**《机器学习》实验课程大纲**

课程代码： G010401790

课程名称：机器学习

开课学期：2024-2025 2

学分/学时：3/48

课程类型：选修

适用专业/开课对象： 智能建造、本科三年级

先修课程： 无

开课单位： 智能建造学院

团队负责人：谢坤达 责任教授：谢坤达

执笔人： 谢坤达 核准院长：

### 实验一：KNN 鸢尾花分类

实验学时：2  
实验类型：验证性实验

一、实验目的

1. 理解KNN（K-Nearest Neighbors）算法的原理。
2. 学习使用KNN算法进行数据分类。
3. 熟悉数据预处理与特征选择的基本方法。
4. 掌握如何评估分类模型的性能（例如准确率、混淆矩阵等）。

二、实验环境

* 操作系统：Windows 10 / Linux
* 编程语言：Python
* 开发工具：Jupyter Notebook / Anaconda
* 使用库：Scikit-learn、Pandas、Matplotlib、Seaborn
* 数据集：鸢尾花数据集（Iris dataset）

三、实验内容

1. **KNN算法简介**
   * 理解KNN算法的基本概念：如何通过计算数据点之间的距离来进行分类。
   * 选择K值对模型性能的影响。
2. **数据加载与预处理**
   * 使用Scikit-learn加载鸢尾花数据集。
   * 数据探索：查看数据集的基本信息，理解每一列的意义。
   * 数据标准化：通过标准化（如Min-Max Scaling）将特征值调整到相同范围，避免某一特征对结果产生过大影响。
3. **KNN模型训练与预测**
   * 使用Scikit-learn的KNN分类器训练模型，并对测试集进行预测。
   * 选择不同的K值并观察模型在测试集上的表现。
4. **模型评估与调优**
   * 使用准确率、混淆矩阵等评估指标对模型进行评估。
   * 通过交叉验证选择最优的K值。

四、实验步骤

**Step 1: 数据加载与探索**

from sklearn.datasets import load\_irisimport pandas as pd

# 加载鸢尾花数据集

iris = load\_iris()

X = iris.data # 特征

y = iris.target # 标签

# 将数据转换为DataFrame，方便查看

df = pd.DataFrame(X, columns=iris.feature\_names)

df['species'] = iris.target\_names[y]

# 数据探索print(df.head())

**Step 2: 数据标准化**

from sklearn.preprocessing import StandardScaler

# 数据标准化

scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X)

# 查看标准化后的数据print(X\_scaled[:5])

**Step 3: KNN模型训练与预测**

from sklearn.model\_selection import train\_test\_splitfrom sklearn.neighbors import KNeighborsClassifier

# 划分训练集和测试集

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_scaled, y, test\_size=0.3, random\_state=42)

# 初始化KNN分类器，使用K=3

knn = KNeighborsClassifier(n\_neighbors=3)

# 训练模型

knn.fit(X\_train, y\_train)

# 对测试集进行预测

y\_pred = knn.predict(X\_test)

# 输出预测结果print("Predicted Labels: ", y\_pred)

**Step 4: 模型评估**

from sklearn.metrics import accuracy\_score, confusion\_matrix

# 准确率评估

accuracy = accuracy\_score(y\_test, y\_pred)print("Accuracy: ", accuracy)

# 混淆矩阵

conf\_matrix = confusion\_matrix(y\_test, y\_pred)print("Confusion Matrix: \n", conf\_matrix)

**Step 5: 调整K值**

# 尝试不同的K值并评估

k\_values = range(1, 11)

accuracies = []

for k in k\_values:

knn = KNeighborsClassifier(n\_neighbors=k)

knn.fit(X\_train, y\_train)

y\_pred = knn.predict(X\_test)

accuracies.append(accuracy\_score(y\_test, y\_pred))

# 绘制K值与准确率的关系import matplotlib.pyplot as plt

plt.plot(k\_values, accuracies, marker='o')

plt.title('KNN Classifier Accuracy vs K value')

plt.xlabel('K value')

plt.ylabel('Accuracy')

plt.show()

五、实验总结

1. 通过本实验，学生掌握了KNN分类算法的基本原理，并能够运用Scikit-learn实现KNN分类。
2. 学生学会了数据标准化的必要性以及如何调整K值以优化模型性能。
3. 通过混淆矩阵等评估指标，学生能够评估模型的分类效果，并分析分类结果。

### 实验二：决策树与随机森林

一、实验目的

1. 理解决策树（Decision Tree）与随机森林（Random Forest）算法的原理。
2. 学习使用决策树和随机森林进行分类任务。
3. 掌握模型评估与调优方法，理解过拟合与欠拟合的概念。
4. 比较决策树与随机森林在分类任务中的表现。

二、实验环境

* 操作系统：Windows 10 / Linux
* 编程语言：Python
* 开发工具：Jupyter Notebook / Anaconda
* 使用库：Scikit-learn、Pandas、Matplotlib、Seaborn
* 数据集：鸢尾花数据集（Iris dataset）

三、实验内容

1. **决策树算法原理**
   * 理解决策树的构建过程，包括信息增益、基尼指数等决策节点的选择方法。
2. **随机森林算法原理**
   * 理解随机森林的集成学习原理，包括Bootstrap抽样、决策树集成等。
3. **决策树模型训练与预测**
   * 使用决策树进行分类，观察模型的训练过程及其决策过程。
4. **随机森林模型训练与预测**
   * 使用随机森林模型进行分类，观察其集成学习的效果。
5. **模型评估与比较**
   * 使用准确率、混淆矩阵、特征重要性等评估指标，比较决策树与随机森林的表现。

四、实验步骤

**Step 1: 数据加载与探索**

from sklearn.datasets import load\_irisimport pandas as pd

# 加载鸢尾花数据集

iris = load\_iris()

X = iris.data # 特征

y = iris.target # 标签

# 将数据转换为DataFrame，方便查看

df = pd.DataFrame(X, columns=iris.feature\_names)

df['species'] = iris.target\_names[y]

# 数据探索print(df.head())

**Step 2: 划分训练集与测试集**

from sklearn.model\_selection import train\_test\_split

# 划分训练集和测试集

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

**Step 3: 决策树模型训练与预测**

from sklearn.tree import DecisionTreeClassifier

# 初始化决策树分类器

dt = DecisionTreeClassifier(random\_state=42)

# 训练模型

dt.fit(X\_train, y\_train)

# 对测试集进行预测

y\_pred\_dt = dt.predict(X\_test)

# 输出预测结果print("Decision Tree Predicted Labels: ", y\_pred\_dt)

**Step 4: 随机森林模型训练与预测**

from sklearn.ensemble import RandomForestClassifier

# 初始化随机森林分类器

rf = RandomForestClassifier(n\_estimators=100, random\_state=42)

# 训练模型

rf.fit(X\_train, y\_train)

# 对测试集进行预测

y\_pred\_rf = rf.predict(X\_test)

# 输出预测结果print("Random Forest Predicted Labels: ", y\_pred\_rf)

**Step 5: 模型评估**

from sklearn.metrics import accuracy\_score, confusion\_matrix

# 决策树评估

accuracy\_dt = accuracy\_score(y\_test, y\_pred\_dt)

conf\_matrix\_dt = confusion\_matrix(y\_test, y\_pred\_dt)print("Decision Tree Accuracy: ", accuracy\_dt)print("Decision Tree Confusion Matrix: \n", conf\_matrix\_dt)

# 随机森林评估

accuracy\_rf = accuracy\_score(y\_test, y\_pred\_rf)

conf\_matrix\_rf = confusion\_matrix(y\_test, y\_pred\_rf)print("Random Forest Accuracy: ", accuracy\_rf)print("Random Forest Confusion Matrix: \n", conf\_matrix\_rf)

**Step 6: 特征重要性分析**

# 随机森林特征重要性import matplotlib.pyplot as plt

feature\_importances = rf.feature\_importances\_

features = iris.feature\_names

# 绘制特征重要性

plt.barh(features, feature\_importances)

plt.title('Feature Importance (Random Forest)')

plt.xlabel('Importance')

plt.ylabel('Feature')

plt.show()

**Step 7: 过拟合与欠拟合分析**

# 调整决策树的最大深度，观察过拟合与欠拟合

dt\_2 = DecisionTreeClassifier(max\_depth=3, random\_state=42)

dt\_2.fit(X\_train, y\_train)

y\_pred\_dt\_2 = dt\_2.predict(X\_test)

# 评估

accuracy\_dt\_2 = accuracy\_score(y\_test, y\_pred\_dt\_2)print("Decision Tree with max depth=3 Accuracy: ", accuracy\_dt\_2)

五、实验总结

1. 学生通过本实验理解了决策树与随机森林算法的基本原理和实现方法，并能够使用Scikit-learn构建并训练决策树与随机森林分类模型。
2. 学生学会了使用不同的评估指标（如准确率、混淆矩阵）来评估模型性能，并理解了如何通过特征重要性分析提高模型的可解释性。
3. 本实验还帮助学生理解了过拟合和欠拟合的概念，并通过调整模型超参数（如决策树的最大深度）进行模型优化。

### 实验三：朴素贝叶斯垃圾短信分类

一、实验目的

1. 理解朴素贝叶斯分类算法的基本原理。
2. 学习如何使用朴素贝叶斯算法进行文本分类。
3. 掌握文本数据的预处理方法，如分词、去停用词等。
4. 使用朴素贝叶斯模型对垃圾短信进行分类，并评估模型性能。

二、实验环境

* 操作系统：Windows 10 / Linux
* 编程语言：Python
* 开发工具：Jupyter Notebook / Anaconda
* 使用库：Scikit-learn、Pandas、Matplotlib、Seaborn、nltk、re
* 数据集：短信数据集（SMS Spam Collection）

三、实验内容

1. **朴素贝叶斯算法简介**
   * 理解朴素贝叶斯算法的基本假设（特征条件独立）及其在文本分类中的应用。
2. **文本数据预处理**
   * 加载并清洗短信数据集。
   * 文本处理：分词、去除停用词、标点符号去除等。
   * 将文本数据转化为数值特征（如TF-IDF）。
3. **朴素贝叶斯模型训练与预测**
   * 使用朴素贝叶斯算法对垃圾短信进行分类。
4. **模型评估与调优**
   * 使用准确率、混淆矩阵、分类报告等评估模型性能。
   * 进行超参数调优，尝试不同的朴素贝叶斯变种（如多项式朴素贝叶斯）。

四、实验步骤

**Step 1: 数据加载与探索**

import pandas as pd

# 加载短信数据集

data = pd.read\_csv('spam.csv', encoding='latin-1')

# 查看数据集结构print(data.head())

# 清理数据：删除无用列

data = data[['v1', 'v2']]

data.columns = ['label', 'message']

# 查看数据集的一些统计信息print(data['label'].value\_counts())

**Step 2: 文本预处理**

import reimport nltkfrom nltk.corpus import stopwordsfrom sklearn.feature\_extraction.text import TfidfVectorizer

# 下载nltk数据

nltk.download('stopwords')

# 去除非字母字符，并转换为小写def preprocess\_text(text):

text = re.sub(r'[^a-zA-Z]', ' ', text)

text = text.lower()

return text

data['message'] = data['message'].apply(preprocess\_text)

# 去除停用词

stop\_words = set(stopwords.words('english'))

vectorizer = TfidfVectorizer(stop\_words=stop\_words)

# 转换文本数据为TF-IDF特征

X = vectorizer.fit\_transform(data['message'])

# 标签编码

y = data['label'].map({'ham': 0, 'spam': 1})

**Step 3: 朴素贝叶斯模型训练与预测**

from sklearn.model\_selection import train\_test\_splitfrom sklearn.naive\_bayes import MultinomialNBfrom sklearn.metrics import accuracy\_score, confusion\_matrix, classification\_report

# 划分训练集和测试集

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

# 初始化多项式朴素贝叶斯模型

nb = MultinomialNB()

# 训练模型

nb.fit(X\_train, y\_train)

# 对测试集进行预测

y\_pred = nb.predict(X\_test)

# 输出预测结果print("Predicted Labels: ", y\_pred)

**Step 4: 模型评估**

# 准确率评估

accuracy = accuracy\_score(y\_test, y\_pred)print("Accuracy: ", accuracy)

# 混淆矩阵

conf\_matrix = confusion\_matrix(y\_test, y\_pred)print("Confusion Matrix: \n", conf\_matrix)

# 分类报告

class\_report = classification\_report(y\_test, y\_pred)print("Classification Report: \n", class\_report)

**Step 5: 超参数调优**

# 调整alpha值（平滑参数）进行调优

nb\_tuned = MultinomialNB(alpha=0.5)

nb\_tuned.fit(X\_train, y\_train)

y\_pred\_tuned = nb\_tuned.predict(X\_test)

# 评估调优后的模型

accuracy\_tuned = accuracy\_score(y\_test, y\_pred\_tuned)print("Tuned Model Accuracy: ", accuracy\_tuned)

conf\_matrix\_tuned = confusion\_matrix(y\_test, y\_pred\_tuned)print("Tuned Model Confusion Matrix: \n", conf\_matrix\_tuned)

五、实验总结

1. 学生通过本实验掌握了朴素贝叶斯算法在文本分类中的应用，能够使用Scikit-learn构建并训练朴素贝叶斯模型进行垃圾短信分类。
2. 学生学会了文本数据的预处理方法，包括分词、去除停用词和标点符号，并将文本数据转化为数值特征。
3. 通过准确率、混淆矩阵和分类报告等评估指标，学生能够有效评估分类模型的表现，进一步理解如何优化模型参数来提高分类效果。

### 实验四：支持矢量机（SVM）分类

一、实验目的

1. 理解支持矢量机（SVM）算法的基本原理。
2. 学习如何使用SVM进行分类任务。
3. 理解线性与非线性SVM的区别，并掌握核函数的使用。
4. 使用SVM模型进行分类，并评估模型性能。

二、实验环境

* 操作系统：Windows 10 / Linux
* 编程语言：Python
* 开发工具：Jupyter Notebook / Anaconda
* 使用库：Scikit-learn、Pandas、Matplotlib、Seaborn
* 数据集：鸢尾花数据集（Iris dataset）

三、实验内容

1. **SVM算法简介**
   * 理解SVM的基本思想：最大化类别间的间隔，找到最优超平面。
   * 了解线性SVM与非线性SVM的区别，理解核函数的概念。
2. **SVM模型训练与预测**
   * 使用SVM进行线性分类。
   * 使用SVM进行非线性分类，并通过核函数转换数据到高维空间。
3. **模型评估与调优**
   * 使用准确率、混淆矩阵等评估指标来评估模型的分类效果。
   * 调整SVM的超参数（如C、核函数类型）来优化模型性能。

四、实验步骤

**Step 1: 数据加载与探索**

import pandas as pdfrom sklearn.datasets import load\_iris

# 加载鸢尾花数据集

iris = load\_iris()

X = iris.data # 特征

y = iris.target # 标签

# 将数据转换为DataFrame，方便查看

df = pd.DataFrame(X, columns=iris.feature\_names)

df['species'] = iris.target\_names[y]

# 查看数据集的一些统计信息print(df.head())

**Step 2: 划分训练集与测试集**

from sklearn.model\_selection import train\_test\_split

# 划分训练集和测试集

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

**Step 3: 训练线性SVM模型**

from sklearn.svm import SVCfrom sklearn.metrics import accuracy\_score, confusion\_matrix, classification\_report

# 初始化线性SVM分类器

svm\_linear = SVC(kernel='linear', random\_state=42)

# 训练模型

svm\_linear.fit(X\_train, y\_train)

# 对测试集进行预测

y\_pred\_linear = svm\_linear.predict(X\_test)

# 输出预测结果print("Linear SVM Predicted Labels: ", y\_pred\_linear)

**Step 4: 训练非线性SVM模型**

# 使用RBF核函数训练SVM

svm\_rbf = SVC(kernel='rbf', random\_state=42)

# 训练模型

svm\_rbf.fit(X\_train, y\_train)

# 对测试集进行预测

y\_pred\_rbf = svm\_rbf.predict(X\_test)

# 输出预测结果print("RBF SVM Predicted Labels: ", y\_pred\_rbf)

**Step 5: 模型评估**

# 线性SVM评估

accuracy\_linear = accuracy\_score(y\_test, y\_pred\_linear)print("Linear SVM Accuracy: ", accuracy\_linear)

conf\_matrix\_linear = confusion\_matrix(y\_test, y\_pred\_linear)print("Linear SVM Confusion Matrix: \n", conf\_matrix\_linear)

# RBF SVM评估

accuracy\_rbf = accuracy\_score(y\_test, y\_pred\_rbf)print("RBF SVM Accuracy: ", accuracy\_rbf)

conf\_matrix\_rbf = confusion\_matrix(y\_test, y\_pred\_rbf)print("RBF SVM Confusion Matrix: \n", conf\_matrix\_rbf)

**Step 6: SVM超参数调优**

# 调整C参数并评估模型

svm\_tuned = SVC(kernel='rbf', C=10, gamma='scale', random\_state=42)

svm\_tuned.fit(X\_train, y\_train)

y\_pred\_tuned = svm\_tuned.predict(X\_test)

# 评估调优后的模型

accuracy\_tuned = accuracy\_score(y\_test, y\_pred\_tuned)print("Tuned SVM Accuracy: ", accuracy\_tuned)

conf\_matrix\_tuned = confusion\_matrix(y\_test, y\_pred\_tuned)print("Tuned SVM Confusion Matrix: \n", conf\_matrix\_tuned)

**Step 7: 可视化支持矢量机的决策边界**

import numpy as npimport matplotlib.pyplot as plt

# 只选择前两个特征进行可视化

X\_train\_2d = X\_train[:, :2]

X\_test\_2d = X\_test[:, :2]

# 训练SVM模型

svm\_linear.fit(X\_train\_2d, y\_train)

# 绘制决策边界

h = .02 # 网格步长

x\_min, x\_max = X\_train\_2d[:, 0].min() - 1, X\_train\_2d[:, 0].max() + 1

y\_min, y\_max = X\_train\_2d[:, 1].min() - 1, X\_train\_2d[:, 1].max() + 1

xx, yy = np.meshgrid(np.arange(x\_min, x\_max, h),

np.arange(y\_min, y\_max, h))

Z = svm\_linear.predict(np.c\_[xx.ravel(), yy.ravel()])

Z = Z.reshape(xx.shape)

# 绘制图形

plt.contourf(xx, yy, Z, alpha=0.8)

plt.scatter(X\_train\_2d[:, 0], X\_train\_2d[:, 1], c=y\_train, edgecolors='k', marker='o', s=100, label='Train')

plt.scatter(X\_test\_2d[:, 0], X\_test\_2d[:, 1], c=y\_test, edgecolors='r', marker='x', s=100, label='Test')

plt.xlabel('Feature 1')

plt.ylabel('Feature 2')

plt.title('SVM Linear Kernel Decision Boundary')

plt.legend()

plt.show()

五、实验总结

1. 学生通过本实验掌握了支持矢量机（SVM）算法的基本原理，并学会了使用Scikit-learn实现SVM分类。
2. 学生理解了线性SVM与非线性SVM的区别，并通过核函数（如RBF）来处理非线性分类问题。
3. 学生学会了如何通过调整SVM的超参数（如C、核函数类型）进行模型调优，进一步提高模型的性能。
4. 通过可视化决策边界，学生能够更直观地理解SVM如何在特征空间中找到最优分割超平面。

实验五：XGBoost 房价预测

一、实验目的

1. 理解XGBoost（Extreme Gradient Boosting）算法的基本原理。
2. 学习如何使用XGBoost进行回归任务。
3. 理解梯度提升树（GBDT）模型及其优点。
4. 使用XGBoost对房价数据进行预测，并评估模型性能。

二、实验环境

* 操作系统：Windows 10 / Linux
* 编程语言：Python
* 开发工具：Jupyter Notebook / Anaconda
* 使用库：XGBoost、Pandas、Matplotlib、Seaborn、Scikit-learn
* 数据集：波士顿房价数据集（Boston Housing dataset）

三、实验内容

1. **XGBoost算法简介**
   * 理解梯度提升树（GBDT）模型的基本概念。
   * 理解XGBoost的优化算法（如正则化、剪枝等）。
2. **房价预测任务**
   * 使用XGBoost对波士顿房价数据集进行回归分析。
   * 理解特征选择、缺失值处理及数据预处理的重要性。
3. **模型训练与预测**
   * 使用XGBoost对训练集进行训练，并对测试集进行预测。
4. **模型评估与调优**
   * 使用均方误差（MSE）、均方根误差（RMSE）等评估指标评估模型的表现。
   * 使用网格搜索等方法调节XGBoost的超参数，优化模型性能。

四、实验步骤

**Step 1: 数据加载与探索**

import pandas as pdfrom sklearn.datasets import load\_boston

# 加载波士顿房价数据集

boston = load\_boston()

X = boston.data # 特征

y = boston.target # 标签

# 将数据转换为DataFrame，方便查看

df = pd.DataFrame(X, columns=boston.feature\_names)

df['MEDV'] = y # 添加房价数据作为标签

# 查看数据集的一些统计信息print(df.head())

**Step 2: 划分训练集与测试集**

from sklearn.model\_selection import train\_test\_split

# 划分训练集和测试集

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

**Step 3: 数据预处理**

from sklearn.preprocessing import StandardScaler

# 数据标准化

scaler = StandardScaler()

X\_train\_scaled = scaler.fit\_transform(X\_train)

X\_test\_scaled = scaler.transform(X\_test)

**Step 4: 使用XGBoost训练模型**

import xgboost as xgbfrom sklearn.metrics import mean\_squared\_error, mean\_absolute\_error

# 初始化XGBoost回归器

xg\_reg = xgb.XGBRegressor(objective='reg:squarederror', colsample\_bytree=0.3, learning\_rate=0.1,

max\_depth=5, alpha=10, n\_estimators=100, random\_state=42)

# 训练模型

xg\_reg.fit(X\_train\_scaled, y\_train)

# 对测试集进行预测

y\_pred = xg\_reg.predict(X\_test\_scaled)

# 输出预测结果print("Predicted Values: ", y\_pred)

**Step 5: 模型评估**

# 计算均方误差（MSE）和均方根误差（RMSE）

mse = mean\_squared\_error(y\_test, y\_pred)

rmse = mean\_squared\_error(y\_test, y\_pred, squared=False)

print("Mean Squared Error: ", mse)print("Root Mean Squared Error: ", rmse)

# 计算均值绝对误差（MAE）

mae = mean\_absolute\_error(y\_test, y\_pred)print("Mean Absolute Error: ", mae)

**Step 6: 模型调优（超参数优化）**

from sklearn.model\_selection import GridSearchCV

# 设置XGBoost的超参数

param\_grid = {

'max\_depth': [3, 4, 5, 6],

'learning\_rate': [0.01, 0.1, 0.2],

'n\_estimators': [50, 100, 200],

'colsample\_bytree': [0.3, 0.5, 0.7]

}

# 使用网格搜索进行超参数优化

grid\_search = GridSearchCV(estimator=xgb.XGBRegressor(objective='reg:squarederror', random\_state=42),

param\_grid=param\_grid, scoring='neg\_mean\_squared\_error', cv=3)

grid\_search.fit(X\_train\_scaled, y\_train)

# 输出最佳超参数print("Best Parameters: ", grid\_search.best\_params\_)

# 使用最佳参数训练模型

best\_model = grid\_search.best\_estimator\_

y\_pred\_best = best\_model.predict(X\_test\_scaled)

# 评估优化后的模型

mse\_best = mean\_squared\_error(y\_test, y\_pred\_best)

rmse\_best = mean\_squared\_error(y\_test, y\_pred\_best, squared=False)print("Optimized Model MSE: ", mse\_best)print("Optimized Model RMSE: ", rmse\_best)

**Step 7: 特征重要性分析**

import matplotlib.pyplot as plt

# 获取特征重要性

importance = best\_model.feature\_importances\_

# 绘制特征重要性图

plt.barh(boston.feature\_names, importance)

plt.xlabel('Feature Importance')

plt.title('XGBoost Feature Importance')

plt.show()

五、实验总结

1. 学生通过本实验理解了XGBoost算法的基本原理，并掌握了如何应用XGBoost进行房价预测任务。
2. 学生学会了如何对数据进行标准化处理，以便更好地训练XGBoost模型。
3. 学生掌握了如何评估回归模型的表现，包括均方误差（MSE）、均方根误差（RMSE）和均值绝对误差（MAE）。
4. 学生学会了使用网格搜索优化XGBoost的超参数，以提高模型的预测精度。
5. 学生通过特征重要性分析，了解哪些特征在房价预测中起到了重要作用。

# 实验六：PCA 降维与 SVD 图像压缩

一、实验目的

1. 理解主成分分析（PCA）算法的基本原理及其应用。
2. 学习如何使用PCA进行降维，减少数据的特征数量。
3. 理解奇异值分解（SVD）的基本概念及其在图像压缩中的应用。
4. 使用PCA和SVD分别进行图像压缩，并评估压缩效果。

二、实验环境

* 操作系统：Windows 10 / Linux
* 编程语言：Python
* 开发工具：Jupyter Notebook / Anaconda
* 使用库：NumPy、Pandas、Matplotlib、Scikit-learn、OpenCV
* 数据集：任意图像数据（例如，通过OpenCV读取的图像）

三、实验内容

1. **PCA降维与图像压缩**
   * 理解PCA的工作原理：如何通过主成分提取数据的主要特征，进行降维。
   * 使用PCA对图像进行降维和压缩，评估压缩后的图像质量。
2. **SVD图像压缩**
   * 理解SVD分解的基本概念：如何将矩阵分解为奇异值、左奇异矢量和右奇异矢量。
   * 使用SVD进行图像压缩，评估压缩后的图像质量。
3. **压缩效果评估**
   * 使用均方误差（MSE）和结构相似性指数（SSIM）等指标评估图像压缩效果。

四、实验步骤

**Step 1: 导入库和加载图像**

import numpy as npimport matplotlib.pyplot as pltimport cv2from sklearn.decomposition import PCAfrom sklearn.metrics import mean\_squared\_errorfrom skimage.metrics import structural\_similarity as ssim

# 读取图像，转换为灰度图像

image = cv2.imread('your\_image.jpg', cv2.IMREAD\_GRAYSCALE)

# 显示原始图像

plt.imshow(image, cmap='gray')

plt.title('Original Image')

plt.axis('off')

plt.show()

**Step 2: 使用PCA进行降维和图像压缩**

# 将图像矩阵转化为二维数组，每一行表示一个像素点

image\_reshaped = image.reshape(-1, 1)

# 使用PCA进行降维，保留k个主成分

k = 50 # 设置主成分的数量

pca = PCA(n\_components=k)

image\_pca = pca.fit\_transform(image\_reshaped)

# 使用PCA逆变换恢复图像

image\_pca\_reconstructed = pca.inverse\_transform(image\_pca)

image\_pca\_reconstructed = image\_pca\_reconstructed.reshape(image.shape)

# 显示PCA压缩后的图像

plt.imshow(image\_pca\_reconstructed, cmap='gray')

plt.title(f'PCA Compressed Image (k={k})')

plt.axis('off')

plt.show()

# 计算PCA压缩的MSE和SSIM

mse\_pca = mean\_squared\_error(image, image\_pca\_reconstructed)

ssim\_pca = ssim(image, image\_pca\_reconstructed)print(f'PCA Compression MSE: {mse\_pca}')print(f'PCA Compression SSIM: {ssim\_pca}')

**Step 3: 使用SVD进行图像压缩**

# 对图像进行奇异值分解（SVD）

U, S, Vt = np.linalg.svd(image, full\_matrices=False)

# 保留k个奇异值进行压缩

k\_svd = 50

U\_k = U[:, :k\_svd]

S\_k = np.diag(S[:k\_svd])

Vt\_k = Vt[:k\_svd, :]

# 重构压缩后的图像

image\_svd\_reconstructed = np.dot(U\_k, np.dot(S\_k, Vt\_k))

# 显示SVD压缩后的图像

plt.imshow(image\_svd\_reconstructed, cmap='gray')

plt.title(f'SVD Compressed Image (k={k\_svd})')

plt.axis('off')

plt.show()

# 计算SVD压缩的MSE和SSIM

mse\_svd = mean\_squared\_error(image, image\_svd\_reconstructed)

ssim\_svd = ssim(image, image\_svd\_reconstructed)print(f'SVD Compression MSE: {mse\_svd}')print(f'SVD Compression SSIM: {ssim\_svd}')

**Step 4: 比较PCA和SVD的压缩效果**

# 可视化原始图像和压缩后的图像（PCA与SVD）

fig, axes = plt.subplots(1, 3, figsize=(15, 5))

axes[0].imshow(image, cmap='gray')

axes[0].set\_title('Original Image')

axes[0].axis('off')

axes[1].imshow(image\_pca\_reconstructed, cmap='gray')

axes[1].set\_title(f'PCA (k={k})')

axes[1].axis('off')

axes[2].imshow(image\_svd\_reconstructed, cmap='gray')

axes[2].set\_title(f'SVD (k={k\_svd})')

axes[2].axis('off')

plt.show()

# 比较不同压缩方法的MSE和SSIMprint(f"Comparison of PCA and SVD Compression:")print(f"PCA Compression MSE: {mse\_pca}, SSIM: {ssim\_pca}")print(f"SVD Compression MSE: {mse\_svd}, SSIM: {ssim\_svd}")

**Step 5: 调整降维参数进行进一步优化**

# 尝试不同的k值进行压缩，并评估压缩效果

k\_values = [20, 50, 100]for k\_val in k\_values:

pca = PCA(n\_components=k\_val)

image\_pca = pca.fit\_transform(image\_reshaped)

image\_pca\_reconstructed = pca.inverse\_transform(image\_pca)

image\_pca\_reconstructed = image\_pca\_reconstructed.reshape(image.shape)

mse\_pca = mean\_squared\_error(image, image\_pca\_reconstructed)

ssim\_pca = ssim(image, image\_pca\_reconstructed)

print(f"PCA Compression (k={k\_val}) MSE: {mse\_pca}, SSIM: {ssim\_pca}")

# 使用SVD

U, S, Vt = np.linalg.svd(image, full\_matrices=False)

U\_k = U[:, :k\_val]

S\_k = np.diag(S[:k\_val])

Vt\_k = Vt[:k\_val, :]

image\_svd\_reconstructed = np.dot(U\_k, np.dot(S\_k, Vt\_k))

mse\_svd = mean\_squared\_error(image, image\_svd\_reconstructed)

ssim\_svd = ssim(image, image\_svd\_reconstructed)

print(f"SVD Compression (k={k\_val}) MSE: {mse\_svd}, SSIM: {ssim\_svd}")

五、实验总结

1. 学生通过本实验理解了PCA降维的原理及其在图像压缩中的应用，能够通过保留主成分来减少图像的维度并进行压缩。
2. 学生了解了奇异值分解（SVD）如何应用于图像压缩，能够通过保留少量奇异值来实现压缩。
3. 学生学会了使用均方误差（MSE）和结构相似性指数（SSIM）等指标评估图像压缩质量，能够比较不同降维方法的效果。
4. 通过调整降维参数（如PCA中的主成分个数k或SVD中的奇异值数量），学生能够优化图像压缩效果。

实验七：K-Means 聚类

一、实验目的

1. 理解K-Means聚类算法的基本原理。
2. 学习如何应用K-Means算法进行无监督学习的聚类分析。
3. 学会如何选择合适的聚类数目（k值）。
4. 通过聚类结果评估K-Means模型的性能。

二、实验环境

* 操作系统：Windows 10 / Linux
* 编程语言：Python
* 开发工具：Jupyter Notebook / Anaconda
* 使用库：NumPy、Pandas、Matplotlib、Scikit-learn
* 数据集：经典的鸢尾花数据集（Iris Dataset）或其他聚类数据集

三、实验内容

1. **K-Means算法原理**
   * 理解K-Means算法的工作原理：通过迭代过程将数据点划分为K个簇，并计算每个簇的中心点。
   * 学习K-Means算法中的初始化、更新簇中心和分配簇的步骤。
2. **聚类数目选择**
   * 学习如何使用肘部法则（Elbow Method）选择最佳的聚类数目（k值）。
   * 通过可视化不同k值的聚类结果，帮助选择最优k值。
3. **应用K-Means进行数据聚类**
   * 使用K-Means算法对数据进行聚类，进行结果分析和可视化。
4. **聚类结果评估**
   * 使用轮廓系数（Silhouette Score）评估聚类结果的质量。

四、实验步骤

**Step 1: 导入库和加载数据集**

import numpy as npimport matplotlib.pyplot as pltfrom sklearn.datasets import load\_irisfrom sklearn.cluster import KMeansfrom sklearn.metrics import silhouette\_score

# 加载鸢尾花数据集

iris = load\_iris()

X = iris.data # 特征数据

# 可视化原始数据的前两列

plt.scatter(X[:, 0], X[:, 1], c='blue', s=50)

plt.title('Original Iris Dataset (First Two Features)')

plt.xlabel('Feature 1')

plt.ylabel('Feature 2')

plt.show()

**Step 2: 使用肘部法则选择聚类数目k**

# 使用肘部法则选择最优的k值

inertia = []

k\_range = range(1, 11)

for k in k\_range:

kmeans = KMeans(n\_clusters=k, random\_state=42)

kmeans.fit(X)

inertia.append(kmeans.inertia\_) # 计算每个k值的总内部误差平方和

# 绘制肘部法则图

plt.plot(k\_range, inertia, marker='o')

plt.title('Elbow Method for Optimal k')

plt.xlabel('Number of Clusters (k)')

plt.ylabel('Inertia')

plt.show()

**Step 3: 使用K-Means进行聚类**

# 根据肘部法则选择最佳k值（例如k=3）

k\_optimal = 3

kmeans = KMeans(n\_clusters=k\_optimal, random\_state=42)

y\_kmeans = kmeans.fit\_predict(X)

# 可视化聚类结果

plt.scatter(X[:, 0], X[:, 1], c=y\_kmeans, s=50, cmap='viridis')

centers = kmeans.cluster\_centers\_ # 聚类中心

plt.scatter(centers[:, 0], centers[:, 1], c='red', s=200, marker='X') # 聚类中心可视化

plt.title(f'K-Means Clustering (k={k\_optimal})')

plt.xlabel('Feature 1')

plt.ylabel('Feature 2')

plt.show()

**Step 4: 聚类结果评估**

# 计算轮廓系数（Silhouette Score）评估聚类质量

sil\_score = silhouette\_score(X, y\_kmeans)print(f'Silhouette Score for k={k\_optimal}: {sil\_score}')

**Step 5: 使用不同的k值评估聚类质量**

# 尝试不同的k值进行聚类，并计算轮廓系数

k\_values = [2, 3, 4, 5]for k in k\_values:

kmeans = KMeans(n\_clusters=k, random\_state=42)

y\_kmeans = kmeans.fit\_predict(X)

sil\_score = silhouette\_score(X, y\_kmeans)

print(f'For k={k}, Silhouette Score: {sil\_score}')

# 可视化聚类结果

plt.scatter(X[:, 0], X[:, 1], c=y\_kmeans, s=50, cmap='viridis')

centers = kmeans.cluster\_centers\_

plt.scatter(centers[:, 0], centers[:, 1], c='red', s=200, marker='X')

plt.title(f'K-Means Clustering (k={k})')

plt.xlabel('Feature 1')

plt.ylabel('Feature 2')

plt.show()

五、实验总结

1. 通过本实验，学生掌握了K-Means聚类算法的基本原理及其应用，能够根据数据特点选择适当的聚类数目。
2. 学生学会了如何使用肘部法则（Elbow Method）选择最佳的聚类数目，避免了人为选择k值的主观性。
3. 学生能够使用轮廓系数（Silhouette Score）评估聚类质量，了解不同k值对聚类结果的影响。
4. 通过实验，学生理解了聚类算法在无监督学习中的重要性，掌握了如何使用K-Means进行实际的数据分析。

实验八：CNN 手写数字识别

一、实验目的

1. 理解卷积神经网络（CNN）的基本结构和工作原理。
2. 学会使用CNN模型进行图像分类，特别是用于手写数字识别任务。
3. 学习如何使用Keras/TensorFlow框架构建和训练CNN模型。
4. 掌握数据预处理、模型训练和评估的基本流程。

二、实验环境

* 操作系统：Windows 10 / Linux
* 编程语言：Python
* 开发工具：Jupyter Notebook / Anaconda
* 使用库：NumPy、Matplotlib、TensorFlow、Keras
* 数据集：MNIST手写数字数据集

三、实验内容

1. **卷积神经网络（CNN）原理**
   * 理解卷积层、池化层、全连接层等基本结构。
   * 了解如何通过卷积操作提取图像的局部特征，并通过池化降低特征维度。
2. **使用Keras/TensorFlow构建CNN模型**
   * 学习如何搭建CNN模型，包括卷积层、池化层和全连接层的堆叠。
   * 使用ReLU激活函数、softmax输出层等技术进行分类。
3. **数据加载与预处理**
   * 学会使用MNIST数据集，并对数据进行归一化处理。
   * 了解如何分割训练集和测试集，进行批量训练。
4. **训练与评估模型**
   * 训练CNN模型并观察训练过程中的准确率和损失值变化。
   * 评估模型在测试集上的性能，分析准确率。

四、实验步骤

**Step 1: 导入库和加载MNIST数据集**

import numpy as npimport matplotlib.pyplot as pltimport tensorflow as tffrom tensorflow.keras.datasets import mnistfrom tensorflow.keras.utils import to\_categorical

# 加载MNIST数据集

(x\_train, y\_train), (x\_test, y\_test) = mnist.load\_data()

# 显示一张手写数字图片

plt.imshow(x\_train[0], cmap='gray')

plt.title(f'Label: {y\_train[0]}')

plt.axis('off')

plt.show()

**Step 2: 数据预处理**

* 归一化处理：将像素值缩放到[0, 1]范围内。
* 类别标签转换：将标签转换为one-hot编码。

# 归一化处理

x\_train = x\_train.astype('float32') / 255.0

x\_test = x\_test.astype('float32') / 255.0

# 重新调整数据形状，MNIST数据集是28x28的灰度图像，需要扩展为28x28x1的格式

x\_train = np.expand\_dims(x\_train, axis=-1)

x\_test = np.expand\_dims(x\_test, axis=-1)

# 类别标签转换为one-hot编码

y\_train = to\_categorical(y\_train, 10)

y\_test = to\_categorical(y\_test, 10)

**Step 3: 构建CNN模型**

from tensorflow.keras.models import Sequentialfrom tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout

# 构建CNN模型

model = Sequential()

# 卷积层1 + 池化层1

model.add(Conv2D(32, kernel\_size=(3, 3), activation='relu', input\_shape=(28, 28, 1)))

model.add(MaxPooling2D(pool\_size=(2, 2)))

# 卷积层2 + 池化层2

model.add(Conv2D(64, kernel\_size=(3, 3), activation='relu'))

model.add(MaxPooling2D(pool\_size=(2, 2)))

# 展平层

model.add(Flatten())

# 全连接层

model.add(Dense(128, activation='relu'))

model.add(Dropout(0.5)) # Dropout层，用于减少过拟合

# 输出层

model.add(Dense(10, activation='softmax'))

# 模型概况

model.summary()

**Step 4: 编译和训练模型**

# 编译模型

model.compile(optimizer='adam', loss='categorical\_crossentropy', metrics=['accuracy'])

# 训练模型

history = model.fit(x\_train, y\_train, epochs=10, batch\_size=128, validation\_data=(x\_test, y\_test))

**Step 5: 可视化训练过程**

# 可视化训练过程中的准确率和损失值变化

plt.plot(history.history['accuracy'], label='Training Accuracy')

plt.plot(history.history['val\_accuracy'], label='Validation Accuracy')

plt.title('Model Accuracy')

plt.xlabel('Epochs')

plt.ylabel('Accuracy')

plt.legend()

plt.show()

plt.plot(history.history['loss'], label='Training Loss')

plt.plot(history.history['val\_loss'], label='Validation Loss')

plt.title('Model Loss')

plt.xlabel('Epochs')

plt.ylabel('Loss')

plt.legend()

plt.show()

**Step 6: 评估模型性能**

# 在测试集上评估模型性能

test\_loss, test\_acc = model.evaluate(x\_test, y\_test)print(f'Test Accuracy: {test\_acc}')print(f'Test Loss: {test\_loss}')

# 使用模型对测试集进行预测

predictions = model.predict(x\_test)

# 显示测试集中的一张图像及其预测结果

plt.imshow(x\_test[0].reshape(28, 28), cmap='gray')

plt.title(f'Predicted Label: {np.argmax(predictions[0])}')

plt.axis('off')

plt.show()

**Step 7: 保存和加载模型**

# 保存训练好的模型

model.save('cnn\_mnist\_model.h5')

# 加载已保存的模型from tensorflow.keras.models import load\_model

loaded\_model = load\_model('cnn\_mnist\_model.h5')

五、实验总结

1. 学生通过本实验了解了卷积神经网络（CNN）的基本结构及其在图像分类中的应用。
2. 学生学会了如何使用Keras/TensorFlow框架构建CNN模型，并完成数据预处理、模型训练、评估及可视化等步骤。
3. 学生掌握了如何使用训练好的CNN模型进行手写数字识别，并且通过训练过程中的准确率和损失值变化分析模型性能。
4. 本实验进一步帮助学生理解如何使用深度学习框架解决实际问题，并通过模型优化和评估来提高分类效果。