Experiement 2

孟昶-2021214431

November 9, 2021

Task:

Use the sklearn classification models to finish the experiment.

Goal:

We hope you know better about the classification models and try to use the models better on the given dataset.

Data:

For CS background students, you should use the attached dataset file.

For Non CS background students, you can use the sklearn package to load the MNIST dataset.

Divide the data into train set and test set.

Experiment Step:

- Use the SVM to classify the data. Try to change kernel and other related parameters to see the change of metric you use. Record the result and try to explain it.
- Use the Decision Tree to classify the data, use the API to visualize the tree. Try to change the related parameters to see the change of metric you use. Record the result and try to explain it.
- Use the KMeans to cluster the data and visualize it. (The PCA may not be covered in the lecture currently, just use it.)

Answer:

1.SVM

代码：

```
from sklearn.svm import SVC
import numpy as np
from numpy.lib.function_base import select
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score,mean_absolute_error,mean_squared_error
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import KFold
from sklearn.metrics import accuracy_score

if __name__ == '__main__':
    data = pd.read_csv('Data.csv')
    X_train,X_test,y_train,y_test = train_test_split(data.iloc[:,:-1],data['fetal_health'],
        test_size=0.1,random_state=666)
    ss = StandardScaler()
    ss.fit(X_train)
    X_train = ss.transform(X_train)
    X_test = ss.transform(X_test)
    kernels = ['linear', 'poly', 'rbf', 'sigmoid']
    Cs = [0.01,1,5]
    for i in kernels:
        for j in Cs:
            p1 = SVC(C = j,kernel= i )
            p1.fit(X_train,y_train)
            print(i,'C = ',j,'score = ',p1.score(X_test,y_test))

    X_train = np.array(X_train)
    y_train = np.array(y_train)
    X_test = np.array(X_test)
    y_test = np.array(y_test)
    y_train = y_train.ravel()
    y_test = y_test.ravel()
    kf = KFold(n_splits=5)

    num = 0
    for j in kernels:
        for i in Cs:
            cls = SVC(C = i,kernel=j)
            t = 1
            for train_index, test_index in kf.split(X_train,y_train):

                X_train_1, X_test_1 = X_train[train_index], X_train[test_index]
                Y_train_1, Y_test_1 = y_train[train_index], y_train[test_index]
                cls.fit(X_train_1, Y_train_1)
                y_pred = cls.predict(X_test_1)
                Y_test = Y_test_1
                print('第',t,'次:',j,i)
                print(accuracy_score(Y_test, y_pred))
                # print ("test_Accuracy:", accuracy)
                # print ("test_error_rate:", 1-accuracy)
                t += 1
```

Table 1: SVM 测试集上的得分和交叉验证得分表

kernel type	**num of C**	**test score**	**cross-validation score**				
linear			1st	2nd	3rd	4th	5th
	0.01	0.901	0.877	0.896	0.846	0.874	0.898
	1	0.915	0.896	0.888	0.872	0.901	0.921
	5	0.920	0.901	0.877	0.872	0.903	0.911
polynomial	0.01	0.850	0.781	0.836	0.809	0.801	0.825
	1	0.939	0.906	0.901	0.867	0.893	0.906
	5	0.944	0.924	0.906	0.880	0.901	0.927
Gaussian	0.01	0.826	0.742	0.804	0.781	0.775	0.764
	1	0.925	0.903	0.901	0.885	0.898	0.919
	5	0.948	0.916	0.911	0.901	0.919	0.935
sigmoid	0.01	0.826	0.742	0.804	0.781	0.775	0.764
	1	0.817	0.789	0.794	0.7395	0.764	0.764
	5	0.803	0.765	0.778	0.744	0.770	0.772

惩罚系数 C 是其中最重要的参数：C 越大则对错误分类样本的惩罚力度越大，因此训练集准确率更高，但是泛化能力会有所降低即测试集分类准确率降低；C 越小则对错误分类样本的惩罚力度越小，泛化能力有所增强。因此若出现过拟合现象，则应该适当减小 C，反之出现欠拟合现象则应该适当增大 C。

从表中可以看出，C 太小的话会导致训练出现欠拟合现象，导致精度下降，从而使得训练集与测试集错误率都较高；而 C 偏大的话可能会出现过拟合的情况，从 Gaussian 和 polynomial 中的结果就可以看出。从使用的 kernel 上可以看出，Gaussian 与 polynomial 核函数在训练集上错误率较低，而 Linear 在测试集上表现最好，其测试集错误率最低，具有较好的泛化能力。

2.Decision Tree

代码：

```
from sklearn.tree import DecisionTreeClassifier
import numpy as np
from numpy.lib.function_base import select
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score,mean_absolute_error,mean_squared_error
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import KFold
from sklearn.metrics import accuracy_score

if __name__ == '__main__':
    data = pd.read_csv('Data.csv')
    X_train,X_test,y_train,y_test = train_test_split(data.iloc[:,:-1],data['fetal_health'],
        test_size=0.1,random_state=666)
    feature_names = X_train.keys()
    class_names = ['1.0','2.0','3.0']
    ss = StandardScaler()
    ss.fit(X_train)
    X_train = ss.transform(X_train)
    X_test = ss.transform(X_test)
    kernels = ["gini", "entropy"]
    Cs = [0,0.015,0.030,0.045]
    for i in kernels:
        for j in Cs:
            p1 = DecisionTreeClassifier(criterion = i,ccp_alpha = j)
            p1.fit(X_train,y_train)
            print(i,'C = ',j,'score = ',p1.score(X_test,y_test))


    X_train = np.array(X_train)
    y_train = np.array(y_train)
    X_test = np.array(X_test)
    y_test = np.array(y_test)
    y_train = y_train.ravel()
    y_test = y_test.ravel()
    kf = KFold(n_splits=3)

    num = 0
    for j in kernels:
        for i in Cs:
            cls = DecisionTreeClassifier(criterion = j,ccp_alpha = i)
            t = 1
            for train_index, test_index in kf.split(X_train,y_train):

                X_train_1, X_test_1 = X_train[train_index], X_train[test_index]
                Y_train_1, Y_test_1 = y_train[train_index], y_train[test_index]
                cls.fit(X_train_1, Y_train_1)
                y_pred = cls.predict(X_test_1)
                Y_test = Y_test_1
                print('第',t,'次:',j,i)
                print(accuracy_score(Y_test, y_pred))

                t += 1
```

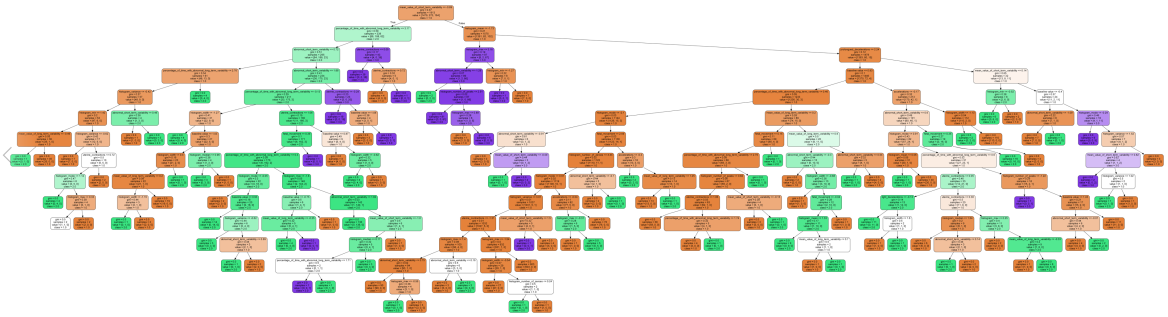

Figure 1: 决策树可视化结果图

Table 2: 决策树测试集上的得分和交叉验证得分表

criterion	**ccp_alpha**	**test score**	**cross-validation score**		
			1st	**2nd**	**3rd**
gini	0	0.892	0.917	0.918	0.922
	0.015	0.883	0.901	0.901	0.926
	0.030	0.836	0.862	0.856	0.879
	0.045	0.836	0.862	0.856	0.879
entropy	0	0.915	0.936	0.923	0.928
	0.015	0.906	0.928	0.925	0.917
	0.030	0.887	0.917	0.909	0.900
	0.045	0.883	0.879	0.900	0.895

决策树学习是一种采用树状结构的有监督机器学习方法。决策树是一个预测模型，表示对象特征和对象值之间的一种映射。其原理通俗的说，即给定一个输入值，从树节点不断往下走，直至走到叶节点，这个叶节点就是对输入值的一个预测或者分类。Sklearn 中实现了其三种算法 ID3、C4.5、CART，最常用的 CART 算法其核心公式就是基尼指数的计算，基尼指数越大不确定越大，基尼指数的计算公式为：

$$Gini\,(D) = 1 - \sum p_i^2$$

其中 p_i 是 D 中元组 C_i 类的概率。

criterion:"gini", "entropy",default="gini": 特征选择标准。

gini 原理：当一个 node p 被分为 k 部分，连续型的属性分割，先把数据有序排列，然后从左到右每个都是一遍，找 gini 最小的那一个。

entropy 原理：log 以 2 为底。当所有分类都一样是 1/k 的时候，熵值最大，是 log（nc）；所有值都属于同一类的时候，熵最小，是 0

ccp_alpha：non-negative float, default=0.0：复杂度参数。用于最小成本复杂度修剪。

随着剪枝的程度越大，会有效抑制过拟合现象，但是当剪枝的程度过大时，模型无法有效拟合，从而使得精度下降。

3.KMeans

代码：

```
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
import numpy as np
from numpy.lib.function_base import select
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import KFold
from sklearn.metrics import accuracy_score
from time import time
from sklearn import metrics
from sklearn.pipeline import make_pipeline
import matplotlib.pyplot as plt

def bench_k_means(kmeans, name, data, labels):
    t0 = time()
    estimator = make_pipeline(StandardScaler(), kmeans).fit(data)
    fit_time = time() - t0
    results = [name, fit_time, estimator[-1].inertia_]

    # Define the metrics which require only the true labels and estimator
    # labels
    clustering_metrics = [
        metrics.homogeneity_score,
        metrics.completeness_score,
        metrics.v_measure_score,
        metrics.adjusted_rand_score,
        metrics.adjusted_mutual_info_score,
    ]
    results += [m(labels, estimator[-1].labels_) for m in clustering_metrics]

    # The silhouette score requires the full dataset
    results += [
        metrics.silhouette_score(
            data,
            estimator[-1].labels_,
            metric="euclidean",
            sample_size=300,
        )
    ]

    # Show the results
    formatter_result = (
        "{:9s}\t{:.3f}s\t{:.0f}\t{:.3f}\t{:.3f}\t{:.3f}\t{:.3f}\t{:.3f}"
    )
    print(formatter_result.format(*results))




if __name__ == '__main__':
    data = pd.read_csv('Data.csv')
    X_train,X_test,y_train,y_test = train_test_split(data.iloc[:,:-1],data['fetal_health'],
        test_size=0.1,random_state=666)
    feature_names = X_train.keys()
```

```
    class_names = ['1.0','2.0','3.0']
    ss = StandardScaler()
    ss.fit(X_train)
    X_train = ss.transform(X_train)
    X_test = ss.transform(X_test)

    (n_samples, n_features), n_digits = X_train.shape, np.unique(y_train).size
    print(82 * "_")
    print("init\t\ttime\tinertia\thomo\tcompl\tv-meas\tARI\tAMI\tsilhouette")
    kmeans = KMeans(init="k-means++", n_clusters=n_digits, n_init=4, random_state=0)
    bench_k_means(kmeans=kmeans, name="k-means++", data=X_train, labels=y_train)

    kmeans = KMeans(init="random", n_clusters=n_digits, n_init=4, random_state=0)
    bench_k_means(kmeans=kmeans, name="random", data=X_train, labels=y_train)

    pca = PCA(n_components=n_digits).fit(X_train)
    kmeans = KMeans(init=pca.components_, n_clusters=n_digits, n_init=1)
    bench_k_means(kmeans=kmeans, name="PCA-based", data=X_train, labels=y_train)
    print(82 * "_")



    # 画图
    reduced_data = PCA(n_components=2).fit_transform(X_train)
    kmeans = KMeans(init="k-means++", n_clusters=n_digits, n_init=4)
    kmeans.fit(reduced_data)

    # Step size of the mesh. Decrease to increase the quality of the VQ.
    h = 0.02  # point in the mesh [x_min, x_max]x[y_min, y_max].

    # Plot the decision boundary. For that, we will assign a color to each
    x_min, x_max = reduced_data[:, 0].min() - 1, reduced_data[:, 0].max() + 1
    y_min, y_max = reduced_data[:, 1].min() - 1, reduced_data[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))

    # Obtain labels for each point in mesh. Use last trained model.
    Z = kmeans.predict(np.c_[xx.ravel(), yy.ravel()])

    # Put the result into a color plot
    Z = Z.reshape(xx.shape)
    plt.figure(1)
    plt.clf()
    plt.imshow(
        Z,
        interpolation="nearest",
        extent=(xx.min(), xx.max(), yy.min(), yy.max()),
        cmap=plt.cm.Paired,
        aspect="auto",
        origin="lower",
    )

    plt.plot(reduced_data[:, 0], reduced_data[:, 1], "k.", markersize=2)
    # Plot the centroids as a white X
    centroids = kmeans.cluster_centers_
    plt.scatter(
        centroids[:, 0],
        centroids[:, 1],
        marker="x",
```

```
        s=169,
        linewidths=3,
        color="w",
        zorder=10,
    )
    plt.title(
        "K-means clustering on the excal dataset (PCA-reduced data)"
    )
    plt.xlim(x_min, x_max)
    plt.ylim(y_min, y_max)
    plt.xticks(())
    plt.yticks(())
    plt.show()
```

如图所示，为结果图

init	time	inertia	homo	compl	v-meas	ARI	AMI	silhouette
k-means++	0.048s	29011	0.248	0.179	0.208	0.179	0.207	0.163
random	0.036s	29005	0.248	0.178	0.207	0.205	0.206	0.162
PCA-based	0.020s	29012	0.240	0.171	0.200	0.170	0.199	0.157

Figure 2: KMeans 各指标结果图

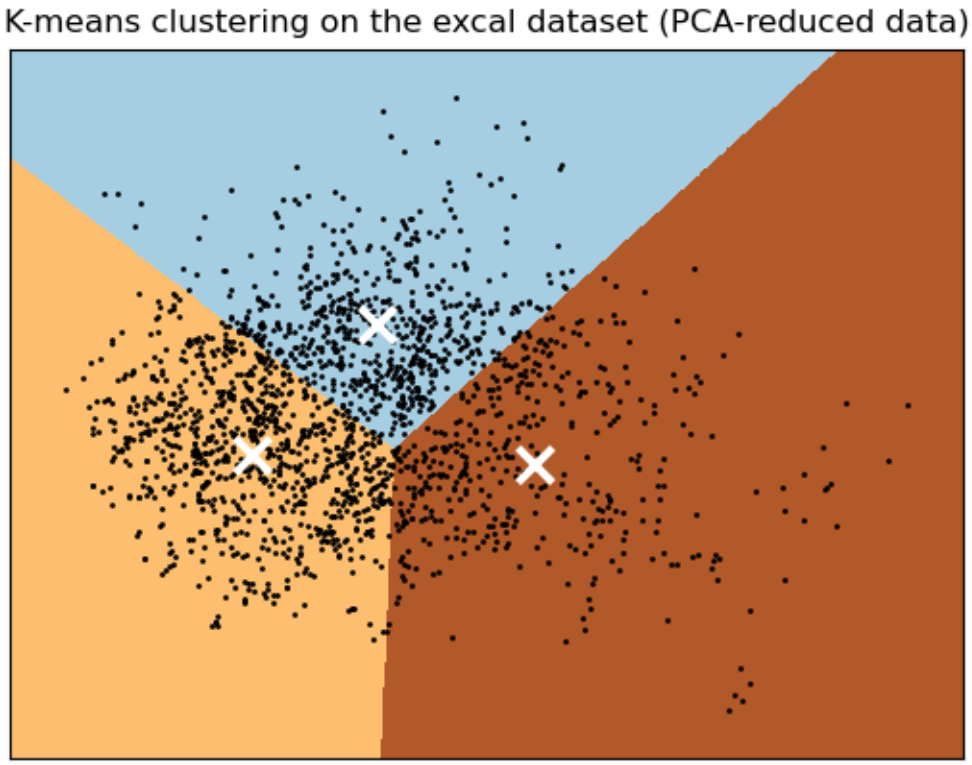

Figure 3: KMeans 分类结果图