Project 3

Cosc425

SiCheng Yi

I. Project Description

(1). Describe the purpose of the project.

The purpose of this Projcet is to apply support vector classification (SVC). The SVC will directly use the SVC given in scikit-learn.

(2). Describe your data that you're examining.

We need to apply SVC to three different data sets. The first is Predicting Good/Bad Interactions of Radar Signals. With 34 measurements and a label (at the end), this data has 351 instances.

The second is Terrain Classification for Multispectrcal Pixel Values, with 7 classes and 36 features, and the last data is also label. There are 6,435 examples, divided into training set (sat.trn) and test set (sat.tst).

II. Pre-Processing Steps

(1). Explain the format of the initial data. Explain how you've manipulated the data including adjusting missing values, separating training and test data, etc.

#We first standardize the data and optimize to pick up between [0,1] by default

X = preprocessing.scale(X)

There are three different data sets. The first is Predicting Good/Bad Interactions of

Radar Signals. With 34 measurements and a label (at the end), this data has 351 instances.

Import the data set, divided into train and test sets:

```
digits = datasets.load_digits()

n_samples = len(digits.images)
X = digits.images.reshape((n_samples, -1))
y = digits.target

X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.5, random_state=0)
```

The second is Terrain Classification for Multispectrcal Pixel Values, with 7 classes and 36 features, and the last data is also label. There are 6,435 examples, divided into training set (sat.trn) and test set (sat.tst).

There are 4,435 instances in the training set, and 2,000 instances in the test set

Because this data has been separated, we do not need to use cross-validation.

III. Solution Description

(1). Detail the technical implementation (e.g., language, parameters, etc.).

kernel: kernel function selection, string type, optional "linear", "poly", "rbf", "sigmoid", "precomputed" and custom kernel function, the default selection is "rbf".

Many samples, many features, two classifications, choose linear kernel function

Many samples, many features, many categories, polynomial kernel function

Not many samples, many features, two-class/multi-class, Gaussian kernel function

Not many samples, not many features, two-class/multi-class, Gaussian kernel function

Here is my choice recommendation. We use cross-validation to select features.

First Read data characteristics and data tags

For example:

```
path1 = r"Test1_features.dat"

X =sat.trn_csv(path1,engine='python',header=None)

path2 = r"Test1_labels.dat"

Y = sat.trn_csv(path2,engine='python',header=None)
```

Then data standardization

```
X = preprocessing.scale(X)
```

Then select the parameters for mesh optimization

AUC indicator when calculating different parameters

Eg:

```
x = y = z = []
for C in range(1,10,1):
    for gamma in range(1,11,1):
cross_val_score(SVC(C=C,kernel='rbf',gamma=gamma/10),X,Y,cv=5,scoring='roc_a
uc').mean()
        x.append(C)
        y.append(gamma/10)
        z.append(auc)
```

then convert the list to a two-dimensional array

```
x = np.array(x).reshape(9,10)

y = np.array(y).reshape(9,10)

z = np.array(z).reshape(9,10)
```

Finally draw the picture, the drawing program comes from svm.py

We want to use grid search,

I think Grid search is a actually brute force search:

First, set a set of candidate values for the parameters you want to adjust, and then the grid search will exhaust various parameter combinations, and find the best set of settings according to the set scoring mechanism.

In the selection of all candidate parameters, through looping and trying every possibility, the best performing parameter is the final result. The principle is like finding the maximum value in an array.

For example, we use the following two grids:

kernel=’rbf’, gamma, ‘C’

kernel=’linear’, ‘C’

```
tuned_parameters = [{'kernel': ['rbf'],'gamma': [1e-3, 1e-4],
                                 'C': [1, 10, 100, 1000]},
                                {'kernel': ['linear'],'C': [1, 10, 100, 1000]}]
```

```
tuned_parameters = [{'kernel': ['rbf'],'gamma': [1e-3, 1e-4],
                    'C': [1, 10, 100, 1000]},
                    {'kernel': ['linear'],'C': [1, 10, 100, 1000]}]
```

Define the scoring method as:

```
scores = ['precision','recall']
```

Eg2:

```
best_score = 0
for gamma in [0.001,0.01,0.1,1,10,100]:
    for C in [0.001,0.01,0.1,1,10,100]:
        svm = SVC(gamma=gamma,C=C)# Perform a training for each possible combination of parameters;
        svm.fit(X_train,y_train)
        score = svm.score(X_test,y_test)
        if score> best_score: #Find the best performing parameter
            best_score = score
            best_parameters = {'gamma':gamma,'C':C}
```

Eg3:

```
X, y = load_digits(return_X_y = True)
parameters = {'gamma': [0.001, 0.01, 0.1, 1],'C':[0.001, 0.01, 0.1, 1,10]}
```

```
gs = GridSearchCV(SVC(), parameters, refit = True, cv = 5, verbose = 1, n_jobs = -1)

gs.fit(X,y) #Run fit with all sets of parameters.

print('Optimal parameter:',gs.best_params_)

print('Best performance:', gs.best_score_)
```

(2). Describe equations in your learning method. Describe the equation's variables

Hyperplane:

If a point is represented in one-dimensional space, a straight line is represented in two-dimensional space, and a plane is represented in three-dimensional space, of course, the space dimension can be more, so we give this linear function a name called "super flat". The mathematical expression of hyperplane can be written as:

$$g(x) = \omega^T x + b$$

w and x are vectors in n-dimensional space, where x is a function variable; w is a normal vector.

We use di to represent the Euclidean distance from point xi to the hyperplane wxi+b=0. Therefore, we require the minimum value of di and use it to represent the shortest distance from this sample to the hyperplane. di can be calculated by the formula:

$$d_i = \frac{|\omega x_i + b|}{\|\omega\|}$$

C

float type, optional parameter, default is 1.0

Penalty factor

(The tolerance of the interval error controls the complexity of the classifier to a certain extent, also known as the complexity parameter, which is also a kind of regularization)

kernel

string type, optional, default is rbf, which means radial basis function

Represents a certain kernel function type. Must be one of 'linear', 'poly', 'rbf', 'sigmoid', 'precomputed' and 'callable', including linear

linear: $K(\mathrm{x}_i, \mathrm{x}_j) = \mathrm{x}_i^T \mathrm{x}_j$.

polynomial: $K(\mathrm{x}_i, \mathrm{x}_j) = (\gamma {\mathrm{x}_i}^T \mathrm{x}_j + r)^d$, $\gamma > 0$.

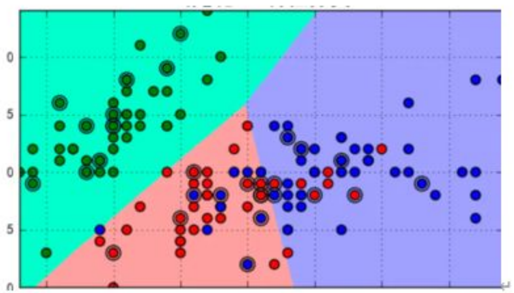
radial basis function (RBF): $K(\mathrm{x}_i, \mathrm{x}_j) = \exp(-\gamma \|\mathrm{x}_i - \mathrm{x}_j\|^2)$, $\gamma > 0$.

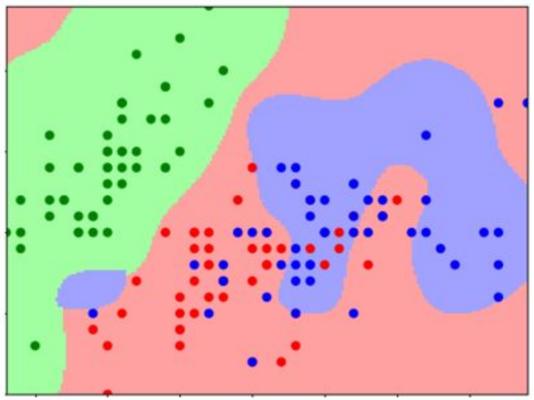
sigmoid: $K(\mathrm{x}_i, \mathrm{x}_j) = \tanh(\gamma {\mathrm{x}_i}^T \mathrm{x}_j + r)$.

y, r, and d are kernel parameters.

IV. Evaluation and Analysis

(1). Provide results in a legible format (e.g. graphical plots, tables, clear writing)





(2). Explain your results (e.g., accuracy) in as much detail as possible.

When kernel='linear', it is a linear kernel. The larger the C, the better the classification effect, but it may overfit (defaul C=1).

When kernel='rbf' (default), it is a Gaussian kernel. The smaller the gamma value, the more continuous the classification interface; the larger the gamma value, the more "scattered" the classification interface, and the better the classification effect, but it may overfit.

When decision_function_shape='ovr', it is one v rest (one-to-many), that is, one category is divided from other categories,

When decision_function_shape='ovo', it is one v one (one to one), that is, the two categories are divided between two categories, and the result of multi-category is simulated by the two-category method.

The larger the gamma value, the better the fit of the training set, but it will cause over-fitting, resulting in poor fit of the test set.

The smaller the gamma value, the better the generalization ability of the model. The fitting of the training set and the test set is similar, but it will cause underfitting of the training set, which will lower the accuracy rate and result in the test set accuracy rate.

Calculate the accuracy of the model:

```
print (clf.score(x_train, y_train))
print ('training set accuracy:', accuracy_score(y_train, clf.predict(x_train)))
print (clf.score(x_test, y_test))
print ('Test set accuracy:', accuracy_score(y_test, clf.predict(x_test)))
```

V. Discussion

(1). Provide a summarization of your results.