机器学习概述

机器学习简要介绍

特征工程

分类算法及实例

01 **机器学习简要介绍**

02 特征工程

03 分类算法及实例

01

机器学习简要介绍

01 定义

02 应用领域

03 开发流程

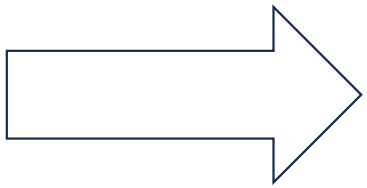
04 算法分类

05 机器学习工具

Large Language Model (LLM) 大语言模型

ChatGPT

文心一言



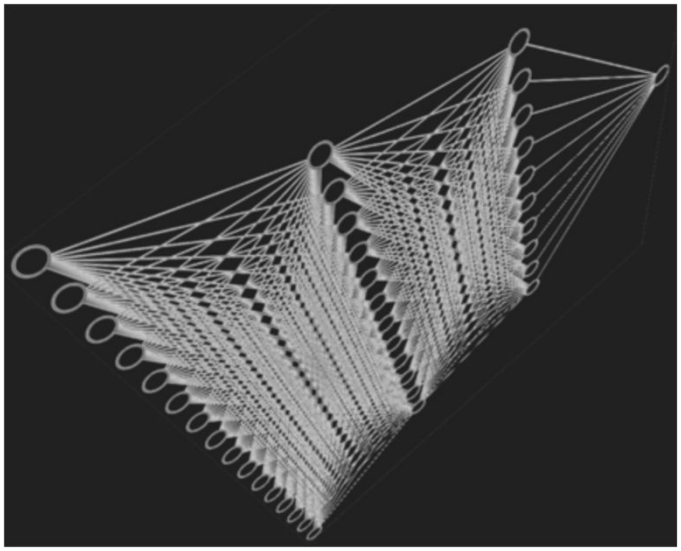
功能：

1. 信息检索
2. 指导与建议
3. 闲聊和娱乐

……

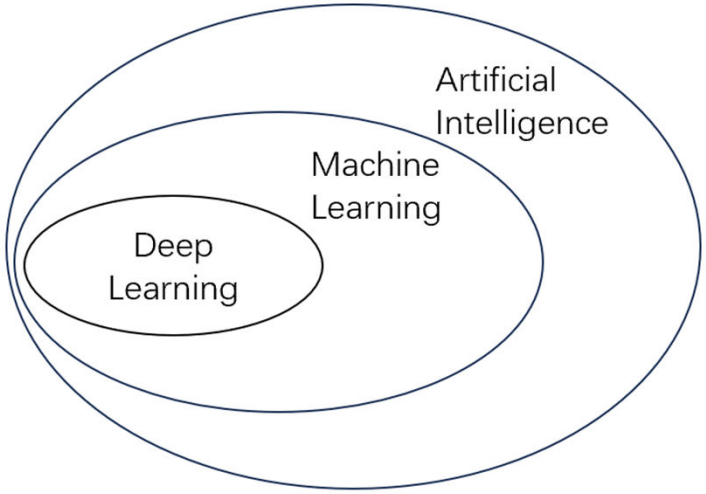
1.1 Definition

- Machine learning (ML) is a field of study in artificial intelligence concerned with the development and study of statistical algorithms that can effectively generalize and thus perform tasks without explicit instructions.
- 机器学习（ML）是人工智能的一个研究领域，涉及统计算法的开发和研究，这些算法可以有效地泛化，从而在没有明确指令的情况下执行任务。

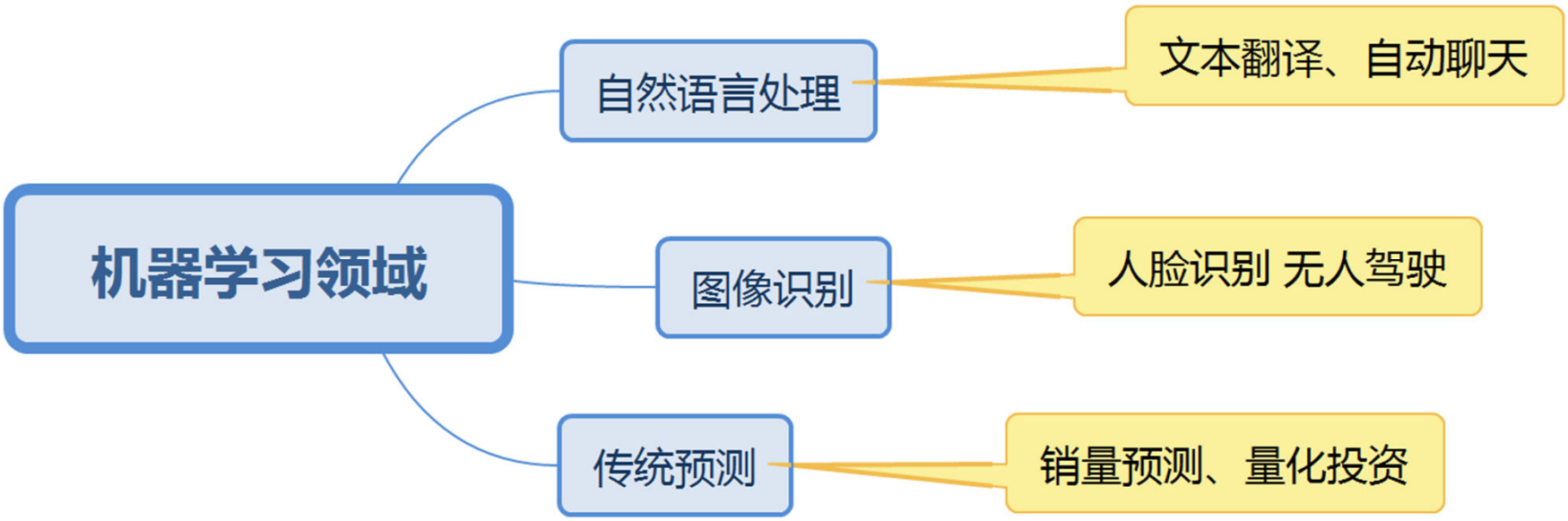


Evolution of Artificial Intelligence

- **Artificial Intelligence (1950s)** – Giving intelligence to machine
- **Machine Learning (1980s)** – realizing artificial intelligence
- **Deep Learning (2006)** –for higher prediction accuracy



1.2 应用领域



1.3 机器学习开发流程

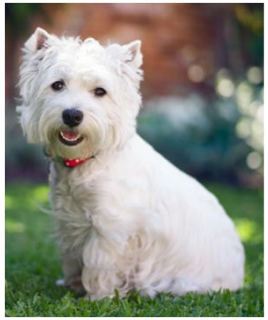
- 机器学习：数据中学习 -> 模型 -> 预测
- 人：经验中总结 -> 规律 -> 预测

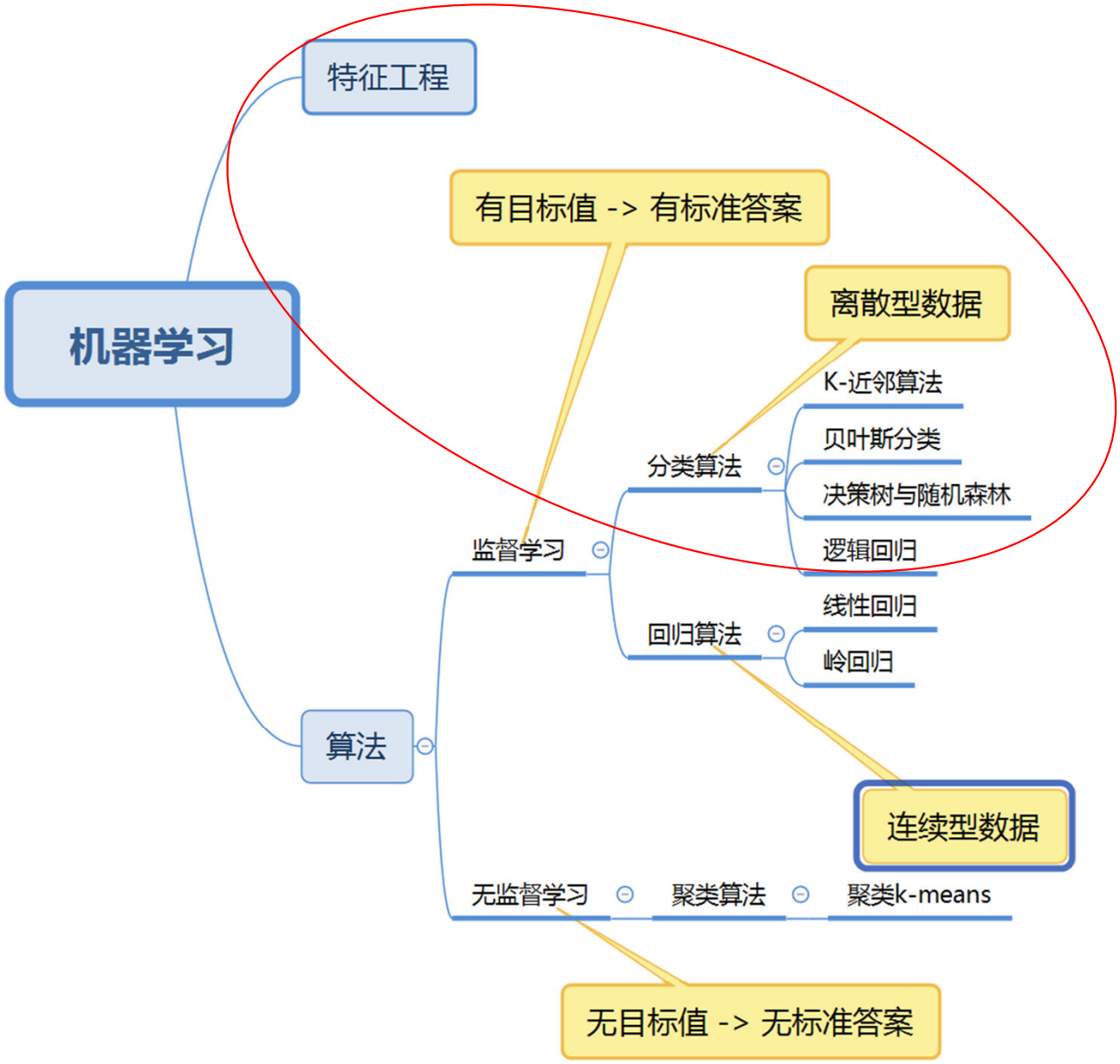
- 1、获取数据
- 2、数据处理
- 3、特征工程
- 4、机器学习算法训练 -> 模型
- 5、模型评估
- 6、应用

1.4 机器学习算法分类

数据集：特征值+目标值（不一定有）

数据集	特征值	目标值
大量猫狗图片	猫狗图片的特征	分类结果：猫、狗
房屋的各种信息	房子面积、位置、楼层	预测结果：房屋价格




机器学习
特征工程
算法
有目标值 -> 有标准答案
监督学习
离散型数据
分类算法
K-近邻算法
贝叶斯分类
决策树与随机森林
逻辑回归
回归算法
线性回归
岭回归
连续型数据
无监督学习
聚类算法
聚类k-means
无目标值 -> 无标准答案

- 算法-核心-高校学者教授课题

掌握算法基本思想 & 会调用算法

特征工程、调参、优化

- 算法背后的数学原理-数学学院的专必专选课程

数理统计、数值分析、应用回归分析、非参数统计、多元统计分析及应用……

1.5 机器学习工具

- **Jupyter notebook** 编程+写作

学习简单，易上手

代码块、单元格方式运行 -> 调试，debug

可插入 Rmarkdown 说明性文字+图片

支持多种编程语言 Ju-Julia; pyt – python; r-R

- **Google colab** https://colab.research.google.com/

基于云端的免费交互式开发环境，由Google提供，用户不需要在本地安装和配置软件环境，只需通过浏览器访问即可开始编写和运行代码

提供了一个基于Jupyter Notebook的环境

还提供了GPU和TPU等硬件加速选项，使用户能够更快地训练深度学习模型

01 机器学习简要介绍

02 **特征工程**

03 分类算法及实例

02

特征工程

01 数据集

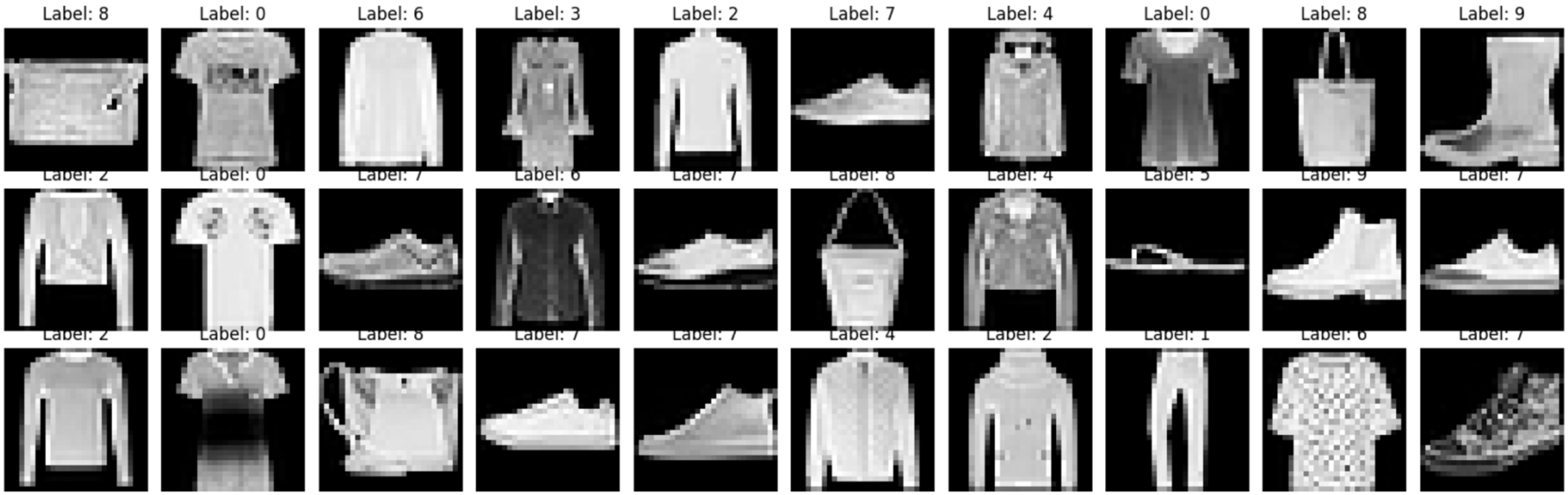
02 特征工程简介

2.1 数据集

- 可用数据集
- scikit-learn https://scikit-learn.org/stable/ 数据量较小
- Kaggle https://www.kaggle.com/ 真实数据，数据量大
- UCI https://archive.ics.uci.edu/

Fashion-MNIST 数据集

Fashion-MNIST数据集包含了 10 个类别的灰度图像，每个类别包含了 6000 张尺寸为 28x28 像素的图像，分别是：t-shirt（T恤），trouser（牛仔裤），pullover（套衫），dress（裙子），coat（外套），sandal（凉鞋），shirt（衬衫），sneaker（运动鞋），bag（包），ankle boot（短靴）。



数据集的划分

训练数据 -> 构建模型 70-80%

测试数据 -> 评估模型 20-30%

加载数据

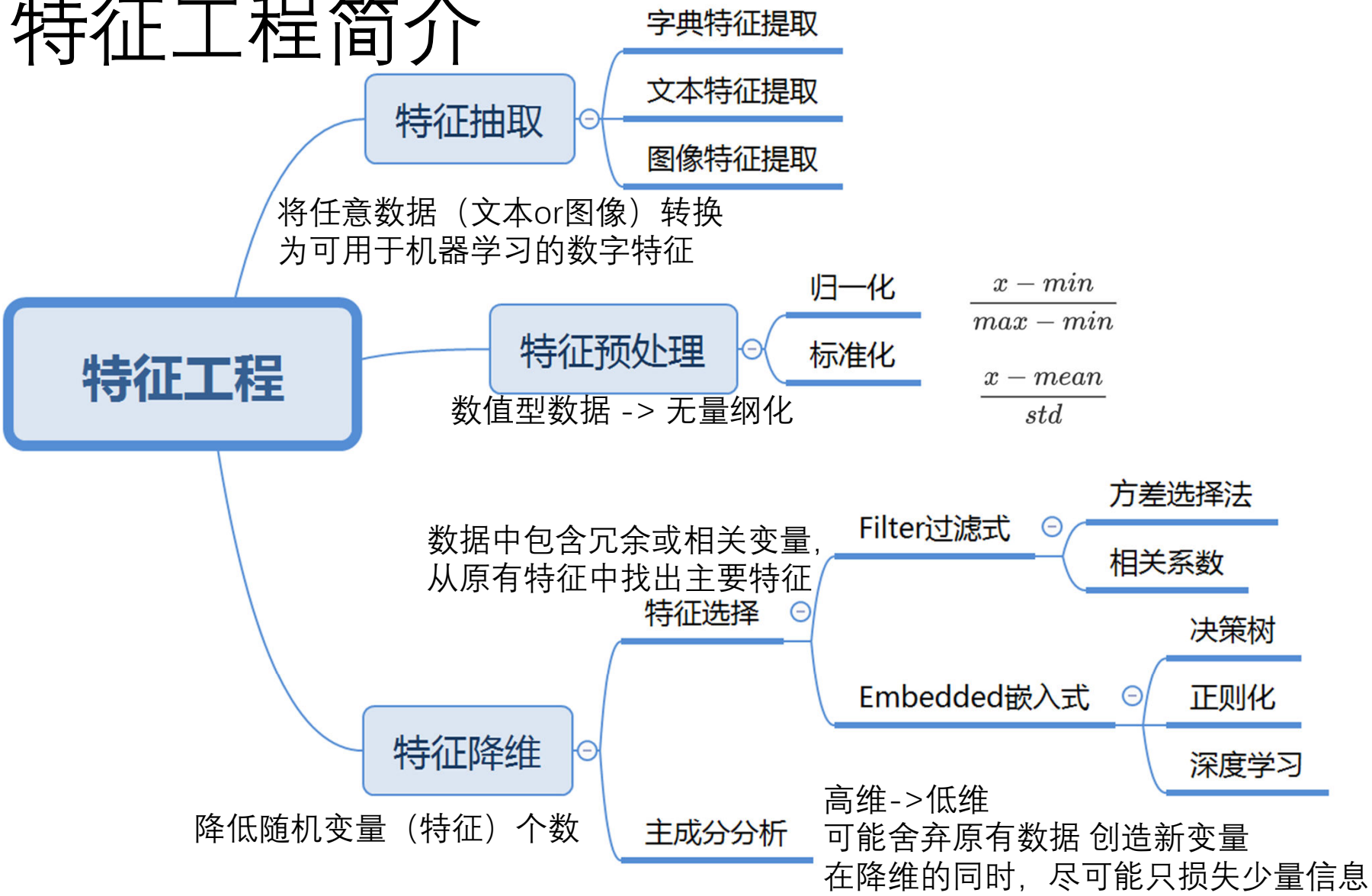
```
[ ]  from sklearn.datasets import fetch_openml
     from sklearn.model_selection import train_test_split

     # 加载Fashion-MNIST数据集
     fashion_mnist = fetch_openml(name='Fashion-MNIST', version=1)    #fetch_openml函数从Fashion-MNIST数据集中加载数据

     # 准备特征和标签
     X = fashion_mnist.data    #特征值
     y = fashion_mnist.target    #目标值

     # 将特征和标签划分为训练集和测试集
     x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)    #测试数据20%，随机种子42
```

2.2 特征工程简介


特征工程
特征抽取
字典特征提取
文本特征提取
图像特征提取
将任意数据（文本or图像）转换
为可用于机器学习的数字特征
特征预处理
归一化
标准化
$\frac{x-min}{max-min}$
$\frac{x-mean}{std}$
数值型数据 -> 无量纲化
特征降维
数据中包含冗余或相关变量，
从原有特征中找出主要特征
特征选择
Filter过滤式
方差选择法
相关系数
Embedded嵌入式
决策树
正则化
深度学习
主成分分析
高维->低维
可能舍弃原有数据 创造新变量
在降维的同时，尽可能只损失少量信息
降低随机变量（特征）个数

数据标准化 $\frac{x - mean}{std}$

```
from sklearn.datasets import fetch_openml
from sklearn.model_selection import train_test_split

# 加载Fashion-MNIST数据集
fashion_mnist = fetch_openml(name='Fashion-MNIST', version=1)    #fetch_openml函数从Fashion-MNIST数据集中加载数据

# 准备特征和标签
X = fashion_mnist.data    #特征值
y = fashion_mnist.target    #目标值

# 将特征和标签划分为训练集和测试集
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)    #测试数据20%，随机种子42
```

```
from sklearn.preprocessing import StandardScaler    #StandardScaler模块用于数据标准化

# 数据标准化
scaler = StandardScaler()
x_train_scaled = scaler.fit_transform(x_train)
x_test_scaled = scaler.transform(x_test)
```

机器学习简要介绍

特征工程

分类算法及实例

03

分类算法及实例

01 K-近邻算法

02 朴素贝叶斯算法

03 决策树

04 随机森林

05 支持向量机

3.1 K-近邻算法

- 在K-NN分类中，输出是一个分类族群。

一个对象的分类是由其邻居的“多数表决”确定的，k个最近邻居（k为正整数，通常较小）中最常见的分类决定了赋予该对象的类别。若K = 1，则该对象的类别直接由最近的一个节点赋予。

- 在k-NN回归中，输出是该对象的属性值。该值是其k个最近邻居的值的平均值。

- 如何确定邻居?
- 距离公式 $a(a_1, a_2, a_3), b(b_1, b_2, b_3)$

-> 欧氏距离（最常见）

$$\sqrt{(a_1 - b_1)^2 + (a_2 - b_2)^2 + (a_3 - b_3)^2}$$

-> 曼哈顿距离

$$|a_1 - b_1| + |a_2 - b_2| + |a_3 - b_3|$$

K值过小，容易受异常点影响；K值过大，容易受样本不均衡的影响
-> 无量纲化的处理 标准化

```
import time
import numpy as np
from sklearn.neighbors import KNeighborsClassifier    #KNeighborsClassifier模块用于KNN分类器
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

# 创建和拟合KNN模型
start_time = time.time()
knn = KNeighborsClassifier(n_neighbors=5)    #设置n_neighbors参数为5，即K值为5
knn.fit(x_train_scaled, y_train)
print('training took %fs!' % (time.time() - start_time))

# 进行预测并计算性能指标
start_time = time.time()
pred_knn = knn.predict(x_test_scaled)
print('predict took %fs!' % (time.time() - start_time))
report_knn = classification_report(y_test, pred_knn)
confusion_mat_knn = confusion_matrix(y_test, pred_knn)
accuracy_knn = accuracy_score(pred_knn, y_test)

# 打印结果
print(report_knn)
print(confusion_mat_knn)
print('KNN准确率: %0.4lf' % accuracy_knn)
```

KNN准确率: 0.8571

- **优点：**

简单直观：KNN算法的原理简单明了，易于理解和实现。

对异常值不敏感：由于KNN采用了多个邻居进行决策，对于异常值的影响较小，具有一定的鲁棒性。

适应多分类问题：KNN可以处理多类别分类问题，并且对于每个类别的样本数量不平衡也没有要求。

- **缺点：**

计算复杂度高：KNN需要计算待预测样本与所有训练样本之间的距离，对于大规模数据集而言，计算复杂度较高。

内存消耗大：KNN需要存储所有训练样本的特征信息，对于大规模数据集和高维数据而言，内存消耗较大。

需要选择合适的K值：KNN的性能受到K值的影响，需要通过交叉验证等方法选择合适的K值。

- **使用场景：**
- 分类和回归问题：KNN适用于分类和回归任务，尤其对于边界明显、数据分布不规则的问题表现良好。
- 小规模数据集：KNN在处理小规模数据集时具有较好的性能，因为它可以直接利用所有训练样本进行预测。
- 非参数模型：KNN是一种非参数模型，不需要对数据做任何假设，因此对数据分布没有要求。

3.2 朴素贝叶斯算法

- 朴素贝叶斯算法是基于贝叶斯定理与特征条件独立假设的分类方法。
- 贝叶斯定理：它根据以往经验和分析得到的概率，即先验概率，以及事件已经发生的条件下，事件发生的概率，即条件概率，来计算两个事件共同发生的概率。
- 特征条件独立假设：每个特征都被假设为条件独立的，这意味着每个特征的变化不会影响其他特征的概率分布。因此，朴素贝叶斯算法是一种生成模型，它在学习过程中学习生成数据的机制。

$$P(Y|X) = \frac{P(X|Y)P(Y)}{P(X)}$$

P(Y|X) 表示在给定观测数据 X 的条件下，事件 Y 发生的概率。

P(X|Y) 表示在事件 Y 发生的条件下，观测数据 X 出现的概率。

P(Y) 表示事件 Y 发生的先验概率。

P(X) 表示观测数据 X 出现的概率。

```
import  time
import  numpy  as  np
from  sklearn.naive_bayes  import  GaussianNB    #GaussianNB模块用于朴素贝叶斯分类器
from  sklearn.metrics  import  classification_report,  confusion_matrix,  accuracy_score

#  创建和拟合朴素贝叶斯模型
start_time  =  time.time()
nb  =  GaussianNB()    #使用高斯朴素贝叶斯模型GaussianNB()
nb.fit(x_train_scaled,  y_train)
print('training  took  %fs!'  %  (time.time()  -  start_time))

#  进行预测并计算性能指标
start_time  =  time.time()
pred_nb  =  nb.predict(x_test_scaled)
print('predict  took  %fs!'  %  (time.time()  -  start_time))
report_nb  =  classification_report(y_test,  pred_nb)
confusion_mat_nb  =  confusion_matrix(y_test,  pred_nb)
accuracy_nb  =  accuracy_score(pred_nb,  y_test)

#  打印结果
print(report_nb)
print(confusion_mat_nb)
print('朴素贝叶斯准确率:  %0.4lf'  %  accuracy_nb)
```

朴素贝叶斯准确率: 0.5753

- **优点：**

算法简单快速：朴素贝叶斯算法的计算速度快，模型训练和预测的时间复杂度较低，适用于大规模数据集和实时应用。

对小样本数据有效：朴素贝叶斯算法在小样本数据集上表现良好，因为它通过条件独立性假设充分利用了特征之间的关系。

对缺失数据鲁棒性强：朴素贝叶斯算法对于缺失数据具有较好的鲁棒性，可以在存在缺失数据的情况下进行分类。

- **缺点：**

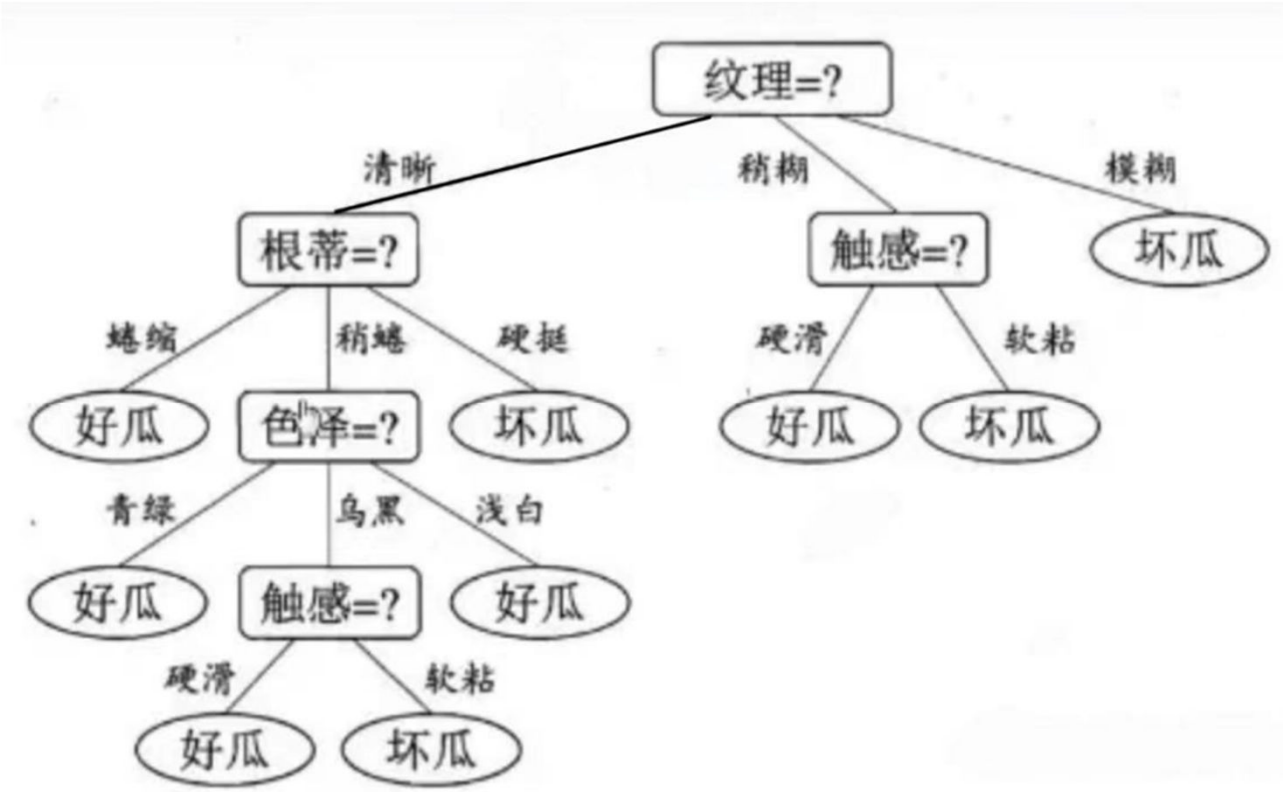
独立性假设限制：朴素贝叶斯算法假设特征之间是独立的，这在某些情况下可能不符合实际情况。如果特征之间具有强相关性，可能导致分类性能下降。

对输入数据的分布假设较强：朴素贝叶斯算法假设特征的分布是离散的或高斯分布的，这对于某些数据集可能不适用。

无法处理连续特征：朴素贝叶斯算法通常不能很好地处理连续特征，需要进行离散化或采用其他方法进行处理。

- **使用场景：**
- 文本分类：朴素贝叶斯算法在文本分类问题中广泛应用，如垃圾邮件识别、情感分析等。它可以根据单词或词组在文本中的出现情况，判断文本所属的类别。
- 垃圾过滤：朴素贝叶斯算法可以根据邮件中包含的特定单词或短语的概率，将邮件划分为垃圾邮件或非垃圾邮件。
- 推荐系统：朴素贝叶斯算法可以用于个性化推荐系统中，根据用户的历史行为和特征，预测用户对不同项目的兴趣。

3.3 决策树



决策树是一种基于树形结构的监督学习算法，用于分类和回归任务。决策树通过一系列的判断条件来对样本进行分类或预测，每个判断条件对应于树的一个分支，最终导向叶节点的类别或预测值。

```
import  time
import  numpy  as  np
from  sklearn.tree  import  DecisionTreeClassifier    #DecisionTreeClassifier模块用于决策树分类器
from  sklearn.metrics  import  classification_report,  confusion_matrix,  accuracy_score

#  创建和拟合决策树模型
start_time  =  time.time()
dt  =  DecisionTreeClassifier()
dt.fit(x_train_scaled,  y_train)
print('training  took  %fs!'  %  (time.time()  -  start_time))

#  进行预测并计算性能指标
start_time  =  time.time()
pred_dt  =  dt.predict(x_test_scaled)
print('predict  took  %fs!'  %  (time.time()  -  start_time))
report_dt  =  classification_report(y_test,  pred_dt)
confusion_mat_dt  =  confusion_matrix(y_test,  pred_dt)
accuracy_dt  =  accuracy_score(pred_dt,  y_test)

#  打印结果
print(report_dt)
print(confusion_mat_dt)
print('决策树准确率:  %0.4lf'  %  accuracy_dt)
```

决策树准确率: 0.7909

- **优点：**

可解释性强：决策树的结构直观，易于理解和解释，可以生成具有可读性的规则。

处理混合数据类型：决策树可以处理同时包含连续型和离散型特征的数据集。

不需要特征缩放：决策树对特征的缩放不敏感，不需要对特征进行归一化或标准化处理。

- **缺点：**

容易过拟合：决策树容易过度拟合训练数据，特别是当树的深度较大或者分支条件过于复杂时。

不稳定性：对于数据中的微小变化非常敏感，可能导致树结构的不稳定性，需要使用剪枝等方法进行处理。

难以处理特征关联性较强的问题：决策树难以处理特征之间存在强相关性的问题，可能导致模型偏向某些特征。

- **使用场景：**
- 分类和回归问题：决策树适用于分类和回归任务，可以处理离散型和连续型特征。
- 多类别问题：决策树可以处理多类别分类问题。
- 可解释性要求高的问题：决策树的结构清晰，可以提供可解释性较强的结果，便于理解和解释。

3.4 随机森林

- 随机森林（Random Forest）是一种集成学习方法，基于决策树构建而成。它通过构建多个决策树，并结合它们的预测结果来进行分类或回归任务（众数）。
- 随机森林的原理基于"集思广益"的概念，通过组合多个决策树的预测结果来提高整体模型的准确性和泛化能力。


随机森林
随机
训练集随机
bootstrap 随机有放回抽样
特征随机
N 个样本中随机有放回的抽样 n 个
从 M 个特征中随机抽取 m 个特征，M>>m
降维
包含多个决策树的分类器

```
import time
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix, accuracy_scor

# 创建和拟合随机森林模型
start_time = time.time()
rf = RandomForestClassifier(n_jobs=-1)
rf.fit(x_train, y_train)
print('training took %fs!' % (time.time() - start_time))

# 进行预测并计算性能指标
start_time = time.time()
pred_rf = rf.predict(x_test)
print('predict took %fs!' % (time.time() - start_time))
report_rf = classification_report(y_test, pred_rf)
confusion_mat_rf = confusion_matrix(y_test, pred_rf)
accuracy_rf = accuracy_score(pred_rf, y_test)

# 打印结果
print(report_rf)
print(confusion_mat_rf)
print('随机森林准确率: %0.4lf' % accuracy_rf)
```

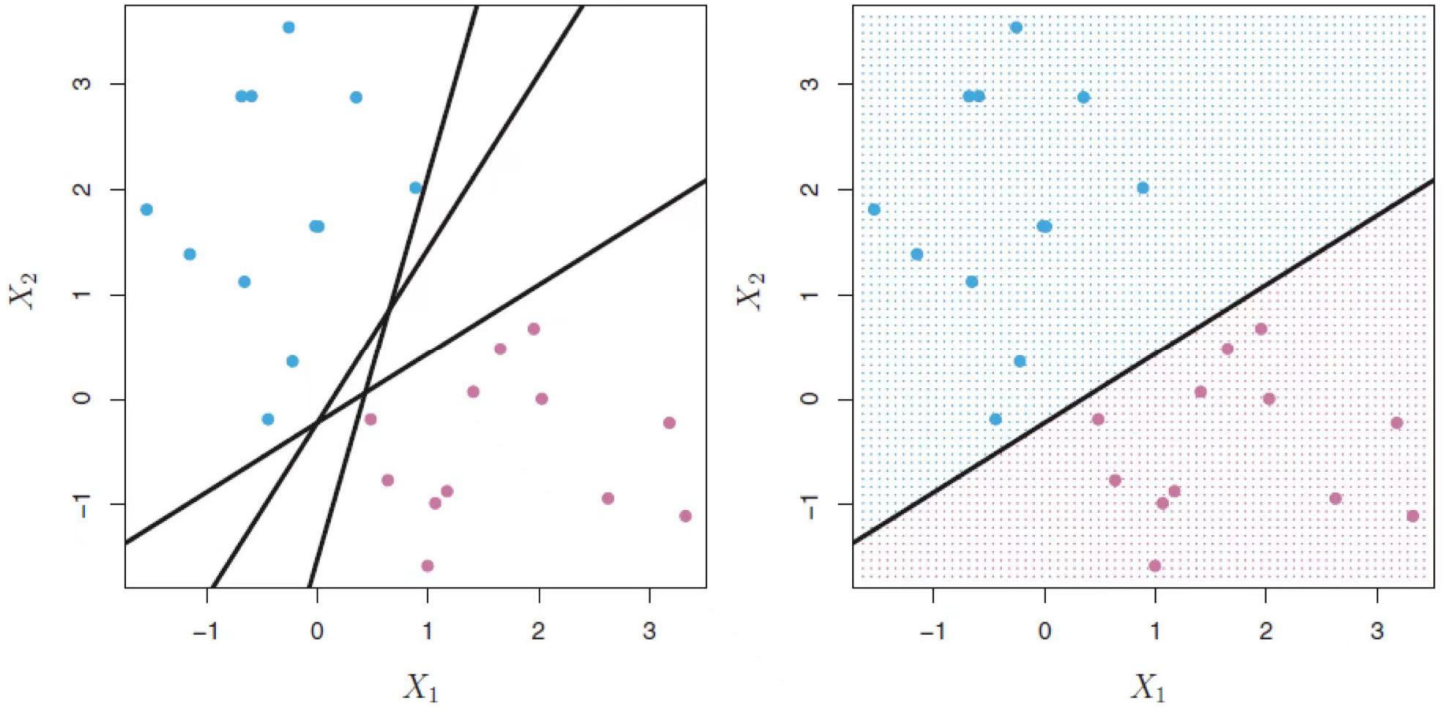
随机森林准确率: 0.8854

- **优点：**
- 高准确性：随机森林通过集成多个决策树的预测结果，可以提供更准确的分类或回归结果。
- 抗噪声能力强：由于随机森林采用随机采样和随机特征选择，对于噪声数据具有较好的鲁棒性。
- 可解释性较强：随机森林可以提供特征重要性排序，帮助理解数据中的关键特征。
- **缺点：**
- 训练时间较长：由于随机森林需要构建多个决策树，训练时间相对较长，尤其是对于大规模数据集和深度较大的决策树。
- 内存消耗较大：随机森林需要存储多个决策树的模型，因此需要较大的内存空间。
- 参数调节困难：随机森林中的参数选择对模型的性能有重要影响，需要进行适当的调参。

- **使用场景：**
- 分类和回归问题：随机森林适用于分类和回归任务，对于多类别分类和复杂非线性关系的问题表现良好。
- 大规模数据集：由于随机森林可以并行化处理，因此适用于大规模数据集的训练和预测。
- 处理高维数据：随机森林在处理高维数据时具有良好的性能，可以自动选择重要特征，不需要降维。

3.5 支持向量机

支持向量机（Support Vector Machine，SVM）是一种强大的监督学习算法，用于分类和回归任务。SVM 的原理基于统计学习理论中的结构风险最小化原则，旨在找到一个最优的超平面或者决策边界，将不同类别的样本分隔开。



```
import  time
from  sklearn.svm  import  SVC
from  sklearn.metrics  import  classification_report,  confusion_matrix,  accuracy_score

#  创建和拟合支持向量机模型
start_time  =  time.time()
svm  =  SVC()
svm.fit(x_train_scaled,  y_train)
print('training  took  %fs!'  %  (time.time()  -  start_time))

#  进行预测并计算性能指标
start_time  =  time.time()
pred_svm  =  svm.predict(x_test_scaled)
print('predict  took  %fs!'  %  (time.time()  -  start_time))
report_svm  =  classification_report(y_test,  pred_svm)
confusion_mat_svm  =  confusion_matrix(y_test,  pred_svm)
accuracy_svm  =  accuracy_score(pred_svm,  y_test)

#  打印结果
print(report_svm)
print(confusion_mat_svm)
print('SVC准确率:  %0.4lf'  %  accuracy_svm)
```

支持向量机准确率: 0.8905

- **优点：**

在高维空间中有效：SVM 在高维空间中处理样本时具有良好的性能，可以处理具有复杂特征的问题。

可以解决非线性问题：通过使用不同的核函数，SVM 可以学习和处理非线性关系。

泛化能力强：SVM 通过最大化间隔来选择决策边界，有助于减少过拟合的风险。

- **缺点：**

对大规模数据集和高维数据敏感：SVM 对于大规模数据集和高维数据集的训练和预测需要较长的时间和较大的计算资源。

参数调节困难：SVM 中的参数选择对于模型的性能和泛化能力至关重要，但调节这些参数可能需要一些经验和领域知识。

不适用于非常噪声的数据集：SVM 对于噪声较多的数据集可能过于敏感，可能导致错误的分类。

- **使用场景：**
- 二分类问题：SVM 在二分类问题中表现出色，尤其在数据集维度较低或具有明显分隔边界的情况下。
- 文本分类、图像分类和人脸识别：SVM 在处理高维特征的任务中也有出色的应用，如文本分类、图像分类和人脸识别等。

总结

分类算法	准确率	训练时间 (s)	预测时间 (s)
支持向量机	0.8905	419.145885	279.511573
随机森林	0.8854	77.639878	0.577737
KNN	0.8571	0.31597	48.537325
决策树	0.7909	47.474089	0.025409
朴素贝叶斯	0.5753	0.933243	0.517503

谢谢！