Programming Assignment 2

JIANJUN WEI

April 16, 2022

1 Introduction

In this programming part we decide to solve some binary classification problems. We will use the Letter Recognition dataset in the UCI repository and we will choose three pairs of letters to do the experiment. Besides the pair H, K and M, Y, the third pair I choose is M, V. For all these three pairs, H and K might be the easiest one to classify since they looks quite different from the right sight, which means they have quite different features. However, M, Y and M, V would be much harder than H, K since V could be seen as part of M and the majority of Y is quite similar to M, which could be concluded that the feature value of these three letters are close to each other. Maybe M, V would be the most hardest pair to classify.

In this experiment, we will train the data with 8 different models, including KNN, Decision tree, Random forest, SVM, Artificial Neural Network, Naive Bayes and Gradient boosting. For each problem, we hope to select the right algorithm. Finding the best classifier is just like to the process finding boy friend or girl friend, which means we need to find the most suitable one. The best classifier not only depends on the algorithm we select, but also lies in the data we need to train. For example, if the dataset is not very large and we hope to get a quick result, we might use the Random Forest, or if the data is quite redundant, we need to first reduce the dimension and choose a specific algorithm. Moreover, validation accuracy is not the only metric to see whether the algorithm is suitable since it is highly like that the model might not perform well on data it has not been trained upon, but giving incredulous results on training data, and then, overfitting occurs.

Since we have several computational resources, we need to test multiple algorithms and parameter settings. After that, we could get the performance of the algorithm and compare them in a reliable way. And then, we could find the best algorithm with suitable parameters, which would much more close to the real situations.

Besides, we will also implement 8 dimension reduction methods. They will be selected and applied in different models. It is useful to use dimension reduction methods when prepossessing data especially when the data has thousands of features. We are generating a tremendous amount of data daily but not all of them have the specific meaning. However, too many features of data gives tremendous pressure on the algorithm and computer hardware. Also, it could cause to a much longer training time. And, more features often lead to an algorithm overfitting as it tries to create a model that explains all the features in the data. Thus, we need the dimension reduction methods and it can reduce the computational demands associated with training a model but also helps combat overfitting by keeping the features that will be fed to the model fairly simple.

There are lots of factors which will be used to determine whether a reduction method is good or bad. For example, we need to see whether the method has already removed the noise of the data, or whether the unrelated data have been discarded. Also, we could have a check to see whether we retain or construct the feature that largely represents the data itself and remove the one who are bringing the old information.

2 Experiment results

In this section, we would like to implement 7 different models. For each model, we will show the cross validation results. Then, we will select one dimension reduction method and retrain the model. The results will also be shown for each of model. Totally there are 8 different dimension reduction methods

that are implemented in the experiment, including Backward Feature Elimination, Forward Feature Elimination, Decision Tree, Random Forest, LASSO regression, PCA, SVD and UMAP.

2.1 k-nearest neighbors

The k-nearest neighbors algorithm is a simple, supervised machine learning algorithm that could be used to solve both classification problems. Compared to SVM and other classify algorithm, its training time complexity is much lower and the the value is converged to $O(n)$. Also, k-nearest neighbors algorithm has high accuracy and it is not sensitive to the singular value. However, its space complexity and computational complexity will be high. Also, it could not give the specific meaning of the data, which means that it has a not good interpret ability.

2.1.1 Classify H and K

Here we would like the show the results of k-nearest neighbors algorithm for pair H and K. With the help of $sklearn.neighbors.KNeighborsClassifier$, I tune the hyperparameter $n_neighbors$, which is the number of neighbors, and $algorithm$, which is the algorithm to compute the nearest neighbors. Here are the value:

- $n_neighbors$: [1, 2, 3, 4, 5].
- $algorithm$: [brute, ball_tree, kd_tree].

The results is shown in Figure 1 and Figure 2. As we have 5 values for $n\_neighbors$ and 3 values for $algorithm$, we finally get 15 combinations.

Figure 1 shows the result using the original dataset of H and K. We could see that when the $algorithm$ is kd_tree and $n\_neighbors$ is 1, we get the best score as 0.95689. For each of the combination, it is not hard to see that when the $n\_neighbors$ equals 2, we get the worst performance.

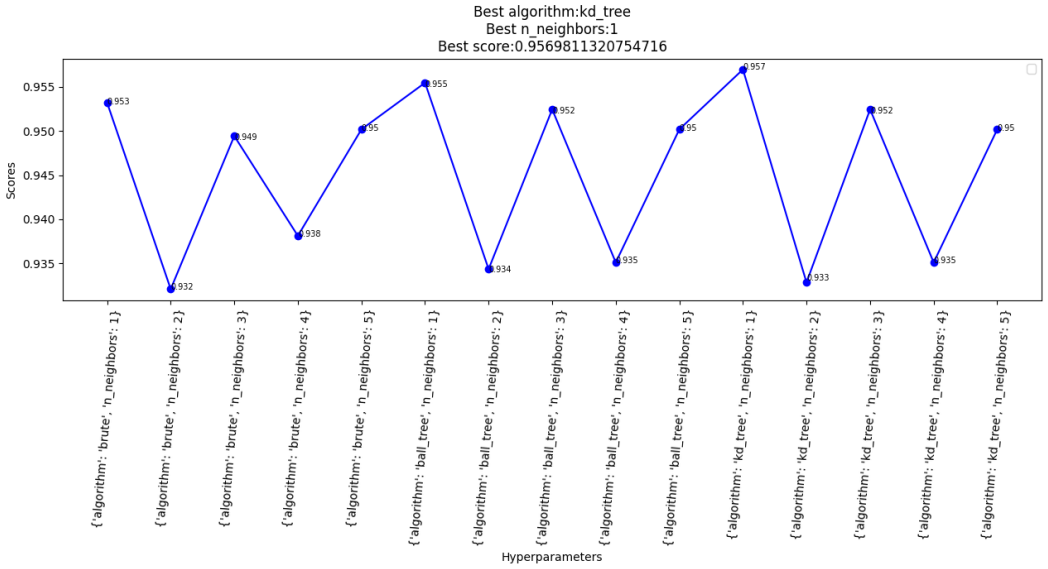

Figure 1: KNN H, K without dimension reduction

However, in Figure 2, we prepossess the data using Backward Feature Elimination method. The Backward Feature Elimination method takes all the variables present in our dataset and train the model using them, and then, calculate the performance of the model and drop one variable every time. After that, we train the model on the remaining n-1 variables and repeat the process until no variable can be dropped.

Since we need to reduce the dimension from 16 into 4, the final feature of data would be (Feature 4, Feature 8, Feature 12, Feature 16). We could see from the result that we get the best score when we set $n\_neighbors$ into 5 with $algorithm$ as kd_tree.

Compared with these two results, we could find that the result without dimension reduction performs a little better than another. And the reason might be the fact that we lost some of the feature information when using Backward Feature Elimination reduce the dimension. Also, since we only have 16 features in the original dataset, roughly reducing the dimension may lose some important feature information, thus getting a lower score.

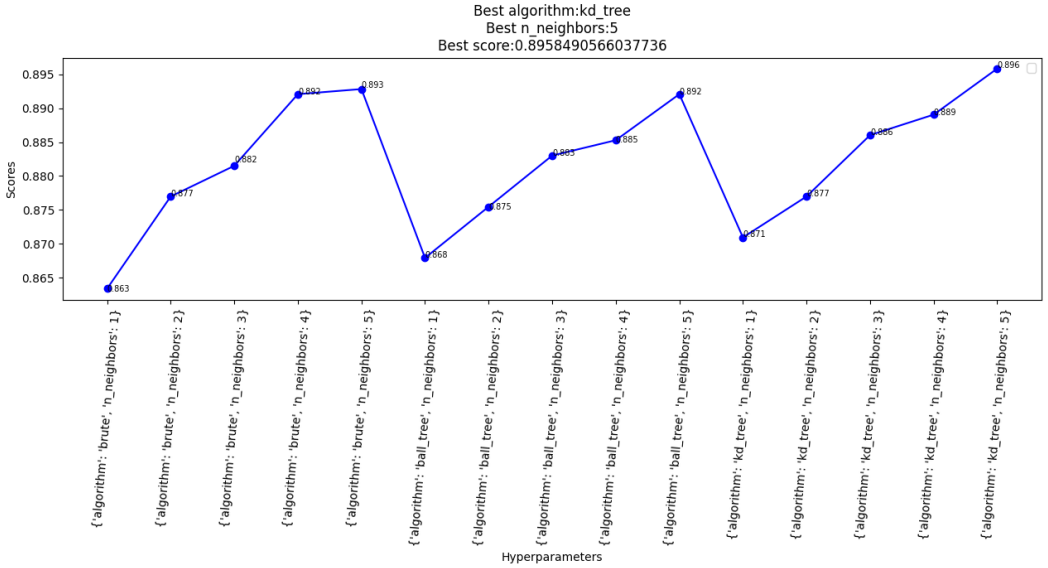

Figure 2: KNN H, K with Backward Feature Elimination reduction

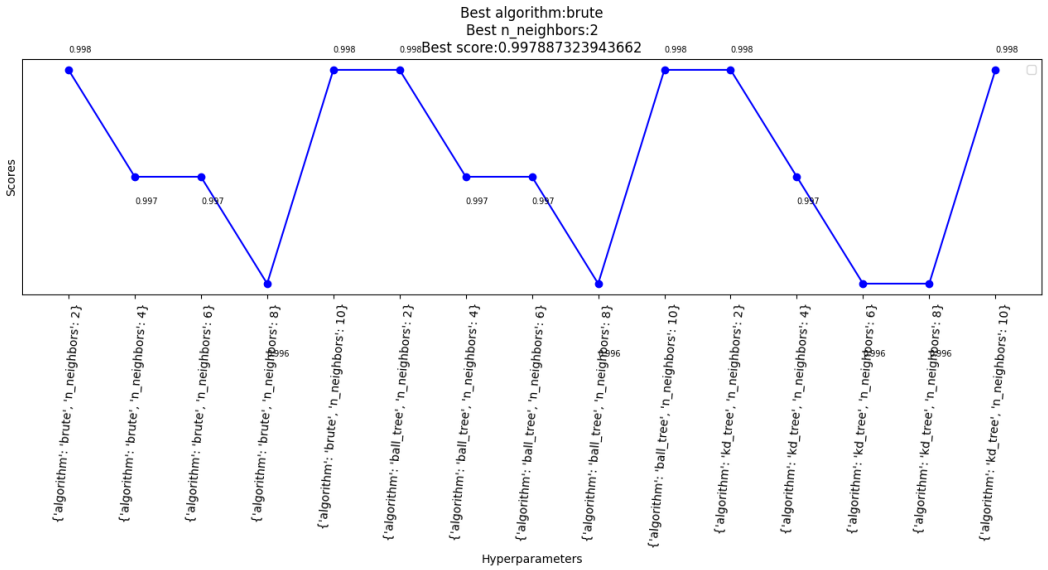

Figure 3: KNN M, Y without dimension reduction

2.1.2 Classify M and Y

Here we will try another different pair M and Y with k-nearest neighbors algorithm. We still focus on the two hyperparameters: *n_neighbors* and *algorithm*. But, we would like to set different values.

- *n_neighbors*: [2, 4, 6, 8, 10].
- *algorithm*: [brute, ball_tree, kd_tree].

The results for pair M and Y is shown in Figure 3 and Figure 4.

In Figure 3, we could see that the binary classify result is pretty well and it reaches 0.998 with *algorithm* as brute and *n_neighbors* as 2.

In Figure 4, we use Forward Feature Selection method, and thus, we get a totally different result. The Forward Feature Selection method is quite similar to the Backward Feature Elimination method, but works in an opponent way. We start with a single feature. Essentially, we train the model n number of times using each feature separately. And then, the variable giving the best performance is selected as the starting variable. We repeat the process until no significant improvement is seen in the model's performance.

Here (Feature 4, Feature 13, Feature 14, Feature 15) are selected. From the result in Figure 4, we could see that when the *n_neighbors* equals 4 and the *algorithm* is brute, we get the peak scores with these two hyperparameters.

Compared with these two results, we could see it is still a little bit lower then the one without dimension reduction, but it gets a higher score and performs much better than the pair H and K using Backward Feature Elimination method.

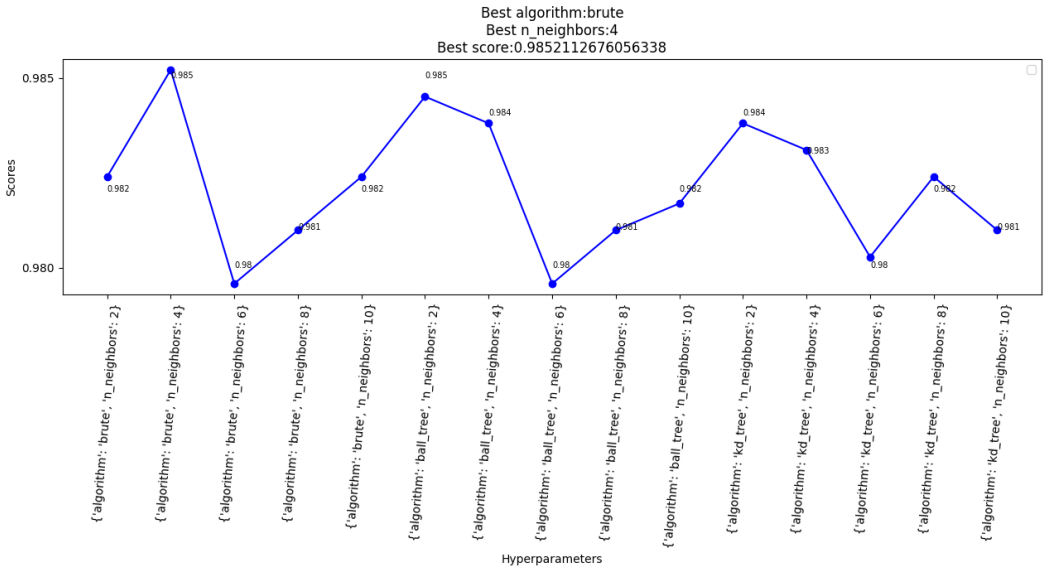

Figure 4: KNN M, Y with Forward Feature Selection reduction

2.1.3 Classify M and V

We will continue do the research on k-nearest neighbors algorithm and the focus on the pair M and V. In this experiment, we still tune two hyperparameters and set the value as:

- *n_neighbors*: [3, 5, 7, 9, 11].
- *algorithm*: [brute, ball_tree, kd_tree].

The results are shown in Figure 5 and Figure 6.

In Figure 5, we could see that the k-nearest neighbors algorithm still performs well and the best score reaches 0.992 when *n_neighbors* is 3 and we select brute algorithm to find nearest neighbors.

However, when we do the dimension reduction for the dataset of pair M and V, we choose the Random Forest method. Random Forest is one of the most widely used algorithms for feature selection. For each tree of the random forest, it can calculate the importance of a feature according to its ability to increase the pureness of the leaves. The higher the increment in leaves purity, the higher the importance of the feature. This is done for each tree, then is averaged among all the trees and, finally, normalized to 1. Once we have the importance of each feature, we perform feature selection recursively.

In Figure 6, we could see that the best score is 0.9935 when the algorithm is brute and *n_neighbors* equals 3.

Compared with these two results, we could the score for dimension reduction is slightly higher than the one without dimension reduction, which means Random Forest method has selected the correct features and largely keeps the features of all 16 features in the original dataset.

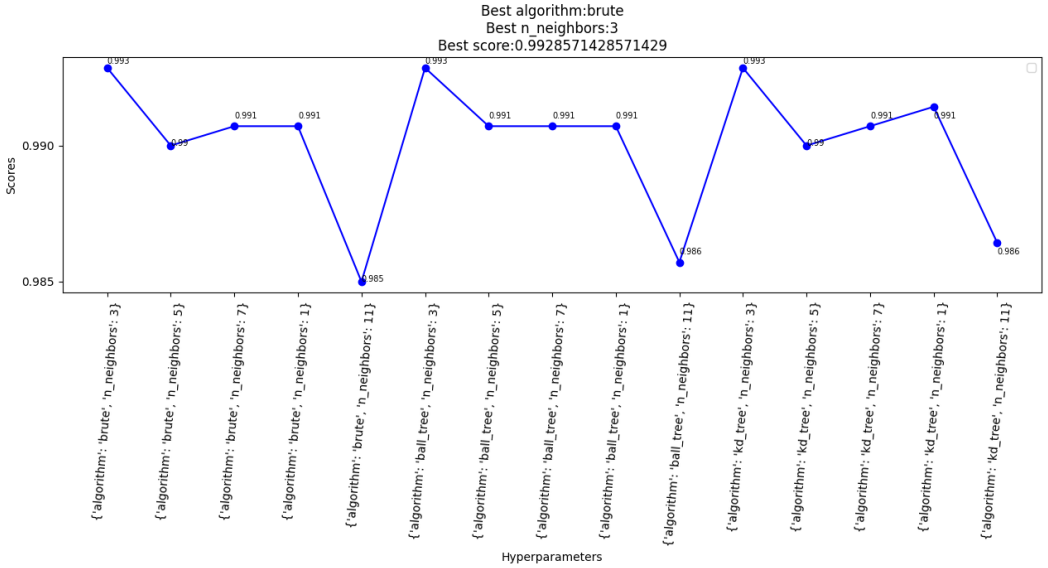

Figure 5: KNN M, V without dimension reduction

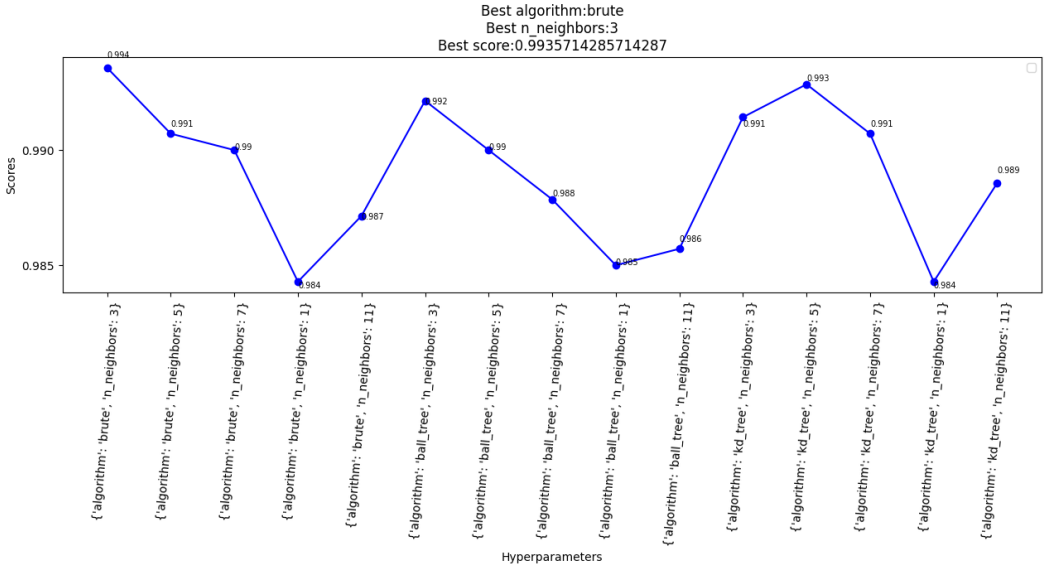

Figure 6: KNN M, V with Random Forest reduction

2.2 Decision tree

In this section, we would like to choose Decision tree as our training model to classify H and K, M and Y, M and V.

Decision Trees are powerful machine learning algorithms capable of performing regression and classification tasks. It looks like an inverted tree-like structure just as a family tree. We start at the root of the tree that contains our training data. At the root, we split our dataset into distinguished leaf nodes, following certain conditions like using an if/else loop and the splitting criteria are carefully calculated using a splitting technique. The advantage of the Decision tree is that the computational

complexity is not very high and and it is friendly to categorical data. However, Decision tree is much more likely to over fit the data. Also, when there are lots of classes, it does not performs well.

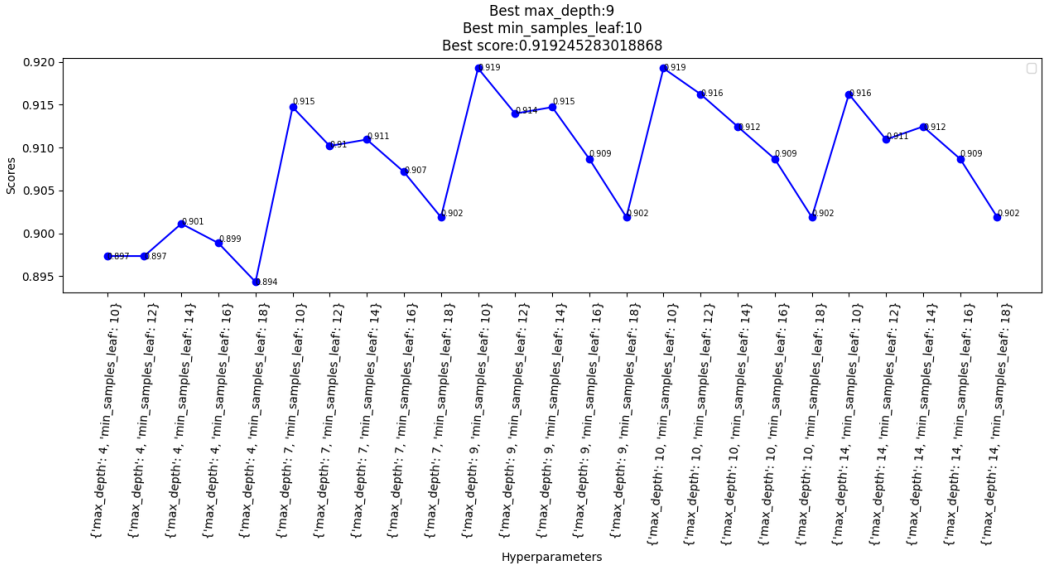

Figure 7: Decision Tree H, K without dimension reduction

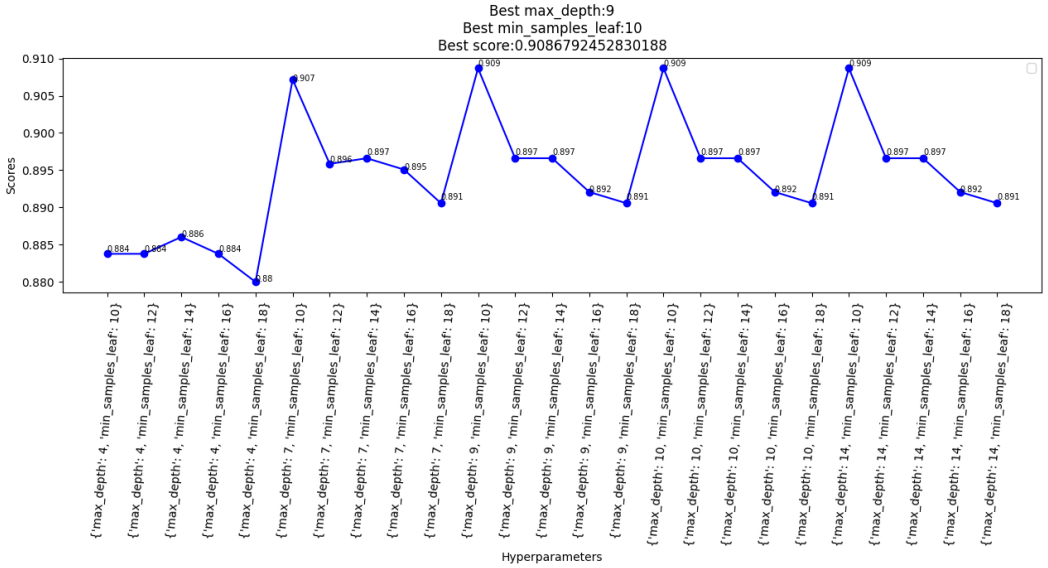

Figure 8: Decision Tree H, K with Decision Tree reduction

2.2.1 Classify H and K

Here we would like to show the results which are trained by Decision tree model. First we choose pair H and K. In this experiment, we use the package $sklearn.tree.DecisionTreeClassifier$ to help us tune the hyperparameters.

We still choose 2 different hyperparameters and see the experiment results. max_depth means the maximum depth of the tree and $min\_samples\_leaf$ means the minimum number of samples required to split an internal node. Since both of the parameters have 5 values, we would generate 25 values for the whole combination.

- max_depth: [4, 7, 9, 10, 14].

- *min_samples_leaf*: [10, 12, 14, 16, 18].

The results are shown in Figure 7 and Figure 8, which are quite straightforward.

In Figure 7, it is not hard to find that when the maximum depth of the tree is 9 and the minimum number of samples required to split an internal node is 10, we get the best score of 0.919. But the score is not much lower when the *min_samples_leaf* is 18.

In Figure 8, however, we introduce a new dimension reduction method and it is still the Decision tree. When we implement Decision tree method to select feature, we still need to calculate the importance of each feature and choose the important ones. The process is quite similar to the one for the random forest.

From the result in Figure 8, we could see that the best score reaches in 0.908. In this situation, the *min_samples_leaf* is 10 and *max_depth* lies in 9, which is quite similar to the one without dimension reduction. Similarly, when the *min_samples_leaf* is 18, we get the worst result in our experiment.

Compared with these two result, we could find that both of the result reaches a high score with the same hyper parameters. Despite the similar result, it proves that Decision tree reduction method works and it retains the feature of these 2 letters among all 16 features we have in the original dataset. The result will be much more obvious when there is much more feature with more different hyper parameters.

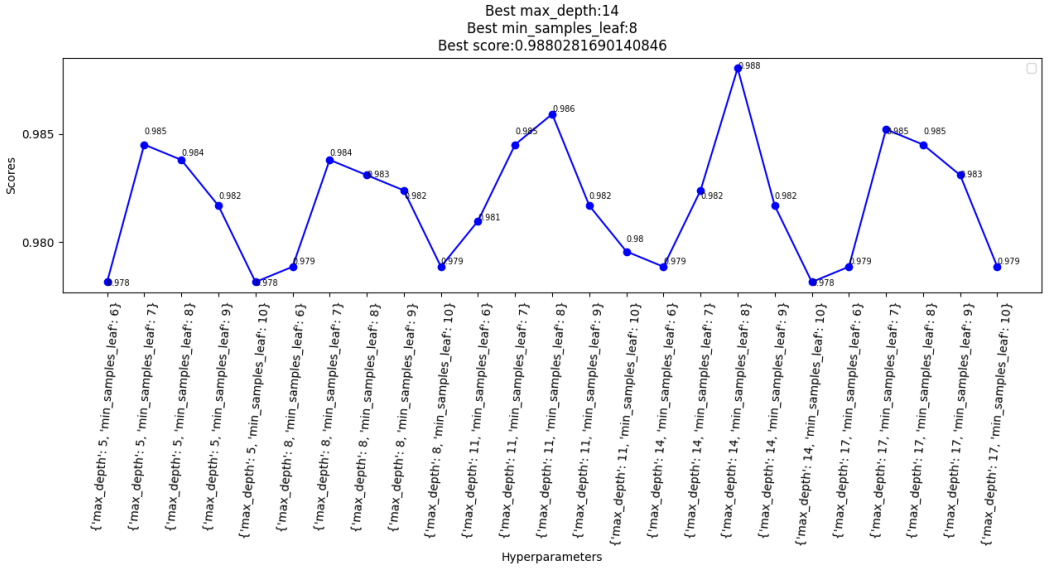

Figure 9: Decision Tree M, Y without dimension reduction

2.2.2 Classify M and Y

Again, we implement our Decision tree model training letter M and Y. We still focus on these two hyper parameters but with different values.

- *max_depth*: [5, 8, 11, 14, 17].
- *min_samples_leaf*: [6, 7, 8, 9, 10].

The results are shown in Figure 9 and Figure 10.

In Figure 9, we could see that when the max depth is 8 and the minimum samples leaf is 14, we get the best score of 0.988 with the original dataset.

In Figure 10, we introduce a new dimension reduction method which is totally new. Also, this method would be the last feature selection method we choose in this experiment. The method we choose here is Lasso regression.

Lasso regression is a linear model that uses a specific cost function. The L1 penalty and α is a hyperparameter that tunes the intensity of this penalty term. Trying to minimize the cost function,

Lasso regression will automatically select those features that are useful, discarding the useless or redundant features. In Lasso regression, discarding a feature will make its coefficient equal to 0. Thus, we choose the largest 4 coefficients in the result array to reduce the dimension from 16 into 4.

From the result, we could see the best score is 0.984 with max depth as 5 and min samples leaf as 6. Similarly, we could see that when the min samples leaf is over 9, the score would decrease sharply.

Compared with these two results, we could see that although both of them reach the high score, the hyperparameters are in totally different value. The score will be lower when $min\_samples\_leaf$ is over 9 when choosing Lasso regression while the result reaches the highest when the $min\_samples\_leaf$ equals 14 when there is no dimension reduction. The reason behind lies in the fact that the dimension reduction changes the feature we select, which leads to different hyperparameters that matches the highest score in the experiment.

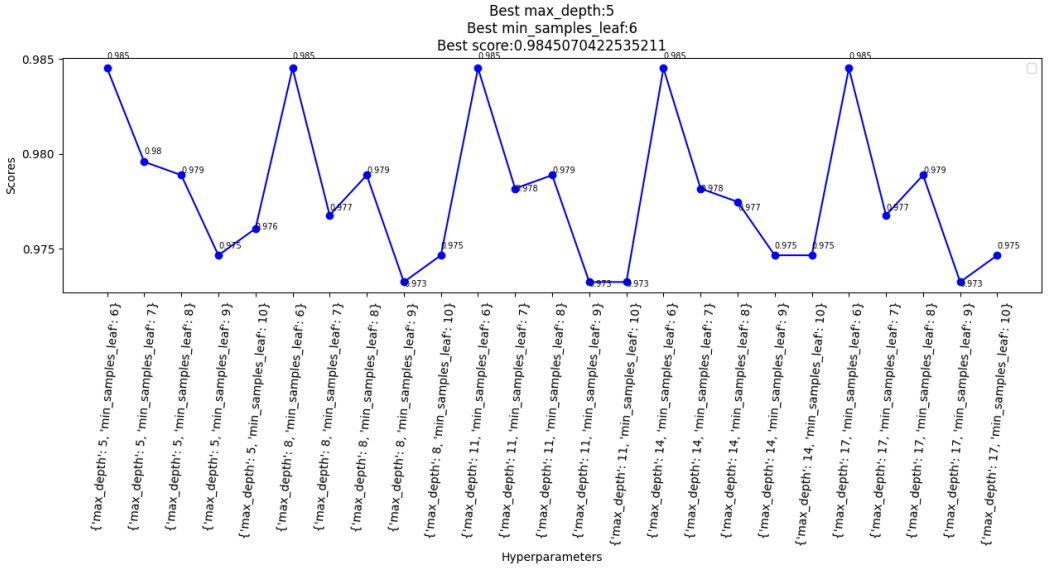

Figure 10: Decision Tree M, Y with Lasso regression reduction

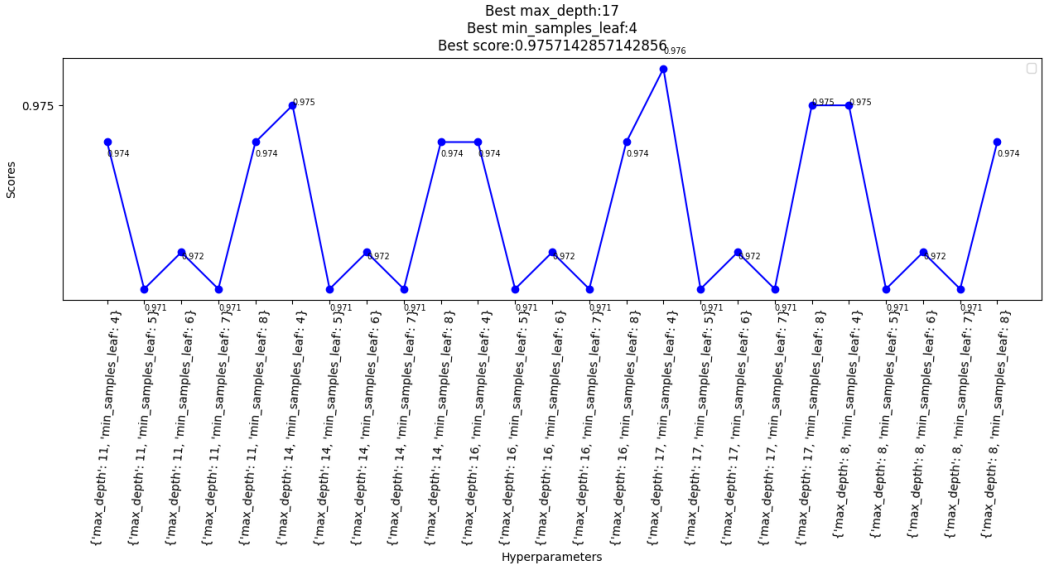

Figure 11: Decision Tree M, V without reduction

2.2.3 Classify M and V

Now, we continue our classify experiment and we will apply our Decision tree training model into letter M and V. The value of hyperparameters we choose to tune would be:

- *max_depth*: [11, 14, 16, 17, 8].
- *min_samples_leaf*: [4, 5, 6, 7, 8].

The results are shown in Figure 11 and Figure 12.

In Figure 11, we get the best score of 0.975 where max depth lies in 17 and min samples left is 4. We could find that when the *max_depth* is high but *min_samples_leaf*, we usually get a bad score.

In Figure 12, we select Random Forest as a reduction method to reduce the 16 features into 4. Easily to see that we get the best score of 0.983 with max depth of 11 and min samples leaf is 4.

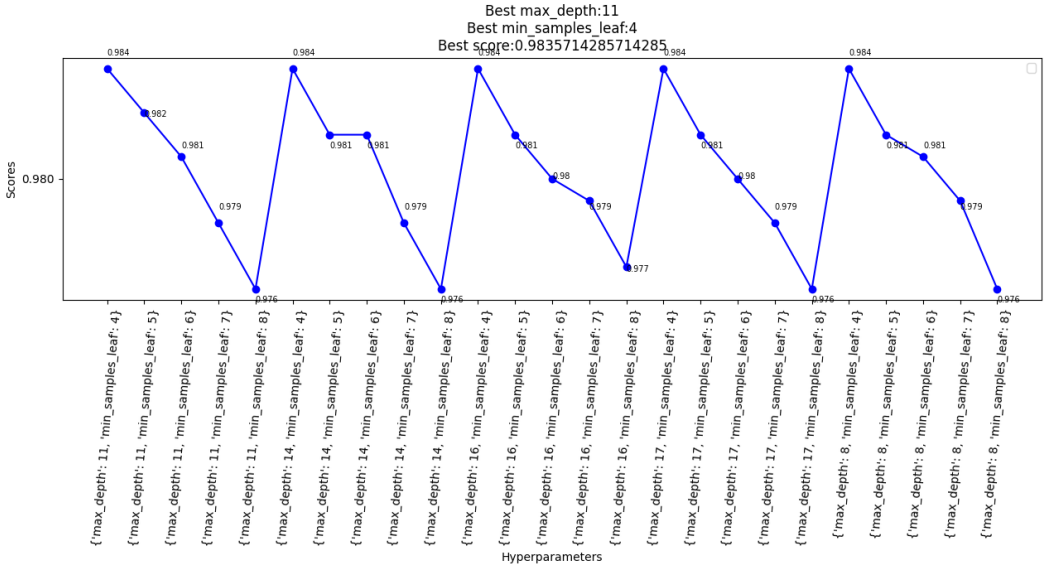

Figure 12: Decision Tree M, V with Random Forest reduction

Compared with these two experiments, we could see that Random Forest reduction method successfully chooses the features that are highly correlated with the letter. Although both of the experiments get the high score with totally different values of hyperparameters, it is understandable since we reduce the dimension, which makes the training data much more precise.

2.3 Random Forest

Now, we will move on the next classify model, Random Forest. The random forest is a classification algorithm consisting of many decisions trees. It uses bagging and feature randomness when building each individual tree to try to create an uncorrelated forest of trees whose prediction by committee is more accurate than that of any individual tree.

The advantage of Random Forest is obvious. It has much higher accuracy and could be used on large dataset. However, just as the Decision tree model, when meeting the classify with much more noise, overfitting situation might easily happen. Besides, if there is a requirement of real-time response, Random forest might not be the suitable one since when there is large amounts of decision trees in the forest, it might take huge amount of training time and could reach a much higher space complexity.

2.3.1 Classify H and K

Similarly, we start from letter H and K. In this section, we would like to ask $sklearn.ensemble.RandomForestClassifier$ for help. We still hope to tune two hyperparameters and here we put more emphasis on $n\_estimators$ and *max_depth*, which means the number of trees in the forest and the maximum depth of the tree respectively. We could see the value would be:

- *max_depth*: [10, 50, 100, 150, 200].
- *n_estimators*: [5, 15, 20, 25, 30].

The results are shown in Figure 13 and Figure 14.

In Figure 13, we could see that when the maximum depth of tree is 150 and there is 25 trees, we get the best score of 0.9698. However, when the *n_estimators* is too small, the model does not performs well.

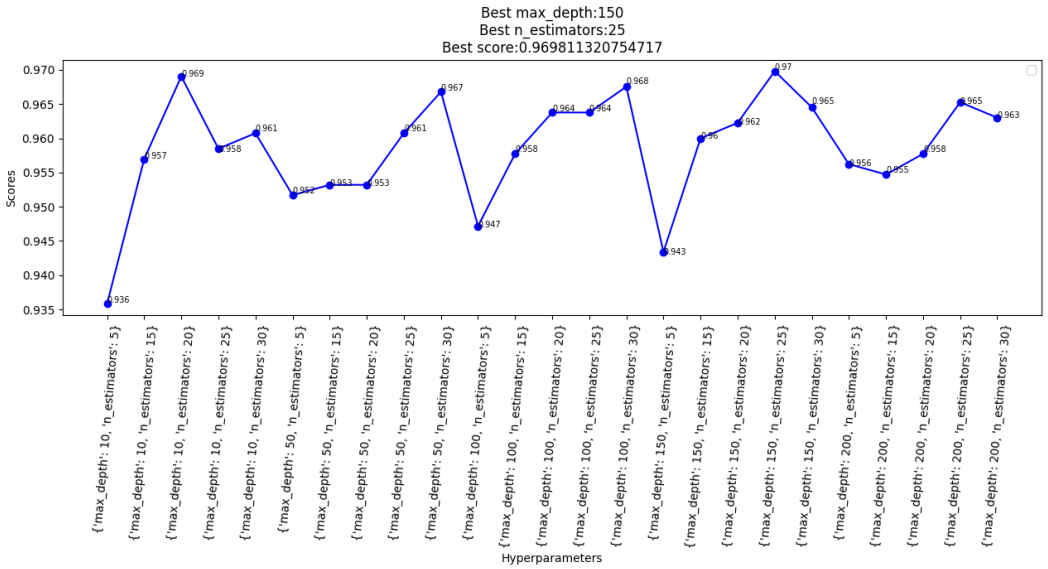

Figure 13: Random Forest H, K without reduction

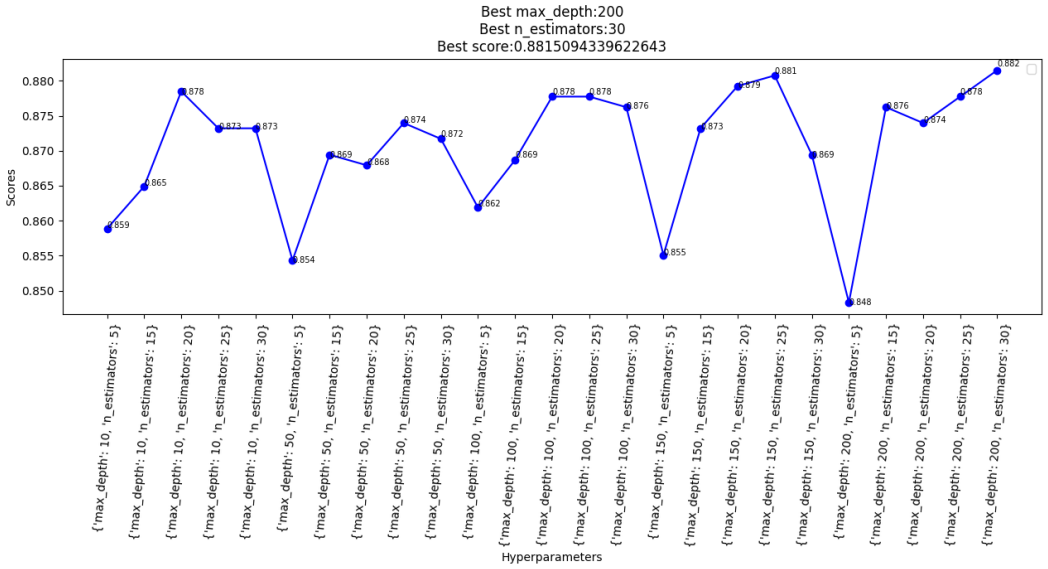

Figure 14: Random Forest H, K with PCA reduction

In Figure 14, we introduce a new dimension reduction method, PCA, which means Principal Component Analysis. Different from previous feature selection methods, PCA is one of the useful feature extraction methods which creates new features or characteristics of data by analyzing the characteristics of the dataset. Usually, the characteristics of the data are summarized or combined together. The combined feature would at utmost keep the features in the original dataset, which makes PCA powerful and influential.

The advantage of PCA is that it removes the mutual influence between features and could be easily implemented. Also, the feature would not be disturbed by information other than the data set. However, PCA might not get the most optimized result when the data is not in Gaussian distribution.

In Figure 14, we could see that the best score reaches 0.8815 with *max_depth* of 200 and *n_estimators* of 30.

Compared with previous two experiments, we could conclude that model with PCA on these two hyperparameters does not perform well. The score is much lower than the one without dimension reduction. The reason may lies in that we may need more trees in the forest and less maximum depth for each tree. Or, we may try other parameters to see whether or not the value might make sense.

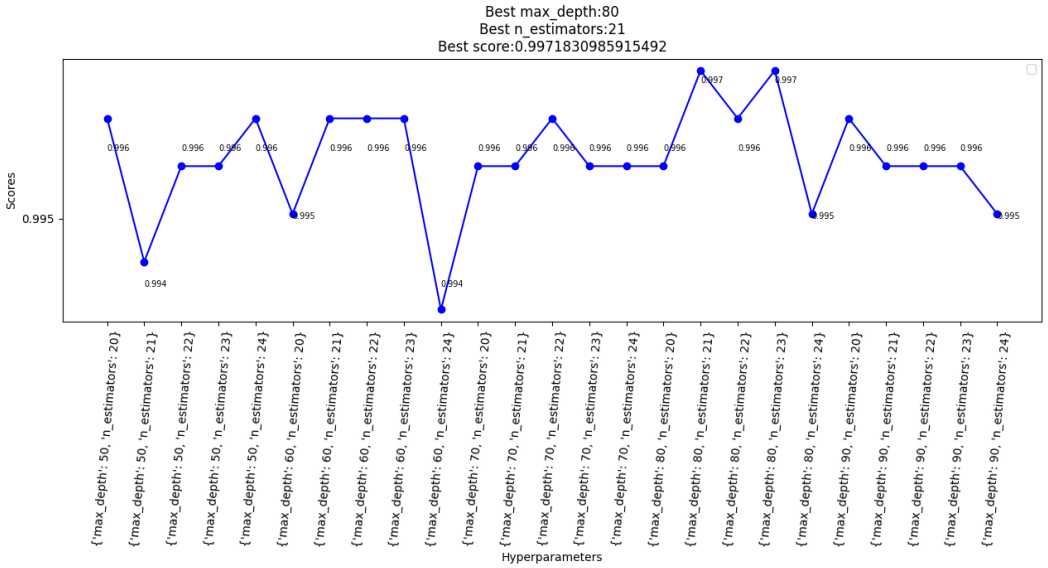

Figure 15: Random Forest M, Y without reduction

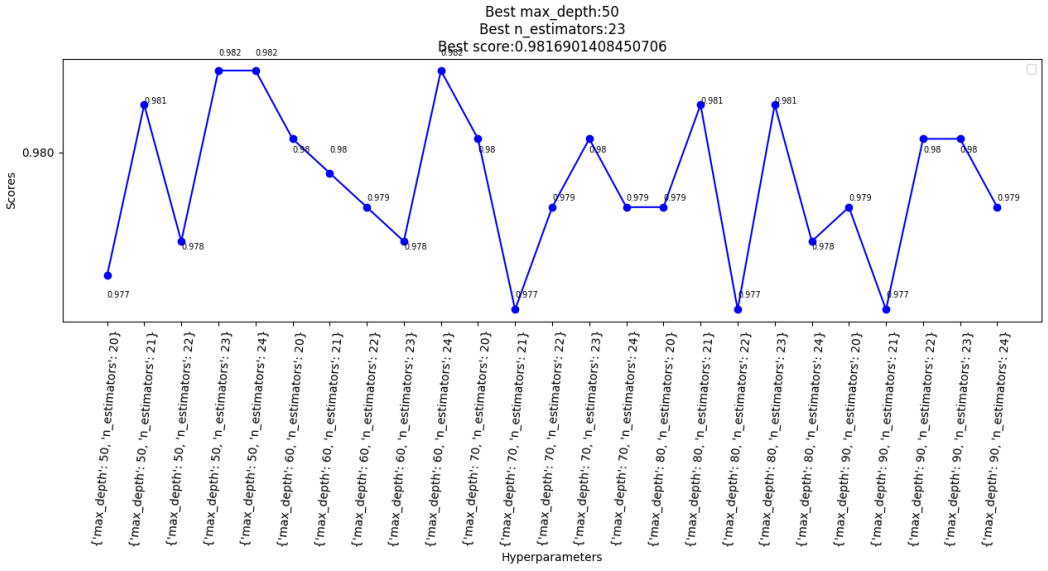

Figure 16: Random Forest M, Y with PCA reduction

2.3.2 Classify M and Y

Then, we move on the second letter pair M and Y. Similarly, we still focus on 2 hyperparameters. But, now we try to set different values.

- *max_depth*: [50, 60, 70, 80, 90].
- *n_estimators*: [20, 21, 22, 23, 24].

The results are shown in Figure 15 and Figure 16.

In Figure 15, we could find that the best score is 0.997. In this situation, the trees we have is 21 and each of them is 80 maximum depth. Through the data we get, it is not hard to see that it is highly likely to get a bad score if we have a larger *n_estimators* and smaller *max_depth*

In Figure 16, we still choose PCA as our dimension reduction method. Here we could see the best score is quite close to the one without reducing the dimension and its value is 0.981. We get the best score when the maximum depth is 50 and we have 23 trees in the forest.

Compared with both experiments, we could see that M and Y is much easier classified than H and K. Also, PCA extracts the features from original data and works well under this circumstance.

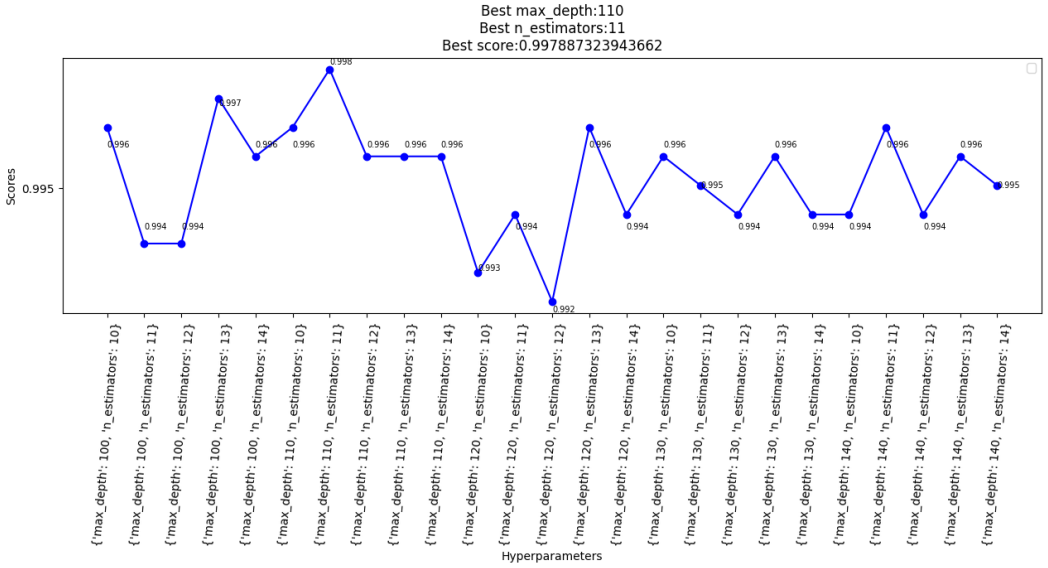

Figure 17: Random Forest M, V without reduction

2.3.3 Classify M and V

We would like to continue our classify and we focus on the letter M and V now. The value of the hyperparameters would be:

- *max_depth*: [100, 110, 120, 130, 140].
- *n_estimators*: [10, 11, 12, 13, 14].

The results would be shown in Figure 17 and Figure 18.

In Figure 17, we could see that the best score is very close to 1 and reaches 0.997. Under such circumstance we have *max_depth* as 110 and *n_estimators* as 11.

In Figure 18, we still select PCA as dimension reduction method. However, it performs pretty much better than the one when we testing the letter M and Y. The best score reaches 0.981 with *max_depth* as 130 and *n_estimators* as 11.

Compared with these two results, we could see that the *max_depth* and *n_estimators* are quite similar when they hit the best score. Thus, we could conclude that when *max_depth* is near 120 and *n_estimators* is near 11, we could get a pretty much better classify result.

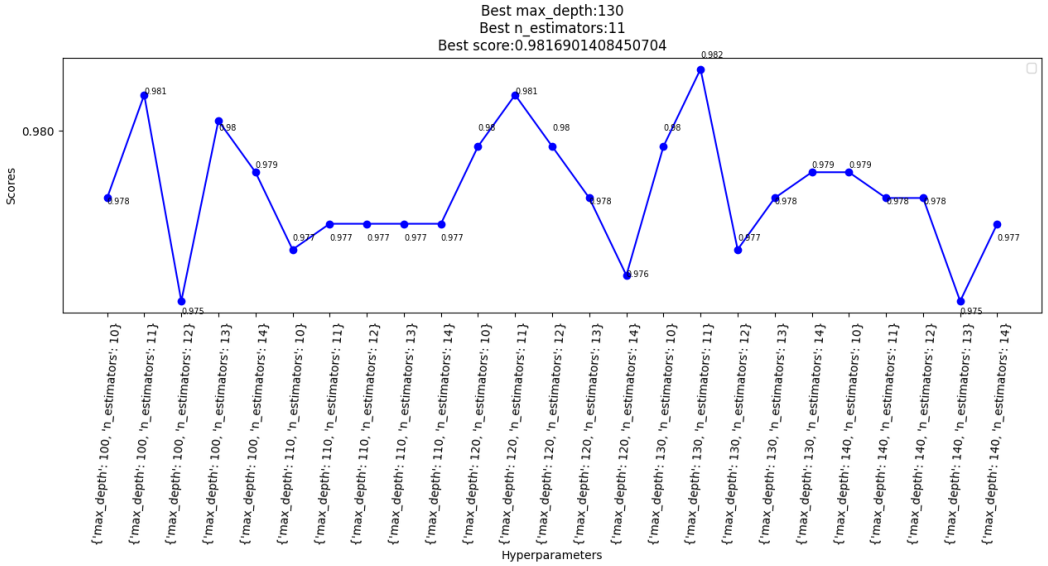

Figure 18: Random Forest M, V with PCA reduction

2.4 SVM

In this section, we will use SVM to train our model. "Support Vector Machine" (SVM) is a supervised machine learning algorithm that is widely used in classification problems. In SVM algorithm, we plot each data item as a point in n-dimensional space, and then, we hope to find the hyper-plane that differentiates the two classes very well to do classification.

The advantage of SVM is quite obvious. It is effective in high dimensional spaces and works well with a clear margin of separation. However, it does not work well when there are lots of noise in the data. Also, when the dataset is large, SVM need pretty much more training time to learn from the data.

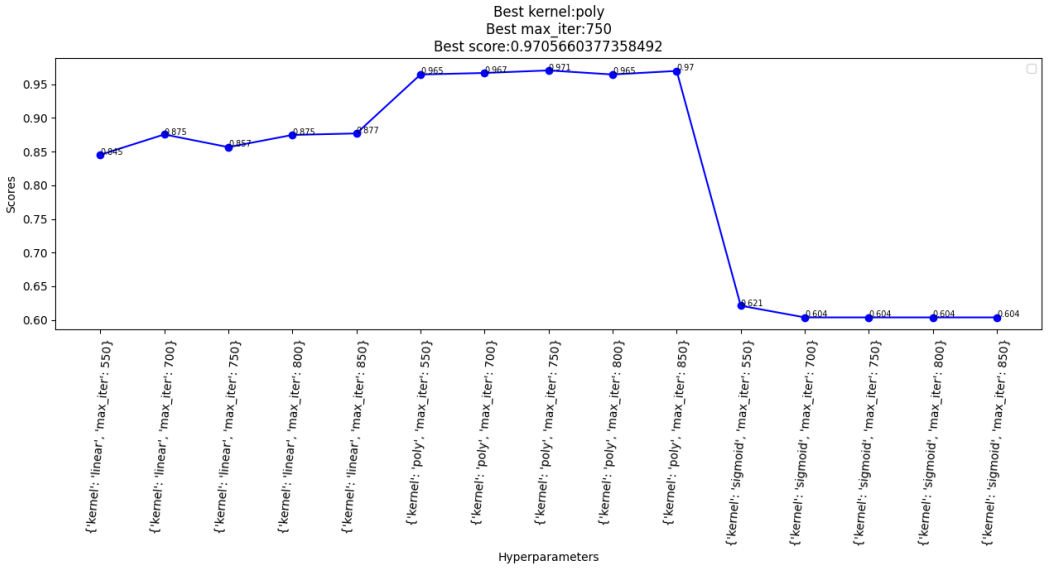

Figure 19: SVM H, K without reduction

2.4.1 Classify H and K

We still choose letter pair H and K first and we would like to implement SVM algorithm to classify H and K. Thanks for the package *sklearn.svm.SVC*, we could easily tune the hyperparameters to do several experiments. Two hyperparameters we choose pay attention to are *kernel*, which specifies the kernel type to be used in the algorithm, and max_iter, which is the hard limit on iterations with in solver. We try these values when classify H and K:

- *kernel*: [linear, poly, sigmoid].
- *max_iter*: [550, 700, 750, 800, 850].

Since we have 3 values for *kernel* and 5 values for *max_iter*, we will generate 15 results after we do the combination. The results are shown in Figure 19 and Figure 20.

In Figure 19, we could see that we could reach 0.970 when we choose poly as kernel algorithm and set the *max_iter* as 750. From the result, it is not easy to see that poly kernel performs well regardless the value of *max_iter*. However, sigmod kernel works bad and get a much lower score. We may choose other hyperparameters to tune the performance if we hope to use sigmod kernel as the main algorithm.

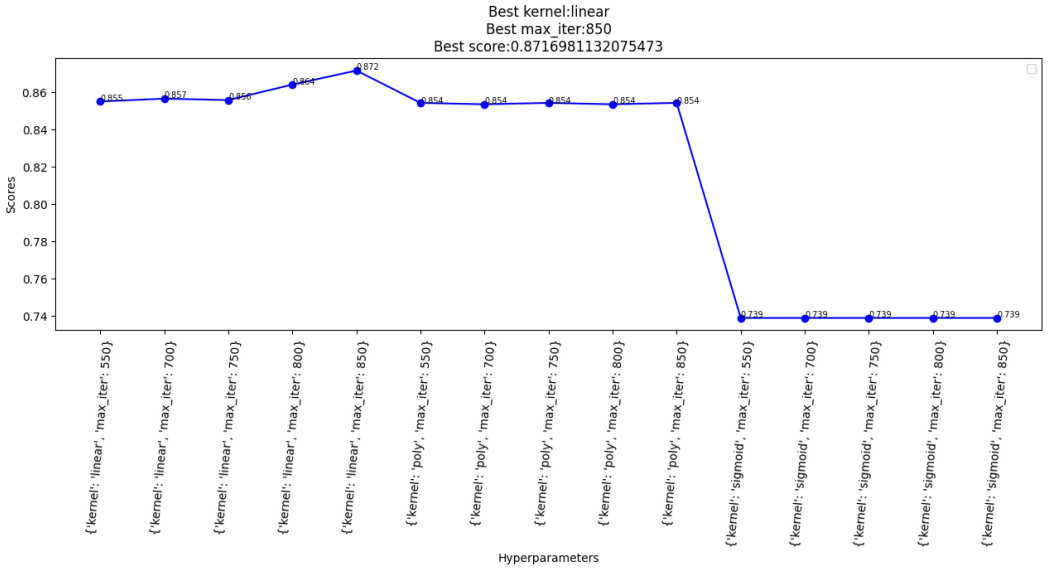

Figure 20: SVM H, K with SVD reduction

In Figure 20, we introduce another feature extraction method, Singular Value Decomposition(SVD). SVD decomposes the original dataset into its constituents, remove redundant features from the dataset, resulting in dimensionality reduction. It is quite suitable for numerical data and quite useful in removing noise. But, the result of SVD is not easy to understood and hard to explain.

From the result, we could see that the best score is 0.871 with linear kernel and 850 maximum iteration. Similar to the previous result, the sigmod function still works bad after we decompose the features from 16 to 4. Besides, SVM may get a better performance with higher maximum iteration.

2.4.2 Classify M and Y

Now we do the experiment on letter pair M and Y. We sill choose *kernel* and *max_iter* as hyperparameters to tune and hope to get a better result. We set the value as:

- *kernel*: [linear, poly, rbf].
- *max_iter*: [800, 900, 1000, 1100, 120].

We could see the results in Figure 21 and Figure 22.

In Figure 21, we get the best score of 0.9929 if we select poly as our kernel function and set the maximum iteration number into 120. Although the result shows that poly function works better than rbf kenel and linear kernel, the difference among their score is slightly. Thus, all of them get the a good result.

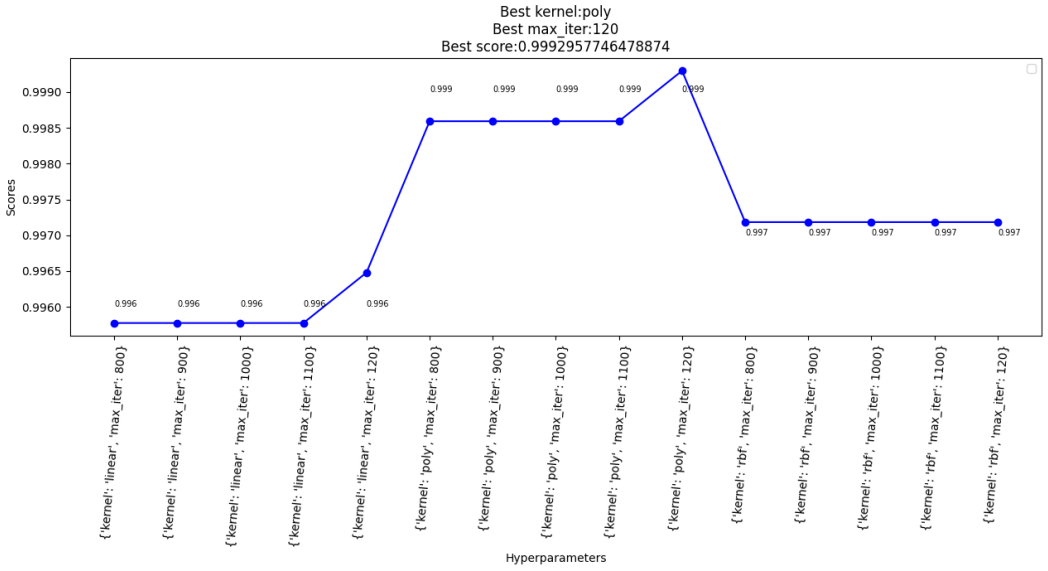

Figure 21: SVM M, Y without reduction

In Figure 22, we still choose SVD to decompose the dimension into 4. From the figure we could see that the best score is 0.9788 with poly kernel and 800 maximum iteration number. Similarly, poly kernel function performs much better than other functions. liner kernel and rbf kernel also performs well in this experiment since their scores are just slighly lower than the poly function.

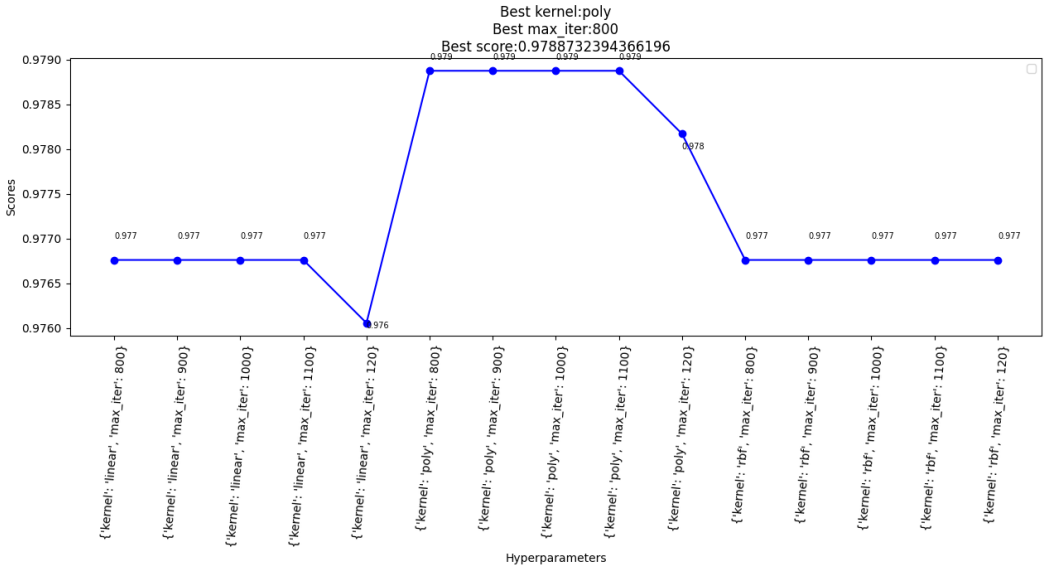

Figure 22: SVM M, Y with SVD reduction

Compared with previous experiments, we may draw the conclusion that whether the performance would be good or bad may not lie much in the maximum iteration since both of the experiments get good results but the first one choose 120 as *max_iter* while the second one is 800. On the contrary, the choose of kernel function is much more important in SVM classification.

2.4.3 Classify M and V

Then, we classify the last pair M and V. This time, we would like to set the maximum iteration into relatively small number and see what will happen. The value would like to be:

- *kernel*: [linear, poly, rbf].
- *max_iter*: [50, 60, 70, 80, 90].

The results are shown in Figure 23 and Figure 24.

In Figure 23, we could see each combination would works well and get a pretty high score. The best score lies in 0.9914 with poly kernel and 90 maximum iteration.

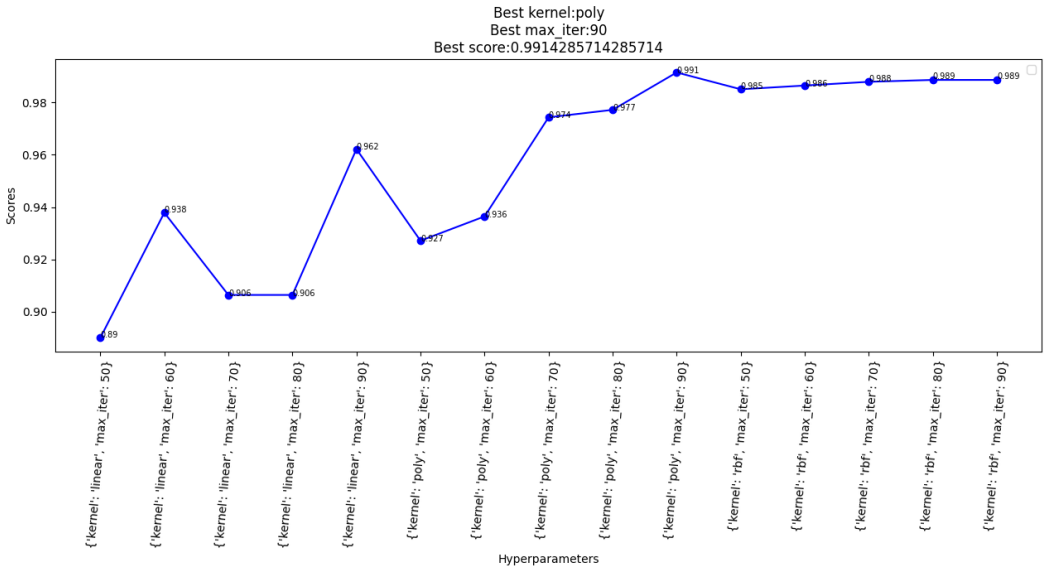

Figure 23: SVM M, V without reduction

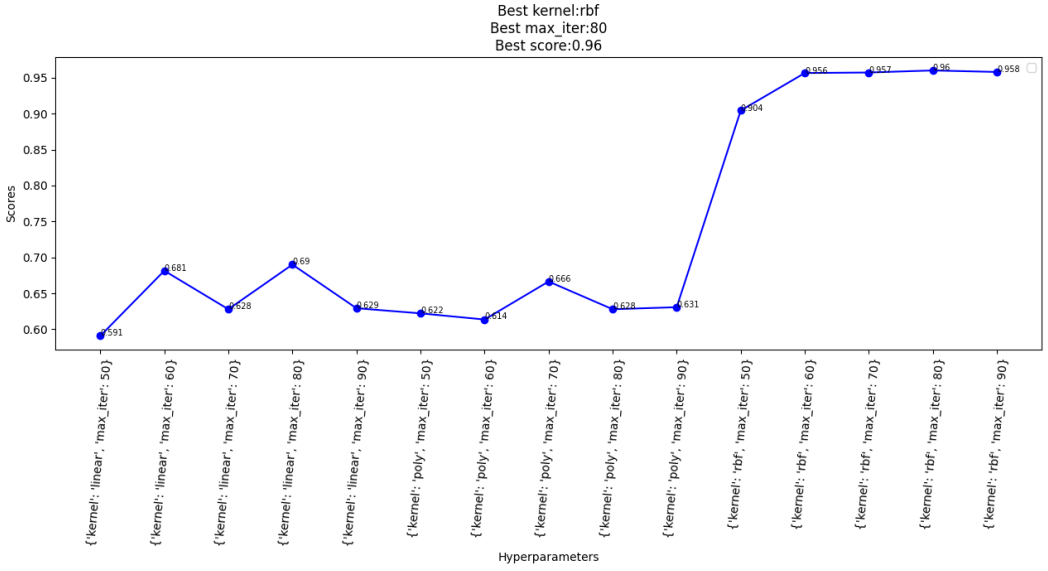

Figure 24: SVM M, V with SVD reduction

In Figure 24, we choose the same dimension reduction method SVD to decompose the 16 features in original dataset into 4. After the reduction, all combinations with higher maximum iteration works

well and others are not. We could get the best score in this situation is 0.96 with 80 maximum iteration number and rbf kernel.

Compared with the previous two experiments, we could see that maximum iteration number could not be too low. For example, we could see that when we set the max_iter as 50, both of the experiments show that the performance it quite bad compared to the one with 80 maximum iteration or 90 maximum iteration. And, the trend shows that the performance could be much better if we choose maximum iteration over 90.

2.5 Artificial Neural Network

Now, we move on the next classification model Artificial Neural Network(ANN). Artificial neural networks are relatively crude electronic networks of neurons based on the neural structure of the brain. They process records one at a time, and learn by comparing their classification of the record with the known actual classification of the record.

The advantage of ANN is quite obvious. Ann is robotic and has strong tolerance ability. Also, the model could be executed in a parallel way with high precise, which makes it one of the popular model in machine learning. But it still meets some disadvantages. For example, the process is hard to understand and the data is not easy to explain. Besides, we it requires much more parameters. In some situations, it may take much more training time and does not get the optimized parameter.

2.5.1 Classify H and K

Again, we start from the letter H and K. In this experiment, we would like to introduce the new classification package $sklearn.neural_network.MLPClassifier$ to help us tune the hyperparameters and do the research. We will focus on $solver$, which means the solver for weight optimization, and $learning_rate$, which is for scheduling for weight updates. Here are the values:

- $solver$: [lbfgs, sgd, adam].
- $learning\_rate$: [constant, invscaling, adaptive].

Since both of the hyperparameters are categorical data, we could totally get 9 combinations and the results are shown in Figure 25 and Figure 26.

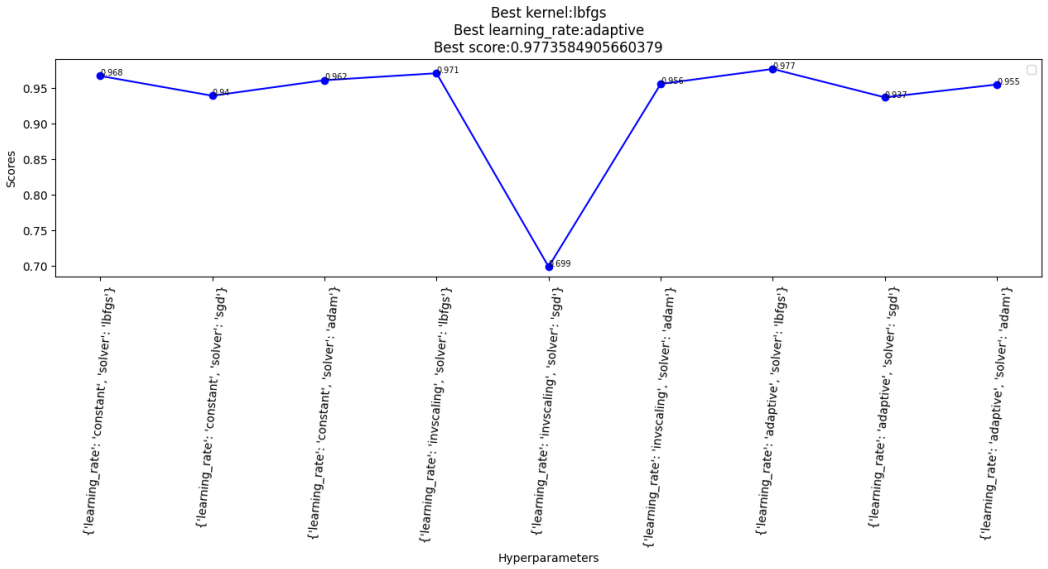

Figure 25: ANN H, K without reduction

In Figure 25, we could see that the best score is 0.977 with lbfgs solver and adaptive learning rate. Also, we could find that the invscaling learning rate and sgd solver contributes to a lower score and its value is near 0.70.

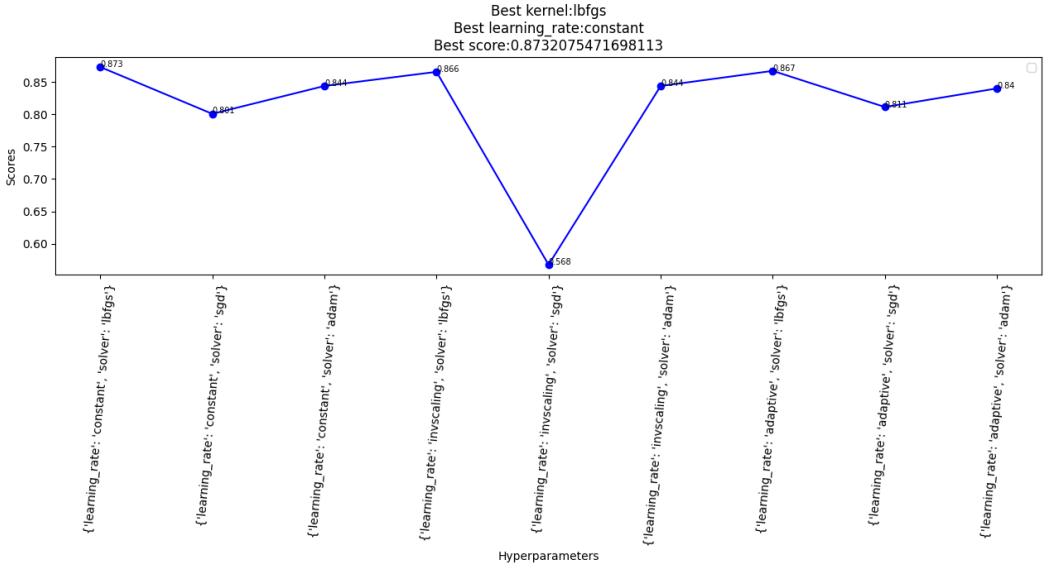

Figure 26: ANN H, K with UMAP reduction

In Figure 26, we would like to use a new method called Uniform Manifold Approximation and Projection(UMAP). The UMAP can preserve as much of the local and global data structure as compared to t-SNE, with a shorter runtime especially dealing with large dataset.

Moreover, UMAP is inspired by k-nearest neighbors algorithm and optimize the results using stochastic gradient descent. It first calculates the distance between the points in high dimensional space, projects them onto the low dimensional space, and calculates the distance again. After that, stochastic gradient descent is used to minimize the difference between distances.

In Figure 26, we could see the result is 0.873 when the solver is lbgfs and learning rate equals constant. Similarly, when the sgd solver and invscaling learning rate is used to train the model, we get a worse score.

Compared with these two experiments, we could find the score with UMAP reduction is lower than the one without reduction. The reason behind that may lie in the fact that the dataset is not very large. Also, since we force to decompose the features from 16 to 4, the features are too refined to fully representing the letter.

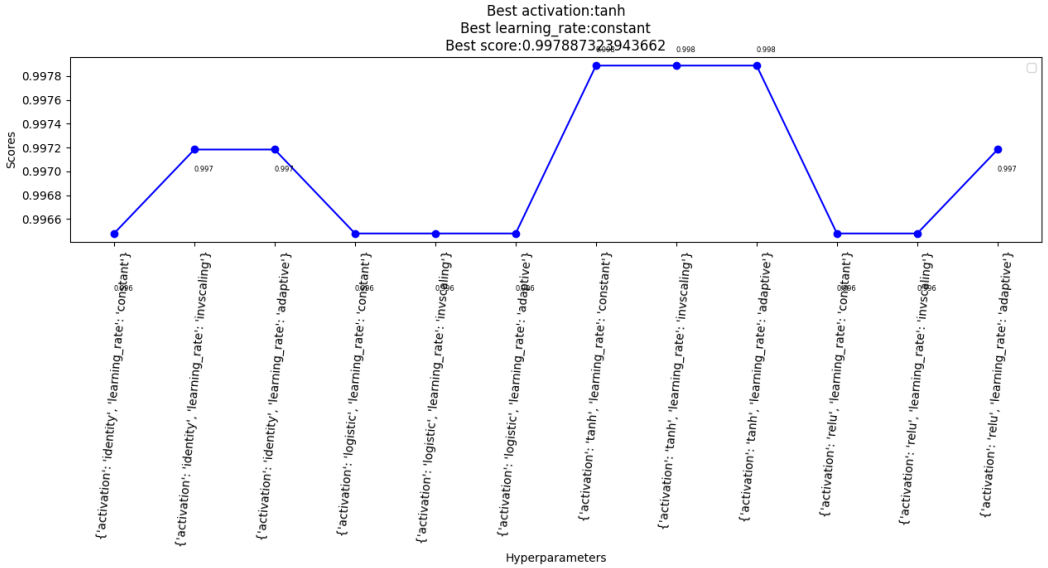

Figure 27: ANN M, Y without reduction

2.5.2 Classify M and Y

Next, we start to classify M and Y. We still tune two hyperparameters, but in this part we replace the *solver* as *activation*, which is the activation function for the hidden layer. Another hyperparameter we choose is *learning_rate*. Here are the values:

- *activation*: [identity, logistic, tanh, relu].
- *learning_rate*: [constant, invscaling, adaptive].

Since we have 4 values for *activation* and 3 values for *learning_rate*, we could get 12 different combinations in this experiment. The result is shown in Figure 27 and Figure 28.

In Figure 27, we could see that the best score reaches 0.997. In this situation, the best activation is tanh and the best learning rate is constant.

On the other hand, Figure 28 shows the result after prepossessing the data with UMAP dimension reduction method. We could find that the best score is 0.999 when we choose relu as activation method and invscaling as learning rate.

Compared with previous two experiments, it is not hard to see the UMAP performs well for all the combinations of the hyperparameters we choose when classifying M and Y. Besides, UMAP also works well in this experiment, which almost equals 1, and get a score a little bit higher than the one without dimension redeuction. In the next experiment, we would like to try another hyperparameter *max_iter* and analyze the results.

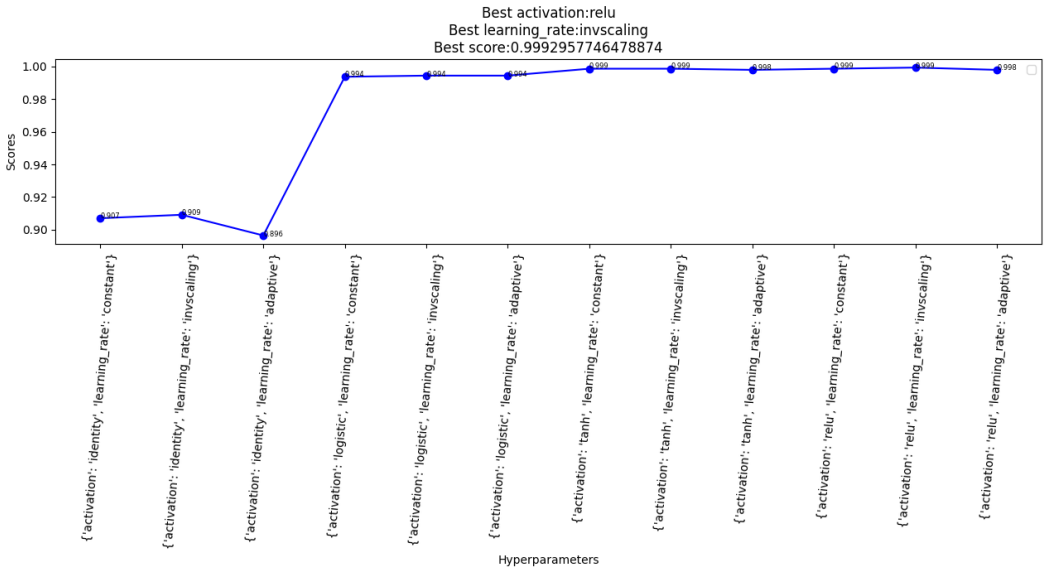

Figure 28: ANN M, Y with UMAP reduction

2.5.3 Classify M and V

We start to classify the last pair M and V using ANN classification. In this part, we may focus on the a different hyperparamter max_iter, which means maximum number of iterations and an old hyperparameter *activation*. Here are the values:

- *activation*: [identity, logistic, tanh, relu].
- *max_iter*: [1000, 1100, 1200, 1300, 1400].

We set the *max_iter* begining from 1000 for a special purpose. The default value for *max_iter* is 200. However, in many circumstances, we may reach the maximum iteration but the optimization may have not converged yet. Thus, we set a relatively higher value to avoid the situation mentioned above.

The results are shown in Figure 29 and Figure 30. Since we have 4 values for *activation* and 5 values for *max_iter*, we could get 20 different combinations.

Figure 29 shows the experiment result without using dimension reduction. We could see the result is pretty much well just as what we hope, and it reaches 0.9978. The best activation would be tanh and the maximum iteration number would be 1100.

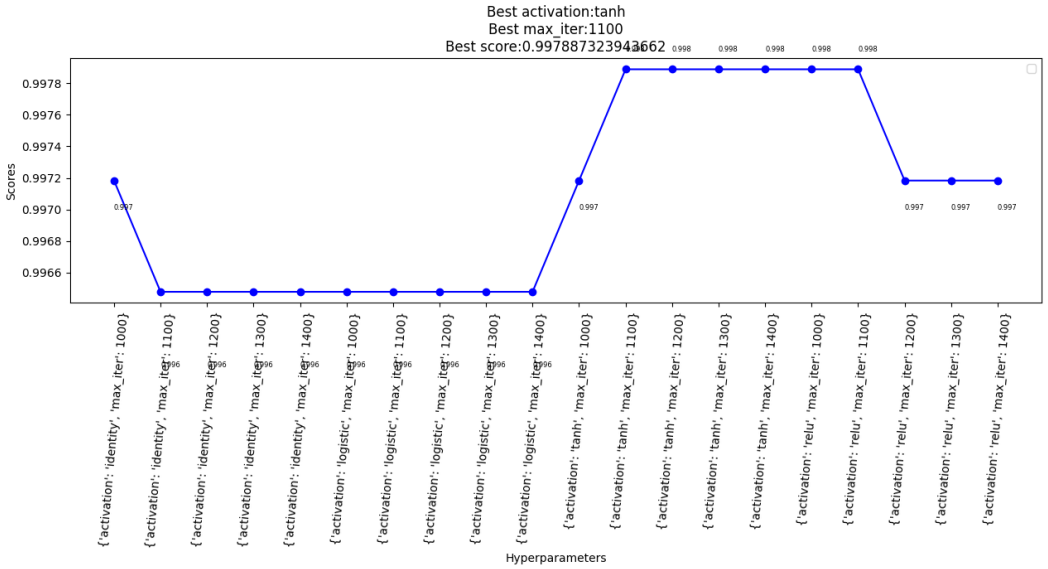

Figure 29: ANN M, V without reduction

In Figure 30, we still apply the UMAP dimension reduction method to decompose the 16 features from original dataset into 4. Under such circumstance, we could see the best score is 0.998, which is a little bit higher than the previous one. Also, the best activation is logistic and the best maximum iteration number is 1000.

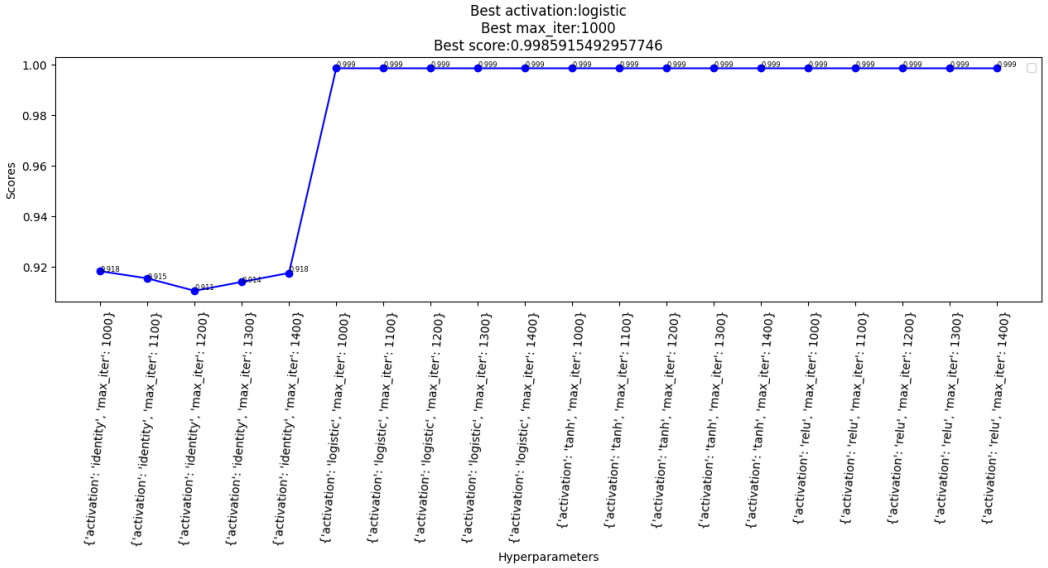

Figure 30: ANN M, V with UMAP reduction

Compared with these two experiments, we could see both of them reaches a great score. However, The former one may be better than the latter one. The reason is that there are more combinations that reaches a score that is close to 1.0 in former one than the latter one. Thus, we could conclude that UMAP does not performs well now, and it loses some of the feature information when decomposing

features in original dataset. Besides, we could see that it is highly possible that ANN could reach a better classification result since the score of most of the combinations we test are close to 1.0.

2.6 Naive Bayes

Now we would like to introduce two additional classification model, Naive Bayes and Gradient boosting method. In this section we would like to talk about Naive Bayes and the Gradient boosting method would be discussed in the next section.

Naive Bayes is a classification technique based on Bayes' Theorem with an assumption of independence among predictors. If I have hundreds of thousands of data points but quite a few variables in your training data set, I would like to choose this method to train the data. For the process, we may first convert the data set into a frequency table, create Likelihood table by finding the probabilities, and then use Naive Bayesian equation to calculate the posterior probability for each class.

Naive Bayes is easily implemented and performs well in multi class prediction. However, it is based on the assumption of independent predictors, and it is almost impossible that we get a set of mutually independent predictors in real world.

2.6.1 Classify H and K

We still start from H and K. There are various of Naive Bayes methods and here we focus on Bernoulli Naive Bayes. Package $sklearn.naive_bayes.BernoulliNB$ gives us a hand to help us tune the hyperparameters. In this part, we would like to pay more attention on *alpha*, which is additive (Laplace/Lidstone) smoothing parameter, and *binarize*, which means the threshold for binarizing of sample features. We would like to set the value as:

- *alpha*: [0.01, 0.1, 0.5, 1.0, 10.0].
- *binarize*: [6, 7, 8, 9, 10].

There are 25 results would be shown in the Figure 31 and Figure 32. In Figure 31, we could see that the best score is 0.831 with best alpha as 0.1 and binarize as 8.

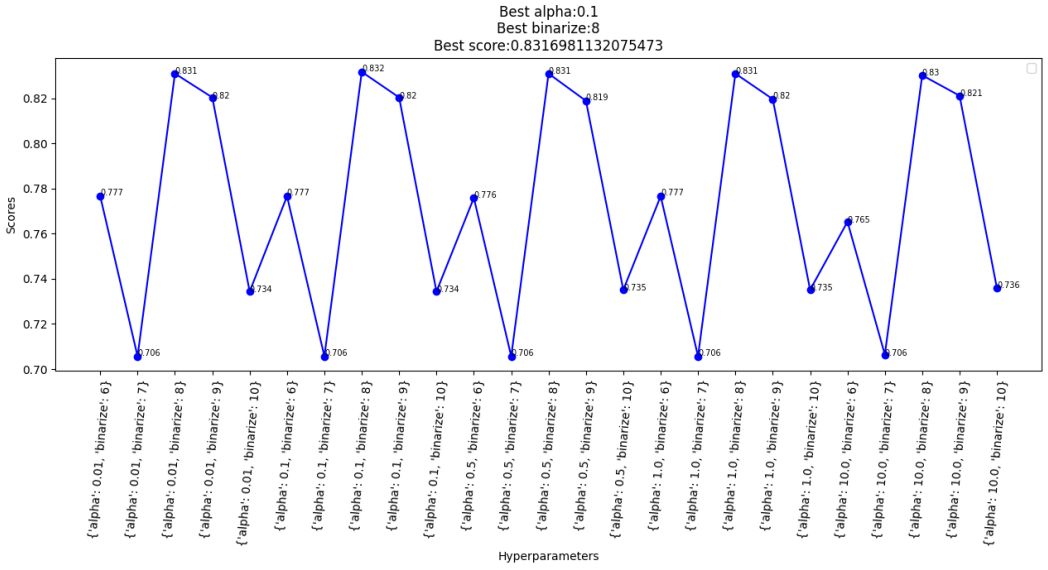

Figure 31: Naive Bayes H, K without reduction

On the other hand, we could see that the best score decreases into 0.6105 after we tune the parameter using PCA dimension reduction method. The result shows that these parameters does not performs well with such combination.

Compared with these two experiments, we could see that the model choose the smallest number in *alpha* and *binarize* to get the best score. In the next experiment, we would like to use a smaller value to tune the model.

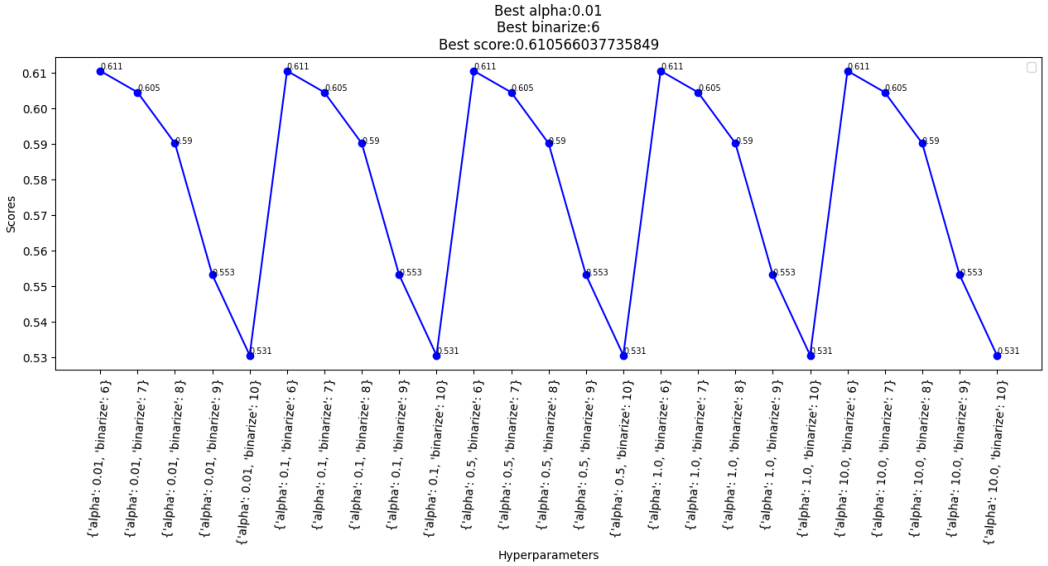

Figure 32: Naive Bayes H, K with PCA reduction

2.6.2 Classify M and Y

We still choose *alpha* and *binarize* as our hyperparameters. In this situation, we would like to set a smaller value and analyze the result. The value would be:

- *alpha*: [0.001, 0.01, 0.15, 0.45, 5.0].
- *binarize*: [1, 2, 3, 4, 5].

The result is shown in Figure 33 and Figure 34. The results are much better than the one when classifying H and K.

In Figure 33, we could see that the best score is 0.954 with best alpha as 5.0 and binarize as 5. The range between the best score and worst score is pretty much larger than pervious models.

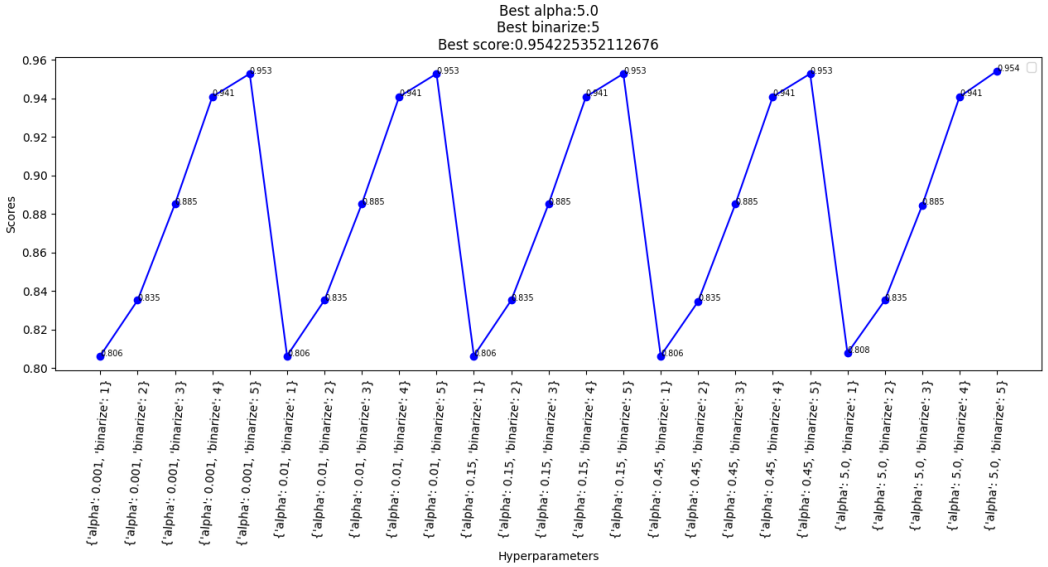

Figure 33: Naive Bayes M, Y without reduction

We still apply PCA to decompose the dimensions and realize the reduction. In Figure 34, we could see the best socre is 0.900. In this situation, the best alpha is 0.001 while the best binarize is 1. The

result has been improved well when compared with the result in the classification of H and K. But it still a little bit far away to what we hope.

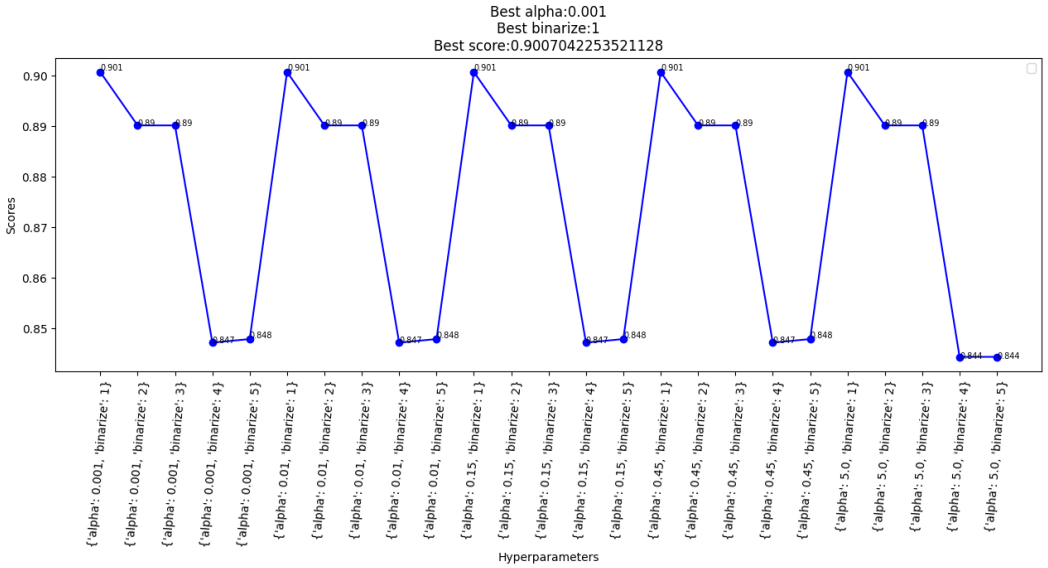

Figure 34: Naive Bayes M, Y with PCA reduction

2.6.3 Classify M and V

We would like to try different values on *alpha* and *binarize* to tune the classification of letter M and V. The value would be:

- *alpha*: [0.001, 0.01, 0.15, 0.45, 5.0].
- *binarize*: [2, 4, 5, 7, 8].

The results are shown in Figure 35 and Figure 36. In Figure 35, we could see that the best score reaches 0.955 when we set the *alpha* as 0.001 and *binarize* as 8.

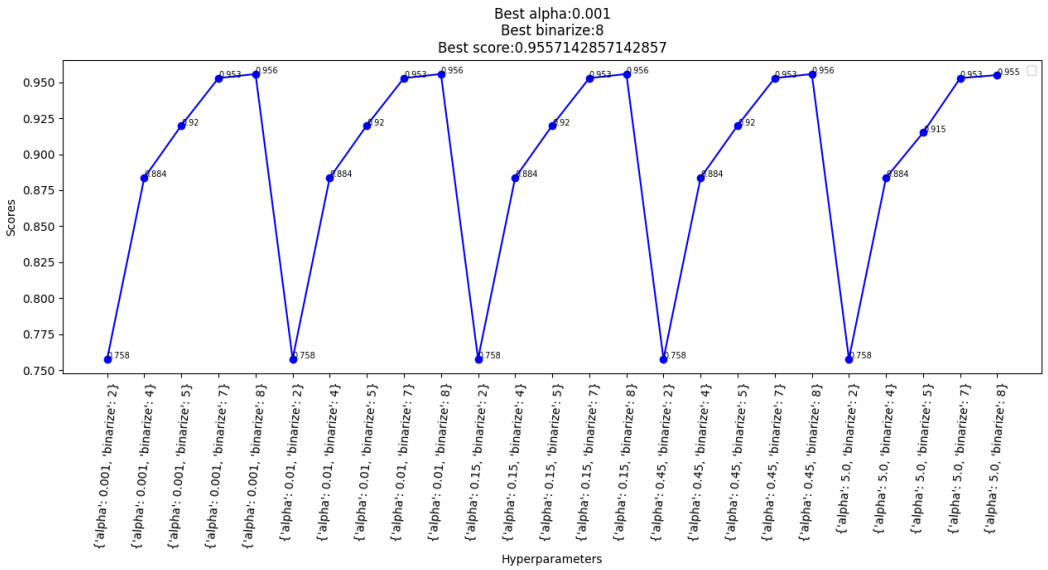

Figure 35: Naive Bayes M, V without reduction

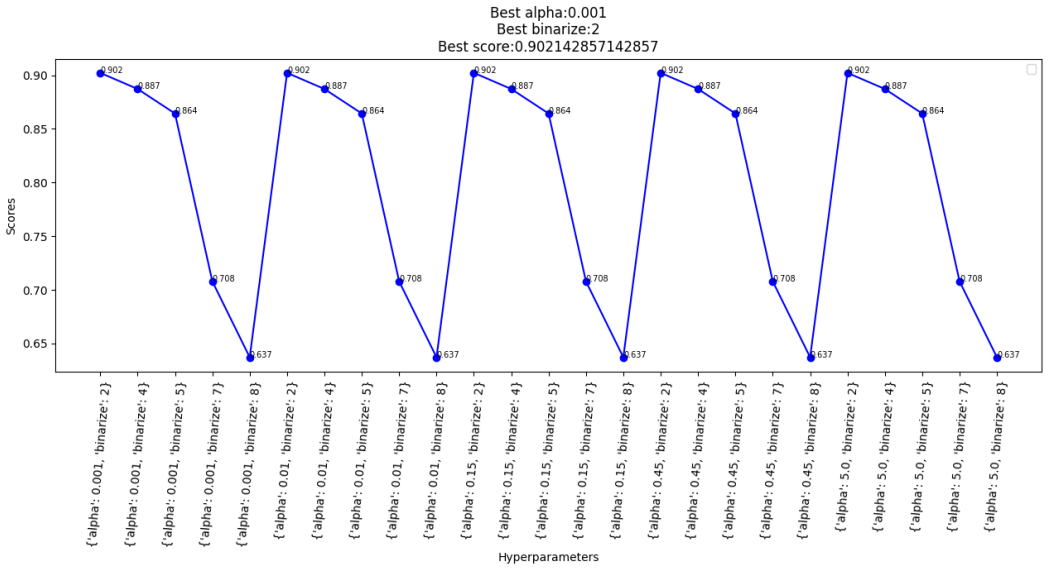

Figure 36: Naive Bayes M, V with PCA reduction

On the other hand, in Figure 36, we show the result after PCA reduction. The best score is 0.902. Here the best *alpha* is still 0.001, but the *binarize* is 2.

There is an interesting thing when we compared the previous two results. We could see that when *alpha* equals 0.001, the score reaches the best when *binarize* is 8 if we do not decompose features but the worst when we prepossessing the data with PCA. Thus, we could draw a conclusion that the value we choose to tune might have totally different meaning in various situations and it may depends on whether we decompose the original features.

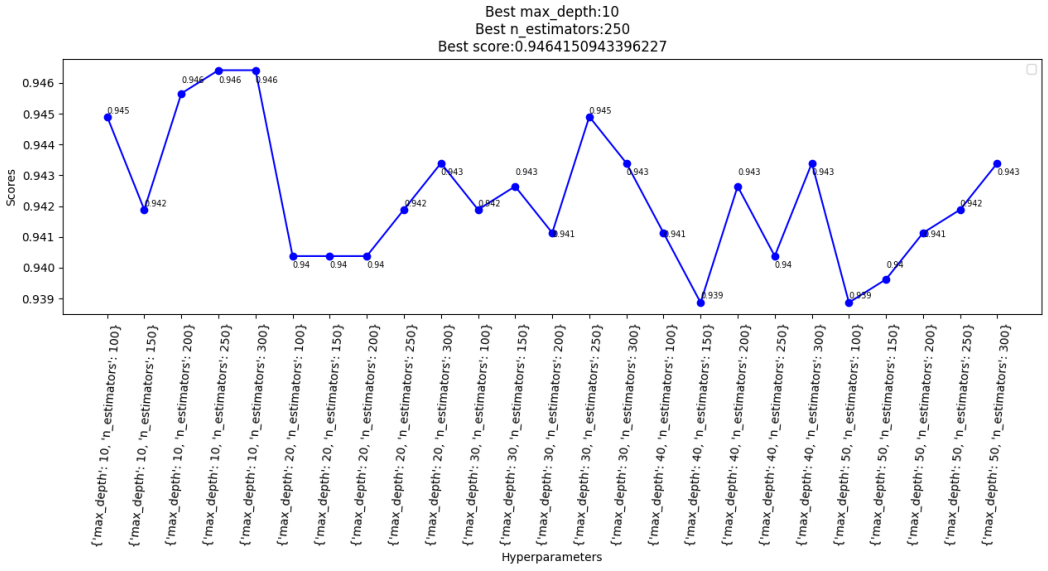

Figure 37: Gradient boosting H, K without reduction

2.7 Gradient boosting

In this section, we would like to introduce another additional training model and it is called Gradient boosting. The main idea behind this algorithm is to build models sequentially and these subsequent models try to reduce the errors of the previous model. The objective here is to minimize this loss

function by adding weak learners using gradient descent.

Gradient boosting method is good at its prediction speed and accuracy, especially when there is a large and complex dataset. However, when the feature dimension is very high, the computational complexity would also be a high value.

2.7.1 Classify H and K

We will still start from the classification of letter H and K. With the $GradientBoostingClassifier$ in *sklearn.ensemble* package, we could tune the hyperparameter to get a better performance.

Here we may pay attention to two hyperparameters: *max_depth*, which is the maximum depth of the individual regression estimators and limits the number of nodes in the tree, and *n_estimators*, which is the number of boosting stages to perform. The value in this classification we test is:

- *max_depth*: [10, 20, 30 , 40, 50].
- *n_estimators*: [100, 150, 200, 250, 300].

The results are shown in Figure 37 and Figure 38.

In Figure 37, we could find that the best score equals to 0.946 when the *n_estimators* is set into 250 and *max_depth* equals 10.

On the other hand, when we use UMAP decomposes the feature first, and then train the dataset, the best score is 0.938 when the *n_estimators* is 10 and *n_estimators* is 200. The result is shown in Figure 38.

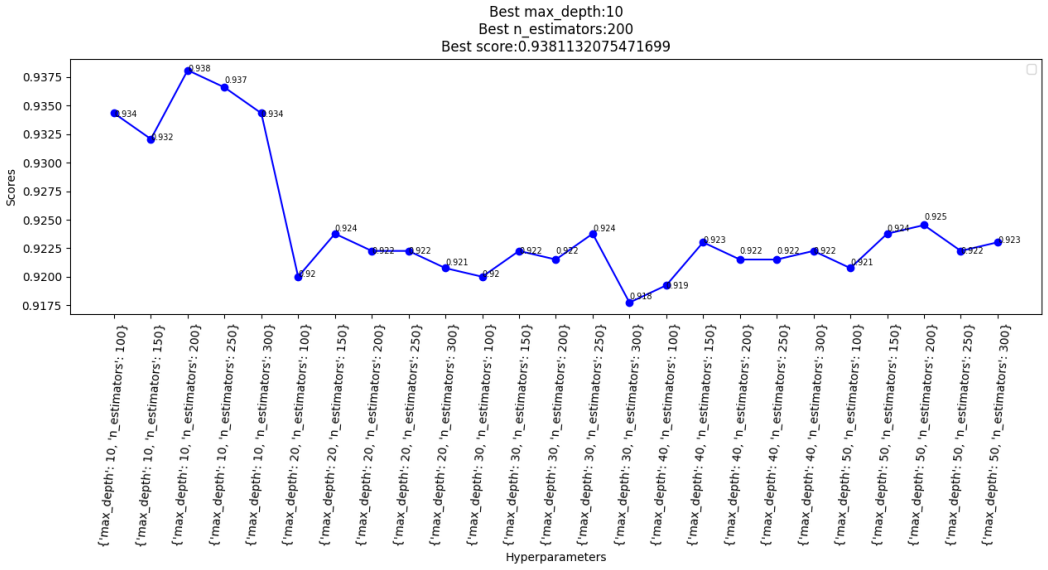

Figure 38: Gradient boosting H, K with UMAP reduction

Compared with the previous experiments, the results of both are not too bad. However, the accuracy of the first one is 0.89 while the second one is 0.94. This says that the UMAP really makes sense when we classify the H and K.

2.7.2 Classify M and Y

Then, we move to classify letter M and Y. In this part, we would like to choose some new values. Here we still choose *n_estimators* and *max_depth* as hyperparameters. The value would be like:

- *max_depth*: [10, 11, 12 , 13, 14].
- *n_estimators*: [210, 220, 230, 240, 260].

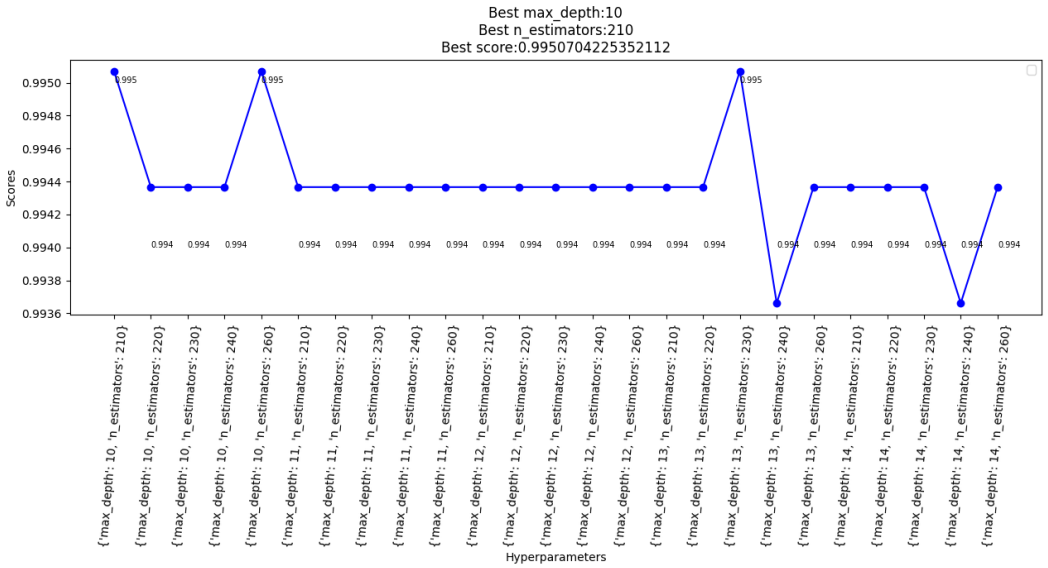

Figure 39: Gradient boosting M, Y without reduction

The results are shown in Figure 39 and Figure 40.

In Figure 39, we could see that all combination performs well and the scores are close to 1.0. The best score is reached when the *max_depth* is 10 and *n_estimators* is 210.

On the other hand, Figure 40 shows the result after we use UMAP to reduce the dimensions. Similarly, the performance are good and the best score is 0.9964.

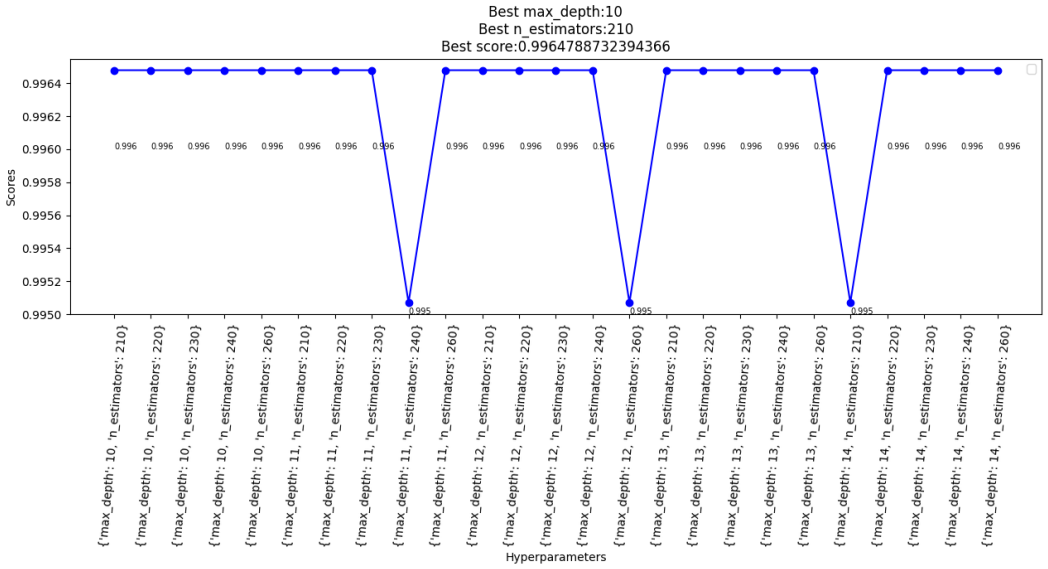

Figure 40: Gradient boosting M, Y with UMAP reduction

Compared with these two experiments, we could draw the conclusion that UMAP also performs well in the classification of M and Y. From the previous experiment result, we could see the best score is reached when we set the *max_depth* equals 10. In the next experiment, we would like to try some smaller *max_depth* to see what will happen.

2.7.3 Classify M and V

In this part, we would like to choose some smaller max_depth to classify M and V. The value would be like:

- *max_depth*: [9, 8, 7, 6, 5].
- *n_estimators*: [270, 280, 290, 300, 310].

The results are shown in Figure 41 and Figure 42.

In Figure 41, we could see that the best score reaches 0.992 when the *max_depth* is 6 and *n_estimators* is 270. Except the best score, all other combinations performs well and the difference among them are extremely slight.

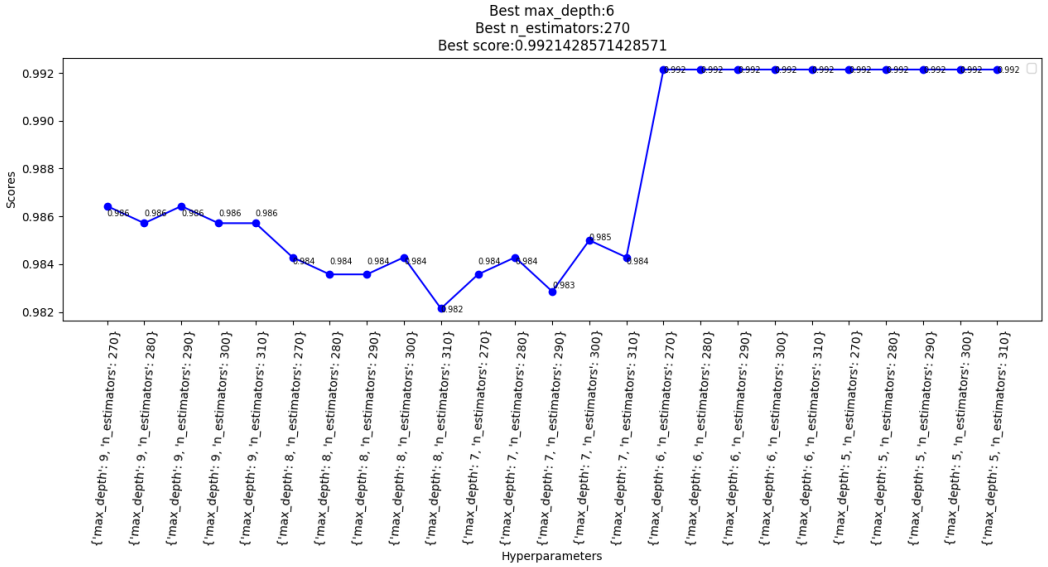

Figure 41: Gradient boosting M, V without reduction

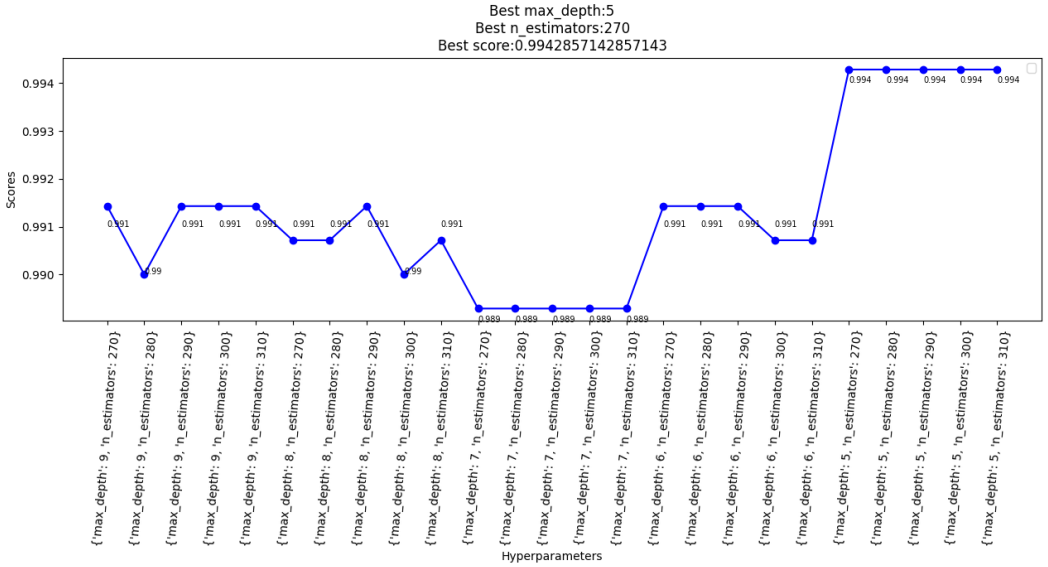

Figure 42: Gradient boosting M, V with UMAP reduction

On the other hand, the Figure 42 shows the result that is after UMAP dimension reduction. The

best score is just a little bit higher than the previous one and reaches 0.997. And, in this situation, the *max_depth* equals 5 and the *n_estimators* equals 270.

Compared with the previous two experiments, we could see that usually we could get a better score when the *n_estimators* is set as a larger number. Also, the smaller *max_depth* may also contributes to the great result when we do the binary classification.

3 Discuss

In this section, we will compare the performance and run time without dimension reduction. Also, the comparison would also be done for the data after dimension reduction and we still focus on the performance and run time. In addition, we will discuss the lessons we have learned among various classifiers.

3.1 Compare the performance and run time without dimension reduction

In this section, we would like to record the time and performance for all the classifiers that do not decompose the features. We would like to compute how many seconds it takes to train and test data. Also, we would like to choose *roc_auc_score*, which is the area under the receiver operating characteristic curve(ROC AUC) from prediction scores, as the metric.

3.1.1 Letter H and K

Table 1 shows the result of H and K. We could see that SVM and ANN works pretty well and get a higher score than others while Naive Bayes works worst. The reason behind it may lies in there is a "zero frequency" in the training data and the probability would be assigned zero. Naive Bayes owns a big improvement since we used different *alpha* smoothing the parameter.

In addition, we could find that we take less time to get the same or higher score after tuning the hyperparameters.

Table 1: H, K without reduction

	Before tune		After tune	
Classifier	Time(s)	Performance	Time(s)	Performance
KNN	0.130	0.938	0.058	0.937
Decision tree	0.014	0.923	0.009	0.904
Random forest	0.204	0.958	0.037	0.952
SVM	0.041	0.979	0.023	0.979
ANN	1.256	0.986	0.570	0.985
Naive Bayes	0.009	0.620	0.008	0.794
Gradient boosting	0.243	0.952	0.659	0.924

Table 2: M, Y without reduction

	Before tune		After tune	
Classifier	Time(s)	Performance	Time(s)	Performance
KNN	0.12	1.0	0.090	1.0
Decision tree	0.009	0.993	0.009	0.987
Random forest	0.189	1.0	0.041	1.0
SVM	0.020	1.0	0.011	1.0
ANN	2.230	1.0	1.153	1.0
Naive Bayes	0.008	0.579	0.007	0.940
Gradient boosting	0.236	1.0	0.591	1.0

3.1.2 Letter M and Y

In Table 2, we could see all the classifiers we choose works well, especially when we choose SVM, ANN, Random Forest and KNN. Again, we spend less time to get the same or a higher score than the one before tuning the hyperparameter.

Compared with H and K, M and Y get a higher performance, which means M and Y are much easier to be classified than H and K. Also, the quality of data may contributes to the high score, and we need more data to verify this assumption.

3.1.3 Letter M and V

Table 3 shows the comparison of different classifiers when testing M and V. Similar to M and Y, M and V gets a higher average score and all the classifiers works well.

Among all three pairs, we could find the fact that SVM takes the least time and get a higher score. However, although ANN works equally well, it usually cost the most time. The result meets the character of SVM and ANN respectively and the result is the same as what we hope to see.

Table 3: M, V without reduction

	Before tune		After tune	
Classifier	Time(s)	Performance	Time(s)	Performance
KNN	0.119	0.987	0.092	0.980
Decision tree	0.010	0.993	0.009	0.980
Random forest	0.193	1.0	0.214	1.0
SVM	0.023	0.987	0.012	0.993
ANN	0.936	0.987	1.934	0.987
Naive Bayes	0.011	0.589	0.008	0.980
Gradient boosting	0.260	1.0	0.895	1.0

3.2 Compare the performance and run time after dimension reduction

In this section, we introduce several dimension reduction methods and apply them into different classifiers. We still record the time and performance to do the comparison. The time would be evaluated by seconds and *roc_auc_score* would be chosen as the metric to evaluate the performance.

Table 4: H, K with reduction

		Before tune		After tune	
Classifier	Reduction	Time(s)	Performance	Time(s)	Performance
KNN	Backward Feature Elimination	0.088	0.912	0.055	0.912
Decision tree	Decision Tree	0.007	0.938	0.008	0.917
Random forest	PCA	0.256	0.905	0080.404	0.912
SVM	SVD	0.078	0.892	0.029	0.871
ANN	UMAP	0.852	0.760	0.561	0.868
Naive Bayes	PCA	0.007	0.756	0.006	0.595
Gradient boosting	UMAP	0.262	0.884	1.271	0.924

3.2.1 Letter H and K

We start from letter H and K. The results are shown in the Table 4. It is not hard to see that Backward Feature Elimination method and Decision Tree method take less time than PCA, UMAP and SVD. It is understandable since the first two methods are feature selection methods and they just remove the

features with lower importance they render. However, PCA, UMAP and SVD are feature extraction methods and they extract features from original data and generate new one. The new features may combine more features compared to feature selection methods.

Also, there is a little bit decline in performance when we decompose the feature. The reason might be lie in the fact that we may lose some feature information so that we could not classify some of the data correctly.

3.2.2 Letter M and Y

Then, we would like to move on letter M and Y. The results are shown in Table 5. We could see that although ANN spends much more time than other classifiers, it gets the best score. Classifiers other than ANN also works well and the performance gets a slight improvement when compared to the one before tuning the hyperparameters.

Through the previous two experiments, we could see that dimension reduction really works even though the improvement is slight. If the original dataset has much more features and training or testing data, the effect would be much more obvious.

Table 5: M, Y with reduction

		Before tune		After tune	
Classifier	Reduction	Time(s)	Performance	Time(s)	Performance
KNN	Forward Feature Selection	0.608	0.987	0.104	0.987
Decision tree	Lasso Regression	0.007	0.993	0.007	0.987
Random forest	PCA	0.220	0.987	0.160	0.993
SVM	SVD	0.016	1.0	0.013	0.993
ANN	UMAP	0.861	1.0	1.054	1.0
Naive Bayes	PCA	0.006	0.823	0.006	0.923
Gradient boosting	UMAP	0.202	1.0	0.526	0.993

Table 6: M, V with reduction

		Before tune		After tune	
Classifier	Reduction	Time	Performance	Time	Performance
KNN	Random Forest	0.059	1.0	0.907	1.0
Decision tree	Random Forest	0.007	0.993	0.008	1.0
Random forest	PCA	0.220	0.967	0.217	0.967
SVM	SVD	0.039	0.967	0.016	0.967
ANN	UMAP	0.927	0.980	1.668	0.974
Naive Bayes	PCA	0.006	0.794	0.005	0.910
Gradient boosting	UMAP	0.215	0.980	0.615	0.980

3.2.3 Letter M and V

Table 6 shows the comparison result of M and V. M and V also get a pretty nice score and works well when we decompose the features and realize the dimension reduction.

Again, SVM takes the least time the ANN costs the most. Thus, we could draw a conclusion that if we hope to get a real-time result with a relatively higher score, we would like to choose SVM or Decision Tree first and then tune the hyperparameters. However, if what we care is the performance and time will not be a factor we pay attention to, we could choose ANN to classify the letter.

3.3 Lessons learned

Just as discussed above, I would choose SVM if I hope to get the result in a short time. However, when the data has much more features with large quantity and training time would be a factor that could be tolerated, I would choose ANN.

Dimension reduction would have an effect on the run time. Usually, we would like to get a less run time when we choose feature selection method. However, if we choose feature extraction method to project the data from one dimension to another, like high dimension to low dimension or low dimension to high dimension, we would like to cost more run time. The reason is that feature extract method contains more feature information, and thus, we may spend more time.

If I were given a new dataset, the first thing I would like to do is to prepossessing the data by removing the unrelated ones. Then, I would like to use decision tree or SVM to get a quick result. According to the result, I would like to try other classifiers and decompose the data into less features. Finally, I would choose one of the classifiers and tune the hyperparameters to get the best results.

Through this project, we could see that there are lost of combination between classifiers and dimension reduction methods. We could not say which one is the best or the worst. What we need to do is to choose the suitable classifier and dimension reduction method according to the goal we hope to achieve and the dataset we have. Besides, we need more patience and spend more time on tuning the hyperparameters if we hope to get the best classification result.

4 Code

The complete code could be find on the github and here is the link https://github.com/weijj27/CSE514-data-mining.

For each of the *classifier.py* file, we have already the code for classification before tuning the hyperparameters and after tuning the hyperparameters. Also, the code of reduction method we choose for each classifier is also included in the file. Since we have done some slightly prepossessing for the original data, like adding title for each features or generating the dataset after feature selection, I really suggest you clone the code from github if you hope to run it on your local machine.

Here is a brief description of the code:

- *knn.py*: code for KNN classifier.
- *decision_tree.py*: code for Decision Tree classifier.
- *random_forest.py*: code for Random Forest classifier.
- *svm.py*: code for SVM classifier.
- *ann.py*: code for ANN classifier.
- *naive_bayes.py*: code for Naive Bayes classifier.
- *gradient_boosting.py*: code for Gradient boosting classifier.
- *reduction_methods.py*: code for all 8 dimension reduction methods we use in this project.