应用数据科学导论

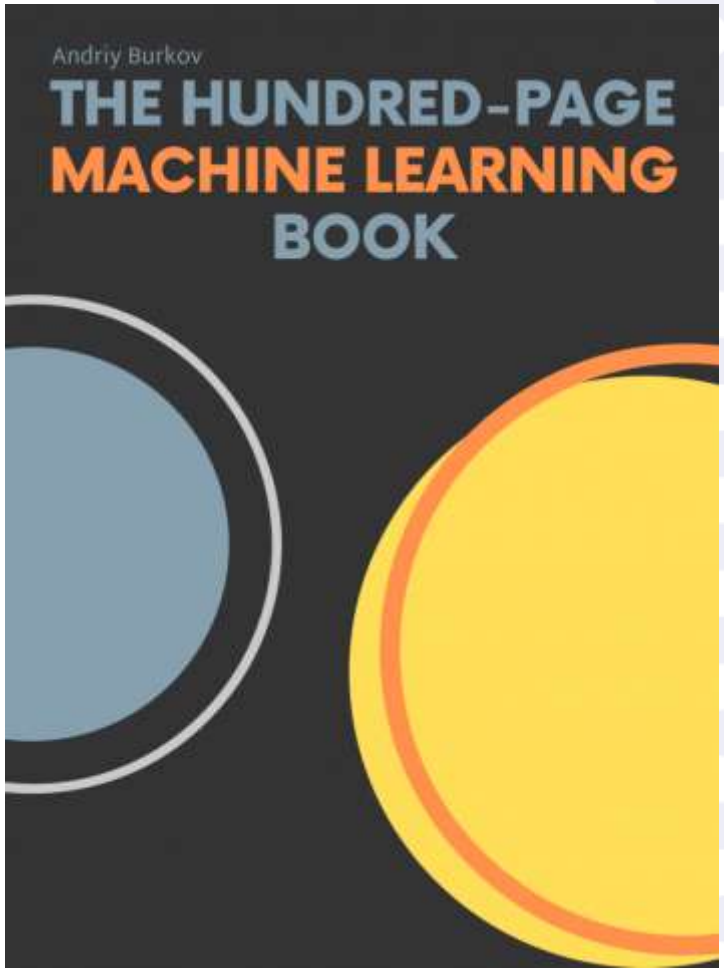
—— 机器学习之KNN

Ht_song@163.com

2020.06

参考书


O'REILLY
TURING 图灵程序设计丛书
数据科学入门
Data Science from Scratch
[美] Joel Grus 著
高蓉 韩波 译
中国工信出版集团
人民邮电出版社
POSTS & TELECOM PRESS


Andriy Burkov
THE HUNDRED-PAGE
MACHINE LEARNING
BOOK

主要内容

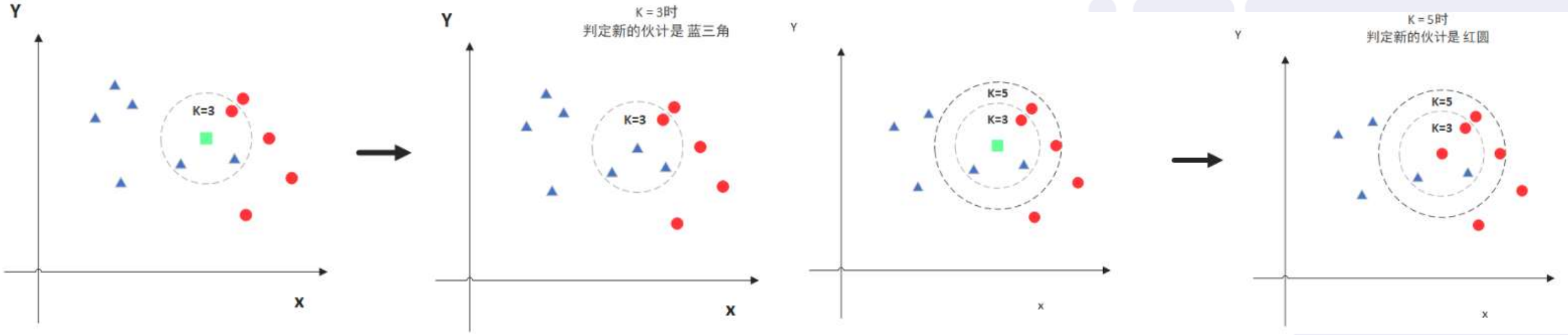
- 数据科学简介
- Python语言
- 数据统计可视化——KNN
- 数据分析方法
- 分析项目实践

k-近邻算法

(k-nearest neighbor，KNN)

kNN法概述

- k近邻法（k-nearest neighbor, KNN）是最常用的分类算法之一。
- 基本做法是：给定测试实例，基于某种距离度量找出训练集中与其最靠近的k个实例点，然后基于这k个最近邻的信息来进行预测。



KNN算法的基本要素

- **k值的选择**：即输入新实例要取多少个训练实例点作为近邻。
- **距离度量方式**：样本空间中两个点之间的距离度量表示的是两个样本点之间的相似程度，距离越小，表示相似程度越高；相反，距离越大，相似程度越低。
- **分类的决策规则**：常用的方式是取k个近邻训练实例中类别出现次数最多者作为输入新实例的类别。

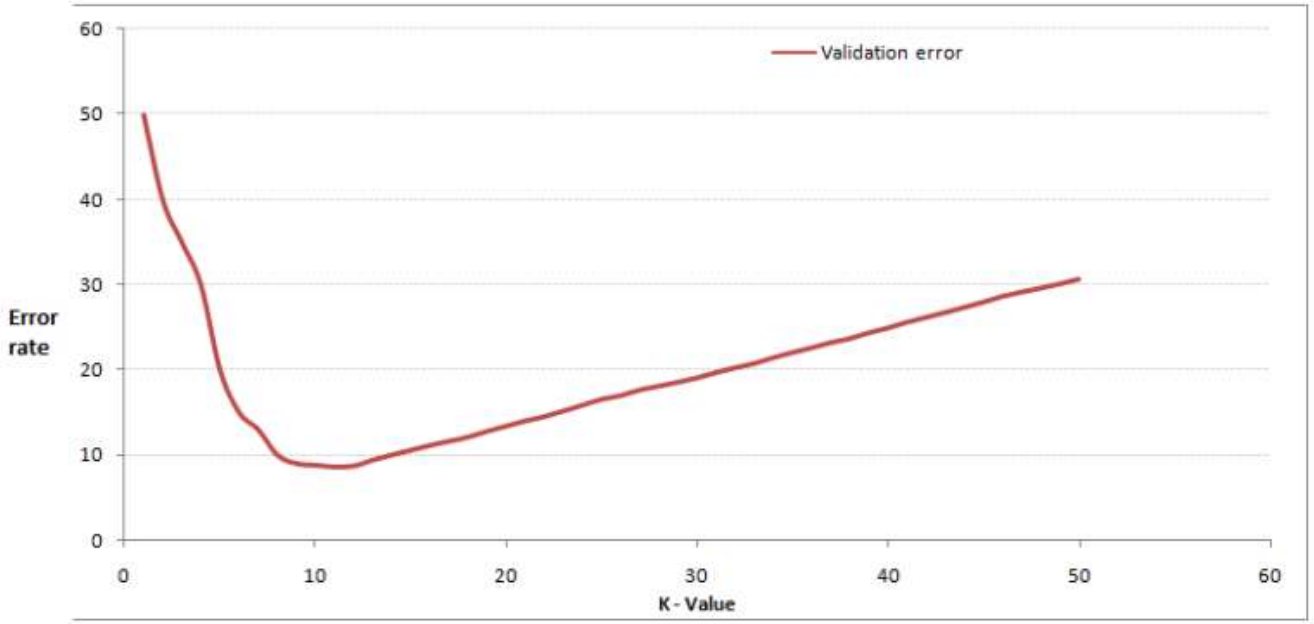
一、 k值的选择

- 交叉验证：

1）将样本数据按照一定比例拆分出训练用的数据和验证用的数据。

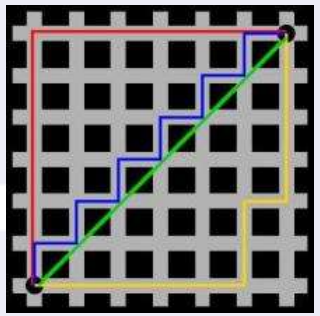
2）从选取一个较小的K值开始，不断增加K的值

3）然后计算验证集合的方差，最终找到合适的K值。



- k值较小时，整体模型会变得复杂，且对近邻的训练实例点较为敏感，容易出现过拟合。
- k值较大时，模型则会趋于简单，此时较远的训练实例点也会起到预测作用，容易出现欠拟合，特殊的，当k取N时，此时无论输入实例是什么，都会将其预测为属于训练实例中最多的类别。

二、距离度量方式



- 1.**欧氏距离**：欧氏空间中两点间的直线距离（图中绿线），适用于各个向量标准统一的情况
- 2.**曼哈顿距离**：两个点在标准坐标系上的绝对轴距总和（图中红、蓝、黄线），应用场景如棋盘、城市里两个点之间的距离等
- 3. **切比雪夫距离**（棋盘距离）：走斜线等同于走直角
- 4. **闵可夫斯基距离**（闵氏距离）：将各个分量的量纲（单位）当作相同的看待。没有考虑各个分量的分布（期望，方差等)可能是不同的。
- 5.**马氏距离**：量纲无关，排除变量之间的相关性的干扰，适用于数据分散、方差大的数据集
- 6.**余弦距离**：取值范围为[-1,1]，夹角余弦越大表示两个向量的夹角越小。常用于文本识别，比如新闻的挖掘
- 7.**汉明距离**：两个等长字符串s1与s2之间的汉明距离定义为将其中一个变为另外一个所需要作的最小替换次数，用于比较字符串的相似程度
- 8.**杰卡德距离**：用两个集合中不同元素占所有元素的比例来衡量两个集合的区分度
- 9.**相关系数& 相关距离**：取值范围是[-1,1]。相关系数的绝对值越大，则表明X与Y相关度越高
- 10.**信息熵**：用于衡量分布的混乱程度或分散程度。分布越分散(或者说分布越平均)，信息熵就越大。

- **如果数据比较密集，变量之间基本都存在共有值，且这些距离数据都是非常重要的，那就使用欧氏距离或者曼哈顿距离。**
- **如果数据存在“分数膨胀“问题，就使用皮尔逊相关系数。**

  例：同样是5分制的评分，每个用户的评分标准不同，如Bill没有打出极端的分数，都在2-4分之间，Jordyn打分都在4-5之间，Hailey的评分不是1就是4，这种情况下可以利用皮尔逊相关系数来找到相似的用户。
- **如果数据是稀疏的，就使用余弦相似度。**

  例：假设有这样一个数据集，一个在线音乐网站，有10000w首音乐（这里不考虑音乐类型，年代等因素），每个用户常听的也就其中的几十首，这种情况下使用曼哈顿或者欧几里得或者皮尔逊相关系数进行计算用户之间相似性，计算相似值会非常小，因为用户之间的交集本来就很少，这样对于计算结果来讲是很不准确的，这个时候就需要余弦相似度了，余弦相似度进行计算时会自动略过这些非零值。

- **欧式距离**
- 欧式距离（L2范数）其实就是空间中两点间的距离，是我们最常用的一种距离计算公式。因为计算是基于各维度特征的绝对数值，所以欧氏度量需要保证各维度指标在相同的刻度级别，比如对身高（cm）和体重（kg）两个单位不同的指标使用欧式距离可能使结果失效。所以在使用欧氏距离时，应该尽量将特征向量的每个分量归一化，以减少因为特征值的刻度级别不同所带来的干扰。

两个n维向量a(x11,x12,...,x1n)与 b(x21,x22,...,x2n)间的欧氏距离：

$$d_{12} = \sqrt{\sum_{k=1}^{n}(x_{1k} - x_{2k})^2}$$

```
from math import *
# 欧式距离
def dist(x, y):
    return sqrt(sum(pow(a - b, 2) for a, b in zip(x, y)))

x = [1, 3, 2, 4]
y = [2, 5, 3, 1]

print(dist(x, y))
```

```
3.872983346207417
```

- **闵可夫斯基距离（闵氏距离）**
- 闵可夫斯基距离不是一种距离，而是一类距离的定义。
- 其数学定义如下：$\left(\sum_{i=1}^{n}|x_i - y_i|^p\right)^{1/p}$.
- 其中p可以随意取值，可以是负数，也可以是正数，或是无穷大。当p=1时，又叫做曼哈顿距离，当p=2时，又叫做欧式距离，当p=∞时，又叫做契比雪夫距离。

class sklearn.neighbors.KNeighborsClassifier中参数p的取值

- **曼哈顿距离**
- 又称为城市街区距离，早期计算图形学中，屏幕是由像素构成，是整数，点的坐标也一般是整数，原因是浮点运算很昂贵，很慢而且有误差，如果直接使用AB的距离，则必须要进行浮点运算，如果使用AC和CB，则只要计算加减法即可，这就大大提高了运算速度

```
# 曼哈顿距离
def dist2(x, y):
    return sum(abs(a - b) for a, b in zip(x, y))

print(dist2(x, y))
```

```
7
```

归一化处理

- 为了不使一些特征变量数值相差太大从而影响模型的准确性，因此要对数据进行处理，即特征的缩放。

$$Xnew = \frac{X - min(X)}{max(X) - min(X)}$$

归一化处理，将原数据处理成[0,1]区间内的数值

```
In [6]: from sklearn.datasets import load_iris
        from sklearn.preprocessing import MinMaxScaler
        iris = load_iris()
        #原数据
        iris
```

```
Out[6]: {'data': array([[5.1, 3.5, 1.4, 0.2],
                        [4.9, 3. , 1.4, 0.2],
                        [4.7, 3.2, 1.3, 0.2],
                        [4.6, 3.1, 1.5, 0.2],
                        [5. , 3.6, 1.4, 0.2],
                        [5.4, 3.9, 1.7, 0.4],
                        [4.6, 3.4, 1.4, 0.3],
                        [5. , 3.4, 1.5, 0.2],
                        [4.4, 2.9, 1.4, 0.2],
                        [4.9, 3.1, 1.5, 0.1],
                        [5.4, 3.7, 1.5, 0.2],
```

```
In [7]: #归一化后
        MinMaxScaler().fit_transform(iris.data)
```

```
Out[7]: array([[0.22222222, 0.625     , 0.06779661, 0.04166667],
               [0.16666667, 0.41666667, 0.06779661, 0.04166667],
               [0.11111111, 0.5       , 0.05084746, 0.04166667],
               [0.08333333, 0.45833333, 0.08474576, 0.04166667],
               [0.19444444, 0.66666667, 0.06779661, 0.04166667],
               [0.30555556, 0.79166667, 0.11864407, 0.125     ],
               [0.08333333, 0.58333333, 0.06779661, 0.08333333],
               [0.19444444, 0.58333333, 0.08474576, 0.04166667],
               [0.02777778, 0.375     , 0.06779661, 0.04166667],
               [0.16666667, 0.45833333, 0.08474576, 0.        ],
               [0.30555556, 0.70833333, 0.08474576, 0.04166667],
               [0.13888889, 0.58333333, 0.10169492, 0.04166667],
               [0.13888889, 0.41666667, 0.06779661, 0.        ],
               [0.        , 0.41666667, 0.01694915, 0.        ],
               [0.41666667, 0.83333333, 0.03389831, 0.04166667],
               [0.38888889, 1.        , 0.08474576, 0.125     ],
               [0.30555556, 0.79166667, 0.05084746, 0.125     ],
               [0.22222222, 0.625     , 0.06779661, 0.08333333],
```

三、 分类决策规则

- 一般是多数表决，即由输入实例的k个近邻的训练实例中的多数类决定输入实例的类别。
- 在样本数不均衡时，可以根据距离远近做权值的修改

class sklearn.neighbors.KNeighborsClassifier(n_neighbors=5, **weights='uniform'**, algorithm='auto', leaf_size=30,p=2, metric='minkowski', metric_params=None, n_jobs=1, **kwargs)

`weights` : str or callable, optional (default = 'uniform')
weight function used in prediction. Possible values::

1. 'uniform' : uniform weights. All points in each neighborhood are weighted equally.
2. 'distance' : weight points by the inverse of their distance. in this case, closer neighbors of a query point will have a greater influence than neighbors which are further away.
3. [callable] : a user-defined function which accepts an array of distances, and returns an array of the same shape containing the weights.

```
class sklearn.neighbors.KNeighborsClassifier(n_neighbors=5, weights='uniform'
, algorithm='auto', leaf_size=30,p=2, metric='minkowski', metric_params=None, n_
jobs=1, **kwargs)
```

- **n_neighbors**：即K 值。
- **weights**：权重
- **algorithm**：
  'brute' ：蛮力实现（直接计算距离存储比较，数据集小）；
  'kd_tree' ：KD 树实现 KNN（数据量大，基于欧氏距离的特性可以快速处理20维以内的数据集）；
  'ball_tree' ：球树实现 KNN（基于更一般的距离特性，适合处理高维数据）；
  'auto' ： 默认参数，自动选择合适的方法构建模型。
  不过当数据较小或比较稀疏时，无论选择哪个最后都会使用 'brute'
- **leaf_size**：当使用KD树或球树，指的是是停止建子树的叶子节点数量的阈值。默认30，但如果数据量增多这个参数需要增大，否则速度过慢不说，还容易过拟合。
- **p**：和metric结合使用的，当metric参数是"minkowski"的时候，p=1为曼哈顿距离， p=2为欧式距离。默认为p=2。
- **metric**：指定距离度量方法，一般都是使用欧式距离。
  'euclidean' ：欧式距离； 'manhattan' ：曼哈顿距离；
  'chebyshev' ：切比雪夫距离；'minkowski'： 闵可夫斯基距离，默认参数
- **n_jobs**：指定多少个CPU进行运算，默认是-1，也就是全部都算。

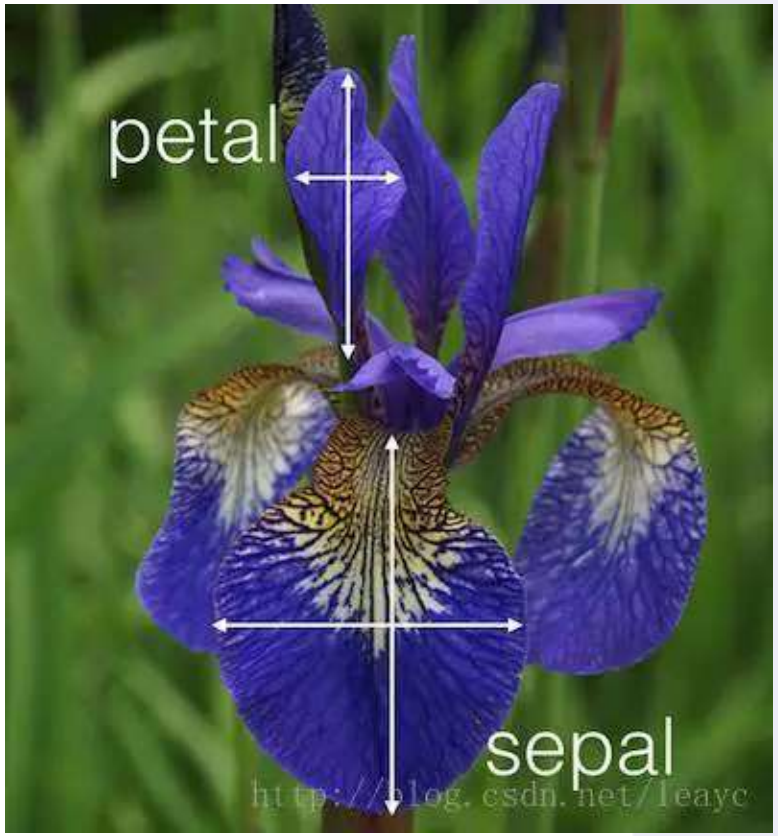
实例：用KNN实现鸢尾花分类

数据集内包含 3 类共 150 条记录，每类各 50 个数据。

每条记录都有 4 项特征：

花萼长度、花萼宽度、花瓣长度、花瓣宽度，

可以通过这4个特征预测鸢尾花属于（setosa, versicolour, virginica）中的哪一品种。



示例1：简单例子了解knn

In [4]:
```
import numpy as np
from sklearn import datasets
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn import datasets
```

加载鸢尾花数据集，拆分属性和类别数据

In [5]:
```
iris = datasets.load_iris()
iris_X = iris.data
iris_y = iris.target
```

拆分数据集为训练数据和测试数据

In [11]:
```
iris_X_train , iris_X_test, iris_y_train ,iris_y_test = train_test_split(iris_X, iris_y,
                                                                         test_size=0.2,
                                                                         random_state=0)
```

提供数据集进行训练

In [12]:
```
knn = KNeighborsClassifier()
knn.fit(iris_X_train, iris_y_train)
```

Out[12]:
```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                     weights='uniform')
```

预测测试集数据鸢尾花类型

In [15]:
```
predict_result = knn.predict(iris_X_test)
print('预测结果：{}'.format(predict_result))
```

```
预测结果：[2 1 0 2 0 2 0 1 1 1 2 1 1 1 2 0 1 1 0 0 2 1 0 0 2 0 0 1 1 0]
```

计算预测的准确率

In [16]:
```
print(knn.score(iris_X_test, iris_y_test))
```

```
0.9666666666666667
```

示例2：选取K值+绘制决策边界

In [4]:
```
from sklearn.datasets import load_iris
from sklearn.model_selection  import cross_val_score
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifier
```

In [5]:
```
#读取鸢尾花数据集
iris = load_iris()
x = iris.data
y = iris.target
print(x)
```

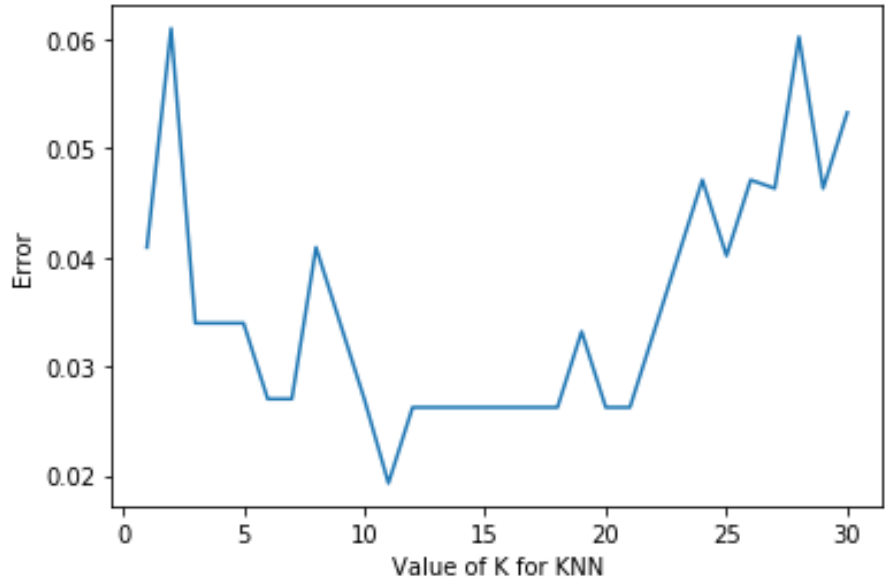
```
[[5.1 3.5 1.4 0.2]
 [4.9 3.  1.4 0.2]
 [4.7 3.2 1.3 0.2]
 [4.6 3.1 1.5 0.2]
 [5.  3.6 1.4 0.2]
 [5.4 3.9 1.7 0.4]
 [4.6 3.4 1.4 0.3]
 [5.  3.4 1.5 0.2]
 [4.4 2.9 1.4 0.2]
 [4.9 3.1 1.5 0.1]
 [5.4 3.7 1.5 0.2]
 [4.8 3.4 1.6 0.2]
 [4.8 3.  1.4 0.1]
```

In [6]:
```
print(y)
```

```
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2]
```

In [7]:
```
k_range = range(1, 31)
k_error = []
#循环，取k=1到k=31，查看误差效果
for k in k_range:
    knn = KNeighborsClassifier(n_neighbors=k)
    #cv参数决定数据集划分比例，这里是按照5:1划分训练集和测试集，accuracy：评价指标是准确度
    #cross_val_score:交叉验证
    scores = cross_val_score(knn, x, y, cv=6, scoring='accuracy')
    k_error.append(1 - scores.mean())
```

In [8]:
```
#画图，x轴为k值，y值为误差值
plt.plot(k_range, k_error)
plt.xlabel('Value of K for KNN')
plt.ylabel('Error')
plt.show()
```



选取合适K值

由图，K=11时误差最小。在实际问题中，如果数据集比较大，为了减少训练时间，K的取值范围可以缩小。

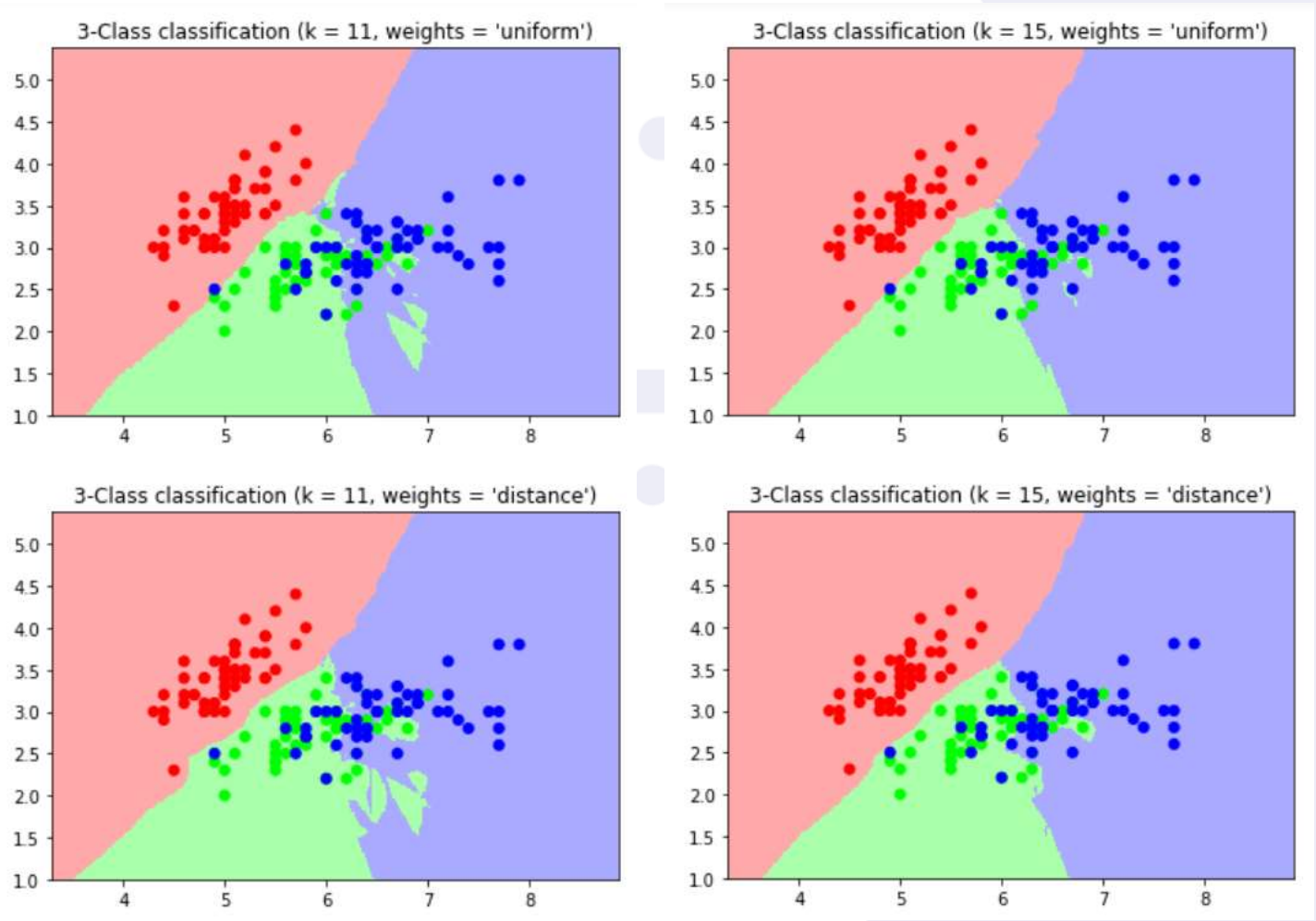
KNN分类决策边界可视化呈现

（对比不同权重和K值下的决策边界）

```
import matplotlib.pyplot as plt
import numpy as np
from matplotlib.colors import ListedColormap
from sklearn import neighbors, datasets

# 导入数据
iris = datasets.load_iris()
# 只采用前两个feature,方便画图在二维平面显示
x = iris.data[:, :2]
y = iris.target
```

```
h = .02  # 网格中的步长

# 创建彩色的图
cmap_light = ListedColormap(['#FFAAAA', '#AAFFAA', '#AAAAFF'])
cmap_bold = ListedColormap(['#FF0000', '#00FF00', '#0000FF'])
```

```
#绘制两种权重参数下KNN的效果图
for n_neighbors in [11,15]:
    for weights in ['uniform', 'distance']:
        # 创建了一个knn分类器的实例，并拟合数据。

        clf = neighbors.KNeighborsClassifier(n_neighbors, weights=weights)
        clf.fit(x, y)
        # 绘制决策边界。为此，我们将为每个类别分配一个颜色
        # 来绘制网格中的点 [x_min, x_max]x[y_min, y_max].
        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, h),
                             np.arange(y_min, y_max, h))
        Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
        # 将结果放入一个彩色图中
        Z = Z.reshape(xx.shape)
        plt.figure()
        plt.pcolormesh(xx, yy, Z, cmap=cmap_light)
        # 绘制训练点
        plt.scatter(x[:, 0], x[:, 1], c=y, cmap=cmap_bold)
        plt.xlim(xx.min(), xx.max())
        plt.ylim(yy.min(), yy.max())
        plt.title("3-Class classification (k = %i, weights = '%s')"
              % (n_neighbors, weights))
        plt.show()
```



k近邻法优势

- **没有明显的训练过程**，而是在程序开始运行时，把数据集加载到内存后，不需要进行训练，直接进行预测，所以训练时间复杂度为0
- **理论简单，效果强大**
- **准确性高**，对噪声和异常值有较高的容忍度
- 区别于感知机、逻辑回归、SVM，k近邻算法**支持解决多分类问题**
- 可以用来**解决回归问题**，使用的库函数为KNeighborsRegressor

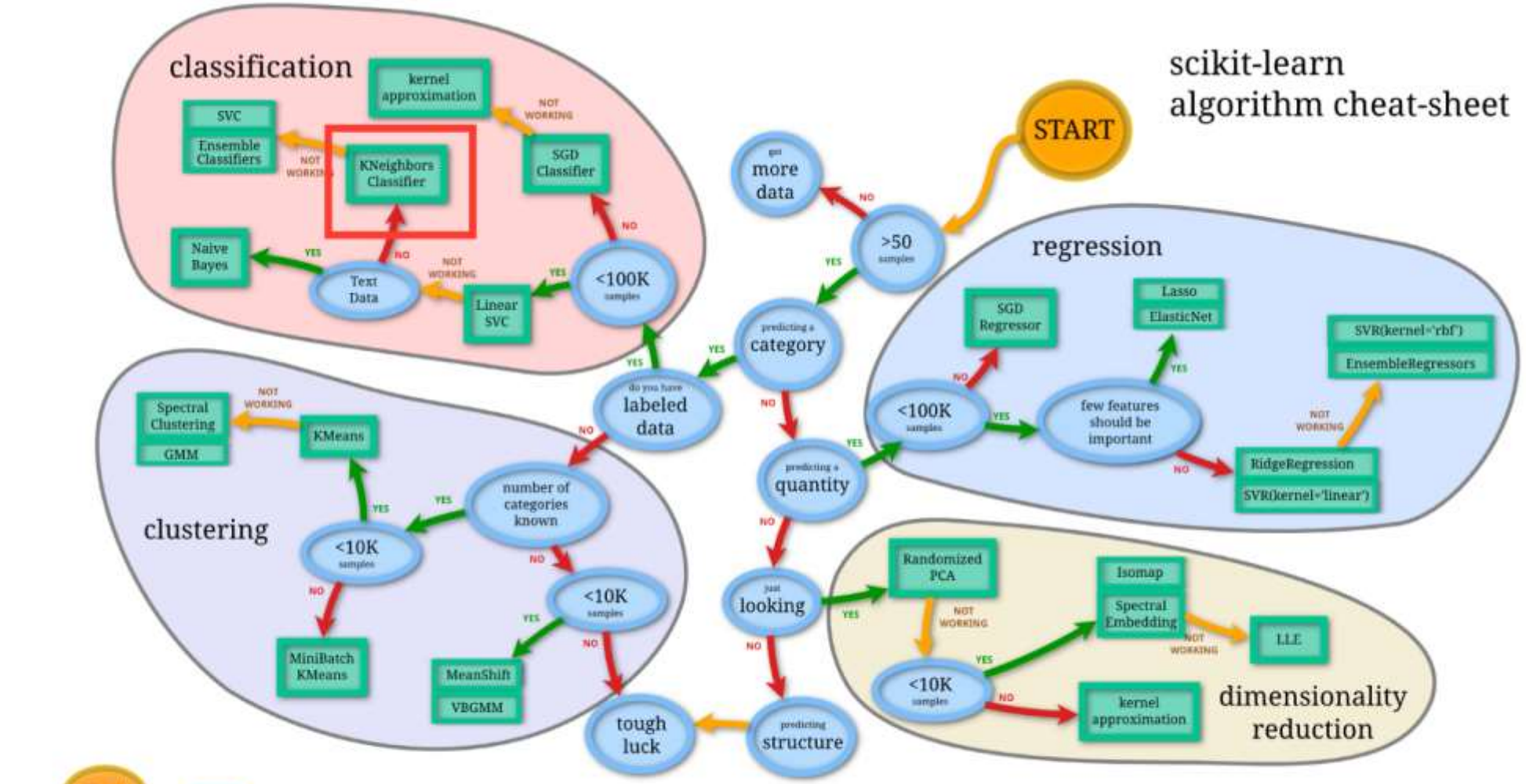
```
from sklearn.neighbors import KNeighborsRegressor
```

k近邻法劣势

- **效率低下**： k近邻算法每预测一个“点”的分类都会重新进行一次全局运算，对于样本容量大的数据集计算量比较大
- **高度数据相关**：一旦数据中存在一些误差数据（最近周边的几个数据一旦出错），则其准确度就会很难保证，很容易出现错误的预测结果。
- **数据预测结果不具备可解释性**：预测结果只是来自于对于测试数据最近的点的属性，整体上很难解释，也导致了很难进行后续的改进和发展
- **维数灾难**：随着数据维度的增加，看似"非常接近"的两个点之间的距离会越来越远；当然可以对其进行降维，不过对于整体算法的影响很大

1维	0到1的距离	1
2维	(0,0)到(1,1)的距离	1.414
3维	(0,0,0)到(1,1,1)的距离	1.73
64维	(0,0,...0)到(1,1,...,1)	8
10000维	(0,0,...0)到(1,1,...,1)	100

什么时候使用knn算法？



对比：KNN和K-Means

K-近邻算法（KNN）（有监督算法，分类算法）

K-均值聚类（K-means）（无监督学习、聚类算法，随机算法），采用距离作为相似性指标，从而发现给定数据集中的K个类，且每个类的中心是根据类中所有值的均值得到，每个类用聚类中心来描述。适合凸数据集

相同：

- K值都是重点
- 都需要计算平面中点的距离

相异：

- Knn和Kmeans的核心都是通过计算空间中点的距离来实现目的，只是他们的目的是不同的。KNN的最终目的是分类，而Kmeans的目的是给所有距离相近的点分配一个类别，也就是聚类。

数据，是信息时代的真相！