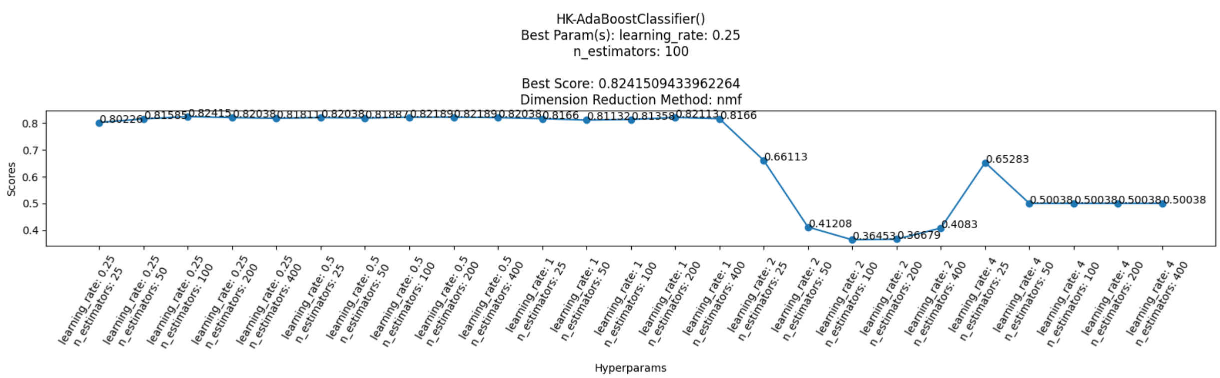
Description of the Dimension Reduction Method (NMF, PCA)

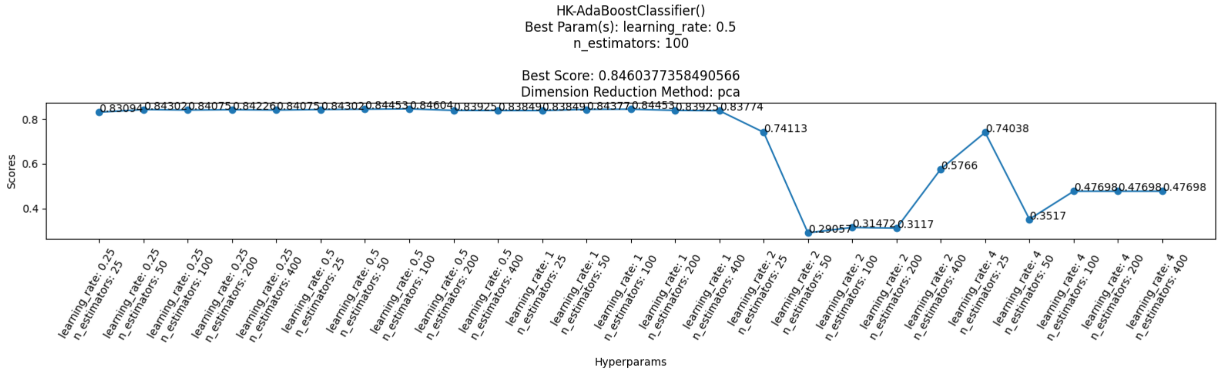
I applied 8 dimensional reduction methods (Forward Feature Selection, Backward Feature Elimination, Random Forest, Decision Tree, Lasso Regression, PCA, SVD, and NMF) to the training set and applied the same reduction to the testing set.

To reduce the length of the report (8 methods * 3 pairs = 24 images are way too many), I will only talk about 2 reduction methods, `NMF` and `PCA`, in the following section. Other graphs can be accessed from the folder I submitted to GradeScope.

Cross Validation Results with Dimension Reduction

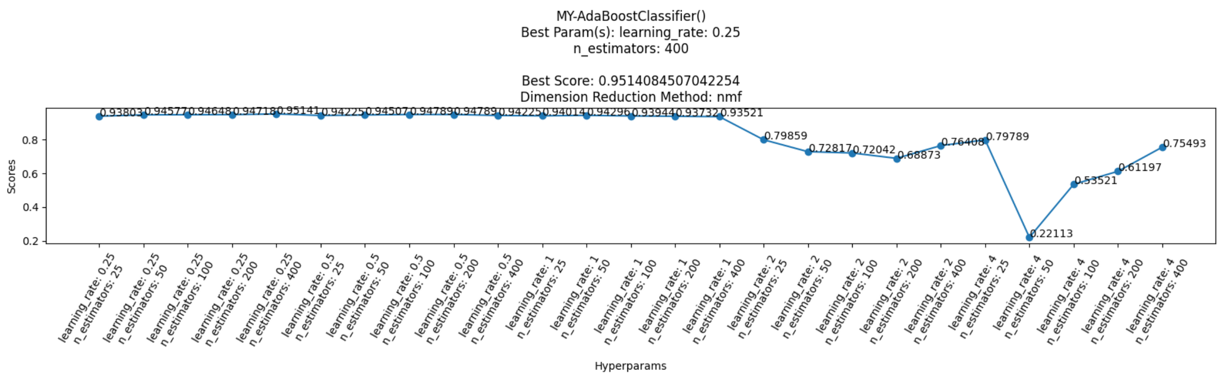
Pair.1 - HK

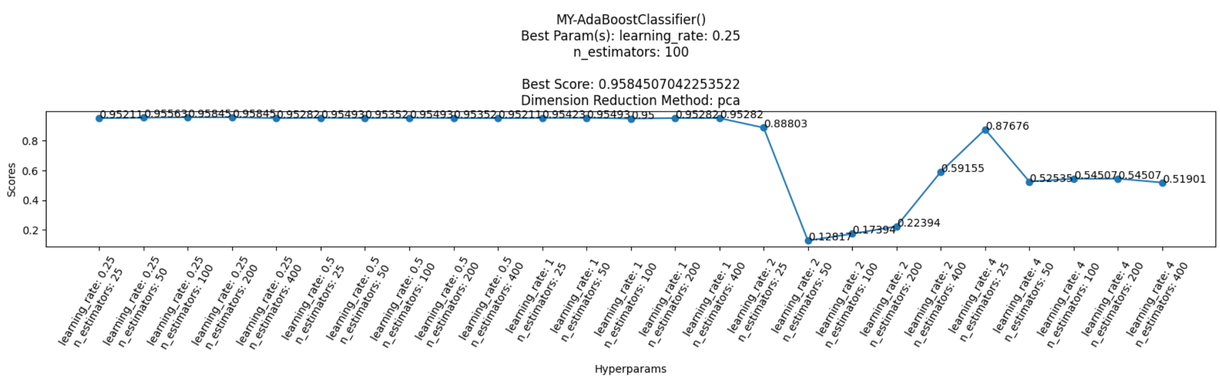




Again, HK are the pair that is hardest to classify, and the dimension reduction made the case even worse. Here the best hyperparameters are `learning_rate = 0.25, n_estimators = 100` and `learning_rate = 0.5, n_estimators = 100` respectively.

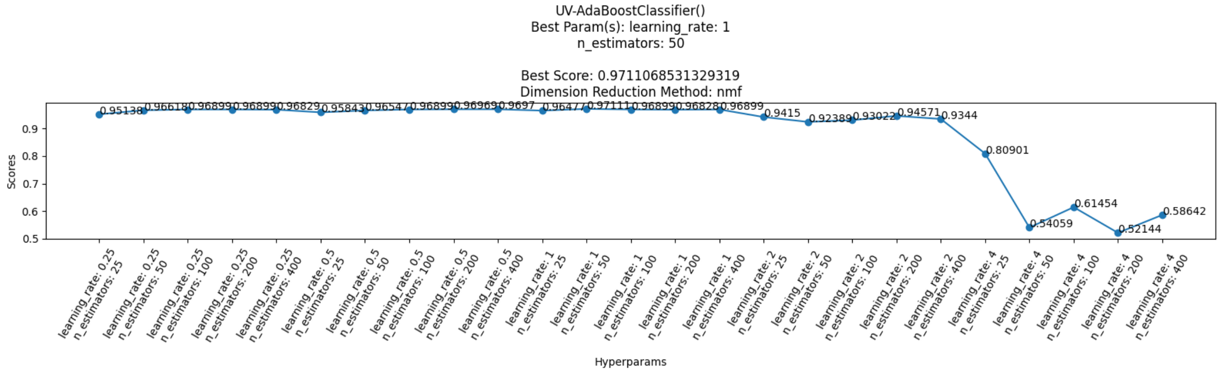
Pair.2 - MY

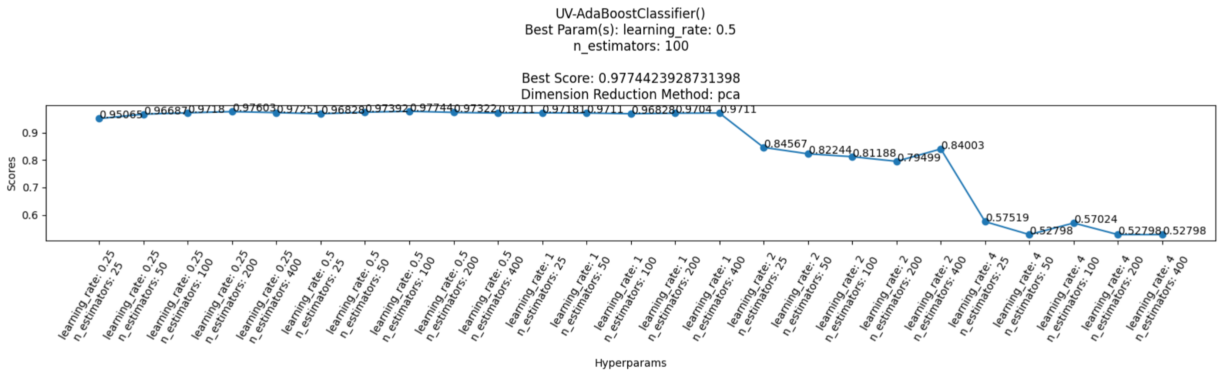




Again, as long as the learning rate is <= 1, the score is nearly the same and acceptable. And the performance impact from both dimension reduction methods are trivial. When NMF is applied, the hyperparameters are `learning_rate = 0.25, n_estimators = 400`, and PCA's hyperparameters are `learning_rate = 0.25, n_estimators = 100`.

Pair.3 - UV





The situation for pair 3 is quite similar to pair 2, and they uses `learning_rate = 1, n_estimators = 50` and `learning_rate = 0.5, n_estimators = 100` respectively.

SVM Classifier

Description

In SVM, to separate the two classes of data points, there are many possible hyperplanes that could be chosen. It is to find a plane with the maximum margin, since maximizing the margin distance provides some reinforcement so that future data points can be classified with more confidence. (Credit: https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47)

It has good generalization capabilities which prevent it from over-fitting. And it is efficient in handling non-linear data. And it is stable since a small change to the data won't greatly affect the hyperplane.

However, it is difficult to choose a appropriate kernel function and it is hard to interpret.

In this assignment, I used the algorithm from `sklearn.svm.SVC`, and tuned 2 hyperparameters, which are `C` and `kernel`.

Cross Validation Results

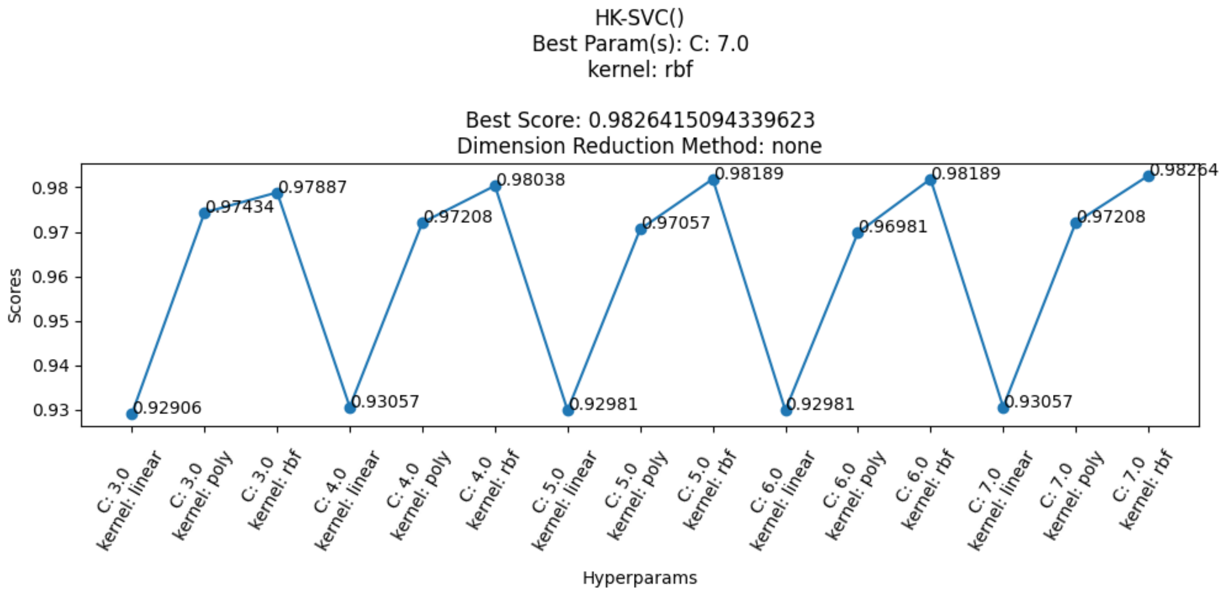
I tuned 2 hyperparameters, which are `C` and `kernel`:

`C = [3.0, 4.0, 5.0, 6.0, 7.0]`

`kernel = ['linear', 'poly', 'rbf']`

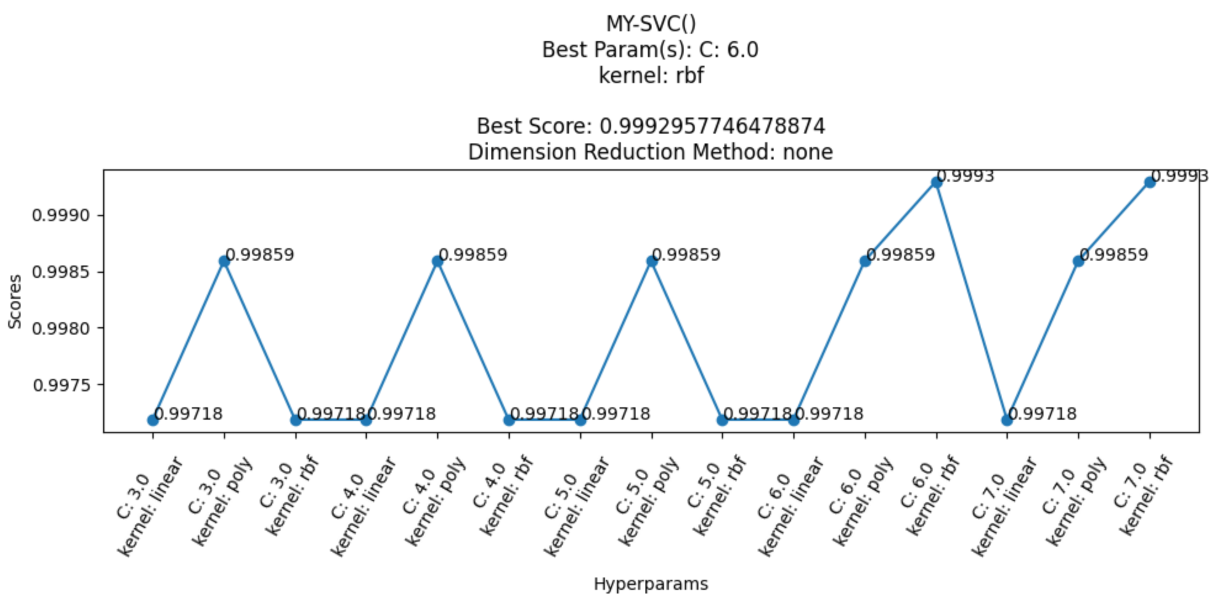
From the graphs, we learnt that the kernel selection has a huge impact to the performance of the model, and `kernel = rbf` tends to be the overall optimal choice.

Pair.1 - HK



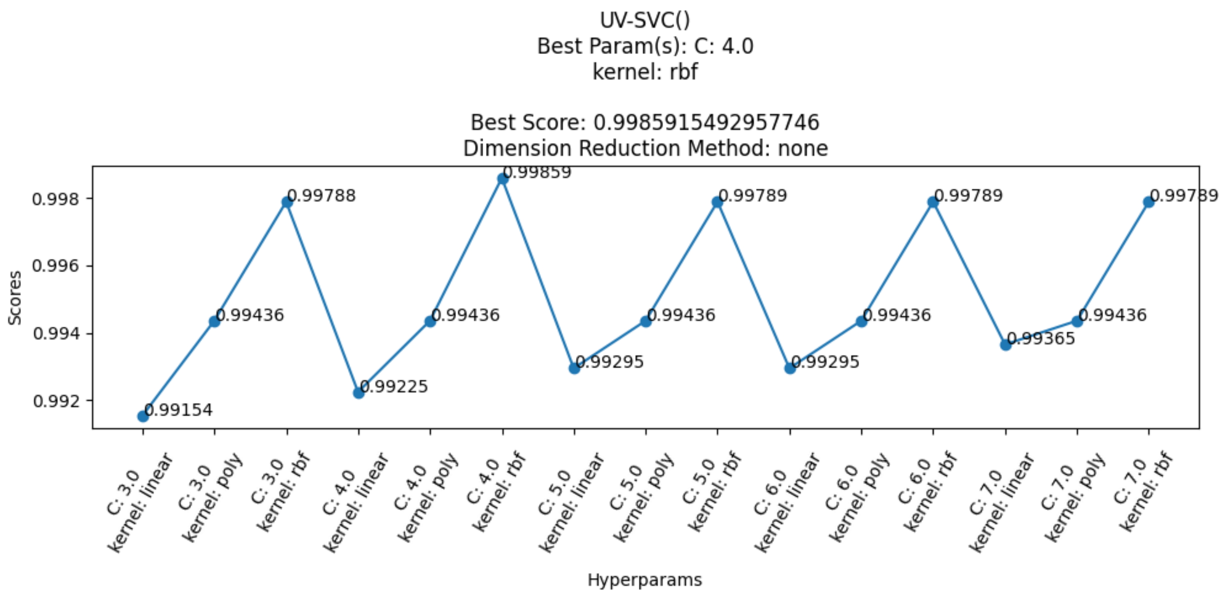
For pair 1, although different `C` value does have tiny impact on the score of the model, the selection of `kernel` dominate the decision, and the best hyperparameters are `C = 7, kernel = rbf`.

Pair.2 - MY



For pair 2, although `kernel = poly` looks promising for low `C` values, `kernel = rbf` becomes more superior when `C >= 6`. The best hyperparameters are `C = 6, kernel = rbf`.

Pair.3 - UV



For pair 3, the choice of `kernel` seems to be the most important, and the choice of `C` becomes trivial. The best hyperparameters are `C = 4, kernel = rbf`.

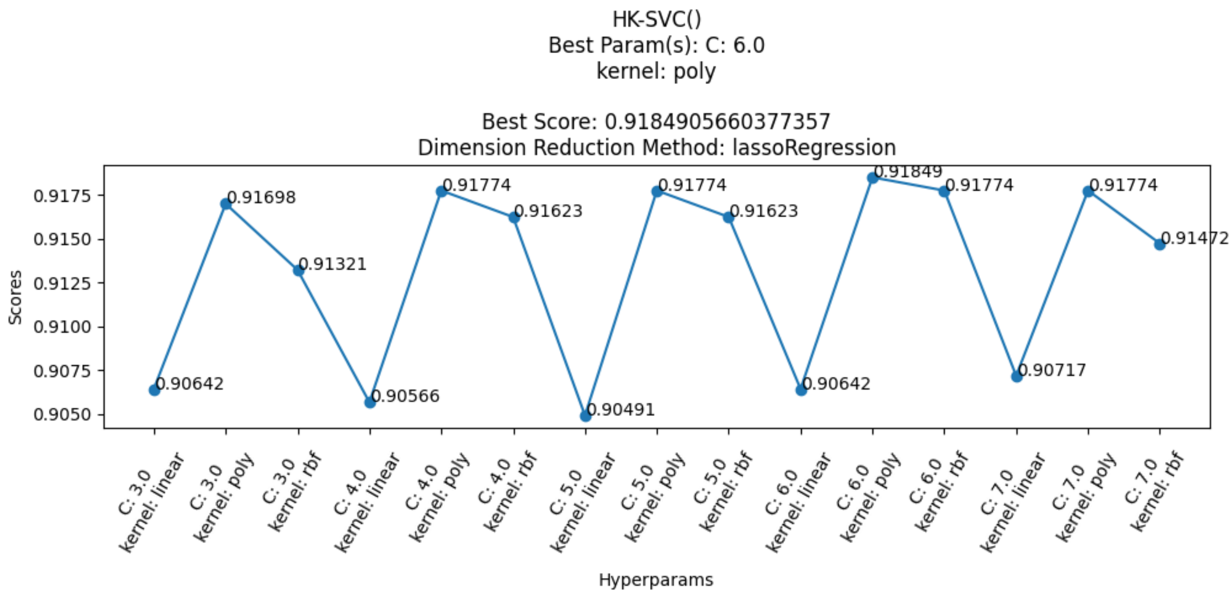
Description of the Dimension Reduction Method (Lasso Regression, PCA)

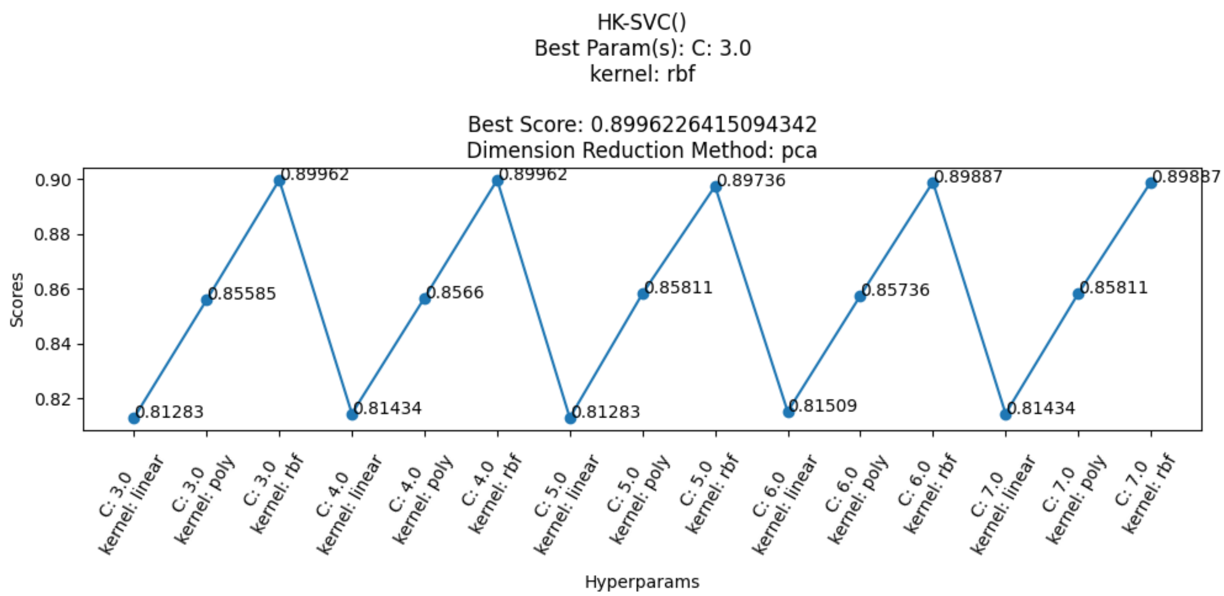
I applied 8 dimensional reduction methods (Forward Feature Selection, Backward Feature Elimination, Random Forest, Decision Tree, Lasso Regression, PCA, SVD, and NMF) to the training set and applied the same reduction to the testing set.

To reduce the length of the report (8 methods * 3 pairs = 24 images are way too many), I will only talk about 2 reduction methods, `Lasso Regression` and `PCA`, in the following section. Other graphs can be accessed from the folder I submitted to GradeScope.

Cross Validation Results with Dimension Reduction

Pair.1 - HK

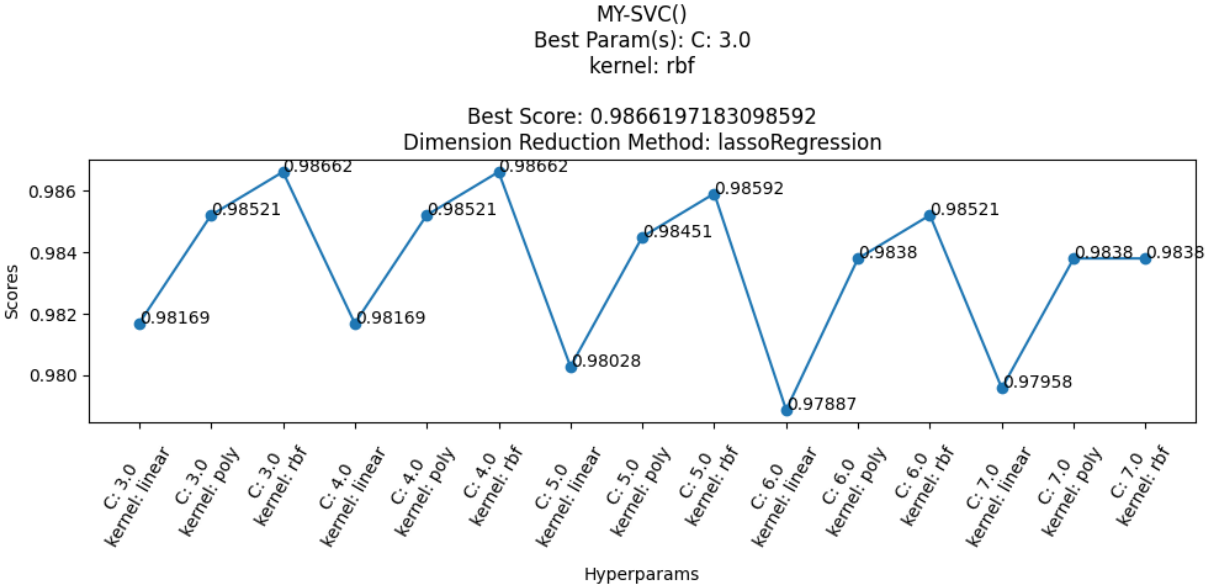


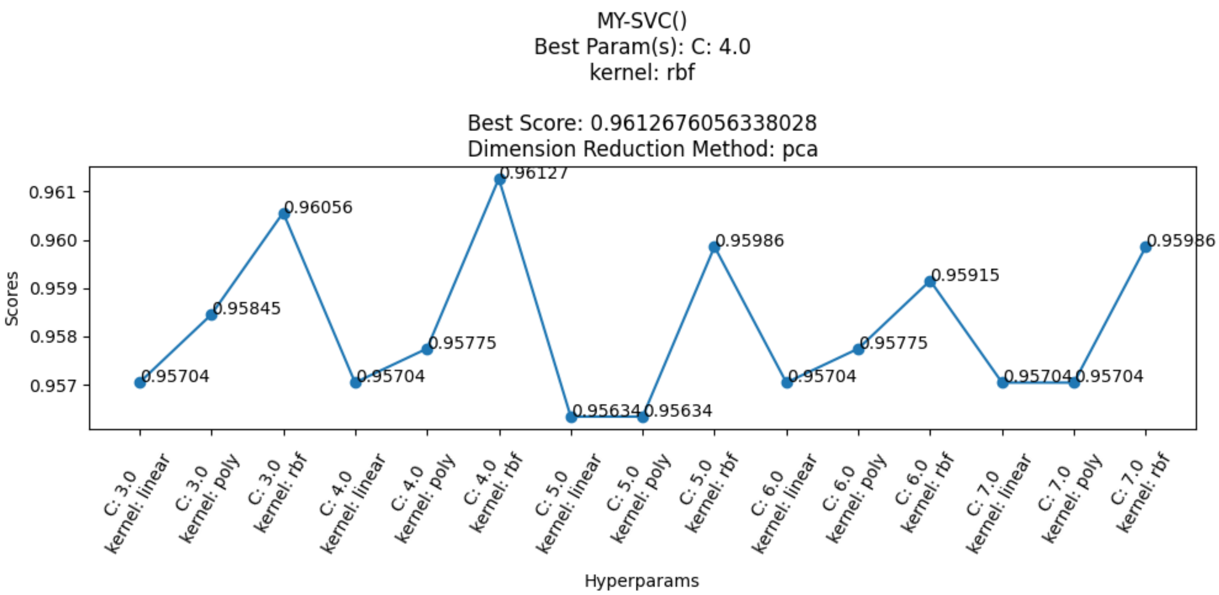


With features selected by Lasso Regression, surprisingly the best kernel changed from `kernel = rbf` to `kernel = poly`. And `C = 6`.

Model trained with PCA'd data, however, `kernel = rbf` is still a significantly better choice overall, and `C = 3`.

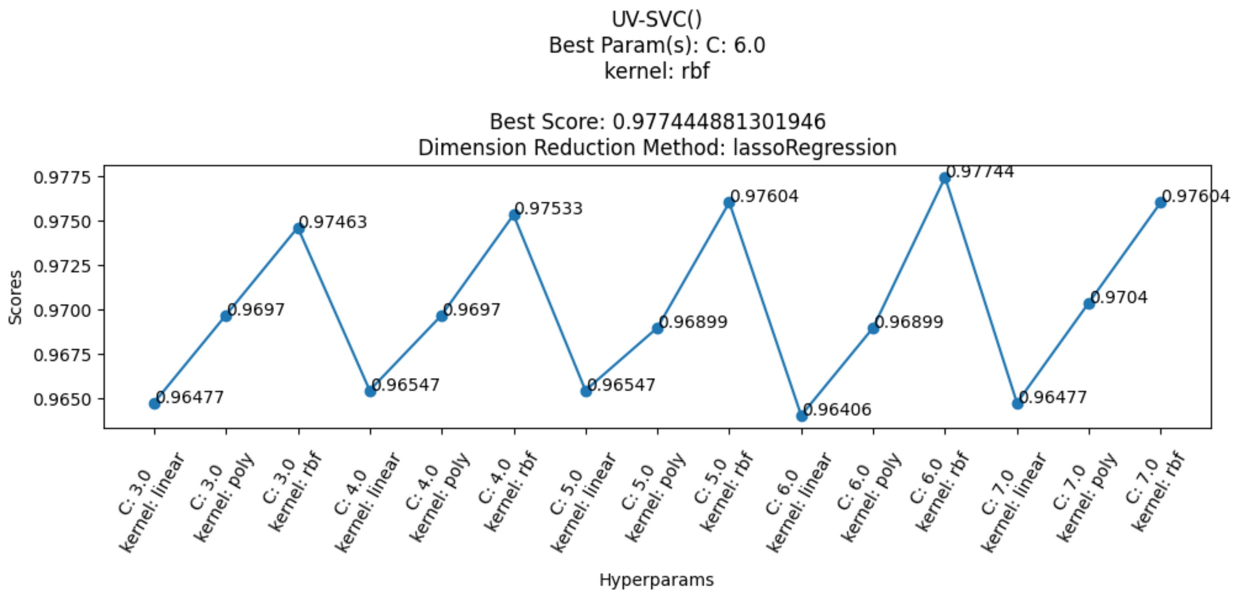
Pair.2 - MY

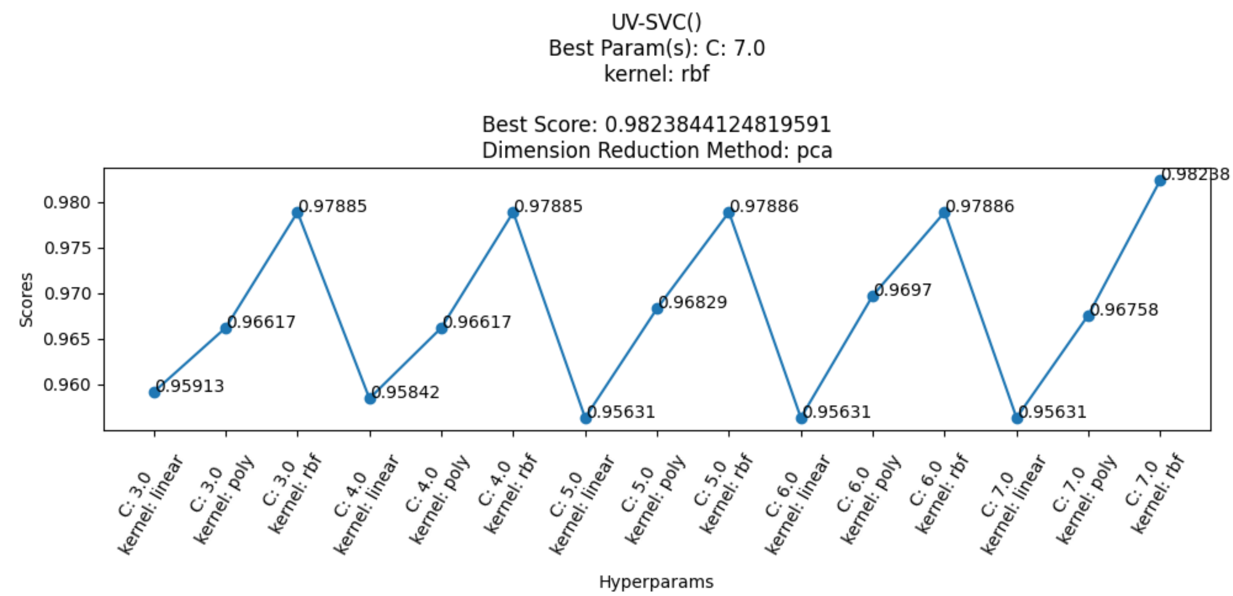




The choices of hyperparameters doesn't change that much, still `kernel = rbf` is the best choice, with `C = 3` for the former and `C = 4` for the latter.

Pair.3 - UV





The choices of hyperparameters doesn't change that much, still `kernel = rbf` is the best choice, with `C = 6` for the former and `C = 7` for the latter.

ANN Classifier

Description

ANN is artificial neural network, which tries to imitate how human brain works. Thanks to its multiple layers and feedback design, ANN is one of the best methods nowadays, as it can fit just about any model with high accuracy. However, it is difficult to interpret by a client with no data science or machine learning background.

In this assignment, I used the algorithm from `sklearn.neural_network.MLPClassifier`, and tuned 2 hyperparameters, which are `activation` and `learning_rate`.

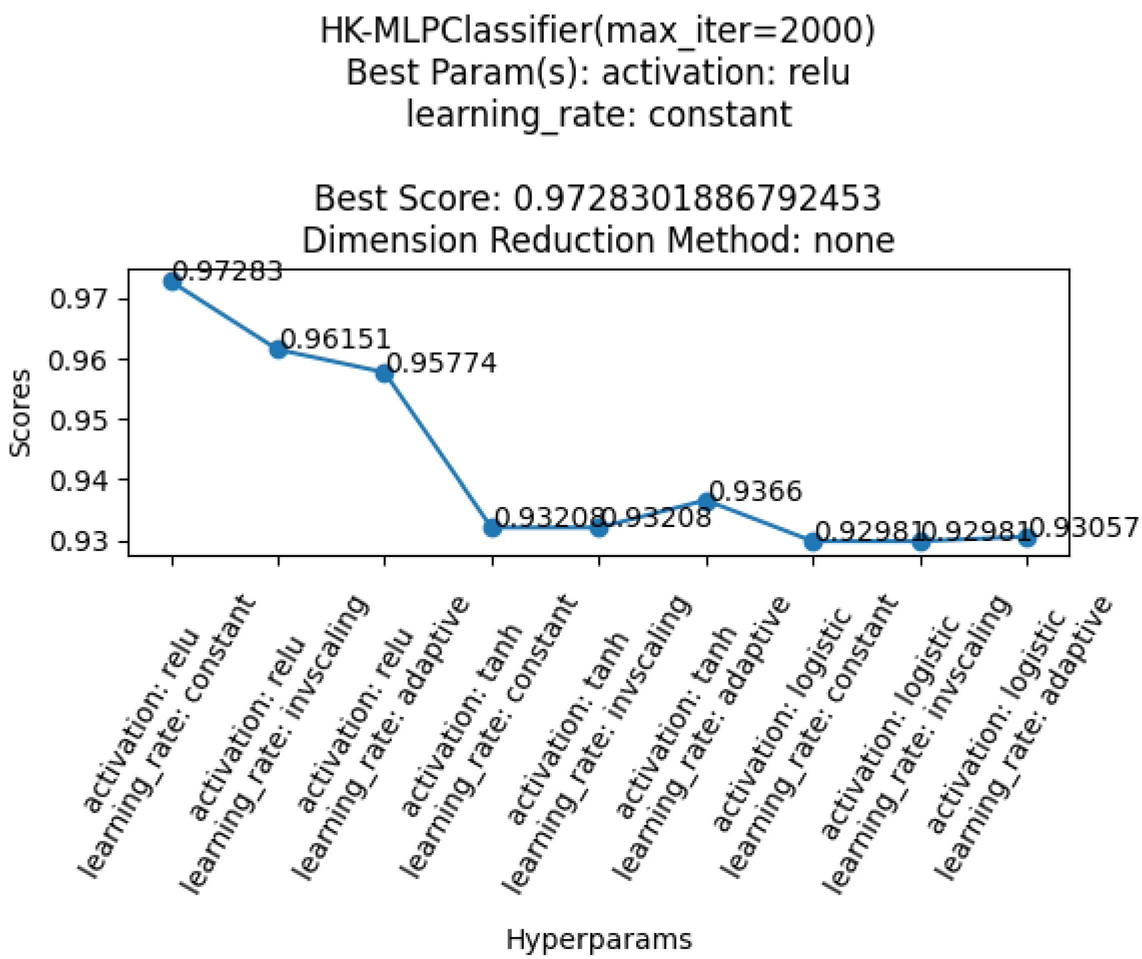
Cross Validation Results

I tuned 2 hyperparameters, which are `activation` and `learning_rate`:

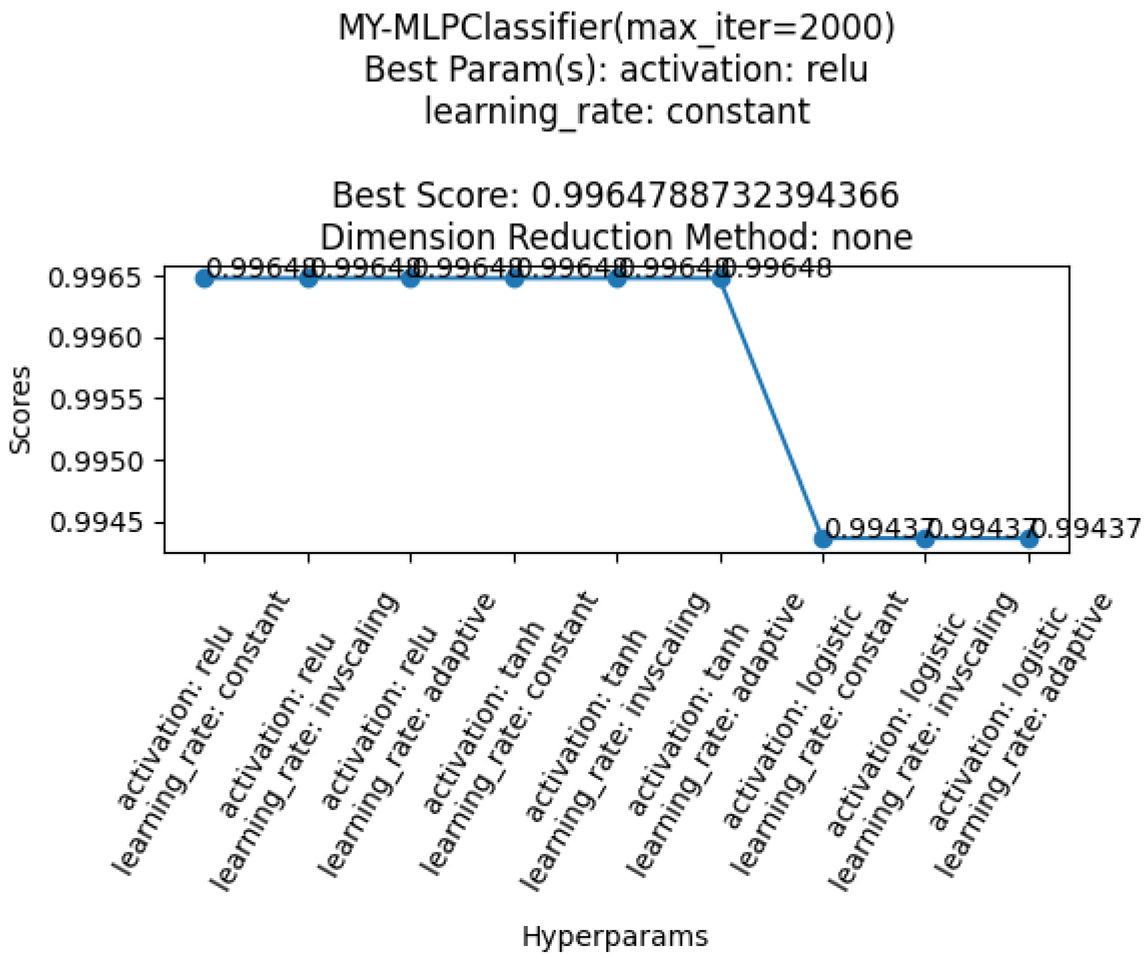
`activation = ['relu', 'tanh', 'logistic']`

`learning_rate = ['constant', 'invscaling', 'adaptive']`

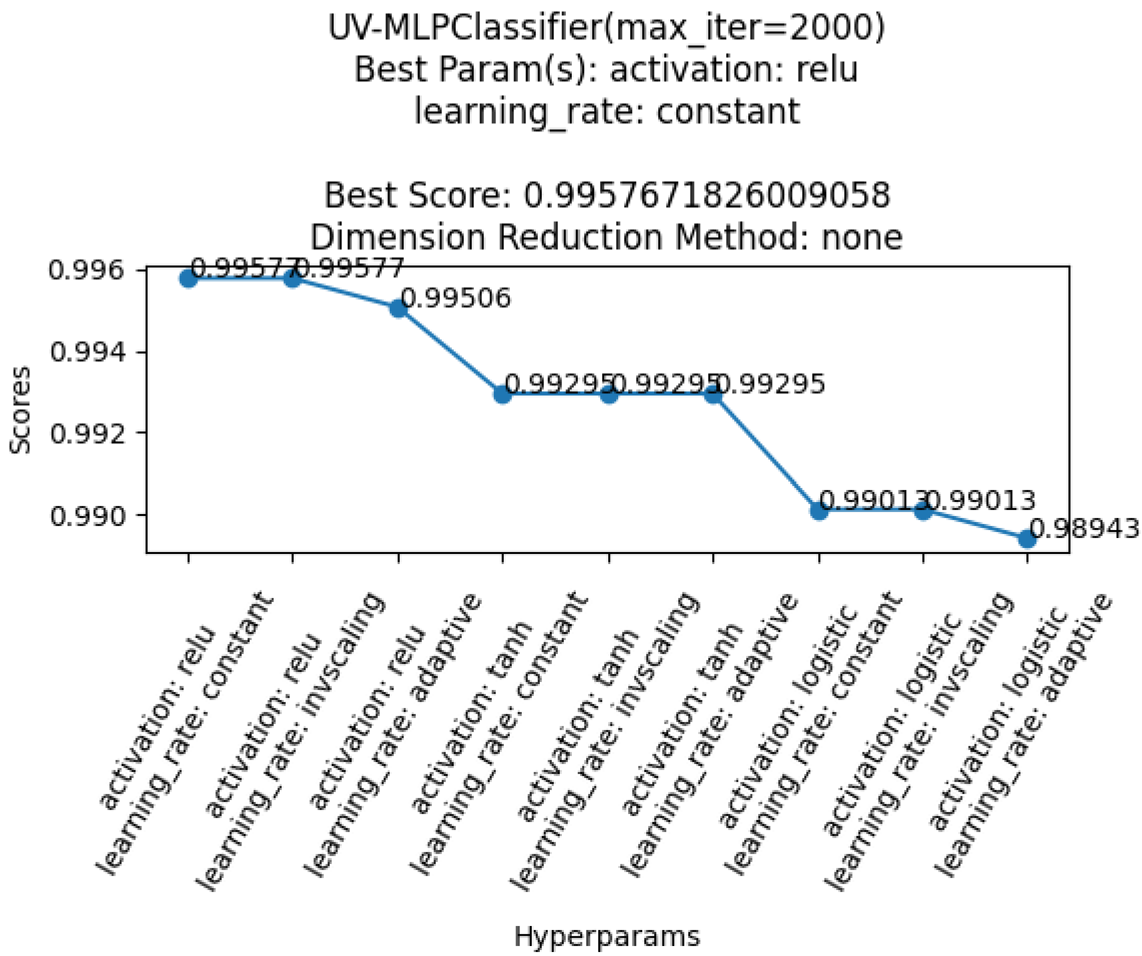
Pair.1 - HK



Pair.2 - MY


MY-MLPClassifier(max_iter=2000)
Best Param(s): activation: relu
learning_rate: constant
Best Score: 0.9964788732394366
Dimension Reduction Method: none
0.9965
0.9960
0.9955
0.9950
0.9945
Scores
0.99648
0.99648
0.99648
0.99648
0.99648
0.99648
0.99437
0.99437
0.99437
activation: relu learning_rate: constant
activation: relu learning_rate: invscaling
activation: relu learning_rate: adaptive
activation: tanh learning_rate: constant
activation: tanh learning_rate: invscaling
activation: tanh learning_rate: adaptive
activation: logistic learning_rate: constant
activation: logistic learning_rate: invscaling
activation: logistic learning_rate: adaptive
Hyperparams

Pair.3 - UV



All the best hyperparameters for all 3 pairs are exactly the same, that is `activation = relu, learning_rate = constant`. The best choice learning_rate is out of my expectation, I never thought a constant learn_rate could beat the adaptive one.

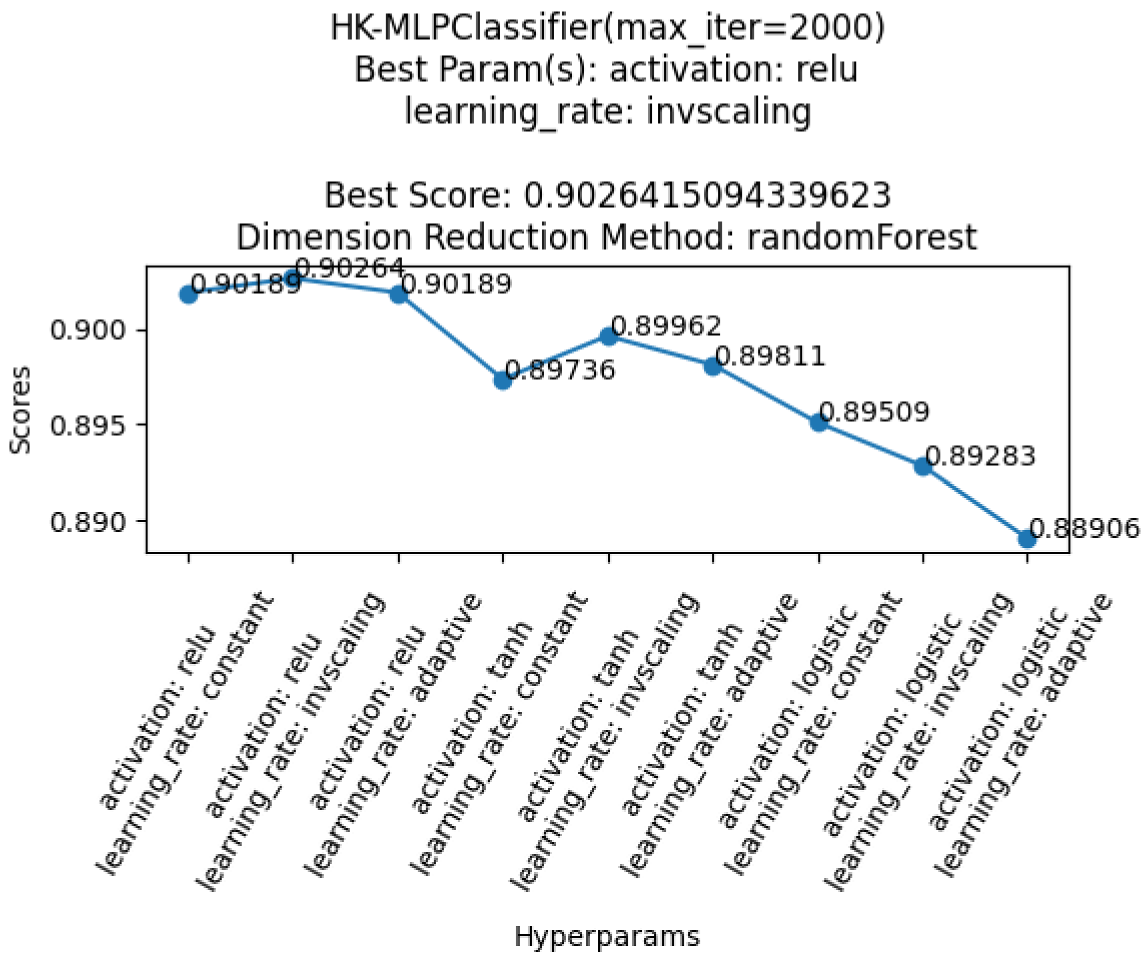
Description of the Dimension Reduction Method (Random Forest, PCA)

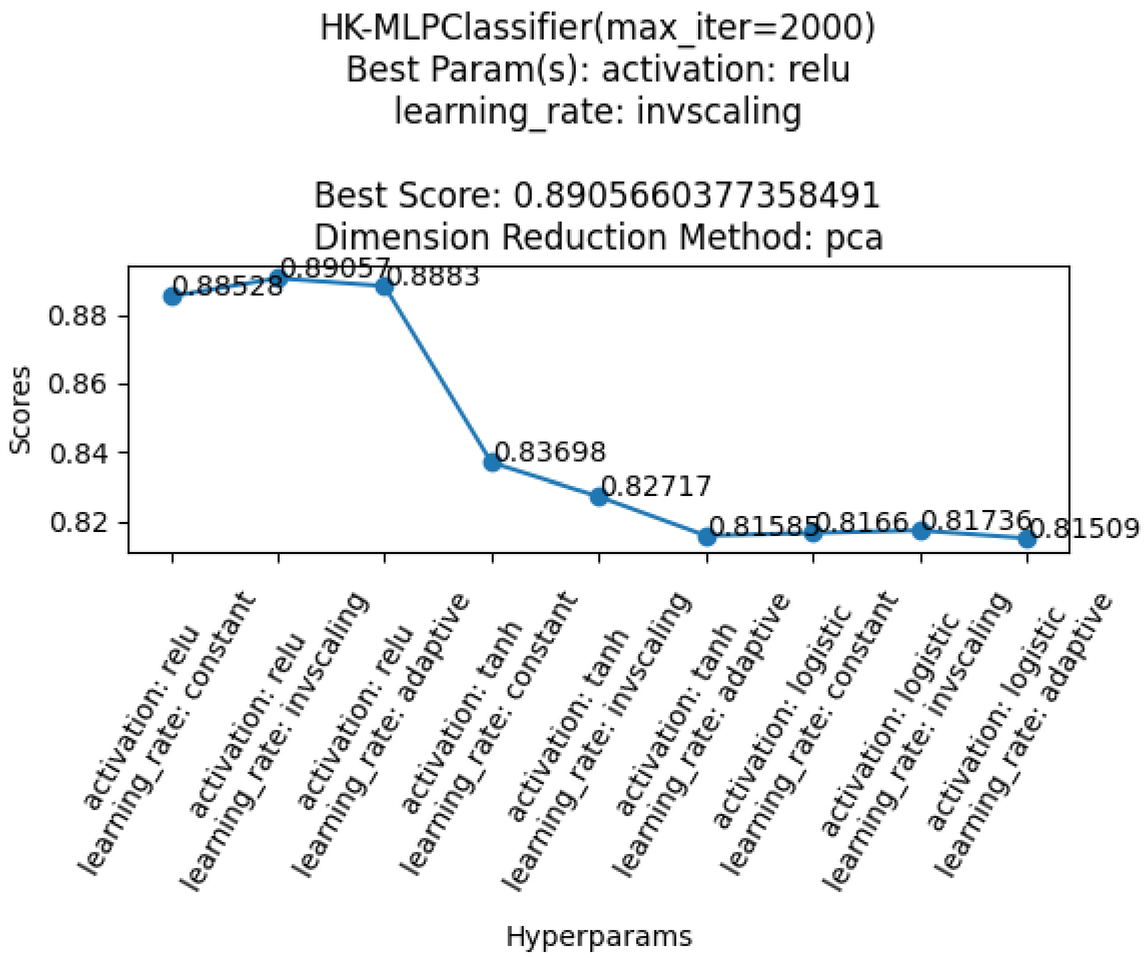
I applied 8 dimensional reduction methods (Forward Feature Selection, Backward Feature Elimination, Random Forest, Decision Tree, Lasso Regression, PCA, SVD, and NMF) to the training set and applied the same reduction to the testing set.

To reduce the length of the report (8 methods * 3 pairs = 24 images are way too many), I will only talk about 2 reduction methods, `Random Forest` and `PCA`, in the following section. Other graphs can be accessed from the folder I submitted to GradeScope.

Cross Validation Results with Dimension Reduction

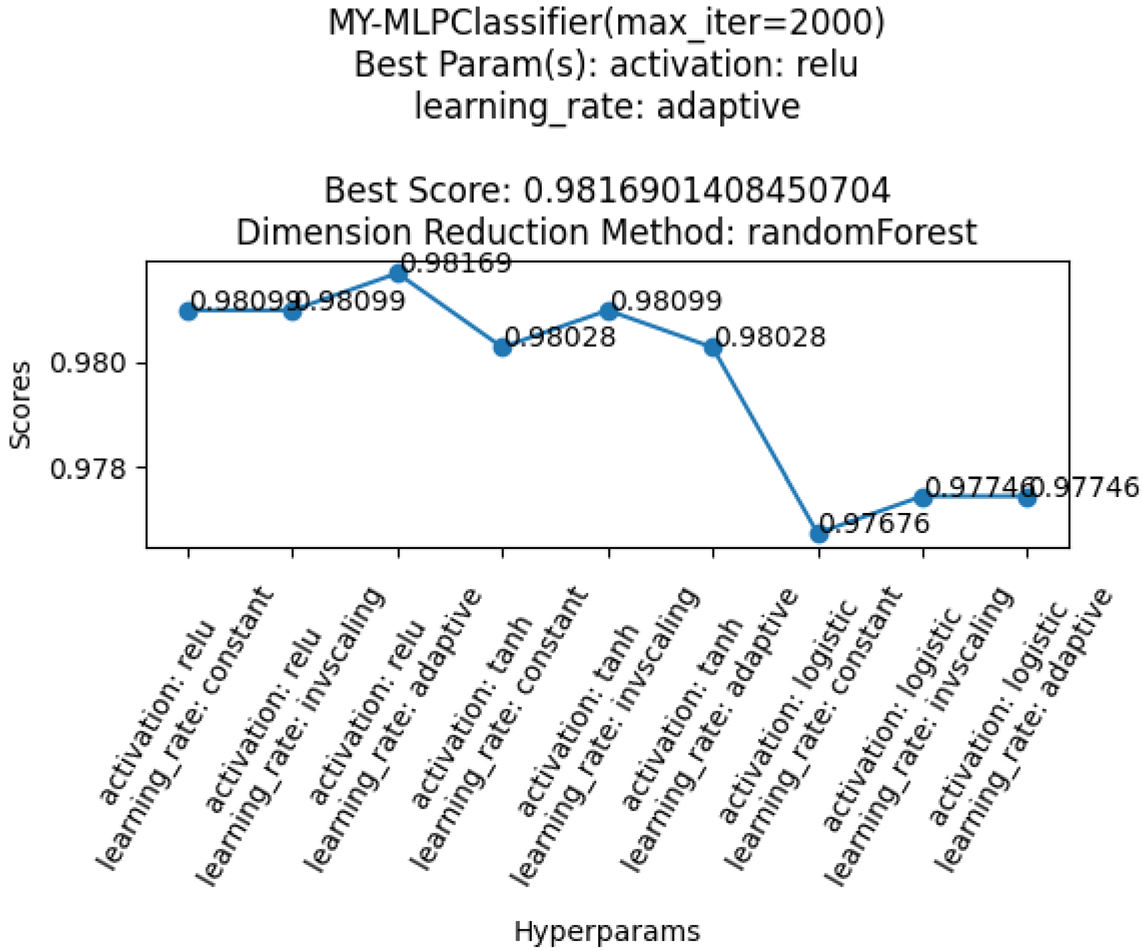
Pair.1 - HK

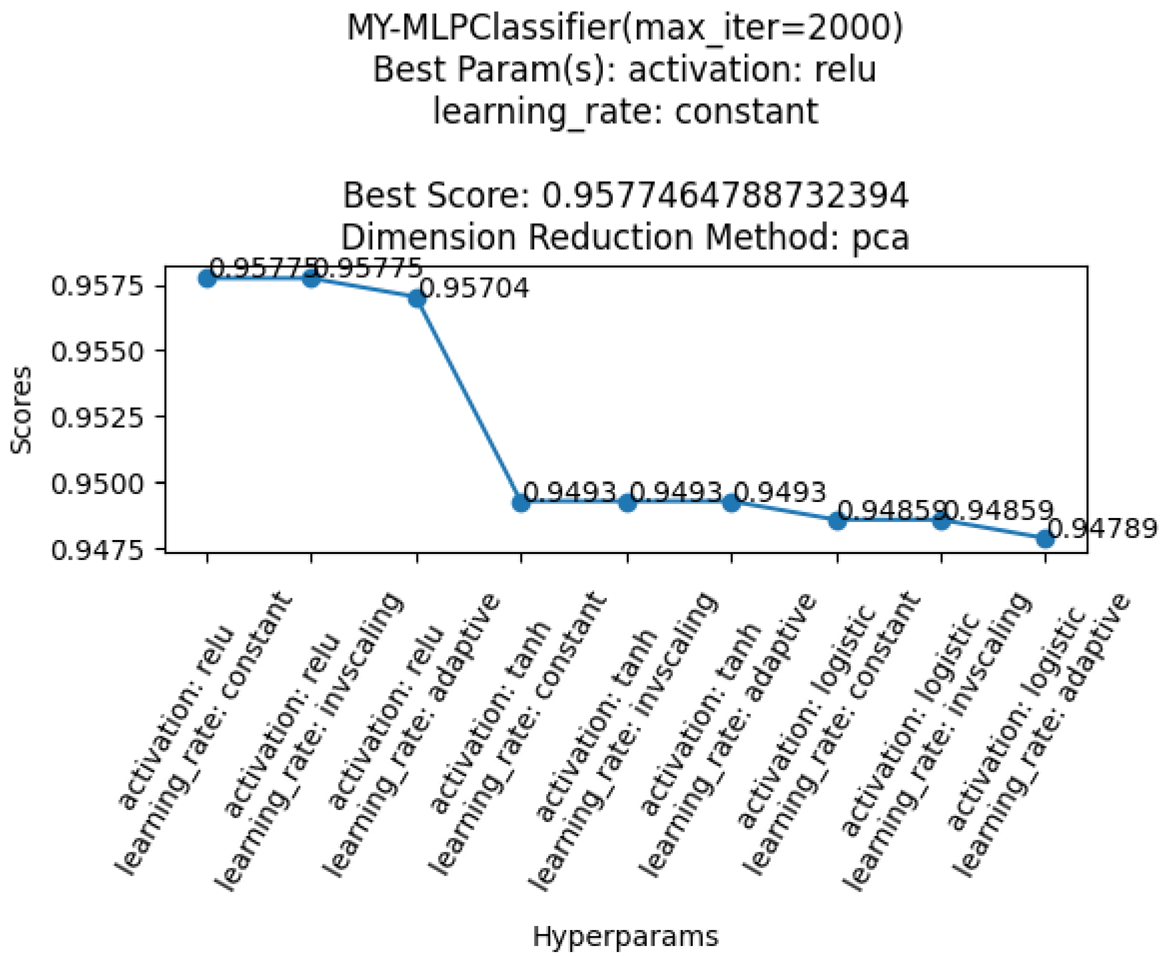

HK-MLPClassifier(max_iter=2000)
Best Param(s): activation: relu
learning_rate: invscaling
Best Score: 0.9026415094339623
Dimension Reduction Method: randomForest
Scores
0.900
0.895
0.890
0.90189
0.90264
0.90189
0.89736
0.89962
0.89811
0.89509
0.89283
0.88906
activation: relu learning_rate: constant
activation: relu learning_rate: invscaling
activation: relu learning_rate: adaptive
activation: tanh learning_rate: constant
activation: tanh learning_rate: invscaling
activation: tanh learning_rate: adaptive
activation: logistic learning_rate: constant
activation: logistic learning_rate: invscaling
activation: logistic learning_rate: adaptive
Hyperparams



With dimension reduction applied to the training samples, the best parameter for learning rate changed to `learning_rate = invscaling`. The another param though, is still `activation = relu`.

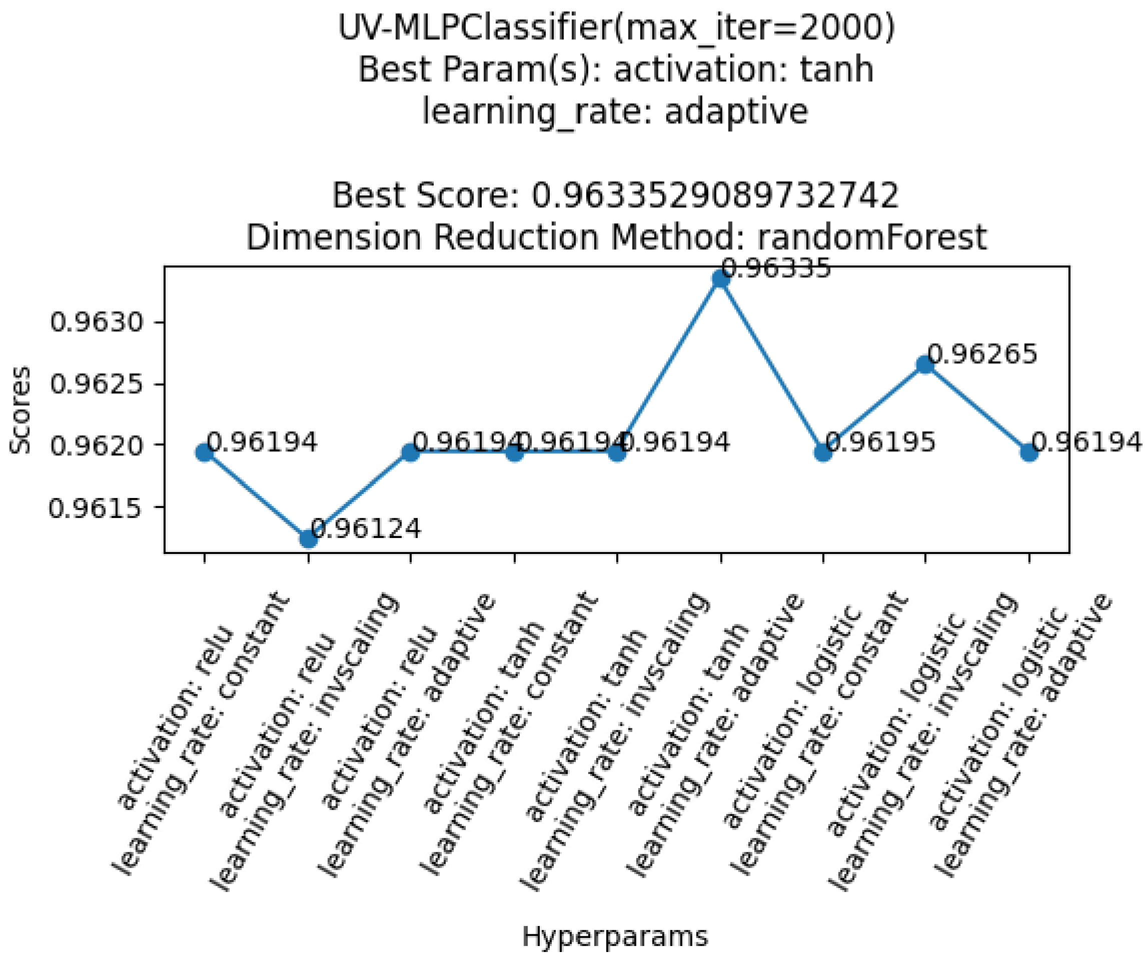
Pair.2 - MY

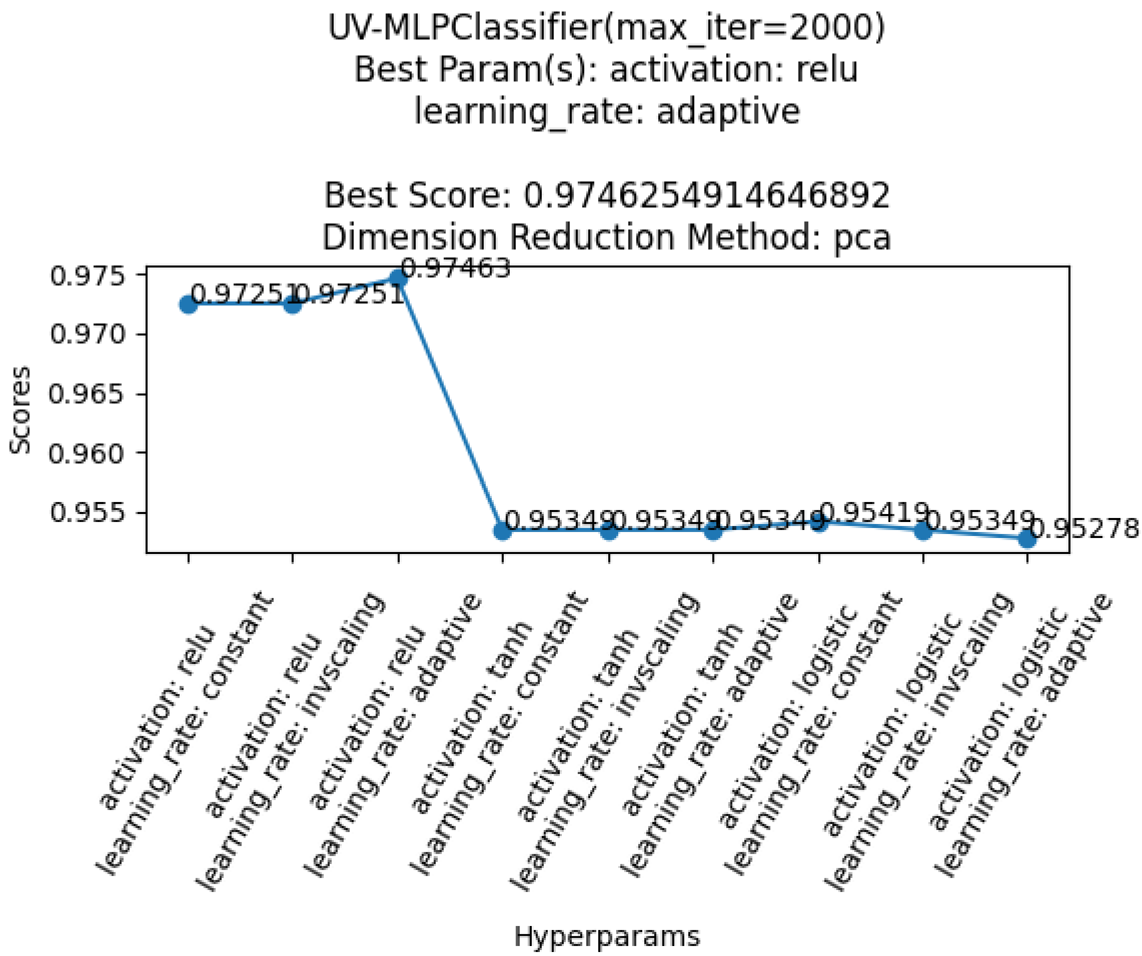

MY-MLPClassifier(max_iter=2000)
Best Param(s): activation: relu
learning_rate: adaptive
Best Score: 0.9816901408450704
Dimension Reduction Method: randomForest
Scores
0.980
0.978
0.98099
0.98099
0.98169
0.98028
0.98099
0.98028
0.97676
0.97746
0.97746
activation: relu
learning_rate: constant
activation: relu
learning_rate: invscaling
activation: relu
learning_rate: adaptive
activation: tanh
learning_rate: constant
activation: tanh
learning_rate: invscaling
activation: tanh
learning_rate: adaptive
activation: logistic
learning_rate: constant
activation: logistic
learning_rate: invscaling
activation: logistic
learning_rate: adaptive
Hyperparams



`activation = relu` is the best for both model, but the first one uses `learning_rate = adaptive` while the latter one uses `learning_rate = constant`.

Pair.3 - UV


UV-MLPClassifier(max_iter=2000)
Best Param(s): activation: tanh
learning_rate: adaptive
Best Score: 0.9633529089732742
Dimension Reduction Method: randomForest
Scores
0.9630
0.9625
0.9620
0.9615
0.96194
0.96124
0.96194
0.96194
0.96194
0.96335
0.96195
0.96265
0.96194
activation: relu
learning_rate: constant
activation: relu
learning_rate: invscaling
activation: relu
learning_rate: adaptive
activation: tanh
learning_rate: constant
activation: tanh
learning_rate: invscaling
activation: tanh
learning_rate: adaptive
activation: logistic
learning_rate: constant
activation: logistic
learning_rate: invscaling
activation: logistic
learning_rate: adaptive
Hyperparams



For random forest method, the best hyperparameters are `activation = tanh, learning_rate = adaptive`; and for the latter is `relu + adaptive`.

Bonus

- [x] +10 bonus points for additional hyperparameters tuned
- [x] +5 bonus points for addition dimension reduction methods implemented
- [x] +10 bonus points for additional classifiers

Discussion

All the data are gathered and stored in `./report/results.md`

Comparisons on Performance and Run Time across Classifiers

(Training Time)	HK	MY	UV
KNN	48ms	46ms	50ms
Decision Tree	9ms	9ms	9ms
Random Forest	1682ms	1585ms	1586ms
SVM	85ms	25ms	32ms
ANN	14281ms	8389ms	10112ms
AdaBoost	1321ms	1401ms	1379ms

Although the training time comparison is not required, I still want to compare it. Although the training time of the same classifier with different pair of dataset may not be that comparable due to differences in the hyperparameters, we can still have an insight of which classification model runs faster or slower.

(Run Time)	HK	MY	UV
KNN	4ms	5ms	5ms
Decision Tree	1ms	0ms	1ms
Random Forest	19ms	29ms	8ms
SVM	2ms	2ms	2ms
ANN	1ms	1ms	1ms
AdaBoost	10ms	10ms	19ms

Based on the result, ANN and Decision Tree has the fastest prediction time, which are closely followed by SVM. KNN is slightly slower, but still much faster than AdaBoost and Random Forest, which took significantly longer prediction time.

(Performance)	HK	MY	UV
KNN	0.9459459459459459	1.0	0.9936708860759493
Decision Tree	0.8918918918918919	1.0	0.9746835443037974
Random Forest	0.9527027027027027	1.0	0.9873417721518988
SVM	0.9864864864864865	1.0	1.0
ANN	0.9662162162162162	1.0	1.0
AdaBoost	0.9459459459459459	1.0	0.9873417721518988

It seems **Pair.2 - MY** is very easy to be classified and all the models hit a 100% accuracy rate. From the **Pair.1 - HK**, we learned that SVM and ANN has the best performance among all, which is followed by Random Forest, AdaBoost and KNN. Decision Tree has the worst performance over all, since it's quite a simple model.

In conclusion, when only considering the run time and performance, SVM and ANN easily become the best choice for this problem. Although Random Forest and AdaBoost has a good accuracy, their running time is way too long comparing to other methods. Decision Tree doesn't work very well when encountering samples that are hard to classify, like H and K. And finally KNN seems to be a cheap solution, which has acceptable performance, run time and implementation difficulty.

If taking the training time into consideration, then SVM is one of the best.

Comparisons on Performance and Run Time across Classifiers after Dimension Reduction

Since I used 8 methods in dimensional reduction. In this section I decided to use the results of PCA since it's a method that being used a lot.

(Training Time)	HK	MY	UV
KNN	37ms	37ms	37ms
Decision Tree	12ms	10ms	10ms
Random Forest	2078ms	1869ms	1868ms
SVM	89ms	38ms	35ms
ANN	7440ms	4433ms	6255ms
AdaBoost	1283ms	1321ms	1310ms

After dimension reduction are applied, depending on which classifier we are using, there is a 0% to 50% variance in the reduction of training time.

(Run Time)	HK	MY	UV
KNN	4ms	4ms	4ms
Decision Tree	0ms	0ms	1ms
Random Forest	38ms	26ms	8ms
SVM	5ms	3ms	2ms
ANN	0ms	1ms	1ms
AdaBoost	9ms	10ms	9ms

However, there is no noticeable difference in the run time of each models even the dimensionality is reducted.

(Performance)	HK	MY	UV
KNN	0.8513513513513513	0.9936708860759493	0.9873417721518988
Decision Tree	0.7972972972972973	0.9873417721518988	0.9746835443037974
Random Forest	0.8716216216216216	0.9873417721518988	0.9810126582278481
SVM	0.8648648648648649	0.9810126582278481	0.9810126582278481
ANN	0.8513513513513513	0.9810126582278481	0.9810126582278481
AdaBoost	0.8175675675675675	0.9873417721518988	0.9556962025316456

But, the prediction accuracy does degrade a considerable amount, especially for the **Pair.1 - HK**, a ~10% decrease in the accuracy is observable, whereas the other two models do have 1% ~ 2% reduction in accuracy as well. So in this case, I don't recommend do any dimension reduction.

Lessons learned

From the testing, I would confidently use SVM since it trains fast, runs fast and has superior accuracy among other models. The ANN is also a good choice, but it requires considerably longer time to train, but if we do require high accuracy and fastest prediction time, ANN would be the best option.

The dimension reduction does have impact on models' accuracy, especially for classifying H and K. It does help reduce the training time, but only has limited effects on the run time.

Next time I would probably give up using decision tree since it's performance isn't that good even comparing to KNN. And if I want the prediction runs fast, then random forest is also not a good choice as it takes huge chunk of time comparing to other methods.