智能数据挖掘上机作业报告

人工智能（图灵）

汤栋文 22009200601

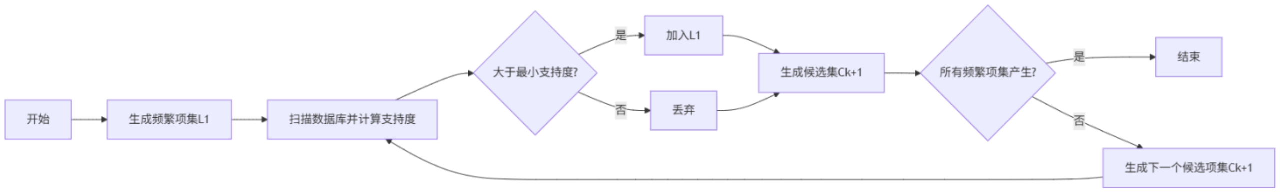
2025-05-18

目录

1. Apriori 算法

题目： 编程实现 Apriori 算法（关联规则挖掘）

流程图：



代码：

```python
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}")
```

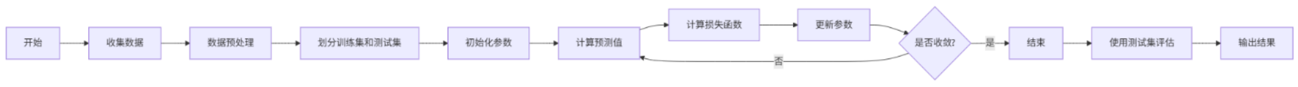
运行结果：

```
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
```

2. 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 $y = ax + b$。按上面的分析，误差函数为 $e = \sum(y_i - ax_i - b)^2$。

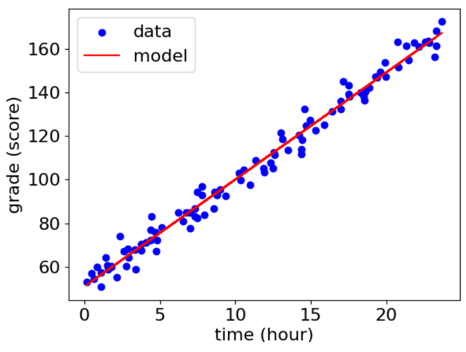
流程图：



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()
```

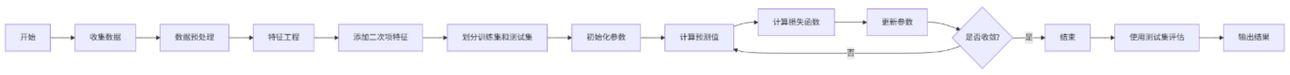
运行结果：Model: $y = 4.90x + 51.08$



3. 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$y = ax^2 + bx + c$。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：$e = \sum\left(y_i - ax_i^2 - bx_i - c\right)^2$

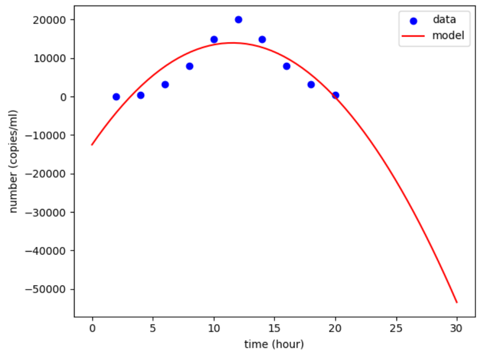
流程图：



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()
```

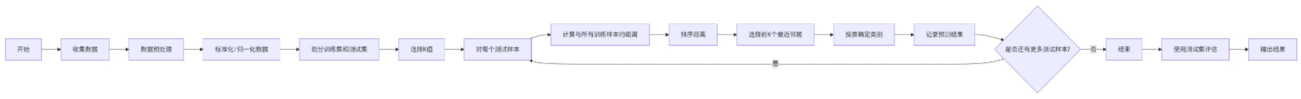
运行结果：Model: $y = -198.28x^2 + 4582.60x - 12527.00$



4. 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：



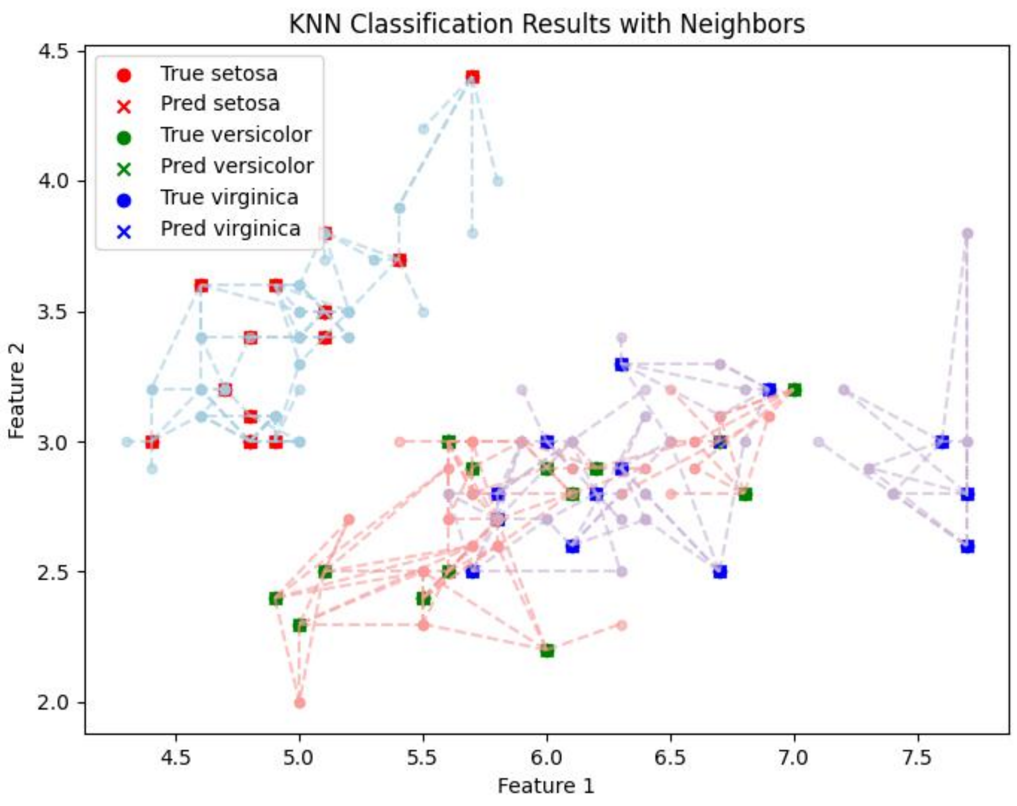
代码：

```
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.t
arget_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()
```

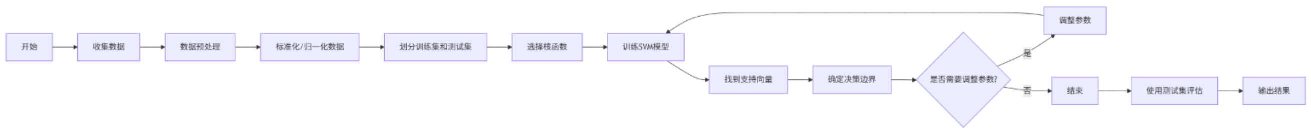
运行结果：

	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



5. 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：

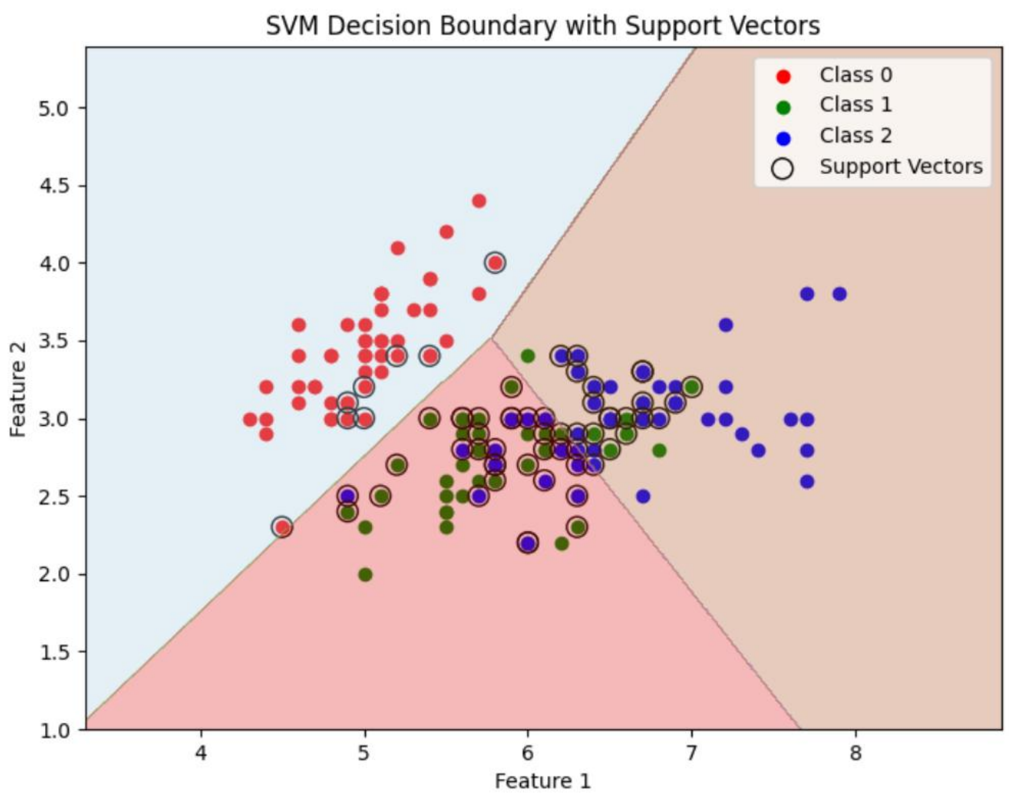


代码：

```python
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：

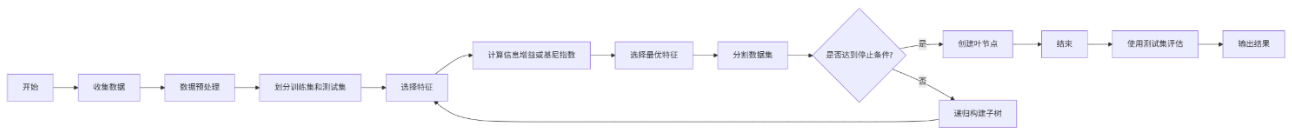
	precision	*recall*	*f1-score*	*support*
accuracy			0.80	45
macro avg	0.78	0.77	0.77	45
weighted	0.81	0.80	0.80	45



6. 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：



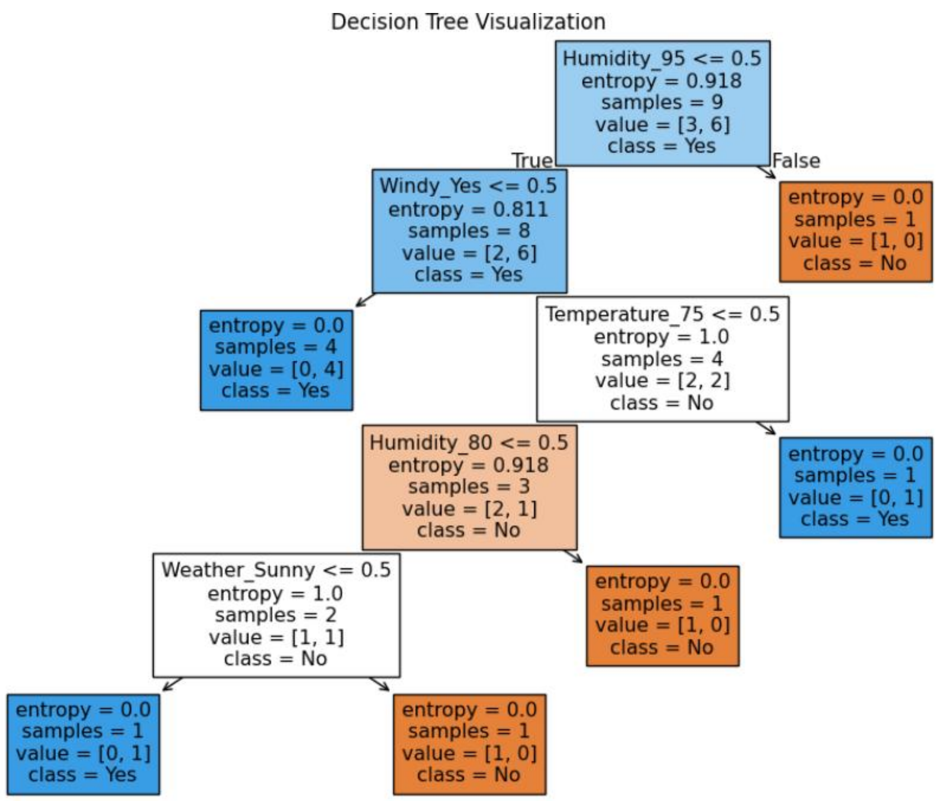
代码：

```
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：

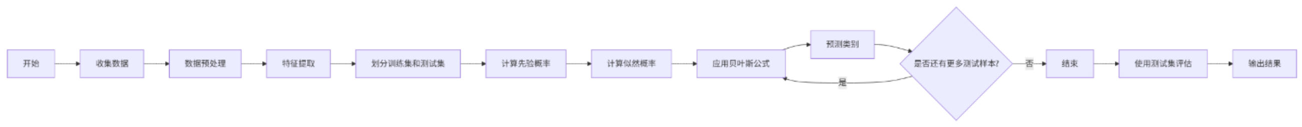
```
Accuracy: 0.6
Decision Tree Rules:
|--- Humidity_95 <= 0.50
|   |--- Windy_Yes <= 0.50
|   |   |--- class: Yes
|   |--- Windy_Yes >  0.50
|   |   |--- Temperature_75 <= 0.50
|   |   |   |--- Humidity_80 <= 0.50
|   |   |   |   |--- Weather_Sunny <= 0.50
|   |   |   |   |   |--- class: Yes
|   |   |   |   |--- Weather_Sunny >  0.50
|   |   |   |   |   |--- class: No
|   |   |   |--- Humidity_80 >  0.50
|   |   |   |   |--- class: No
|   |   |--- Temperature_75 >  0.50
|   |   |   |--- class: Yes
|--- Humidity_95 >  0.50
|   |--- class: No
```



7. 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：

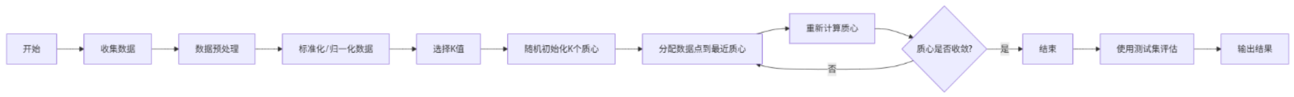
	precision	*recall*	*f1-score*	*support*
accuracy			0.40	45
macro avg	0.42	0.42	0.40	45
weighted	0.43	0.40	0.40	45

```
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
```

8. 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）。

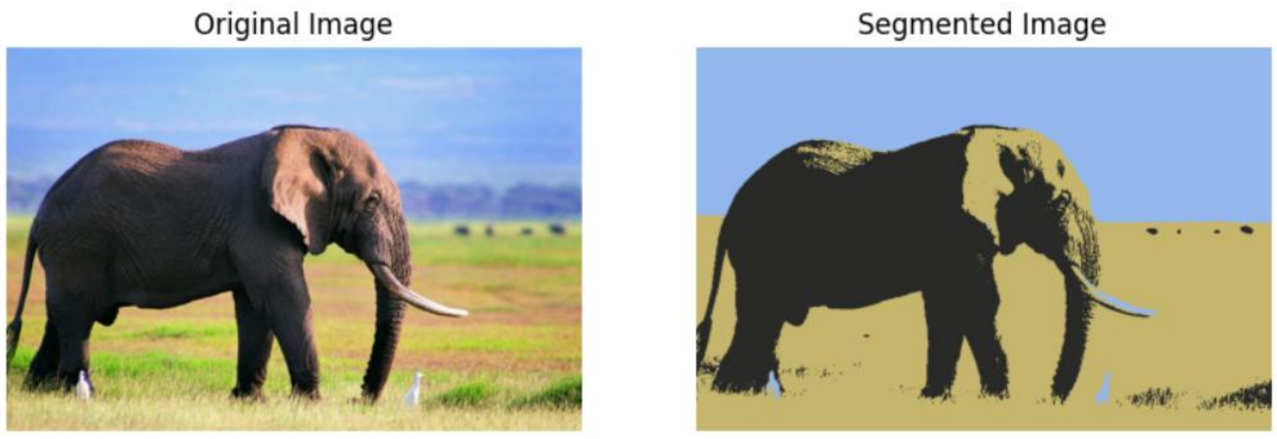
流程图：



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：

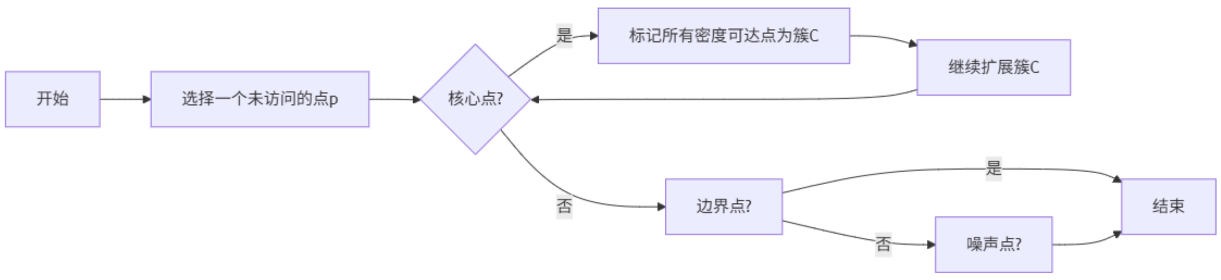


9. DBSCAN 算法

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：



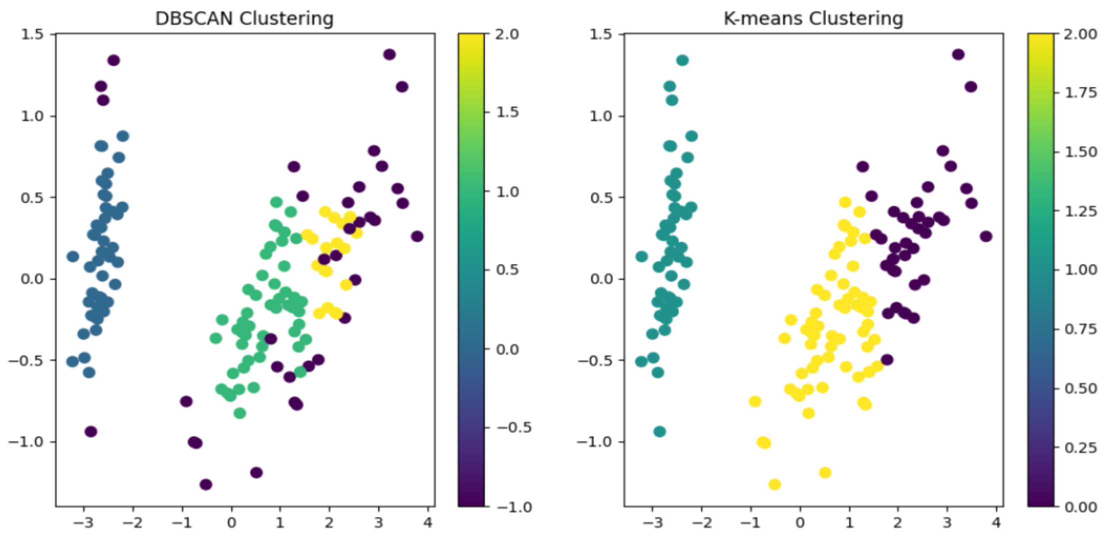
代码：

```
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

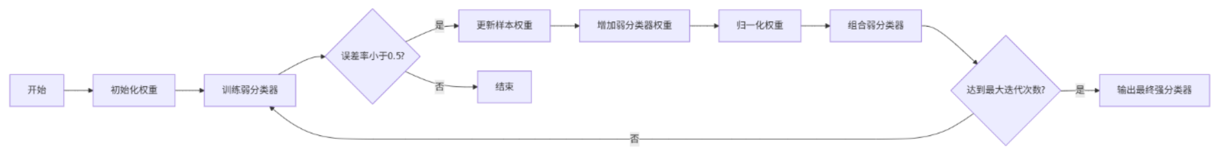
K-means Silhouette Score: 0.5512



10. 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