总论

2022年10月10日　19:47

机器学习是人工智能的一个实现方式、深度学习是机器学习一个方法
人工智能>机器学习>深度学习
1956年：人工智能元年

机器学习是从数据中自动分析获得模型，并利用模型对未知数据进行预测
三大特点：数据、模型、预测
机器学习和人类似，都是从历史数据中得到解决目前问题的方法

数据集构成：矩阵类型

特征值+目标值，例如：

	房子面积	房子位置	房子楼层	房子朝向	目标值
数据1	80	9	3	0	80
数据2	100	9	5	1	120
数据3	80	10	3	0	100

每一行数据可称之为一个样本
有些数据集可以没有目标值，需要靠机器学习聚类方法将其分类
注意：目标值是我们要估计的对象，但不是所有目标值都是估计的，肯定需要先传入一些特征值跟它们对应的目标值，才能建立模型，随后传入新的特征值，这样才能估计新的目标值
需要注意的是，估计目标值就是一种参数估计，即未知量可视作一个参数，所有已知的数据的值都是特征值。

机器学习模式：

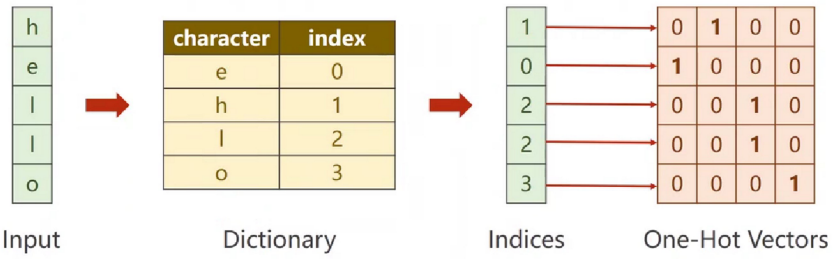
对抗学习：找对手互相对抗，共同进步学习
无监督学习：自己找标签学习
弱监督学习：学校拧螺丝，工作造火箭，即实际应用高一个层次
半监督学习：标签不够多、不够准
多模态学习：像人一样调用各种“感官”同时进行学习
强化学习
迁移学习：举一反三

机器学习算法：

监督学习(supervised learning)：
定义：输入数据是由输入特征值和目标值所组成。函数的输出可以是一个连续的值（称为回归），或输出是有限个离散值（称作分类）
分类算法：k-近邻算法、贝叶斯分类、决策树与随机森林、逻辑（逻辑斯蒂）回归
回归算法：线性回归、岭回归
无监督学习(unsupervised learning)：
定义：输入数据是由输入特征值所组成
聚类算法：K-means

遗传算法：模拟大自然的物竞天择适者生存，并将优良基因遗传下去
遗传算法的本质是试错，相当于在所有可能的情况中随机抽取几个，稳定性太差
基本概念：
染色体：某个问题的可行解，即只要该模型输出了这个可行解，就说明它的遗传物质适应了环境
基因：可行解的每一个元素都是一个基因，只有所有基因都符合环境，才是一个可行解，才能生存下去
适应度函数：每次迭代都会衡量染色体的优劣，根据打分淘汰不适应环境的个体
交叉：子代的染色体，一部分来自父亲一部分来自母亲。即一部分基因来自父一部分来自母
变异：交叉只能保留优良基因，但不能产生新基因，由此引入变异。一般是随机选择并修改基因的值
复制：每次进化（迭代）中，为保留优良染色体，需要将上一代适应度最高的几条染色体原封不动复制给下一代
注意：遗传算法并没有真正的”进化“，它只能输出特定的结果，不能增加模型的复杂度，进而不能拟合更多的情况。如果在迭代的过程中，想办法增加模型权重的数量，使其变复杂，或许是一个突破口
另外遗传算法稳定性非常差，随机性过高，同一个程序，最快和最慢次数竟然能差距30倍以上。

独热编码（one-hot）：对数据进行特征处理，将其转换为稀疏矩阵。
特点：用0和1表示，而不用0、1、2...表示，虽然增大了矩阵，但数之间没有大小关系
矩阵的横向长度表示特征值个数，纵向长度表示总的字符个数
维度太高、硬编码（即编码字符的时候不是学习到的而是人为指定的）



机器学习开发流程：

1、获取数据

2、数据预处理：数据清洗等，如一些垃圾数据需要丢弃（例如乱填的问卷）

3、特征工程：

特征工程（Feature Engineering）特征工程是将原始数据转化成更好的表达问题本质的特征的过程，使得将这些特征运用到预测模型中能提高对不可见数据的模型预测精度

数据处理和特征工程之间并没有明显的界限，可以视作一个是初级的，一个是高级的，前者主要负责数据清洗，后者主要负责数据转换成更好表达问题的数据，如主成分分析

4、机器学习算法训练-模型

5、模型评估

6、应用

算法是核心、数据与计算是基础

找准定位

可用数据集（现成的数据集，能用于模型训练，不用自己收集数据）：

Scikit-learn：数据量小、方便学习

Python语言的机器学习工具，包括许多知名的机器学习算法的实现；文档完善，API众多。需要numpy等库的支持。

导入：import sklearn

UCI（加州大学欧文分校）：收集了500多个数据集，数据量几十万，覆盖科学、生活、经济等领域

Kaggle：大数据竞赛平台，真实数据，数据量巨大

sklearn.datasets加载流行数据集

dataset.load_*()：获取小规模数据集，数据包含在datasets中

datasets.fetch_*(data_home = None，subset='all')：获取大规模数据集，需要从网上下载，函数第一个参数是data_home，表示数据下载的目录，默认是~/scikit_learn_data，第二个参数填'all'表示测试集和训练集都要，'train'是只要训练集，'test'是只要测试集

注：*代表具体数据集的名字

load和fetch返回值都是datasets.base.Bunch（继承自字典的类型）

字典的key：data（特征数据数组）、target（目标值数组）、DESCR（数据描述，比如描述特征具体是啥）、feature_named（特征名）、target_names（标签名）

```
from sklearn.datasets import load_iris
iris = load_iris()
# load_iris()方法返回iris的有关数据，包含data、target、frame等
a = iris.data   # 只需要data的数据
print(a)
```

因为返回数据集是继承字典的，所以可调用字典的所有函数，并且可以直接.key的名字直接获取数据（一般字典不行），如iris.data

字典可以直接通过dict['key']来获取key的值

机器学习一般的数据集会划分成两个部分：

训练数据：用于训练，构建模型

测试数据：在检验模型的时候使用，用于评估模型

一般用7-3或8-2的原则划分

标签值（目标值）：需要估计的参数的值。训练后模型对输入的数据，进行该值的预测

特征值：完全不需要估计的参数的值

此外还有标签值或特征值的名字，比较容易混淆

下面是将sklearn的数据集进行划分，主要使用train_test_split函数

```
from sklearn.datasets import load_iris
# 导入将数据集划分成训练数据和测试数据的模块
from sklearn.model_selection import train_test_split
def datasets_demo():
    iris = load_iris()
    #print(iris.target)
```

```
    # 分割训练集，前两个参数为特征值和目标值，test_size表示从中抽取多少作为测试数据，剩下的都用作训练，最后一个是随机数种子
    x_train, x_test, y_train, y_test = train_test_split(iris.data,iris.target,test_size=0.2,random_state=22)
    # 打印测试集和测试集的形状（多少个，和多少列）
    print('训练集的特征值：',x_test,x_test.shape)
```

sklearn模型的保存和加载API：from sklearn.externals import joblib

保存：joblib.dump(rf,'test.pkl')

rf：预估器

test.pkl：模型路径

加载：estimator = joblib.load(test.pkl)

特征工程

2022年11月3日 10:46

特征工程（Feature Engineering）：处理数据

数据和特征决定了机器学习的上限，而模型和算法只是在逼近这个上限而已。人们使用的工具就那么几个，深度学习框架TF、Pytorch，算法也是常用的算法，**真正拉开差距的，一个是数据的质量，另一个就是特征工程**

特征工程是使用专业背景知识和技巧处理数据，使得特征能在机器学习算法上发挥更好的作用，它是非常重要的，会直接影响机器学习的效果

特征工程包含：特征抽取、特征预处理、特征降维

特征抽取（特征提取）

很多数据，我们是不能直接进行处理的，比如文本类，图像类等，都不是直接的数据，所以我们要对其进行转换，转换成特征数据，以便处理，常用的方法：定量化、哑变量等。特征提取API（sklearn）：sklearn.feature_extraction

哑变量（Dummy Variable）：也叫虚拟变量，其存在的目的是将不能够定量处理的变量（即定性变量）量化，比如（血型、评级：好、良好、一般等）。在线性回归引入哑变量的目的是考察定性变量对哑变量的影响。哑变量取值一般只有0和1。

哑变量的确定：对于n个定性变量，如好、良好……需要引入n-1个哑变量x1，x2，x3……，x1==1表示取到“好”，0表示不取到“好”，x2==1表示取到“良好”……最后一个变量，是当前面所有变量都取0时得到，所以只需要n-1个哑变量。当然有的算法还是用了n个哑变量，具体分析吧。

特点：哑变量方法只在离散型变量水平较小时使用，一般在3个以内；哑变量的取值只有0和1，就像一个开关，可以屏蔽掉某些情况，使之不进入分析

one-hot编码（独热编码）：跟哑变量基本相同，使用n个变量将非数值类的变量转换成数值变量的一种方法，使用1表示有这个特征，0表示没有这个特征

稀疏矩阵（sparse matrix）：矩阵中非零数，远小于矩阵中零的数目。稀疏矩阵将数值按位置表示出来，可以节省存储（矩阵的压缩存储）。使用哑变量或one-hot编码方法，都会产生稀疏矩阵

入门阶段，我们主要对三类数据：字典、文本、图像进行特征提取

步骤：

1、导入sklearn的特征提取模块

2、实例化一个转换器（将原数据如文本等转化成矩阵等数据）

3、调用转换器的fit_transfer方法，转换文件

字典：有很多类别特征或本身数据就是字典则可使用字典方法（原数据不是必须为字典数据类型）

```
from sklearn.feature_extraction import DictVectorizer
data = [{'city':'北京','temperature':20},{'city':'上海','temperature':25},{'city':'广州','temperature':30}]
# 实例化一个转换器对象，sparse=False表示不转换成稀疏矩阵（而是用二维数组表示）
transfer = DictVectorizer(sparse=False)
# 调用转换器的fit_transfer方法，发生转换
data_new = transfer.fit_transform(data)
print(data_new)
```

结果为二维数组：

```
[[ 0.  1.  0. 20.]
 [ 1.  0.  0. 25.]
 [ 0.  0.  1. 30.]]
```

如果将sparse改成True，则结果为稀疏矩阵：

```
(0, 1)    1.0
(0, 3)    20.0
(1, 0)    1.0
(1, 3)    25.0
(2, 2)    1.0
(2, 3)    30.0
```

这确实是一个稀疏矩阵，前面为矩阵的格点坐标（行列均从0开始），后面是这个元素的数值，其他位置元素均为0

这个转换器除了上面这种方法除此之外，还有方法：

DictVectorizer.inverse_transform(X)，X为数组或稀疏矩阵，返回转换前的数据格式

DictVectorizer.get_feature_names_out(X)返回类别名称，在这个例子中为：

['city=上海', 'city=北京', 'city=广州', 'temperature']

文本：所有文本类型，均可使用文本方法进行分词、提取特征

特征选取问题：到底哪个作为特征比较合适：单词、字母、短语、句子。一般用单词作为特征，文本的每一个句子作为一个样本。

```
from sklearn.feature_extraction import text
# 标点符号、字母均不统计
data = ['life is too too short, i like python.','life is too long, i dislike python.']
transfer = text.CountVectorizer(stop_word=[])  #参数为停用词，即不计入统计的词，默认是空列表
# 返回词频矩阵，data不能为字符串；送代器里有多少个字符串，返回的矩阵就有多少行
data_new = transfer.fit_transform(data)
# 转换器不能把稀疏矩阵转成数组。但转换之后的数据类型有toarray方法，字典转换器的sparse参数就是调用了这个方法
print(data_new.toarray())
# 文章中的单词将作为特征词，一般会和词频矩阵一起输出
a = transfer.get_feature_names_out(data)
print(a)
```

如果是中文，则词与词之间若没有空格进行分割，会把整个句子都看成一个单词（英文本身就有空格分割），就算有分割，单个字同样会被排除掉不统计。可以使用jieba库或其他库+join方法来进行中文分词

```
import jieba
from sklearn.feature_extraction.text import CountVectorizer
data_origin = '一段文本太长，不粘贴'
sentance_list = data_origin.split('。')
data = []
for sentance in sentance_list:
    # 先将句子转换成分词列表，再用空格拼成一个句子
    a = jieba.lcut(sentance)
    b = ' '.join(a)
    data.append(b)
transfer = CountVectorizer(stop_words=['一种','所以'])
data_new = transfer.fit_transform(data)
features_names = transfer.get_feature_names_out(data)
# 将稀疏矩阵转化成二维数组
print(data_new.toarray())
print(features_names)
```

二维数组其实就是一个矩阵，而稀疏矩阵不直观。特征提取得到的是一个矩阵，矩阵行个数就是迭代器中成员个数（文本则是每一个句子），列个数则是总的特征值的个数。矩阵的格点的值，表示这一行（这个成员，在文本中是一个句子）的这个特征值（文本中是词组）出现的次数

但是，上面的这种方式，只能得出一个词频统计，如果想要具体分析这段文本，需要一些关键词：在某一个类别的文章中，出现的次数很多，但是在其他类别的文章中出现很少。这样的词更能帮助我们得到结果，这就是我们下面的方法二：TfidVectorizer

TF-IDF（term frequency-inverse document frequency词频-逆向文档频率）的主要思想：如果某个词在一篇文章中出现概率很高，并且在其他文章中很少出现，则认为这个词或者短语具有很好的类别区分能力，是我们需要的关键词

词频为某文件中该词的出现频率（不是频数），idf表示词语的重要性度量，某个词的idf，可通过总文件数目，除以包含这个词的文件数目，将商取10的对数得到。tf-idf = tf*idf，tf*idf可理解为重要程度

代码实现：TfildVectorizer方法，且使用方法和CountVectorizer完全相同。但输出的矩阵的值就不是词频了，而是tf-idf值，最大则说明越有统计意义

Tfidf的重要性：分类机器学习算法进行文章分类中前期数据处理方式

图像：后期专门强调

特征预处理

定义：通过一些转换函数将特征数据转换成更加适合算法模型的特征数据的过程，主要是将特征抽取得到的数值型数据进行无量纲化

注意：训练集做了什么变换，则测试集也需要进行变换

无量纲化：包括归一化和标准化

所有需要计算值的，无论是分类的计算距离，还是回归分析，都需要进行无量纲化（单变量不需要）

归一化：通过对原始数据进行变换，把数据映射到某区间（默认0-1）之间

归一化（转化成0-1区间）中，某特征最大的值将转化成1，最小值将转化成0.min、max都是一列（某特征）中的极值

$$X' = \frac{x - min}{max - min}$$

如果转化到其他区间，则还需要进行第二步（区间伸缩和移动）：

$$X'' = X' * (mx - mi) + mi$$

代码实现：sklearn.preprocessing.MinMaxScaler(feature_range=(0,1)...) 也是一个转换器，参数表示放缩范围，默认0-1

MinMaxScaler.fit_transform(X) X表示numpy array格式的数据（矩阵），返回转换后形状相同的array

标准化：归一化容易受到异常值影响（因为要根据极值归一化），即鲁棒性较差，只适用于传统精确小数据场景

标准化则通过变换到均值为0、标准差为1的范围内，就算出现少量异常点也不影响总体均值和std，要是有大量异常点，还是remake吧

$$X' = \frac{x - \text{mean}}{\sigma}$$

代码实现：sklearn.preprocessing.StandardScaler()和StandardScaler.fit_transform(X)

标准化适合大数据的场景，用的比较多，优先考虑标准化

归一化/标准化的原因：特征的单位相差较大，或某特征的方差比其他特征要大出几个数量级，造成权重不平衡，容易影响目标结果，使得一些算法无法学习到其他特征。比如，算样本的欧氏距离，某特征的值都是几万，而某个特征的值只有个位数，则距离几乎只受前一个特征影响，但两特征本应该相同

特征降维

降维：在某些限定条件下，降低随机变量（特征）的个数。得到一组不相关的主要变量的过程，常见方法：主成分分析、因子分析、特征选择

0维：标量，1维：向量，2维：矩阵，3维及以上：矩阵或嵌套矩阵（矩阵分块法表明这两其实是一样的）

因为在进行训练时，我们都是用特征进行学习，如果特征本身相关或存在问题，则对算法学习预测会影响较大

两种方法：

特征选择

代码实现：sklearn.feature_selection

Filter过滤式

方差选择法：低方差特征过滤：方差过小，变化不明显，包含的信息少，没有研究的意义。

比如要分类某个种类的鸟，对于“是否有爪子”这个特征就排掉，因为所有鸟都有爪子，方差很小，所以不需要研究

代码实现：sklearn.feature_selection.VarianceThreshold(threshold = 0) 参数为方差临界值，默认为0，即默认方差=0的指标才删掉

Variance.fit_transform(X) X为array格式的数据，返回值：训练集差异低于threshold的特征将被删除，默认保留所有非0方差

相关系数法：皮尔逊相关系数衡量两个特征之间相关性，如果相关系数很大（绝对值几乎=1），则考虑排掉其中一个或加权求和。另外还可以主成分分析，自动将一些相关性强的特征排掉

当r为正，则表示两变量正相关，r为负则是负相关，为0表示无相关关系（但可能有其他关系如二次关系）

代码实现：不在sklearn里，而是在scipy里：from scipy.stats import pearsonr， pearsonr(x,y)

```
from scipy.stats import pearsonr
r = pearsonr([3,4,5,6],[2,1,5,6])
print(r)
```

x、y为两个变量，可以传入列表、元组等，不能是集合

Embeded嵌入式

决策树

正则化

深度学习

主成分分析

将高维数据转化为低维数据，损失少量信息，排掉相关性强的特征变量，创造主成分特征变量

代码实现：sklearn.decomposition.PCA(n_compositions = None) 参数若是小数，则表示保留多少的信息(1为全保存)，整数表示减少到几个特征

transfer = PCA()， transfer.fit_transform(data)

分类

2022年11月3日 14:37

进行完特征工程后，则可将数据集用来训练模型

二分类问题，输出一个概率值p就行，第二个类的概率就是1-p

多分类问题，我们则需要输出一个分布，这个样本属于每个类的概率满足：和等于1；每个都大于等于0

sklearn转换器和估计器

转换器就是特征提取的哪个transfer，使用方法就是先导入该转换器，然后实例化，最后传入数据

fit_transform由fit和transform两个方法进一步封装而成

fit：计算参数，比如要原数据标准化，还需要计算每一个特征的平均值、标准差

transform：将原数据转换

估计器：sklearn机器学习算法的实现

在sklearn中，估计器（estimator）是一个重要角色，是一类实现了各种机器学习算法的API

用于分类的估计器：

sklearn.neighbors：kNN算法（k-近邻算法）

sklearn.naive_bayes：朴素贝叶斯算法

sklearn.linear_model.LogisticRegression：逻辑（逻辑斯蒂）回归

sklearn.tree：决策树与随机森林

用于回归的估计器：

sklearn.linear_model.LinearRegression：线性回归

sklearn.linear_model.Ridge：岭回归

用于无监督学习的估计器：

sklearn.cluster.KMeans：k-means聚类

使用方法：

1、导入并实例化一个estimator

2、estimator.fit(x_train, y_train)传入训练集的特征值和目标值，进行训练，调用完成后生成模型

3、模型评估：

1、直接把测试集特征值传入模型，得到预测的目标值，拿来和测试集的目标值对比

y_predict = estimator.predict(x_test)

y_test == y_predict

该方法返回一个布尔值矩阵，如果预测正确就是True

2、传入测试集，计算得分（准确率）

accuracy = estimator.score(x_test, y_test)

该方法返回一个0-1的小数

分类算法：目标值为类别

KNN算法（K近邻算法）

k的含义是取前k个和你最近的样本点

根据你的“邻居”来定位你的类别，具体定义见多元统计分析--判别分析

训练目标：得到“邻居”的位置，以便于进行距离计算并划分

k值取得过大，可能会出错，即容易受到样本不均匀的影响，而k过小容易受到异常值影响

sklearn.neighbors.KNeighborsClassifier(n_neighbors=5, algorithm='auto')

参数1即代表k的值，即选取的邻居个数；第二个参数表示算法，可选：auto、ball_tree、kd_tree、brute

```
# 导入：数据集、划分数据集工具、标准化、KNN算法
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
# 1、获取数据
iris = load_iris()
# 2、划分数据集，注意返回的4个值按顺序为：训练集特征、测试集特征、训练集目标、训练集目标
# 即训练集在前，特征值在前
x_train,x_test,y_train,y_test = train_test_split(iris.data,iris.target,random_state=6)
# 3、特征工程：标准化，无论是训练集还是测试集的都需要标准化
transfer = StandardScaler()
x_train = transfer.fit_transform(x_train)
x_test = transfer.transform(x_test)
```

```
# 4、KNN算法预估器，它使用的距离为明可夫斯基距离，该API有参数p，其值1为曼哈顿距离，2为欧氏距离（默认），两者都是明式距离的推广
estimator = KNeighborsClassifier(n_neighbors=3)
estimator.fit(x_train,y_train)
# 5、模型评估：两种方法
y_predict = estimator.predict(x_test)
print(y_predict)
print('直接对比真实值和预测值：',y_test == y_predict)
score = estimator.score(x_test,y_test)
print('准确率为：',score)
```

注意：不能在划分数据集之前统一进行标准化，这样的标准化会使划分后的训练集用到了有关测试集的信息

另外，在标准化的时候，测试集特征值需要transform而不用fit，即需要沿用上面的fit计算得到的均值和标准差，因为对数据而言，必须进行完全相同的数值运算，否则结果会出问题

另外，对于目标值，无论是训练集还是测试集，都是不需要标准化的

优缺点：

简单，易于理解，易于实现，无需训练

懒惰算法，对测试样本分类时计算量大，内存开销大；必须指定K值，K值选择不当则精度可能下降

使用场景：小数据场景，如几千--几万

朴素贝叶斯算法(naive bayes)

贝叶斯算法是一系列以贝叶斯公式为核心的算法，这里所说的特指最大后验概率（最小错误率）贝叶斯

基础知识知识：概率、联合概率、条件概率、相互独立、贝叶斯公式等

朴素：假定特征与特征之间是相互独立的，即加上条件独立假设的贝叶斯判别为朴素贝叶斯判别

朴素贝叶斯算法：朴素+贝叶斯公式

原理：计算已经出现了一个样本，将其分配给i类的概率（该式子可由贝叶斯公式进行分解），将样本分配到概率最大的类

损失函数：将本为a类的样本错误分到b类的损失，一般表示为c(b|a)，在最小风险化贝叶斯中用到

$$P(\omega_i|x) = \frac{P(x|\omega_i)P(\omega_i)}{P(x)} = \frac{P(x|\omega_i)P(\omega_i)}{\sum_i^2 P(x|\omega_i)P(\omega_i)}$$

训练目标：很明显样本越多下面的各个概率估计得越准

p(x)：总的出现特征值为x的概率，因为是朴素的，因此多个特征值则是联合概率（单个特征求概率乘积）

p(wi)：等于每个类的面积/总面积

p(x|wi)：在i类中发生x的概率，即根据i类的样本求分布函数，计算取值为x的概率。它和p(x)的区别，就是前者使用总体的分布求x的概率，而它使用该样本的分布求x概率，求法都是一样的

拉普拉斯平滑系数：

一个问题：当样本量很小的时候，可能出现某个概率为0的情况，导致估计不准

目的：防止计算出的分类概率=0

$P(F1|C) = \frac{Ni+\alpha}{N+\alpha m}$

其中α为指定的系数，一般为1，m为训练集中统计特征词的个数，即一共有多少个特征

该公式计算的就是p(x|wi)，不过是只有一个特征（一维）的时候，含义为在样本C中出现某个特征值的概率。如果是多维的情况则是连乘。不要纠结这个公式的字母含义，理解意思就行

注意：只在小样本情况下使用该系数，且使用后，非a和a的概率和可能不等于1

sklearn的API：

sklearn.naive_bayes.MultinomialNB(alpha=1)

参数为拉普拉斯平滑的那个α，若=0则表示不平滑

使用方法和KNN几乎一致，把estimator换一下就行

优缺点：

朴素贝叶斯算法发源于古典数学理论，有稳定的分类效率

对缺失数据不敏感，算法比较简单

分类准确度较高，速度快

由于使用了样本属性独立的假设，如果特征之间存在关联则用不了

朴素贝叶斯有一个词袋模型，即将一个句子的所有词放入一个袋子里搅合，再以随机的顺序取出来，朴素贝叶斯方法认为它们都是完全一致的，即不考虑词的顺序关系。这也是它的弊端之一

决策树

决策树就是利用分支结构来对数据进行划分

定理：任何n叉树、森林，都可以通过一定步骤化为二叉树

因此若有多分支问题，只需要建立二叉树模型

训练目标：

找到每个特征的重要程度，越重要的放的越接近根；以及每个特征的判定标准

找到这个重要程度后：

如果是二分类的情况：按从大到小的顺序，给一颗n层完全二叉树（n为特征个数）的每一层依次填入该特征的判定标准。这样可以最精确拟合训练集，但可能导致过拟合，且有些节点后已经分类完成不需要继续判定下面的节点了，继续判定不仅浪费资源，也没得到效果，这个时候就需要进行树的“剪枝”处理，进行机器学习有关算法的包都会自动进行剪枝

如果是多分类，或是多个值（或区间）进行划分，导致这个节点的度高于2，即这是一颗n叉树，原理和二分类的一致，但是要注意，这颗n叉树转换成二叉树时，其层数会增加，原本重要程度高的特征仍在上面，比如一个在第二层的节点有3个分支，则转化成二叉树后，这个第二层的节点，将拆分成二叉树的第二和第三层，原本第三层的节点往下顺延

判定特征重要程度的相关概念：

信息：消除随机不定性的东西（香农定义）

不能消除不确定的东西，比如废话，不能算作信息

信息量：信息的大小，消除不确定性的大小

信息熵：用H表示，单位比特bit，用于表示信息量的大小。

下面为计算某特征的信息熵公式，X表示随机变量，xi表示X每个可能的取值，注意公式的负号，因为P小于1，log出来的是负数

$$H(X) = -\sum_{i=1}^{n} P(x_i) log_b P(x_i))$$

b为自定的一个量，可以取任意正数但对每个变量的计算必须都是一致的，一般b=2

若求总体的信息熵，则X用它的目标值（比如类别）

信息增益：某个特征对信息熵的贡献，决策树划分依据之一

特征A对训练数据集D的信息增益g(D,A)，定义为集合D的信息熵H(D)与特征A给定条件（知道了特征A）下D的信息条件熵H(D|A)之差，公式为：

$$g(D,A) = H(D) - H(D|A)$$

显然，如果知道了特征A，则D里包含的信息就减少了，这个减少的部分，就是A对D的信息增益

决策树划分依据：

ID3：信息增益越大则越放在上层

C4.5：信息增益比越大则越放在上层

CART（分类回归树）：

包含了分类树和回归树两种决策树，分类树是目标变量（隐变量）是类别标签（即算法输出类别），而回归树

基尼系数最小的准则（基尼系数是熵以外另一种指标）

优势：划分更加细致

sklearn的决策树API：sklearn.tree.DecisionTreeClassifier(criterion='gini', max_depth=None, random_state=None)

criterion：默认为基尼系数，可选信息增益的熵'entropy'

max_depth：树的深度，即有几层。树不一定越大越好，越大可能出现过拟合的情况

ranodm_state：随机数种子

sklearn决策树的可视化：sklearn.tree.export_graphviz()：该函数可以导出DOT格式，重要的3个参数为：

estimator：就是估计器

out_file：导出的文件路径，包括文件名字，注意后缀是.dot的文件

feature_names=[]：特征名，不输入也可以

Pycharm有查看.dot文件的插件，若要看到树，需要去webgraphviz这个网站（需要科学上网）

优缺点：

简单的理解和解释，树木能可视化

初学者非常容易出现过拟合的现象

改进：1、剪枝（使用CART算法），2、使用随机森林算法

随机森林算法

集成学习算法：集成学习通过建立几个模型组合来解决单一预测问题，它的工作原理是生成多个分类器/模型，各自独立地学习和做出预测，这些预测最后结合成组合预测，因此优于任何一个单分类做出的预测

随机森林是决策树的一种改进，它是一个包含多个决策树的分类器，并且其输出的类别是由个别树输出的类别的众数决定的

判断正确的树都是相同的，错的树各有各的错法，会相互抵消

树不是越多越好

原理：N表示训练集样本个数，M表示每个样本的特征的数目

两个随机，为了保证每棵树的独立性。这种情况下，单棵树已经不能独立完成一个分类过程，必须整个森林一起进行预测

1、训练集随机：

一次随机选出一个样本，重复N次（可能出现重复的样本），即每棵树训练样本数目都是一样的，但是组成不一样，因为有重复的样本

2、特征随机：

在每棵树专有的训练集随机选出m个特征，m<<M，建立决策树

采用BootStrap抽样（随机有放回抽样）

sklearn中的API：sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None, bootstrap=True, random_state=None, min_samples_split=2)

有关参数为：

n_estimators：森林中树木的数量

criterion：分割特征的测量方法，即判定方法

max_depth：树的最大深度

max_features='auto'：每个决策树的最大特征数量：

- If "auto", then `max_features=sqrt(n_features)`.
- If "sqrt", then `max_features=sqrt(n_features)` (same as "auto").
- If "log2", then `max_features=log2(n_features)`.
- If None, then `max_features=n_features`.

bootstrap：是否使用放回抽样，默认为True

min_samples_split：节点划分最少样本数

min_samples_leaf：叶子节点的最小样本数

优缺点：

在当前算法中具有非常好的准确性

能够有效运行在大数据集上，具有处理高维特征的输入样本，而且不需要降维（特征随机已经降维了）

能够评估各个特征在分类问题上的重要性

对性能要求较高

模型选择和参数调优

交叉验证（cross validation）：为了让被评估的模型更加准确可信

将数据划分成训练集和测试集后，又将训练集继续划分成多份，其中一份为验证集。比如将训练集划分为3份训练集和1份验证集，然后经过四次测试，每次都更换不同的验证集，即得到4组模型的结果，取平均值为最终结果，又称4折交叉验证

这里的验证集可以理解为一个小的测试集，用来评估这一小组的准确率

超参数搜索-网格搜索（Grid Search）：

超参数：模型中需要手动指定的参数（可能不止一个），如KNN里的k值

手动设置超参数过程繁杂，所以需要对模型预设几个超参数组合，每组超参数都采用交叉验证来进行评估，最终选出最优参数（准确率最高）组合建立模型

这种方法非常暴力，但是一旦搜索到最合适的参数，则模型能建得非常好

sklearn中模型调优的API：sklearn.model_selection.GridSearchCV(estimator, param_grid=None, cv=None)

对估计器的指定参数值进行详细搜索，这个时候实例化估计器时就不要指定参数了

实例化一个估计器后，将其传入到这个API里面，后面的工作都由这个API完成，也就是把这个API当成一个估计器来使用就行

参数：

estimator：估计器对象

param_grid：估计器参数(dict)，字典的值可以是列表或是其他序列，序列每一个值就是参数可取的值，但是注意：字典的键，必须是estimator里的参数名字

cv：指定几折交叉验证，一般是10折以下

输出方法：直接用 . 调用

fit()：输入训练数据

score()：准确率

结果分析：

最佳参数：best_params_

最佳结果：best_socre_

最佳估计器：best_estimator_

交叉验证结果：cv_results_

上面KNN第四步的步骤，其他方面几乎完全相同

```
k_dict = {'n_neighbors':[1,3,5,7]}
estimator = KNeighborsClassifier()
a = GridSearchCV(estimator=estimator,param_grid=k_dict,cv=10)
a.fit(x_train,y_train)
print(a.best_params_)
score = a.score(x_test,y_test)
```

分类的评估方法

精确率与召回率

混淆矩阵：预测和正确结果，存在4种不同的组合

		预测结果	
		正例	假例
真实结果	正例	真正例TP	伪反例FN
	假例	伪正例FP	真反例TN

精确率（Precision）：预测结果为正例的部分，确实为正例的比例：TP+FP分之TP

召回率（Recall）：真实为正例的部分，预测结果也是正例的比例：TP+FN分之TP

F-score：反映模型的稳健性

$$F1 = \frac{2TP}{2TP + FN + FP}$$

衡量样本不均匀下的评估：ROC曲线和AUC指标

回归

2022年11月4日　　22:02

回归问题：目标值为连续型，这里主要学习线性回归

若目标值是非连续型的，都可划为广义上的分类问题

线性模型（广义）：如果某个自变量只受最多一个参数的影响，则可称之为线性模型

下面两种情况都是线性模型：

1、自变量最高次数只有一次（最常理解的），存在线性关系

2、自变量有高次项，或有其他函数关系，即存在非线性关系，也就是说可以用曲线来拟合

下面两图中。第一个是线性模型，第二个不是，因为w5和w1都影响到了x1，w5和w2都影响x2

$$y = \frac{1}{1 + e^{w0+w1*x1+w2*x2}}$$

$$y = \frac{1}{1 + w_5 * e^{w0+w1*x1+w2*x2}}$$

线性关系、非线性关系都是可能用线性模型

神经网络是非线性模型，即对于某个自变量有多个参数会影响

线性回归（Linear regression）：是利用多元线性回归方程（函数）对一个或多个自变量（特征值）和因变量（目标值）之间关系进行建模的一种分析方式

只有一个自变量，则称为一元线性回归，否则称为多元线性回归

下面的线性回归通用公式右侧，w和x都是矩阵。b称为偏置，wi称为回归系数，两者统称为模型参数

$$h(w) = w_1x_1 + w_2x_2 + w_3x_3 \ldots + \mathrm{b} = w^Tx + b$$

损失函数（cost）：求使损失函数最小时候的参数取值，即参数 = arg min(j())

$$J(\theta) = (h_w(x_1) - y_1)^2 + (h_w(x_2) - y_2)^2 + \cdots + (h_w(x_m) - y_m)^2$$

$$= \sum_{i=1}^{m}(h_w(x_i) - y_i)^2$$

即每个样本点到回归直线的距离的平方和（避免负值），本质上就是最小二乘法

注意：这里点到直线的距离，不是垂线的距离，而是通过样本做平行y轴的线，与回归直线的交点之间的距离（残差平方和）

需要明确的是，损失函数本质上是参数w的函数，随着模型参数不断变化，损失函数的大小也会发生变化

优化方法：如何求得模型中的参数，使得损失最小

优化方法就是使用不同的算法，来对loss函数进行处理，使其达到最小值，并得到这个时候的参数，达到目的

1、正规方程：公式一步到位，直接求到最好的结果，但特征过多时求解非常复杂，速度慢，适用于小数据

$$w = (X^T X)^{-1} X^T y$$

来源：根据矩阵形式的损失函数，矩阵求导，求最值，得到这个解，一步到位

sklearn的API：sklearn.linear_model.LinearRegression(fit_intercept=True)

通过正规方程优化

fit_intercept：是否计算偏置

LinearRegression.coef_：回归系数

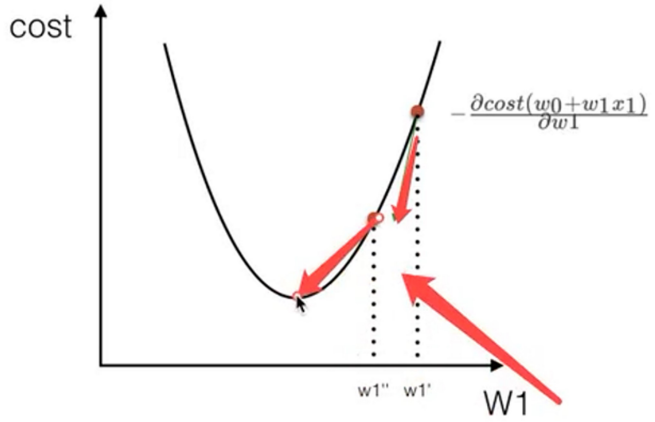
LinearRegression.intercept_：偏置

2、梯度下降（Gradient Descent）：迭代算法之一，不断迭代达到最优，通用性强、适用广泛

$$w1 := w1 - \alpha \frac{\partial cost(w0+w1x1)}{\partial w1}$$

$$w0 := w0 - \alpha \frac{\partial cost(w0+w1x1)}{\partial w1}$$

注意损失函数只是权重的函数，跟x无关，x只是一个参数



其中α为学习速率，又称步长step，为正数，需手动指定（超参数），在梯度下降算法，越接近“谷”的时候，步长应该变得越小，这样才能有一定精度

α旁边的整体表示方向沿着这个函数下降的方向找，最后就能找到山谷的最低点，然后更新w值，这一部分也称梯度

公式即表示：参数w1-α乘损失函数对w1的偏导，损失函数里的w0+w1x1表示损失函数是回归模型的函数，不用管

sklearn的API：sklearn.linear_model.SGDRegressor(loss='squared_loss', fit_intercept=True, learn_rate='invscaling', eta0=0.01)

SGD：随机梯度下降（Stochastic Gradient Descent）：从样本中随机抽取一组，训练后按梯度，让参数更新一次，再抽取再更新。再每次迭代过程中，样本都要被随机打乱

优点：性能更好，容易实现，可能跨过鞍点继续迭代，且大样本时计算量更低

缺点：需要许多超参数（如迭代次数、正则项参数），对特征标准化比较敏感，且不能并行

SGD还有多种衍生版本：如mini-batch SGD，用的最多，现在有的API名字为batch实际为mini-batch SGD

梯度下降是一类算法，除了SGD，还有：

GD：普通梯度下降，需要计算所有样本的值才能得出梯度，计算量大，一般不用

SAG：随机平均梯度法，由于传统算法收敛慢而提出

SGDRegressor类实现了随机梯度下降算法，支持不同的loss函数和正则化惩罚项来拟合线性回归模型

loss：损失类型，默认普通最小二乘法

fit_intercept：是否计算偏置

learn_rate：指定学习速率的算法，因为学习速率需要慢慢变小

constant：常数，学习速率保持不变，通过eta0指定这个常数，不精确。

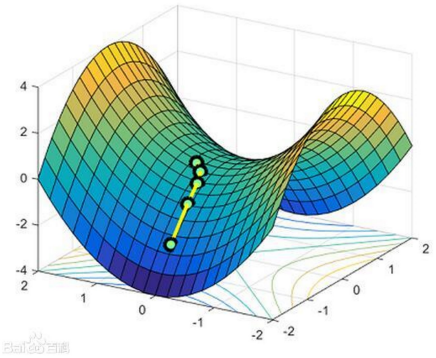
optimal、invscaling：得到α的算法，不需要了解

SGDRegressor.coef_：回归系数

SGDRegressor.intercept_：偏置

问题：可能找到局部最优，或者找到鞍点（更重要），导致无法继续迭代

鞍点：既不是最大值也不是最小值的临界点（泛函），一个数在所在行是最大值在所在列是最小值（矩阵），一个方向上是最大值，但在另一个方向上却是最小值（普遍）。



两种优化方法对比：

梯度下降：需要选择学习率，需要迭代求解，特征数量（维数）较大可以使用

正规方程：不需要选择学习率，一次运算得出，需要计算方程，时间复杂度高

线性回归算法：

LASSO回归和岭回归（Ridge）：带有正则化的线性回归，岭回归用的更多

正则化力度越大，权重系数（参数）越小，用于解决过拟合问题

L1正则化：使一些特征的参数w接近于0，减少这个特征的影响

和L2正则化类似，同样有惩罚系数，只是不是wi的平方加和，而是绝对值加和

L2正则化用的更多，比较缓和

这种方法也被称为LASSO回归，L2正则化方法也被称为岭回归（Ridge）

L2正则化：使得一些w直接等于0，相当于直接删除这个特征的影响

优点：越小的参数说明模型越简单，越简单的模型越不容易产生过拟合

Ridge回归（岭回归，一种改良的最小二乘法）：

岭回归的损失函数为：本质上是损失函数+惩罚项，λ为惩罚系数（超参数），乘二分之一是为了求导方便

使损失函数最小，需要使原损失函数+惩罚项的和最小，降低高次项的影响

$$J(w)=\frac{1}{2m}\sum_{i=1}^{m}(h_w(x_i)-y_i)^2+\lambda\sum_{j=1}^{n}w_j^2$$

> 注：m为样本数，n为特征数

sklearn中的API：sklearn.linear_model.Ridge(alpha=1.0, fit_intercept=True, solver='auto',normalize=False)

具有L2正则化的线性回归

alpha：正则化力度，也就是λ，取值为$0-1$，$1-10$

fit_intercept=True：表示添加偏置，建议添加

solver：'auto'会根据数据自动选择优化方法

normalize：数据是否进行标准化

normalize=False：可以在fit之前调用preprocessing.StandardScaler标准化数据，True就不用手动标准化了

Ridge.coef_：回归权重

Ridge.intercept_：回归偏置

逻辑回归（Logistic Regression）：是分类算法，主要解决二分类问题

所以逻辑回归只会输出0或1

逻辑回归是机器学习中一种分类模型，即分类算法，因为回归之间有联系所以名字带回归

原理：线性回归的输出，就是逻辑回归的输入

利用回归的思想，计算出预测值，用来进行分类

激活函数（activation function）：其中一种

为：$\frac{1}{1+e^{-x}}$（学名Logistic function，但其在sigmod函数中最具代表性，因此也直接称之为sigmod函数），x就是回归的输出，可以直接将回归的公式代替x

sigmod函数是一类S型函数的通称

激活函数的目的是将回归得到的值转换到0-1区间，使其符合概率的含义（因为逻辑斯蒂回归是二分类问题）

在线性模型里，激活函数还有第二个作用：使模型转换成非线性的

输出结果：[0，1]区间的一个概率值，默认0.5为阈值，如果大于阈值就认为属于这个类

逻辑回归的损失函数：对数似然损失（BCE损失：交叉熵损失函数）

$$cost(h_\theta(x),y)=\begin{cases}-log(h_\theta(x)) & \text{if y=1}\\ -log(1-h_\theta(x)) & \text{if y=0}\end{cases}$$

加入激活函数后，我们不是在比较距离的差异，本质上在比较两个分布的差异，使用KL散度或交叉熵来进行判断。图中最下面的式子就是交叉熵公式（一般用在离散分布）

$P_D(x=1)=0.2$　　$P_T(x=1)=0.3$

$P_D(x=2)=0.3$　　$P_T(x=2)=0.4$

$P_D(x=3)=0.5$　　$P_T(x=3)=0.3$

$$\sum_i P_D(x=i)\ln P_T(x=i)$$

将线性回归的最小二乘法损失函数和y的值，新构建一个对数似然损失函数，y=1表示这个样本确实属于这一类

sklearn的API：sklearn.linear_model.LogisticRegression(solver='liblinear', penalty='l2', C=1.0)

solver：优化求解方式，默认使用开源的liblinear库，内部使用坐标轴下降算法

penalty：正则化种类

C：正则化力度

回归性能评估：均方误差（MSE）准则

MSE = E $(\widehat{\theta_i} - \theta)^2$

即均方误差为：每个特征的估计值与实际值之差的平方，加和然后除以个数

可分解为：MSE = θ_i的方差 + 系统误差

sklearn中的API：sklearn.metrics.mean_squared_error(y_true, y_pred)，返回浮点数结果

欠拟合和过拟合：

欠拟合：特征学习过少，模型泛化能力太强，导致判定过于宽泛

模型过于简单，在训练数据上不能很好拟合，在测试数据上也不能很好拟合

判定图片是否为鸭子，只学到了有翅膀这一个特征，导致把鹅、鸡等均判定到了鸭子

解决：增加数据的特征数量

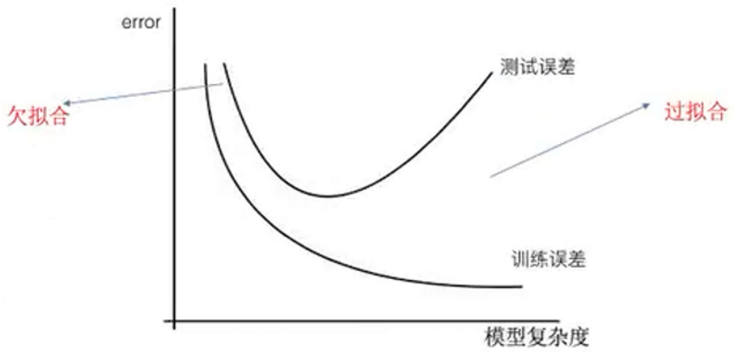
过拟合：特征学习过多，模型没有泛化能力，存在一些嘈杂特征，导致判定过于严密

模型过于复杂，在训练集上拟合非常好，但在测试集上拟合不佳

判定图片是鸭子，由于训练集都是白色的鸭子，导致认为鸭子必须是白色的，然后总把黑色的鸭子误判到其他类

解决：正则化、删除或合并特征

下图，过拟合和欠拟合的临界点在测试误差和训练误差函数的差的最小值处（图不精确）



回归的选择：

小规模数据：LinearRegression：线性回归（不能解决拟合问题）、岭回归

大规模数据：SGDRegressor：随机梯度下降回归算法

聚类

2022年11月5日　　23:53

无监督学习：没有目标值，主要包含两类问题：

聚类：K-means、系统聚类

降维：PCA（主成分）、FA（因子）

无监督学习基本都可以抽象成以上两类问题

K-means算法：

k为超参数，通过需求（要聚成多少类）或者网格搜索得到

原理：去多元统计分析看

sklearn的API：sklearn.cluster.Kmeans(n_cluster=8, init='k-means++')

n_cluster：聚成多少类，即k的值

init：初始化方法

聚类效果指标：

轮廓系数：高内聚、低耦合原理，即外部距离最大化，内部距离最小化

$$SC_i = \frac{b_i - a_i}{\max(b_i, a_i)}$$

a、b为两个类

轮廓系数取值为[-1，1]，越接近1效果越好，越接近-1越差

sklearn的API：sklearn.metrics.silhouette_score(X, labels)

计算所有样本的平均轮廓系数

X：特征值（矩阵）

labels：被聚类标记的目标值（矩阵）

优缺点：

采用迭代式算法，直观易懂且实用

容易收敛到局部最优解：比如开始分配的k个点靠得非常近

采用多次聚类解决