第6讲　最近邻算法

- kNN原理
- kNN实例
- kNN应用
- 实践：Python

2019-04-03

WangBianqin, Public Laboratory & Teaching Center, Guangzhou East Campus, Sun Yat-sen University

kNN原理

- 已知一个样本数据集合，也称为训练样本集，并且每个数据都存在标签，即已知样本集中每一个数据与所属类别的对应关系。
- 输入没有标签的新数据后，将新数据的每个特征与样本集中数据对应的特征进行比较，然后提取样本集中特征最相似数据（最近邻）的分类标签。

2019-04-03

Distance Measure

2019-04-03

$d(A,B) = d(B,A)$ Symmetry

$d(A,A) = 0$ Constancy of Self-Similarity

$d(A,B) = 0$ iff $A=B$ Positivity Separation

$d(A,B) \leq d(A,C)+d(B,C)$ Triangular Inequality

Distance Measure

Minkowski Distance

$$dist(\mathbf{x},\mathbf{y}) = \sqrt[r]{\sum_{i=1}^{d} |x_i - y_i|^r}$$

Euclidean distance (r=2)

$$dist(\mathbf{x},\mathbf{y}) = \sqrt{\sum_{k=1}^{d} (x_k - y_k)^2} = \|\mathbf{x}-\mathbf{y}\|_2$$

Manhattan distance (r=1)

$$dist(\mathbf{x},\mathbf{y}) = \sum_{k=1}^{d} |x_k - y_k| = \|\mathbf{x}-\mathbf{y}\|_1$$

Jaccard distance

$$sim(x_i, x_j) = \frac{\sum_{d=1}^{k} x_{id} x_{jd}}{\sum_{d=1}^{k} x_{id}^2 + \sum_{d=1}^{k} x_{jd}^2 - \sum_{d=1}^{k} x_{id} x_{jd}}$$

2019-04-03

Cosine

$$sim(x_i, x_j) = \frac{\sum_{d=1}^{k} x_{id} x_{jd}}{\sqrt{\sum_{d=1}^{k} x_{id}^2 \sum_{d=1}^{k} x_{jd}^2}}$$

数据规范化

- 数据规范化：将样本的特征缩放到某个指定的范围
- 消除不同特征具有不同量级的影响：

2019-04-03

数量级的差异将导致量级较大的特征占据主导地位

数量级的差异将导致迭代收敛速度减慢

依赖于样本距离的算法对于数据的数量级非常敏感

两种常用方法

- **min-max归一化**

$$X_{new} = \frac{X - \min}{\max - \min}$$

2019-04-03

- **z-score标准化**

$$X_{new} = \frac{X - u}{\sigma}$$

实现数据归一化

```
def autoNorm(dataSet):
    # 每一列最小值
    minVals = dataSet.min(0)
    # 每一列最大值
    maxVals = dataSet.max(0)
    # 每一列的差
    ranges = maxVals - minVals
    # 复制矩阵 dataSet的行，列。值都是0
    normDataSet = zeros(shape(dataSet))
    # 得到多少行
    m = dataSet.shape[0]

    # tile(minVals, (m,1)) 创建datSet的行列的矩阵，每一个值都是minVals。
    # dataSet - 矩阵相减
    normDataSet = dataSet - tile(minVals, (m,1))

    # tile(ranges, (m,1)) 创建datSet的行列的矩阵，每一个值都是ranges。
    # 并非矩阵除法，而是对应位置的值相除
    normDataSet = normDataSet / tile(ranges, (m,1))
    return normDataSet, ranges, minVals
```

2019-04-03

```
datingDataMat
```

```
array([[  4.09200000e+04,   8.32697600e+00,   9.53952000e-01],
       [  1.44880000e+04,   7.15346900e+00,   1.67390400e+00],
       [  2.60520000e+04,   1.44187100e+00,   8.05124000e-01],
       ...,
       [  2.65750000e+04,   1.06501020e+01,   8.66627000e-01],
       [  4.81110000e+04,   9.13452800e+00,   7.28045000e-01],
       [  4.37570000e+04,   7.88260100e+00,   1.33244600e+00]])
```

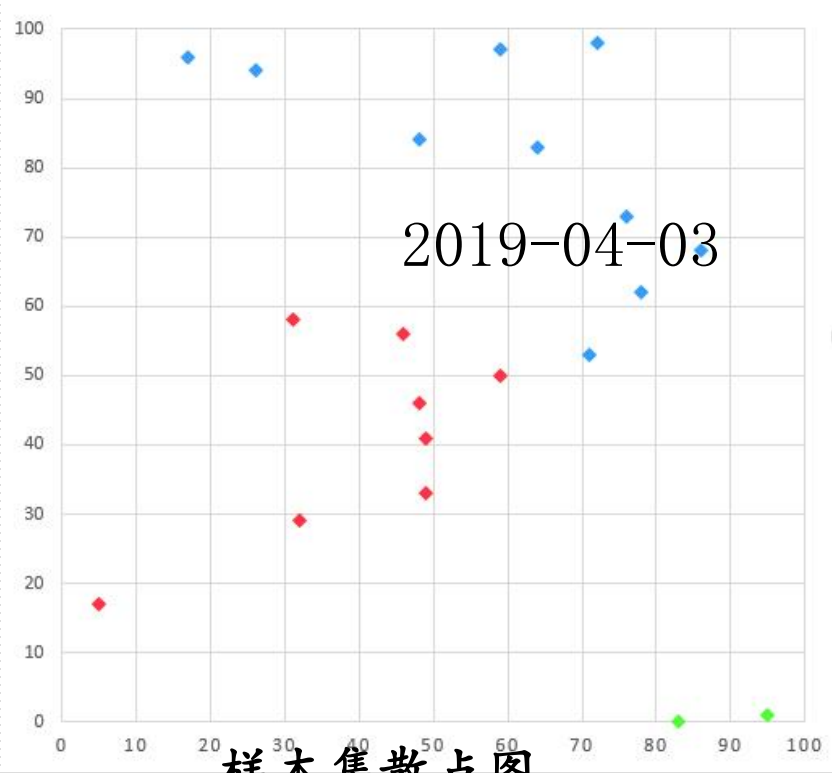
```
normMat, ranges, minVals = autoNorm(datingDataMat)
print(normMat)
print(ranges)
print(minVals)
```

```
[[ 0.44832535  0.39805139  0.56233353]
 [ 0.15873259  0.34195467  0.98724416]
 [ 0.28542943  0.06892523  0.47449629]
 ...,
 [ 0.29115949  0.50910294  0.51079493]
 [ 0.52711097  0.43665451  0.4290048 ]
 [ 0.47940793  0.3768091   0.78571804]]
[  9.12730000e+04   2.09193490e+01   1.69436100e+00]
[ 0.          0.          0.001156]
```

kNN实例

样本数据集

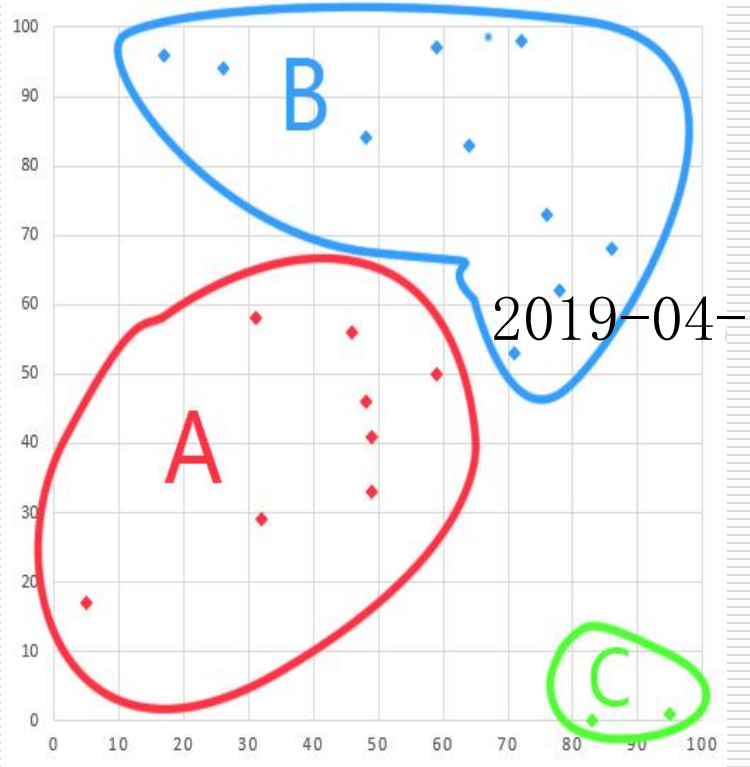
x	y
49	33
46	56
64	83
76	73
59	50
72	98
48	84
49	41
78	62
48	46
83	0
59	97
71	53
26	94
86	68
17	96
95	1
31	58
32	29
5	17



样本集散点图

kNN实例

x	y
49	33
46	56
64	83
76	73
59	50
72	98
48	84
49	41
78	62
48	46
83	0
59	97
71	53
26	94
86	68
17	96
95	1
31	58
32	29
5	17



2019-04-03

样本数据集分类为A、B、C三个类别

问题：通过20个已知类别的样本，对一个新数据(x=60, y=64)进行分类

kNN实例

x	y	距离
49	33	32.89377
46	56	16.12452
64	83	19.41649
76	73	18.35756
59	50	14.03567
72	98	36.05551
48	84	23.32381
49	41	25.4951
78	62	18.11077
48	46	21.63331
83	0	68.00735
59	97	33.01515
71	53	15.55635
26	94	45.34314
86	68	26.30589
17	96	53.60037
95	1	72.06941
31	58	29.61419
32	29	44.82187
5	17	72.34639

- 先求新数据与各样本数据间的距离

2019-04-03

kNN实例

x	y	距离
59	50	14.03567
71	53	15.55635
46	56	16.12452
78	62	18.11077
76	73	18.35756
64	83	19.41649
48	46	21.63331
48	84	23.32381
49	41	25.4951
86	68	26.30589
31	58	29.61419
49	33	32.89377
59	97	33.01515
72	98	36.05551
32	29	44.82187
26	94	45.34314
17	96	53.60037
83	0	68.00735
95	1	72.06941
5	17	72.34639

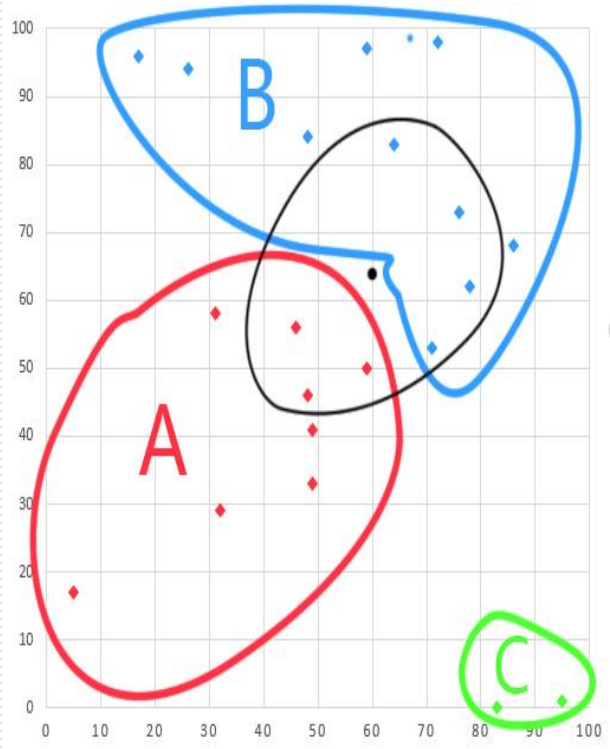
- 按照距离递增次序排序
- 选取与当前点距离最小的k个点

2019-04-03

k = 7

- 确定前k个点所属类别的出现概率

kNN实例



7个点中有4个属于B类，3个属于A类，即
P(A) = 3 / 20, P(B) = 4 / 20

2019-04-03

- **返回前k个点出现概率最高的类别作为当前点的预测分类。**

由此将新数据归为？类

实现kNN步骤

输入：训练样本，k为近邻数，未知数据点x

输出：x所属的类别

方法：

2019-04-03

（1）计算当前数据点与已知类别数据集中的每个点之间的距离；

（2）按照距离递增次序排序；

（3）选取与当前点距离最小的k个点；

（4）确定前k个点所属类别的出现概率；

（5）返回前k个点出现概率最高的类别作为当前点的预测分类。

sklearn构建kNN分类器

```
In  [50]: from sklearn.neighbors import KNeighborsClassifier
          knn = KNeighborsClassifier(n_neighbors=1)

In  [51]: knn.fit(X_train, y_train)   2019-04-03

Out[51]: KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                              metric_params=None, n_jobs=1, n_neighbors=1, p=2,
                              weights='uniform')
```

评估模型

```
In [55]: y_pred = knn.predict(X_test)
         print("Test set predictions:\n {}".format(y_pred))

Test set predictions:
 [2 1 0 2 0 2 0 1 1 1 2 1 1 1 1 0 1 1 0 0 2 1 0 0 2 0 0 1 1 0 2 1 0 2 2 1 0
 2]
```

2019-04-03

```
In [56]: print("Test set score: {:.2f}".format(np.mean(y_pred == y_test)))

Test set score: 0.97

In [59]: print("Test set score: {:.2f}".format(knn.score(X_test, y_test)))

Test set score: 0.97
```

模型预测

In [53]:
```
X_new = np.array([[5, 2.9, 1, 0.2]])
print("X_new.shape: {}".format(X_new.shape))
```

```
X_new.shape: (1, 4)
```

2019-04-03

In [54]:
```
prediction = knn.predict(X_new)
print("Prediction: {}".format(prediction))
print("Predicted target name: {}".format(
       iris_dataset['target_names'][prediction]))
```

```
Prediction: [0]
Predicted target name: ['setosa']
```

Sklearn定义模型

- 训练模型
  model.fit(X_train, y_train)
- 模型预测
  model.predict(X_test)
- 获取模型参数
  model.get_params()
- 模型评估
  model.score(data_X, data_y)
  线性回归：**R square**； 分类：**accuracy**

2019-04-03

Tf构建kNN分类器

In [5]:
```
# 导入相关库
import tensorflow as tf
import numpy as np

#直接从sklearn自带的datasets中导入iris
from sklearn.datasets import load_iris
iris = load_iris()

# 查看数据集
# print(iris.target)
# print(iris.feature_names)
# print(iris.data)
# print (iris.data.shape)

# 将数据集按照3:1分为训练集和测试集，每次分割的结果不同
from sklearn.model_selection import train_test_split
Xtrain, Xtest, Ytrain, Ytest = train_test_split(iris.data, iris.target,test_size = 0.25)

# 查看分割后的数据集
# print(Xtrain)
# print(Ytrain)
# print(Xtest)
# print(Ytest)
```

2019-04-03

定义模型

```
# 定义占位符：用于传入输入数据，形如[ 6.4  3.2  4.5  1.5]
x_train = tf.placeholder("float", [None, 4])
x_test = tf.placeholder("float", [4])

# 计算距离：
distance = tf.reduce_sum(tf.abs(tf.add(x_train, tf.negative(x_test))), axis=1)
```

2019-04-03

```
# 获得距离最小的index
pred = tf.argmin(distance, 0)

# 定义正确率标量
accuracy = 0

# 初始化变量
# init = tf.global_variables_initializer()
```

- 如果k不等于1，请修改程序代码

评估模型

```
with tf.Session() as sess:
    #sess.run(init)
    m = len(Xtest)
    for i in range(m):
        # index[0]为最小值所在的索引，index[1]为所有距离大小的列表，
        index = sess.run([pred, distance], feed_dict={x_train: Xtrain, x_test: Xtest[i,:]})

        # 预测值
        pred_label = Ytrain[index[0]]

        # 真值
        true_label = Ytest[i]

        # 计算预测正确的样本数
        if pred_label == true_label:
            accuracy += 1
        print("test", i, "predict label:", pred_label, "true label:", true_label)

    print("accuracy:", accuracy / m)
```

2019-04-03

```
test 20 predict label: 0 true label: 0
test 21 predict label: 1 true label: 1
test 22 predict label: 0 true label: 0
test 23 predict label: 2 true label: 2
test 24 predict label: 0 true label: 0
test 25 predict label: 2 true label: 2
test 26 predict label: 2 true label: 2
test 27 predict label: 0 true label: 0
test 28 predict label: 1 true label: 1
test 29 predict label: 2 true label: 2
test 30 predict label: 2 true label: 2
test 31 predict label: 0 true label: 0
test 32 predict label: 2 true label: 2
test 33 predict label: 1 true label: 1
test 34 predict label: 1 true label: 1
test 35 predict label: 0 true label: 0
test 36 predict label: 2 true label: 2
test 37 predict label: 0 true label: 0
done
accuracy: 0.9473684210526315
```

构建kNN模型一般流程

（1）收集数据：使用任何方法

（2）准备数据：距离计算所需要的数值，最好是结构化的数据格式

（3）分析数据：使用任意方法 2019-04-03

（4）训练模型：不适用此算法

（5）测试模型：计算错误率

（6）使用模型：首先需要输入样本数据集的结构化输出结果，然后运行kNN模型判定输入数据分别属于哪个分类

kNN应用

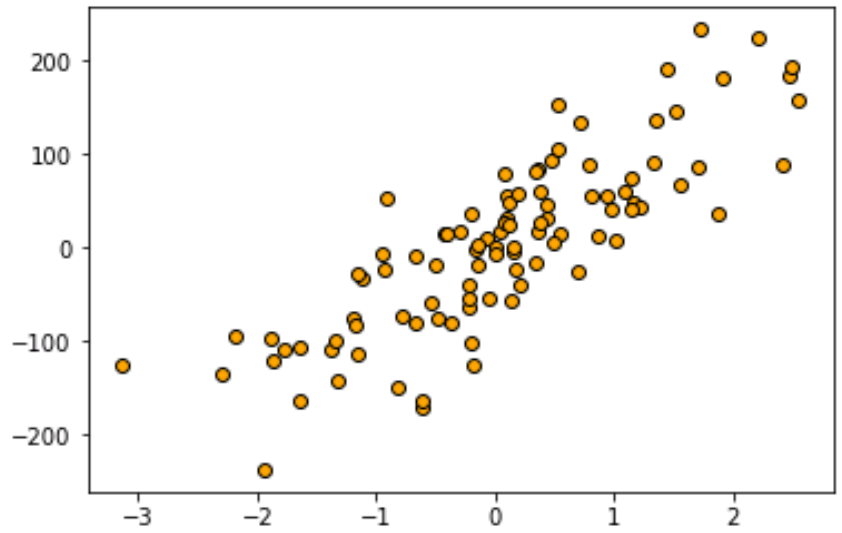
- 分类：适合多类别分类
- 回归：与分类相似。预测某个数据点的预测值时，模型会选择离该数据点最近的若干个训练数据集中的点，并且将它们的y值取平均值，并把该平均值作为新数据点的预测值。

2019-04-03

kNN回归模型

```
# 导入make_regression数据shengchen生成器
from sklearn.datasets import make_regression
# 生成特征数量为1，噪声为50数据集
X, y = make_regression(n_features=1,n_informative=1,noise=50,random_state=8)
# 用散点图将数据集进行可视化
plt.scatter(X,y,c='orange',edgecolor='k')
```

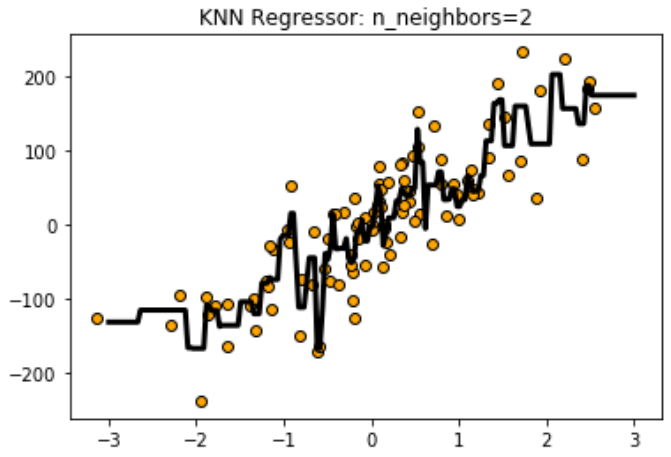
```
<matplotlib.collections.PathCollection at 0x978d630>
```



2019-04-03

```
from sklearn.neighbors import KNeighborsRegressor
# k = 2
reg2 = KNeighborsRegressor(n_neighbors=2)
reg2.fit(X,y)
# 预测结果可视化
plt.scatter(X,y,c='orange',edgecolor='k')
plt.plot(z, reg2.predict(z),c='k',linewidth=3)
plt.title('KNN Regressor: n_neighbors=2')
```

```
<matplotlib.text.Text at 0x98397b8>
```




```
print('模型评分：{:.2f}'.format(reg2.score(X,y)))
```

```
模型评分：0.86
```

kNN优缺点

- 最简单有效的分类方法
- 无需估计参数，即无需训练，属非参数模型
- 只计算“最近的”邻居样本，当样本不平衡时，k个邻居中大容量类的样本占多数或占极少数。
- 计算复杂度高，需计算新的数据点与样本集中每个数据的距离，复杂度是O(n)，空间度杂度也高。