Introduction

1. The problem is dimension reduction and the comparison of different classification models for identifying letters in various fonts. It can be done in 3 steps: data preprocessing, model fitting, and dimension reduction.
   In order to complete the task efficiently, the optimal classifier needs to be selected. And there are many factors that can be considered to choosing the best classifier. The factors are times costs, accuracy, memory usage, ease of implementation, degree of fit, and some other factors.
2. Binary classification is the task of classifying the elements of a set into two groups on the basis of a classification rule. For the third problem, I choose 'A' and 'B'. I think pairs that are more similar are harder to classify like they have the same width and the same height. So 'H' and 'K' are the hardest to classify, and 'M' and 'Y' are the easiest to classify.
3. The dimension reduction is good for reducing time costs, but it will cause the accuracy to decrease. The more dimension reduced, the lower accuracy I got.

Results

1. KNN
   1) The kNN algorithm is when predicting a new value x, judging the category of x according to the category of the K points closest to it.
      Advantage: Easy to implement.
      Disadvantage: When the data dimension is high, the amount of computation is large; samples with close distances may not belong to the same category.

   2) Pair 1: H and K

```
0.925 for {'n_neighbors': 2, 'p': 1}
0.909 for {'n_neighbors': 2, 'p': 2}
0.904 for {'n_neighbors': 2, 'p': 3}
0.902 for {'n_neighbors': 2, 'p': 4}
0.898 for {'n_neighbors': 2, 'p': 5}
0.948 for {'n_neighbors': 3, 'p': 1}
0.931 for {'n_neighbors': 3, 'p': 2}
0.925 for {'n_neighbors': 3, 'p': 3}
0.920 for {'n_neighbors': 3, 'p': 4}
0.916 for {'n_neighbors': 3, 'p': 5}
0.943 for {'n_neighbors': 5, 'p': 1}
0.922 for {'n_neighbors': 5, 'p': 2}
0.915 for {'n_neighbors': 5, 'p': 3}
0.905 for {'n_neighbors': 5, 'p': 4}
0.905 for {'n_neighbors': 5, 'p': 5}
0.934 for {'n_neighbors': 8, 'p': 1}
0.900 for {'n_neighbors': 8, 'p': 2}
0.888 for {'n_neighbors': 8, 'p': 3}
0.889 for {'n_neighbors': 8, 'p': 4}
0.886 for {'n_neighbors': 8, 'p': 5}
0.922 for {'n_neighbors': 10, 'p': 1}
0.891 for {'n_neighbors': 10, 'p': 2}
0.891 for {'n_neighbors': 10, 'p': 3}
0.882 for {'n_neighbors': 10, 'p': 4}
0.877 for {'n_neighbors': 10, 'p': 5}
```

   Best result in cross-validation: 0.9479245283018868.

   Pair 2: M and Y

```
0.997 for {'n_neighbors': 2, 'p': 1}
0.998 for {'n_neighbors': 2, 'p': 2}
0.999 for {'n_neighbors': 2, 'p': 3}
0.998 for {'n_neighbors': 2, 'p': 4}
0.997 for {'n_neighbors': 2, 'p': 5}
0.998 for {'n_neighbors': 3, 'p': 1}
0.998 for {'n_neighbors': 3, 'p': 2}
0.999 for {'n_neighbors': 3, 'p': 3}
0.999 for {'n_neighbors': 3, 'p': 4}
0.999 for {'n_neighbors': 3, 'p': 5}
0.997 for {'n_neighbors': 5, 'p': 1}
0.998 for {'n_neighbors': 5, 'p': 2}
0.998 for {'n_neighbors': 5, 'p': 3}
0.997 for {'n_neighbors': 5, 'p': 4}
0.998 for {'n_neighbors': 5, 'p': 5}
0.998 for {'n_neighbors': 8, 'p': 1}
0.998 for {'n_neighbors': 8, 'p': 2}
0.996 for {'n_neighbors': 8, 'p': 3}
0.996 for {'n_neighbors': 8, 'p': 4}
0.995 for {'n_neighbors': 8, 'p': 5}
0.997 for {'n_neighbors': 10, 'p': 1}
0.997 for {'n_neighbors': 10, 'p': 2}
0.996 for {'n_neighbors': 10, 'p': 3}
0.995 for {'n_neighbors': 10, 'p': 4}
0.994 for {'n_neighbors': 10, 'p': 5}
```

Best result in cross-validation: 0.9985915492957746.

Pair 3: A and B

```
1.000 for {'n_neighbors': 2, 'p': 1}
0.998 for {'n_neighbors': 2, 'p': 2}
0.998 for {'n_neighbors': 2, 'p': 3}
0.998 for {'n_neighbors': 2, 'p': 4}
0.998 for {'n_neighbors': 2, 'p': 5}
0.998 for {'n_neighbors': 3, 'p': 1}
0.999 for {'n_neighbors': 3, 'p': 2}
0.998 for {'n_neighbors': 3, 'p': 3}
0.999 for {'n_neighbors': 3, 'p': 4}
0.999 for {'n_neighbors': 3, 'p': 5}
0.998 for {'n_neighbors': 5, 'p': 1}
0.998 for {'n_neighbors': 5, 'p': 2}
0.998 for {'n_neighbors': 5, 'p': 3}
0.998 for {'n_neighbors': 5, 'p': 4}
0.998 for {'n_neighbors': 5, 'p': 5}
0.999 for {'n_neighbors': 8, 'p': 1}
0.999 for {'n_neighbors': 8, 'p': 2}
0.999 for {'n_neighbors': 8, 'p': 3}
0.996 for {'n_neighbors': 8, 'p': 4}
0.996 for {'n_neighbors': 8, 'p': 5}
0.999 for {'n_neighbors': 10, 'p': 1}
0.999 for {'n_neighbors': 10, 'p': 2}
0.999 for {'n_neighbors': 10, 'p': 3}
0.996 for {'n_neighbors': 10, 'p': 4}
0.996 for {'n_neighbors': 10, 'p': 5}
```

Best result in cross-validation: 1.0.

3) The method is Low Variance of Simple Quality Filtering. And the threshold is 4.85 to get the final 4 dimensions for the pair1, threshold is 8 for pair2, threshold is 4.815 for pair3. If a feature's value changes very little across the samples, it's probably not informative, and the method will put these features away.

4) Pair 1: H and K

```
0.257 for {'n_neighbors': 2, 'p': 1}
0.258 for {'n_neighbors': 2, 'p': 2}
0.258 for {'n_neighbors': 2, 'p': 3}
0.258 for {'n_neighbors': 2, 'p': 4}
0.260 for {'n_neighbors': 2, 'p': 5}
0.288 for {'n_neighbors': 3, 'p': 1}
0.292 for {'n_neighbors': 3, 'p': 2}
0.292 for {'n_neighbors': 3, 'p': 3}
0.292 for {'n_neighbors': 3, 'p': 4}
0.292 for {'n_neighbors': 3, 'p': 5}
0.287 for {'n_neighbors': 5, 'p': 1}
0.286 for {'n_neighbors': 5, 'p': 2}
0.288 for {'n_neighbors': 5, 'p': 3}
0.288 for {'n_neighbors': 5, 'p': 4}
0.289 for {'n_neighbors': 5, 'p': 5}
0.282 for {'n_neighbors': 8, 'p': 1}
0.282 for {'n_neighbors': 8, 'p': 2}
0.280 for {'n_neighbors': 8, 'p': 3}
0.280 for {'n_neighbors': 8, 'p': 4}
0.283 for {'n_neighbors': 8, 'p': 5}
0.287 for {'n_neighbors': 10, 'p': 1}
0.283 for {'n_neighbors': 10, 'p': 2}
0.282 for {'n_neighbors': 10, 'p': 3}
0.282 for {'n_neighbors': 10, 'p': 4}
0.287 for {'n_neighbors': 10, 'p': 5}
```

Best result in cross-validation: 0.2920754716981132.

Pair 2: M and Y

```
0.962 for {'n_neighbors': 2, 'p': 1}
0.961 for {'n_neighbors': 2, 'p': 2}
0.961 for {'n_neighbors': 2, 'p': 3}
0.961 for {'n_neighbors': 2, 'p': 4}
0.961 for {'n_neighbors': 2, 'p': 5}
0.963 for {'n_neighbors': 3, 'p': 1}
0.963 for {'n_neighbors': 3, 'p': 2}
0.962 for {'n_neighbors': 3, 'p': 3}
0.962 for {'n_neighbors': 3, 'p': 4}
0.962 for {'n_neighbors': 3, 'p': 5}
0.959 for {'n_neighbors': 5, 'p': 1}
0.959 for {'n_neighbors': 5, 'p': 2}
0.958 for {'n_neighbors': 5, 'p': 3}
0.958 for {'n_neighbors': 5, 'p': 4}
0.958 for {'n_neighbors': 5, 'p': 5}
0.962 for {'n_neighbors': 8, 'p': 1}
0.958 for {'n_neighbors': 8, 'p': 2}
0.957 for {'n_neighbors': 8, 'p': 3}
0.958 for {'n_neighbors': 8, 'p': 4}
0.958 for {'n_neighbors': 8, 'p': 5}
0.958 for {'n_neighbors': 10, 'p': 1}
0.957 for {'n_neighbors': 10, 'p': 2}
0.957 for {'n_neighbors': 10, 'p': 3}
0.957 for {'n_neighbors': 10, 'p': 4}
0.957 for {'n_neighbors': 10, 'p': 5}
```

Best result in cross-validation: 0.9626760563380282.

Pair 3: A and B

```
0.478 for {'n_neighbors': 2, 'p': 1}
0.478 for {'n_neighbors': 2, 'p': 2}
0.478 for {'n_neighbors': 2, 'p': 3}
0.478 for {'n_neighbors': 2, 'p': 4}
0.478 for {'n_neighbors': 2, 'p': 5}
0.499 for {'n_neighbors': 3, 'p': 1}
0.499 for {'n_neighbors': 3, 'p': 2}
0.499 for {'n_neighbors': 3, 'p': 3}
0.499 for {'n_neighbors': 3, 'p': 4}
0.499 for {'n_neighbors': 3, 'p': 5}
0.522 for {'n_neighbors': 5, 'p': 1}
0.520 for {'n_neighbors': 5, 'p': 2}
0.520 for {'n_neighbors': 5, 'p': 3}
0.520 for {'n_neighbors': 5, 'p': 4}
0.520 for {'n_neighbors': 5, 'p': 5}
0.533 for {'n_neighbors': 8, 'p': 1}
0.536 for {'n_neighbors': 8, 'p': 2}
0.538 for {'n_neighbors': 8, 'p': 3}
0.537 for {'n_neighbors': 8, 'p': 4}
0.537 for {'n_neighbors': 8, 'p': 5}
0.530 for {'n_neighbors': 10, 'p': 1}
0.535 for {'n_neighbors': 10, 'p': 2}
0.533 for {'n_neighbors': 10, 'p': 3}
0.535 for {'n_neighbors': 10, 'p': 4}
0.535 for {'n_neighbors': 10, 'p': 5}
```

Best result in cross-validation: 0.5375166410650282.

5) Hyperparameters are ‘n_neighbors’ and ‘p’.

2. Decision Tree
   1) A Decision Tree is a tree-like structure in which each internal node represents a test output on each attribute in a test, and each leaf node represents a category.
      Advantage: less time complexity, easy to understand and explain.
      Disadvantage: easy to overfit, ignore dependencies between attributes.
   2) Pair 1: H and K

```
0.864 for {'max_depth': 3, 'random_state': 1}
0.863 for {'max_depth': 3, 'random_state': 4}
0.863 for {'max_depth': 3, 'random_state': 6}
0.863 for {'max_depth': 3, 'random_state': 7}
0.864 for {'max_depth': 3, 'random_state': 9}
0.894 for {'max_depth': 4, 'random_state': 1}
0.894 for {'max_depth': 4, 'random_state': 4}
0.893 for {'max_depth': 4, 'random_state': 6}
0.894 for {'max_depth': 4, 'random_state': 7}
0.896 for {'max_depth': 4, 'random_state': 9}
0.903 for {'max_depth': 5, 'random_state': 1}
0.903 for {'max_depth': 5, 'random_state': 4}
0.902 for {'max_depth': 5, 'random_state': 6}
0.901 for {'max_depth': 5, 'random_state': 7}
0.901 for {'max_depth': 5, 'random_state': 9}
0.920 for {'max_depth': 6, 'random_state': 1}
0.922 for {'max_depth': 6, 'random_state': 4}
0.919 for {'max_depth': 6, 'random_state': 6}
0.922 for {'max_depth': 6, 'random_state': 7}
0.925 for {'max_depth': 6, 'random_state': 9}
0.932 for {'max_depth': 7, 'random_state': 1}
0.933 for {'max_depth': 7, 'random_state': 4}
0.933 for {'max_depth': 7, 'random_state': 6}
0.929 for {'max_depth': 7, 'random_state': 7}
0.934 for {'max_depth': 7, 'random_state': 9}
```

Best result in cross-validation: 0.9343396226415095.

Pair 2: M and Y

```
0.851 for {'max_depth': 3, 'random_state': 1}
0.851 for {'max_depth': 3, 'random_state': 4}
0.851 for {'max_depth': 3, 'random_state': 6}
0.851 for {'max_depth': 3, 'random_state': 7}
0.851 for {'max_depth': 3, 'random_state': 9}
0.861 for {'max_depth': 4, 'random_state': 1}
0.861 for {'max_depth': 4, 'random_state': 4}
0.861 for {'max_depth': 4, 'random_state': 6}
0.861 for {'max_depth': 4, 'random_state': 7}
0.861 for {'max_depth': 4, 'random_state': 9}
0.857 for {'max_depth': 5, 'random_state': 1}
0.855 for {'max_depth': 5, 'random_state': 4}
0.856 for {'max_depth': 5, 'random_state': 6}
0.855 for {'max_depth': 5, 'random_state': 7}
0.855 for {'max_depth': 5, 'random_state': 9}
0.866 for {'max_depth': 6, 'random_state': 1}
0.866 for {'max_depth': 6, 'random_state': 4}
0.866 for {'max_depth': 6, 'random_state': 6}
0.865 for {'max_depth': 6, 'random_state': 7}
0.866 for {'max_depth': 6, 'random_state': 9}
0.878 for {'max_depth': 7, 'random_state': 1}
0.878 for {'max_depth': 7, 'random_state': 4}
0.878 for {'max_depth': 7, 'random_state': 6}
0.878 for {'max_depth': 7, 'random_state': 7}
0.878 for {'max_depth': 7, 'random_state': 9}
```

Best result in cross-validation: 0.8784905660377358.

Pair: A and B

```
1.000 for {'max_depth': 3, 'random_state': 1}
1.000 for {'max_depth': 3, 'random_state': 4}
1.000 for {'max_depth': 3, 'random_state': 6}
1.000 for {'max_depth': 3, 'random_state': 7}
1.000 for {'max_depth': 3, 'random_state': 9}
1.000 for {'max_depth': 4, 'random_state': 1}
1.000 for {'max_depth': 4, 'random_state': 4}
1.000 for {'max_depth': 4, 'random_state': 6}
1.000 for {'max_depth': 4, 'random_state': 7}
1.000 for {'max_depth': 4, 'random_state': 9}
1.000 for {'max_depth': 5, 'random_state': 1}
1.000 for {'max_depth': 5, 'random_state': 4}
1.000 for {'max_depth': 5, 'random_state': 6}
1.000 for {'max_depth': 5, 'random_state': 7}
1.000 for {'max_depth': 5, 'random_state': 9}
1.000 for {'max_depth': 6, 'random_state': 1}
1.000 for {'max_depth': 6, 'random_state': 4}
1.000 for {'max_depth': 6, 'random_state': 6}
1.000 for {'max_depth': 6, 'random_state': 7}
1.000 for {'max_depth': 6, 'random_state': 9}
1.000 for {'max_depth': 7, 'random_state': 1}
1.000 for {'max_depth': 7, 'random_state': 4}
1.000 for {'max_depth': 7, 'random_state': 6}
1.000 for {'max_depth': 7, 'random_state': 7}
1.000 for {'max_depth': 7, 'random_state': 9}
```

Best result in cross-validation: 1.0.

3) The dimension reduction method is Univariate Feature Selection of Feature Selection. It can get the top k features with the highest score.

4) Pair 1: H and K

```
0.864 for {'max_depth': 3, 'random_state': 1}
0.864 for {'max_depth': 3, 'random_state': 4}
0.864 for {'max_depth': 3, 'random_state': 6}
0.864 for {'max_depth': 3, 'random_state': 7}
0.864 for {'max_depth': 3, 'random_state': 9}
0.895 for {'max_depth': 4, 'random_state': 1}
0.895 for {'max_depth': 4, 'random_state': 4}
0.895 for {'max_depth': 4, 'random_state': 6}
0.895 for {'max_depth': 4, 'random_state': 7}
0.895 for {'max_depth': 4, 'random_state': 9}
0.896 for {'max_depth': 5, 'random_state': 1}
0.896 for {'max_depth': 5, 'random_state': 4}
0.896 for {'max_depth': 5, 'random_state': 6}
0.896 for {'max_depth': 5, 'random_state': 7}
0.896 for {'max_depth': 5, 'random_state': 9}
0.885 for {'max_depth': 6, 'random_state': 1}
0.885 for {'max_depth': 6, 'random_state': 4}
0.885 for {'max_depth': 6, 'random_state': 6}
0.884 for {'max_depth': 6, 'random_state': 7}
0.885 for {'max_depth': 6, 'random_state': 9}
0.888 for {'max_depth': 7, 'random_state': 1}
0.888 for {'max_depth': 7, 'random_state': 4}
0.888 for {'max_depth': 7, 'random_state': 6}
0.888 for {'max_depth': 7, 'random_state': 7}
0.888 for {'max_depth': 7, 'random_state': 9}
```

Best result in cross-validation: 0.8958490566037736.

Pair 2: M and Y

```
0.976 for {'max_depth': 3, 'random_state': 1}
0.976 for {'max_depth': 3, 'random_state': 4}
0.976 for {'max_depth': 3, 'random_state': 6}
0.976 for {'max_depth': 3, 'random_state': 7}
0.976 for {'max_depth': 3, 'random_state': 9}
0.983 for {'max_depth': 4, 'random_state': 1}
0.984 for {'max_depth': 4, 'random_state': 4}
0.982 for {'max_depth': 4, 'random_state': 6}
0.983 for {'max_depth': 4, 'random_state': 7}
0.982 for {'max_depth': 4, 'random_state': 9}
0.989 for {'max_depth': 5, 'random_state': 1}
0.989 for {'max_depth': 5, 'random_state': 4}
0.988 for {'max_depth': 5, 'random_state': 6}
0.989 for {'max_depth': 5, 'random_state': 7}
0.988 for {'max_depth': 5, 'random_state': 9}
0.987 for {'max_depth': 6, 'random_state': 1}
0.990 for {'max_depth': 6, 'random_state': 4}
0.989 for {'max_depth': 6, 'random_state': 6}
0.989 for {'max_depth': 6, 'random_state': 7}
0.989 for {'max_depth': 6, 'random_state': 9}
0.985 for {'max_depth': 7, 'random_state': 1}
0.988 for {'max_depth': 7, 'random_state': 4}
0.986 for {'max_depth': 7, 'random_state': 6}
0.985 for {'max_depth': 7, 'random_state': 7}
0.984 for {'max_depth': 7, 'random_state': 9}
```

Best result in cross-validation: 0.9901408450704224.

Pair 3: A and B

```
0.953 for {'max_depth': 3, 'random_state': 1}
0.953 for {'max_depth': 3, 'random_state': 4}
0.953 for {'max_depth': 3, 'random_state': 6}
0.953 for {'max_depth': 3, 'random_state': 7}
0.953 for {'max_depth': 3, 'random_state': 9}
0.966 for {'max_depth': 4, 'random_state': 1}
0.966 for {'max_depth': 4, 'random_state': 4}
0.966 for {'max_depth': 4, 'random_state': 6}
0.966 for {'max_depth': 4, 'random_state': 7}
0.966 for {'max_depth': 4, 'random_state': 9}
0.969 for {'max_depth': 5, 'random_state': 1}
0.969 for {'max_depth': 5, 'random_state': 4}
0.969 for {'max_depth': 5, 'random_state': 6}
0.969 for {'max_depth': 5, 'random_state': 7}
0.969 for {'max_depth': 5, 'random_state': 9}
0.971 for {'max_depth': 6, 'random_state': 1}
0.973 for {'max_depth': 6, 'random_state': 4}
0.972 for {'max_depth': 6, 'random_state': 6}
0.972 for {'max_depth': 6, 'random_state': 7}
0.972 for {'max_depth': 6, 'random_state': 9}
0.972 for {'max_depth': 7, 'random_state': 1}
0.972 for {'max_depth': 7, 'random_state': 4}
0.971 for {'max_depth': 7, 'random_state': 6}
0.972 for {'max_depth': 7, 'random_state': 7}
0.973 for {'max_depth': 7, 'random_state': 9}
```

Best result in cross-validation: 0.9728315412186381.

5) Hyperparameters are ‘max_depth’ and ‘random_state’.

3. Random Forest
   1) Random Forest is an algorithm that integrates multiple trees through the idea of ensemble learning. Its basic unit is a decision tree.
      Advantage: good performance, strong anti-interference ability.
      Disadvantage: many similar decision trees masking the real results, easy to overfit.
   2) Pair 1: H and K

```
0.888 for {'max_depth': 3, 'n_estimators': 100}
0.897 for {'max_depth': 3, 'n_estimators': 110}
0.894 for {'max_depth': 3, 'n_estimators': 120}
0.884 for {'max_depth': 3, 'n_estimators': 130}
0.882 for {'max_depth': 3, 'n_estimators': 140}
0.911 for {'max_depth': 4, 'n_estimators': 100}
0.911 for {'max_depth': 4, 'n_estimators': 110}
0.915 for {'max_depth': 4, 'n_estimators': 120}
0.917 for {'max_depth': 4, 'n_estimators': 130}
0.912 for {'max_depth': 4, 'n_estimators': 140}
0.925 for {'max_depth': 5, 'n_estimators': 100}
0.923 for {'max_depth': 5, 'n_estimators': 110}
0.924 for {'max_depth': 5, 'n_estimators': 120}
0.921 for {'max_depth': 5, 'n_estimators': 130}
0.924 for {'max_depth': 5, 'n_estimators': 140}
0.937 for {'max_depth': 6, 'n_estimators': 100}
0.937 for {'max_depth': 6, 'n_estimators': 110}
0.938 for {'max_depth': 6, 'n_estimators': 120}
0.940 for {'max_depth': 6, 'n_estimators': 130}
0.938 for {'max_depth': 6, 'n_estimators': 140}
0.946 for {'max_depth': 7, 'n_estimators': 100}
0.945 for {'max_depth': 7, 'n_estimators': 110}
0.948 for {'max_depth': 7, 'n_estimators': 120}
0.949 for {'max_depth': 7, 'n_estimators': 130}
0.946 for {'max_depth': 7, 'n_estimators': 140}
```

Best result in cross-validation: 0.9486792452830188.

Pair 2: M and Y

```
0.990 for {'max_depth': 3, 'n_estimators': 100}
0.989 for {'max_depth': 3, 'n_estimators': 110}
0.985 for {'max_depth': 3, 'n_estimators': 120}
0.989 for {'max_depth': 3, 'n_estimators': 130}
0.987 for {'max_depth': 3, 'n_estimators': 140}
0.992 for {'max_depth': 4, 'n_estimators': 100}
0.991 for {'max_depth': 4, 'n_estimators': 110}
0.992 for {'max_depth': 4, 'n_estimators': 120}
0.992 for {'max_depth': 4, 'n_estimators': 130}
0.991 for {'max_depth': 4, 'n_estimators': 140}
0.992 for {'max_depth': 5, 'n_estimators': 100}
0.992 for {'max_depth': 5, 'n_estimators': 110}
0.993 for {'max_depth': 5, 'n_estimators': 120}
0.994 for {'max_depth': 5, 'n_estimators': 130}
0.995 for {'max_depth': 5, 'n_estimators': 140}
0.995 for {'max_depth': 6, 'n_estimators': 100}
0.994 for {'max_depth': 6, 'n_estimators': 110}
0.994 for {'max_depth': 6, 'n_estimators': 120}
0.996 for {'max_depth': 6, 'n_estimators': 130}
0.994 for {'max_depth': 6, 'n_estimators': 140}
0.995 for {'max_depth': 7, 'n_estimators': 100}
0.995 for {'max_depth': 7, 'n_estimators': 110}
0.996 for {'max_depth': 7, 'n_estimators': 120}
0.995 for {'max_depth': 7, 'n_estimators': 130}
0.996 for {'max_depth': 7, 'n_estimators': 140}
```

Best result in cross-validation: 0.995774647887324.

Pair 3: A and B

```
0.986 for {'max_depth': 3, 'n_estimators': 100}
0.979 for {'max_depth': 3, 'n_estimators': 110}
0.983 for {'max_depth': 3, 'n_estimators': 120}
0.979 for {'max_depth': 3, 'n_estimators': 130}
0.981 for {'max_depth': 3, 'n_estimators': 140}
0.996 for {'max_depth': 4, 'n_estimators': 100}
0.996 for {'max_depth': 4, 'n_estimators': 110}
0.996 for {'max_depth': 4, 'n_estimators': 120}
0.996 for {'max_depth': 4, 'n_estimators': 130}
0.994 for {'max_depth': 4, 'n_estimators': 140}
0.999 for {'max_depth': 5, 'n_estimators': 100}
0.999 for {'max_depth': 5, 'n_estimators': 110}
0.999 for {'max_depth': 5, 'n_estimators': 120}
0.998 for {'max_depth': 5, 'n_estimators': 130}
0.999 for {'max_depth': 5, 'n_estimators': 140}
0.999 for {'max_depth': 6, 'n_estimators': 100}
0.999 for {'max_depth': 6, 'n_estimators': 110}
1.000 for {'max_depth': 6, 'n_estimators': 120}
0.999 for {'max_depth': 6, 'n_estimators': 130}
1.000 for {'max_depth': 6, 'n_estimators': 140}
0.998 for {'max_depth': 7, 'n_estimators': 100}
1.000 for {'max_depth': 7, 'n_estimators': 110}
1.000 for {'max_depth': 7, 'n_estimators': 120}
1.000 for {'max_depth': 7, 'n_estimators': 130}
1.000 for {'max_depth': 7, 'n_estimators': 140}
```

Best result in cross-validation: 1.0.

3) The dimension reduction method is Univariate Feature Selection of Feature Selection. It can get the top k features with the highest score.

4) Pair 1: H and K

```
0.966 for {'max_depth': 3, 'n_estimators': 100}
0.968 for {'max_depth': 3, 'n_estimators': 110}
0.964 for {'max_depth': 3, 'n_estimators': 120}
0.968 for {'max_depth': 3, 'n_estimators': 130}
0.966 for {'max_depth': 3, 'n_estimators': 140}
0.976 for {'max_depth': 4, 'n_estimators': 100}
0.975 for {'max_depth': 4, 'n_estimators': 110}
0.975 for {'max_depth': 4, 'n_estimators': 120}
0.976 for {'max_depth': 4, 'n_estimators': 130}
0.976 for {'max_depth': 4, 'n_estimators': 140}
0.973 for {'max_depth': 5, 'n_estimators': 100}
0.975 for {'max_depth': 5, 'n_estimators': 110}
0.976 for {'max_depth': 5, 'n_estimators': 120}
0.975 for {'max_depth': 5, 'n_estimators': 130}
0.975 for {'max_depth': 5, 'n_estimators': 140}
0.977 for {'max_depth': 6, 'n_estimators': 100}
0.976 for {'max_depth': 6, 'n_estimators': 110}
0.976 for {'max_depth': 6, 'n_estimators': 120}
0.977 for {'max_depth': 6, 'n_estimators': 130}
0.976 for {'max_depth': 6, 'n_estimators': 140}
0.978 for {'max_depth': 7, 'n_estimators': 100}
0.979 for {'max_depth': 7, 'n_estimators': 110}
0.979 for {'max_depth': 7, 'n_estimators': 120}
0.978 for {'max_depth': 7, 'n_estimators': 130}
0.977 for {'max_depth': 7, 'n_estimators': 140}
```

Best result in cross-validation: 0.979267793138761.

Pair 2: M and Y

```
0.984 for {'max_depth': 3, 'n_estimators': 100}
0.984 for {'max_depth': 3, 'n_estimators': 110}
0.986 for {'max_depth': 3, 'n_estimators': 120}
0.985 for {'max_depth': 3, 'n_estimators': 130}
0.986 for {'max_depth': 3, 'n_estimators': 140}
0.986 for {'max_depth': 4, 'n_estimators': 100}
0.986 for {'max_depth': 4, 'n_estimators': 110}
0.986 for {'max_depth': 4, 'n_estimators': 120}
0.986 for {'max_depth': 4, 'n_estimators': 130}
0.987 for {'max_depth': 4, 'n_estimators': 140}
0.988 for {'max_depth': 5, 'n_estimators': 100}
0.988 for {'max_depth': 5, 'n_estimators': 110}
0.989 for {'max_depth': 5, 'n_estimators': 120}
0.987 for {'max_depth': 5, 'n_estimators': 130}
0.987 for {'max_depth': 5, 'n_estimators': 140}
0.989 for {'max_depth': 6, 'n_estimators': 100}
0.989 for {'max_depth': 6, 'n_estimators': 110}
0.989 for {'max_depth': 6, 'n_estimators': 120}
0.992 for {'max_depth': 6, 'n_estimators': 130}
0.990 for {'max_depth': 6, 'n_estimators': 140}
0.990 for {'max_depth': 7, 'n_estimators': 100}
0.989 for {'max_depth': 7, 'n_estimators': 110}
0.989 for {'max_depth': 7, 'n_estimators': 120}
0.989 for {'max_depth': 7, 'n_estimators': 130}
0.988 for {'max_depth': 7, 'n_estimators': 140}
```

Best result in cross-validation: 0.991549295774648.

Pair 3: A and B

```
0.967 for {'max_depth': 3, 'n_estimators': 100}
0.964 for {'max_depth': 3, 'n_estimators': 110}
0.967 for {'max_depth': 3, 'n_estimators': 120}
0.969 for {'max_depth': 3, 'n_estimators': 130}
0.969 for {'max_depth': 3, 'n_estimators': 140}
0.974 for {'max_depth': 4, 'n_estimators': 100}
0.976 for {'max_depth': 4, 'n_estimators': 110}
0.974 for {'max_depth': 4, 'n_estimators': 120}
0.973 for {'max_depth': 4, 'n_estimators': 130}
0.974 for {'max_depth': 4, 'n_estimators': 140}
0.976 for {'max_depth': 5, 'n_estimators': 100}
0.976 for {'max_depth': 5, 'n_estimators': 110}
0.975 for {'max_depth': 5, 'n_estimators': 120}
0.974 for {'max_depth': 5, 'n_estimators': 130}
0.976 for {'max_depth': 5, 'n_estimators': 140}
0.976 for {'max_depth': 6, 'n_estimators': 100}
0.976 for {'max_depth': 6, 'n_estimators': 110}
0.977 for {'max_depth': 6, 'n_estimators': 120}
0.976 for {'max_depth': 6, 'n_estimators': 130}
0.977 for {'max_depth': 6, 'n_estimators': 140}
0.978 for {'max_depth': 7, 'n_estimators': 100}
0.979 for {'max_depth': 7, 'n_estimators': 110}
0.978 for {'max_depth': 7, 'n_estimators': 120}
0.976 for {'max_depth': 7, 'n_estimators': 130}
0.978 for {'max_depth': 7, 'n_estimators': 140}
```

Best result in cross-validation: 0.9785535074244752.

5) Hyperparameters are ‘max_depth’ and ‘n_estimators’.

4. SVM
   1) SVM is a two-class classifier, and its goal is to find a hyperplane, using two classes of data as far away from the hyperplane as possible, so that it can classify new data more accurately, even if the classifier is more robust.
      Advantage: performs well on various datasets.
      Disadvantage: not suitable for large sample data.
   2) Pair 1: H and K

```
0.835 for {'C': 7, 'gamma': 2}
0.730 for {'C': 7, 'gamma': 4}
0.680 for {'C': 7, 'gamma': 6}
0.649 for {'C': 7, 'gamma': 8}
0.631 for {'C': 7, 'gamma': 10}
0.835 for {'C': 9, 'gamma': 2}
0.730 for {'C': 9, 'gamma': 4}
0.680 for {'C': 9, 'gamma': 6}
0.649 for {'C': 9, 'gamma': 8}
0.631 for {'C': 9, 'gamma': 10}
```

```
0.828 for {'C': 1, 'gamma': 2}
0.719 for {'C': 1, 'gamma': 4}
0.663 for {'C': 1, 'gamma': 6}
0.636 for {'C': 1, 'gamma': 8}
0.614 for {'C': 1, 'gamma': 10}
0.835 for {'C': 3, 'gamma': 2}
0.730 for {'C': 3, 'gamma': 4}
0.680 for {'C': 3, 'gamma': 6}
0.649 for {'C': 3, 'gamma': 8}
0.631 for {'C': 3, 'gamma': 10}
0.835 for {'C': 5, 'gamma': 2}
0.730 for {'C': 5, 'gamma': 4}
0.680 for {'C': 5, 'gamma': 6}
0.649 for {'C': 5, 'gamma': 8}
0.631 for {'C': 5, 'gamma': 10}
```

Best result in cross-validation: 0.8347169811320754.

Pair 2: M and Y

```
0.977 for {'C': 7, 'gamma': 2}
0.850 for {'C': 7, 'gamma': 4}
0.735 for {'C': 7, 'gamma': 6}
0.674 for {'C': 7, 'gamma': 8}
0.644 for {'C': 7, 'gamma': 10}
0.977 for {'C': 9, 'gamma': 2}
0.850 for {'C': 9, 'gamma': 4}
0.735 for {'C': 9, 'gamma': 6}
0.674 for {'C': 9, 'gamma': 8}
0.644 for {'C': 9, 'gamma': 10}
```

```
0.976 for {'C': 1, 'gamma': 2}
0.846 for {'C': 1, 'gamma': 4}
0.713 for {'C': 1, 'gamma': 6}
0.657 for {'C': 1, 'gamma': 8}
0.636 for {'C': 1, 'gamma': 10}
0.977 for {'C': 3, 'gamma': 2}
0.850 for {'C': 3, 'gamma': 4}
0.735 for {'C': 3, 'gamma': 6}
0.674 for {'C': 3, 'gamma': 8}
0.644 for {'C': 3, 'gamma': 10}
0.977 for {'C': 5, 'gamma': 2}
0.850 for {'C': 5, 'gamma': 4}
0.735 for {'C': 5, 'gamma': 6}
0.674 for {'C': 5, 'gamma': 8}
0.644 for {'C': 5, 'gamma': 10}
```

Best result in cross-validation: 0.9774647887323944.

Pair 3: A and B

```
0.948 for {'C': 1, 'gamma': 2}
0.764 for {'C': 1, 'gamma': 4}
0.653 for {'C': 1, 'gamma': 6}
0.600 for {'C': 1, 'gamma': 8}
0.573 for {'C': 1, 'gamma': 10}
0.949 for {'C': 3, 'gamma': 2}
0.779 for {'C': 3, 'gamma': 4}
0.666 for {'C': 3, 'gamma': 6}
0.613 for {'C': 3, 'gamma': 8}
0.589 for {'C': 3, 'gamma': 10}
0.949 for {'C': 5, 'gamma': 2}
0.779 for {'C': 5, 'gamma': 4}
0.666 for {'C': 5, 'gamma': 6}
0.613 for {'C': 5, 'gamma': 8}
0.589 for {'C': 5, 'gamma': 10}
0.949 for {'C': 7, 'gamma': 2}
0.779 for {'C': 7, 'gamma': 4}
0.666 for {'C': 7, 'gamma': 6}
0.613 for {'C': 7, 'gamma': 8}
0.589 for {'C': 7, 'gamma': 10}
0.949 for {'C': 9, 'gamma': 2}
0.779 for {'C': 9, 'gamma': 4}
0.666 for {'C': 9, 'gamma': 6}
0.613 for {'C': 9, 'gamma': 8}
0.589 for {'C': 9, 'gamma': 10}
```

Best result in cross-validation: 0.9485330261136713.

3) The method is PCA. In PCA, data is transformed from the original coordinate system to the new coordinate system. The choice of a new coordinate system is determined by the data itself. The first new axis selects the direction with the largest variance in the original data, and the second new axis selects the direction that is orthogonal to the first axis and has the greatest variance. This process is repeated for the number of features in the original data. You will find that most of the variance is contained in the first few new axes. Therefore, we can only select the first few coordinate axes, that is, the data is dimensionally reduced. Advantage: reduce the computational cost of the algorithm, avoid noisy. Disadvantage: feature decomposition has limitations.
4) Pair 1: H and K

```
0.908 for {'C': 1, 'gamma': 2}
0.900 for {'C': 1, 'gamma': 4}
0.897 for {'C': 1, 'gamma': 6}
0.893 for {'C': 1, 'gamma': 8}
0.886 for {'C': 1, 'gamma': 10}
0.911 for {'C': 3, 'gamma': 2}
0.900 for {'C': 3, 'gamma': 4}
0.887 for {'C': 3, 'gamma': 6}
0.882 for {'C': 3, 'gamma': 8}
0.875 for {'C': 3, 'gamma': 10}
0.909 for {'C': 5, 'gamma': 2}
0.897 for {'C': 5, 'gamma': 4}
0.885 for {'C': 5, 'gamma': 6}
0.884 for {'C': 5, 'gamma': 8}
0.877 for {'C': 5, 'gamma': 10}
0.907 for {'C': 7, 'gamma': 2}
0.894 for {'C': 7, 'gamma': 4}
0.888 for {'C': 7, 'gamma': 6}
0.883 for {'C': 7, 'gamma': 8}
0.877 for {'C': 7, 'gamma': 10}
0.905 for {'C': 9, 'gamma': 2}
0.894 for {'C': 9, 'gamma': 4}
0.886 for {'C': 9, 'gamma': 6}
0.881 for {'C': 9, 'gamma': 8}
0.876 for {'C': 9, 'gamma': 10}
```

Best result in cross-validation: 0.9109433962264152.

Pair 2: M and Y

```
0.981 for {'C': 1, 'gamma': 2}
0.982 for {'C': 1, 'gamma': 4}
0.980 for {'C': 1, 'gamma': 6}
0.980 for {'C': 1, 'gamma': 8}
0.979 for {'C': 1, 'gamma': 10}
0.982 for {'C': 3, 'gamma': 2}
0.980 for {'C': 3, 'gamma': 4}
0.982 for {'C': 3, 'gamma': 6}
0.982 for {'C': 3, 'gamma': 8}
0.983 for {'C': 3, 'gamma': 10}
0.981 for {'C': 5, 'gamma': 2}
0.982 for {'C': 5, 'gamma': 4}
0.981 for {'C': 5, 'gamma': 6}
0.981 for {'C': 5, 'gamma': 8}
0.982 for {'C': 5, 'gamma': 10}
0.982 for {'C': 7, 'gamma': 2}
0.981 for {'C': 7, 'gamma': 4}
0.980 for {'C': 7, 'gamma': 6}
0.981 for {'C': 7, 'gamma': 8}
0.982 for {'C': 7, 'gamma': 10}
0.982 for {'C': 9, 'gamma': 2}
0.980 for {'C': 9, 'gamma': 4}
0.980 for {'C': 9, 'gamma': 6}
0.981 for {'C': 9, 'gamma': 8}
0.982 for {'C': 9, 'gamma': 10}
```

Best result in cross-validation: 0.9830985915492958.

Pair 3: A and B

```
0.983 for {'C': 1, 'gamma': 2}
0.983 for {'C': 1, 'gamma': 4}
0.981 for {'C': 1, 'gamma': 6}
0.975 for {'C': 1, 'gamma': 8}
0.967 for {'C': 1, 'gamma': 10}
0.984 for {'C': 3, 'gamma': 2}
0.981 for {'C': 3, 'gamma': 4}
0.980 for {'C': 3, 'gamma': 6}
0.973 for {'C': 3, 'gamma': 8}
0.965 for {'C': 3, 'gamma': 10}
0.984 for {'C': 5, 'gamma': 2}
0.980 for {'C': 5, 'gamma': 4}
0.979 for {'C': 5, 'gamma': 6}
0.971 for {'C': 5, 'gamma': 8}
0.964 for {'C': 5, 'gamma': 10}
0.984 for {'C': 7, 'gamma': 2}
0.979 for {'C': 7, 'gamma': 4}
0.977 for {'C': 7, 'gamma': 6}
0.971 for {'C': 7, 'gamma': 8}
0.963 for {'C': 7, 'gamma': 10}
0.985 for {'C': 9, 'gamma': 2}
0.979 for {'C': 9, 'gamma': 4}
0.977 for {'C': 9, 'gamma': 6}
0.971 for {'C': 9, 'gamma': 8}
0.964 for {'C': 9, 'gamma': 10}
```

Best result in cross-validation: 0.9849897593445981.

5) Hyperparameters are ‘C’ and ‘gamma’.

5. Artificial Neural Network

1) A single neuron collects the information transmitted by other neurons on the dendrite. After the amount of information reaches a threshold, a pulse signal is generated, which is transmitted through the axon to the synapse, and then transmitted to the next neuron.

   Advantage: ability to find optimal solutions at high speed, high classification accuracy.

   Disadvantage: require a large number of parameters, it may fall into a local

minimum.

2) Pair 1: H and K

```
0.976 for {'alpha': 0.01, 'learning_rate': 'constant'}
0.977 for {'alpha': 0.01, 'learning_rate': 'invscaling'}
0.974 for {'alpha': 0.01, 'learning_rate': 'adaptive'}
0.977 for {'alpha': 0.001, 'learning_rate': 'constant'}
0.975 for {'alpha': 0.001, 'learning_rate': 'invscaling'}
0.977 for {'alpha': 0.001, 'learning_rate': 'adaptive'}
0.977 for {'alpha': 0.0001, 'learning_rate': 'constant'}
0.976 for {'alpha': 0.0001, 'learning_rate': 'invscaling'}
0.976 for {'alpha': 0.0001, 'learning_rate': 'adaptive'}
```

Best result in cross-validation: 0.9773584905660379.

Pair 2: M and Y

```
0.999 for {'alpha': 0.01, 'learning_rate': 'constant'}
0.999 for {'alpha': 0.01, 'learning_rate': 'invscaling'}
0.999 for {'alpha': 0.01, 'learning_rate': 'adaptive'}
0.999 for {'alpha': 0.001, 'learning_rate': 'constant'}
1.000 for {'alpha': 0.001, 'learning_rate': 'invscaling'}
0.999 for {'alpha': 0.001, 'learning_rate': 'adaptive'}
1.000 for {'alpha': 0.0001, 'learning_rate': 'constant'}
0.999 for {'alpha': 0.0001, 'learning_rate': 'invscaling'}
0.999 for {'alpha': 0.0001, 'learning_rate': 'adaptive'}
```

Best result in cross-validation: 1.0.

Pair 3: A and B

```
0.997 for {'alpha': 0.01, 'learning_rate': 'constant'}
0.997 for {'alpha': 0.01, 'learning_rate': 'invscaling'}
0.998 for {'alpha': 0.01, 'learning_rate': 'adaptive'}
0.997 for {'alpha': 0.001, 'learning_rate': 'constant'}
0.996 for {'alpha': 0.001, 'learning_rate': 'invscaling'}
0.997 for {'alpha': 0.001, 'learning_rate': 'adaptive'}
0.998 for {'alpha': 0.0001, 'learning_rate': 'constant'}
0.997 for {'alpha': 0.0001, 'learning_rate': 'invscaling'}
0.998 for {'alpha': 0.0001, 'learning_rate': 'adaptive'}
```

Best result in cross-validation: 0.9978571428571428.

3) The method is PCA. The method is PCA. In PCA, data is transformed from the original coordinate system to the new coordinate system. The choice of a new coordinate system is determined by the data itself. The first new axis selects the direction with the largest variance in the original data, and the second new axis selects the direction that is orthogonal to the first axis and has the greatest variance. This process is repeated for the number of features in the

original data. You will find that most of the variance is contained in the first few new axes. Therefore, we can only select the first few coordinate axes, that is, the data is dimensionally reduced. Advantage: reduce the computational cost of the algorithm, avoid noisy.

4) Pair 1: H and K

```
0.911 for {'alpha': 0.01, 'learning_rate': 'constant'}
0.902 for {'alpha': 0.01, 'learning_rate': 'invscaling'}
0.906 for {'alpha': 0.01, 'learning_rate': 'adaptive'}
0.912 for {'alpha': 0.001, 'learning_rate': 'constant'}
0.907 for {'alpha': 0.001, 'learning_rate': 'invscaling'}
0.912 for {'alpha': 0.001, 'learning_rate': 'adaptive'}
0.908 for {'alpha': 0.0001, 'learning_rate': 'constant'}
0.915 for {'alpha': 0.0001, 'learning_rate': 'invscaling'}
0.911 for {'alpha': 0.0001, 'learning_rate': 'adaptive'}
```

Best result in cross-validation: 0.9154716981132076.

Pair 2: M and Y

```
0.982 for {'alpha': 0.01, 'learning_rate': 'constant'}
0.981 for {'alpha': 0.01, 'learning_rate': 'invscaling'}
0.983 for {'alpha': 0.01, 'learning_rate': 'adaptive'}
0.982 for {'alpha': 0.001, 'learning_rate': 'constant'}
0.980 for {'alpha': 0.001, 'learning_rate': 'invscaling'}
0.982 for {'alpha': 0.001, 'learning_rate': 'adaptive'}
0.982 for {'alpha': 0.0001, 'learning_rate': 'constant'}
0.980 for {'alpha': 0.0001, 'learning_rate': 'invscaling'}
0.981 for {'alpha': 0.0001, 'learning_rate': 'adaptive'}
```

Best result in cross-validation: 0.9830985915492958.

Pair 3: A and B

```
0.984 for {'alpha': 0.01, 'learning_rate': 'constant'}
0.984 for {'alpha': 0.01, 'learning_rate': 'invscaling'}
0.984 for {'alpha': 0.01, 'learning_rate': 'adaptive'}
0.984 for {'alpha': 0.001, 'learning_rate': 'constant'}
0.981 for {'alpha': 0.001, 'learning_rate': 'invscaling'}
0.984 for {'alpha': 0.001, 'learning_rate': 'adaptive'}
0.984 for {'alpha': 0.0001, 'learning_rate': 'constant'}
0.985 for {'alpha': 0.0001, 'learning_rate': 'invscaling'}
0.984 for {'alpha': 0.0001, 'learning_rate': 'adaptive'}
```

Best result in cross-validation: 0.9849948796722992.

5) Hyperparameters are ‘alpha’ and ‘learning-rate’.

Discussion

1. Before dimension reduction

		KNN	Decision Tree	Random Forest	SVM	Artificial Neural Network
H and K	Performance(Best result in cross-validation)	0.947924528301886 8	0.9343396226415095	0.9486792452830188	0.8347169811320754	0.9773584905660379
	Run time(fit time)	0.00030794	0.00283959	0.10282288	0.03439016	1.76284561
M and Y	performance	0.998591549295774 6	0.8784905660377358	0.995774647887324	0.9774647887323944	1.0
	Run time	0.00040011	0.00159845	0.10129547	0.03834763	0.83972168
A and B	performance	1.0	1.0	1.0	0.9485330261136713	0.9978571428571428
	Run time	0.00059272	0.00139999	0.1081883	0.04058228	0.882418

2. After dimension reduction

		KNN	Decision Tree	Random Forest	SVM	Artificial Neural Network
H and K	performance	0.2920754716981132	0.8958490566037736	0.979267793138761	0.9109433962264152	0.9154716981132076
	Run time	0.00090333	0.00082231	0.09867988	0.0255929	3.00140815
M and Y	performance	0.9626760563380282	0.9901408450704224	0.991549295774648	0.9830985915492958	0.9830985915492958
	Run time	0.00078974	0.00080872	0.09299951	0.02519207	6.15156178
A and B	performance	0.5375166410650282	0.9728315412186381	0.9785535074244752	0.9849897593445981	0.9849948796722992
	Run time	0.00060096	0.0007987	0.10241613	0.02301206	3.6501545

3. I would choose Decision Tree. Because it accuracy is high and time costs is low. Dimension reduction will reduce the run time but decrease the accuracy for most of the models. For the new dataset, I will see if there some missing values, and then use the suit model according to the dataset's size. And I find in this problem, some models are all work well, like Decision Tree, Random Forest and SVM.