**智能数据挖掘上机作业报告**

人工智能（图灵）

汤栋文 22009200601

2025-05-18

**目录**

[**1.** **Apriori算法** 2](#_Toc198483823)

[**2.** **线性回归模型** 3](#_Toc198483824)

[**3.** **二次回归模型** 4](#_Toc198483825)

[**4.** **K-最近邻** 4](#_Toc198483826)

[**5.** **SVM算法** 6](#_Toc198483827)

[**6.** **决策树算法** 7](#_Toc198483828)

[**7.** **朴素贝叶斯算法** 8](#_Toc198483829)

[**8.** **K-means算法** 9](#_Toc198483830)

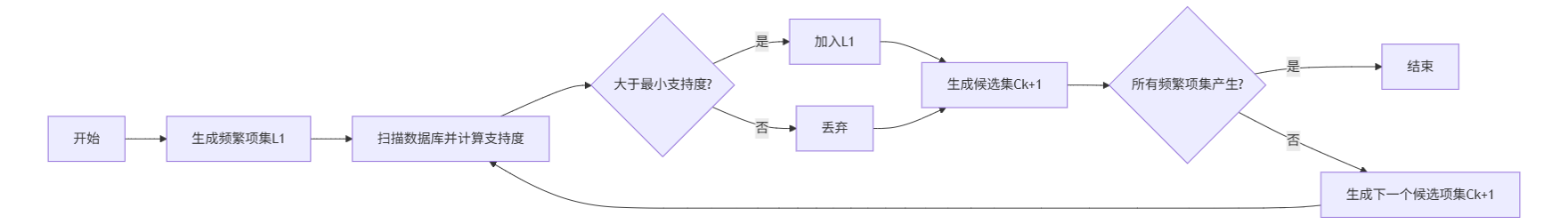
[**9.** **DBSCAN算法** 10](#_Toc198483831)

[**10.** **AdaBoost算法** 11](#_Toc198483832)

## **Apriori算法**

**题目：**编程实现Apriori算法（关联规则挖掘）

**流程图：**



**代码：**

from itertools import combinations

def load\_data():

    return [["面包", "牛奶", "啤酒"],

            ["啤酒", "泡面", "尿布"],

            ["矿泉水", "泡面", "尿布"],

            ["啤酒", "尿布"],

            ["面包", "牛奶", "啤酒", "尿布"],

            ["面包", "牛奶", "啤酒"],

            ["啤酒", "牛奶", "尿布"]]

def get\_frequent\_itemsets(data, min\_support):

    item\_count = {}

    for transaction in data:

        for item in transaction:

            if item not in item\_count:

                item\_count[item] = 0

            item\_count[item] += 1

    num\_transactions = len(data)

    frequent\_itemsets = {frozenset([item]): count / num\_transactions for item, count in item\_count.items() if count / num\_transactions >= min\_support}

    k = 2

    while True:

        candidate\_itemsets = generate\_candidate\_itemsets(frequent\_itemsets.keys(), k)

        candidate\_count = {candidate: 0 for candidate in candidate\_itemsets}

        for transaction in data:

            transaction\_set = set(transaction)

            for candidate in candidate\_itemsets:

                if candidate.issubset(transaction\_set):

                    candidate\_count[candidate] += 1

        new\_frequent\_itemsets = {itemset: count / num\_transactions for itemset, count in candidate\_count.items() if count / num\_transactions >= min\_support}

        if not new\_frequent\_itemsets:

            break

        frequent\_itemsets.update(new\_frequent\_itemsets)

        k += 1

    return frequent\_itemsets

def generate\_candidate\_itemsets(itemsets, k):

    candidates = set()

    for itemset1 in itemsets:

        for itemset2 in itemsets:

            union = itemset1 | itemset2

            if len(union) == k and all(frozenset(subset) in itemsets for subset in combinations(union, k-1)):

                candidates.add(union)

    return candidates

def generate\_association\_rules(frequent\_itemsets, min\_confidence):

    rules = []

    for itemset in frequent\_itemsets:

        support\_itemset = frequent\_itemsets[itemset]

        for i in range(1, len(itemset)):

            for antecedent in combinations(itemset, i):

                antecedent\_set = frozenset(antecedent)

                consequent\_set = itemset - antecedent\_set

                confidence = support\_itemset / frequent\_itemsets[antecedent\_set]

                if confidence >= min\_confidence:

                    rules.append((antecedent\_set, consequent\_set, confidence))

    return rules

if \_\_name\_\_ == "\_\_main\_\_":

    data = load\_data()

    min\_support = 0.3

    min\_confidence = 0.8

    frequent\_itemsets = get\_frequent\_itemsets(data, min\_support)

    print("Frequent Itemsets:")

    for itemset, support in frequent\_itemsets.items():

        print(f"{list(itemset)}: {support}")

    association\_rules = generate\_association\_rules(frequent\_itemsets, min\_confidence)

    print("\nAssociation Rules:")

    for antecedent, consequent, confidence in association\_rules:

        print(f"{list(antecedent)} -> {list(consequent)}: Confidence = {confidence:.2f}")

**运行结果：**

Frequent Itemsets:

['面包']: 0.42857142857142855

['牛奶']: 0.5714285714285714

['啤酒']: 0.8571428571428571

['尿布']: 0.7142857142857143

['啤酒', '尿布']: 0.5714285714285714

['啤酒', '牛奶']: 0.5714285714285714

['面包', '啤酒']: 0.42857142857142855

['面包', '牛奶']: 0.42857142857142855

['面包', '啤酒', '牛奶']: 0.42857142857142855

Association Rules:

['牛奶'] -> ['啤酒']: Confidence = 1.00

['面包'] -> ['啤酒']: Confidence = 1.00

['面包'] -> ['牛奶']: Confidence = 1.00

['面包'] -> ['啤酒', '牛奶']: Confidence = 1.00

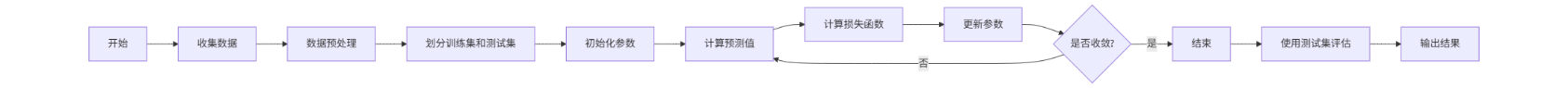
['面包', '啤酒'] -> ['牛奶']: Confidence = 1.00

['面包', '牛奶'] -> ['啤酒']: Confidence = 1.00

## **线性回归模型**

**题目：**基于线性回归模型拟合一个班学生的学习成绩，建立预测模型。数据可由自己建立100个学生的学习成绩。经验方程是线性的，形如。按上面的分析，误差函数为 。

**流程图：**

****

**代码：**

import numpy as np

import matplotlib.pyplot as plt

plt.rcParams["font.size"] = 16

np.random.seed(42)

num\_students = 100

study\_hours = np.random.uniform(0, 24, num\_students)

grades = 5 \* study\_hours + 50 + np.random.normal(0, 5, num\_students)

X = np.vstack([study\_hours, np.ones(len(study\_hours))]).T

theta = np.linalg.lstsq(X, grades, rcond=None)[0]

a, b = theta

print(f"Model: y = {a:.2f}x + {b:.2f}")

predicted\_grades = a \* study\_hours + b

plt.scatter(study\_hours, grades, color='blue', label='data')

plt.plot(study\_hours, predicted\_grades, color='red', label='model')

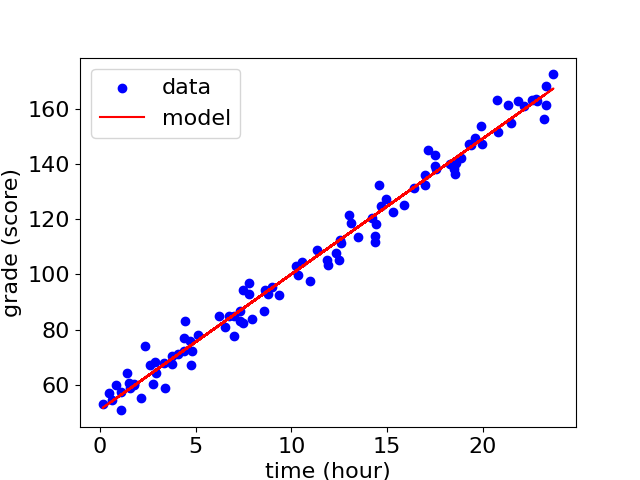
plt.xlabel('time (hour)')

plt.ylabel('grade (score)')

plt.legend()

plt.show()

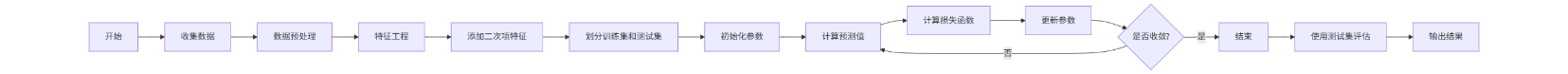
**运行结果：**Model:

****

## **二次回归模型**

**题目：**基于二次回归模型拟合人体感染某种病毒后，身体内病毒载量，建立预测模型。数据可由自己建立，感染病毒后100个时刻，病人体内病毒载量。（经验方程是二次的，形如。病毒载量在感染初期较低，但随后病毒会大量复制，载量升高，随后免疫系统开始清除病毒，载量降低，最终完全杀灭所有病毒。误差函数为：

**流程图：**

****

**代码：**

import numpy as np

import matplotlib.pyplot as plt

time = np.array([2, 4, 6, 8, 10, 12, 14, 16, 18, 20])

viral\_load = np.array([70, 500, 3200, 8000, 15000, 20000, 15000, 8000, 3200, 500])

X = np.vstack([time\*\*2, time, np.ones(len(time))]).T

y = viral\_load

theta = np.linalg.lstsq(X, y, rcond=None)[0]

a, b, c = theta

print(f"Model: y = {a:.2f}x^2 + {b:.2f}x + {c:.2f}")

time\_pred = np.linspace(0, 30, 60)

viral\_load\_pred = a \* time\_pred\*\*2 + b \* time\_pred + c

plt.scatter(time, viral\_load, color='blue', label='data')

plt.plot(time\_pred, viral\_load\_pred, color='red', label='model')

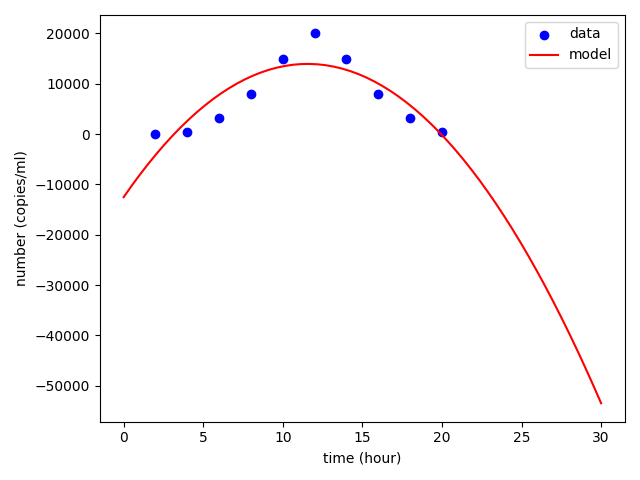
plt.xlabel('time (hour)')

plt.ylabel('number (copies/ml)')

plt.legend()

plt.show()

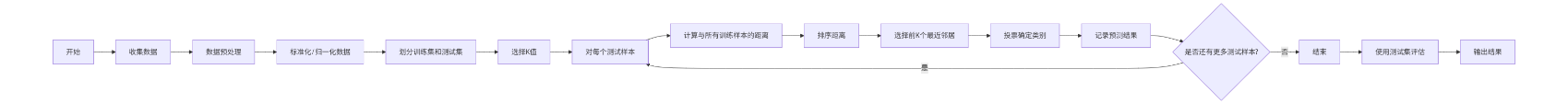
**运行结果：**Model:

****

## **K-最近邻**

**题目：**基于K-最近邻的标准数据集分类。（对UCI-Iris数据集进行K-最近邻分类）

**流程图：**



**代码：**

from sklearn.datasets import load\_iris

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

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import accuracy\_score, classification\_report

iris = load\_iris()

X = iris.data

y = iris.target

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=43)

k = 5

knn = KNeighborsClassifier(n\_neighbors=k)

knn.fit(X\_train, y\_train)

y\_pred = knn.predict(X\_test)

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

print("\nClassification Report:\n", classification\_report(y\_test, y\_pred))

import matplotlib.pyplot as plt

plt.figure(figsize=(8, 6))

for i, color in zip(range(len(iris.target\_names)), ['red', 'green', 'blue']):

    plt.scatter(X\_test[y\_test == i, 0], X\_test[y\_test == i, 1], color=color, label=f"True {iris.target\_names[i]}")

    plt.scatter(X\_test[y\_pred == i, 0], X\_test[y\_pred == i, 1], color=color, marker='x', label=f"Pred {iris.target\_names[i]}")

neighbors = knn.kneighbors(X\_test, return\_distance=False)

for idx, neighbor\_indices in enumerate(neighbors):

    for neighbor\_idx in neighbor\_indices:

        plt.scatter(X\_train[neighbor\_idx, 0], X\_train[neighbor\_idx, 1],

                    color=plt.cm.Paired(y\_test[idx] / len(iris.target\_names)),

                    s=20, alpha=0.6)

        plt.plot([X\_test[idx, 0], X\_train[neighbor\_idx, 0]],

                 [X\_test[idx, 1], X\_train[neighbor\_idx, 1]],

                 color=plt.cm.Paired(y\_test[idx] / len(iris.target\_names)),

                 alpha=0.6, linestyle='--')

plt.title("KNN Classification Results with Neighbors")

plt.xlabel("Feature 1")

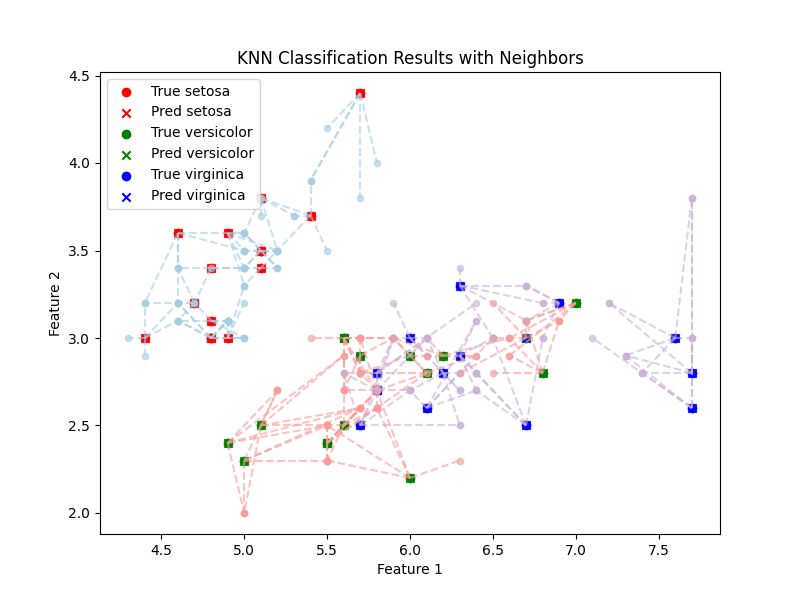
plt.ylabel("Feature 2")

plt.legend()

plt.show()

**运行结果：**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | precision | recall | f1-score | support |
| accuracy |  |  | 0.98 | 45 |
| macro avg | 0.98 | 0.98 | 0.98 | 45 |
| weighted | 0.98 | 0.98 | 0.98 | 45 |

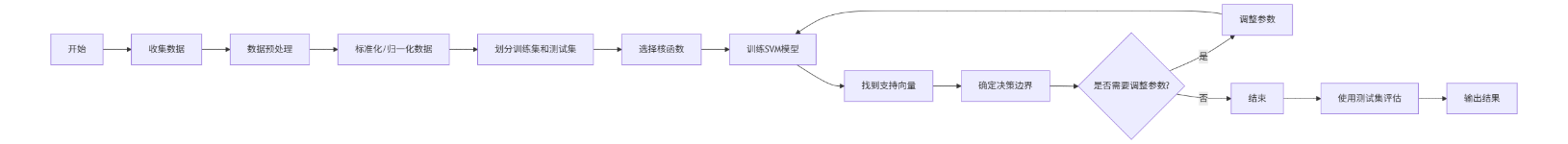
****

## **SVM算法**

**题目：**基于SVM的标准数据集分类。（对UCI-Iris数据集进行分类）, 并和K-最近邻算法效果比较。

**答：**SVM基于支持向量划分，划分只能是线性的，但KNN可以有丰富的非线性划分，但有过拟合风险。

**流程图：**



**代码：**

import numpy as np

from sklearn import datasets

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

from sklearn.svm import SVC

from sklearn.metrics import accuracy\_score, classification\_report

def load\_data():

    iris = datasets.load\_iris()

    X = iris.data[:, :2]

    y = iris.target

    return X, y

def split\_data(X, y, test\_size=0.3, random\_state=42):

    return train\_test\_split(X, y, test\_size=test\_size, random\_state=random\_state)

def train\_svm(X\_train, y\_train, kernel='linear', C=1.0):

    model = SVC(kernel=kernel, C=C, random\_state=42)

    model.fit(X\_train, y\_train)

    return model

def evaluate\_model(model, X\_test, y\_test):

    y\_pred = model.predict(X\_test)

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

    print("Accuracy:", accuracy)

    print("\nClassification Report:\n", classification\_report(y\_test, y\_pred))

if \_\_name\_\_ == "\_\_main\_\_":

    X, y = load\_data()

    X\_train, X\_test, y\_train, y\_test = split\_data(X, y)

    svm\_model = train\_svm(X\_train, y\_train)

    evaluate\_model(svm\_model, X\_test, y\_test)

    import matplotlib.pyplot as plt

    def plot\_svm\_decision\_boundary(model, X, y):

        plt.figure(figsize=(8, 6))

        for i, color in zip(range(len(set(y))), ['red', 'green', 'blue']):

            plt.scatter(X[y == i, 0], X[y == i, 1], color=color, label=f"Class {i}")

        plt.scatter(model.support\_vectors\_[:, 0], model.support\_vectors\_[:, 1],

                    s=100, facecolors='none', edgecolors='k', label='Support Vectors')

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

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

        xx, yy = np.meshgrid(np.arange(x\_min, x\_max, 0.01), np.arange(y\_min, y\_max, 0.01))

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

        Z = Z.reshape(xx.shape)

        plt.contourf(xx, yy, Z, alpha=0.3, cmap=plt.cm.Paired)

        plt.title("SVM Decision Boundary with Support Vectors")

        plt.xlabel("Feature 1")

        plt.ylabel("Feature 2")

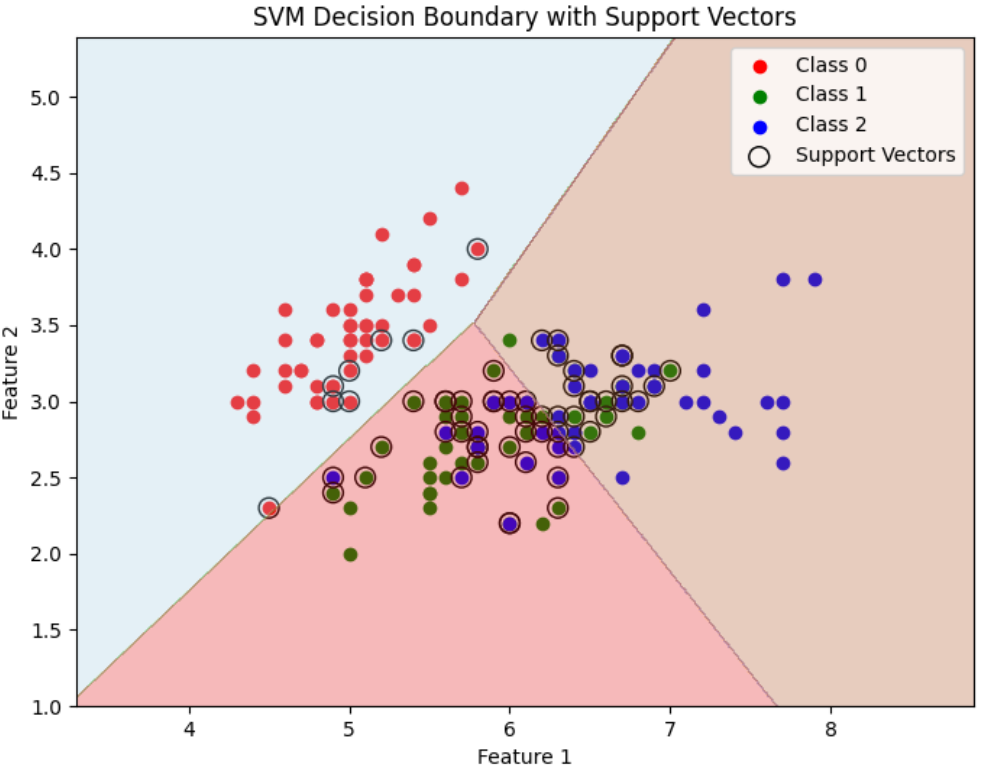
        plt.legend()

        plt.show()

    plot\_svm\_decision\_boundary(svm\_model, X, y)

**运行结果：**

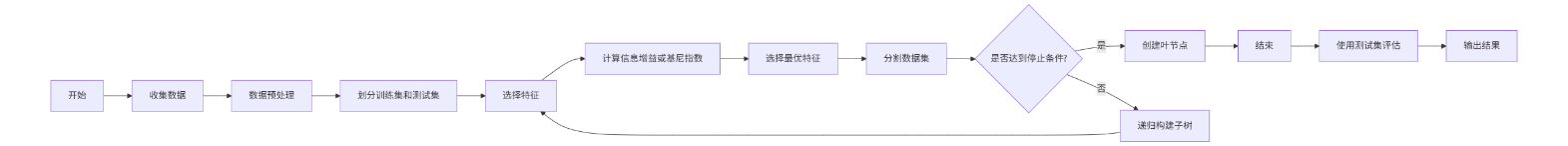
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | precision | recall | f1-score | support |
| accuracy |  |  | 0.80 | 45 |
| macro avg | 0.78 | 0.77 | 0.77 | 45 |
| weighted | 0.81 | 0.80 | 0.80 | 45 |

****

## **决策树算法**

**题目：**天气因素有温度、湿度和刮风等，通过给出数据，使用决策树算法学习分类，输出一个人是运动和不运动与天气之间的规则树。

**流程图：**



**代码：**

from sklearn.tree import DecisionTreeClassifier, export\_text, plot\_tree

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

from sklearn.metrics import accuracy\_score

import pandas as pd

import matplotlib.pyplot as plt

data = {

    "Weather": ["Sunny", "Sunny", "Cloudy", "Rainy", "Rainy", "Rainy", "Cloudy", "Sunny", "Sunny", "Rainy", "Sunny", "Cloudy", "Cloudy", "Rainy"],

    "Temperature": [85, 80, 83, 70, 68, 65, 64, 72, 69, 75, 75, 72, 81, 71],

    "Humidity": [85, 90, 78, 96, 80, 70, 65, 95, 70, 80, 70, 90, 75, 80],

    "Windy": ["No", "Yes", "No", "No", "No", "Yes", "Yes", "No", "No", "No", "Yes", "Yes", "No", "Yes"],

    "Play": ["No", "No", "Yes", "Yes", "Yes", "No", "Yes", "No", "Yes", "Yes", "Yes", "Yes", "Yes", "No"]

}

df = pd.DataFrame(data)

df\_encoded = pd.get\_dummies(df, columns=["Weather", "Temperature", "Humidity", "Windy"], drop\_first=True)

X = df\_encoded.drop("Play", axis=1)

y = df["Play"]

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

clf = DecisionTreeClassifier(criterion="entropy", random\_state=42)

clf.fit(X\_train, y\_train)

y\_pred = clf.predict(X\_test)

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

tree\_rules = export\_text(clf, feature\_names=list(X.columns))

print("\nDecision Tree Rules:\n", tree\_rules)

plt.figure(figsize=(12, 8))

plot\_tree(clf, feature\_names=X.columns, class\_names=clf.classes\_, filled=True)

plt.title("Decision Tree Visualization")

plt.show()

**运行结果：**

Accuracy: 0.6

Decision Tree Rules:

|--- Humidity\_95 <= 0.50

| |--- Windy\_Yes <= 0.50

| | |--- class: Yes

| |--- Windy\_Yes > 0.50

| | |--- Temperature\_75 <= 0.50

| | | |--- Humidity\_80 <= 0.50

| | | | |--- Weather\_Sunny <= 0.50

| | | | | |--- class: Yes

| | | | |--- Weather\_Sunny > 0.50

| | | | | |--- class: No

| | | |--- Humidity\_80 > 0.50

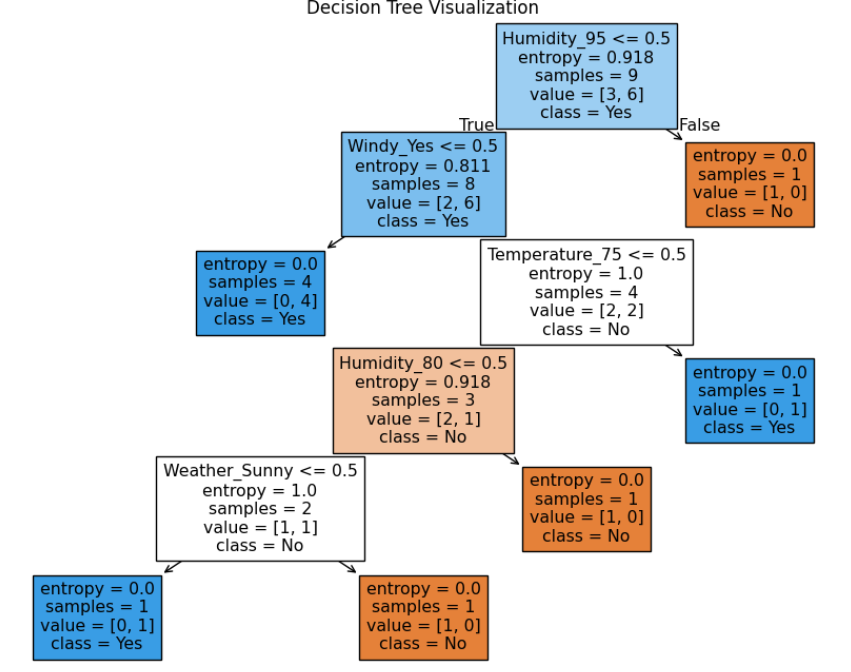
| | | | |--- class: No

| | |--- Temperature\_75 > 0.50

| | | |--- class: Yes

|--- Humidity\_95 > 0.50

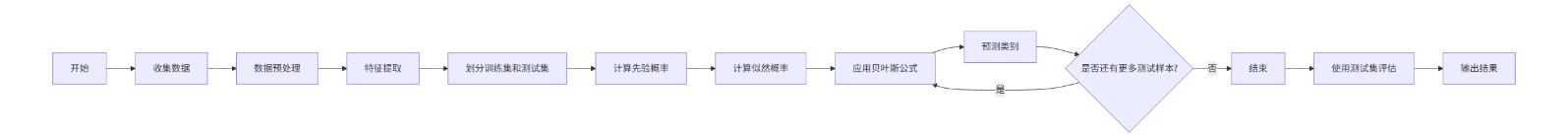
| |--- class: No

****

## **朴素贝叶斯算法**

**题目：**天气因素有温度、湿度和刮风等，通过给出数据，使用朴素贝叶斯算法学习分类，输出一个人是运动和不运动与天气之间的概率关系。

**流程图：**



**代码：**

import pandas as pd

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

from sklearn.naive\_bayes import GaussianNB

from sklearn.metrics import accuracy\_score, classification\_report

data = {

    "Weather": ["Sunny", "Sunny", "Cloudy", "Rainy", "Rainy", "Rainy", "Cloudy", "Sunny", "Sunny", "Rainy", "Sunny", "Cloudy", "Cloudy", "Rainy"],

    "Temperature": [85, 80, 83, 70, 68, 65, 64, 72, 69, 75, 75, 72, 81, 71],

    "Humidity": [85, 90, 78, 96, 80, 70, 65, 95, 70, 80, 70, 90, 75, 80],

    "Windy": ["No", "Yes", "No", "No", "No", "Yes", "Yes", "No", "No", "No", "Yes", "Yes", "No", "Yes"],

    "Play": ["No", "No", "Yes", "Yes", "Yes", "No", "Yes", "No", "Yes", "Yes", "Yes", "Yes", "Yes", "No"]

}

df = pd.DataFrame(data)

df\_encoded = pd.get\_dummies(df, columns=["Weather", "Temperature", "Humidity", "Windy"], drop\_first=True)

X = df\_encoded.drop("Play", axis=1)

y = df["Play"].map({"No": 0, "Yes": 1})

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

model = GaussianNB()

model.fit(X\_train, y\_train)

y\_pred = model.predict(X\_test)

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

print("\nReport:\n", classification\_report(y\_test, y\_pred))

probabilities = model.predict\_proba(X\_test)

for i, prob in enumerate(probabilities):

    print(f"Sample {i + 1} probability: Yes={prob[0]:.2f}, No={prob[1]:.2f}")

**运行结果：**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | precision | recall | f1-score | support |
| accuracy |  |  | 0.40 | 45 |
| macro avg | 0.42 | 0.42 | 0.40 | 45 |
| weighted | 0.43 | 0.40 | 0.40 | 45 |

Sample 1 probability: Yes=0.00, No=1.00

Sample 2 probability: Yes=1.00, No=0.00

Sample 3 probability: Yes=1.00, No=0.00

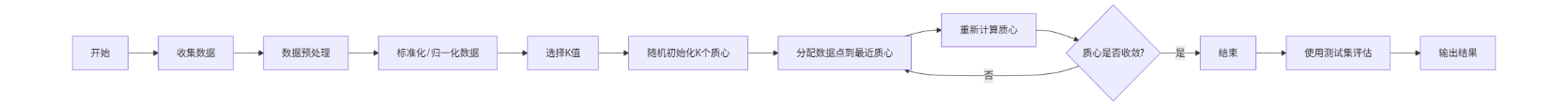
Sample 4 probability: Yes=1.00, No=0.00

Sample 5 probability: Yes=0.00, No=1.00

## **K-means算法**

**题目：**基于K-means算法的图像分割，实验数据如下图。要求输出图片分割结果：合适的背景区域（两个）和前景区域（大象）。图片大小500\*800像素，格式为.jpg，每个像素可以表示为三维向量（分别对应JPEG图像中的红色、绿色和蓝色通道）。

**流程图：**



**代码：**

from PIL import Image

import numpy as np

from sklearn.cluster import KMeans

import matplotlib.pyplot as plt

image\_path = "image.jpg"

image = Image.open(image\_path).convert("RGB")

image = np.array(image)

pixels = image.reshape(-1, 3)

k = 3

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

kmeans.fit(pixels)

segmented\_pixels = kmeans.cluster\_centers\_[kmeans.labels\_]

segmented\_image = segmented\_pixels.reshape(image.shape).astype(np.uint8)

plt.figure(figsize=(10, 5))

plt.subplot(1, 2, 1)

plt.title("Original Image")

plt.imshow(image)

plt.axis("off")

plt.subplot(1, 2, 2)

plt.title("Segmented Image")

plt.imshow(segmented\_image)

plt.axis("off")

plt.show()

**运行结果：**

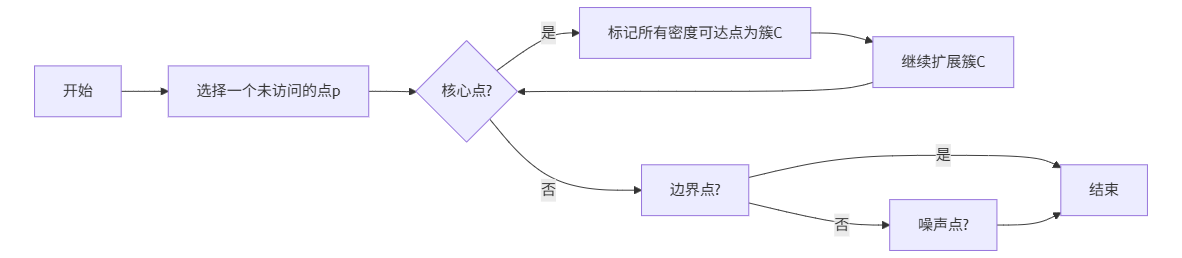
****

## **DBSCAN算法**

**题目：**基于DBSCAN算法的标准数据集聚类。（对UCI-Iris数据集进行聚类），和K-means算法效果比较。

**答：**DBSCAN可以自动确定类别，但K-means不能，当类别数确定时，使用K-means效果更好，但类别数不定时，使用DBSCAN算法可以自动确定合理的类别数。

**流程图：**



**代码：**

import pandas as pd

import numpy as np

from sklearn.datasets import load\_iris

from sklearn.cluster import DBSCAN, KMeans

from sklearn.decomposition import PCA

from sklearn.metrics import silhouette\_score

import matplotlib.pyplot as plt

iris = load\_iris()

X = iris.data

y = iris.target

dbscan = DBSCAN(eps=0.5, min\_samples=12)

dbscan\_labels = dbscan.fit\_predict(X)

kmeans = KMeans(n\_clusters=3, random\_state=42)

kmeans\_labels = kmeans.fit\_predict(X)

dbscan\_silhouette = silhouette\_score(X, dbscan\_labels) if len(set(dbscan\_labels)) > 1 else -1

kmeans\_silhouette = silhouette\_score(X, kmeans\_labels)

print("DBSCAN Silhouette Score:", dbscan\_silhouette)

print("K-means Silhouette Score:", kmeans\_silhouette)

pca = PCA(n\_components=2)

X\_pca = pca.fit\_transform(X)

plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1)

plt.scatter(X\_pca[:, 0], X\_pca[:, 1], c=dbscan\_labels, cmap="viridis", s=50)

plt.title("DBSCAN Clustering")

plt.colorbar()

plt.subplot(1, 2, 2)

plt.scatter(X\_pca[:, 0], X\_pca[:, 1], c=kmeans\_labels, cmap="viridis", s=50)

plt.title("K-means Clustering")

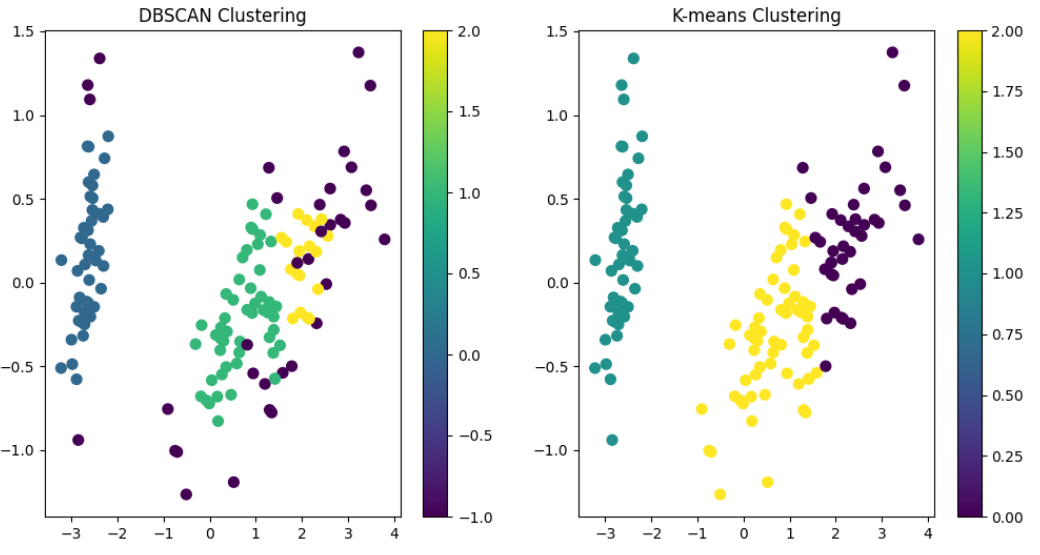
plt.colorbar()

plt.show()

**运行结果：**

DBSCAN Silhouette Score: 0.3286

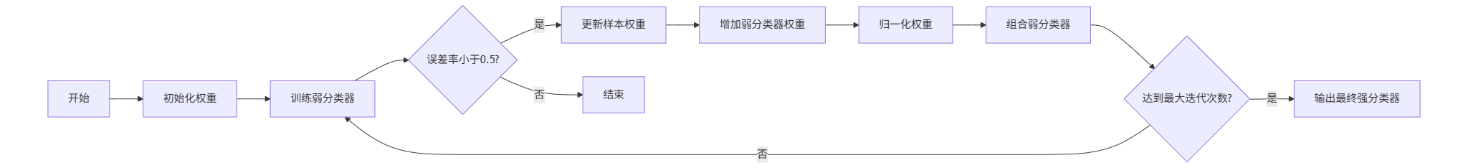
K-means Silhouette Score: 0.5512

****

## **AdaBoost算法**

**题目：**基于AdaBoost算法实现标准数据集分类。（对UCI-Iris数据集进行分类）

**流程图：**



**代码：**

import pandas as pd

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

from sklearn.ensemble import AdaBoostClassifier

from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import accuracy\_score, classification\_report

from sklearn.datasets import load\_iris

iris = load\_iris()

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

y = pd.Series(iris.target)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=43)

base\_estimator = DecisionTreeClassifier(max\_depth=1)

adaboost = AdaBoostClassifier(estimator=base\_estimator, n\_estimators=50, random\_state=42)

adaboost.fit(X\_train, y\_train)

y\_pred = adaboost.predict(X\_test)

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

print("\nClassification Report:\n", classification\_report(y\_test, y\_pred))

**运行结果：**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | precision | recall | f1-score | support |
| accuracy |  |  | 0.96 | 45 |
| macro avg | 0.95 | 0.95 | 0.95 | 45 |
| weighted | 0.96 | 0.96 | 0.96 | 45 |