Week 5 Hands-On
Activity-alzheimers_disease_data_new

Week 5 Hands-On Activity

1. **Dataset Preparation**

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import pandas as pd

# Ensure numpy is imported
import numpy as np

# Load dataset
df = pd.read_csv(r'C:\Users\Saba\Documents\Semester - 04\Itauma\Directories\Machine_Learning'

print(df.columns)
print(df.dtypes)
```

```
Index(['PatientID', 'Age', 'Gender', 'Ethnicity', 'EducationLevel', 'BMI',
       'Smoking', 'AlcoholConsumption', 'PhysicalActivity', 'DietQuality',
       'SleepQuality', 'FamilyHistoryAlzheimers', 'CardiovascularDisease',
       'Diabetes', 'Depression', 'HeadInjury', 'Hypertension', 'SystolicBP',
       'DiastolicBP', 'CholesterolTotal', 'CholesterolLDL', 'CholesterolHDL',
       'CholesterolTriglycerides', 'MMSE', 'FunctionalAssessment',
       'MemoryComplaints', 'BehavioralProblems', 'ADL', 'Confusion',
       'Disorientation', 'PersonalityChanges', 'DifficultyCompletingTasks',
       'Forgetfulness', 'Diagnosis', 'DoctorInCharge'],
      dtype='object')
PatientID                    int64
Age                          int64
```

```
Gender                          int64
Ethnicity                       int64
EducationLevel                  int64
BMI                           float64
Smoking                         int64
AlcoholConsumption            float64
PhysicalActivity              float64
DietQuality                   float64
SleepQuality                  float64
FamilyHistoryAlzheimers         int64
CardiovascularDisease           int64
Diabetes                        int64
Depression                      int64
HeadInjury                      int64
Hypertension                    int64
SystolicBP                      int64
DiastolicBP                     int64
CholesterolTotal              float64
CholesterolLDL                float64
CholesterolHDL                float64
CholesterolTriglycerides      float64
MMSE                          float64
FunctionalAssessment          float64
MemoryComplaints                int64
BehavioralProblems              int64
ADL                           float64
Confusion                       int64
Disorientation                  int64
PersonalityChanges              int64
DifficultyCompletingTasks       int64
Forgetfulness                   int64
Diagnosis                       int64
DoctorInCharge                 object
dtype: object
```

```python
from sklearn.preprocessing import LabelEncoder
import pandas as pd

# Sample DataFrame for demonstration
data = {'DoctorInCharge': ['Dr. Smith', 'Dr. Jones', 'Dr. Lee']}
X = pd.DataFrame(data)
non_numeric_cols = ['DoctorInCharge']  # Example list of columns to encode
```

```
label_encoder = LabelEncoder()

for col in non_numeric_cols:
    if col in X.columns:
        X[col] = label_encoder.fit_transform(X[col])
    else:
        print(f"Column '{col}' not found in DataFrame")
print(X)

X.columns = X.columns.str.strip()  # Remove any leading/trailing spaces
```

```
   DoctorInCharge
0               2
1               0
2               1
```

```
X = df.drop('Diagnosis', axis=1)
y = df['Diagnosis']
```

```
# Split dataset into features (X) and target (y)
X = df.drop('Diagnosis', axis=1)
y = df['Diagnosis']

# Check for non-numeric columns
non_numeric_cols = X.select_dtypes(include=['object']).columns
print(non_numeric_cols)

# One-hot encoding for categorical columns (if needed)
X = pd.get_dummies(X, drop_first=True)

# Split into train and test sets
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Feature scaling (if needed for algorithms like SVM)
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

```
Index(['DoctorInCharge'], dtype='object')
```

```
# Check rows with problematic values like 'XXXConfid'
invalid_rows = X[X.apply(lambda row: row.astype(str).str.contains('XXXConfid').any(), axis=1)
print(invalid_rows)

# Remove or fix invalid rows
X = X[~X.apply(lambda row: row.astype(str).str.contains('XXXConfid').any(), axis=1)]
```

```
Empty DataFrame
Columns: [PatientID, Age, Gender, Ethnicity, EducationLevel, BMI, Smoking, AlcoholConsumption
Index: []

[0 rows x 33 columns]
```

```
# Feature scaling after cleaning the data
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

2. **Model Implementation**

```
from sklearn.svm import SVC
from sklearn.ensemble import GradientBoostingClassifier, RandomForestClassifier

# Define models
svm = SVC()
gbm = GradientBoostingClassifier()
rf = RandomForestClassifier()

# Train the models
svm.fit(X_train_scaled, y_train)
gbm.fit(X_train, y_train)
rf.fit(X_train, y_train)
```

```
RandomForestClassifier()
```

3. **Hyperparameter Tuning**

```
from sklearn.model_selection import GridSearchCV

# Example: Hyperparameter tuning for Random Forest
param_grid_rf = {
    'n_estimators': [50, 100, 200],
    'max_depth': [10, 20, 30],
```

```
    'min_samples_split': [2, 5, 10]
}

grid_rf = GridSearchCV(RandomForestClassifier(), param_grid_rf, cv=5, scoring='accuracy'
grid_rf.fit(X_train, y_train)

# Best parameters
print(f"Best parameters for Random Forest: {grid_rf.best_params_}")
```

```
Best parameters for Random Forest: {'max_depth': 30, 'min_samples_split': 2, 'n_estimato
```

4. **Model Evaluation**

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc

# Predict on test data
y_pred_svm = svm.predict(X_test_scaled)
y_pred_gbm = gbm.predict(X_test)
y_pred_rf = rf.predict(X_test)
```

```
# Calculate evaluation metrics
def evaluate_model(y_true, y_pred):
    accuracy = accuracy_score(y_true, y_pred)
    precision = precision_score(y_true, y_pred)
    recall = recall_score(y_true, y_pred)
    f1 = f1_score(y_true, y_pred)
    return accuracy, precision, recall, f1
```

```
# Evaluate each model
auc_svm, prec_gbm, rec_gbm, f1_gbm = evaluate_model(y_test, y_pred_svm)
acc_gbm, prec_gbm, rec_gbm, f1_gbm = evaluate_model(y_test, y_pred_gbm)
acc_rf, prec_rf, rec_rf, f1_rf = evaluate_model(y_test, y_pred_rf)
```

```
# AUC-ROC
auc_svm = roc_auc_score(y_test, svm.decision_function(X_test_scaled))
auc_gbm = roc_auc_score(y_test, gbm.predict_proba(X_test)[:, 1])
auc_rf = roc_auc_score(y_test, rf.predict_proba(X_test)[:, 1])
```

```
print(f"SVM Accuracy: {auc_svm}, Precision: {prec_gbm}, Recall: {rec_gbm}, F1: {f1_gbm},
print(f"GBM Accuracy: {acc_gbm}, Precision: {prec_gbm}, Recall: {rec_gbm}, F1: {f1_gbm},
print(f"RF Accuracy: {acc_rf}, Precision: {prec_rf}, Recall: {rec_rf}, F1: {f1_rf}, AUC-
```

```
SVM Accuracy: 0.901913593355513, Precision: 0.9470198675496688, Recall: 0.93464052287581
GBM Accuracy: 0.958139534883721, Precision: 0.9470198675496688, Recall: 0.93464052287581
```

```
RF Accuracy: 0.9302325581395349, Precision: 0.9624060150375939, Recall: 0.83660130718954
```

5. **Model Comparison & Reflection**

```python
import pandas as pd

# Model performance data
data = {
    'Model': ['SVM', 'GBM', 'Random Forest'],
    'Accuracy': [0.9019, 0.9581, 0.9279],
    'Precision': [0.9470, 0.9470, 0.9621],
    'Recall': [0.9346, 0.9346, 0.8301],
    'F1-Score': [0.9408, 0.9408, 0.8912],
    'AUC-ROC': [0.9019, 0.9871, 0.9839]
}

# Create DataFrame
df = pd.DataFrame(data)

# Display the DataFrame
print(df)
```

```
           Model  Accuracy  Precision  Recall  F1-Score  AUC-ROC
0            SVM    0.9019     0.9470  0.9346    0.9408   0.9019
1            GBM    0.9581     0.9470  0.9346    0.9408   0.9871
2  Random Forest    0.9279     0.9621  0.8301    0.8912   0.9839
```