Machine Learning Code Collection

Hierarchical Clustering

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import scipy.cluster.hierarchy as sch
from sklearn.cluster import AgglomerativeClustering
from sklearn.datasets import load_iris
from sklearn.preprocessing import StandardScaler

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

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

# Check if dataset is properly scaled before dendrogram generation
print("Dataset mean after scaling:", np.mean(X_scaled, axis=0))
print("Dataset std deviation after scaling:", np.std(X_scaled, axis=0))

# Plot Dendrogram
plt.figure(figsize=(10, 5))
linkage_matrix = sch.linkage(X_scaled, method='ward')
sch.dendrogram(linkage_matrix)
plt.title('Dendrogram for Hierarchical Clustering on Iris Dataset')
plt.xlabel('Data Points')
plt.ylabel('Euclidean Distance')
plt.show()

# Determine optimal number of clusters from dendrogram
from scipy.cluster.hierarchy import fcluster
max_d = 7  # Adjust this threshold based on dendrogram visualization
optimal_clusters = len(set(fcluster(linkage_matrix, max_d, criterion='c
print(f"Optimal number of clusters based on dendrogram: {optimal_cluste

# Implement Agglomerative Hierarchical Clustering with optimal clusters
hierarchical = AgglomerativeClustering(n_clusters=optimal_clusters, met
labels = hierarchical.fit_predict(X_scaled)

# Plot the clusters
```

```
plt.scatter(X_scaled[:, 0], X_scaled[:, 1], c=labels, cmap='viridis', m
plt.title(f'Hierarchical Clustering with {optimal_clusters} Clusters or
plt.show()
```

K-Means Clustering

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.datasets import load_iris
from sklearn.preprocessing import StandardScaler

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

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

# Implement K-Means Clustering
k = 3  # Number of clusters (same as the number of Iris classes)
kmeans = KMeans(n_clusters=k, random_state=42)
kmeans.fit(X_scaled)

# Predict cluster labels
labels = kmeans.predict(X_scaled)

# Plot the clusters (using the first two features for 2D visualization)
plt.scatter(X_scaled[:, 0], X_scaled[:, 1], c=labels, cmap='viridis', m
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1
plt.title('K-Means Clustering on Iris Dataset')
plt.legend()
plt.xlabel('Sepal Length (scaled)')
plt.ylabel('Sepal Width (scaled)')
plt.show()
```

Support Vector Machine (SVM)

```
# Import necessary libraries
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
```

```python
from sklearn.metrics import accuracy_score, confusion_matrix, classific
from sklearn.datasets import load_iris

# Loading the Iris dataset
data = load_iris()
X = data.data  # Features
y = data.target  # Labels (multi-class classification)

# Splitting the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3

# Initializing SVM model with RBF kernel (default)
svm_model = SVC(kernel='rbf', C=1.0, gamma='scale')

# Fitting the model on the training data
svm_model.fit(X_train, y_train)

# Making predictions on the test data
y_pred = svm_model.predict(X_test)

# Evaluating the model
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
report = classification_report(y_test, y_pred)

print("Accuracy:", accuracy)
print("Confusion Matrix:\n", conf_matrix)
print("Classification Report:\n", report)

# Example: Predicting for new data
sample_data = np.array([X_test[0]])
prediction = svm_model.predict(sample_data)
print("Prediction:", prediction)
```

Decision Tree Classifier

```python
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score

data = load_iris()
X, y = data.data, data.target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3

model = DecisionTreeClassifier()
```

```
model.fit(X_train, y_train)

y_pred = model.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

K-Nearest Neighbors (KNN)

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

# Generate a sample dataset (Iris dataset)
from sklearn.datasets import load_iris
data = load_iris()
X = data.data
y = data.target

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2

# Normalize features for better performance
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Implement KNN Classifier
k = 5  # Number of neighbors
knn = KNeighborsClassifier(n_neighbors=k)
knn.fit(X_train, y_train)

# Make predictions
y_pred = knn.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f'KNN Classifier Accuracy: {accuracy:.2f}')
```

Artificial Neural Network (ANN)

```
# Import necessary libraries
import numpy as np
```

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, classification_report, conf
from sklearn.datasets import load_wine
from sklearn.neural_network import MLPClassifier

# Loading the Wine dataset
data = load_wine()
X = data.data  # Features
y = data.target  # Labels (multi-class classification)

# Spliting the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2

# Normalizing the data using StandardScaler
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Initializing the MLPClassifier
mlp = MLPClassifier(hidden_layer_sizes=(16, 8), activation='relu', solv

# Training the model
mlp.fit(X_train, y_train)

# Making predictions on the test data
y_pred = mlp.predict(X_test)

# Evaluating the model
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
report = classification_report(y_test, y_pred)

print("Accuracy:", accuracy)
print("Confusion Matrix:\n", conf_matrix)
print("Classification Report:\n", report)

# Predicting for new data
sample_data = np.array([X_test[0]])
prediction = mlp.predict(sample_data)
print("Prediction:", prediction)
```

Logistic Regression

```
# Import necessary libraries
import pandas as pd
```

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix, classific

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

# For binary classification, choose only two classes (e.g., Setosa and
X_binary = X[y != 2]
y_binary = y[y != 2]

# Select only two features (e.g., sepal length and sepal width)
X_binary = X_binary[:, :2]

# Split the data
X_train, X_test, y_train, y_test = train_test_split(X_binary, y_binary,

# Initialize and train the Logistic Regression model
log_reg = LogisticRegression()
log_reg.fit(X_train, y_train)

# Make predictions
y_pred = log_reg.predict(X_test)

# Evaluate the model
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test, y_pred,

# Example: Predict on new data
import numpy as np
sample = np.array([[5.0, 3.5]])  # Sample sepal length and width
prediction = log_reg.predict(sample)
print("Prediction (0 = Setosa, 1 = Versicolor):", prediction)
```

Lasso & Ridge Regression

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_wine
from sklearn.linear_model import Ridge, Lasso
from sklearn.model_selection import train_test_split
```

```
from sklearn.metrics import mean_squared_error

# Load Wine dataset
wine = load_wine()
X = wine.data
y = X[:, 0]  # Let's predict 'Alcohol' (feature 0)
X = np.delete(X, 0, axis=1)  # Remove Alcohol from features

# Split data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2

# Ridge Regression
ridge_model = Ridge(alpha=1.0)
ridge_model.fit(X_train, y_train)
ridge_pred = ridge_model.predict(X_test)
ridge_mse = mean_squared_error(y_test, ridge_pred)
print(f"Ridge Regression MSE: {ridge_mse:.4f}")

# Lasso Regression
lasso_model = Lasso(alpha=1.0, max_iter=10000)
lasso_model.fit(X_train, y_train)
lasso_pred = lasso_model.predict(X_test)
lasso_mse = mean_squared_error(y_test, lasso_pred)
print(f"Lasso Regression MSE: {lasso_mse:.4f}")

# Plotting
plt.scatter(y_test, ridge_pred, color='red', label='Ridge Predictions',
plt.scatter(y_test, lasso_pred, color='green', label='Lasso Predictions
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], 'k--',
plt.xlabel("Actual Alcohol")
plt.ylabel("Predicted Alcohol")
plt.title("Ridge vs Lasso Regression on Wine Dataset")
plt.legend()
plt.grid(True)
plt.show()
```

Polynomial Regression

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_wine
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

# Load the Wine dataset
wine = load_wine()
```

```
X = wine.data[:, [0]]  # Feature: Alcohol
y = wine.data[:, 1]    # Target: Malic Acid
y = y.reshape(-1, 1)

# Transform features to polynomial features
degree = 3
poly_features = PolynomialFeatures(degree=degree)
X_poly = poly_features.fit_transform(X)

# Train the polynomial regression model
model = LinearRegression()
model.fit(X_poly, y)

# Predict values
y_pred = model.predict(X_poly)

# Evaluate model performance
mse = mean_squared_error(y, y_pred)
print(f"Mean Squared Error: {mse:.4f}")

# Plot the results
plt.scatter(X, y, label="Actual Data", alpha=0.7)
# Sort X for a smooth curve
sorted_idx = X[:, 0].argsort()
plt.plot(X[sorted_idx], y_pred[sorted_idx], color='red', label="Polynom
plt.xlabel("Alcohol")
plt.ylabel("Malic Acid")
plt.title("Polynomial Regression on Wine Data")
plt.legend()
plt.show()
```

Linear Regression Models

```
import numpy as np
import pandas as pd
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
from sklearn.datasets import load_wine

# Load Wine Dataset
wine = load_wine()
X = wine.data
Y = wine.target

# Train-Test Split
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2
```

```
# Simple Linear Regression Model (Using first feature for demonstration
simple_model = LinearRegression()
simple_model.fit(X_train[:, [0]], Y_train)  # Using only the first feat
Y_pred_simple = simple_model.predict(X_test[:, [0]])

# Multiple Linear Regression Model
multiple_model = LinearRegression()
multiple_model.fit(X_train, Y_train)
Y_pred_multiple = multiple_model.predict(X_test)

# Evaluation Metrics for Simple Linear Regression
simple_mse = mean_squared_error(Y_test, Y_pred_simple)
simple_r2 = r2_score(Y_test, Y_pred_simple)

# Evaluation Metrics for Multiple Linear Regression
multiple_mse = mean_squared_error(Y_test, Y_pred_multiple)
multiple_r2 = r2_score(Y_test, Y_pred_multiple)

# K-Fold Cross-Validation for Multiple Values of k
k_values = [3, 5, 7, 10]
simple_cv_results = {}
multiple_cv_results = {}

for k in k_values:
    simple_cv_scores = cross_val_score(simple_model, X[:, [0]], Y, cv=k
    multiple_cv_scores = cross_val_score(multiple_model, X, Y, cv=k, sc

    simple_cv_results[k] = np.mean(simple_cv_scores)
    multiple_cv_results[k] = np.mean(multiple_cv_scores)

# Display Results
print("\nSimple Linear Regression:")
print(f"MSE: {simple_mse}")
print(f"R² Score: {simple_r2}")
for k, score in simple_cv_results.items():
    print(f"{k}-Fold CV Average R² Score: {score}")

print("\nMultiple Linear Regression:")
print(f"MSE: {multiple_mse}")
print(f"R² Score: {multiple_r2}")
for k, score in multiple_cv_results.items():
    print(f"{k}-Fold CV Average R² Score: {score}")
```

Naive Bayes Classifier

```python
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, classification_report
from sklearn.datasets import load_iris

# Load dataset (Iris dataset as an example)
data = load_iris()
X, y = data.data, data.target

# Split into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2

# Initialize and train Naïve Bayes Classifier
nb_classifier = GaussianNB()
nb_classifier.fit(X_train, y_train)

# Make predictions
y_pred = nb_classifier.predict(X_test)

# Evaluate model
accuracy = accuracy_score(y_test, y_pred)
report = classification_report(y_test, y_pred)

print(f"Accuracy: {accuracy:.2f}")
print("Classification Report:\n", report)
```