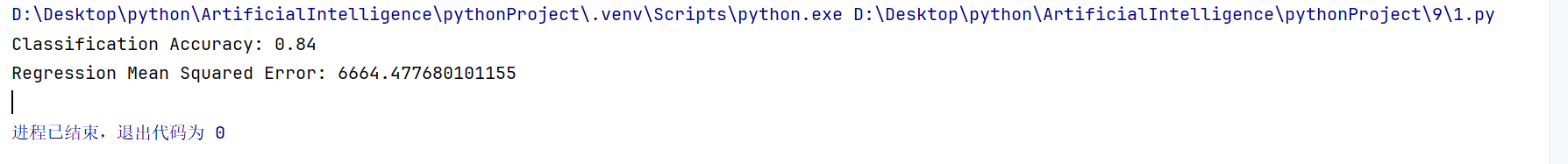
1、决策树模型

（1）使用make\_classification生成分类数据集（要求样本个数1000，特征个数为10），划分训练集和测试集，使用决策树分类模型训练数据，输出测试集的性能指标

（2）使用make\_regression()生成回归数据集（要求样本个数1000，特征个数为10）或者导入from sklearn.datasets 房价回归数据集，划分训练集和测试集，使用决策树回归模型训练数据，输出测试集的性能指标

from sklearn.datasets import make\_classification, make\_regression  
from sklearn.model\_selection import train\_test\_split  
from sklearn.tree import DecisionTreeClassifier, DecisionTreeRegressor  
from sklearn.metrics import accuracy\_score, mean\_squared\_error  
  
# 生成分类数据集  
X\_classification, y\_classification = make\_classification(n\_samples=1000, n\_features=10, random\_state=42)  
X\_train\_clf, X\_test\_clf, y\_train\_clf, y\_test\_clf = train\_test\_split(X\_classification, y\_classification, test\_size=0.2, random\_state=42)  
  
# 训练决策树分类模型  
clf = DecisionTreeClassifier(random\_state=42)  
clf.fit(X\_train\_clf, y\_train\_clf)  
  
# 输出测试集的性能指标  
y\_pred\_clf = clf.predict(X\_test\_clf)  
accuracy = accuracy\_score(y\_test\_clf, y\_pred\_clf)  
print("Classification Accuracy:", accuracy)  
  
# 生成回归数据集  
X\_regression, y\_regression = make\_regression(n\_samples=1000, n\_features=10, random\_state=42)  
X\_train\_reg, X\_test\_reg, y\_train\_reg, y\_test\_reg = train\_test\_split(X\_regression, y\_regression, test\_size=0.2, random\_state=42)  
  
# 训练决策树回归模型  
reg = DecisionTreeRegressor(random\_state=42)  
reg.fit(X\_train\_reg, y\_train\_reg)  
  
# 输出测试集的性能指标  
y\_pred\_reg = reg.predict(X\_test\_reg)  
mse = mean\_squared\_error(y\_test\_reg, y\_pred\_reg)  
print("Regression Mean Squared Error:", mse)



2、贝叶斯模型

对sklearn中的肿瘤数据集进行分类，该数据集包含569个样本数据，30个特征值。

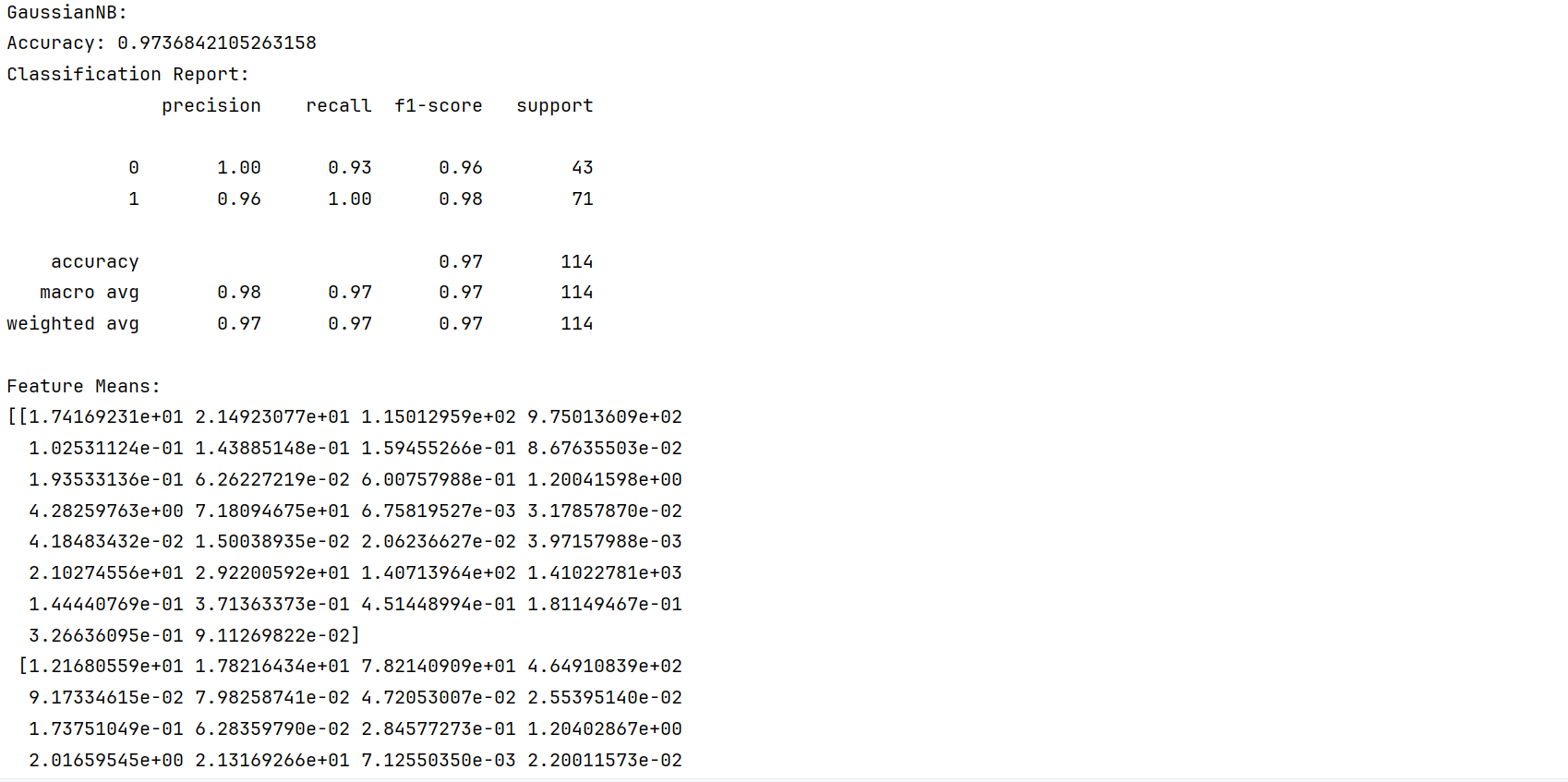
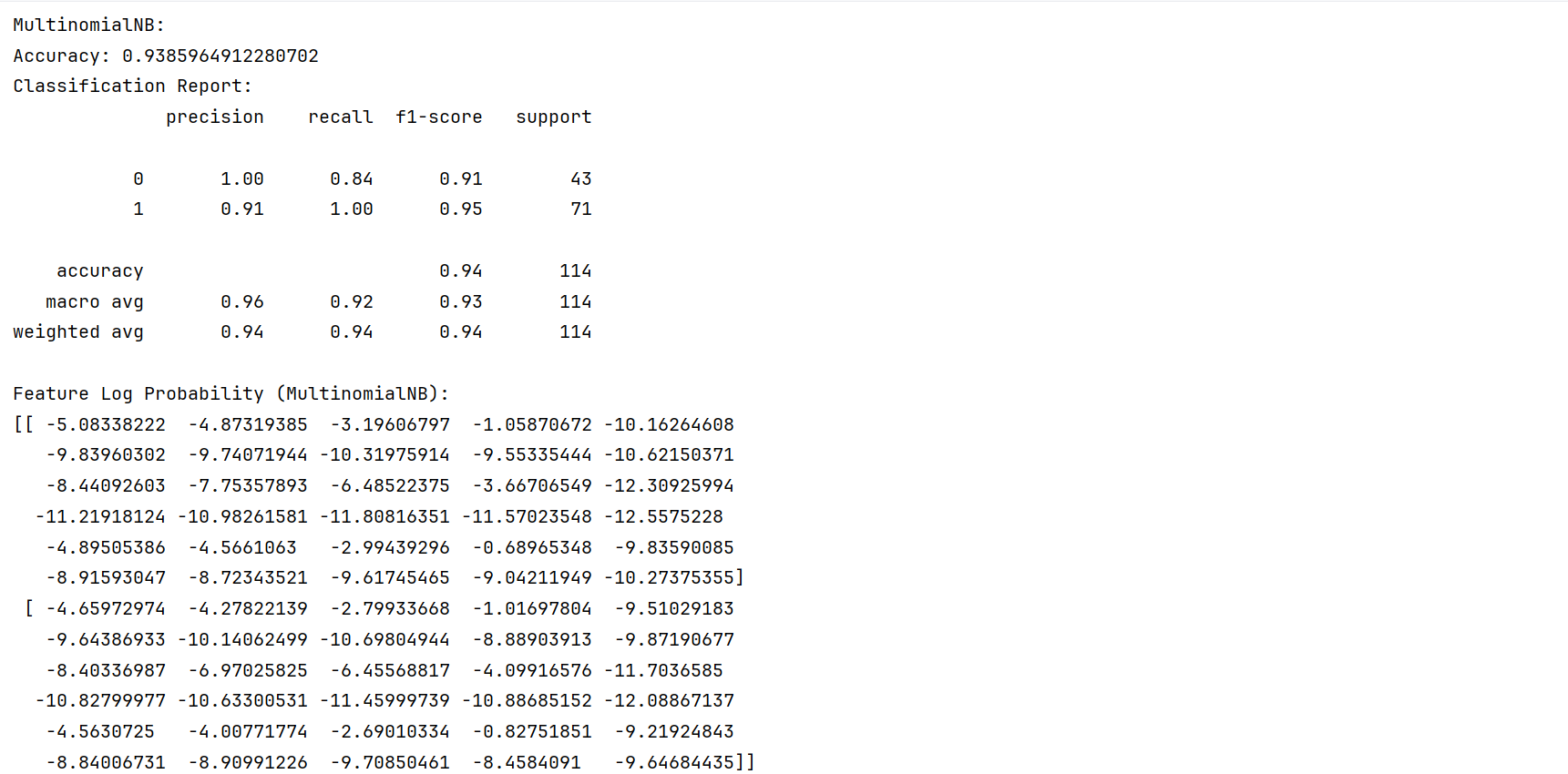
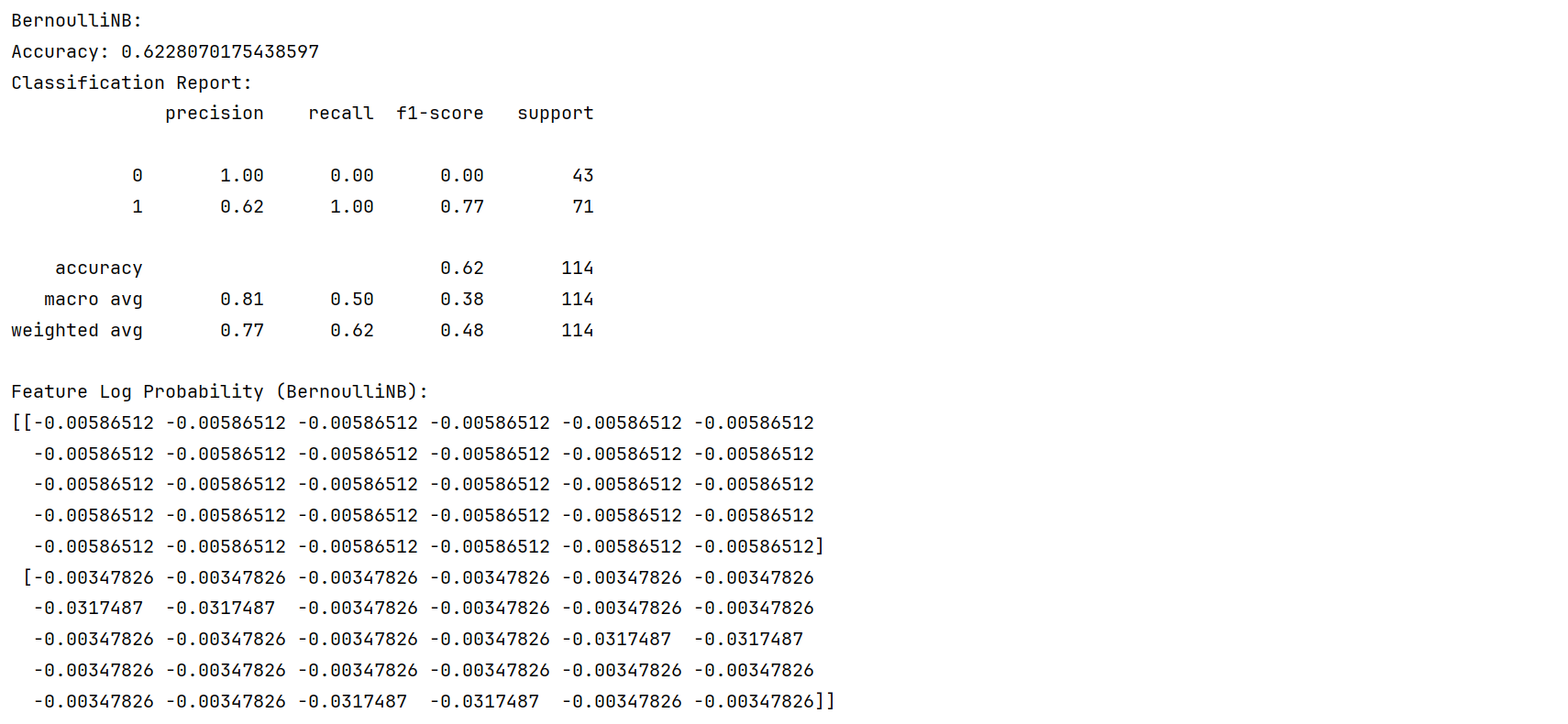
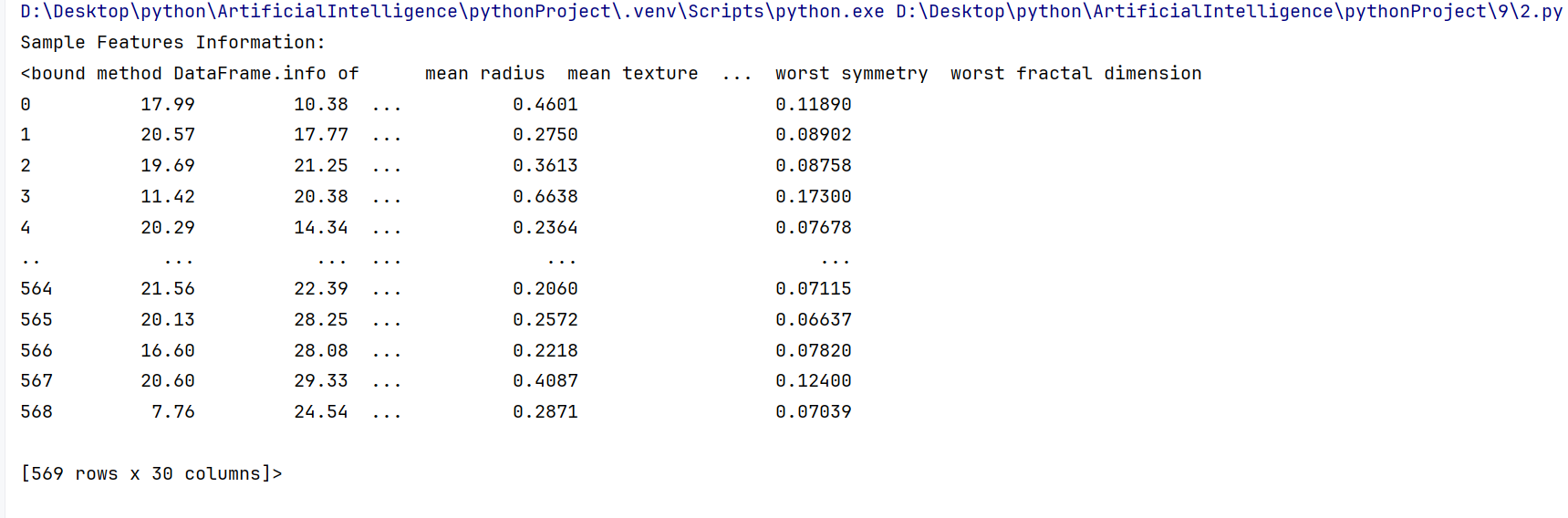
from sklearn.datasets import load\_breast\_cancer，输出数据样本特征信息

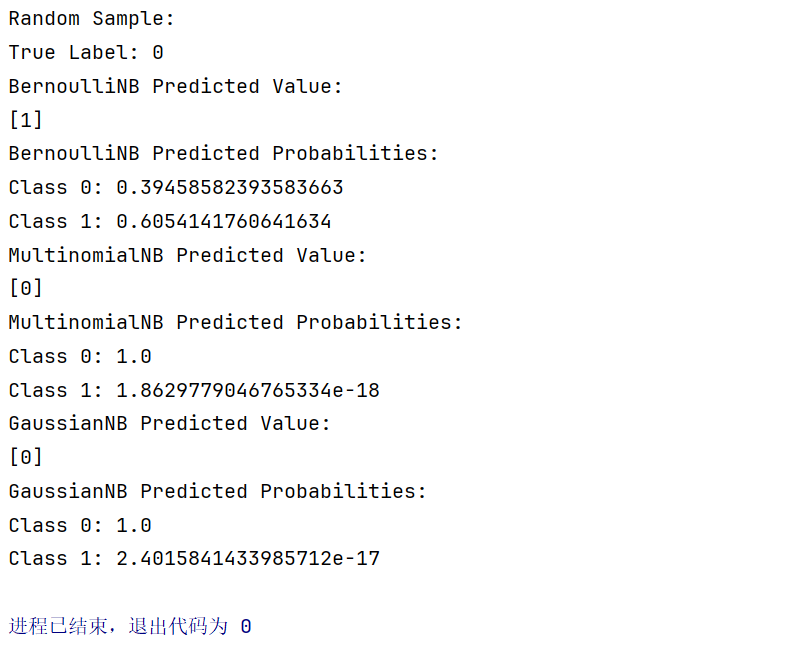
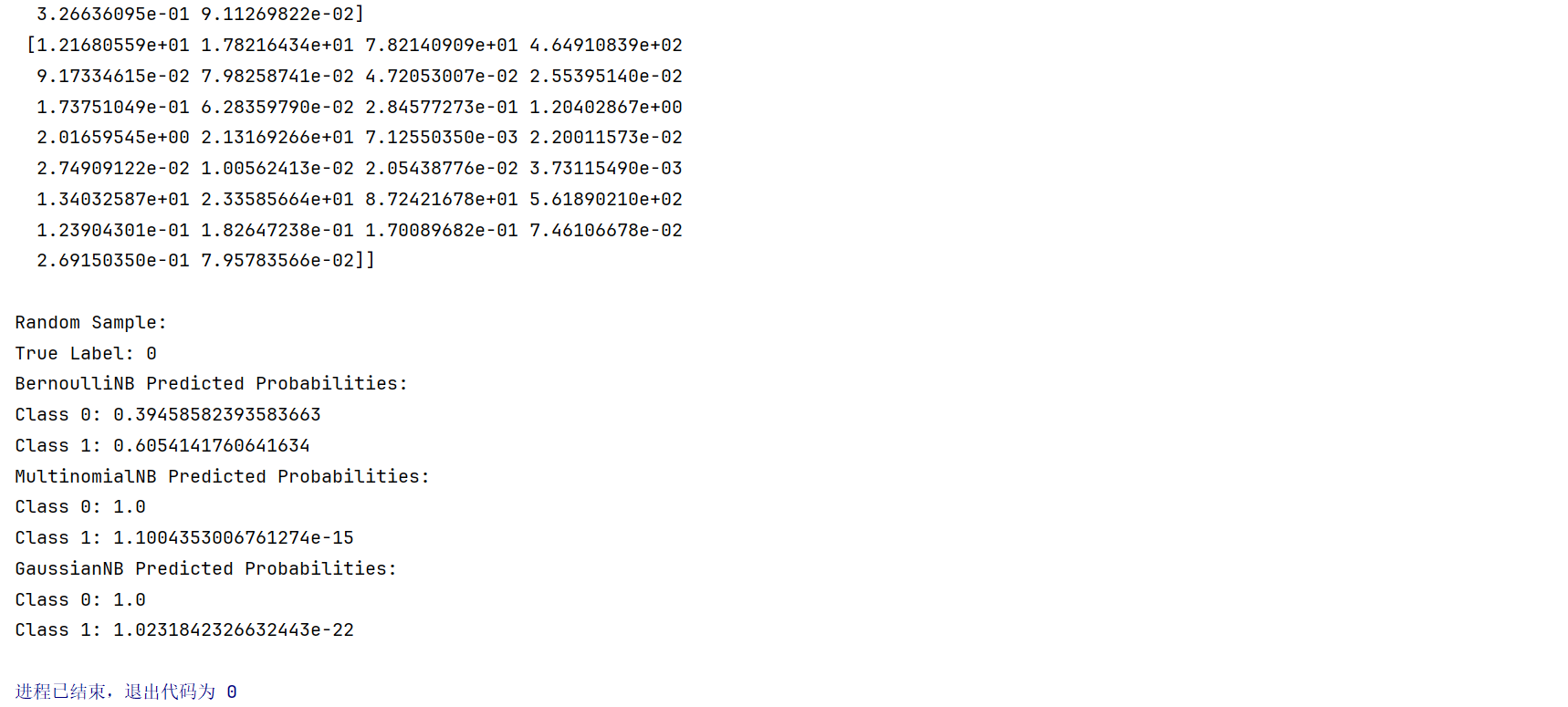
(1)划分训练集和测试集，

(2)分别使用BernoulliNB、MultinomialNB和GaussianNB三种朴素贝叶斯模型进行训练，并输出分类性能指标，特征重要性，

(3)随机测试其中一个样本，输出其预测值，及每个类别的概率。

import pandas as pd  
from sklearn.datasets import load\_breast\_cancer  
from sklearn.model\_selection import train\_test\_split  
from sklearn.naive\_bayes import BernoulliNB, MultinomialNB, GaussianNB  
from sklearn.metrics import accuracy\_score, classification\_report  
import numpy as np  
  
# Load breast cancer dataset  
data = load\_breast\_cancer()  
# print(type(data))  
X = data.data  
y = data.target  
# print(type(X))  
df = pd.DataFrame(data.data, columns=data.feature\_names)  
# Output sample feature information  
print("Sample Features Information:")  
print(df.info)  
  
# Split the dataset into training and testing sets  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)  
  
# Train and evaluate three types of Naive Bayes models  
models = {  
 "BernoulliNB": BernoulliNB(),  
 "MultinomialNB": MultinomialNB(),  
 "GaussianNB": GaussianNB()  
}  
  
for name, model in models.items():  
 print(f"\n{name}:")  
 model.fit(X\_train, y\_train)  
 y\_pred = model.predict(X\_test)  
 accuracy = accuracy\_score(y\_test, y\_pred)  
 print("Accuracy:", accuracy)  
 print("Classification Report:")  
 print(classification\_report(y\_test, y\_pred, zero\_division=1))  
  
 # Output feature importance for GaussianNB  
 if name == "GaussianNB":  
 print("Feature Means:")  
 print(model.theta\_)  
  
 # For BernoulliNB and MultinomialNB, as they are not probability-based,  
 # we can't directly get feature importance in the same way as GaussianNB.  
 # However, we can output the learned parameters for inspection.  
 elif name == "BernoulliNB":  
 print("Feature Log Probability (BernoulliNB):")  
 print(model.feature\_log\_prob\_)  
  
 elif name == "MultinomialNB":  
 print("Feature Log Probability (MultinomialNB):")  
 print(model.feature\_log\_prob\_)  
  
# Randomly test one sample  
random\_index = np.random.randint(0, len(X\_test))  
sample = X\_test[random\_index]  
true\_label = y\_test[random\_index]  
  
print("\nRandom Sample:")  
print("True Label:", true\_label)  
  
# Output predicted probabilities for each class for each model  
for name, model in models.items():  
 prob = model.predict\_proba([sample])[0]  
 print(f"{name} Predicted Value:")  
 print(model.predict([sample]))  
 print(f"{name} Predicted Probabilities:")  
 for i, p in enumerate(prob):  
 print(f"Class {i}: {p}")





3、集成模型

（1）使用make\_moons生成分类数据，

（2）自定义三个基学习器

（3）分别使用voting集成模型、bagging集成模型、随机森林和Boosting模型训练数据，输出用于评估模型的性能指标。

from sklearn.datasets import make\_moons  
from sklearn.model\_selection import train\_test\_split  
from sklearn.ensemble import VotingClassifier, BaggingClassifier, RandomForestClassifier, AdaBoostClassifier  
from sklearn.tree import DecisionTreeClassifier  
from sklearn.linear\_model import LogisticRegression  
from sklearn.svm import SVC  
from sklearn.metrics import accuracy\_score, classification\_report  
  
# 生成分类数据  
X, y = make\_moons(n\_samples=1000, noise=0.3, random\_state=42)  
  
# 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)  
  
# 定义基学习器  
log\_clf = LogisticRegression(solver="liblinear", random\_state=42)  
rnd\_clf = RandomForestClassifier(n\_estimators=10, random\_state=42)  
svm\_clf = SVC(gamma="auto", random\_state=42, probability=True)  
  
# Hard Voting Classifier  
voting\_clf = VotingClassifier(  
 estimators=[('lr', log\_clf), ('rf', rnd\_clf), ('svc', svm\_clf)],  
 voting='hard'  
)  
voting\_clf.fit(X\_train, y\_train)  
y\_pred\_voting = voting\_clf.predict(X\_test)  
print("Hard Voting Classifier Accuracy:", accuracy\_score(y\_test, y\_pred\_voting))  
print("Hard Voting Classifier Classification Report:")  
print(classification\_report(y\_test, y\_pred\_voting))  
  
# Soft Voting Classifier  
voting\_clf\_soft = VotingClassifier(  
 estimators=[('lr', log\_clf), ('rf', rnd\_clf), ('svc', svm\_clf)],  
 voting='soft'  
)  
voting\_clf\_soft.fit(X\_train, y\_train)  
y\_pred\_voting\_soft = voting\_clf\_soft.predict(X\_test)  
print("Soft Voting Classifier Accuracy:", accuracy\_score(y\_test, y\_pred\_voting\_soft))  
print("Soft Voting Classifier Classification Report:")  
print(classification\_report(y\_test, y\_pred\_voting\_soft))  
  
# Bagging Classifier  
bag\_clf = BaggingClassifier(  
 DecisionTreeClassifier(random\_state=42), n\_estimators=500,  
 max\_samples=100, bootstrap=True, n\_jobs=-1, random\_state=42  
)  
bag\_clf.fit(X\_train, y\_train)  
y\_pred\_bagging = bag\_clf.predict(X\_test)  
print("Bagging Classifier Accuracy:", accuracy\_score(y\_test, y\_pred\_bagging))  
print("Bagging Classifier Classification Report:")  
print(classification\_report(y\_test, y\_pred\_bagging))  
  
# Random Forest Classifier  
rnd\_clf = RandomForestClassifier(n\_estimators=500, max\_leaf\_nodes=16, n\_jobs=-1, random\_state=42)  
rnd\_clf.fit(X\_train, y\_train)  
y\_pred\_rf = rnd\_clf.predict(X\_test)  
print("Random Forest Classifier Accuracy:", accuracy\_score(y\_test, y\_pred\_rf))  
print("Random Forest Classifier Classification Report:")  
print(classification\_report(y\_test, y\_pred\_rf))  
  
# AdaBoost Classifier  
ada\_clf = AdaBoostClassifier(  
 DecisionTreeClassifier(max\_depth=1), n\_estimators=200,  
 algorithm="SAMME", learning\_rate=0.5, random\_state=42  
)  
ada\_clf.fit(X\_train, y\_train)  
y\_pred\_ada = ada\_clf.predict(X\_test)  
print("AdaBoost Classifier Accuracy:", accuracy\_score(y\_test, y\_pred\_ada))  
print("AdaBoost Classifier Classification Report:")  
print(classification\_report(y\_test, y\_pred\_ada))

