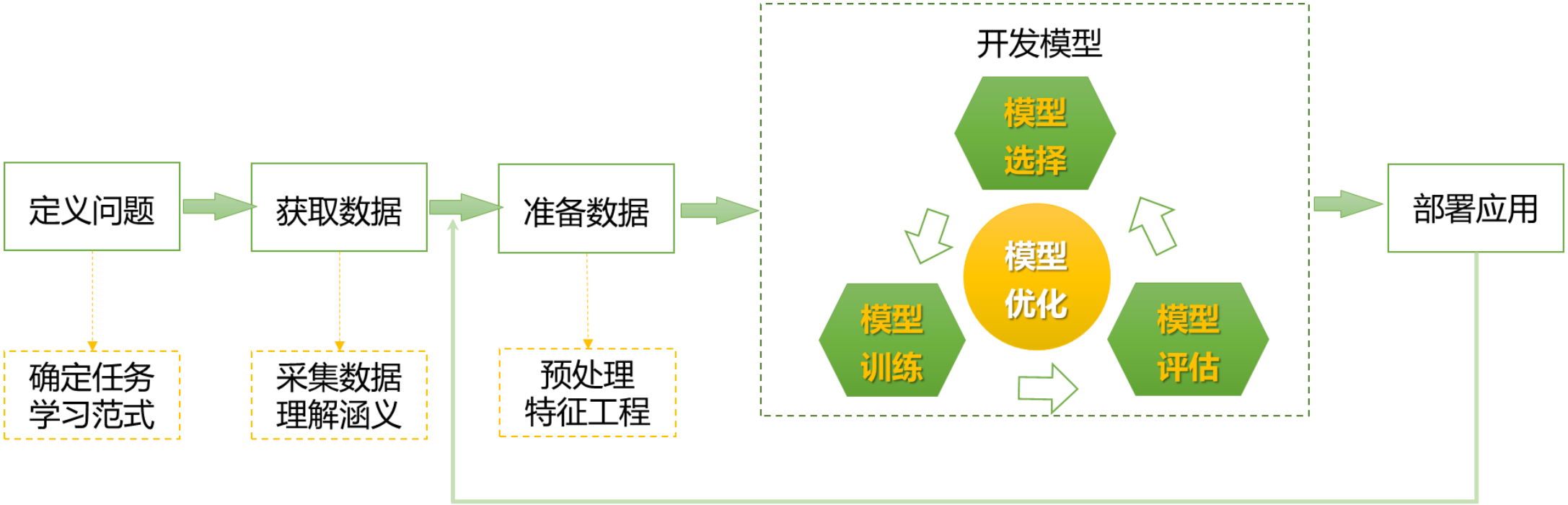
同濟大學
TONGJI UNIVERSITY

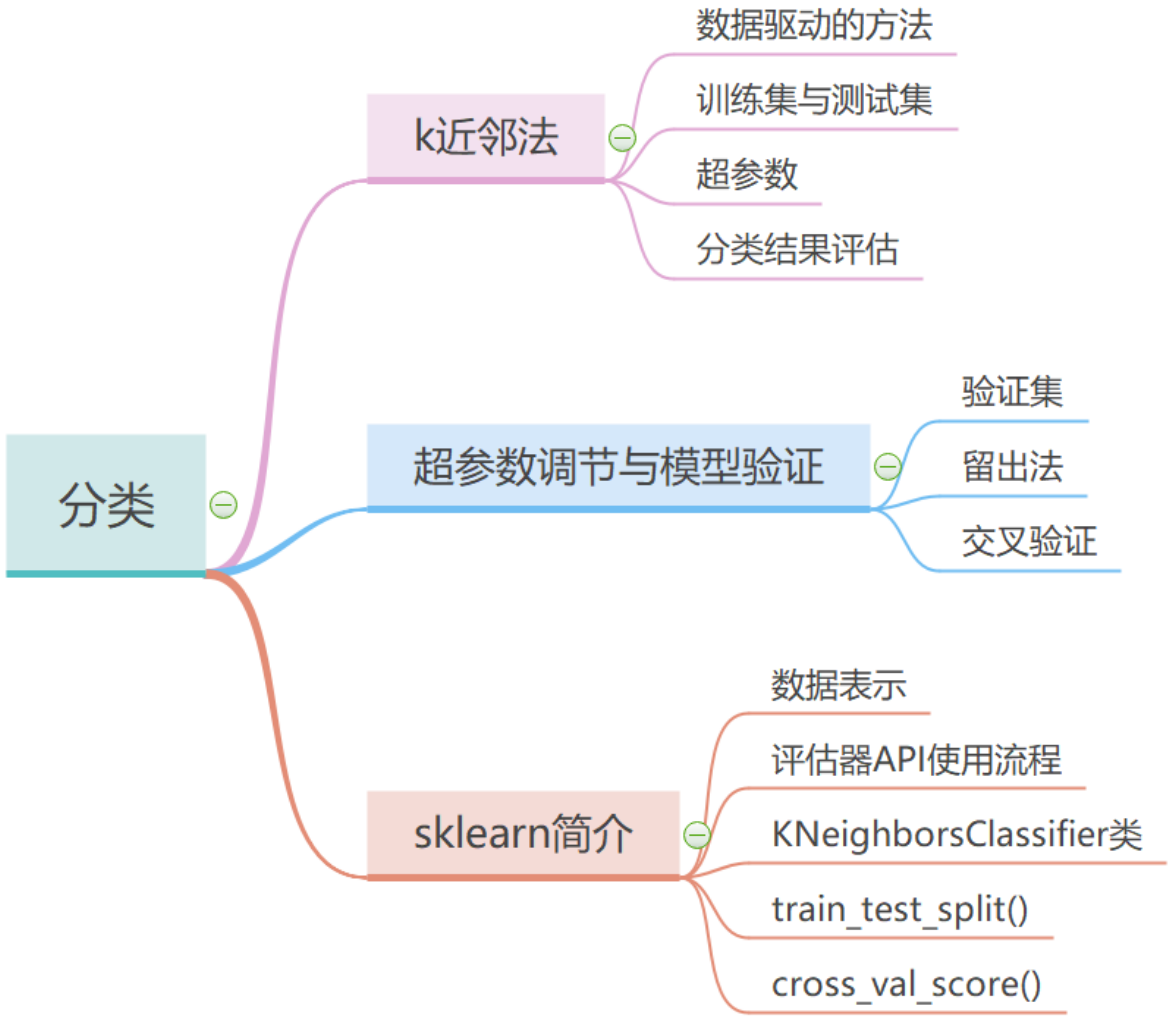
王睿智 ruizhiwang@tongji.edu.cn

人工智能技术与应用

监督学习回顾

- 监督学习
- 监督学习的两类任务：分类、回归
- 机器学习工作流程





3.1.1 引例 鸢尾花分类

例3.1 某地区有三种鸢尾花（virginica、Setosa、Versicolor），已收集一些数据，请设计一个模型，给定鸢尾花的特征能自动识别它的种类。

1. 定义问题

可否用机器学习解决？属于哪类问题？

2. 获取数据

3. 划分数据集

4. 选一种模型，如 kNN

例3.1 鸢尾花分类 | 划分数据集

读入iris.data，随机抽取出个20%（30个）作测试，余下80%作训练。

```
import pandas as pd
import numpy as np

iris = pd.read_csv('iris.data',names=['sepal_length','sepal_width',
                                      'petal_length', 'petal_width','species'])
data = iris.values  # 返回numpy数组
num_test = 30       # 30个样本作测试

np.random.seed(3)
np.random.shuffle(data)  # 打乱数据

test_set = data[:num_test] # 定义测试集，取前30个样本
Xtest = np.array(test_set[:,:4],dtype='float')
ytest = np.array(test_set[:,-1])

train_set = data[num_test:] # 定义训练集，取后120个样本
Xtrain = np.array(train_set[:,:4],dtype='float')
ytrain = np.array(train_set[:,-1])
```

随机洗牌

代码：例3.1 鸢尾花分类-2022春v1.ipynb

3.1.2 k近邻算法

k-近邻算法(kNN) 是监督学习算法中最简单的一种，

可用于分类和回归任务。

基本思想:

新数据点 x 的预测结果由（训练集中）离其最近的k个近邻样本所决定。 k是近邻个数，k=1时的近邻法即为最近邻法。

- 对分类任务, x 的类别由这k个近邻投票决定;
- 而对回归任务， x 的取值是这k个近邻目标值的平均。

例3.2 性别预测

有如下9个人的身高、体重和性别（标签）数据，

请预测一个身高175cm、体重75kg的人是男性还是女性?

身高（cm）	体重（kg）	标签
158	64	male
170	66	male
183	84	male
191	80	male
155	49	Female
163	59	Female
180	67	Female
158	54	Female
170	67	Female

思路：

1. 首先确定距离衡量方法，这里选择欧氏距离

$$d(p,q)=\sqrt{(p_1-q_1)^2+(p_2-q_2)^2}$$

2. 计算测试实例与所有训练实例之间的距离

3. 设 k=3，选取3个最近的训练实例

4. 找出距离最近的3个邻居中最普遍的性别

例 kNN预测性别 | 计算距离

身高（cm）	体重（kg）	标签	和测试实例的距离
158	64	male	20.25
170	66	male	12.08
183	84	male	12.04
191	80	male	16.76
155	49	Female	32.80
163	59	Female	20.00
180	67	Female	9.43
158	54	Female	27.02
170	67	Female	9.43

例 kNN预测性别 | 代码

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
plt.rcParams["font.sans-serif"]=["SimHei"] #设置字体
plt.rcParams["axes.unicode_minus"]=False #解决图像中的“-”负号的乱码问题
```

```
#  1.绘制训练数据
X_train = np.array([[158, 64],[170, 86],[183, 84], [191, 80], [155, 49],
                    [163, 59], [180, 67], [158, 54], [170, 67]])
y_train = ['male', 'male', 'male', 'male', 'female', 'female', 'female', 'female', 'female']
plt.title('分性别的身高体重图')
plt.xlabel('身高（cm）')
plt.ylabel('体重（kg）')
for i, x in enumerate(X_train):
    plt.scatter(x[0], x[1], c='k', marker='x' if y_train[i] == 'male' else 'D')
plt.grid(True)
plt.show()
```

例 kNN预测性别 | 代码

```
# 2. 计算距离
x = np.array([[175, 75]])     # 测试实例
distances = np.sqrt(np.sum((X_train - x)**2, axis=1)) # 欧氏距离
print(distances)
```

```
[20.24845673 12.08304597 12.04159458 16.76305461 32.80243893 20.
  9.43398113 27.01851217  9.43398113]
```

```
# 3. 找k个近邻实例
k=3
nearest_neighbor_indices = distances.argsort()[:k] # 最近3近邻的索引
print('3个近邻的索引：',nearest_neighbor_indices)
nearest_neighbor_genders = np.take(y_train, nearest_neighbor_indices)
print('3个近邻的标签：',nearest_neighbor_genders)
```

```
3个近邻的索引：  [6 8 2]
3个近邻的标签：  ['female' 'female' 'male']
```

例 kNN预测性别 | 代码

```
# 4. 预测
from collections import Counter
b = Counter(np.take(y_train, distances.argsort()[:k]))
print('预测结果:',b.most_common(1)[0][0])
```

```
预测结果: female
```

```
# 5. 绘图显示结果
for i, x in enumerate(X_train):
    plt.scatter(x[0], x[1], c='k', marker='x' if y_train[i] == 'male' else 'D')
for i in nearest_neighbor_indices:
    plt.scatter(X_train[i,0],X_train[i,1], c='k', s=200,marker='x' if y_train[i] == 'male' else 'D')
plt.scatter(175, 75, c='r', s=200,marker='o')
plt.title('分性别的身高体重图')
plt.xlabel('身高（cm）')
plt.ylabel('体重（kg）')
plt.grid(True)
plt.show()
```

1.如果k=5呢?

2.如果距离度量用其他方法呢?

补充内容 numpy.take()

numpy.take(a, indices, axis=None)： 从数组a中获取indices指定的元素，沿axis指定轴。

参数：a是源数组或列表。indices 是a中待提取的值对应的索引序列。

```
import numpy as np
a = ['setosa','setosa','virginica','versicolor','setosa','virginica']
indices = [0, 1, 3]
np.take(a, indices)
```

```
array(['setosa', 'setosa', 'versicolor'], dtype='<U10')
```

补充内容 Collections.Counter()

Counter(a): Python的Collections模块中**字典子类**，对一个序列中元素出现频率进行计数。

参数 a 是字符串、列表或数组。

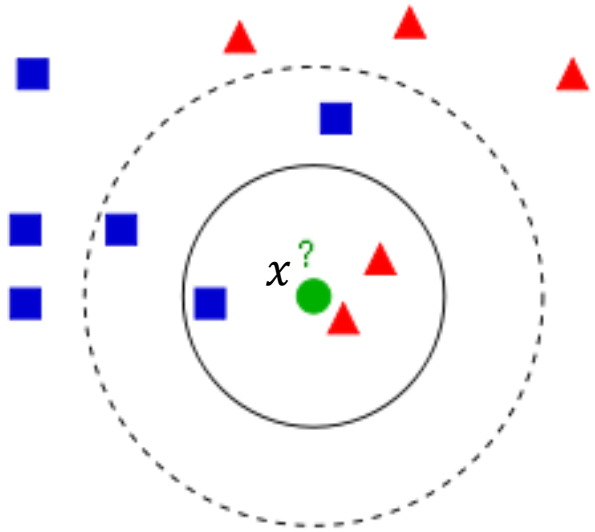
常用方法：most_common([n])，返回一列表，返回n个访问频率最高的元素和频数。

```
from collections import Counter

a =['setosa','setosa','setosa',
    'versicolor','versicolor','setosa',
 'versicolor','versicolor','virginica',
 'virginica','setosa','virginica',
 'ENGLISH CHARM','Clarence', 'Clarence']
c = Counter(a)  # 计算列表中每个元素出现的次数
c.most_common(3) # 3个次数最高的元素及其次数
```

```
[('setosa', 5), ('versicolor', 4), ('virginica', 3)]
```

```
c
```

```
Counter({'setosa': 5,
         'versicolor': 4,
         'virginica': 3,
         'ENGLISH CHARM': 1,
         'Clarence': 2})
```



如，绿圆点 x是一个新数据点，

当k=3时，判定它的类别是红三角；

而当k=5时，判定它的类别属于蓝方块。

- k是影响k近邻算法的一个关键因素。
- k是一个**超参数**，它不能通过训练数据学习出来，一般需要人为指定。
- 超参数k通常设置为奇数，以防止出现平局现象。

常用的距离计算方法：L2距离 和 L1距离

设$\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)}$为两个样本，每个样本有$d$个特征。

L2 (Euclidean) 距离(也称为L2范数)

$$L_2\left(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)}\right) = \sqrt{\sum_{t=1}^{d}\left(\boldsymbol{x}_t^{(i)} - \boldsymbol{x}_t^{(j)}\right)^2}$$

L1 (Manhattan) 距离(也称为L1范数)

$$L_1\left(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)}\right) = \sum_{t=1}^{d}\left|\boldsymbol{x}_t^{(i)} - \boldsymbol{x}_t^{(j)}\right|$$

闵可夫斯基（Minkowski）距离

$$\left(\sum_{t=1}^{d}\left|\boldsymbol{x}_t^{(i)} - \boldsymbol{x}_t^{(j)}\right|^p\right)^{1/p}$$

例 L1和L2距离比较

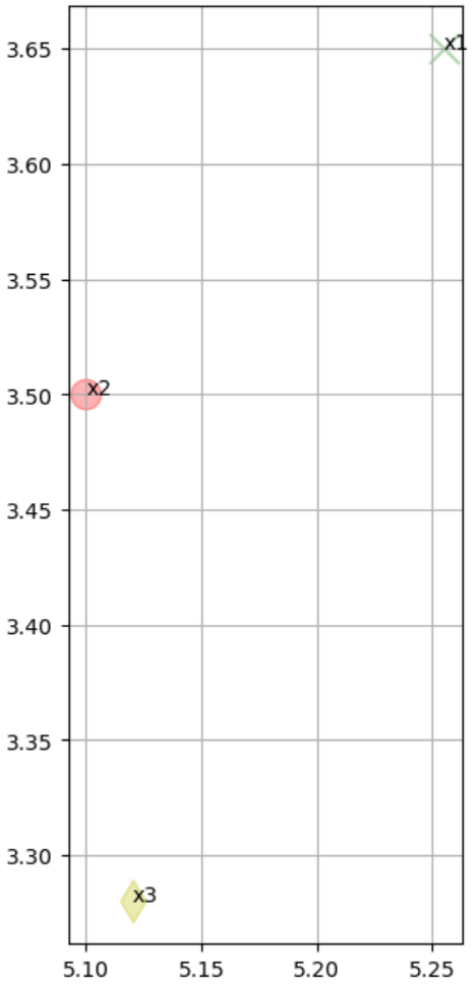
$$\boldsymbol{x}^{(1)}=\begin{pmatrix}5.255\\3.65\end{pmatrix}\quad \boldsymbol{x}^{(2)}=\begin{pmatrix}5.1\\3.5\end{pmatrix}\quad \boldsymbol{x}^{(3)}=\begin{pmatrix}5.21\\3.28\end{pmatrix}$$

分别用L2、L1计算样本$\boldsymbol{x}^{(1)}$到$\boldsymbol{x}^{(2)}$距离，$\boldsymbol{x}^{(2)}$到$\boldsymbol{x}^{(3)}$距离。

有什么发现?

$$\begin{cases}L2\left(\boldsymbol{x}^{(1)},\boldsymbol{x}^{(2)}\right)=\sqrt[2]{(5.255-5.1)^2+(3.65-3.5)^2}=0.2157\\ L2\left(\boldsymbol{x}^{(2)},\boldsymbol{x}^{(3)}\right)=\sqrt[2]{(5.1-5.21)^2+(3.5-3.28)^2}=0.2209\end{cases}$$

$$\begin{cases}L1\left(\boldsymbol{x}^{(1)},\boldsymbol{x}^{(2)}\right)=|5.255-5.1|+|3.65-3.5|=0.305\\ L1\left(\boldsymbol{x}^{(2)},\boldsymbol{x}^{(3)}\right)=|5.1-5.21|+|3.5-3.28|=0.24\end{cases}$$



已知训练集（Xtrain,ytrain）数据，近邻个数k，距离测度

输入：待预测的新数据Xtest

1个或多个测试实例

过程：

1. 根据给定的距离度量方法，计算训练集中的每个点与新数据点之间的距离；
2. 按距离递增次序排序；
3. 确定前k个点所在类别的出现频率
4. 返回前k个点出现频率最高的类别，作为新数据点的预测类别。

输出：新数据的预测标签ypred

kNN算法实现

根据kNN算法，设计knn()函数，参数：
训练集X和y、测试数据和个数、k和距离测度(L1或L2)

代码保存为python文件，文件名：my_kNN.py

```
import numpy as np
from collections import Counter

def knn(Xtrain,ytrain,Xtest,num_test_sample,k=3,measure='L2'):
    ypred = []  # 存放预测结果
    for i in range(num_test_sample):  # 遍历所有测试样本
        if measure=='L2':
            distances = np.sqrt(np.sum((Xtrain-Xtest[i,:])**2, axis=1)) # L2距离
        elif measure =='L1':
            distances = np.sum(np.abs(Xtrain - Xtest[i,:]), axis = 1)  # L1距离
        nearest_neighbor_indices = distances.argsort()[:k] # 返回前k个样本的索引
        nearest_neighbor_types = np.take(ytrain,nearest_neighbor_indices)  # k个近邻的类别
        k_neighbor = Counter(nearest_neighbor_types)  # 确定k个近邻类别的出现频次
        ypred.append(k_neighbor.most_common(1)[0][0])  # 获取频次最高的类标签
    return ypred
```


代码：my_kNN.py

（1）考察k值的影响

固定距离测度（如选L2距离），分别训练k=1，k=3两个近邻分类器，并评估他们。

关键步骤：

Step1 划分数据集 随机抽30个做测试（代码见第8页）

Step2 **训练分类器** (kNN实质上没做训练)

Step3 **预测并评估**

例3.1 鸢尾花分类|（1）考察k值的影响

Step3 **预测并评估**

使用accuracy=正确预测样本数/总样本数，来衡量预测的准确率。

```
from my_kNN import knn  # 导入自定函数knn
ypred = knn(Xtrain,ytrain,Xtest,num_test,k=1,measure='L2') #令k=1，L2范数
print('L2距离 k=1 近邻分类器的accuracy: %f' % ( np.mean(ypred == ytest) ))
```

```
L2距离 k=1 近邻分类器的accuracy: 0.933333
```

```
ypred = knn(Xtrain,ytrain,Xtest,num_test,k=3,measure='L2') #令k=3，L2范数
print ('L2距离 k=3近邻分类器的accuracy: %f' % ( np.mean(ypred == ytest) ))
```

```
L2距离 k=3近邻分类器的accuracy: 0.966667
```

说明：k值选取对kNN分类结果确有影响

代码：例3.1 鸢尾花分类-2023春v2.ipynb

(2) 考察**距离测度**的影响

修改（1）中代码，固定k（如 k=3），分别训练**L1距离**的3近邻分类器 和**L2距离**的3近邻分类器，比较它们在测试集上的分类准确率。

Step1 划分数据集　同“例3.1 鸢尾花分类 (1)”

Step2 **训练分类器**

Step3 **预测并评估**

例3.1 鸢尾花分类 | （2）考察距离测度的影响

只需在调用knn()函数时，修改距离测度参数：

```
from my_kNN import knn  # 导入自定函数knn
ypred = knn(Xtrain,ytrain,Xtest,num_test,k=3,measure='L1') #令k=3，L1范数
print('L1距离 k=3 近邻分类器的accuracy: %f' % ( np.mean(ypred == ytest) ))
```

```
L1距离  k=3近邻分类器的accuracy:  0.933333
```

```
ypred = knn(Xtrain,ytrain,Xtest,num_test,k=3,measure='L2') #令k=3，L2范数
print ('L2距离 k=3近邻分类器的accuracy: %f' % ( np.mean(ypred == ytest) ))
```

```
L2距离  k=3近邻分类器的accuracy:  0.966667
```

说明：距离测度选取对kNN分类结果也有影响

代码：例3.1 鸢尾花分类-2023春v2.ipynb

小结

- K近邻分类：K个近邻多数投票
- 影响因素
- 超参数

3.1.3 分类性能评估

思考

例3.1 鸢尾花分类中，选择不同的k将训出不同的模型，且选择不同的距离测度也会产生不同的模型。

1. 若有**多个模型**可选，怎样从中**选**出最佳模型呢?
2. 对选出的模型，怎样**评估**其泛化性能呢?

- **评估一个模型**，可通过在测试集上测试模型，获得泛化误差的估计值。该值说明了模型在从未见过的实例上的性能。
- 模型在新数据上的错误率称为模型的**泛化误差**，在训练数据集上的错误率称为模型的**训练误差**。
  - 如果训练误差很低，但泛化误差很高，则说明模型对于训练数据**过拟合（overfitting）**。
  - 若训练误差就很高，说明模型对训练数据**拟合不足（underfitting，称为欠拟合）**，无法匹配训练数据。

例3.1鸢尾花近邻分类中，有多个模型，源于近邻法的k和距离测度这两个**超参数**。

超参数依赖于问题，必须试看哪个参数好。通过调节超参数，选择泛化性能最佳的一组参数，即所谓“调参”。

以不同的超参数组合训出不同的模型，然后评估这些模型在新数据上的分类效果，这称为**模型验证**，最后从中选择最优的一组超参数。**可能的做法有：**

(1) 选择在**全部数据**上效果最好的超参数。

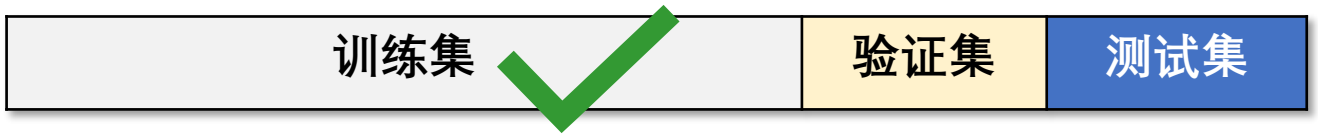
错误： 训练集数据已被看过，应在新数据上查看模型的性能。

(2) 将数据分为**训练集**和**测试集**，选择对测试数据最有效的超参数。

错误： 无法知晓模型在新数据上的表现。

(3) **留出法**：测试集不动的情况下，从训练集中留出一部分，以评估各候选模型并选择最佳模型（或最优超参数），新留出集称为验证集。

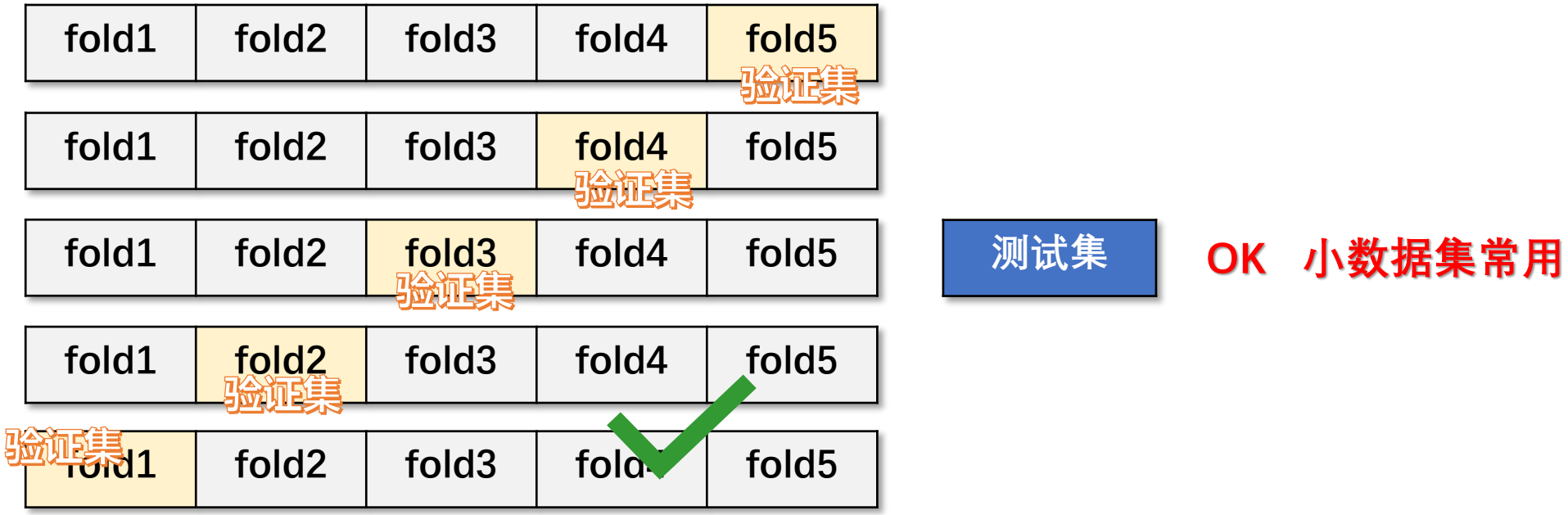
具体做法：首先在简化的训练集（即完整训练集减去验证集）上训练具有不同超参数的多个模型，选择在验证集上表现最佳的模型。然后，再用完整的训练集（包括验证集）训练最佳模型。最后，在测试集上评估这个模型，以获得泛化误差的估计值。



OK √ 数据量很大时用

缺点：在数据量小的情况下，若留出的验证集太小，则模型验证阶段的评估就将不精确，可能得到次优模型；若留出的验证集太大，则剩余训练集将比完整的训练集小得多，也不好。

(4) **交叉验证法（Cross-Validation）**：测试集不动的情况下，将训练数据拆分为多个子集（多折），尝试每折作为验证数据。每个模型都在其余数据上进行训练，在每个验证集上评估一次，并平均评估结果。通过对模型的所有评估求平均值，可更准确地衡量模型的性能。**缺点**是训练时间是验证集个数的倍数。



例3.1 鸢尾花分类 | (3) 超参数选择

(3) 用交叉验证法选择超参数k，找出最佳值。

① 尝试不同的k（k =1、3、5、7、…、39），分别交叉验证。

对每个k，创建一个kNN分类器，用交叉验证法在验证集上评估该分类器，计算平均分类准确率。

平均分类准确率=∑(每折分类准确率) /折数

② 选出使平均分类准确率最高的那个k。

采用最佳k, 在整个训练集上重新训练模型，测试其分类准确率。

例3.1 鸢尾花分类 | （3）超参数选择

Step1 划分数据集
Step2 超参数选择与交叉验证
Step3 最优模型训练与评估

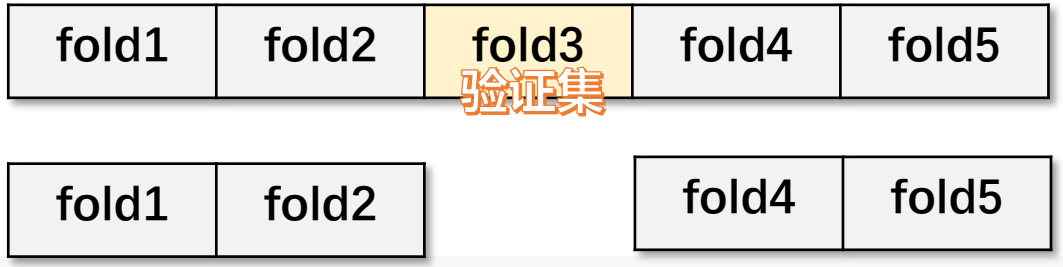
Step1 划分数据集

```
import pandas as pd
import numpy as np

iris = pd.read_csv('iris.data',names=['sepal_length','sepal_width',
                                      'petal_length', 'petal_width','species'])
data = iris.values  # 返回numpy数组
num_test_samples = 30  # 测试集样本数

np.random.seed(1)
np.random.shuffle(data)    # 打乱数据
test_set = data[:num_test_samples] # 定义测试集

data = data[num_test_samples:] # 取出测试集后，剩下数据用作超参数选择
```

代码：例3.1 鸢尾花分类-2023春v3.ipynb

例3.1 鸢尾花分类 | （3）超参数选择

Step2 超参数选择与交叉验证




```
from my_kNN import knn
cv = 5    # cv是交叉验证的折数
num_val_sample = len(data) // cv  # 验证集样本数目
validation_score=[] # 验证得分

for k in range(1,40,2):  # k是近邻个数，k分别取 1,3,5,7,9,...,39
    cv_score = []  # 交叉验证每折分类准确度得分
    for fold in range(cv):
        validation_set = data[num_val_sample*fold:num_val_sample*(fold+1)]  # 选出验证分区
        train_set = np.concatenate([data[:num_val_sample*fold], data[num_val_sample*(fold+1):]]) # 余下合并做训练
        Xtrain, ytrain=np.array(train_set[:, :4],dtype='float'), np.array(train_set[:, -1])
        Xval, yval=np.array(validation_set[:, :4],dtype='float'), np.array(validation_set[:, -1])
        Yval_pred = knn(Xtrain,ytrain,Xval,num_val_sample,k)  # 预测验证集数据的类标签
        cv_score.append(np.mean(Yval_pred == yval))

    validation_score.append((k, np.average(cv_score)))  # 平均验证得分
```

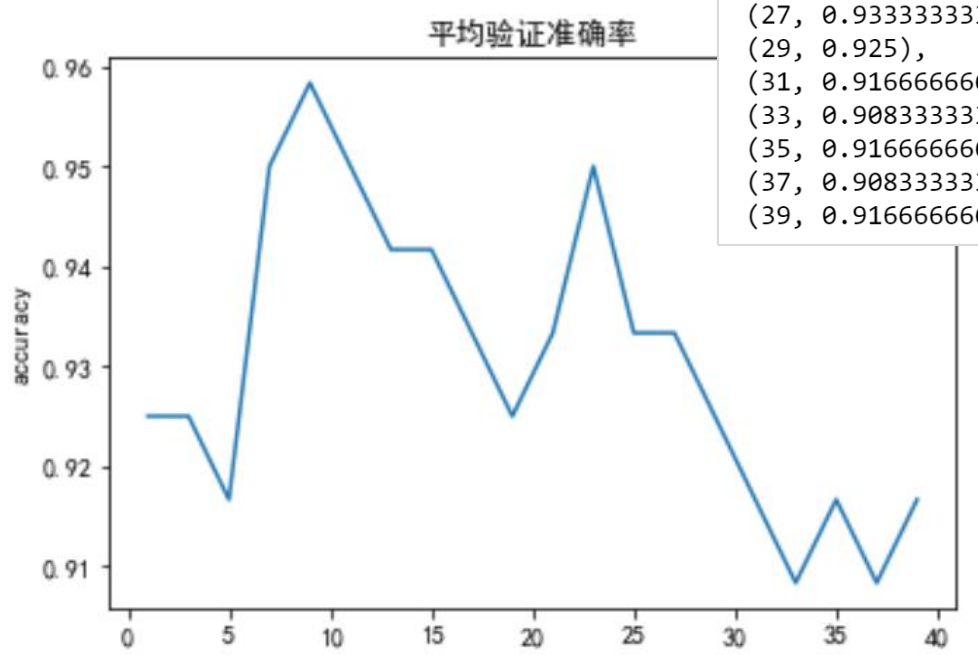
每轮交叉验证得分

代码：例3.1 鸢尾花分类-2023春v3.ipynb

例3.1 鸢尾花分类 | （3）超参数选择

用线形图显示k与验证得分的关系

```
%matplotlib inline
import matplotlib.pyplot as plt
x=[]; y=[]
for i in range(len(validation_score)):
    x.append(validation_score[i][0])
    y.append(validation_score[i][1])
plt.plot(x,y)
plt.xlabel('k')
plt.ylabel('accuracy')
plt.rcParams['font.sans-serif'] = ['SimHei']
plt.title('平均验证准确率')
plt.show()
```




```
validation_score
```

```
[(1, 0.925),
 (3, 0.925),
 (5, 0.9166666666666666),
 (7, 0.95),
 (9, 0.9583333333333334),
 (11, 0.95),
 (13, 0.9416666666666667),
 (15, 0.9416666666666667),
 (17, 0.9333333333333333),
 (19, 0.925),
 (21, 0.9333333333333333),
 (23, 0.95),
 (25, 0.9333333333333333),
 (27, 0.9333333333333333),
 (29, 0.925),
 (31, 0.9166666666666667),
 (33, 0.9083333333333334),
 (35, 0.9166666666666667),
 (37, 0.9083333333333334),
 (39, 0.9166666666666667)]
```

代码：例3.1 鸢尾花分类-2023春v3.ipynb

例3.1 鸢尾花分类（3）| 实现

Step3 最优模型训练与评估

```
# 选出最优K:对验证得分列表（每个元素是一个元组）按得分（元素的第二项）降序排序
sorted_score=sorted(validation_score,key=lambda d:d[1],reverse=True)
k = sorted_score[0][0]
print(k)
```

```
9
```

```
#令k=9，用整个训练集重新训练模型，并在测试集上评估
Xtrain = np.array(data[:,:4],dtype='float')
ytrain = np.array(data[:,-1])
Xtest = np.array(test_set[:,:4],dtype='float')
ytest = np.array(test_set[:,-1])

ypred = knn(Xtrain,ytrain,Xtest,num_test_samples,k)  # 预测验证集数据的类标签
print ('L2距离,k=%d 近邻分类器的accuracy: %f' % (k,np.mean(ypred == ytest) ))
```

```
L2距离,k=9 近邻分类器的accuracy: 0.966667
```


代码：例3.1 鸢尾花分类-2023春v3.ipynb

补充内容

sorted(*iterable*, *key*= *None*, *reverse*= *False*)

从可迭代对象（iterable）中的项，返回一个新的排序列表。

参数：iterable是可迭代对象。reverse设置是否逆序。

key：指定一个带参数的函数，用于从iterable中的每个元素中提取比较键。默认值为None（直接比较元素）。

```
lst = [(1,0.9),(3,0.92),(5,0.95),(7,0.91)]
sorted_lst = sorted(lst,key=lambda d:d[1],reverse=True)
sorted_lst
```

```
[(5, 0.95), (3, 0.92), (7, 0.91), (1, 0.9)]
```

样本的类标签与预测类别相同，即为**正确预测**。准确率是被正确预测标签的样本所占比例。若一种病的发病率为5%，有一个分类器对此预测准确率为95%，这个分类器如何?

- **整体准确率的缺陷：**当不同类别预测错误的影响不同时，准确率不能反映这种差别。
- **混淆矩阵(Confusion Matrix)** **评估分类性能的更好方法**

总体思路是统计A类别实例被分成B类别的次数。

- 独立地考察模型对某个类别（标签）分类能力。将被考察的类视为正类(Positive)，其他为负类(Negative)，混淆矩阵表示某个标签在验证/测试集中真实数量与预测数量的关系。

预测 / 真实	预测为正类	预测为负类
真实是正类	TP	FN
真实是负类	FP	TN

TP：是真正类的样本数。
FP：是假正类的样本数。
TN：是真负类的样本数。
FN：是假负类的样本数。

基于混淆矩阵可计算：准确率(accuracy)、精确率(precision)、召回率(recall)、F1分数(F1score)。

- **准确率**：$\text{Accuracy}=\frac{TP+TN}{TP+TN+FP+FN}$ 所有样本中被正确预测的比例。 整体准确率
- **精确率**：$\text{Precision}=\frac{TP}{TP+FP}$ 预测为正类的样本中正确预测的比例，也称**精度**。
- **召回率**：$\text{Recall}=\frac{TP}{TP+FN}$ 真实为正类的样本中被正确预测出来的比例，也称灵敏度。
- **F1分数**：$\text{F1}=\frac{2}{\frac{1}{\text{精确率}}+\frac{1}{\text{召回率}}}=\frac{2TP}{2TP+FP+FN}$ **是精度和召回率的调和均值.**

关注被考察类别

- ✓ 一个好的分类器，要具有良好的精度和召回率。
- ✓ F1score是精度和召回率的合成指标，是比较两个或多个分类器的直接简单方法。
- ✓ 只有当召回率和精度都很高时，分类器才能得到较高的F1分数。
- ✓ 有时我们更关心精度，另一些情况则更注重召回率，要权衡。

思考

1. 测试集有500个样本，其中真实类别为正类的样本有50个。现用一个模型对测试集进行预测，它预测为正类的样本有60个，其中真正类有45个，该模型的召回率是（ ）%，精度是（ ）%。

2. 近邻分类法训练快还是预测快？

小结

- **超参数选择、模型选择的常用方法：k折交叉验证**
- **分类性能评估：**

  准确率、精度、召回率、混淆矩阵、分类性能报告

近邻分类法的性能

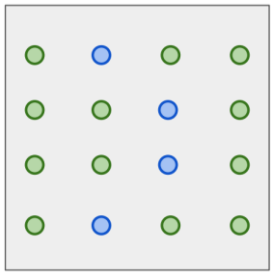
❑ **预测效率低**

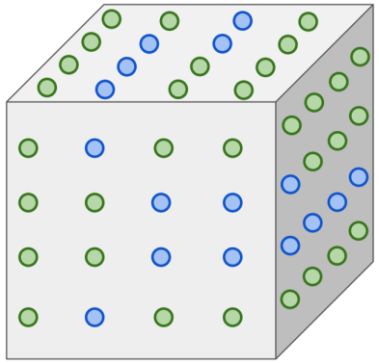
- 训练时，只需记住样本，快！
- 预测时，需计算待测样本与每个训练样本的距离，若有N个训练样本，就需N次距离计算，慢！

我们希望分类器在**预测时快**；训练时为获得好模型，慢些可以！如图像分类等高维数据分类问题，一般不用kNN。

❑ **对高维、稀疏数据集预测效果差**

- 维数灾难





3.1.4 利用Scikit-learn

Scikit-learn 涵盖几乎所有主流机器学习算法，提供了算法的高效实现。

主要特点：

1. 为**数据预处理**、**模型训练**、**优化和评估**提供一致调用接口。
2. 提供常用**数据集加载函数**；

   提供用于合成**数据集的生成器**；

   提供从第三方公开**数据集下载数据的函数**。
3. 用**Pipelines** 将所有机器学习工作流串到一起
4. 有自己的绘图 API，**可绘制混淆矩阵**等。

再谈　例3.1 鸢尾花分类

例3.1　某地区有三种鸢尾花（virginica、Setosa、Versicolor），已收集一些数据，请设计一个模型，给定鸢尾花的特征能自动识别它的种类。

1. 整理数据
2. 训练分类器
3. 评估分类器

回顾 Pandas读取csv文件

Step1 Pandas读csv文件，存入一个DataFrame对象中；

Step2 拆分数据集，取其中特征信息存入特征矩阵X，取标签信息存入目标数组y中。

X和y是Pandas的数据结构

```
import pandas as pd
iris=pd.read_csv('iris.csv')
X = iris.drop('species',axis = 1) # X是DataFrame对象
y = iris['species'] # y是一个Series对象
```


代码：sklearn读iris数据.ipynb

利用sklearn的**数据加载函数**

sklearn.datasets中内置了多个常用数据集，提供相应函数来加载数据集。

load_iris() 可加载iris数据集，使用前需导入：

```
from sklearn.datasets import load_iris
```

语法： ***load_iris(return_X_y = False)***

参数:

return_X_y：默认为False，返回一个类似字典的bunch对象。

若为True，则返回 (X，y) 元组，其中X, y都是numpy.Ndarray类型。

Bunch类型

- Sklearn提供的一种类似字典的数据结构Bunch。
- Bunch 对象常用作sklearn的数据函数的返回值。

bunch对象的主要属性：

- data：特征矩阵。类似ndarray类型（二维）
- target：类标签，即目标数组。 类似ndarray类型
- target_names：标签的涵义。 ndarray类型
- feature_names：特征的涵义。 list类型
- DESCR：数据集的完整描述。string类型

如，

```
iris = load_iris()
X = iris.data    # 特征矩阵
y = iris.target # 目标数组
```

例 用sklearn的load_iris()获取iris数据

```
from sklearn.datasets import load_iris
iris = load_iris() # 返回bunch对象

X = iris.data    # 抽取特征矩阵,返回ndarray
print("X的形状:",X.shape) # 输出?
print("X的类型:",type(X)) # 输出?
```

```
y = iris.target # 获取目标数组,返回ndarray
print("y的形状:",y.shape) # 输出?
print("y的类型:",type(y)) # 输出?
```

利用sklearn的**数据下载函数**

sklearn.datasets包中fetch_openml函数，可下载流行的数据集到本地缓存。

使用前需导入：**from sklearn.datasets import fetch_openml**

语法： **fetch_openml(*name=None, version='active'*)**

按名称从openml.org下载数据集

参数:

① name：string，数据集的名称。注意，OpenML可以有多个同名数据集。

② version : int, 或 'active'（默认,最活跃版本），数据集的版本。若数据集有多个版本，必须用int设置准确的版本。

返回值：默认返回一个Bunch对象。

利用fetch_openml获取iris数据

```
from sklearn.datasets import fetch_openml

# 从https://www.openml.org/data/v1/download/61/iris.arff下载
iris = fetch_openml('iris',version=1,parser='auto') # 数据若有多个版本就必须提供version

print(type(iris))
```

```
<class 'sklearn.utils._bunch.Bunch'>
```

```
X = iris.data    # 返回ndarray
print("X的类型:{}; 形状:{}".format(type(X),X.shape))
y = iris.target   #返回ndarray
print("y的类型:{}; 形状:{}".format(type(y),y.shape))
```

```
X的类型:<class 'pandas.core.frame.DataFrame'>; 形状:(150, 4)
y的类型:<class 'pandas.core.series.Series'>; 形状:(150,)
```

sklearn.model_selection中的train_test_split()，将数据集划分成训练集和测试集。使用前先导入 from sklearn.model_selection import train_test_split

train_test_split(*X, y, test_size=None, random_state=None,shuffle=True*)

参数	• X：待拟合的特征矩阵。类型可以是NumPy数组、Panadas Dataframes等 • y：目标向量(监督学习)。类型可以是List、NumPy数组、panadas Series • test_size：float 或 int，测试集大小。默认为0.25 • random_state：int，随机数种子 • shuffle：拆分前是否洗牌。默认为True，即随机采样
返回值	返回输入数据集划分结果列表，包括训练X、测试X、训练y、测试y。类型与输入同

如 X_train,X_test,y_train,y_test = train_test_split(X,y,random_state=0)

- 任何能够基于数据集估计某些参数的对象都被称为估计器 (estimator)。
- Scikit-learn为所有估计器提供一致的API，其使用流程如下：

(1) **选择模型类**：从Scikit-learn中导入所选模型类（统称estimator）。

(2) **创建模型**：新建一个模型对象，配置超参数。

(3) **训练模型**：调用模型对象的`fit()`对数据进行拟合。

(4) **评估模型**：调用模型对象的`score()`或 metrics模块的性能评估方法，评估性能。

(5) **应用模型**：对新数据应用模型

- 通常使用模型对象的`predict()`预测新数据的标签；
- 有些估计器用于变换数据集，这些称为**变换器**，除`fit()`外，其API还有`transform()`、`fit_transform()`。

在sklearn.neighbors模块下，用于分类的KNN估计器是KNeighborsClassifier，

使用前需导入： `from sklearn.neighbors import KNeighborsClassifier`

KNeighborsClassifier(*n_neighbors=k*)

超参数 n_neighbors=k 用来设置近邻个数，默认为5。

主要方法： *假设 knn = KNeigborsClassifier(3)*

- **knn.fit(X_train,y_train)**：用训练集数据(X_train,y_train)训练模型。X_train,y_train 都是array-like类型。
- **knn.predict(X_new)**：预测新数据X_new的类别。array-like类型。
- **knn.score(X_test,y_test)**：返回模型对测试数据(X_test,y_test)的分类准确率。
- **knn.kneighbors(X, n_neighbors, return_distance)**：找X的k个近邻，返回k个近邻索引,及其与X的距离。若 return_distance=False，则不返回距离，其默认值为True。

例3.1 鸢尾花分类 | (4) 利用KNeighborsClassifier

(4) 用scikit-learn的KNeighborsClassifier类，为iris数据创建一个kNN分类器（k取3），训练后评估其分类准确率。

关键步骤:

1）整理数据，划分数据集（X,y）为训练集和测试集

2）创建一个kNN分类器，k取 3

3）训练分类器，拟合训练数据

4）评估分类器，计算它在测试集上的分类准确率

例3.1 鸢尾花分类（4）| 实现

1）整理数据

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
iris = load_iris() # 返回bunch对象
X = iris.data    # 抽取特征矩阵,返回ndarray
y = iris.target # 获取目标数组,返回ndarray
Xtrain,Xtest,ytrain,ytest = train_test_split(X,y,test_size=0.3,
                                             random_state=3)
```


代码：例3.1 鸢尾花分类-2023春v4.ipynb

例3.1 鸢尾花分类（4）| 实现

2）建立、训练、评估模型

```
from sklearn.neighbors import KNeighborsClassifier # 1.选择模型类
model = KNeighborsClassifier(3)                    # 2.初始化模型
model.fit(Xtrain,ytrain)                           # 3.用模型拟合数据
score = model.score(Xtest,ytest)                   # 4.评估模型
print("测试集分类准确率：%.4f" % score)
```

测试集分类准确率：0.9556

K怎么取?

Scikit-learn有支持交叉验证的方法吗?

代码：例3.1 鸢尾花分类-2023春v4.ipynb

在sklearn.**model_selection**模块下，提供了**cross_val_score**交叉验证评分函数，执行k折交叉验证，返回每折的验证得分。

cross_val_score(*estimator, X, y, cv=None, ...*)	
参数	• estimator：用于拟合数据的对象的估计器 • X：list 或 array-like；待拟合的特征矩阵 • y：array-like；目标向量（监督学习） • cv：int（折数）或None, 或交叉验证产生器；用于确定交叉验证的划分策略。如果为int或None, 采用分层交叉验证。取默认值None时，cv=5。
返回值	得分数组：实型；每折交叉验证的得分构成的数组。

例3.1 鸢尾花分类（4）| 实现

新增功能： 用5折交叉验证得分的均值，来更稳定地评估模型的分类准确率。

```
from sklearn.model_selection import cross_val_score
Xtrain,Xtest,ytrain,ytest = train_test_split(X,y,test_size=0.3,random_state=3)
knn = KNeighborsClassifier(3)
cv_result = cross_val_score(knn, Xtrain, ytrain, cv=5)
print("5折交叉验证得分",cv_result)
print("交叉验证得分均值:%.4f"%cv_result.mean())
```

```
5折交叉验证得分 [0.9047619  1.         0.95238095 1.         0.95238095]
交叉验证得分均值:0.9619
```

例3.1 鸢尾花分类 | （5）利用cross_val_score选最优

(5) 令近邻个数k分别为1,3,5,7,...,39，用交叉验证评分评估各个近邻分类器，选出均分最高的超参数k。然后，以这个k创建一个新的近邻分类器，并测试其分类准确率。最后，用新分类器预测一朵新鸢尾花（花萼长5cm宽2.9cm，花瓣长1cm宽0.2cm）所属类别。

关键步骤:

1）整理数据，划分数据集。与“例3.1鸢尾花分类（4）”相同

2）超参数调节 k分别取1,3,⋯,39，利用交叉验证评分函数选最优的k。

3）以最优k为参数，重新训练一个分类器，测试其分类准确率。

4）用新分类器预测（5, 2.9, 1, 0.2）所属类别。

例3.1 鸢尾花分类（5）| 实现

1) 数据整理与划分　（略）

2) 建立模型、调交叉验证函数训练模型

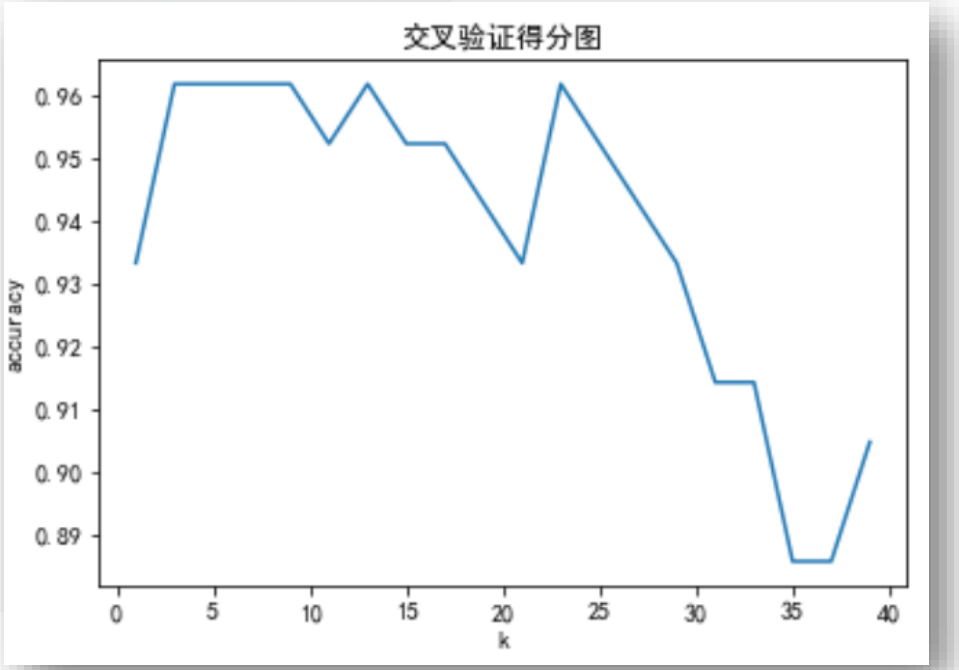
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import cross_val_score

results=[] # 存放 (k,cv得分均值) 的列表
for k in range(1,40,2):  # k是近邻个数，k分别取 1,3,5,7,9,...,39
    model = KNeighborsClassifier(k)
    cv_result=cross_val_score(model,Xtrain,ytrain,cv=5)
    results.append((k,cv_result.mean()))
```


代码：例3.1 鸢尾花分类-2023春v5.ipynb

例3.1 鸢尾花分类（5）| 实现

3) 画出k与分类准确率之间关系的线形图

```
%matplotlib inline
import matplotlib.pyplot as plt

plt.rcParams['font.sans-serif'] = ['SimHei']   # 为汉字显示正常
plt.rcParams['axes.unicode_minus'] = False

x=[]; y=[]
for i in range(len(results)):
    x.append(results[i][0])
    y.append(results[i][1])
plt.plot(x,y)
plt.xlabel('k')
plt.ylabel('accuracy')
plt.title('交叉验证得分图')
```



代码：例3.1 鸢尾花分类-2023春v5.ipynb

例3.1 鸢尾花分类（5）| 实现

4) 选出得分最高的k，创建最优模型

```
# 对验证得分列表（每个元素是一个元组）按得分（元素的第二项）降序排序
sorted_score=sorted(results,key=lambda d:d[1],reverse=True)
k = sorted_score[0][0]

best_model = KNeighborsClassifier(k)
best_model.fit(Xtrain,ytrain)

test_score = best_model.score(Xtest,ytest)
print("k = %d,   test accuracy = %.4f" % (k,test_score))
```

```
k = 9,   test accuracy = 0.9778
```


代码：例2.1 鸢尾花分类-2023春v5.ipynb

例3.1 鸢尾花分类（5）| 实现

5）预测新数据

```
# 预测新数据 (5, 2.9, 1, 0.2) 所属类别
import numpy as np
X =np.array([5.,2.9,1.,0.2])
X_new = X[np.newaxis,:]

best_model.predict(X_new)
```

新数据也要是（d1,d2）的形状

```
array(['setosa'], dtype=object)
```

代码：例3.1 鸢尾花分类-2023春v5.ipynb

sklearn.metrics中提供了多种分类性别评估的方法，包括：

准确率	accuracy_score(*y_true,y_pred*)
精确率	precision_score(*y_true,y_pred*)
召回率	recall_score(*y_true,y_pred*)
F1值	f1_score(*y_true,y_pred*)
混淆矩阵	confusion_matrix(*y_true,y_pred*)
分类性能报告	classification_report(*y_true,y_pred*)

参数： y_true：待预测量的真实值

y_pred：模型预测结果

例3.1 鸢尾花分类（5）| 实现

新增功能：6) 对上面训得的分类器，输出其测试集上的性能评估报告，打印其混淆矩阵。

```
from sklearn.metrics import classification_report

y_pred = best_model.predict(Xtest)
print('分类评估报告: \n', classification_report(ytest,y_pred))
```

```
分类评估报告:
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        17
  versicolor       1.00      0.93      0.96        14
   virginica       0.93      1.00      0.97        14

    accuracy                           0.98        45
   macro avg       0.98      0.98      0.98        45
weighted avg       0.98      0.98      0.98        45
```


代码：例3.1 鸢尾花分类-2023春v5.ipynb

打印其混淆矩阵

```
from sklearn.metrics import confusion_matrix
conf_mx = confusion_matrix(ytest,y_pred)
print(conf_mx)
```

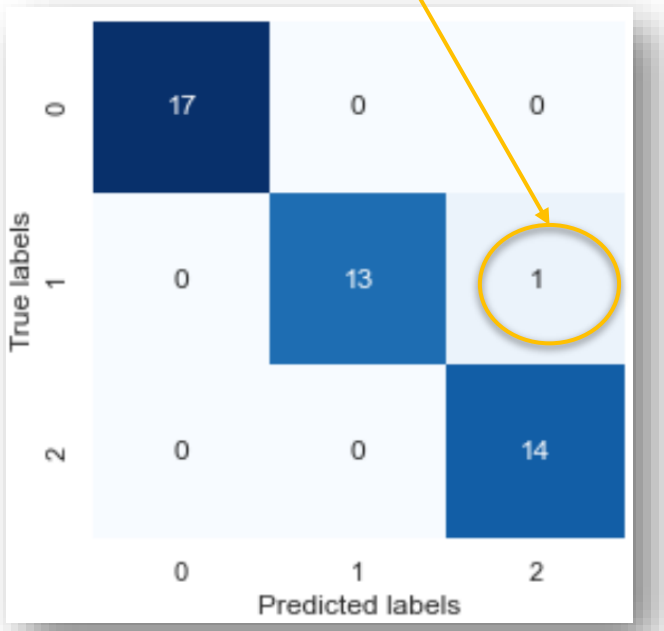
```
[[17  0  0]
 [ 0 13  1]
 [ 0  0 14]]
```

行代表真实类
列代表预测类

真实类别为1，预测为2的样本有1个

画出混淆矩阵

```
import matplotlib.pyplot as plt
import seaborn as sns;sns.set()
plt.figure(figsize=(4, 4))
sns.heatmap(conf_mx, annot=True, cmap='Blues',cbar=False)
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
```



思考

构建kNN分类器的主要步骤?

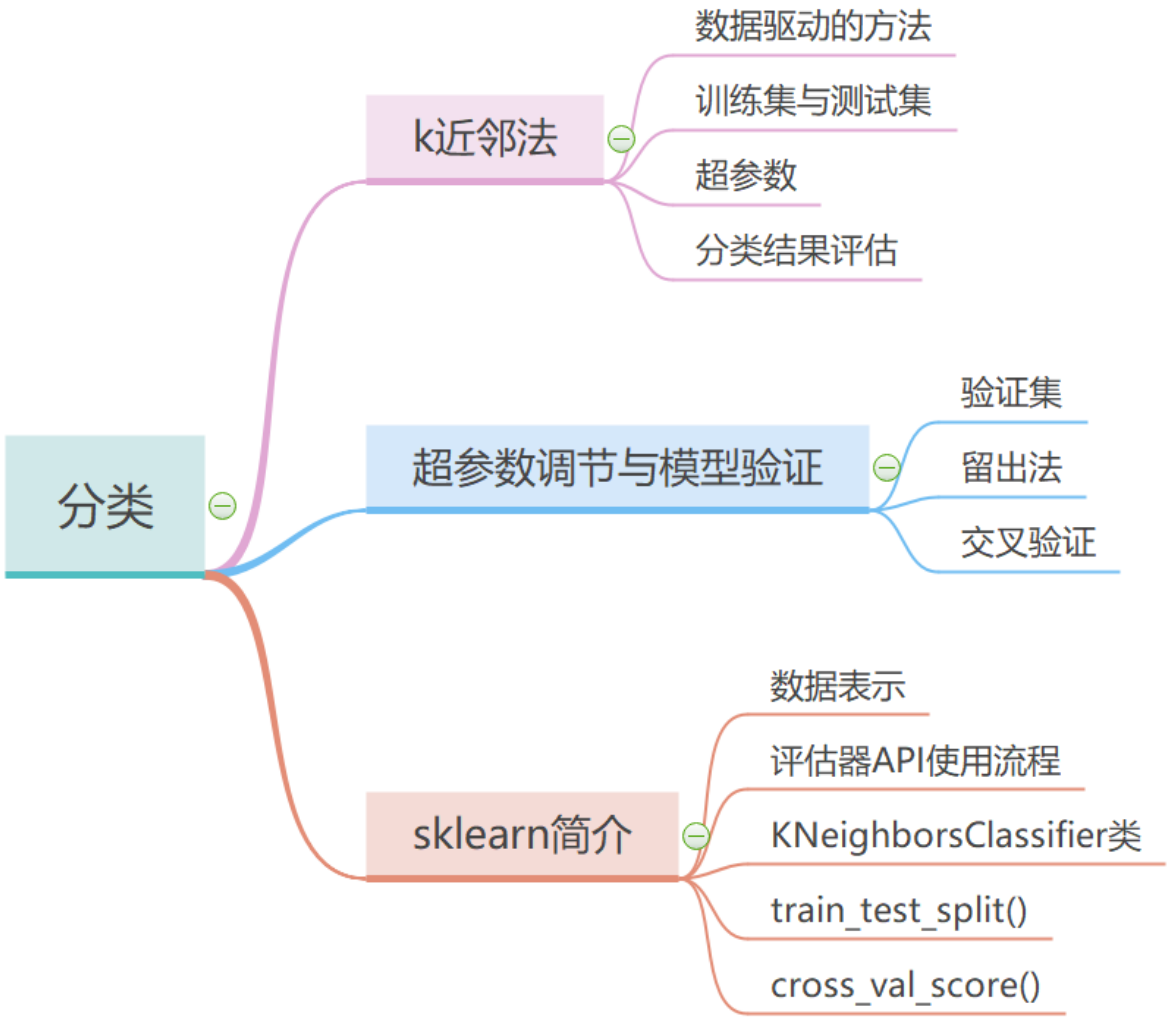
1. 整理数据，包括数据集拆分
2. 选择模型类，选择不同的k初始化模型，超参数k调整

   from sklearn.neighbors import KNeighborsClassifier

   knn=KNeighborsClassifier(n_neighbors=k)

   cross_val_score(knn, Xtrain,ytrain,cv=7)　选出最佳模型bestModel
3. 拟合数据，bestModel.fit(X_train,y_train)
4. 评估模型，bestModel.score(X_test,y_test)
5. 预测新数据，bestModel.predict(X_new)

小结


分类
k近邻法
数据驱动的方法
训练集与测试集
超参数
分类结果评估
超参数调节与模型验证
验证集
留出法
交叉验证
sklearn简介
数据表示
评估器API使用流程
KNeighborsClassifier类
train_test_split()
cross_val_score()

作业 3.1

1. 为什么要从整体数据集中取出部分留作测试呢?

2. **kNN及其在iris上的应用**

（1）重新编写knn函数（k=1），修改其中距离测度参数，传入闵可夫斯基距离的p。

（2）给定iris数据集，p分别取1、2、…、7，比较各种情况下kNN模型的分类准确率，从中选择最好的p;

（3）利用选出p，重新训练模型，测试其分类准确率。

作业 3.2

天气预测

用给定的天气测量数据训练一个KNN分类器，来预测天气类型。

(1) 读入weather.csv，以Description（天气类别: **warm,normal,cold**）列为目标，整理数据（X,y），划分数据集。

(2) 用sklearn的KNeighborsClassifier模型类，尝试k=1、3、5、7、9分别建立模型，采用cross_val_score函数验证各个模型，选出最优超参数k。

(3) 用最佳k重新训练模型，测试分类准确率、输出分类性能报告。

天气数据（Weather）简介

weather.csv是某城市10年间每小时的天气测量值。该数据集原有8个变量，本实验取其中6个变量。共有10000个观测值。

6个特征及其涵义如下表：

Temperature_c	Humidity	Wind_Speed_kmh	Pressure_millibars	Rain	Description
摄氏度	湿度比	风速,单位为km/h	大气压力，单位为MPa	下雨是1;下雪时0	值:warm,normal,cold

weather数据集的部分数据

Temperatu	Humidity	Wind_Spee	Pressure_n	Rain	Description
-0.55556	0.92	11.27	1021.6	0	Cold
21.11111	0.73	20.93	1017	1	Warm
16.6	0.97	5.9731	1013.99	1	Normal
1.6	0.82	3.22	1031.59	1	Cold
2.194444	0.6	10.8836	1020.88	1	Cold
27.53889	0.32	21.4613	1015.33	1	Warm
19.97778	0.84	7.9695	1009.04	1	Warm
11.11111	0.86	14.49	1009.6	1	Normal
8.405556	0.73	14.007	1018.39	1	Normal
1.7	0.81	6.44	1003.89	1	Cold