实验二　KNN&多项式分类器的设计及应用实验

实验目标：理解 KNN\多项式分类器的原理；

能独立实现 KNN\多项式类器的设计；

准确评估分类器精度。

实验工具：Python(推荐) 或 C/C++

实验步骤：

PART1: KNN 分类器的构造

一、KNN 算法的思路:

存在一个样本数据集合 ,称为训练样本集 ,且样本集中每个数据都存在标签，即样本集中每一数据与所属分类的对应关系。

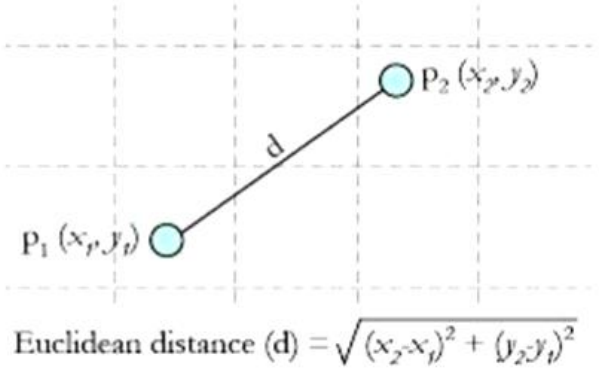
输入没有标签的数据后 ,将新数据中的每个特征与样本集中数据对应的特征进行比较，提取出样本集中特征最相似数据（最近邻）的分类标签。选择 k 个最相似数据中出现次数最多的分类作为新数据的分类。

二、算法步骤：

1. 计算未知实例到所有已知实例的距离；
2. 选择参数 K；
3. 根据多数表决(majority-voting)规则 ,将未知实例归类为样本中最多数的类别。

- **距离的衡量方法**

欧拉距离 这种测量方式就是简单的平面几何中两点之间的直线距离。



上述方法延伸至三维或更多维的情况，总结公式为:

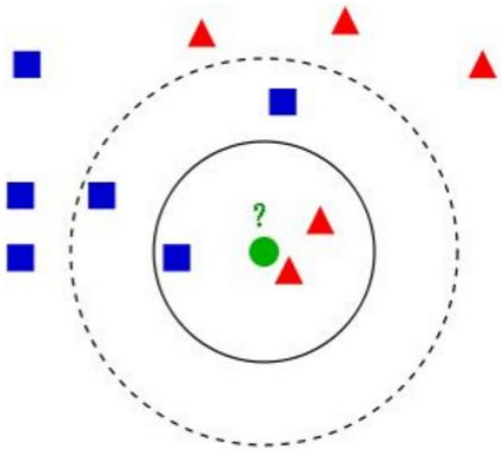
$$d(x,y)=\sqrt{\sum_{i=0}^{n}(x_i-y_i)^2}$$

曼哈顿距离 ，街区的距离。

$$d(x,y)=\sum_{i=0}^{n}|x_i-y_i|$$

- **K 值的选择**

K 值的选择会影响结果，如下图:



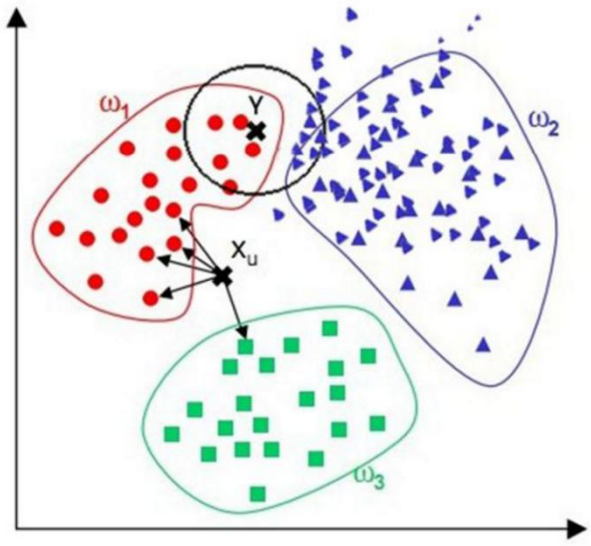
图中的数据集都打好了 label ，一类是蓝色正方形，一类是红色三角形，绿色圆形是待分类的数据。

K = 3 时，范围内红色三角形多，这个待分类点属于红色三角形。

K = 5 时，范围内蓝色正方形多，这个待分类点属于蓝色正方形。

如何选择一个最佳的 K 值取决于数据。一般，较大 K 值能减小噪声的影响，但会使类别之间的界限变得模糊。因此 K 的取值一般比较小（K < 20）。

➢ **改进：**



在点 Y 的预测中，范围内三角形类数量占优，因此将 Y 点归为三角形。但从视觉上观测，分为圆形类更合理。根据这种情况，可以在距离测量中加入权重，如 1/d (d: 距离)。

三、实验步骤：

训练集是 sklearn 中关于 Iris 的数据。

数据载入

```
from sklearn import datasets
iris = datasets.load_iris()
```

数据存储在 .data 成员中，是一个 (n_samples, n_features) numpy 数组：

```
        print(iris.data)
# [[ 5.1  3.5  1.4  0.2]
#  [ 4.9  3.   1.4  0.2]
#  ...
```

四个特征 :萼片长度 ,萼片宽度 ,花瓣长度 ,花瓣宽度 (sepal length, sepal width, petal length and petal width)。

```
print iris.data.shape
# output:(150L, 4L)
```

品种分类有山鸢尾，变色鸢尾，菖蒲锦葵（Iris setosa, Iris versicolor, Iris virginica. ）三种。

构造 KNN 分类器

完整参考代码如下：

```
from __future__ import print_function
import sys
import os
import math
import numpy as np
from sklearn import datasets
import matplotlib.pyplot as plt
from collections import Counter
from sklearn.datasets import make_classification
%matplotlib inline

def shuffle_data(X, y, seed=None):
    if seed:
        np.random.seed(seed)

    idx = np.arange(X.shape[0])
```

```
    np.random.shuffle(idx)

    return X[idx], y[idx]

# 正规化数据集 X
def normalize(X, axis=-1, p=2):
    lp_norm = np.atleast_1d(np.linalg.norm(X, p, axis))
    lp_norm[lp_norm == 0] = 1
    return X / np.expand_dims(lp_norm, axis)

# 标准化数据集 X
def standardize(X):
    X_std = np.zeros(X.shape)
    mean = X.mean(axis=0)
    std = X.std(axis=0)

    # 分母不能等于0 的情形
    # X_std = (X - X.mean(axis=0)) / X.std(axis=0)
    for col in range(np.shape(X)[1]):
        if std[col]:
            X_std[:, col] = (X_std[:, col] - mean[col]) / std[col]

    return X_std

# 划分数据集为训练集和测试集
def train_test_split(X, y, test_size=0.2, shuffle=True, seed=None):
    if shuffle:
        X, y = shuffle_data(X, y, seed)

    n_train_samples = int(X.shape[0] * (1-test_size))
    x_train, x_test = X[:n_train_samples], X[n_train_samples:]
    y_train, y_test = y[:n_train_samples], y[n_train_samples:]

    return x_train, x_test, y_train, y_test

def accuracy(y, y_pred):
    y = y.reshape(y.shape[0], -1)
    y_pred = y_pred.reshape(y_pred.shape[0], -1)
    return np.sum(y == y_pred)/len(y)

class KNN():
    """ K 近邻分类算法.
    Parameters:
    -----------
```

```
    k: int
        最近邻个数.
      """
    def __init__(self, k=5):
        self.k = k

    # 计算一个样本与训练集中所有样本的欧氏距离的平方
    def euclidean_distance(self, one_sample, X_train):
        one_sample = one_sample.reshape(1, -1)
        X_train = X_train.reshape(X_train.shape[0], -1)
        distances = np.power(np.tile(one_sample, (X_train.shape[0], 1)) - X_train,
2).sum(axis=1)
        return distances

    # 获取 k 个近邻的类别标签
    def get_k_neighbor_labels(self, distances, y_train, k):
        k_neighbor_labels = []
        for distance in np.sort(distances)[:k]:

            label = y_train[distances==distance]
            k_neighbor_labels.append(label)

        return np.array(k_neighbor_labels).reshape(-1, )

    # 进行标签统计，得票最多的标签就是该测试样本的预测标签
    def vote(self, one_sample, X_train, y_train, k):
        distances = self.euclidean_distance(one_sample, X_train)
        #print(distances.shape)
        y_train = y_train.reshape(y_train.shape[0], 1)
        k_neighbor_labels = self.get_k_neighbor_labels(distances, y_train, k)
        #print(k_neighbor_labels.shape)
        find_label, find_count = 0, 0
        for label, count in Counter(k_neighbor_labels).items():
            if count > find_count:
                find_count = count
                find_label = label
        return find_label

    # 对测试集进行预测
    def predict(self, X_test, X_train, y_train):
        y_pred = []
        for sample in X_test:
            label = self.vote(sample, X_train, y_train, self.k)
            y_pred.append(label)
```

```
        #print(y_pred)
        return np.array(y_pred)

def main():
    data = make_classification(n_samples=200, n_features=4, n_informative=2,
                               n_redundant=2, n_repeated=0, n_classes=2)
    X, y = data[0], data[1]
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33,
shuffle=True)
    clf = KNN(k=5)
    y_pred = clf.predict(X_test, X_train, y_train)

    accu = accuracy(y_test, y_pred)
    print ("Accuracy:", accu)

if __name__ == "__main__":
    main()
```

PART2: 感知机模型的构造

一、感知机原理

感知机是一个二类分类的线性分类器，是支持向量机和神经网络的基础。假设数据是线性可分的，目标是通过梯度下降法，极小化损失函数，最后找到一个分割超平面，可以将数据划分成两个类别。

决策函数：

$$f(x) = sign(wx + b)$$

对 n 维来说，线性方程 ω·x+b=0 对应特征空间的一个超平面，其中ω是超平面的法向量，b 是超平面的截距。ω·x 即各项对应相乘后求和。

要建立感知机模型，只要确定参数ω 和 b。通过梯度下降法，不断调整两个参数，向最优解靠近。

损失函数：函数值越小，说明离最好的模型越近。

$$L(\omega,b)=-\sum_{x_i\in M}y_i(\omega\bullet x_i+b)$$

其中（x_i,y_i)为分类错误的一个样例，M 为所有分类错误样例的集合。

梯度下降：

要极小化损失函数

$$\min_{\omega,b}L(\omega,b)=-\sum_{x_i\in M}y_i(\omega\cdot x_i+b)$$

分别对ω和 b 求偏导

$$\begin{cases}\nabla_\omega L(\omega\ \ b)=-\sum_{x_i\in M}y_i x_i\\ \nabla_b L(\omega\ \ b)=-\sum_{x_i\in M}y_i\end{cases}$$

权重更新公式：

$$\begin{cases}\omega\leftarrow\omega+\eta y_i x_i\\ b\leftarrow b+\eta y_i\end{cases}$$

对于每个样本（x_i,y_i)，若 $y_i(\omega\cdot x_i+b)\le 0$，则用梯度下降更新权重 ω,b。直到遍历整个样本都没有误分点，算法停止。

二、参考代码如下：

```
# -*- coding: utf-8 -*-
"""
Created on Sat Mar 19 13:49:04 2016

@author: fengxinhe
"""
import copy
```

```
8.    from matplotlib import pyplot as pl
9.    from matplotlib import animation as ani
10.
11.   w=[0,0]  #weight vector
12.   b=0      #bias
13.   yita=0.5   #learning rate
14.   data=[[(1,4),1],[(0.5,2),1],[(2,2.3), 1], [(1, 0.5), -
   1], [(2, 1), -1],[(4,1),-1],[(3.5,4),1],[(3,2.2),-1]]
15.   #data=[[(3, 3), 1], [(4, 3), 1], [(1, 1), -1]]
16.   record=[]
17.
18.   """
19.   if y(wx+b)<=0,return false; else, return true
20.   """
21.   def sign(vec):
22.       global w,b
23.       res=0
24.       res=vec[1]*(w[0]*vec[0][0]+w[1]*vec[0][1]+b)
25.       if res>0: return 1
26.       else: return -1
27.
28.   """
29.   update the paramaters w&b
30.   """
31.   def update(vec):
32.       global w,b,record
33.       w[0]=w[0]+yita*vec[1]*vec[0][0]
34.       w[1]=w[1]+yita*vec[1]*vec[0][1]
35.       b=b+yita*vec[1]
36.       record.append([copy.copy(w),b])
37.
38.   """
39.   check and calculate the whole data one time
40.   if all of the input data can be classfied correctly at one tim
   e,
41.   we get the answer
```

```
42.  """
43.  def perceptron():
44.      count=1
45.      for ele in data:
46.          flag=sign(ele)
47.          if not flag>0:
48.              count=1
49.              update(ele)
50.          else:
51.              count+=1
52.      if count>=len(data):
53.          return 1
54.
55.
56.
57.  if __name__ == "__main__":
58.    while 1:
59.         if perceptron() > 0:
60.              break
61.    print record
62.  x1=[]
63.  y1=[]
64.  x2=[]
65.  y2=[]
66.
67.  #display the animation of the line change in searching process
68.  fig = pl.figure()
69.  ax = pl.axes(xlim=(-1, 5), ylim=(-1, 5))
70.  line,=ax.plot([],[],'g',lw=2)
71.
72.  def init():
73.      line.set_data([],[])
74.      for p in data:
75.          if p[1]>0:
76.              x1.append(p[0][0])
```

```
77.                 y1.append(p[0][1])
78.             else:
79.                 x2.append(p[0][0])
80.                 y2.append(p[0][1])
81.         pl.plot(x1,y1,'or')
82.         pl.plot(x2,y2,'ob')
83.         return line,
84.
85.
86.     def animate(i):
87.         global record,ax,line
88.         w=record[i][0]
89.         b=record[i][1]
90.         x1=-5
91.         y1=-(b+w[0]*x1)/w[1]
92.         x2=6
93.         y2=-(b+w[0]*x2)/w[1]
94.         line.set_data([x1,x2],[y1,y2])
95.         return line,
96.
97.     animat=ani.FuncAnimation(fig,animate,init_func=init,frames=len(re
    cord),interval=1000,repeat=True,
98.                                         blit=True)
99.     pl.show()
100.    animat.save('/Users/dingpeien/perceptron.gif', fps=2,writer='imag
    emagick')
101.
102.
```

实验要求：

- 理解 KNN 分类器，参考文档中代码，以 sklearn 中的 Iris 数据集作为训练集，用 KNN 算法实现分类，变化 K 的取值，统计分类结果。
- 理解感知机的参考程序，用 adult.data 数据集，实现二分类，并统计分类预

测精度。

实验报告要求：

- 本次实验课上检查结果。
- 有课上检查不通过的同学，顺延到下次实验课检查。

本次实验参考网址：

https://blog.csdn.net/zhangxb35/article/details/48615661

https://blog.csdn.net/Dream_angel_Z/article/details/48915561

https://zhuanlan.zhihu.com/p/30210438