Challenge SD 210
WEI CHEN
weichen@whu.edu.cn

1. Pre step

1.1 Data Standardization

Standardization of a dataset is a common requirement for many machine learning estimators: they might behave badly if the individual feature do not more or less look like standard normally distributed data. For instance the RBF kernel of Support Vector Machines assume that all features are centred on 0 and have variance in the same order. If a feature has a variance that is orders of magnitude larger than others, it might dominate the objective function and make the estimator unable to learn from other features correctly as expected.

Use *sklearn.preprocessing.**StandardScaler*** to standardize the training and testing data. This function Standardizes features by removing the mean and scaling to unit variance. And we apply SVM classifier to test the cross validation score respectively with and without data standardization.

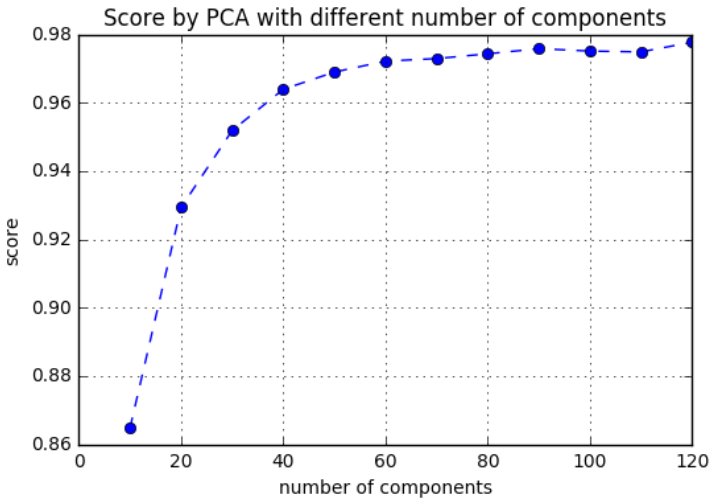
Since training process of SVC is quite time-costing if the volume of data gets very large, for simplicity, we use part of the whole data set, and use **4-fold** cross validation.

	without Standardization	With Standardization
cross validation score	0.9132	0.9783

1.2 PCA decompostion

Notice that the data sets we are dealing with have 128 features. It may exist high dependency between those features. So we try to apply PCA to reduce the dependency of data features.

With *sklearn.decomposition.**PCA*** and the SVM classifier, compare the cross validation score with different components of PCA.



1.3 Data reduction via KNN

For the current training set, we will see from the testing results that it is not very compatible with the test set. This may be due to the fact that the training and test sets are drawn from two different populations, or the training set contains more noise samples which do no good to the model.

To this end, we have to cut off the noisy samples or 'prune' the training set making it more coherent with the test set that we try to predict.

We apply *sklearn.neighbors.**NearestNeighbors*** to address this problem. For each sample in the test set, we look for its K nearest samples in training set, and add them the new defined set X_select. Certainly, we will eliminate the repeating samples in X_select in the end. By this method we will get the optimised training set X_select specially designed for test set.

$$X\_select = \{kneighbors(X\_test, X\_train)\}$$

In the experiment, we set **K=10**, and only consider the **first 64 features** of X_test, because both X_train and X_test have been transformed with PCA, and their first few features carry most importance.

Finally we got **38943** selected samples from entire X_train, which signify that it exists many irrelevant samples in X_train.

1.4 Determine the predicted labels

The score for predicting results is defined by

$$score = \frac{1}{N}\sum_{i=1}^{N}\Big(1(\hat{y}_i = 0) + 10 \times 1(y_i\hat{y}_i = -1)\Big)$$

Since the false prediction will be heavily punished, we cannot give an uncertain prediction for the samples which may cause error. We have to create a rule to label 0 for these uncertain samples to avoid the error prediction.

In the following classifiers, the ***predict_proba()*** function will be used to estimate the probability of a certain sample belonging to a class. And we define the label of X_i as

$$\hat{y}_i = \begin{cases} \text{predict}(X_i), & \text{if predict\_proba}(X_i) > 1 - threshold \text{ or } < threshold \\ 0, & \text{if } threshold \leq \text{predict\_proba}(X_i) \leq 1 - threshold \end{cases}$$

where the parameter ***threshold*** will selected as **0.1** in the following classifiers.

2. Classifier

2.1 SVM

By calculating the mean value of y_select, we find the selected neighbours of X_test are not balanced, which means it is very likely that the numbers of sample in two classes are not equal.

$$\text{mean(y\_select)} = 0.2593$$

Considering this, the SVM should be adjusted to the unbalanced data. We set the parameter ***class_weight="balanced"*** to adjust weights inversely proportional to class frequencies in the input data.

Use Gaussian kernel SVC, and set **C=1**. Divide the training set into two parts: try to fit the classifier with the first 20000 samples, and predict the labels of the rest part of samples. Then we get the training score and time spent in fitting the classifier.

```
Training score: 0.12996885393021168
Running time: 364.73169231414795 seconds
```

Based on the running time, we can see that SVC is very expensive with the setting ***probability=True***. This is because SVMs do not directly provide probability estimates, these are calculated using an expensive five-fold cross-validation. So we avoid to use the determining rule defined in 1.3, and turn to ***decision_function()*** to estimate the probability. Define $dist_i = decision\_function(X_i)$

$$\hat{y}_i = \begin{cases} \text{predict}(X_i), & \text{if } \text{dist}_i > 0.8\min(\boldsymbol{dist}) + 0.2\max(\boldsymbol{dist}) \\ 0, & \text{else} \end{cases}$$

Apply this rule and get a new prediction with 20000 training samples, we get the cross validation result

```
Training score: 0.19168030407010506
Running time: 466.835693359375 seconds
```

2.2 Random forest

Since random forest classifier will be easily over fitted if the parameters are ill posed, for optimizing the parameters of random forest classifier, we use *sklearn.grid_search.**RandomizedSearchCV*** to get the optimal max_depth and max_features.

With **n_estimators=50**, we get the CV result that is far from expection, which proves decision tree or random forest is not the best classifier in this case.

```
Training score: 0.9289975188724067
Training time: 89.1927399635315 seconds
```

2.3 MLP (Adopted method)

This is the classifier we finally adopted to classify the test set.

Also use *sklearn.grid_search.**RandomizedSearchCV*** to get the optimal parameters *'alpha', 'learning_rate_init', 'activation',* and *'hidden_layer_sizes'* for MLP classifier. This step is quite expensive, once we get the optimal results, we will fix them in the following process.

The parameters of the best estimator are listed:

activation	'relu'
alpha	0.012337222794656134
batch_size	'auto'
beta_1	0.9
beta_2	0.999
early_stopping	True

epsilon	1e-08
hidden_layer_sizes	(260,)
learning_rate	'adaptive'
learning_rate_init	0.00036445635408383824
max_iter	200
momentum	0.9
nesterovs_momentum	True
power_t	0.5
random_state	1
shuffle	True
solver	'adam'
tol	0.0001
validation_fraction	0.3
verbose	False
warm_start	False

Apply the best estimator obtained in the previous step, fitting with the first 30000 samples of the selected dataset, and predicting the labels of the rest part. We get the results as

```
Training score: 0.105390625
Running time: 20.481074810028076 seconds
```

2.4 Boosting recursively

In the test set, as the MLP classifier will leave too many uncertain prediction, now we try to reduce the uncertain predicted samples recursively, by adding the certainly predicted samples to the training set.

If the uncertainly predicted samples are fewer than last iteration, we consider our algorithm can be still optimised by adding the volume of training set; otherwise, we consider it has converged.

With the iteration, we get the increasing number of certainly predicted samples (+1/-1) in the test set.

```
Exactly estimate 92.62 % of test dataset.
Exactly estimate 96.06 % of test dataset.
Exactly estimate 96.29 % of test dataset.
Exactly estimate 96.95 % of test dataset.
Exactly estimate 96.17 % of test dataset.
```