EE7207-NEURAL & FUZZY SYSTEM

Solution of Continuous Assessment 1

NAME: ZHOU YICHEN

Matric NO.: G2100967F

EMAIL: YICHEN002@e.ntu.edu.sg

Github: https://github.com/Ea510chan/EE7207

I. RBF neural network classifier

A. Network Structure

a. Radial basis function: The Gaussian function

I choose *sklearn.metrics.pairwise.rbf_kernel* to compute the RBF(Gaussian) kernel between x and c_i as shown below.

$$K(x,c_i) = \exp(-gamma\|x-c_i\|^2)$$

Where x is the input vector; c_i is called center vector; *gamma* is the parameter about the width of the basis function, which directly determines the size of the receptive field.

b. Center selection

I choose K-Means clustering to generate center vectors, which can be achieved by using *sklearn.cluster.KMeans*.

c. Weight estimation

Use linear least square estimates to calculate weight as shown below.

$$w = (\Phi^T\Phi)^{-1}\Phi^T d$$

Where Φ is the output of hidden layer, which is calculated by radial basis function; d is the target value.

B. Get the best parameters of network

Use *sklearn.model_selection.GridSearchCV* to search over specified parameter values for an estimator. In RBF neural network, we need to find the best pair of parameters: (*no. of* c_i, *gamma*,), which perform the best in the training process.

Given 10×10 *gamma* and c_i to train the training set and score the model. The result of the scores is shown below in Figure 1. It's clear from Figure 1 that the best pair of parameters is (0.34, 70), which the score of cross-validation is 0.79.

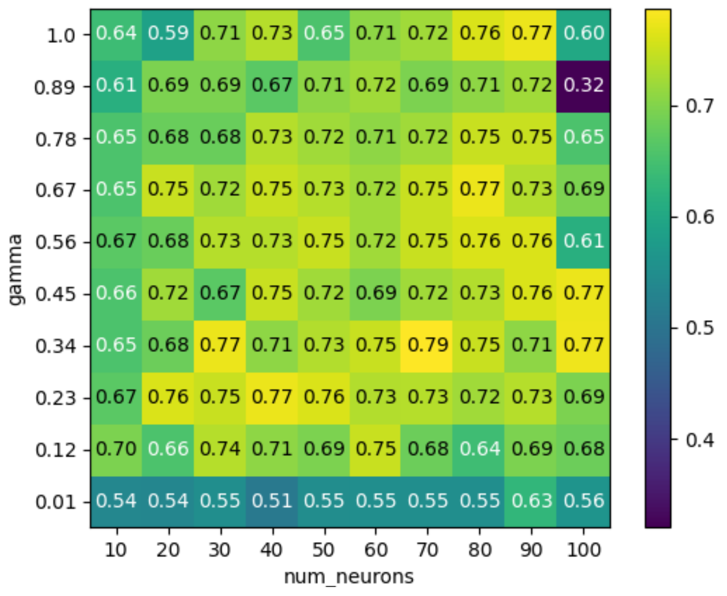

Figure 1 Cross Validation Scores (RBF)

II. Kernel SVM classifier

A. Network Structure: Gaussian kernel

I choose *sklearn.svm.SVC* to do classifying, which can apply Gaussian kernel. The details of this function is given below.

$$sklearn.svm.SVC(C, kernel = 'rbf', gamma)$$

Where C is regularization parameter, which directly influences the performance of classifier; *gamma* is called kernel coefficient of 'rbf', which is related to number of features in Gaussian kernel.

B. Get the best parameters of network

The operation is same with what we do in RBF neural network. Given 6×6 *gamma* and C to find the best pair of parameters. The result is shown below in Figure 2. As we can see from Figure 2, SVM performs best at (*gamma, C*) = (1, 1).

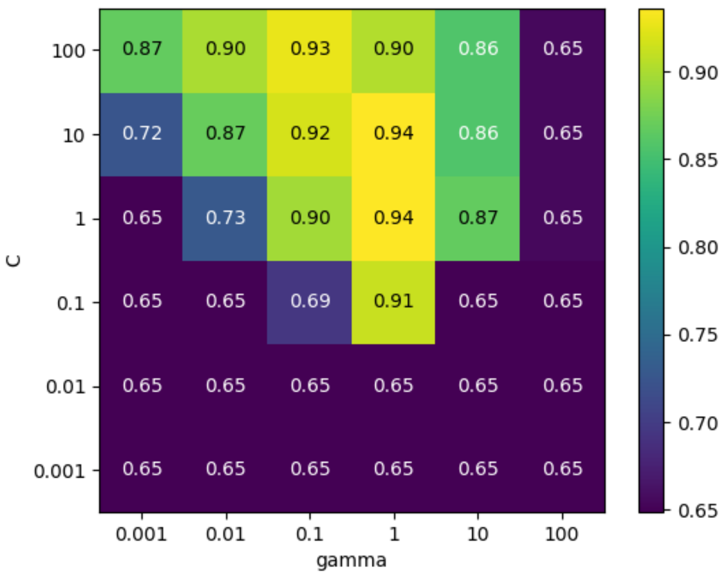

Figure 2 Cross Validation Scores (SVM)

III. Comparison of two classifiers on the training data

A. Comparison of cross validation results

Use *sklearn.model_selection.Kfold* to spilt training data into 4 pieces and do cross validation, calculating the accuracy of training and valid set. The result is shown below. The parameters of Kfold is defined by:

KFold(*n_splits=4, shuffle=False*)

For RBF classifier, it's obvious that the accuracy of valid data is much lower than the accuracy of training data each time. To be specific, the accuracy of valid data in the first time accounts for the lowest accuracy in 4 times training, which is only 0.651.

Cross validation times	Accuracy of training data	Accuracy of valid data
1	0.968	0.651
2	0.992	0.904
3	0.984	0.951
4	0.964	0.977
Mean accuracy	0.977	0.870

For the SVM classifier, the accuracy of training data is similar to the accuracy of training data in the RBF classifier, which is close to 1. The accuracy of valid data fluctuates from 0.91 to 0.98. Therefore, SVM shows the strong ability of global optimization.

Cross validation times	Accuracy of training data	Accuracy of valid data
1	0.976	0.916
2	0.992	0.940
3	0.972	0.939
4	0.980	0.976
Mean accuracy	0.980	0.943

From the table, we can know that the accuracy of valid data in the RBF classifier is unstable, which indicates that the ability of global optimization in this classifier is weaker than the SVM classifier.

As mentioned in the slide, the hidden layer in RBF consists of the locally sensitive neurons, whose response(output) is localized and decays as a Gaussian function of the distance from the input to the center of the neuron's receptive field. So the result of the output layer is highly sensitive with this distance, which is determined by the centers of neurons. In my RBF neural network, centers of neurons are selected by K-means clustering and different training data can produce different results of centers. Therefore, the accuracy of valid data is low if the network selects unsuitable centers which can't predict valid data well but fit training data perfectly.

In the SVM network, the decision result is only determined by the support vector. So if we add or delete the input vectors which do not support the vector, it has no effect on the ability of classification which means the accuracy of valid data is high and stable. Therefore, we conclude that SVM is robust and not sensitive to the number of changes of input vectors if we don't delete the support vector.

B. Comparison of decision boundaries

I Apply t-SNE to visualize the training data, the result is shown below.

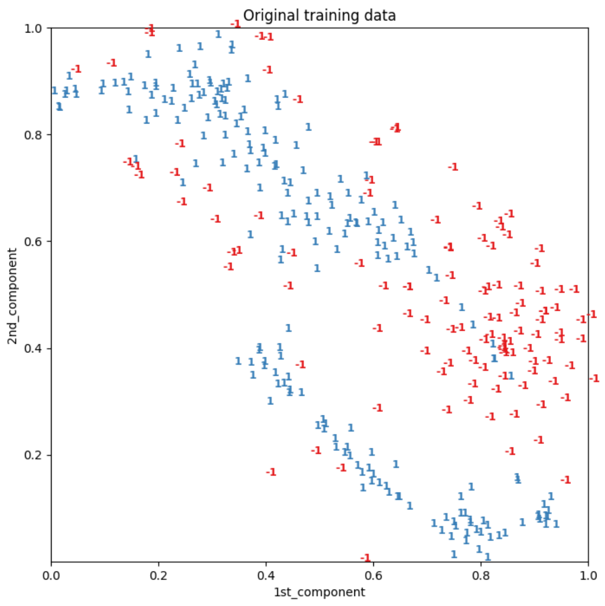

Figure 3 Scatter Diagram of Training Data

Then I use the training data which is preprocessed by t-SNE to draw the decision boundaries, the result is shown below. The black ellipse indicates the points which are classified wrongly by the decision boundary. The blue dot represents '-1' and the red dot represents '1'.

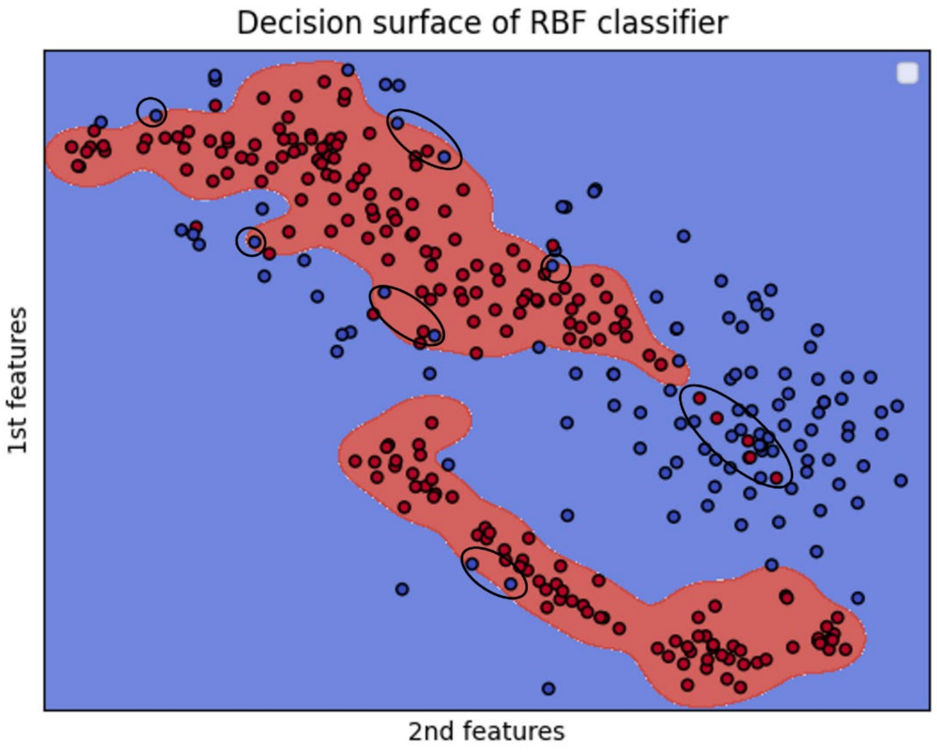

Figure 4 Decision bounary of RBF

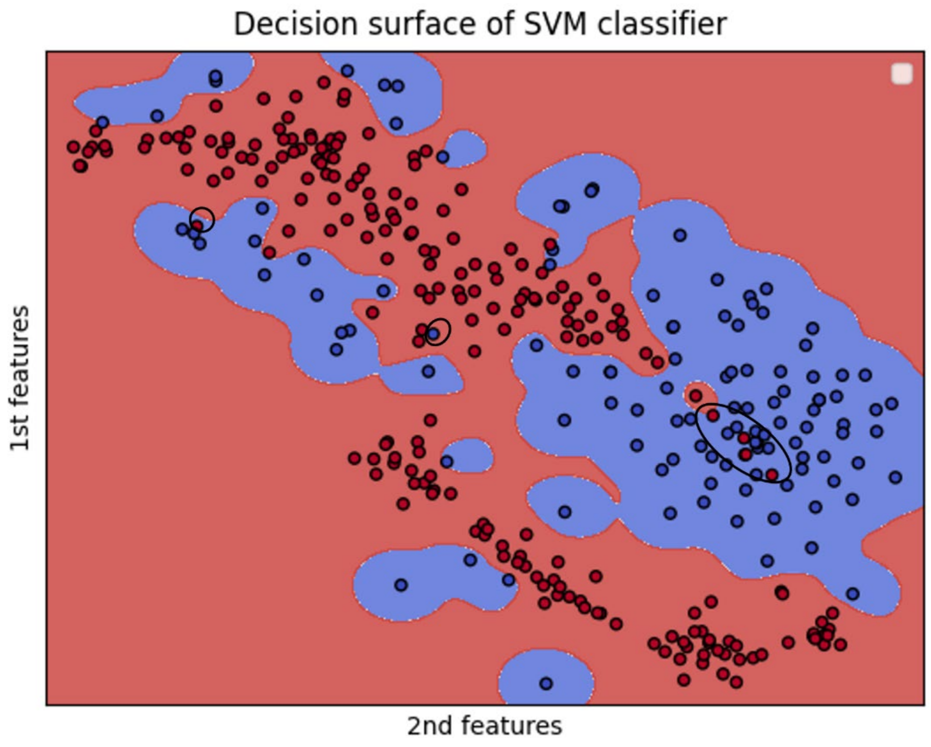

Figure 5 Decision boundary of SVM

As we can see from the above two figures, The decision boundary of SVM is more accurate than the RBF.

IV. Prediction of testing data

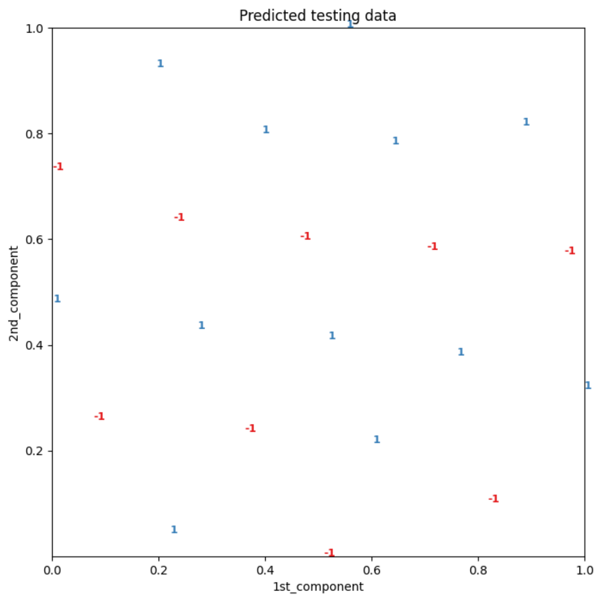

Figure 6 Scatter Diagram of Testing Data

A. For RBF classifier:

$[\ 1\ \ -1\ \ 1\ \ -1\ \ 1\ \ -1\ \ 1\ \ -1\ \ 1\ \ -1\ \ 1\ \ -1\ \ 1\ \ 1\ \ 1\ \ -1\ \ 1\ \ -1\ \ 1\ \ -1\ \ 1\ ]^T$

B. For SVM classifier:

$[\ 1\ \ -1\ \ 1\ \ -1\ \ 1\ \ -1\ \ 1\ \ -1\ \ 1\ \ -1\ \ 1\ \ -1\ \ 1\ \ 1\ \ 1\ \ -1\ \ 1\ \ -1\ \ 1\ \ -1\ \ 1\ ]^T$

V. APPENDIX

A. RBF classifier

```python
# -*- coding: utf-8 -*-
# @author:ZHOU YICHEN
import numpy as np
from RBF_classifier import RBF
import scipy.io as sio
from sklearn.model_selection import GridSearchCV, KFold, StratifiedKFold
from sklearn.preprocessing import MinMaxScaler
from visualization import plot_cvscores_RBF, plot_decision

# Load data
matfn=u'datasets.mat'
data=sio.loadmat(matfn)
data_train = data['data_train']
label_train = data['label_train']
data_test = data['data_test']
# Feature scaling
scaler1 = MinMaxScaler().fit(data_train)
scaler2 = MinMaxScaler().fit(data_test)
X_train, X_test = scaler1.transform(data_train), scaler2.transform(data_test)
# define params
params = {'num_neurons': list(np.linspace(10, 100, 10).astype(int)),
          "gamma": list(np.linspace(0.01, 1, 10))}
# use GridsearchCV to find the best params
skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=5)
grid_search = GridSearchCV(RBF(), params, cv=skf, return_train_score=True)
grid_result = grid_search.fit(X_train, label_train)
scores = grid_search.cv_results_['mean_test_score'].reshape(10, 10)
plot_cvscores_RBF(scores, params)
print(skf)
# produce the best model
model = grid_search.best_estimator_
# KFold cross validation
kfold = KFold(n_splits=4, shuffle=False)
acc_train = 0
acc_valid = 0
for train_index, valid_index in kfold.split(label_train):  # 4-fold

    model.fit(X_train[train_index], label_train[train_index])
    res_train = model.predict(X_train[train_index])
    res_valid = model.predict(X_train[valid_index])
    # print(accuracy_score(res_train, label_train[train_index]))
    # print(accuracy_score(res_valid, label_train[valid_index]))
      acc_train += float(sum(res_train == label_train[train_index]) / len(train_index))
      acc_valid += float(sum(res_valid == label_train[valid_index]) / len(valid_index))

print('ACC using SVM: ' + str(acc_train / 4) +'|'+'Valid: ' + str(acc_valid / 4))

# predict the label of testing set
y_hat = model.predict(X_test)

print("Best: %f using %s" % (grid_result.best_score_, grid_search.best_params_))
means = grid_result.cv_results_['mean_test_score']
params = grid_result.cv_results_['params']
for mean,param in zip(means,params):
    print("%f  with:   %r" % (mean,param))
print(y_hat.reshape(1,-1))
plot_decision(data_train, label_train, model)
======================================
#Define the RBF neural network
import numpy as np
from sklearn.metrics.pairwise import rbf_kernel
from sklearn.cluster import KMeans
from sklearn.base import BaseEstimator, RegressorMixin
from sklearn.metrics import r2_score

class RBF(BaseEstimator, RegressorMixin):
    # define hidden neurons, gamma, weights
    def __init__(self, num_neurons=10, gamma=1 , w_weights=None):
        self.num_neurons = num_neurons
        self.gamma = gamma
        self.w_weights = w_weights

    def fit(self, x_train, y_train):
        x_train = np.c_[-1 * np.ones(x_train.shape[0]), x_train]
        kmeans = KMeans(n_clusters=self.num_neurons).fit(x_train)
        self.centers = kmeans.cluster_centers_
        H = rbf_kernel(x_train, self.centers, gamma=self.gamma)
        H = np.c_[-1 * np.ones(H.shape[0]), H]
        try:
         self.w_weights = np.linalg.lstsq(H, np.asmatrix(y_train).T, rcond=-1)[0]
        except:
            self.w_weights = np.linalg.pinv(H) @ y_train.reshape(-1, 1)
        return self

    def predict(self, x_test):
        x_test = np.c_[-1 * np.ones(x_test.shape[0]), x_test]
        H = rbf_kernel(x_test, self.centers, gamma=self.gamma)
        H = np.c_[-1 * np.ones(H.shape[0]), H]
        y_hat = np.asmatrix(H) @ np.asmatrix(self.w_weights)
        y_len = len(y_hat)
        for i in range(0, y_len):
            if y_hat[i,:] < 0:
                y_hat[i,:] = -1
            else:
                y_hat[i,:] = 1
        return y_hat

    def score(self, X, y, sample_weight=None):
        # from scipy.stats import pearsonr
        # r, p_value = pearsonr(y.reshape(-1, 1), self.predict(X))
        # return r ** 2
        return r2_score(y.reshape(-1, 1), self.predict(X))
```

B. SVM Classifier

```python
# -*- coding: utf-8 -*-
# @author:ZHOU YICHEN
import scipy.io as sio
from sklearn.model_selection import GridSearchCV, cross_val_score, KFold
from sklearn.svm import SVC
from sklearn.preprocessing import MinMaxScaler
import pandas as pd
from visualization import plot_cvscores_SVM, plot_tsne, plot_decision, plot_leaning
from sklearn.metrics import accuracy_score

# load data
matfn = u'datasets.mat'
data = sio.loadmat(matfn)
data_train = data['data_train']
label_train = data['label_train']
data_test = data['data_test']
plot_tsne(data_train, label_train.ravel())

# Feature scaling
scaler1 = MinMaxScaler().fit(data_train)
scaler2 = MinMaxScaler().fit(data_test)
X_train, X_test = scaler1.transform(data_train), scaler2.transform(data_test)

# define the type of SVM kernel: rbf(Gaussian kernel function) and range of C and gamma
params = {'kernel': ['rbf'],'C': [0.001, 0.01, 0.1, 1, 10, 100],'gamma': [0.001, 0.01, 0.1,
1, 10, 100]}

# search the best params in SVM
grid_search = GridSearchCV(SVC(), params, cv=5, return_train_score=True)
grid_result = grid_search.fit(X_train, label_train.ravel())

# save the GridsearchCV results and plot the scores of params
results = pd.DataFrame(grid_search.cv_results_)
scores = grid_search.cv_results_['mean_test_score'].reshape(6, 6)
plot_cvscores_SVM(scores, params)

# get the best params and produce the best model
model = grid_search.best_estimator_

# KFold cross validation
kfoldscores = cross_val_score(estimator=model, X=X_train, y=label_train.ravel(), cv=10, n_jo
bs=1)
```

```python
print(kfoldscores)

kfold = KFold(n_splits=4, shuffle=False)
acc_train = 0
acc_valid = 0
for train_list, valid_list in kfold.split(label_train):  # 4-fold

    model.fit(X_train[train_list], label_train[train_list])
    res_train = model.predict(X_train[train_list])
    res_valid = model.predict(X_train[valid_list])
    print(accuracy_score(res_train, label_train[train_list]))
    print(accuracy_score(res_valid, label_train[valid_list]))
    acc_train += float(sum(res_train == label_train[train_list].squeeze()) / len(train_list)
)
    acc_valid += float(sum(res_valid == label_train[valid_list].squeeze()) / len(valid_list)
)

print('ACC using SVM: ' + str(acc_train / 4) +'|'+'Valid: ' + str(acc_valid / 4))

# predict the label of testing set
print("Best: %f using %s" % (grid_result.best_score_, grid_search.best_params_))
means = grid_result.cv_results_['mean_test_score']
params = grid_result.cv_results_['params']
for mean,param in zip(means,params):
    print("%f  with:   %r" % (mean,param))
y_hat = model.predict(X_test)
plot_tsne(X_test, y_hat.ravel())
print(y_hat)
plot_decision(data_train, label_train, model)
```

C. Visualization

```python
import numpy as np
import matplotlib.pyplot as plt
import mglearn
from sklearn.manifold import TSNE
from sklearn.metrics import  accuracy_score

# visualize cross-validation params
# plot the mean cross-validation scores
def plot_cvscores_RBF(scores, params):
    scores_image = mglearn.tools.heatmap(scores, xlabel='num_neurons', xticklabels=params['num_neurons'],
                          ylabel='gamma', yticklabels=params['gamma'], cmap="viridis")

    plt.colorbar(scores_image)
    plt.show()

def plot_cvscores_SVM(scores, params):
    scores_image = mglearn.tools.heatmap(scores, xlabel='gamma', xticklabels=params['gamma'],
                          ylabel='C', yticklabels=params['C'], cmap="viridis")

    plt.colorbar(scores_image)
    plt.show()

# visualize training set results
def plot_tsne(data, label):
    X = data
    y = label
    tsne = TSNE(n_components=2, init='pca', random_state=501)
    X_tsne = tsne.fit_transform(X)
    x_min, x_max = X_tsne.min(0), X_tsne.max(0)
    X_norm = (X_tsne - x_min) / (x_max - x_min)  # 归一化
    plt.figure(figsize=(8, 8))
    for i in range(X_norm.shape[0]):
        plt.text(X_norm[i, 0], X_norm[i, 1], str(y[i]), color=plt.cm.Set1(y[i]),
                 fontdict={'weight': 'bold', 'size': 9})
    plt.xticks([0, 0.2, 0.4, 0.6, 0.8, 1])
    plt.yticks([0.2, 0.4, 0.6, 0.8, 1])
    plt.xlabel('1st_component')
    plt.ylabel('2nd_component')
    plt.title('Predicted testing data')
    plt.show()
```

```python

def plot_decision(data, label, model):
    tsne = TSNE(n_components=2, init='pca', random_state=501)
    X_tsne = tsne.fit_transform(data)
    x = X_tsne[:, 0]
    y = X_tsne[:, 1]
    model.fit(X_tsne, label)
    def make_meshgrid(x, y , h=.02):
        x_min, x_max = x.min() - 1, x.max() + 1
        y_min, y_max = y.min() - 1, y.max() + 1
        xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))
        return xx, yy

    def plot_contours(ax, clf, xx, yy, **params):
        Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
        Z = Z.reshape(xx.shape)
        out = ax.contourf(xx, yy, Z, **params)
        return out

    fig, ax = plt.subplots()
    # title for the plots
    title = ('Decision surface of SVM classifier ')
    # Set-up grid for plotting.
    xx, yy = make_meshgrid(x, y)
    plot_contours(ax, model, xx, yy, cmap=plt.cm.coolwarm, alpha=0.8)
    ax.scatter(x, y, c=label, cmap=plt.cm.coolwarm, s=20, edgecolors='k')
    ax.set_ylabel('1st features')
    ax.set_xlabel('2nd features')
    ax.set_xticks(())
    ax.set_yticks(())
    ax.set_title(title)
    ax.legend()
    plt.show()

def plot_leaning(data, label, model):
    X_train = data
    y_train = label.ravel()
    scores = []
    for m in range(2,330):#循环2-79
        model.fit(X_train[:m, :], y_train[:m])
        y_train_predict = model.predict(X_train[:m])
        scores.append(accuracy_score(y_train_predict,y_train[:m]))
    plt.plot(range(2,330),scores,c='green', alpha=0.6)
    plt.show()
```