KNN

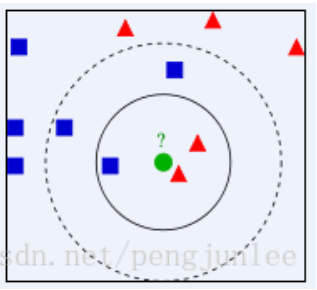
1.KNN

KNN（K-Nearest Neighbors）是一种常见的机器学习算法，主要用于**分类和回归**问题。KNN算法的核心思想是根据某个样本的特征，找到与该样本最接近的K个邻居，并基于这些邻居的特征来进行分类或回归预测。

KNN的工作原理如下：

1. 首先，给定一个训练数据集，该数据集包含了已经标记好的样本和它们的特征。
2. 当需要对一个新样本进行分类或回归预测时，KNN算法会计算该样本与训练数据集中每个样本之间的距离。常用的距离度量包括欧氏距离、曼哈顿距离、余弦相似度等。
3. 然后，KNN会选择与新样本距离最近的K个训练样本，这些样本被称为K个最近邻居。
4. 对于分类问题，KNN会统计这**K个最近邻居中每个类别的数量**，然后将新样本分配给具有**最多最近邻的类别**。
5. 对于回归问题，KNN会计算**K个最近邻居的平均值或加权平均值**，并将**这个值作为新样本的预测输出**。

举个例子，如下图所示，如何判断绿色圆应该属于哪一类，是属于红色三角形还是属于蓝色四方形？如果K=3，由于红色三角形所占比例为2/3，绿色圆将被判定为属于红色三角形那个类，如果K=5，由于蓝色四方形比例为3/5，因此绿色圆将被判定为属于蓝色四方形类。



KNN算法的主要优点是简单易懂，不需要模型训练，且适用于各种数据类型。然而，它也有一些缺点，包括计算成本高，对数据量大和维度高的数据集效率较低，对数据的分布和特征缩放敏感。因此，在实际应用中，需要根据问题的特点来选择适当的K值和距离度量，并可能需要进行特征工程以改进KNN的性能。

1.KNN算法的关键：

```
        （1）样本的所有特征都要做可比较的量化

        若是样本特征中存在非数值的类型，必须采取手段将其量化为数值。例如样本特征中包含颜色，可通过将颜色转换为灰度值来实现距离计算。

        （2）样本特征要做归一化处理

        样本有多个参数，每一个参数都有自己的定义域和取值范围，他们对距离计算的影响不一样，如取值较大的影响力会盖过取值较小的参数。所以样本参数必须做一些 scale 处理，最简单的方式就是所有特征的数值都采取归一化处置。
```

（3）需要一个距离函数以计算两个样本之间的距离

通常使用的距离函数有：欧氏距离、余弦距离、汉明距离、曼哈顿距离等，一般选欧氏距离作为距离度量，但是这是只适用于连续变量。在文本分类这种非连续变量情况下，汉明距离可以用来作为度量。通常情况下，如果运用一些特殊的算法来计算度量的话，K近邻分类精度可显著提高，如运用大边缘最近邻法或者近邻成分分析法。

2.knn的优缺点

knn优点：

1. 理论成熟，思想简单，既可以用来做分类又可以做回归
2. KNN是一种在线技术，新数据可以直接加入数据集而不必进行重新训练
3. 可用于非线性分类（数据集不要求线性可分）
4. 和朴素贝叶斯之类的算法比，对数据没有假设，**准确度高，对异常点不敏感**

knn缺点：

1. **计算量大，尤其是数据集非常大的时候**
2. **样本不平衡的时候，对稀有类别的预测准确率低：**

KNN算法在分类时有个主要的不足是，当样本不平衡时，如一个类的样本容量很大，而其他类样本容量很小时，有可能导致当输入一个新样本时，该样本的K个邻居中大容量类的样　　本占多数，如下图所示。该算法只计算最近的邻居样本，某一类的样本数量很大，那么或者这类样本并不接近目标样本，或者这类样本很靠近目标样本。无论怎样，数量并不能影响运行 结果。可以采用权值的方法(和该样本距离小的邻居权值大)来改进。

3. 建立kd树的时候，内存占用多。

3.适合什么场景

使用于特征与目标类之间的关系为比较复杂的数字类型，适合用于小规模数据集、无需假设数据分布、多类别分类和数据不平衡等场景。

4.常用的距离衡量公式都有哪些？具体说明它们的计算流程，以及使用场景？

1. **欧氏距离（Euclidean Distance）**：这是最常用的距离度量方法，它衡量样本点之间的直线距离。对于两个点A和B，欧氏距离为：
   $d(A,B) = \sqrt{\sum_{i=1}^{n}(A_i - B_i)^2}$
   其中，A_i 和 B_i 分别表示点A和B在第i个特征上的取值。
2. **曼哈顿距离（Manhattan Distance）**：曼哈顿距离是沿着坐标轴的距离总和，也称为“城市街区距离”。对于两个点A和B，曼哈顿距离为：
   $d(A,B) = \sum_{i=1}^{n}|A_i - B_i|$
3. **闵可夫斯基距离（Minkowski Distance）**：闵可夫斯基距离是欧氏距离和曼哈顿距离的通用形式，可以根据参数p来调整距离计算的方式。当p=2时，闵可夫斯基距离等同于欧氏距离，当p=1时等同于曼哈顿距离。
   $d(A,B) = \left(\sum_{i=1}^{n}|A_i - B_i|^p\right)^{1/p}$
4. **余弦相似度（Cosine Similarity）**：余弦相似度用于衡量两个向量之间的夹角余弦值，而不是点的距离。它通常用于文本分类和自然语言处理中。
   $d(A,B) = \frac{A \cdot B}{\|A\|\|B\|}$
   其中，$A \cdot B$ 表示向量A和B的点积，$\|A\|$ 和 $\|B\|$ 分别表示向量A和B的范数。

选择哪种距离度量方法通常取决于数据的特性和具体问题。欧氏距离适用于连续型特征，曼哈顿距离适用于有序的特征，而余弦相似度适用于文本等高维稀疏数据。在KNN中，通常需要根据实际情况选择合适的距离度量方法，并可能需要进行特征缩放以避免某些特征对距离计算的主导影响。

5.超参数K值过大或者过小对结果有什么影响，应该如何选择k值

如果选择较小的K值，就相当于用较小的领域中的训练实例进行预测，“学习”近似误差会减小，只有与输入实例较近或相似的训练实例才会对预测结果起作用，与此同时带来的问题是“学习”的估计误差会增大，换句话说，**K值的减小就意味着整体模型变得复杂，容易发生过拟合**；

如果选择较大的K值，就相当于用较大领域中的训练实例进行预测，其优点是可以减少学习的估计误差，但缺点是学习的近似误差会增大。这时候，与**输入实例较远（不相似的）训练实例也会对预测器作用，使预测发生错误**，且K值的增大就意味着整体的模型变得简单。

K=N，则完全不足取，因为此时无论输入实例是什么，都只是简单的预测它属于在训练实例中最多的累，模型过于简单，忽略了训练实例中大量有用信息。

在实际应用中，**K值一般取一个比较小的数值，例如采用交叉验证法**（简单来说，就是一部分样本做训练集，一部分做测试集）来选择最优的K值。

6.介绍一下Kd树？如何建树，以及如何搜索最近节点？

Kd树（K-dimensional tree）是一种用于高维空间中的数据结构，旨在加速搜索和检索最近邻点的算法。Kd树的名称源于它是K维树的缩写。Kd树的主要应用之一是K-Nearest Neighbors（KNN）算法，用于搜索K

个最近邻居。以下是Kd树的基本原理以及如何构建树和搜索最近节点的过程：

构建Kd树：

1. **选择轴（Dimension Selection）：** 选择一个轴来划分数据。通常，轴的选择是根据数据中方差最大的维度进行的，以确保均匀划分。例如，如果有三个维度（x、y、z），则可以选择具有最大方差的维度。
2. **选择中位数（Median Selection）：** 在选定的轴上，找到数据中位数。中位数是将数据集划分为两个子集的点，其中一半点位于左子树，另一半位于右子树。
3. **分割数据（Partition Data）：** 使用中位数将数据集划分为两个子集。一边包含小于中位数的点，另一边包含大于中位数的点。这将构成树中的一个节点。
4. **递归（Recursion）：** 递归地重复上述步骤，将数据划分为更小的子集，构建Kd树。递归结束的条件通常是数据集中只有一个点或没有点。
5. **交替轴（Alternate Axes）：** 在递归构建过程中，轴的选择会交替变化。这意味着每一级的树都沿着不同的维度划分数据，以确保树在高维空间中保持平衡。

搜索最近节点：

1. **从根节点开始：** 从Kd树的根节点出发，根据查询点的坐标选择与当前节点相应的轴。
2. **比较查询点：** 比较查询点与当前节点的坐标在选定轴上的值。根据比较结果，确定应该向左子树还是右子树移动。
3. **递归搜索：** 递归地重复上述步骤，直到找到最近邻的叶节点。在递归搜索的过程中，可以维护一个距离最近点的当前最小距离，以便在树的不同分支中进行剪枝。
4. **回溯（Backtracking）：** 在搜索过程中，可能需要回溯，以检查其他子树是否包含更接近的点。这是通过检查当前节点到查询点的距离与当前最小距离之间的关系来实现的。
5. **维护最近邻：** 最终，找到的叶节点就是最近邻的候选点。然后，向上回溯，检查其他分支以确保没有更近的点存在。维护最近邻的过程可能会多次回溯，直到确定了最近邻点。

Kd树是一种高效的数据结构，特别适用于高维空间中的最近邻搜索问题，因为它可以减少搜索空间的维数。然而，Kd树的构建和搜索过程可能受到数据分布的影响，如果数据分布不均匀，Kd树的性能可能会下降。

KNN的算法实现和代码

要自己动手实现KNN算法其实不难，主要有以下三个步骤：

```
算距离：给定待分类样本，计算它与已分类样本中的每个样本的距离；

找邻居：圈定与待分类样本距离最近的K个已分类样本，作为待分类样本的近邻；

做分类：根据这K个近邻中的大部分样本所属的类别来决定待分类样本该属于哪个分类；
```

```
import math
import csv
import operator
import random
import numpy as np
from sklearn.datasets import make_blobs

#Python version 3.6.5

# 生成样本数据集 samples(样本数量) features(特征向量的维度) centers(类别个数)
def createDataSet(samples=100, features=2, centers=2):
    return make_blobs(n_samples=samples, n_features=features, centers=centers, cluster_std=1.0, random_state=8)

# 加载鸢尾花卉数据集 filename(数据集文件存放路径)
def loadIrisDataset(filename):
    with open(filename, 'rt') as csvfile:
        lines = csv.reader(csvfile)
        dataset = list(lines)
        for x in range(len(dataset)):
            for y in range(4):
                dataset[x][y] = float(dataset[x][y])
        return dataset

# 拆分数据集 dataset(要拆分的数据集) split(训练集所占比例) trainingSet(训练集) testSet(测试集)
def splitDataSet(dataSet, split, trainingSet=[], testSet=[]):
    for x in range(len(dataSet)):
        if random.random() <= split:
            trainingSet.append(dataSet[x])
        else:
            testSet.append(dataSet[x])
# 计算欧氏距离
def euclideanDistance(instance1, instance2, length):
    distance = 0
    for x in range(length):
        distance += pow((instance1[x] - instance2[x]), 2)
    return math.sqrt(distance)

# 选取距离最近的K个实例
def getNeighbors(trainingSet, testInstance, k):
    distances = []
    length = len(testInstance) - 1
    for x in range(len(trainingSet)):
        dist = euclideanDistance(testInstance, trainingSet[x], length)
        distances.append((trainingSet[x], dist))
    distances.sort(key=operator.itemgetter(1))

    neighbors = []
    for x in range(k):
        neighbors.append(distances[x][0])
    return neighbors

#  获取距离最近的K个实例中占比例较大的分类
def getResponse(neighbors):
    classVotes = {}
    for x in range(len(neighbors)):
        response = neighbors[x][-1]
        if response in classVotes:
            classVotes[response] += 1
        else:
            classVotes[response] = 1
    sortedVotes = sorted(classVotes.items(), key=operator.itemgetter(1), reverse=True)
```

```
    return sortedVotes[0][0]

# 计算准确率
def getAccuracy(testSet, predictions):
    correct = 0
    for x in range(len(testSet)):
        if testSet[x][-1] == predictions[x]:
            correct += 1
    return (correct / float(len(testSet))) * 100.0


def main():
    # 使用自定义创建的数据集进行分类
    # x,y = createDataSet(features=2)
    # dataSet= np.c_[x,y]

    # 使用鸢尾花卉数据集进行分类
    dataSet = loadIrisDataset(r'C:\DevTolls\eclipse-pureh2b\python\DeepLearning\KNN\iris_dataset.txt')

    print(dataSet)
    trainingSet = []
    testSet = []
    splitDataSet(dataSet, 0.75, trainingSet, testSet)
    print('Train set:' + repr(len(trainingSet)))
    print('Test set:' + repr(len(testSet)))
    predictions = []
    k = 7
    for x in range(len(testSet)):
        neighbors = getNeighbors(trainingSet, testSet[x], k)
        result = getResponse(neighbors)
        predictions.append(result)
        print('>predicted=' + repr(result) + ',actual=' + repr(testSet[x][-1]))
    accuracy = getAccuracy(testSet, predictions)
    print('Accuracy: ' + repr(accuracy) + '%')
main()
```

KNN算法应用

使用KNN算法处理简单分类任务

在scikit-learn中，内置了若干个玩具数据集（Toy Datasets），还有一些API让我们可以自己动手生成一些数据集。接下来我们将使用scikit-learn的make_blobs函数来生成一个样本数量为200，分类数量为2的数据集，并使用KNN算法来对其进行分类。

```
# 导入画图工具
import matplotlib.pyplot as plt
# 导入数组工具
import numpy as np
# 导入数据集生成器
from sklearn.datasets import make_blobs
# 导入KNN 分类器
from sklearn.neighbors import KNeighborsClassifier
# 导入数据集拆分工具
from sklearn.model_selection import train_test_split

# 生成样本数为200，分类数为2的数据集
data=make_blobs(n_samples=200, n_features=2,centers=2, cluster_std=1.0, random_state=8)
X,Y=data

# 将生成的数据集进行可视化
```

```
# plt.scatter(X[:,0], X[:,1],s=80, c=Y,  cmap=plt.cm.spring, edgecolors='k')
# plt.show()

clf = KNeighborsClassifier()
clf.fit(X,Y)

# 绘制图形
x_min,x_max=X[:,0].min()-1,X[:,0].max()+1
y_min,y_max=X[:,1].min()-1,X[:,1].max()+1
xx,yy=np.meshgrid(np.arange(x_min,x_max,.02),np.arange(y_min,y_max,.02))
z=clf.predict(np.c_[xx.ravel(),yy.ravel()])

z=z.reshape(xx.shape)
plt.pcolormesh(xx,yy,z,cmap=plt.cm.Pastel1)
plt.scatter(X[:,0], X[:,1],s=80, c=Y,  cmap=plt.cm.spring, edgecolors='k')
plt.xlim(xx.min(),xx.max())
plt.ylim(yy.min(),yy.max())
plt.title("Classifier:KNN")

# 把待分类的数据点用五星表示出来
plt.scatter(6.75,4.82,marker='*',c='red',s=200)

# 对待分类的数据点的分类进行判断
res = clf.predict([[6.75,4.82]])
plt.text(6.9,4.5,'Classification flag: '+str(res))

plt.show()
```

使用KNN算法处理多元分类任务

接下来，我们再使用scikit-learn的make_blobs函数来生成一个样本数量为500，分类数量为5的数据集，并使用KNN算法来对其进行分类。

```
# 导入画图工具
import matplotlib.pyplot as plt
# 导入数组工具
import numpy as np
# 导入数据集生成器
from sklearn.datasets import make_blobs
# 导入KNN 分类器
from sklearn.neighbors import KNeighborsClassifier
# 导入数据集拆分工具
from sklearn.model_selection import train_test_split

# 生成样本数为500，分类数为5的数据集
data=make_blobs(n_samples=500, n_features=2,centers=5, cluster_std=1.0, random_state=8)
X,Y=data

# 将生成的数据集进行可视化
# plt.scatter(X[:,0], X[:,1],s=80, c=Y,  cmap=plt.cm.spring, edgecolors='k')
# plt.show()

clf = KNeighborsClassifier()
clf.fit(X,Y)

# 绘制图形
x_min,x_max=X[:,0].min()-1,X[:,0].max()+1
y_min,y_max=X[:,1].min()-1,X[:,1].max()+1
xx,yy=np.meshgrid(np.arange(x_min,x_max,.02),np.arange(y_min,y_max,.02))
```

```
z=clf.predict(np.c_[xx.ravel(),yy.ravel()])

z=z.reshape(xx.shape)
plt.pcolormesh(xx,yy,z,cmap=plt.cm.Pastel1)
plt.scatter(X[:,0], X[:,1],s=80, c=Y,  cmap=plt.cm.spring, edgecolors='k')
plt.xlim(xx.min(),xx.max())
plt.ylim(yy.min(),yy.max())
plt.title("Classifier:KNN")

# 把待分类的数据点用五星表示出来
plt.scatter(0,5,marker='*',c='red',s=200)

# 对待分类的数据点的分类进行判断
res = clf.predict([[0,5]])
plt.text(0.2,4.6,'Classification flag: '+str(res))
plt.text(3.75,-13,'Model accuracy: {:.2f}'.format(clf.score(X, Y)))

plt.show()
```

使用KNN算法进行回归分析

这里我们使用scikit-learn的make_regression生成数据集来进行实验，演示KNN算法在回归分析中的表现。

```
# 导入画图工具
import matplotlib.pyplot as plt
# 导入数组工具
import numpy as np

# 导入用于回归分析的KNN模型
from sklearn.neighbors import KNeighborsRegressor
# 导入数据集拆分工具
from sklearn.model_selection import train_test_split
# 导入数据集生成器
from sklearn.datasets.samples_generator import make_regression
from docutils.utils.math.math2html import LineWriter

# 生成样本数为200，分类数为2的数据集
X,Y=make_regression(n_samples=100,n_features=1,n_informative=1,noise=50,random_state=8)

# 将生成的数据集进行可视化
# plt.scatter(X,Y,s=80, c='orange',  cmap=plt.cm.spring, edgecolors='k')
# plt.show()
reg = KNeighborsRegressor(n_neighbors=5)

reg.fit(X,Y)

# 将预测结果用图像进行可视化
z = np.linspace(-3,3,200).reshape(-1,1)
plt.scatter(X,Y,c='orange',edgecolor='k')
plt.plot(z,reg.predict(z),c='k',Linewidth=3)
#
plt.title("KNN Regressor")

plt.show()
```