k-NN

一. 实验要求：

a) 1) Develop a k-NN classifier with Euclidean distance and simple voting

b) 2) Perform 5-fold cross validation, find out which k performs the best (in terms of accuracy)

c) 3) Use PCA to reduce the dimensionality, then perform 2) again. Does PCA improve the accuracy? Which dimensionality is better?

二. Plus (at most 3/10)

a) Try other distance metrics or distance-based voting (+0.5~1)

b) Try other dimensionality reduction methods (+1~2)

c) Perform feature selection (+1)

How to set the k value, if not using cross validation? Verify your idea (+3)

二. 实验步骤：

要求 1:采用欧氏距离来并编写一个 k-nn 分类器。

首先我们先把数据加载进去，并且可视化数据,这里我们是采用了 read_csv 函数先把数据读出来，然后在使用 descirbe 函数去统计每个特征集的各个值。

具体代码如下：

```
import math
import pandas

from sklearn.preprocessing import PolynomialFeatures
import matplotlib.pyplot as plt
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
url="wine.data"
names=['Label','A1','A2','A3','A4','A5','A6','A7','A8','A9','A10','A11','A12','A13']
dateset=pandas.read_csv(url,names=names)
origindata=dateset.iloc[range(0,178),range(1,14)]
print(origindata.describe())
```

实验结果:

```
                A1          A2      ...             A12          A13
count  178.000000  178.000000      ...      178.000000   178.000000
mean    13.000618    2.336348      ...        2.611685   746.893258
std      0.811827    1.117146      ...        0.709990   314.907474
min     11.030000    0.740000      ...        1.270000   278.000000
25%     12.362500    1.602500      ...        1.937500   500.500000
50%     13.050000    1.865000      ...        2.780000   673.500000
75%     13.677500    3.082500      ...        3.170000   985.000000
max     14.830000    5.800000      ...        4.000000  1680.000000
```

采用欧式距离做简单的 knn 选择。具体代码如下：

```python
##knn 欧氏距离
def euclideanKnn(x_train,x_test,y_train,y_test):
    knn = KNeighborsClassifier(n_neighbors=3, metric='euclidean')
    knn.fit(x_train, y_train)
    print(knn.score(x_test, y_test))
```

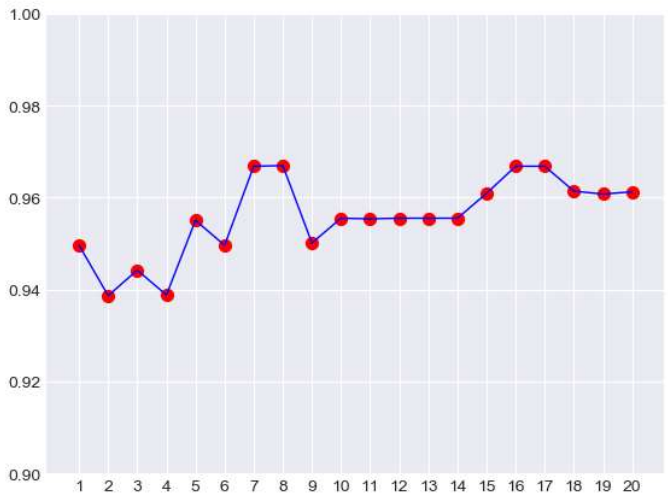
实验结果： 分类准确率为 1.0

要求 2：执行 5 折交叉验证，找出在准确性方面表现最好的 k

具体实现代码：

```python
def fiveFolder(x,y):
    k_rane=[i+1 for i in range (20)]
    k_ranges=[str(i) for i in k_rane]
    print(k_rane)
    scores=[]
    for k in k_rane:
        knn=KNeighborsClassifier(n_neighbors=k)
        score=cross_val_score(knn,x,y,cv=5,scoring='accuracy')
        scores.append(score.mean())
    print(scores)
    print(max(scores))
    print(scores.index(max(scores)))
    pltScores(k_rane,k_ranges,scores)
##画图
def pltScores(k_range,k_ranges,scores):
    plt.style.use('seaborn-darkgrid')
    fig=plt.figure()
    scoreImg=fig.add_axes([0.11,0.1,0.8,0.8])
    plt.ylim(0.9,1)
    plt.xticks(k_range,k_ranges)
    scoreImg.plot(k_range,scores,color='b',marker='',lw=1)
    scoreImg.scatter(k_range,scores,color='r',s=50)
    plt.show()
```

实验结果：从实验结果可以看出 k 值在 8 的的时候最大，分类的准确率最高，所以我们使用 k 值为 8，准确率为：0.966940469。



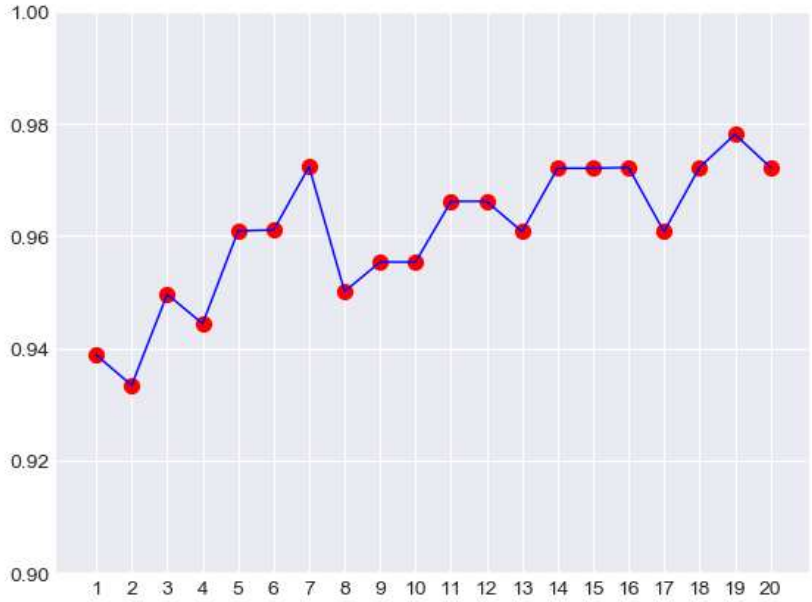
要求 3：采用 PCA 降维

首先采用 pca 降维，然后重复要求二。得到最大的值。

代码如下：

```python
pca=PCA(n_components=6)
x_pca=pca.fit_transform(x)
fiveFolder(x_pca,y)
```

实验结果如下：从实验结果可以看出来在采用 pca 进行降维了以后我们的准确率发生了变化，现在在 k 取 19 的时候最大，准确率为 0.9780780708.



要求 4：采用不同距离的 KNN

这里我们另外选取两种距离方式，一种曼哈顿距离，一种切尔雪夫距离；

具体代码：

```
def otherDistanceKNN(x_train,x_test,y_train,y_test):
    knn = KNeighborsClassifier(n_neighbors=3, metric='manhattan')
    knn.fit(x_train, y_train)
    print(knn.score(x_test, y_test))
    knn = KNeighborsClassifier(n_neighbors=3, metric='chebyshev')
    knn.fit(x_train, y_train)
    print(knn.score(x_test, y_test))
```

实验结果：从实验结果中我们可以看出来，曼哈顿距离的准确率为 1.0.切尔雪夫的距离为 0.93333。

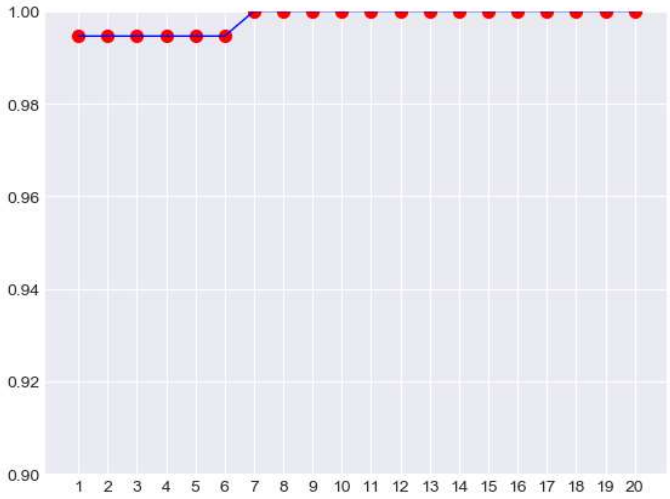
```
1.0
0.9333333333333333
```

要求 5：采用 LDA 进行降维：

具体代码如下：

```
'''
lda=LinearDiscriminantAnalysis();
x_lda=lda.fit_transform(x,y)
fiveFolder(x_lda, y)
```

实验结果：从实验结果可以看出来，使用 LDA 降维比 PCA 的效果好许多。



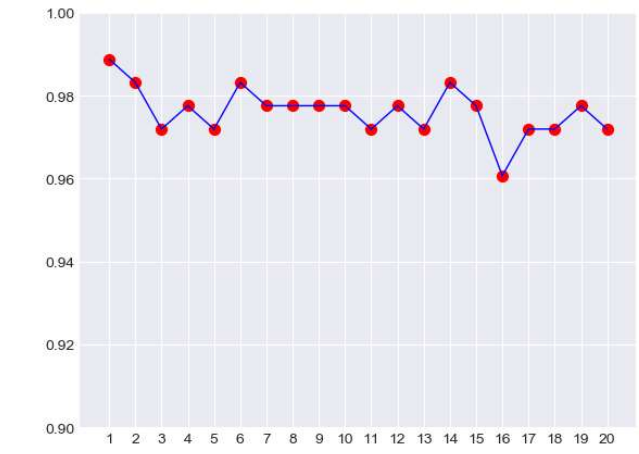
要求 5：使用 BAGGING 方法寻找最佳 k 值

除了 K 折交叉验证法，我们还可以利用委员会算法寻找最佳 K 值。Bagging 是其代表，其基本思想是 i.对一个单独数据集，利用 bootstrap（放回抽样）产生 M 个自助数据集。ii.对每一个 K 值，我们利用这 M 个自助数据集训练 M 个独立模型 iii.对每一个实例，委员会预测为 M 个独立模型的平均值。

具体代码如下：

```
##使用BAGGING方法寻找最佳k值
def baggingKNN(x,y):
    k_rane = [i+1 for i in range(20)]
    k_ranges = [str(i) for i in k_rane]
    scores = []
    for k in k_rane:
        bagging=BaggingClassifier(KNeighborsClassifier(n_neighbors=k),
                                  max_samples=0.9,n_estimators=11,bootstrap=True)
        bagging.fit(x,y)
        scores.append(bagging.score(x,y))
    print(scores)
    print(max(scores))
    print(scores.index(max(scores)))
    pltScores(k_rane,k_ranges,scores)
```

实验结果：我们发现 K=1 时，取到最高准确率。这是因为 K=1 时，模型方差很大，但是加和求平均后偏置很小，从而产生了预测效果的提升。利用 bagging 算法，我们选取 K=6,此时准确率为 0.983



.参考文献

[1]李航 统计学习方法