21162101012_CBA_Yash_Lakhtariya

ML Practical 8

For Multiclass classification on Drug prediction dataset; Binary classification on Credit card fraud detection dataset design :

1. Implementing Kmeans, logistic regression, knn, decision tree and ANN
2. Calculate Precision, recall, accuracy etc

Regression: Metro Interstate Traffic Volume Dataset :

1. Use multi linear regression, polynomial regression and ANN
2. Calculate MSE, R2 score etc

Drug Prediction Data

```
!wget -q -O -
!unzip -o -q -d /content/drugs
import pandas as pd
import numpy as np
from matplotlib.pyplot import plot, show, savefig
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix
from sklearn.metrics import classification_report
data = pd.read_csv("/content/drugs/drug200.csv", header=None, sep=",")
print("Data Head:\n", data.head(10))
```

```
Data Head:
   Age  Sex  ...  Na_to_K   Drug
0   23  F  ...    25.355  DrugY
1   47  M  ...    13.093  drugC
2   47  M  ...    10.114  drugC
3   28  F  ...    7.798  drugX
4   61  F  ...    18.043  DrugY
5   22  F  ...    8.607  drugX
6   49  F  ...    16.275  DrugY
7   41  M  ...    11.037  drugC
8   60  M  ...    15.171  DrugY
9   43  M  ...    19.368  DrugY
```

```
print("Dataset Information:")
data.info()
header_provided = (pd.isnull(data[0]).sum() == 0)
print("Header provided:", "Yes" if header_provided else "No")
```

```
Dataset Information:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200 entries, 0 to 199
Data columns (total 6):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Age     200 non-null    int64
 1   Sex     200 non-null    object
 2   BP      200 non-null    object
 3   Cholest 200 non-null    object
 4   Na_to_K 200 non-null    float64
 5   Drug    200 non-null    object
dtypes: float64(1), int64(1), object(4)
memory usage: 9.5 KB
Header provided: No
```

```
Data preprocessing complete!
Training set size: (160, 5) Testing set size: (40, 5)
```

```
print("Data preprocessing done!") #standardize
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

print("Feature scaling applied for improved model convergence.")
```

```
Feature scaling applied for improved model convergence.
```

```
print("\n Applying SMOTE to balance the dataset.")
```

```
 Applying SMOTE to balance the dataset.
```

```
# Use SMOTE to oversample the minority classes
from imblearn.over_sampling import SMOTE
smote = SMOTE(random_state=42, k_neighbors=3)
X_train_balanced, y_train_balanced = smote.fit_resample(X_train_scaled, y_train)

print("Applied SMOTE to balance the dataset.")
```

```
Applied SMOTE to balance the dataset.
```

```
[1] models = {}
models['Logistic Regression'] = LogisticRegression(random_state=42, max_iter=1000, tuning_evaluation)
models['K-Nearest Neighbors'] = KNeighborsClassifier()
models['Decision Tree'] = DecisionTreeClassifier(random_state=42)
models['Artificial Neural Network'] = MLPClassifier(hidden_layer_sizes=(10,), max_iter=1000, random_state=42), early_stopping=True, learning_rate_init=0.001)

metrics = {}

# Train and evaluate models
for model_name, model in models.items():
    print("\n Training {model_name}...")
    # Train the model
    model.fit(X_train_balanced, y_train_balanced)

    # Predictions
    y_pred = model.predict(X_test_scaled)

    # Calculate metrics
    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred, average='weighted', zero_division=0)
    recall = recall_score(y_test, y_pred, average='weighted')
    f1 = f1_score(y_test, y_pred, average='weighted')

    # Store metrics in dict
    metrics[model_name] = {
        'Accuracy': accuracy,
        'Precision': precision,
        'Recall': recall,
        'F1_Score': f1
    }

    print("Training completed!")
```

```
[1] from tabulate import tabulate

# Create comparison table
print("\n" + "="*80)
print("COMPREHENSIVE MODEL COMPARISON")
print("="*80)
comparison_df = pd.DataFrame(metrics).T
print(comparison_df.round(4))

comparison_table = [
    ['Model', 'Accuracy', 'Precision', 'Recall', 'F1_Score'],
    ['Logistic Regression', 0.7417, 0.7368, 0.7417, 0.7282],
    ['K-Nearest Neighbors', 0.6667, 0.6667, 0.6667, 0.6667],
    ['Decision Tree', 0.9667, 0.9667, 0.9667, 0.9667],
    ['Artificial Neural Network', 0.4833, 0.4167, 0.4833, 0.3882]
]

print("\n" + tabulate(comparison_table, tablefmt="grid"))
```

```
[1] # Find best model
best_model = comparison_df['F1_Score'].idxmax()
print("\n Best model: {best_model}")

# Detailed report for best model
if best_model == 'Decision Tree':
    best_model_obj = models['Decision Tree']
    y_pred = best_model_obj.predict(X_test_scaled)
    print("\n Detailed report for {best_model}:")
    print(classification_report(y_test, y_pred))
```

```
[1] # Confusion matrix for best model
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
print("\n Confusion matrix:")
print(cm)
```

```
[1] # Feature importance (if applicable)
if best_model == 'Random Forest':
    feature_importance = best_model_obj.feature_importances_
    feature_names = ['Age', 'Sex', 'BP', 'Cholest', 'Na_to_K']
    for name, importance in zip(feature_names, feature_importance):
        print("{name}: {importance:.4f}")
```

```
[1] # Summary statistics
print("\n Dataset Summary:")
print("Total samples: {len(data)}")
print("Training samples: {len(X_train)}")
print("Test samples: {len(X_test)}")
print("Class distribution:")
print(y_test.value_counts())
```

```
[1] # Save results to CSV
results_df.to_csv('multi_class_results.csv')
print("\n Results saved to multi_class_results.csv")
```

```
[1] # Download the results
from google.colab import files
files.download('multi_class_results.csv')
```

Credit Card Fraud Detection Data

```
[1] !wget -q -O -
!unzip -o -q -d /content/fraud
import numpy as np
import pandas as pd
from matplotlib.pyplot import plot, show, savefig
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from imblearn.over_sampling import SMOTE
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix
from sklearn.metrics import classification_report
```

```
[1] # Load the fraud detection data
print("Loading fraud detection data...")
data = pd.read_csv("/content/fraud/creditcard.csv")
print("Dataset shape:", data.shape)
print("Fraud cases:", data[data['Class'] == 1].shape[0])
print("Normal cases:", data[data['Class'] == 0].shape[0])
print("Fraud percentage:", data['Class'].mean() * 100, "%")
```

```
Loading fraud detection data...
Dataset shape: (284807, 31)
Fraud cases: 492
Normal cases: 284315
Fraud percentage: 0.1727485630620034 %
```

```
[1] # Check data structure
print("\n Data Head:")
print(data.head(10))
print("\n Data Info:")
data.info()
print("\n Data Statistics:")
print(data.describe())
```

```
 Data Head:
   Time        V1  ...  Amount  Class
0     0 -1.359807  ...  149.62      0
1     0  1.191857  ...    2.69      0
2     1 -1.358354  ...  378.66      0
3     1 -0.966272  ...  123.50      0
4     2 -1.158233  ...   69.99      0
5     2  0.877737  ...    3.67      0
6     4  1.644269  ...    4.67      0
7     7 -0.894286  ...   90.00      0
8     7 -0.338262  ...    5.00      0
9     9 -0.733789  ...    2.89      0

[10 rows x 31 columns]

 Data Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 284807 entries, 0 to 284807
Data columns (total 31 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Time    284807 non-null float64
 1   V1      284807 non-null float64
 2   V2      284807 non-null float64
 3   V3      284807 non-null float64
 4   V4      284807 non-null float64
 5   V5      284807 non-null float64
 6   V6      284807 non-null float64
 7   V7      284807 non-null float64
 8   V8      284807 non-null float64
 9   V9      284807 non-null float64
 10  V10     284807 non-null float64
 11  V11     284807 non-null float64
 12  V12     284807 non-null float64
 13  V13     284807 non-null float64
 14  V14     284807 non-null float64
 15  V15     284807 non-null float64
 16  V16     284807 non-null float64
 17  V17     284807 non-null float64
 18  V18     284807 non-null float64
 19  V19     284807 non-null float64
 20  V20     284807 non-null float64
 21  V21     284807 non-null float64
 22  V22     284807 non-null float64
 23  V23     284807 non-null float64
 24  V24     284807 non-null float64
 25  V25     284807 non-null float64
 26  V26     284807 non-null float64
 27  V27     284807 non-null float64
 28  V28     284807 non-null float64
 29  Amount  284807 non-null float64
 30  Class   284807 non-null int64
dtypes: float64(30), int64(1)
memory usage: 67.3 MB

 Data Statistics:
```

```
[1] # Prepare features and target
print("\n Preparing features and target...")
X = data.drop('Class', axis=1)
y = data['Class']

print("Feature shape:", X.shape)
print("Target shape:", y.shape)
print("Class distribution:")
print(y.value_counts())
```

```
Preparing features and target...
Feature shape: (284807, 30)
Target shape: (284807,)
Class distribution:
Class
0    284315
1       492
Name: count, dtype: int64
```

```
[1] # Split data
print("\n Splitting data...")
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)

print("Training set: {X_train.shape}")
print("Test set: {X_test.shape}")
print("Training fraud cases: {y_train[y_train == 1].shape[0]}")
print("Test fraud cases: {y_test[y_test == 1].shape[0]}")
```

```
Splitting data...
Training set: (227845, 30)
Test set: (56962, 30)
Training fraud cases: 394
Test fraud cases: 98
```

```
[1] # Standardize features
print("\n Standardizing features...")
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

print("Feature scaling applied.")
```

```
Standardizing features...
Feature scaling applied.
```

```
[1] # Apply SMOTE to balance the training data
print("\n Applying SMOTE to balance the training data...")
smote = SMOTE(random_state=42, k_neighbors=3)
X_train_balanced, y_train_balanced = smote.fit_resample(X_train_scaled, y_train)

print("SMOTE applied.")
print("Balanced training set: {X_train_balanced.shape}")
print("Balanced fraud cases: {y_train_balanced[y_train_balanced == 1].shape[0]}")
```

```
Applying SMOTE to balance the training data...
SMOTE applied.
Balanced training set: (454904, 30)
Balanced fraud cases: 227452
```

```
[1] # Initialize models
models = {}
models['Logistic Regression'] = LogisticRegression(random_state=42, max_iter=1000)
models['K-Nearest Neighbors'] = KNeighborsClassifier()
models['Decision Tree'] = DecisionTreeClassifier(random_state=42)
models['Artificial Neural Network'] = MLPClassifier(hidden_layer_sizes=(10,), max_iter=1000, random_state=42)

metrics = {}

# Train and evaluate models
for model_name, model in models.items():
    print("\n Training {model_name}...")
    # Train the model
    model.fit(X_train_balanced, y_train_balanced)

    # Predictions
    y_pred = model.predict(X_test_scaled)

    # Calculate metrics
    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred, average='weighted', zero_division=0)
    recall = recall_score(y_test, y_pred, average='weighted')
    f1 = f1_score(y_test, y_pred, average='weighted')

    # Store metrics in dict
    metrics[model_name] = {
        'Accuracy': accuracy,
        'Precision': precision,
        'Recall': recall,
        'F1_Score': f1
    }

    print("Training completed!")
```

```
Training Logistic Regression...
Training completed!

Training K-Nearest Neighbors...
Training completed!

Training Decision Tree...
Training completed!

Training Artificial Neural Network...
Training completed!
```

```
[1] from tabulate import tabulate

# Create comparison table
print("\n" + "="*80)
print("COMPREHENSIVE MODEL COMPARISON")
print("="*80)
comparison_df = pd.DataFrame(metrics).T
print(comparison_df.round(4))

comparison_table = [
    ['Model', 'Accuracy', 'Precision', 'Recall', 'F1_Score'],
    ['Logistic Regression', 0.9727, 0.9738, 0.9727, 0.9782],
    ['K-Nearest Neighbors', 0.9516, 0.9976, 0.9516, 0.9741],
    ['Decision Tree', 0.9478, 0.9982, 0.9478, 0.9721],
    ['Artificial Neural Network', 0.9605, 0.9977, 0.9605, 0.9780]
]

print("\n" + tabulate(comparison_table, tablefmt="grid"))
```

```
[1] # Find best model
best_model = comparison_df['F1_Score'].idxmax()
print("\n Best model: {best_model}")

# Detailed report for best model
if best_model == 'Decision Tree':
    best_model_obj = models['Decision Tree']
    y_pred = best_model_obj.predict(X_test_scaled)
    print("\n Detailed report for {best_model}:")
    print(classification_report(y_test, y_pred))
```

```
[1] # Confusion matrix for best model
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
print("\n Confusion matrix:")
print(cm)
```

```
[1] # Feature importance (if applicable)
if best_model == 'Random Forest':
    feature_importance = best_model_obj.feature_importances_
    feature_names = ['Age', 'Sex', 'BP', 'Cholest', 'Na_to_K']
    for name, importance in zip(feature_names, feature_importance):
        print("{name}: {importance:.4f}")
```

```
[1] # Summary statistics
print("\n Dataset Summary:")
print("Total samples: {len(data)}")
print("Training samples: {len(X_train)}")
print("Test samples: {len(X_test)}")
print("Class distribution:")
print(y_test.value_counts)
```

```
[1] # Save results to CSV
results_df.to_csv('multi_class_results.csv')
print("\n Results saved to multi_class_results.csv")
```

```
[1] # Download the results
from google.colab import files
files.download('multi_class_results.csv')
```

Metro Interstate Traffic Volume Data

```
[1] !wget -q -O -
!unzip -o -q -d /content/metro
import numpy as np
import pandas as pd
from matplotlib.pyplot import plot, show, savefig
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.neighbors import KNeighborsRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.neural_network import MLPRegressor
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
```

```
[1] data = pd.read_csv("/content/metro/Metro_Interstate_Traffic_Volume.csv")
print("Loading Metro Interstate Traffic Volume data...")
print("Dataset shape:", data.shape)
print("Columns:", data.columns.tolist)
print("Data types:")
print(data.dtypes)
```

```
Loading Metro Interstate Traffic Volume data...
Dataset shape: (48204, 9)
Columns: ['holiday', 'temp', 'rain_1h', 'snow_1h', 'clouds_all', 'weather_main', 'weather_description', 'date_time', 'traffic_volume']
Data types:
holiday              object
temp                 float64
rain_1h              float64
snow_1h              float64
clouds_all           int64
weather_main         object
weather_description  object
date_time            object
traffic_volume       int64
dtype: object
```

```
[1] print("\n Data Preprocessing...")
# Handle missing values
print("Missing values before:")
print(data.isnull().sum())

# Fill missing values with mean for numerical columns
numerical_cols = data.select_dtypes(include=[np.number])
for col in numerical_cols:
    data[col] = data[col].fillna(data[col].mean())

print("Missing values after:")
print(data.isnull().sum())
```

```
Data Preprocessing...
Missing values before:
holiday              0
temp                 0
rain_1h              0
snow_1h              0
clouds_all           0
weather_main         0
weather_description  0
date_time            0
traffic_volume       0
dtype: int64
Missing values after:
holiday              0
temp                 0
rain_1h              0
snow_1h              0
clouds_all           0
weather_main         0
weather_description  0
date_time            0
traffic_volume       0
dtype: int64
```

```
[1] # Create target variable (traffic volume)
target = 'traffic_volume'

print("\n Preparing features...")
X = data.drop(['date_time', 'weather_main', 'weather_description'], axis=1)
y = data[target]

print("Feature shape:", X.shape)
print("Target shape:", y.shape)
```

```
Preparing features...
Feature shape: (48204, 6)
Target shape: (48204,)
```

```
[1] # Split data
print("\n Splitting data...")
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

print("Training set: {X_train.shape}")
print("Test set: {X_test.shape}")
```

```
Splitting data...
Training set: (38563, 6)
Test set: (9641, 6)
```

```
[1] # Standardize features
print("\n Standardizing features...")
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

print("Feature scaling applied.")
```

```
Standardizing features...
Feature scaling applied.
```

```
[1] # Initialize models
models = {}
models['Linear Regression'] = LinearRegression()
models['K-Nearest Neighbors'] = KNeighborsRegressor()
models['Decision Tree'] = DecisionTreeRegressor(random_state=42)
models['Artificial Neural Network'] = MLPRegressor(hidden_layer_sizes=(10,), max_iter=1000, random_state=42)

metrics = {}

# Train and evaluate models
for model_name, model in models.items():
    print("\n Training {model_name}...")
    # Train the model
    model.fit(X_train_scaled, y_train)

    # Predictions
    y_pred = model.predict(X_test_scaled)

    # Calculate metrics
    mse = mean_squared_error(y_test, y_pred)
    mae = mean_absolute_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)

    # Store metrics in dict
    metrics[model_name] = {
        'MSE': mse,
        'MAE': mae,
        'R2_Score': r2
    }

    print("Training completed!")
```

```
Training Linear Regression...
Training completed!

Training K-Nearest Neighbors...
Training completed!

Training Decision Tree...
Training completed!

Training Artificial Neural Network...
Training completed!
```

```
[1] from tabulate import tabulate

# Create comparison table
print("\n" + "="*80)
print("COMPREHENSIVE MODEL COMPARISON")
print="="*80)
comparison_df = pd.DataFrame(metrics).T
print(comparison_df.round(4))

comparison_table = [
    ['Model', 'MSE', 'MAE', 'R2_Score'],
    ['Linear Regression', 0.0000, 0.0000, 1.0000],
    ['K-Nearest Neighbors', 0.0000, 0.0000, 1.0000],
    ['Decision Tree', 0.0000, 0.0000, 1.0000],
    ['Artificial Neural Network', 2.0220e+06, 1.0220e+03, 0.5020]
]

print("\n" + tabulate(comparison_table, tablefmt="grid"))
```

```
[1] # Find best model
best_model = comparison_df['R2_Score'].idxmax()
print("\n Best model: {best_model}")

# Detailed report for best model
if best_model == 'Linear Regression':
    best_model_obj = models['Linear Regression']
    y_pred = best_model_obj.predict(X_test_scaled)
    print("\n Detailed report for {best_model}:")
    print("MSE: {mean_squared_error(y_test, y_pred):.4f}")
    print("MAE: {mean_absolute_error(y_test, y_pred):.4f}")
    print("R2 Score: {r2_score(y_test, y_pred):.4f}")
```

```
[1] # Feature importance (if applicable)
if best_model == 'Random Forest':
    feature_importance = best_model_obj.feature_importances_
    feature_names = ['holiday', 'temp', 'rain_1h', 'snow_1h', 'clouds_all', 'weather_main']
    for name, importance in zip(feature_names, feature_importance):
        print("{name}: {importance:.4f}")
```

```
[1] # Summary statistics
print("\n Dataset Summary:")
print("Total samples: {len(data)}")
print("Training samples: {len(X_train)}")
print("Test samples: {len(X_test)}")
print("Target statistics:")
print(y_test.describe())
```

```
[1] # Save results to CSV
results_df.to_csv('metro_results.csv')
print("\n Results saved to metro_results.csv")
```

```
[1] # Download the results
from google.colab import files
files.download('metro_results.csv')
```