机器学习

什么是knn

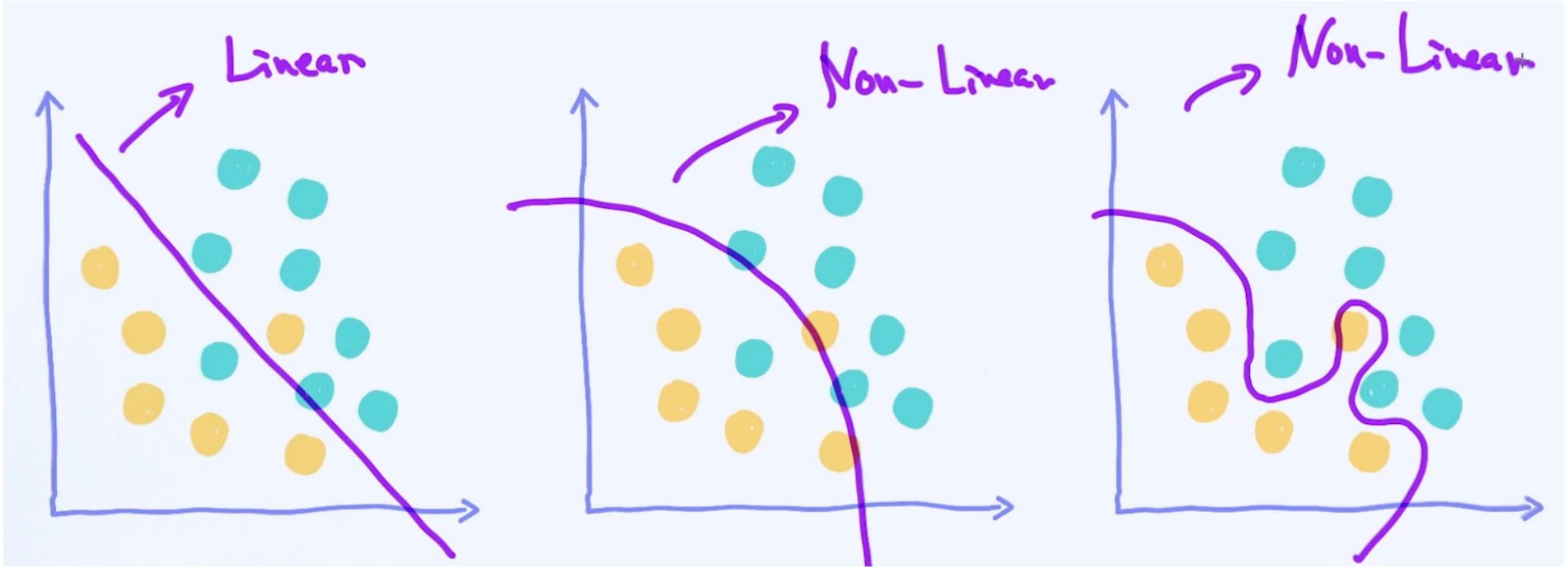
knn案例

决策边界

什么是决策边界呢？其实我们每天都做各式各样的决策，比如我的决策是：当华为Mate降价到2000元的时候购买一个。对于这个问题，我的决策边界是2000元，也就是大于2000元的时候我不会购买，但小于2000元时我会选择购买，类似的生活中的例子很多。

决策边界分成两大类，分别是线性决策边界和非线性决策边界。拥有线性决策边界的模型我们称为线性模型，反之非线性模型。

决策边界



后两个是非线性的决策边界。最后一个决策边界在数据上的表现是最好的。但这不代表未来数据上，也就是测试数据上表现最好。这里涉及到了一个很重要的概念，叫做**模型的泛化能力**，可以简单理解成“它在新的环境中的适应能力”，当然这个环境需要跟已有的环境类似才行。

模型泛化能力

举个例子，一个学生平时成绩非常好，都是年级第一。但一到重要的考试比如中考，高考就不理想。这其实说明这个学生举一反三能力，对新环境的适应能力比较弱。这就要求这名学生对知识点有个更深入的理解，而不仅仅停留在表面，或者题海战术。其实这些理念也适用于建模当中。

交叉验证与算法优化

对于KNN算法来说，K就是需要调整的超参数。对于一般初学者来说，你可能会尝试不同的值，看哪个值表现最好就选哪个。有一种更专业的穷举调参方法叫Grid Search，即在所有候选的参数中,通过循环遍历，尝试每一种可能性，表现最好的参数就是最终的结果。

交叉验证与算法优化

那么选用哪些数据集进行调参呢，我们来分析一下：↵

方法一：选择整个数据集进行测试。但这种方法有个非常明显的问题，就是设定 K=1 总是最好的，因为每个测试样本的位置总是和整个训练集中的自己最接近。如下图 3-11 所示：↵

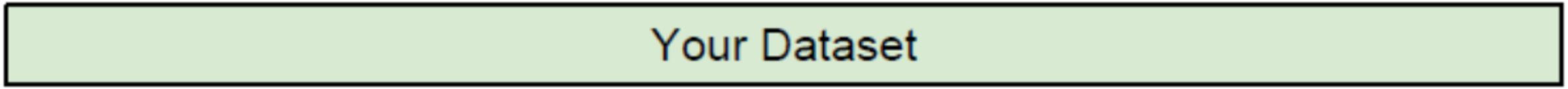


图 3-11 整个数据集↵

交叉验证与算法优化

方法二：将整个数据集拆分成训练集和测试集，然后在测试集中选择合适的超参数。这里也有一个问题，就是不清楚这样训练出来的算法模型对于接下来的新的测试数据表现会如何？如下图 3-12 所示：

图 3-12 整个数据集拆分成训练和测试集

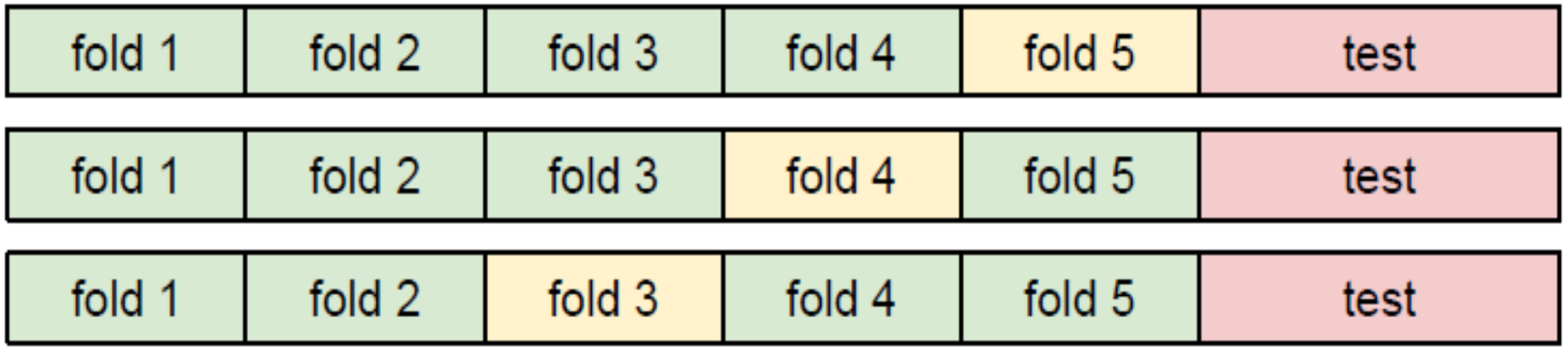
交叉验证与算法优化

方法三：将整个数据集拆分成训练集、验证集和测试集，然后在验证集中选择合适的超参数最后在测试集上进行测试。这个方法相对就比之前两种方法好很多，也是在实践中经常使用的方法。如下图 3-13 所示：↵

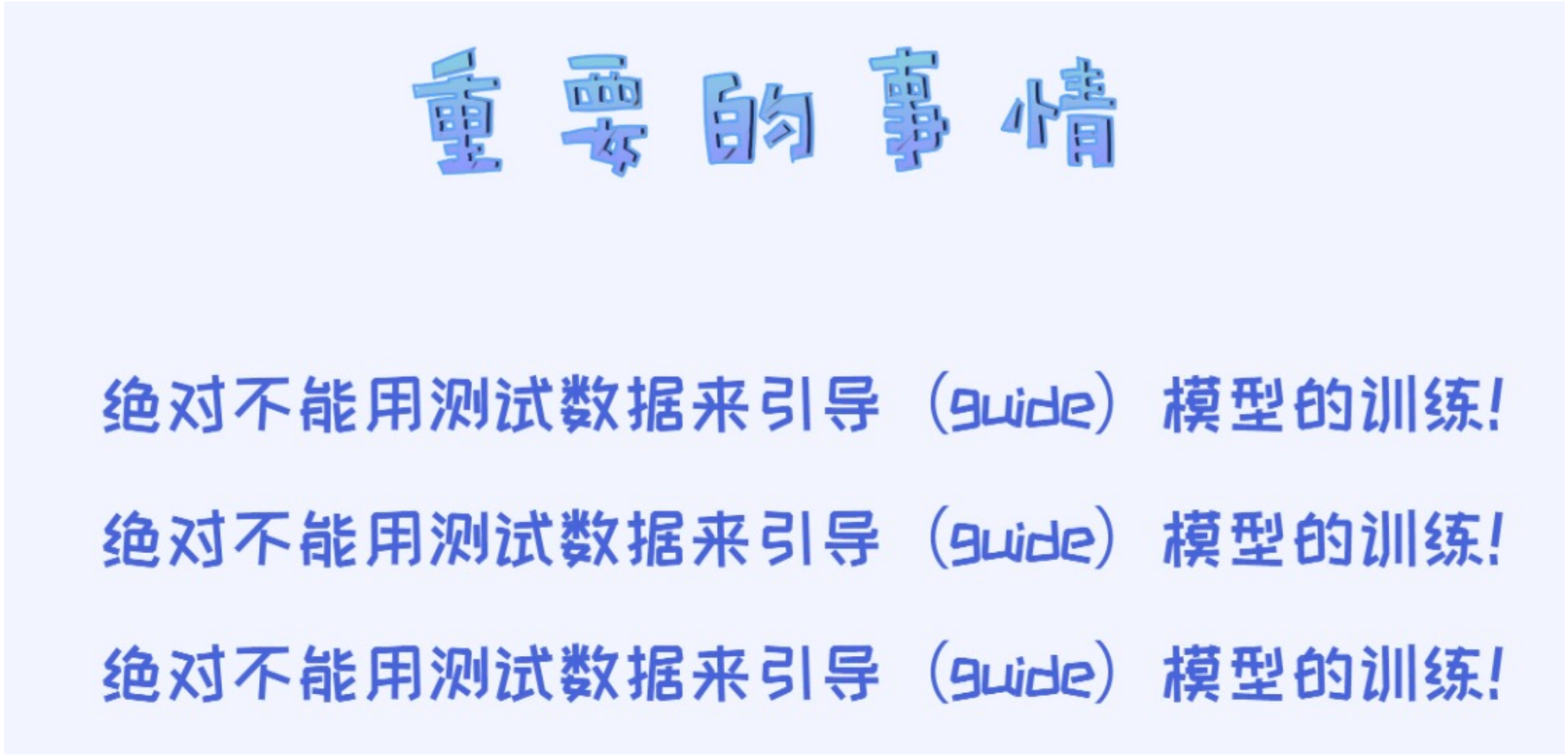
图 3-13 训练集、验证集和测试集↵

交叉验证与算法优化

方法四：使用交叉验证，将数据分成若干份，将其中的每一份作为验证集之后给出**平均准确率(每一次的验证得到的准确类进行累加，之后求平均)**，最后把评估得到的合适的超参数在测试集中进行测试。这个方法更加严谨，但实际中常在较小数据集上使用，深度学习很少使用。



注意事项


重要的事情
绝对不能用测试数据来引导（guide）模型的训练!
绝对不能用测试数据来引导（guide）模型的训练!
绝对不能用测试数据来引导（guide）模型的训练!

交叉验证与算法优化

对于KNN算法来说，K就是需要调整的超参数。对于一般初学者来说，你可能会尝试不同的值，看哪个值表现最好就选哪个。有一种更专业的穷举调参方法叫Grid Search，即在所有候选的参数中,通过循环遍历，尝试每一种可能性，表现最好的参数就是最终的结果。

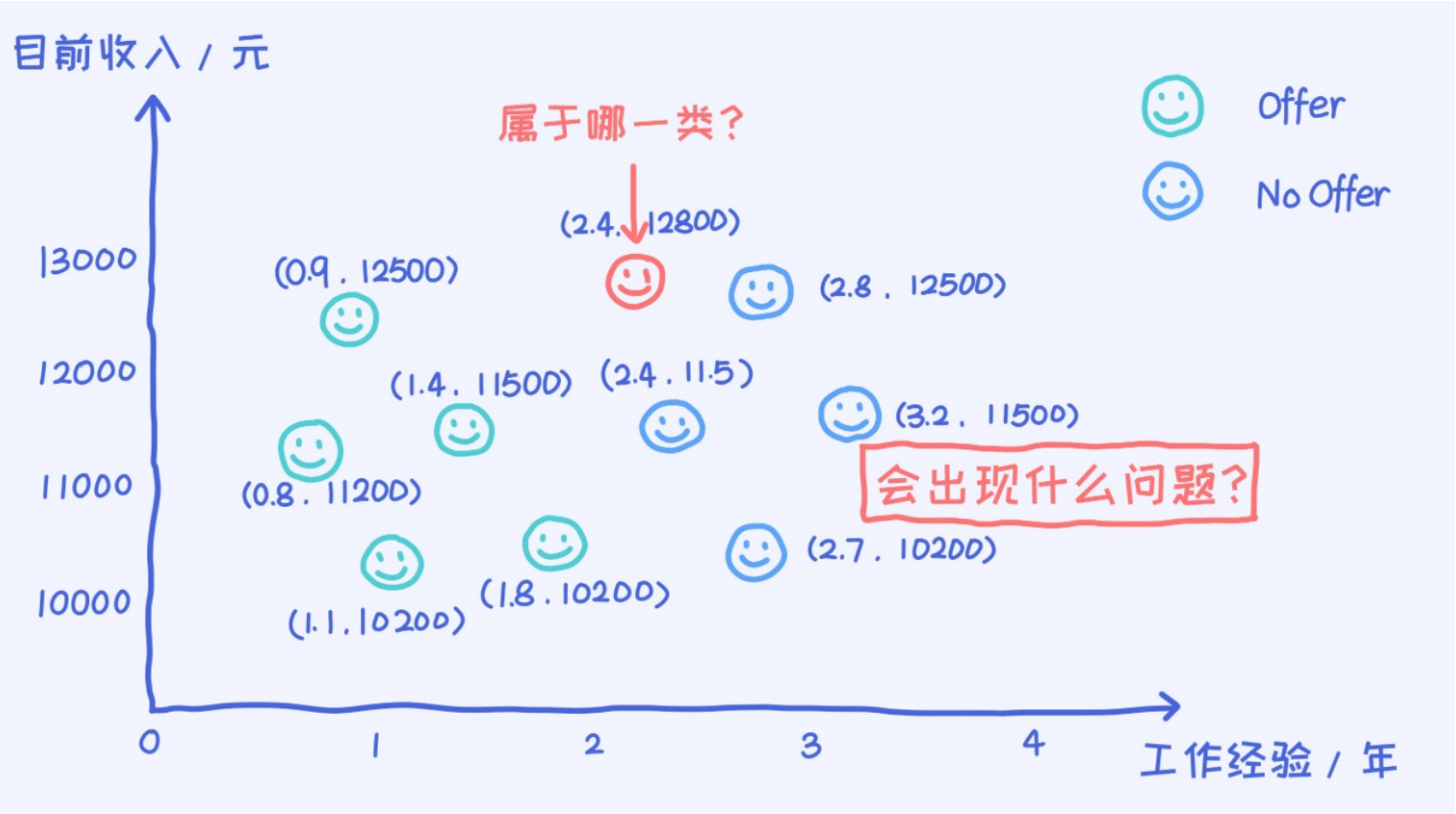
KFold和StratifiedKFold
都可以作为GridSearchCV里参数cv的输入

其实就是把数据集拆分成不同的训练集+验证集，如下下面的代码所示：

```
floder = KFold(n_splits=4, random_state=0, shuffle=False)
sfolder = StratifiedKFold(n_splits=4, random_state=0,
shuffle=False)
```

KFold：是对数据集顺序按顺序K折划分；
StratifiedKFold：则是分层采样，确保训练集，验证集中各类别样本的比例与原始数据集中相同；

特征缩放



距离计算过程中Y轴的影响倍增了很多，甚至工作经验基本上起不到什么作用了。

特征缩放

在使用KNN算法的时候，特征值上的范围的差异对算法影响非常大。

特征的缩放

1. 线性归一化（Min-max Normalization）

2. 标准差标准化（Z-score Normalization）

特征缩放 - 线性归一化
(Min-max Normalization)

助教经验
1.5
2
3
4
1.5
2.5
2
2.3
3
1.5

数值范围 = [1.5, 4]

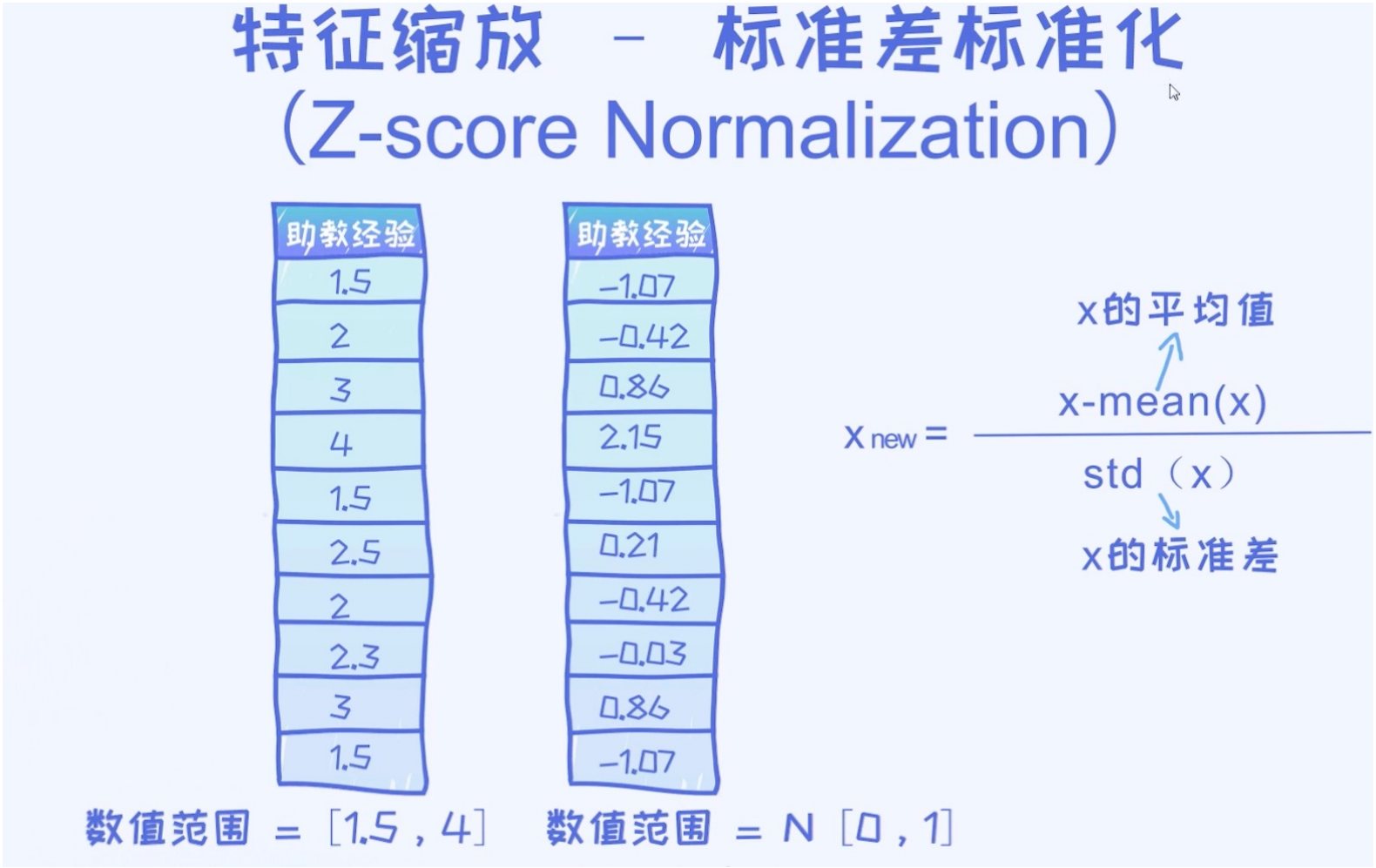
助教经验
0
0.2
0.6
1.0
0
0.4
0.2
0.33
0.6
0

数值范围 = [0, 1]

$$x_{new} = \frac{x - \min(x)}{\max(x) - \min(x)}$$

x的最小值

x的最大值



均值为0，标准方差为1的高斯分布

需要注意的一点是转换后的特征有负有正，而且它们的均值为0。

平均数：$M = \dfrac{x_1 + x_2 + x_3 + \cdots + x_n}{n}$ （n表示这组数据个数，x_1、x_2、x_3……x_n表示这组数据具体数值）

方差公式：$s^2 = \dfrac{(x_1 - M)^2 + (x_2 - M)^2 + (x_3 - M)^2 + \cdots + (x_n - M)^2}{n}$

具体哪一个方法更好呢？ 这其实很难说，一般情况下还是需要尝试，通过效果来得到结论。对于有些问题第一种方案可能比较好，在其他问题上第二种方案或许更好，所以不一定的。

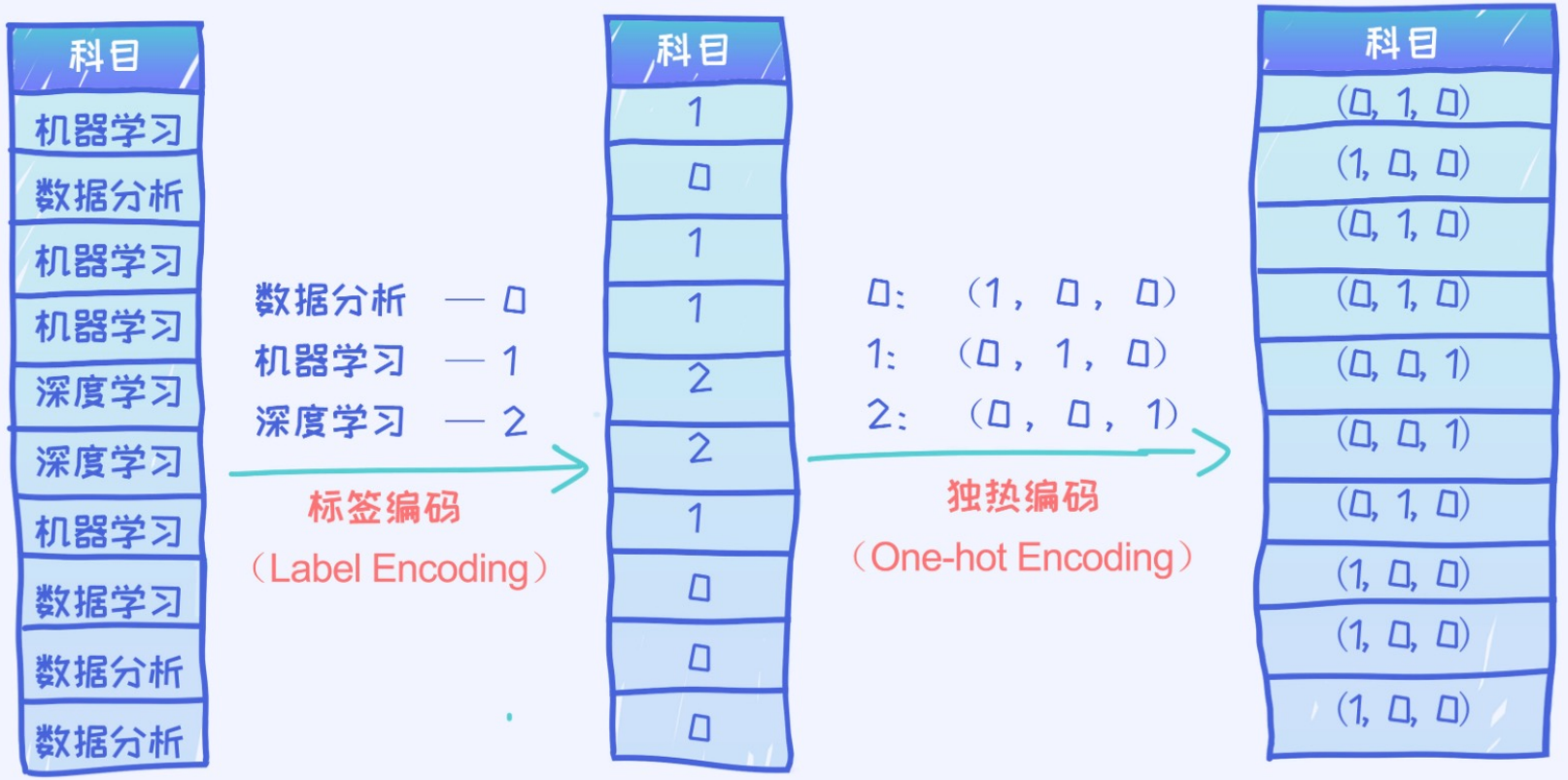
特征编码(feature encoding)

这三个特征都叫做类别(categorical)特征，因为这些特征值之间没有大小关系，而且只代表某一种类别。所以需要采用针对类别特征的编码技术。其中最常用的技术叫做独热编码(one-hot encoding)。

假如我们直接使用标签编码来代表这里每一个类别特征是有问题的。那具体存在什么问题呢？ 可以这么想，如果我们直接把类别特征看作是具体的数比如0，1，2... 那这时候，数与数之间是有大小关系的，比如2要大于1，1要大于0，而且这些大小相关的信息必然会用到模型当中。

对于深度学习，数据分析来说它们之间并不存在所谓的“大小”，可以理解为平行关系。所以对于这类特征来说，直接用0，1，2.. 的方式来表示是存在问题的，所以结论是不能这么做。

特征编码 - 独热编码（One-hot Encoding）



数值离散化

x 1 1 1 1 1 3 3 3 3 1 1 1 2 2 3 2 3 2 1 2 2 3

年龄特征集合 = [20, 21, 22, 21, 20, 29, 30, 29, 30, 21, 20, 19, 23, 24, 26, 24, 28, 23, 20, 23, 24, 30]

$x \leq 22 \Rightarrow x=1$

$22 < x \leq 24 \Rightarrow x=2$

$x > 24 \Rightarrow x=3$

连续性特征的离散化操作可以增加模型的非线性型，同时也可以有效地处理数据分布的不均匀的特点。

分类算法-KNN工作原理

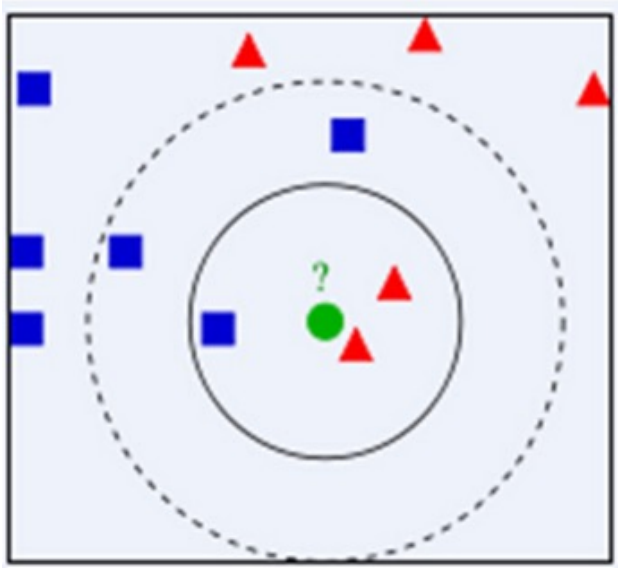
对于KNN，可以用两句话来做个总结。首先，它是最容易理解的机器学习算法，这也是为什么我们把它作为第一个算法来重点讲解。学习编程语言的第一步通常是学习编写"Hello World"，那对于机器学习来说，KNN就像是一个Hello World算法。

分类算法-KNN工作原理

存在一个样本数据集合，也称作为训练样本集，并且样本集中每个数据都存在标签，即我们知道样本集中每一个数据与所属分类的对应关系。输入没有标签的新数据后，将新的数据的每个特征与样本集中数据对应的特征进行比较，然后算法提取样本最相似数据(最近邻)的分类标签。一般来说，我们只选择样本数据集中前k个最相似的数据，这就是k-近邻算法中k的出处，通常k是不大于20的整数。最后，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。

KNN工作原理

下面通过一个简单的例子说明一下：如下图，绿色圆要被决定赋予哪个类，是红色三角形还是蓝色四方形？如果K=3，由于红色三角形所占比例为2/3，绿色圆将被赋予红色三角形那个类，如果K=5，由于蓝色四方形比例为3/5，因此绿色圆被赋予蓝色四方形类。

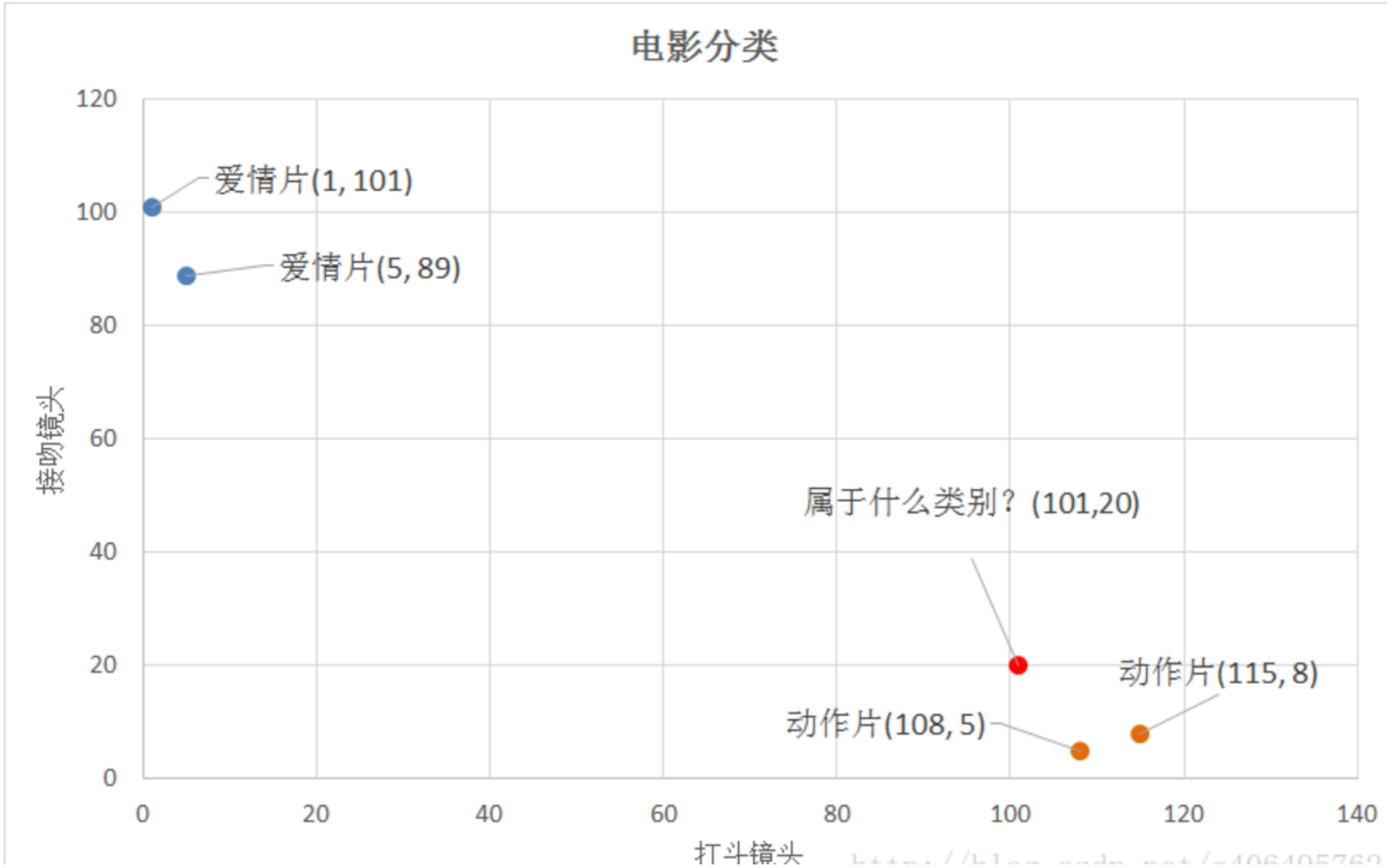


小案例

电影名称	打斗镜头	接吻镜头	电影类型
电影1	1	101	爱情片
电影2	5	89	爱情片
电影3	108	5	动作片
电影4	115	8	动作片

k-近邻算法也可以像我们人一样做到这一点，不同的地方在于，我们的经验更”牛逼”，而k-邻近算法是靠已有的数据。比如，你告诉我这个电影打斗镜头数为2，接吻镜头数为102，我的经验会告诉你这个是爱情片，k-近邻算法也会告诉你这个是爱情片。值得注意的是knn只是基于历史数据，不会出现其他可能性！

小案例-距离计算公式



$$|AB| = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

欧氏距离

Sklearn介绍

Scikit learn 也简称sklearn，是机器学习领域当中最知名的python模块之一。sklearn包含了很多机器学习的方式：
Classification 分类
Regression 回归
Clustering 非监督分类
Dimensionality reduction 数据降维
Model Selection 模型选择
Preprocessing 数据与处理
使用sklearn可以很方便地让我们实现一个机器学习算法。一个复杂度算法的实现，使用sklearn可能只需要调用几行API即可。所以学习sklearn，可以有效减少我们特定任务的实现周期。

Sklearn-knn介绍

我们使用sklearn.neighbors.KNeighborsClassifier就可以是实现上小结，我们实现的k-近邻算法。KNeighborsClassifier函数一共有8个参数

Methods

fit (X, y)	Fit the model using X as training data and y as target values
get_params ([deep])	Get parameters for this estimator.
kneighbors ([X, n_neighbors, return_distance])	Finds the K-neighbors of a point.
kneighbors_graph ([X, n_neighbors, mode])	Computes the (weighted) graph of k-Neighbors for points in X
predict (X)	Predict the class labels for the provided data
predict_proba (X)	Return probability estimates for the test data X.
score (X, y[, sample_weight])	Returns the mean accuracy on the given test data and labels.
set_params (**params)	Set the parameters of this estimator.

Sklearn-train-test

```
from sklearn import datasets
from sklearn.model_selection import train_test_split
import numpy as np

# 导入iris数据
iris = datasets.load_iris()
X = iris.data
y = iris.target
X_train, X_test, y_train, y_test = train_test_split(X, y,test_size=0.25,
random_state=2003)
```

这里的random_state就像随机生成器中的seed。通过不同的值采样的训练数据和测试数据是不一样的。设定一个固定的random_state有助于诊断程序本身，因为每次所期待的结果都会一样。

KNneighborsClassifier参数说明

- n_neighbors：默认为5，就是k-NN的k的值，选取最近的k个点。
- weights：默认是uniform，参数可以是uniform、distance，也可以是用户自己定义的函数。uniform是均等的权重，就说所有的邻近点的权重都是相等的。distance是不均等的权重，距离近的点比距离远的点的影响大。用户自定义的函数，接收距离的数组，返回一组维数相同的权重。

KNneighborsClassifier参数说明

- algorithm：快速k近邻搜索算法，默认参数为auto，可以理解为算法自己决定合适的搜索算法。除此之外，用户也可以自己指定搜索算法ball_tree、kd_tree、brute方法进行搜索，brute是蛮力搜索，也就是线性扫描，当训练集很大时，计算非常耗时。kd_tree，构造kd树存储数据以便对其进行快速检索的树形数据结构，kd树也就是数据结构中的二叉树。以中值切分构造的树，每个结点是一个超矩形，在维数小于20时效率高。ball tree是为了克服kd树高纬失效而发明的，其构造过程是以质心C和半径r分割样本空间，每个节点是一个超球体。
- leaf_size：默认是30，这个是构造的kd树和ball树的大小。这个值的设置会影响树构建的速度和搜索速度，同样也影响着存储树所需的内存大小。需要根据问题的性质选择最优的大小。

KNneighborsClassifier参数说明

- metric：用于距离度量，默认度量是minkowski，也就是p=2的欧氏距离(欧几里德度量)。
- p：距离度量公式。在上小结，我们使用欧氏距离公式进行距离度量。除此之外，还有其他的度量方法，例如曼哈顿距离。这个参数默认为2，也就是默认使用欧式距离公式进行距离度量。也可以设置为1，使用曼哈顿距离公式进行距离度量。
- metric_params：距离公式的其他关键参数，这个可以不管，使用默认的None即可。
- n_jobs：并行处理设置。默认为1，临近点搜索并行工作数。如果为-1，那么CPU的所有cores都用于并行工作。

距离度量metric	K近邻法和限定半径最近邻法类可以使用的距离度量较多，一般来说默认的欧式距离（即p=2的闵可夫斯基距离）就可以满足我们的需求。可以使用的距离度量参数有： a) 欧式距离 “euclidean”: $\sqrt{\sum_{i=1}^{n}(x_i-y_i)^2}$ b) 曼哈顿距离 “manhattan”: $\sum_{i=1}^{n}\|x_i-y_i\|$ c) 切比雪夫距离“chebyshev”: $max\|x_i-y_i\|(i=1,2,...n)$ d) 闵可夫斯基距离 “minkowski”(默认参数): $\sqrt[p]{\sum_{i=1}^{n}(\|x_i-y_i\|)^p}$ p=1为曼哈顿距离， p=2为欧式距离。 e) 带权重闵可夫斯基距离 “wminkowski”: $\sqrt[p]{\sum_{i=1}^{n}(w*\|x_i-y_i\|)^p}$ 其中w为特征权重 f) 标准化欧式距离 “seuclidean”: 即对于各特征维度做了归一化以后的欧式距离。此时各样本特征维度的均值为0，方差为1. g) 马氏距离“mahalanobis”: $\sqrt{(x-y)^T S^{-1}(x-y)}$其中，$S^{-1}$为样本协方差矩阵的逆矩阵。当样本分布独立时， S为单位矩阵，此时马氏距离等同于欧式距离 还有一些其他不是实数的距离度量，一般在KNN之类的算法用不上，这里也就不列了。
距离度量附属参数p	p是使用距离度量参数 metric 附属参数，只用于闵可夫斯基距离和带权重闵可夫斯基距离中p值的选择，p=1为曼哈顿距离， p=2为欧式距离。默认为2

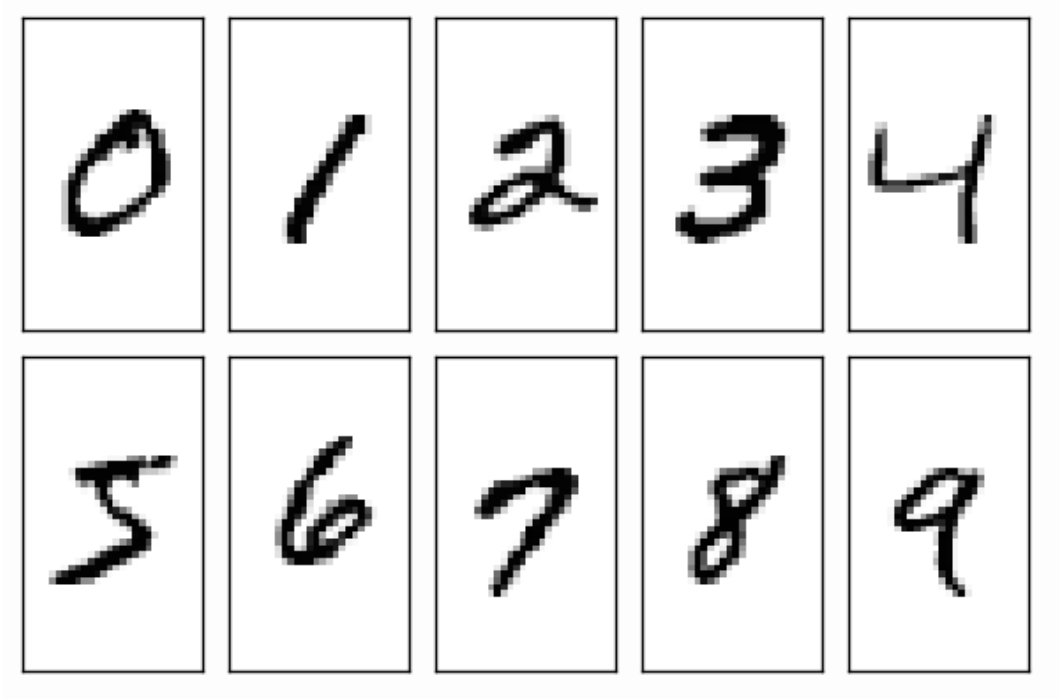
KNN工作原理

```
import numpy as np
from sklearn.neighbors import KNeighborsClassifier as kNN

def createDataSet():
    #四组二维特征
    group = np.array([[1,101],[5,89],[108,5],[115,8]])
    #思维特征的标签
    labels = [ '爱情片' , '爱情片' , '动作片' , '动作片' ]
    return group,labels

if __name__== '__main__' :
    group,labels = createDataSet()
    neigh = kNN(n_neighbors=3,algorithm= 'auto' )#从近到远的样本，选取前3个
    neigh.fit(group,labels) #拟合历史数据
    classifierResult = neigh.predict([[6,20]]) #预测，或者使用[6,20].reshape(1,-1)
    print(classifierResult)
```

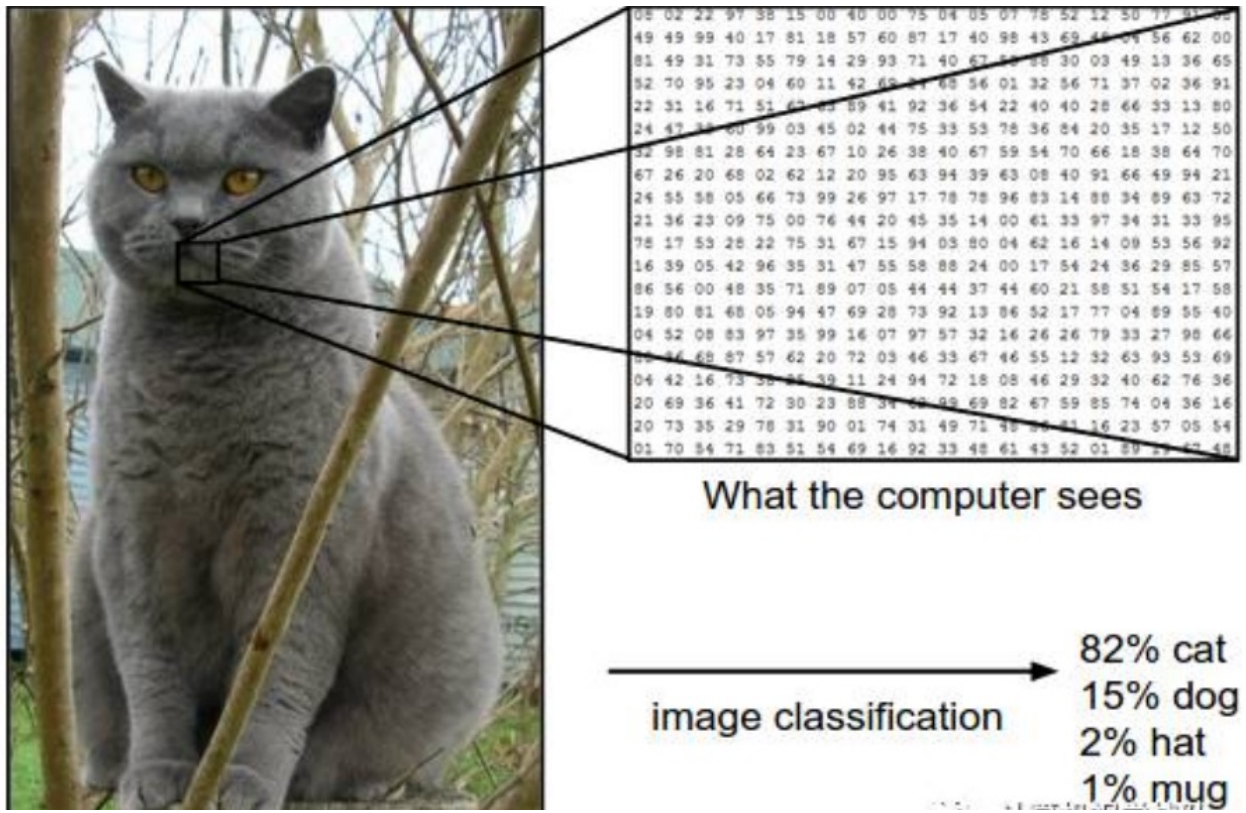
数据集

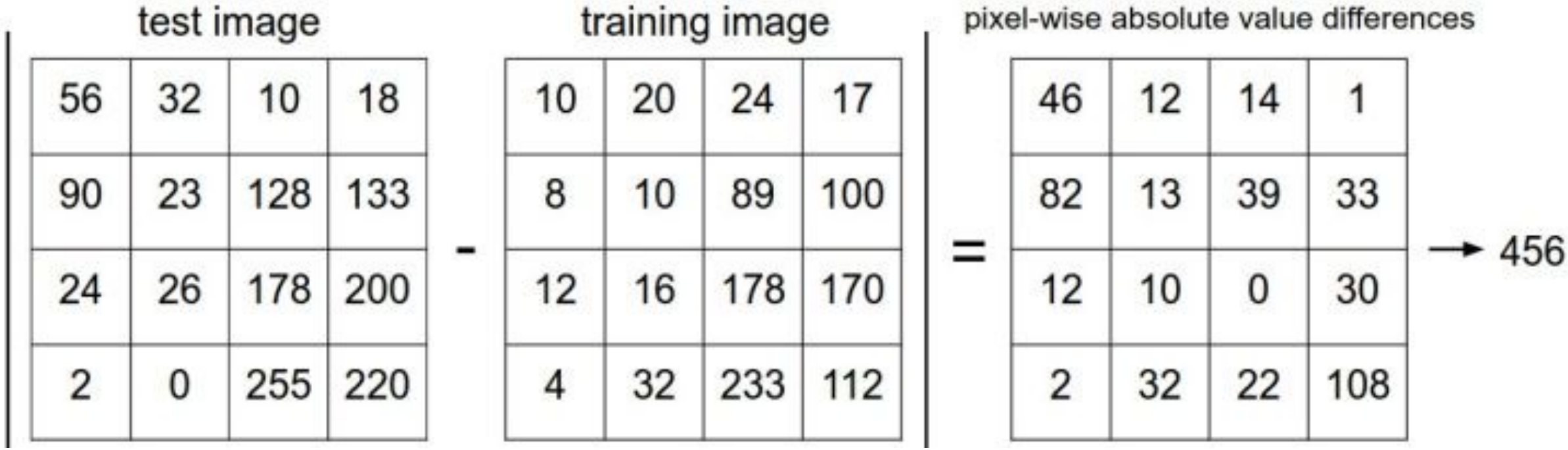


MNIST数据集可视化

MNIST数据集是一个比较经典的数据集。MNIST数据集来自美国国家标准与技术研究所（ National Institute of Standards and Technolo，NIST）。训练集由 250 个人手写的数字构成, 其中 50% 是高中学生, 50% 是人口普查的工作人员。 测试数据集也是同样比例的手写数字数据。

计算机眼中的图片


What the computer sees
image classification
82% cat
15% dog
2% hat
1% mug


test image
training image
pixel-wise absolute value differences
56 32 10 18
90 23 128 133
24 26 178 200
2 0 255 220
-
10 20 24 17
8 10 89 100
12 16 178 170
4 32 233 112
=
46 12 14 1
82 13 39 33
12 10 0 30
2 32 22 108
→ 456

K-Nearest Neighbors (KNN)

The most important hyperparameter for KNN is the number of neighbors (*n_neighbors*).

Test values between at least 1 and 21, perhaps just the odd numbers.

n_neighbors in [1 to 21]

It may also be interesting to test different distance metrics (*metric*) for choosing the composition of the neighborhood.

p in [1,2]

For a fuller list see:

sklearn.neighbors.DistanceMetric API

It may also be interesting to test the contribution of members of the neighborhood via different weightings (*weights*).

weights in ['uniform' , 'distance']

For the full list of hyperparameters, see:

sklearn.neighbors.KNeighborsClassifier API.

```
param_grid = [
    {
        'weights':['uniform'],
        'n_neighbors':[i for i in range(1,11)]
    },
    {
        'weights':['distance'],
        'n_neighbors':[i for i in range(1,11)],
        'p':[i for i in range(1,6)]
    }
]
grid_search = GridSearchCV(knn_clf,param_grid,n_jobs=-1,verbose=2)
```

接下来就用grid_search对象训练数据

```
%%time
grid_search.fit(X_train,y_train)
```

训练结果如下

```
Fitting 3 folds for each of 60 candidates, totalling 180 fits
Wall time: 1.99 s
[Parallel(n_jobs=-1)]: Done 180 out of 180 | elapsed:    1.5s finished
```

Out[28]:
```
GridSearchCV(cv=None, error_score='raise',
       estimator=KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=1, n_neighbors=6, p=2,
           weights='uniform'),
       fit_params=None, iid=True, n_jobs=-1,
       param_grid=[{'weights': ['uniform'], 'n_neighbors': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]}, {'weights': ['dista
nce'], 'n_neighbors': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10], 'p': [1, 2, 3, 4, 5]}],
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring=None, verbose=2)
```

```
grid_search.best_estimator_
```

这是取出参数最好的一组对应的分类器

Out[15]:
```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=1, n_neighbors=6, p=2,
           weights='uniform')
```

```
model = KNeighborsClassifier()
n_neighbors = range(1, 21, 2)
weights = ['uniform', 'distance']
p = [1, 2]
# define grid search
grid = dict(n_neighbors=n_neighbors,weights=weights)
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
grid_search = GridSearchCV(estimator=model, param_grid=grid,
 n_jobs=-1, cv=cv, scoring='accuracy',error_score=0)
grid_result = grid_search.fit(X, y)
# summarize results
print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
params = grid_result.cv_results_['params’ ]

#p：p=1表示曼哈顿距离，p=2表示欧式距离

https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.
```

总结

kNN的总结

1. kNN是一个极其简单的算法

2. 算法比较适合应用在低维空间

3. kNN在训练过程中实质上不需要做任何事情，
   所以训练本身不产生任何时间上的消耗

4. 然而，kNN在预测过程中需要循环所有
   的样本数据，复杂度线性依赖于样本个数，
   这成为kNN应用在大数据上时的瓶颈