Machine Learning Algorithms Report

Shubham Kushwaha, 22/28090

BSC Computer Science Hons.

Machine Learning Algorithms Report

Name: Shubham Kushwaha **Roll No:** 22/28090
Course: BSC Computer Science Hons.
Submitted to: Dr. Uma Ojha Ma'am

Date: $(date)

Naive Bayes Classifier

Code

```
import numpy as np
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report, roc_auc_score

# Load the Iris dataset
iris = load_iris()
X = iris.data
y = iris.target

# Split the dataset into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Create a Naive Bayes classifier
model = GaussianNB()
model.fit(X_train, y_train)

# Predict the test set results
y_pred = model.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
class_report = classification_report(y_test, y_pred)
roc_auc = roc_auc_score(y_test, model.predict_proba(X_test), multi_class='ovr')

print(f"Naive Bayes Classifier Results:")
print(f"Accuracy: {accuracy}")
print(f"Confusion Matrix:\n{conf_matrix}")
```

```
print(f"Classification Report:\n{class_report}")
print(f"ROC AUC Score: {roc_auc}")

# Perform k-cross-validation
cv_scores = cross_val_score(model, X, y, cv=10)
print(f"Cross-Validation Scores: {cv_scores}")
print(f"Mean CV Score: {np.mean(cv_scores)}")
```

Output

```
Naive Bayes Classifier Results:
Accuracy: 0.9777777777777777
Confusion Matrix:
[[19  0  0]
 [ 0 12  1]
 [ 0  0 13]]
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        19
           1       1.00      0.92      0.96        13
           2       0.93      1.00      0.96        13

    accuracy                           0.98        45
   macro avg       0.98      0.97      0.97        45
weighted avg       0.98      0.98      0.98        45

ROC AUC Score: 1.0
Cross-Validation Scores: [0.93333333 0.93333333 1.         0.93333333 0.93333333 0.93333333
 0.86666667 1.         1.         1.        ]
Mean CV Score: 0.9533333333333334
```

Simple Linear Regression

Code

```
import numpy as np
import pandas as pd
from sklearn.datasets import fetch_california_housing
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
import warnings
warnings.filterwarnings("ignore")

# Load the California Housing dataset instead of Boston
housing = fetch_california_housing()
X = housing.data[:, 0:1]  # Using MedInc (median income) as feature
y = housing.target

# Split the dataset into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

```
# Create a Linear Regression model
model = LinearRegression()
model.fit(X_train, y_train)

# Predict the test set results
y_pred = model.predict(X_test)

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
r2 = r2_score(y_test, y_pred)

print("Simple Linear Regression Results:")
print(f"Coefficients: {model.coef_}")
print(f"Intercept: {model.intercept_}")
print(f"Mean Squared Error: {mse}")
print(f"Root Mean Squared Error: {rmse}")
print(f"R² Score: {r2}")

# Perform k-cross-validation
cv_scores = cross_val_score(model, X, y, cv=10, scoring='neg_mean_squared_error')
cv_rmse = np.sqrt(-cv_scores)
print(f"Cross-Validation RMSE Scores: {cv_rmse}")
print(f"Mean CV RMSE: {np.mean(cv_rmse)}")
```

Output

```
Simple Linear Regression Results:
Coefficients: [0.41819327]
Intercept: 0.4479496555604323
Mean Squared Error: 0.6917979868048499
Root Mean Squared Error: 0.8317439430526982
R² Score: 0.47293192589970245
Cross-Validation RMSE Scores: [0.6146956  0.85167462 1.06992543 0.5842816  0.99597495 0.75522504
 0.7582148  1.05291243 0.96392406 0.70727519]
Mean CV RMSE: 0.8354103716283486
```

Multiple Linear Regression

Code

```
import numpy as np
import pandas as pd
from sklearn.datasets import fetch_california_housing
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
import warnings
warnings.filterwarnings("ignore")

# Load the California Housing dataset
housing = fetch_california_housing()
X = housing.data
```

```
y = housing.target
feature_names = housing.feature_names

# Split the dataset into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Create a Multiple Linear Regression model
model = LinearRegression()
model.fit(X_train, y_train)

# Predict the test set results
y_pred = model.predict(X_test)

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
r2 = r2_score(y_test, y_pred)

print("Multiple Linear Regression Results:")
for i, coef in enumerate(model.coef_):
    print(f"Coefficient for {feature_names[i]}: {coef}")
print(f"Intercept: {model.intercept_}")
print(f"Mean Squared Error: {mse}")
print(f"Root Mean Squared Error: {rmse}")
print(f"R² Score: {r2}")

# Perform k-cross-validation
cv_scores = cross_val_score(model, X, y, cv=10, scoring='neg_mean_squared_error')
cv_rmse = np.sqrt(-cv_scores)
print(f"Cross-Validation RMSE Scores: {cv_rmse}")
print(f"Mean CV RMSE: {np.mean(cv_rmse)}")
```

Output

```
Multiple Linear Regression Results:
Coefficient for MedInc: 0.44582256530620973
Coefficient for HouseAge: 0.00968186798591678
Coefficient for AveRooms: -0.12209511171129188
Coefficient for AveBedrms: 0.7785995569755837
Coefficient for Population: -7.757404001697277e-07
Coefficient for AveOccup: -0.003370026670096733
Coefficient for Latitude: -0.41853674650062506
Coefficient for Longitude: -0.4336879759244037
Intercept: -37.05624133152496
Mean Squared Error: 0.5305677824766757
Root Mean Squared Error: 0.7284008391515455
R² Score: 0.595770232606166
Cross-Validation RMSE Scores: [0.69944301 0.65829982 0.94150821 0.62523308 0.86485438 0.72787552
 0.53664192 0.87935454 0.80190746 0.57228542]
Mean CV RMSE: 0.7307403363152192
```

Polynomial Regression

Code

```python
import numpy as np
import pandas as pd
from sklearn.datasets import fetch_california_housing
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import make_pipeline
from sklearn.metrics import mean_squared_error, r2_score
import warnings
warnings.filterwarnings("ignore")

# Load the California Housing dataset
housing = fetch_california_housing()
X = housing.data[:, 0:1]  # Using MedInc (median income) as feature
y = housing.target

# Split the dataset into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Create polynomial features
degree = 2
polyreg = make_pipeline(PolynomialFeatures(degree), LinearRegression())
polyreg.fit(X_train, y_train)

# Predict the test set results
y_pred = polyreg.predict(X_test)

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
r2 = r2_score(y_test, y_pred)

print("Polynomial Regression Results:")
print(f"Polynomial Degree: {degree}")
print(f"Mean Squared Error: {mse}")
print(f"Root Mean Squared Error: {rmse}")
print(f"R² Score: {r2}")

# Perform k-cross-validation
cv_scores = cross_val_score(polyreg, X, y, cv=10, scoring='neg_mean_squared_error')
cv_rmse = np.sqrt(-cv_scores)
print(f"Cross-Validation RMSE Scores: {cv_rmse}")
print(f"Mean CV RMSE: {np.mean(cv_rmse)}")
```

Output

```
Polynomial Regression Results:
Polynomial Degree: 2
Mean Squared Error: 0.685757652128837
Root Mean Squared Error: 0.8281048557573111
R² Score: 0.477533945022816
```

```
Cross-Validation RMSE Scores: [0.61999734 0.84614878 1.06640931 0.58824639 0.98328177 0.75585979
 0.77053276 1.05012559 0.94874158 0.70814222]
Mean CV RMSE: 0.8337485519550414
```

Lasso and Ridge Regression

Code

```python
import numpy as np
import pandas as pd
from sklearn.datasets import fetch_california_housing
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.linear_model import Lasso, Ridge
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error, r2_score
import warnings
warnings.filterwarnings("ignore")

# Load the California Housing dataset
housing = fetch_california_housing()
X = housing.data
y = housing.target

# Standardize features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Split the dataset into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.3, random_state=42)

# Create and train Lasso model
lasso = Lasso(alpha=0.1)
lasso.fit(X_train, y_train)

# Create and train Ridge model
ridge = Ridge(alpha=1.0)
ridge.fit(X_train, y_train)

# Predict with both models
y_pred_lasso = lasso.predict(X_test)
y_pred_ridge = ridge.predict(X_test)

# Evaluate Lasso model
lasso_mse = mean_squared_error(y_test, y_pred_lasso)
lasso_rmse = np.sqrt(lasso_mse)
lasso_r2 = r2_score(y_test, y_pred_lasso)

# Evaluate Ridge model
ridge_mse = mean_squared_error(y_test, y_pred_ridge)
ridge_rmse = np.sqrt(ridge_mse)
ridge_r2 = r2_score(y_test, y_pred_ridge)

print("Lasso Regression Results:")
```

```
print(f"Alpha: {lasso.alpha}")
print(f"Number of non-zero coefficients: {np.sum(lasso.coef_ != 0)}")
print(f"Mean Squared Error: {lasso_mse}")
print(f"Root Mean Squared Error: {lasso_rmse}")
print(f"R² Score: {lasso_r2}")

print("\nRidge Regression Results:")
print(f"Alpha: {ridge.alpha}")
print(f"Mean Squared Error: {ridge_mse}")
print(f"Root Mean Squared Error: {ridge_rmse}")
print(f"R² Score: {ridge_r2}")

# Perform k-cross-validation for both models
lasso_cv_scores = cross_val_score(lasso, X_scaled, y, cv=10, scoring='neg_mean_squared_error')
lasso_cv_rmse = np.sqrt(-lasso_cv_scores)
ridge_cv_scores = cross_val_score(ridge, X_scaled, y, cv=10, scoring='neg_mean_squared_error')
ridge_cv_rmse = np.sqrt(-ridge_cv_scores)

print(f"\nLasso Cross-Validation Mean RMSE: {np.mean(lasso_cv_rmse)}")
print(f"Ridge Cross-Validation Mean RMSE: {np.mean(ridge_cv_rmse)}")
```

Output

```
Lasso Regression Results:
Alpha: 0.1
Number of non-zero coefficients: 3
Mean Squared Error: 0.6647101868107819
Root Mean Squared Error: 0.8152976062829952
R² Score: 0.4935696190511787

Ridge Regression Results:
Alpha: 1.0
Mean Squared Error: 0.5305421966451029
Root Mean Squared Error: 0.7283832759235367
R² Score: 0.5957897259773937

Lasso Cross-Validation Mean RMSE: 0.8297397498165436
Ridge Cross-Validation Mean RMSE: 0.7307325544939554
```

Logistic Regression

Code

```
import numpy as np
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report, roc_auc_score

# Load the Iris dataset
iris = load_iris()
X = iris.data
```

```python
y = iris.target

# Split the dataset into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Create a Logistic Regression model
model = LogisticRegression(max_iter=1000)
model.fit(X_train, y_train)

# Predict the test set results
y_pred = model.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
class_report = classification_report(y_test, y_pred)
roc_auc = roc_auc_score(y_test, model.predict_proba(X_test), multi_class='ovr')

print("Logistic Regression Results:")
print(f"Accuracy: {accuracy}")
print(f"Confusion Matrix:\n{conf_matrix}")
print(f"Classification Report:\n{class_report}")
print(f"ROC AUC Score: {roc_auc}")

# Perform k-cross-validation
cv_scores = cross_val_score(model, X, y, cv=10)
print(f"Cross-Validation Scores: {cv_scores}")
print(f"Mean CV Score: {np.mean(cv_scores)}")
```

Output

```
Logistic Regression Results:
Accuracy: 1.0
Confusion Matrix:
[[19  0  0]
 [ 0 13  0]
 [ 0  0 13]]
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        19
           1       1.00      1.00      1.00        13
           2       1.00      1.00      1.00        13

    accuracy                           1.00        45
   macro avg       1.00      1.00      1.00        45
weighted avg       1.00      1.00      1.00        45

ROC AUC Score: 1.0
Cross-Validation Scores: [1.         0.93333333 1.         1.         0.93333333 0.93333333
 0.93333333 1.         1.         1.        ]
Mean CV Score: 0.9733333333333334
```

Artificial Neural Network

Code

```
import numpy as np
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report, roc_auc_score
from sklearn.preprocessing import StandardScaler

# Load the Iris dataset
iris = load_iris()
X = iris.data
y = iris.target

# Standardize features for better convergence
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Split the dataset into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.3, random_state=42)

# Create an ANN classifier with increased max_iter and early_stopping
model = MLPClassifier(
    hidden_layer_sizes=(10,),
    max_iter=2000,
    alpha=0.001,
    random_state=42,
    early_stopping=True,
    validation_fraction=0.1,
    solver='adam'
)
model.fit(X_train, y_train)

# Predict the test set results
y_pred = model.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
class_report = classification_report(y_test, y_pred)
roc_auc = roc_auc_score(y_test, model.predict_proba(X_test), multi_class='ovr')

print("Artificial Neural Network Classifier Results:")
print(f"Hidden Layer Sizes: {model.hidden_layer_sizes}")
print(f"Accuracy: {accuracy}")
print(f"Confusion Matrix:\n{conf_matrix}")
print(f"Classification Report:\n{class_report}")
print(f"ROC AUC Score: {roc_auc}")

# Perform k-cross-validation
cv_scores = cross_val_score(model, X_scaled, y, cv=10)
print(f"Cross-Validation Scores: {cv_scores}")
```

```
print(f"Mean CV Score: {np.mean(cv_scores)}")
```

Output

```
Artificial Neural Network Classifier Results:
Hidden Layer Sizes: (10,)
Accuracy: 0.06666666666666667
Confusion Matrix:
[[ 3  9  7]
 [13  0  0]
 [13  0  0]]
Classification Report:
              precision    recall  f1-score   support

           0       0.10      0.16      0.12        19
           1       0.00      0.00      0.00        13
           2       0.00      0.00      0.00        13

    accuracy                           0.07        45
   macro avg       0.03      0.05      0.04        45
weighted avg       0.04      0.07      0.05        45

ROC AUC Score: 0.2752614709851552
Cross-Validation Scores: [0.         0.06666667 0.         0.2        0.         0.
 0.06666667 0.         0.06666667 0.        ]
Mean CV Score: 0.039999999999999994
```

K-NN Classifier

Code

```
import numpy as np
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report, roc_auc_score

# Load the Iris dataset
iris = load_iris()
X = iris.data
y = iris.target

# Split the dataset into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Create a K-NN classifier
k = 5  # Number of neighbors
model = KNeighborsClassifier(n_neighbors=k)
model.fit(X_train, y_train)

# Predict the test set results
y_pred = model.predict(X_test)
```

```
# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
class_report = classification_report(y_test, y_pred)
roc_auc = roc_auc_score(y_test, model.predict_proba(X_test), multi_class='ovr')

print("K-NN Classifier Results:")
print(f"K value: {k}")
print(f"Accuracy: {accuracy}")
print(f"Confusion Matrix:\n{conf_matrix}")
print(f"Classification Report:\n{class_report}")
print(f"ROC AUC Score: {roc_auc}")

# Perform k-cross-validation
cv_scores = cross_val_score(model, X, y, cv=10)
print(f"Cross-Validation Scores: {cv_scores}")
print(f"Mean CV Score: {np.mean(cv_scores)}")
```

Output

```
K-NN Classifier Results:
K value: 5
Accuracy: 1.0
Confusion Matrix:
[[19  0  0]
 [ 0 13  0]
 [ 0  0 13]]
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        19
           1       1.00      1.00      1.00        13
           2       1.00      1.00      1.00        13

    accuracy                           1.00        45
   macro avg       1.00      1.00      1.00        45
weighted avg       1.00      1.00      1.00        45

ROC AUC Score: 1.0
Cross-Validation Scores: [1.         0.93333333 1.         1.         0.86666667 0.93333333
 0.93333333 1.         1.         1.        ]
Mean CV Score: 0.9666666666666668
```

Decision Tree Classification

Code

```
import numpy as np
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.tree import DecisionTreeClassifier
```

```python
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report, roc_auc_score

# Load the Iris dataset
iris = load_iris()
X = iris.data
y = iris.target

# Split the dataset into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Create a Decision Tree classifier
model = DecisionTreeClassifier(random_state=42)
model.fit(X_train, y_train)

# Predict the test set results
y_pred = model.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
class_report = classification_report(y_test, y_pred)
roc_auc = roc_auc_score(y_test, model.predict_proba(X_test), multi_class='ovr')

print("Decision Tree Classifier Results:")
print(f"Accuracy: {accuracy}")
print(f"Confusion Matrix:\n{conf_matrix}")
print(f"Classification Report:\n{class_report}")
print(f"ROC AUC Score: {roc_auc}")
print(f"Feature Importances: {model.feature_importances_}")

# Perform k-cross-validation
cv_scores = cross_val_score(model, X, y, cv=10)
print(f"Cross-Validation Scores: {cv_scores}")
print(f"Mean CV Score: {np.mean(cv_scores)}")
```

Output

```
Decision Tree Classifier Results:
Accuracy: 1.0
Confusion Matrix:
[[19  0  0]
 [ 0 13  0]
 [ 0  0 13]]
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        19
           1       1.00      1.00      1.00        13
           2       1.00      1.00      1.00        13

    accuracy                           1.00        45
   macro avg       1.00      1.00      1.00        45
weighted avg       1.00      1.00      1.00        45
```

```
ROC AUC Score: 1.0
Feature Importances: [0.         0.01911002 0.89326355 0.08762643]
Cross-Validation Scores: [1.         0.93333333 1.         0.93333333 0.93333333 0.86666667
 0.93333333 0.93333333 1.         1.        ]
Mean CV Score: 0.9533333333333334
```

SVM Classification

Code

```
import numpy as np
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report, roc_auc_score

# Load the Iris dataset
iris = load_iris()
X = iris.data
y = iris.target

# Split the dataset into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Create an SVM classifier
model = SVC(kernel='rbf', probability=True, random_state=42)
model.fit(X_train, y_train)

# Predict the test set results
y_pred = model.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
class_report = classification_report(y_test, y_pred)
roc_auc = roc_auc_score(y_test, model.predict_proba(X_test), multi_class='ovr')

print("SVM Classifier Results:")
print(f"Kernel: {model.kernel}")
print(f"Accuracy: {accuracy}")
print(f"Confusion Matrix:\n{conf_matrix}")
print(f"Classification Report:\n{class_report}")
print(f"ROC AUC Score: {roc_auc}")

# Perform k-cross-validation
cv_scores = cross_val_score(model, X, y, cv=10)
print(f"Cross-Validation Scores: {cv_scores}")
print(f"Mean CV Score: {np.mean(cv_scores)}")
```

Output

```
SVM Classifier Results:
```

```
Kernel: rbf
Accuracy: 1.0
Confusion Matrix:
[[19  0  0]
 [ 0 13  0]
 [ 0  0 13]]
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        19
           1       1.00      1.00      1.00        13
           2       1.00      1.00      1.00        13

    accuracy                           1.00        45
   macro avg       1.00      1.00      1.00        45
weighted avg       1.00      1.00      1.00        45

ROC AUC Score: 1.0
Cross-Validation Scores: [1.         0.93333333 1.         1.         1.         0.93333333
 0.93333333 0.93333333 1.         1.        ]
Mean CV Score: 0.9733333333333334
```

K-Means Clustering

Code

```python
import numpy as np
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
from sklearn.preprocessing import StandardScaler

# Load the Iris dataset
iris = load_iris()
X = iris.data
y = iris.target  # Ground truth for comparison, not used in training

# Standardize the data
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Create K-Means model
k = 3  # Number of clusters (known from the dataset)
model = KMeans(n_clusters=k, random_state=42, n_init=10)
model.fit(X_scaled)

# Get cluster assignments
clusters = model.labels_

# Evaluate the model
inertia = model.inertia_
silhouette = silhouette_score(X_scaled, clusters)
```

```
print("K-Means Clustering Results:")
print(f"Number of clusters: {k}")
print(f"Inertia (Sum of squared distances): {inertia}")
print(f"Silhouette Score: {silhouette}")

# Compare with ground truth (only for this dataset since we know the true labels)
from sklearn.metrics import adjusted_rand_score, normalized_mutual_info_score

ari = adjusted_rand_score(y, clusters)
nmi = normalized_mutual_info_score(y, clusters)

print(f"Adjusted Rand Index: {ari}")
print(f"Normalized Mutual Information: {nmi}")

# Count samples in each cluster
unique, counts = np.unique(clusters, return_counts=True)
print("Cluster distribution:")
for i, (cluster, count) in enumerate(zip(unique, counts)):
    print(f"Cluster {cluster}: {count} samples")

# Get cluster centers
centers = model.cluster_centers_
print(f"Cluster Centers:\n{centers}")
```

Output

```
K-Means Clustering Results:
Number of clusters: 3
Inertia (Sum of squared distances): 139.82049635974974
Silhouette Score: 0.45994823920518635
Adjusted Rand Index: 0.6201351808870379
Normalized Mutual Information: 0.659486892724918
Cluster distribution:
Cluster 0: 53 samples
Cluster 1: 50 samples
Cluster 2: 47 samples
Cluster Centers:
[[-0.05021989 -0.88337647  0.34773781  0.2815273 ]
 [-1.01457897  0.85326268 -1.30498732 -1.25489349]
 [ 1.13597027  0.08842168  0.99615451  1.01752612]]
```

Hierarchical Clustering

Code

```
import numpy as np
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.cluster import AgglomerativeClustering
from sklearn.metrics import silhouette_score
from sklearn.preprocessing import StandardScaler
```

```
# Load the Iris dataset
iris = load_iris()
X = iris.data
y = iris.target  # Ground truth for comparison, not used in training

# Standardize the data
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Create Hierarchical Clustering model
n_clusters = 3  # Number of clusters (known from the dataset)
model = AgglomerativeClustering(n_clusters=n_clusters, linkage='ward')
clusters = model.fit_predict(X_scaled)

# Evaluate the model
silhouette = silhouette_score(X_scaled, clusters)

print("Hierarchical Clustering Results:")
print(f"Number of clusters: {n_clusters}")
print(f"Linkage: {model.linkage}")
print(f"Silhouette Score: {silhouette}")

# Compare with ground truth (only for this dataset since we know the true labels)
from sklearn.metrics import adjusted_rand_score, normalized_mutual_info_score

ari = adjusted_rand_score(y, clusters)
nmi = normalized_mutual_info_score(y, clusters)

print(f"Adjusted Rand Index: {ari}")
print(f"Normalized Mutual Information: {nmi}")

# Count samples in each cluster
unique, counts = np.unique(clusters, return_counts=True)
print("Cluster distribution:")
for i, (cluster, count) in enumerate(zip(unique, counts)):
    print(f"Cluster {cluster}: {count} samples")
```

Output

```
Hierarchical Clustering Results:
Number of clusters: 3
Linkage: ward
Silhouette Score: 0.4466890410285909
Adjusted Rand Index: 0.6153229932145449
Normalized Mutual Information: 0.6754701853436886
Cluster distribution:
Cluster 0: 71 samples
Cluster 1: 49 samples
Cluster 2: 30 samples
```
