20171847118 金正旭

1.PCA+逻辑回归（Iris）

- 首先导入我们需要的包名，并指明文件为UTF-8编码：

```
In [1]: import numpy as np
   ...: import matplotlib.pyplot as plt
   ...: import matplotlib as mpl
   ...: from matplotlib import colors
   ...: from sklearn.decomposition import PCA
   ...: from sklearn.linear_model import LogisticRegression
   ...: from sklearn.model_selection import train_test_split
```

- 定义精度函数,数学定义为 $$ \begin{align*} & P=\frac{TP}{TP+FP} \end{align*} $$

```
In [2]: def show_accuracy(y_hat, y_test, param):
   ...:     pass
```

- 通过input_file读入本地数据

```
In [3]: input_file = '/home/jason/Documents/ML/flower/Data/Iris.data'
   ...: line,lines,data=[],[],[]
   ...: Flower_set,Flower_sample,Flower_lable=[],[],[]
   ...: file=open(input_file,'r')
   ...: lines = file.readlines()
   ...: for line in lines:
   ...:     data=line.split(',')
   ...:     Flower_set.append(list(map(float,data)))
   ...: Flower_set=np.array(Flower_set)
   ...: Flower_sample=Flower_set[...,0:4]
   ...: Flower_lable=Flower_set[...,4]
   ...: x=Flower_sample
   ...: y=Flower_lable
```

- 通过PCA对数据降维，PCA是通过如下的协方差来比较变量X和Y的 $$ \begin{align*} & cov(X,Y)= \frac{1}{n-1} \sum^{n}*{i=1}(X*{i}-\bar{x})(Y_{i}-\bar{y}) \end{align*} $$

```
In [4]: pca=PCA(n_components=2)
   ...: reduced_x=pca.fit_transform(x)
```

- 通过sklearn分隔为测试集和数据集

```
In [5]: x=reduced_x
   ...: x_train, x_test, y_train, y_test = train_test_split(x, y, random_state=1,
train_size=0.6)
```

- 逻辑回归模型，展示在训练集和测试集上的精度

```
In [6]: clf=LogisticRegression()
   ...: clf.fit(x_train, y_train.ravel())
   ...: print (clf.score(x_train, y_train))  # 精度
   ...: y_hat = clf.predict(x_train)
   ...: show_accuracy(y_hat, y_train, 'train_set')
   ...: print (clf.score(x_test, y_test))
   ...: y_hat = clf.predict(x_test)
```

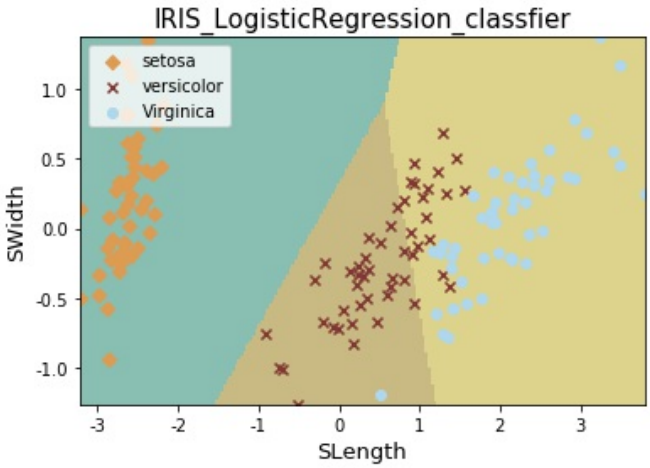
```
   ...: show_accuracy(y_hat, y_test, 'test_set')

Out [6]: 0.8888888888888888
    ...: 0.85
```

可以看到我们的模型在训练集上的精度达到了0.89在测试集上为0.85

- 画图

```
In [8]: x1_min, x1_max = x[:, 0].min(), x[:, 0].max()  # 第0列的范围
   ...: x2_min, x2_max = x[:, 1].min(), x[:, 1].max()  # 第1列的范围
   ...: x1, x2 = np.mgrid[x1_min:x1_max:200j, x2_min:x2_max:200j]  # 生成网格采样点
   ...: grid_test = np.stack((x1.flat, x2.flat), axis=1)  # 测试点
   ...: mpl.rcParams['font.sans-serif'] = [u'SimHei']
   ...: mpl.rcParams['axes.unicode_minus'] = False
   ...: cm_light = mpl.colors.ListedColormap(['#89beb2', '#c9ba83', '#ded38c'])
   ...: cm_dark = mpl.colors.ListedColormap(['g', 'r', 'b'])
   ...: grid_hat = clf.predict(grid_test)       # 预测分类值
   ...: grid_hat = grid_hat.reshape(x1.shape)  # 使之与输入的形状相同
   ...: alpha = 0.5
   ...: plt.pcolormesh(x1, x2, grid_hat, cmap=cm_light)     # 预测值的显示
   ...: red_x,red_y=[],[]
   ...: blue_x,blue_y=[],[]
   ...: green_x,green_y=[],[]
   ...: for i in range(len(x)):
   ...:     if y[i] ==0:
   ...:         red_x.append(x[i][0])
   ...:         red_y.append(x[i][1])
   ...:     elif y[i]==1:
   ...:         blue_x.append(x[i][0])
   ...:         blue_y.append(x[i][1])
   ...:     else:
   ...:         green_x.append(x[i][0])
   ...:         green_y.append(x[i][1])
   ...:plt.scatter(red_x,red_y,c='#de9c53',marker='D',label='setosa')
   ...: plt.scatter(blue_x,blue_y,c='#823935',marker='x', label='versicolor')
   ...: plt.scatter(green_x,green_y,c='#afd7ed',marker='o', label='Virginica')
   ...: plt.scatter(x_test[:, 0], x_test[:, 1], s=120, facecolors='none', zorder=10)
# 圈中测试集样本
   ...: plt.xlabel(u'SLength', fontsize=13)
   ...: plt.ylabel(u'SWidth', fontsize=13)
   ...: plt.xlim(x1_min, x1_max)
   ...: plt.ylim(x2_min, x2_max)
   ...: plt.title(u'IRIS_LogisticRegression_classfier', fontsize=15)
   ...: plt.legend(loc=2)
   ...: plt.show()
```

得到如下的结果：



2.调试程序

- utilities.py

对程序所做修改如下cross_validation 已经在0.18版本后被移除，所以使用的话要使用如下的语句 `from sklearn.model_selection import cross_validate`，并且修改函数中的语句

听名字就可以知道这个文件里都是一些工具，这个程序中主要定义了三个函数，分别是

1. load_data()

- 读入数据并将数据分为X和y两个数组

2. plot_classifier()

- 画图体现数据分类结果

3. print_accuracy_report()

- 展示精度等参数

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import cross_validate

# Load multivar data in the input file
def load_data(input_file):
    X = []
    y = []
    with open(input_file, 'r') as f:
        for line in f.readlines():
            data = [float(x) for x in line.split(',')]
            X.append(data[:-1])
            y.append(data[-1])

    X = np.array(X)
    y = np.array(y)

    return X, y

# Plot the classifier boundaries on input data
def plot_classifier(classifier, X, y, title='Classifier boundaries', annotate=False):
    # define ranges to plot the figure
    x_min, x_max = min(X[:, 0]) - 1.0, max(X[:, 0]) + 1.0
    y_min, y_max = min(X[:, 1]) - 1.0, max(X[:, 1]) + 1.0

    # denotes the step size that will be used in the mesh grid
    step_size = 0.01

    # define the mesh grid
    x_values, y_values = np.meshgrid(np.arange(x_min, x_max, step_size),
np.arange(y_min, y_max, step_size))

    # compute the classifier output
    mesh_output = classifier.predict(np.c_[x_values.ravel(), y_values.ravel()])

    # reshape the array
    mesh_output = mesh_output.reshape(x_values.shape)

    # Plot the output using a colored plot
    plt.figure()
```

```
    # Set the title
    plt.title(title)

    # choose a color scheme you can find all the options
    # here: http://matplotlib.org/examples/color/colormaps_reference.html
    plt.pcolormesh(x_values, y_values, mesh_output, cmap=plt.cm.gray)

    # Overlay the training points on the plot
    plt.scatter(X[:, 0], X[:, 1], c=y, s=80, edgecolors='black', linewidth=1,
cmap=plt.cm.Paired)

    # specify the boundaries of the figure
    plt.xlim(x_values.min(), x_values.max())
    plt.ylim(y_values.min(), y_values.max())

    # specify the ticks on the X and Y axes
    plt.xticks(())
    plt.yticks(())

    if annotate:
        for x, y in zip(X[:, 0], X[:, 1]):
            # Full documentation of the function available here:
            # http://matplotlib.org/api/text_api.html#matplotlib.text.Annotation
            plt.annotate(
                '(' + str(round(x, 1)) + ',' + str(round(y, 1)) + ')',
                xy = (x, y), xytext = (-15, 15),
                textcoords = 'offset points',
                horizontalalignment = 'right',
                verticalalignment = 'bottom',
                bbox = dict(boxstyle = 'round,pad=0.6', fc = 'white', alpha = 0.8),
                arrowprops = dict(arrowstyle = '-', connectionstyle = 'arc3,rad=0'))

# Print performance metrics
def print_accuracy_report(classifier, X, y, num_validations=5):
    accuracy = cross_validate.cross_val_score(classifier,
            X, y, scoring='accuracy', cv=num_validations)
    print ("Accuracy: " + str(round(100*accuracy.mean(), 2)) + "%")

    f1 = cross_validate.cross_validation.cross_val_score(classifier,
            X, y, scoring='f1_weighted', cv=num_validations)
    print ("F1: " + str(round(100*f1.mean(), 2)) + "%")

    precision = cross_validate.cross_val_score(classifier,
            X, y, scoring='precision_weighted', cv=num_validations)
    print ("Precision: " + str(round(100*precision.mean(), 2)) + "%")

    recall = cross_validate.cross_val_score(classifier,
            X, y, scoring='recall_weighted', cv=num_validations)
    print ("Recall: " + str(round(100*recall.mean(), 2)) + "%")
```

- svm.py

```
import numpy as np
import matplotlib.pyplot as plt
import utilities

# Load input data
input_file = 'data_multivar.txt'
X, y = utilities.load_data(input_file)

###############################################
# Separate the data into classes based on 'y'
```

```
class_0 = np.array([X[i] for i in range(len(X)) if y[i]==0])
class_1 = np.array([X[i] for i in range(len(X)) if y[i]==1])

# Plot the input data
plt.figure()
plt.scatter(class_0[:,0], class_0[:,1], facecolors='black', edgecolors='black',
marker='s')
plt.scatter(class_1[:,0], class_1[:,1], facecolors='None', edgecolors='black',
marker='s')
plt.title('Input data')

###############################################
# Train test split and SVM training
from sklearn import cross_validation
from sklearn.svm import SVC

X_train, X_test, y_train, y_test = cross_validation.train_test_split(X, y,
test_size=0.25, random_state=5)

params = {'kernel': 'linear'}
#params = {'kernel': 'poly', 'degree': 3}
#params = {'kernel': 'rbf'}
classifier = SVC(**params)
classifier.fit(X_train, y_train)
utilities.plot_classifier(classifier, X_train, y_train, 'Training dataset')

y_test_pred = classifier.predict(X_test)
utilities.plot_classifier(classifier, X_test, y_test, 'Test dataset')

###############################################
# Evaluate classifier performance

from sklearn.metrics import classification_report

target_names = ['Class-' + str(int(i)) for i in set(y)]
print ("\n" + "#"*30)
print ("\nClassifier performance on training dataset\n")
print (classification_report(y_train, classifier.predict(X_train),
target_names=target_names))
print ("#"*30 + "\n")

print ("#"*30)
print ("\nClassification report on test dataset\n")
print (classification_report(y_test, y_test_pred, target_names=target_names))
print ("#"*30 + "\n")

plt.show()
```

运行后得到：

```
$ python svm.py
##############################

Classifier performance on training dataset

              precision    recall  f1-score   support

     Class-0       0.55      0.88      0.68       105
     Class-1       0.78      0.38      0.51       120

   micro avg       0.61      0.61      0.61       225
   macro avg       0.66      0.63      0.59       225
```

```
weighted avg       0.67      0.61      0.59       225

################################

################################

Classification report on test dataset

              precision    recall  f1-score   support

     Class-0       0.64      0.96      0.77        45
     Class-1       0.75      0.20      0.32        30

   micro avg       0.65      0.65      0.65        75
   macro avg       0.70      0.58      0.54        75
weighted avg       0.69      0.65      0.59        75

################################
```

并得到截图：