

模拟练习

讲师名称：李川

CanWay 嘉为数字咨询

数 字 化 人 才 培 养 先 行 者



目录

1

数据分析实例

2

可视化实例

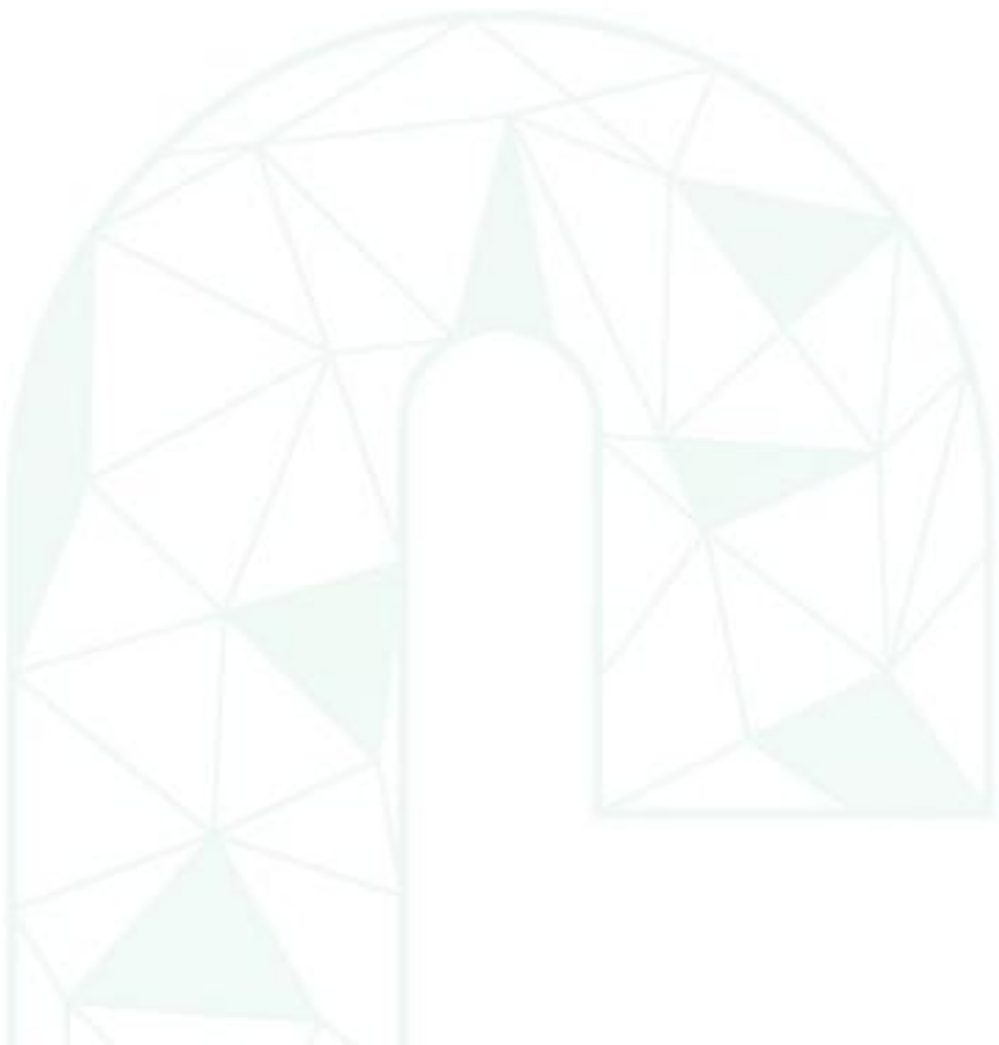
3

中文分词实例

4

机器学习实例

CONTENTS



/01

数据分析实例

1.1 数据分析基础实例

例6-1：使用协同过滤进行商品推荐。

问题描述：假设已有大量用户对若干商品的购买评分数据，现有某用户，也购买过一些商品并进行过评分，要求根据已有打分数据为该用户进行推荐。

基本思路：使用基于用户的协同过滤算法，也就是根据用户喜好来确定与当前用户最相似的用户，然后再根据最相似用户的喜好为当前用户进行推荐。本例采用字典来存放打分数据，格式为{用户1:{商品名称1: 打分1, 商品名称2: 打分2,...}, 用户2:{...}}，首先在已有数据中查找与当前用户共同打分商品（使用集合的交集运算）数量最多的用户，如果有多个这样的用户就再从中选择打分最接近（打分的差距最小）的用户。代码中使用到了random模块中的randrange()函数，用来生成指定范围内的一个随机数。

1.1 数据分析基础实例

```
from random import randrange

# 模拟历史商品评分数据，一共20个用户，商品一共15个，每个用户对4到9个商品进行随机评分
# 每个商品的评分最低1分最高5分
data = {'user'+str(i):{'product'+str(randrange(1, 16)):randrange(1, 6)
                        for j in range(randrange(4, 10))}
        for i in range(20)}

# 模拟当前用户评分数据，为5种随机商品评分
user = {'product'+str(randrange(1, 16)):randrange(1,6) for i in range(5)}
# 最相似的用户及其对商品评分情况，这个 lambda 函数返回一个元组，作为判断用户相似度的 “排序键”
# 第一个值：两个用户共同评分的商品的数量的负数
# 第二个值：sum(((历史用户评分 - 当前用户评分)2) for 共同商品)
f = lambda item:(-len(item[1].keys()&user.keys()),
                 sum(((item[1].get(product)-user.get(product))2)
                     for product in user.keys()&item[1].keys()))
similarUser, products = min(data.items(), key=f)
```

1.1 数据分析基础实例

```
# 在输出结果中，第一列表示两个人共同评分的商品的数量
# 第二列表示用户与当前用户的评分平方误差和（值越小越相似）
# 然后是该用户对商品的评分数据
print('known data'.center(50, '='))
for item in data.items():
    print(len(item[1].keys()&user.keys()),
          sum(((item[1].get(product)-user.get(product))**2
               for product in user.keys()&item[1].keys()))),
          item,
          sep=':')
print('current user'.center(50, '='))
print(user)
print('most similar user and his products'.center(50, '='))
print(similarUser, products, sep=':')
print('recommended product'.center(50, '='))
# 在当前用户没购买过的商品中选择评分最高的进行推荐
print(max(products.keys()-user.keys(), key=lambda product: products[product]))
```

1.2 数据统计分析实例

例6-2 读取企业的资产设备表文件，对表中不同数据中的缺失值、重复值采用合适的策略处理，并通过年份、使用部门、分类等维度对数据进行统计分析。

资产编号	资产分类	资产名称	数量	价值	财务入账日期	使用部门
15151900	同步照相机	数码相机	1	13960.00	2015-09-08	行政办公室
15187900	其他打印机	打印机	1	1180.00	2015-10-21	行政办公室
15188000	其他打印机	打印机		1180.00	2015-10-21	行政办公室
15190100	微型笔记本电子	微型笔记本电子计算机	1	4950.00	2015-10-28	行政办公室
15190200	微型笔记本电子	微型笔记本电子计算机	1	4950.00	2015-10-28	行政办公室
15192600	微型笔记本电子	微型笔记本电子计算机	1		2015-10-28	行政办公室
15325900	FT处理器	无线网络管理设备	1	8800.00	2015-12-02	研发一部
15326000	FT处理器	无线接入点	1	1280.00	2015-12-02	研发一部
15326100	FT处理器	无线接入点	1	1280.00	2015-12-02	研发一部
15326800	应用软件	网络管理软件	1	15480.00	2015-12-02	研发一部
15332500	微型台式电子计	微型台式电子计算机	1	4380.00	2015-12-05	研发一部
15332600	微型台式电子计	微型台式电子计算机	1	4380.00	2015-12-05	研发一部
15332700	微型台式电子计	微型台式电子计算机	1	4380.00	2015-12-05	研发一部
15332800	微型台式电子计	微型台式电子计算机	1	4380.00	2015-12-05	研发一部
TY2016001154	光电交换机（网	光电交换机（网络交换机）	1	1000.00	2016-10-19	

1.2 数据统计分析实例

```
import pandas as pd
import numpy as np
import os
from pathlib import Path
def analyze_assets(file_path):
    """处理资产设备表并进行统计分析"""
    # 1. 读取Excel文件
    try:
        # 根据文件后缀选择合适的引擎
        file_ext = Path(file_path).suffix.lower()
        engine = 'xlrd' if file_ext == '.xls' else 'openpyxl'
        df = pd.read_excel(file_path, engine=engine)
        print(f"成功读取文件: {file_path}, 原始数据共 {len(df)} 条记录, {len(df.columns)} 列")
        print("原始数据前5行:")
        print(df.head())
    except Exception as e:
        print(f"读取文件失败: {str(e)}")
        return None
```


1.2 数据统计分析实例

```
# 2. 数据清洗
print("\n" + "=" * 50 + "\n数据清洗:")
# 2.1 处理缺失值
missing_values = df.isnull().sum()
print("\n各列缺失值数量:")
print(missing_values[missing_values > 0]) # 只显示有缺失值的列
if missing_values.sum() > 0:
    # 对于数值列(数量、价值), 用均值填充
    numeric_cols = ['数量', '价值']
    for col in numeric_cols:
        if df[col].isnull().sum() > 0:
            #df[col] = df[col].fillna(df[col].mean())
            df[col] = df[col].fillna(df[col].mode()[0])
            print(f"已用均值填充'{col}'列的缺失值")
print(df)
```

1.2 数据统计分析实例

```
# 对于日期列，用众数填充
date_cols = ['财务入账日期']
for col in date_cols:
    if df[col].isnull().sum() > 0:
        df[col] = df[col].fillna(df[col].mode()[0])
        print(f"已用众数填充'{col}'列的缺失值")
# 对于其他列，用'前值'填充
other_cols = [col for col in df.columns if col not in numeric_cols + date_cols]
for col in other_cols:
    if df[col].isnull().sum() > 0:
        # df[col] = df[col].fillna('未知')
        df[col] = df[col].bfill()
        print(f"已用'前值'填充'{col}'列的缺失值")
```

1.2 数据统计分析实例

```
# 2.2 处理重复值
duplicate_rows = df.duplicated().sum()
print(f"\n重复记录数量: {duplicate_rows}")
if duplicate_rows > 0:
    df.drop_duplicates(inplace=True)
    print(f"已删除所有重复记录, 剩余 {len(df)} 条记录")
# 2.3 数据类型转换
try:
    df['财务入账日期'] = pd.to_datetime(df['财务入账日期'])
    df['入账年份'] = df['财务入账日期'].dt.year
    df['入账月份'] = df['财务入账日期'].dt.month
    print("\n数据类型转换完成, 已添加'入账年份'和'入账月份'列")
except Exception as e:
    print(f"日期转换警告: {str(e)}")
```

1.2 数据统计分析实例

```
# 3. 统计分析
print("\n" + "=" * 50 + "\n统计分析结果:")
# 3.1 基本统计量
print("\n数值型数据基本统计量:")
print(df[['数量', '价值']].describe().round(2))
# 3.2 按资产分类统计
print("\n按资产分类统计:")
category_stats = df.groupby('资产分类').agg({
    '资产编号': 'count', # 设备数量
    '数量': 'sum', # 总数量
    '价值': ['sum', 'mean'] # 总价值和平均价值
}).round(2)
category_stats.columns = ['设备种类数', '总数量', '总价值', '平均价值']
print(category_stats)
```

1.2 数据统计分析实例

```
# 3.3 按使用部门统计
print("\n按使用部门统计:")
dept_stats = df.groupby('使用部门').agg({
    '资产编号': 'count', # 设备数量
    '价值': 'sum' # 总价值
}).round(2)
dept_stats.columns = ['设备数量', '总价值(元)']
# 计算各部门资产占比
dept_stats['价值占比(%)'] = (dept_stats['总价值(元)'] /
                             dept_stats['总价值(元)'].sum() * 100).round(2)

print(dept_stats)

# 3.4 按年份统计
if '入账年份' in df.columns:
    print("\n按入账年份统计:")
    year_stats = df.groupby('入账年份').agg({
        '资产编号': 'count', # 当年新增设备数
        '价值': 'sum' # 当年新增资产总价值
    }).round(2)
    year_stats.columns = ['新增设备数', '新增资产价值(元)']
    print(year_stats)
```

1.2 数据统计分析实例

```
# 4. 保存清洗后的数据
output_file = r"清洗后的资产设备表.xlsx"
df.to_excel(output_file, index=False, engine='openpyxl')
print(f"\n清洗后的数据集已保存至: {output_file}")
return df
if __name__ == "__main__":
    # 资产设备表路径
    input_file = r"资产设备表.xls"
    # 检查文件是否存在
    if not os.path.exists(input_file):
        print(f"错误: 文件 '{input_file}' 不存在!")
    else:
        # 执行分析
        analyze_assets(input_file)
```



/02

可视化实例

2.1 曲线拟合图实例

例6-3 历年数据变化曲线拟合

曲线拟合是一种数学方法，用于找到最适合一组数据点的曲线或函数。通过拟合曲线，可以预测未知数据点的值或分析数据的趋势。

曲线拟合的常见方法包括：

1. 最小二乘法 (Least Squares Method)：这是一种常见的曲线拟合方法，通过最小化观测数据点与拟合曲线之间的残差平方和来确定最佳拟合曲线。最小二乘法可以适用于线性和非线性拟合。
2. 多项式拟合 (Polynomial Fitting)：多项式拟合是一种将数据拟合到多项式函数的方法。通过选择适当的多项式阶数，可以在一定程度上逼近数据点。
3. 曲线拟合的其他方法：除了最小二乘法和多项式拟合，还有其它方法可用于曲线拟合，如样条插值、指数拟合、指数函数拟合等。

2.1 曲线拟合图实例

```
import matplotlib.pyplot as plt
import numpy as np

data_x = ['1995', '1996', '1997', '1998', '1999', '2000', '2001', '2002', '2003', '2004', '2005', '2006',
'2007', '2008', '2009']
data_y = [0.32, 0.32, 0.32, 0.32, 0.33, 0.33, 0.34, 0.37, 0.37, 0.37, 0.37, 0.39, 0.41, 0.42, 0.44]

# 将年份字符串转换为整数类型
data_x = np.array(data_x).astype(int)

# 使用matplotlib绘制折线图
plt.figure(figsize=(8, 6))
plt.scatter(data_x, data_y, marker='o', label='Data Points')
for a,b in zip(data_x, data_y):
    plt.text(a,b,b, ha='center', va='bottom')
```

2.1 曲线拟合图实例

绘制曲线

`poly = np.polyfit(data_x, data_y, deg=2)` #函数使用二次多项式 (deg=2) 对数据进行拟合, 返回拟合多项式的系数。

`y_value = np.polyval(poly, data_x)` #函数使用拟合得到的多项式系数和原始数据点的横坐标, 计算拟合曲线上对应的数据值。

`plt.plot(data_x, y_value, label='Fitted Curve')`

`plt.title('Data Trend with Quadratic Fit')`

`plt.xlabel('Year')`

`plt.ylabel('Value')`

`plt.legend()`

`plt.show()`

2.2 雷达图实例

雷达图 (Radar Chart)，又称蜘蛛图或星图，是一种以多维度数据为基础的可视化图表，通过从同一点出发的多条放射状坐标轴展示数据分布。它的核心价值在于直观呈现多个维度的数值对比，尤其适合分析单个对象的多方面特征或多个对象的综合能力差异。

核心特点

放射状结构：所有坐标轴从中心原点出发，呈辐射状排列，共享同一个起点。

多维度对比：可同时展示 3 个以上维度的数据（如产品的多项指标、用户的多维度评分）。

形状化解读：通过闭合多边形的形状和面积，快速判断数据的均衡性或优势 / 劣势维度。

适用场景

多维度能力评估：如员工技能评分（沟通、专业、效率等）、城市综合实力（教育、医疗、经济等）。

竞品分析：对比不同产品在性能、价格、外观等维度的表现。

用户画像：展示用户在消费、社交、娱乐等方面的偏好分布。

数据均衡性分析：判断某对象在各维度发展是否均衡（如学生各科成绩的全面性）。

绘制关键要素

维度（轴）：每个坐标轴代表一个数据维度，需明确标签（如“续航”“拍照”）。

刻度：各轴刻度范围需一致（除非特殊需求），否则会扭曲对比效果。

数据点：每个维度的具体数值在轴上的位置。

闭合多边形：连接同一对象的所有数据点，形成独特形状，便于直观对比。

2.2 雷达图实例

例6-4 王者荣耀参数选手重要指标统计分析

排名,icon,选手,比赛场次,胜场,胜率,场均KDA,场均击杀数,场均死亡数,场均助攻数,场均经济,分均经济,经济占比,场均伤害,分均伤害,伤害占比,伤害转化率,场均承伤,分均承伤,承伤占比,场均推塔数,推塔占比,参团率,场均比赛时长

1,<https://smobatv-pic.tga.qq.com/38e591cf1ed8efa83ea56f34bbb88af7.png>,花

月,1,1,100%,9(5 / 1 /

4),5,1,4,7554,755,21%,49291,4518.5,21.8%,103.9%,23929,1832.3,14.8%,3,37.5%,75%,0
0:10:55

2.2 雷达图实例

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
# 设置中文字体（解决中文显示问题）
plt.rcParams["font.family"] = ["SimHei"]
plt.rcParams["axes.unicode_minus"] = False # 解决负号显示异常的问题
fp = r"player-2023春季赛.csv"
df = pd.read_csv(fp)
df
data = df.sort_values("比赛场次", ascending=False)
data = data.iloc[:6]
data
filter_cols = ["选手", "经济占比", "伤害占比", "承伤占比", "推塔占比", "参团率"]
data = data.loc[:, filter_cols]
data
for col in filter_cols[1:]:
    data[col] = data[col].str.replace("%", "", regex=False)
    data[col] = data[col].astype("float")
data
```

2.2 雷达图实例

```
N = 5 # 雷达图属性个数
angles = np.linspace(0, 2 * np.pi, N, endpoint=False)
#将第一个角度值添加到角度数组的末尾，使得角度数组形成一个闭合的环形，用于雷达图的绘制。
angles = np.concatenate((angles, [angles[0]]))
fig = plt.figure(figsize=[10, 6])
for i in range(len(data)):
    values = data.iloc[i, 1:].tolist()
    #将第一个数据值再次添加到列表末尾，使数据形成一个闭合的环形，与角度数组对应。
    values.append(values[0])
    #生成一个字符串，表示当前子图在图形中的位置。例如，第一次循环时，position为231，第二次循环时为232，依此类推
    position = "23" + str(i + 1)
    #在图形中添加一个子图，使用生成的位置参数和polar=True表示创建一个极坐标子图，用于绘制雷达图。
    ax = fig.add_subplot(int(position), polar=True)
```

2.2 雷达图实例

#ax.plot绘制雷达图的线条，其中angles是角度数组，values是数据值数组，"o-"表示使用圆形标记和实线连接。

```
ax.plot(angles, values, "o-")
```

#填充雷达图的内部区域，设置透明度为0.4

```
ax.fill(angles, values, alpha=0.4)
```

#设置角度网格线的标签，将角度数组转换为度数，并使用数据集中的列名作为标签。

```
ax.set_thetagrids(angles[:-1] * 180 / np.pi,  
                  data.columns[1:].tolist())
```

```
ax.set_title(data.iloc[i, 0], color="b")
```

```
ax.set_ylim(0, 100)
```

#调整子图之间的垂直间距，这里设置为 0.5。

```
plt.subplots_adjust(hspace=0.5)
```



/03

中文分词实例

3 全模式分词实例

例6-5 全模式分词与文本歧义分析

全模式分词的特点是将句子中所有可能的词语都切分出来（存在冗余），适合用于文本歧义检测、词语组合挖掘等场景

3 全模式分词实例

```
import jieba
import re
import matplotlib.pyplot as plt
from collections import Counter
import pandas as pd
# -----
# 1. 配置与工具函数
# -----
# 加载停用词（过滤无意义词汇）
def load_stopwords(filepath="stopwords.txt"):
    with open(filepath, "r", encoding="utf-8") as f:
        return set([line.strip() for line in f.readlines()])

stopwords = load_stopwords()
# 文本预处理（保留标点，全模式可能需要标点辅助分析）
def preprocess_text(text):
    """轻量预处理：仅去除多余空格，保留标点和特殊符号"""
    text = re.sub(r"\s+", " ", text).strip()
    return text
```

3 全模式分词实例

```
# 读取文本（支持字符串或文件）
def get_text(input_data):
    if isinstance(input_data, str) and input_data.endswith(".txt"):
        with open(input_data, "r", encoding="utf-8") as f:
            return preprocess_text(f.read())
    return preprocess_text(input_data)
# -----
# 2. 全模式分词核心功能
# -----
def full_mode_cut(text):
    """全模式分词（尽可能多的切分）"""
    # cut_all=True开启全模式
    full_words = jieba.lcut(text, cut_all=True)
    # 过滤停用词和空字符串
    filtered_full = [word for word in full_words if word and word not in stopwords]
    return full_words, filtered_full
def accurate_mode_cut(text):
    """精确模式分词（对比用）"""
    acc_words = jieba.lcut(text, cut_all=False)
    filtered_acc = [word for word in acc_words if word and word not in stopwords]
    return acc_words, filtered_acc
```

3 全模式分词实例

```
# -----  
# 3. 分词结果对比分析  
# -----  
def compare_modes(full_words, acc_words):  
    """对比全模式与精确模式的分词差异"""  
    # 统计词数差异  
    print(f"\n【模式对比】")  
    print(f"全模式分词总数: {len(full_words)} (含重复和停用词)")  
    print(f"精确模式分词总数: {len(acc_words)} (含重复和停用词)")  
  
    # 提取全模式特有词语 (在精确模式中未出现的词)  
    acc_set = set(acc_words)  
    full_unique = [word for word in full_words if word not in acc_set]  
    print(f"全模式特有词语 (前10个): {full_unique[:10]}")
```

3 全模式分词实例

```
# 生成对比表格
compare_df = pd.DataFrame({
    "模式": ["全模式", "精确模式"],
    "分词数量": [len(full_words), len(acc_words)],
    "独特词语数": [len(set(full_unique)), 0],
    "平均词长": [round(sum(len(w) for w in full_words)/len(full_words), 2) if full_words
else 0,
                  round(sum(len(w) for w in acc_words)/len(acc_words), 2) if acc_words
else 0]
})
print("\n【分词模式对比表】 ")
print(compare_df)
return full_unique
```

3 全模式分词实例

```
# -----
# 4. 歧义分析（基于全模式）
# -----
def detect_ambiguity(text, full_filtered):
    """从全模式结果中检测潜在歧义短语"""
    # 歧义特征：同一位置有多种切分方式（通过全模式特有词与上下文组合判断）
    # 简单规则：全模式中出现的、长度≥2且在精确模式中未出现的词，可能对应歧义
    ambiguity_candidates = [word for word in full_filtered if len(word)>=2]
    # 统计高频歧义候选词
    ambiguity_counts = Counter(ambiguity_candidates).most_common(5)
    print("\n【潜在歧义短语（全模式特有高频词）】")
    for word, count in ambiguity_counts:
        # 查找词语在原文中的位置，展示上下文
        idx = text.find(word)
        if idx != -1:
            context = text[max(0, idx-5):min(len(text), idx+len(word)+5)]
            print(f"短语: {word}（出现{count}次），上下文: ...{context}...")
    return ambiguity_candidates
```

3 全模式分词实例

```
# -----
# 5. 全模式词频统计与可视化
# -----
def full_mode_word_freq(filtered_full, topk=15):
    """统计全模式分词的词频（含重复切分）"""
    freq = Counter(filtered_full).most_common(topk)
    print(f"\n【全模式高频词（前{topk}）】")
    for word, count in freq:
        print(f"{word}: {count}次")
    return freq

def plot_freq(freq, title):
    """可视化词频"""
    words, counts = zip(*freq) if freq else ([], [])
    plt.figure(figsize=(12, 6))
    plt.bar(words, counts, color='orange')
    plt.xticks(rotation=45, ha='right')
    plt.title(title)
    plt.tight_layout()
    plt.show()
```

3 全模式分词实例

```
# -----  
# 6. 全模式N-gram分析 (二元词)  
# -----  
def full_mode_bigram(full_filtered):  
    """从全模式结果中提取二元词 (词语组合) """  
    bigrams = [f"{full_filtered[i]}-{full_filtered[i+1]}"  
               for i in range(len(full_filtered)-1)]  
    bigram_freq = Counter(bigrams).most_common(10)  
    print("\n【全模式高频二元词 (前10) 】")  
    for bg, count in bigram_freq:  
        print(f"{bg}: {count}次")  
    return bigram_freq
```

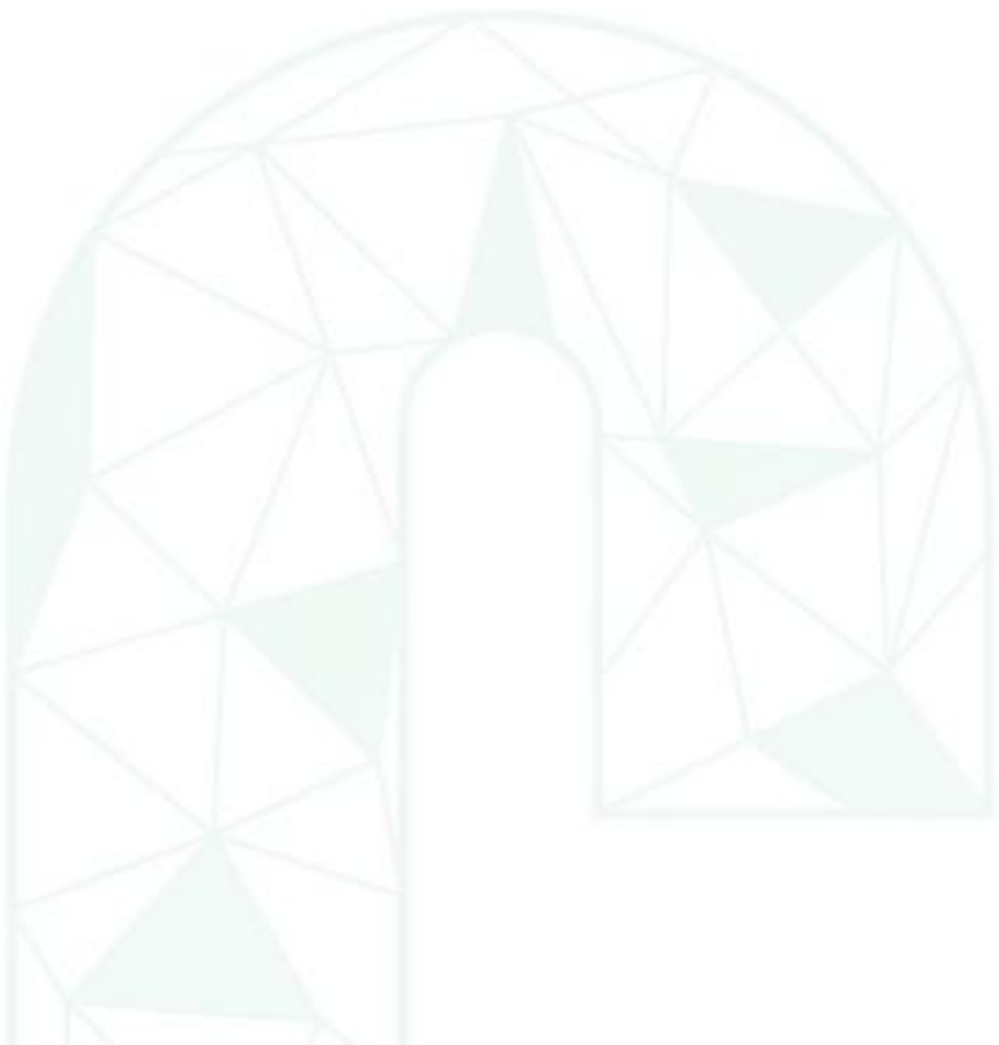

3 全模式分词实例

```
# -----  
# 主函数：串联所有功能  
# -----  
def full_mode_analyzer(input_text):  
    # 1. 读取并预处理文本  
    text = get_text(input_text)  
    print(f"【分析文本预览】：{text[:100]}...")  
    # 2. 两种模式分词  
    full_words, filtered_full = full_mode_cut(text)  
    acc_words, filtered_acc = accurate_mode_cut(text)  
    # 3. 模式对比  
    full_unique = compare_modes(full_words, acc_words)  
    # 4. 歧义分析  
    detect_ambiguity(text, filtered_full)  
    # 5. 全模式词频统计与可视化  
    full_freq = full_mode_word_freq(filtered_full)  
    plot_freq(full_freq, "全模式分词高频词分布")  
    # 6. 二元词分析  
    bigram_freq = full_mode_bigram(filtered_full)  
    plot_freq(bigram_freq, "全模式高频二元词分布")
```

3 全模式分词实例

```
return {
    "full_words": full_words,
    "acc_words": acc_words,
    "full_freq": full_freq,
    "bigrams": bigram_freq
}

# -----
# 运行示例
# -----
if __name__ == "__main__":
    # 示例文本（选择有歧义或多义词较多的文本，如新闻、散文）
    sample_text = """
    乒乓球拍卖完了。这件事的背后，是他和她的故事。
    他说：“我看见她那年，正是春暖花开的时候。”
    她则记得，那天的风很大，吹乱了她的头发，也吹乱了他的心。
    """
    # 运行分析
    result = full_mode_analyzer(sample_text)
```



/04

机器学习实例

4 机器学习实例

例6-6 用户信用风险评估系统

某金融公司现有客户数据集，包含 1000 条用户数据，字段如下：

输入特征（原始）：年龄（age）、工作年限（work_year）、教育程度（education：0=高中及以下，1=本科，2=硕士及以上）、负债比例（debt_ratio）、信用卡数量（credit_card_num）；

中间目标：月收入（monthly_income，连续值，需用线性回归预测）；

中间标签：是否逾期（is_overdue，二分类，需用逻辑回归预测）；

最终标签：信用等级（credit_rating，多分类：A/B/C/D，需用决策树预测）。

4 机器学习实例

例6-6 用户信用风险评估系统

公司需要构建信用风险评估系统，流程如下：

先用线性回归预测用户的 "月收入水平" (连续值) ；

再用逻辑回归基于月收入及其他特征预测用户 "是否有逾期记录" (二分类：0 = 无逾期，1 = 有逾期) ；

最后用决策树基于逾期记录、收入预测结果及原始特征，预测用户 "信用等级" (多分类：A/B/C/D) ；

验证三个模型的串联效果，并分析决策树的特征重要性。

4 机器学习实例

例6-6 用户信用风险评估系统

实现步骤

- 1.数据预处理（缺失值填充、特征标准化）；
- 2.训练线性回归模型预测月收入；
- 3.训练逻辑回归模型预测逾期记录（输入含线性回归的预测结果）；
- 4.训练决策树模型预测信用等级（输入含前两个模型的预测结果）；
- 5.模型评估与结果分析。

4 机器学习实例

例6-6 用户信用风险评估系统

线性回归：通过MSE评估月收入预测精度（值越小越好），体现连续值预测能力；

逻辑回归：通过准确率和分类报告（精确率、召回率）评估逾期预测效果，需关注对“有逾期”（少数类）的识别能力；

决策树：通过多分类准确率和特征重要性，分析哪些因素（如预测的收入、是否逾期）对信用等级影响最大；

串联逻辑：前一个模型的输出作为后一个模型的输入，模拟真实业务中“多步骤评估”的场景（如先估收入，再评逾期风险，最后定信用等级）。

4 机器学习实例

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer
from sklearn.linear_model import LinearRegression, LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import mean_squared_error, accuracy_score, classification_report
from sklearn.tree import plot_tree
import matplotlib.pyplot as plt
# ----- 1. 数据生成与预处理 -----
np.random.seed(42)
n = 1000
data = {
    "age": np.random.randint(18, 65, n),
    "work_year": np.random.randint(0, 40, n),
    "education": np.random.choice([0, 1, 2], n, p=[0.3, 0.5, 0.2]),
    "debt_ratio": np.random.uniform(0.1, 0.8, n),
    "credit_card_num": np.random.randint(1, 10, n)
}
```


4 机器学习实例

```
# 生成月收入（受年龄、工作年限、教育程度影响）
data["monthly_income"] = 3000 + data["work_year"]*800 + data["education"]*2000 +
data["age"]*50 + np.random.normal(0, 1000, n)
# 生成是否逾期（受月收入、负债比例影响）
overdue_prob = 1 / (1 + np.exp(-( -0.0005*data["monthly_income"] + 3*data["debt_ratio"] -
0.01*data["age"] )))
data["is_overdue"] = np.random.binomial(1, overdue_prob)
# 生成信用等级（A/B/C/D, 受逾期、收入、教育程度影响）
rating_score = (data["monthly_income"]/100) - (data["is_overdue"]*500) +
(data["education"]*200) - (data["debt_ratio"]*1000)
data["credit_rating"] = pd.cut(rating_score, bins=[-np.inf, 30, 60, 90, np.inf],
labels=["D", "C", "B", "A"])

df = pd.DataFrame(data)

# 划分数据集（保留20%作为测试集）
train_df, test_df = train_test_split(df, test_size=0.2, random_state=42)
```

4 机器学习实例

```
# 缺失值填充（模拟部分缺失）
train_df.loc[train_df.sample(frac=0.1).index, "monthly_income"] = np.nan # 10%收入缺失
imputer = SimpleImputer(strategy="median")
train_df["monthly_income"] = imputer.fit_transform(train_df[["monthly_income"]])
test_df["monthly_income"] = imputer.transform(test_df[["monthly_income"]])

# 特征标准化（线性回归和逻辑回归需要）
scaler = StandardScaler()
num_features = ["age", "work_year", "debt_ratio", "credit_card_num", "monthly_income"]
train_df[num_features] = scaler.fit_transform(train_df[num_features])
test_df[num_features] = scaler.transform(test_df[num_features])
```

4 机器学习实例

```
# ----- 2. 线性回归：预测月收入 -----  
# 特征：年龄、工作年限、教育程度、信用卡数量（假设这些特征可用于预测收入）  
lr_features = ["age", "work_year", "education", "credit_card_num"]  
X_lr_train = train_df[lr_features]  
y_lr_train = train_df["monthly_income"]  
X_lr_test = test_df[lr_features]  
y_lr_test = test_df["monthly_income"]  
  
# 训练模型  
lr = LinearRegression()  
lr.fit(X_lr_train, y_lr_train)  
  
# 预测与评估  
y_lr_pred = lr.predict(X_lr_test)  
print(f"线性回归月收入预测 MSE: {mean_squared_error(y_lr_test, y_lr_pred):.4f}")
```

4 机器学习实例

```
# ----- 3. 逻辑回归：预测是否逾期 -----
# 特征：线性回归预测的月收入 + 原始特征（负债比例、年龄）
logit_features = ["debt_ratio", "age"] # 原始特征
# 合并线性回归的预测结果作为新特征
train_df["pred_income"] = lr.predict(train_df[lr_features])
test_df["pred_income"] = y_lr_pred # 测试集用之前的预测结果

X_logit_train = train_df[logit_features + ["pred_income"]]
y_logit_train = train_df["is_overdue"]
X_logit_test = test_df[logit_features + ["pred_income"]]
y_logit_test = test_df["is_overdue"]

# 训练模型
logit = LogisticRegression(class_weight="balanced")
logit.fit(X_logit_train, y_logit_train)

# 预测与评估
y_logit_pred = logit.predict(X_logit_test)
print(f"逻辑回归逾期预测准确率：{accuracy_score(y_logit_test, y_logit_pred):.4f}")
print("逻辑回归分类报告：")
print(classification_report(y_logit_test, y_logit_pred))
```

4 机器学习实例

```
# ----- 4. 决策树：预测信用等级 -----
# 特征：前两个模型的预测结果 + 原始特征（教育程度、信用卡数量）
tree_features = ["education", "credit_card_num"] # 原始特征
# 合并逻辑回归的预测结果作为新特征
train_df["pred_overdue"] = logit.predict(train_df[logit_features + ["pred_income"]])
test_df["pred_overdue"] = y_logit_pred # 测试集用之前的预测结果

X_tree_train = train_df[tree_features + ["pred_income", "pred_overdue"]]
y_tree_train = train_df["credit_rating"]
X_tree_test = test_df[tree_features + ["pred_income", "pred_overdue"]]
y_tree_test = test_df["credit_rating"]

# 训练模型
tree = DecisionTreeClassifier(max_depth=5, random_state=42)
tree.fit(X_tree_train, y_tree_train)

# 预测与评估
y_tree_pred = tree.predict(X_tree_test)
print(f"决策树信用等级预测准确率：{accuracy_score(y_tree_test, y_tree_pred):.4f}")
print("决策树分类报告：")
print(classification_report(y_tree_test, y_tree_pred))
```

4 机器学习实例

```
# ----- 5. 结果分析 -----
# 特征重要性 (决策树)
print("\n决策树特征重要性: ")
for name, imp in zip(X_tree_train.columns, tree.feature_importances_):
    print(f"{name}: {imp:.4f}")

# 可视化决策树 (前两层)
plt.figure(figsize=(12, 8))
plot_tree(tree, feature_names=X_tree_train.columns, class_names=["D", "C", "B", "A"],
          filled=True, max_depth=2, fontsize=10)
plt.title("信用等级预测决策树 (前两层)")
plt.show()
```

The logo features a central green diamond with the text 'Canllay 嘉为数字咨询' and '数字化人才培养 先行者'. This is surrounded by a light green diamond and a dark green arrow-like shape pointing left. There are also several small dark green and grey diamonds scattered around the main elements.

Canllay 嘉为数字咨询

数字化人才培养
先行者