Lab 05: Treating a Classification Problem as a Machine Learning Problem

Overview

This lab introduces how to treat a real-world classification problem as a machine learning task. Using the **Titanic Passenger Survival Prediction** case study, students will learn how to load data, preprocess it, build models, evaluate their performance, and save the trained models for future use.

Objectives

- Understand how to approach a classification problem as a machine learning task.
- Perform data preprocessing and feature engineering.
- Train and compare multiple classification algorithms.
- Evaluate model performance using standard metrics.
- Save the final trained model using `joblib`.

Required Libraries

```
pip install numpy pandas matplotlib seaborn scikit-learn joblib
```

Import Required Libraries

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix,
classification_report, roc_auc_score, roc_curve
import joblib
```

Purpose: Import all necessary Python libraries used for data manipulation, visualization, model training, evaluation, and persistence.

Step 1: Data Loading and Exploration

```
df = pd.read_csv('titanic.csv')
print(df.shape)
df.head()
```

Purpose: Load the Titanic dataset and inspect its structure to understand what kind of data is available.

Exploratory Data Analysis (EDA)

```
print(df.info())
print(df.isnull().sum())
sns.countplot(x='Survived', data=df)
plt.title('Survival Counts')
plt.show()
```

Purpose: Identify missing values, data types, and target distribution.

Step 2: Data Preprocessing & Feature Engineering

```
df['FamilySize'] = df['SibSp'] + df['Parch'] + 1
df['IsAlone'] = (df['FamilySize'] == 1).astype(int)

# Fill missing values
df['Age'].fillna(df['Age'].median(), inplace=True)
df['Embarked'].fillna(df['Embarked'].mode()[0], inplace=True)

# Drop irrelevant columns
df.drop(columns=['PassengerId','Name','Ticket','Cabin'], inplace=True)

# Encode categorical variables
df = pd.get_dummies(df, columns=['Sex','Embarked'], drop_first=True)
```

Purpose: Clean and transform data into numerical form suitable for machine learning models.

Step 3: Splitting and Scaling Data

```
X = df.drop(columns=['Survived'])
y = df['Survived']

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42, stratify=y)

scaler = StandardScaler()
num_cols = ['Age','Fare','FamilySize']
X_train[num_cols] = scaler.fit_transform(X_train[num_cols])
X_test[num_cols] = scaler.transform(X_test[num_cols])
```

Purpose: Split dataset into training and testing subsets and normalize numerical features.

Step 4: Model Training and Evaluation

```
models = {
    'LogisticRegression': LogisticRegression(max_iter=500),
    'DecisionTree': DecisionTreeClassifier(random_state=42),
    'RandomForest': RandomForestClassifier(random_state=42)
}

results = {}
for name, model in models.items():
    model.fit(X_train, y_train)
    preds = model.predict(X_test)
    acc = accuracy_score(y_test, preds)
    results[name] = acc
    print(name)
    print('Accuracy:', acc)
    print(confusion_matrix(y_test, preds))
    print(classification_report(y_test, preds))
```

Purpose: Train multiple classification models and compare their performance.

Step 5: Hyperparameter Tuning (Random Forest)

```
param_grid = {
    'n_estimators': [50, 100],
    'max_depth': [None, 5, 10]
}
rf = RandomForestClassifier(random_state=42)
gs = GridSearchCV(rf, param_grid, cv=5, scoring='accuracy')
gs.fit(X_train, y_train)

print('Best Parameters:', gs.best_params_)
```

Purpose: Find the best-performing hyperparameters for the Random Forest model using grid search.

Step 6: Model Persistence

```
joblib.dump(gs.best_estimator_, 'titanic_best_model.joblib')
print('Model saved successfully!')
```

Purpose: Save the trained model for later use or deployment.

Exercises

1. Compare model performance when you include/exclude certain features.
2. Visualize the ROC curve for your best-performing model.
3. Use cross-validation (`cross_val_score`) to assess model stability.
4. Try additional models such as `KNeighborsClassifier` or `SVC`.
5. Write a short summary report comparing all model performances.

Deliverables

- Completed notebook implementing all lab steps.
- A short report summarizing EDA findings, model selection, and evaluation.
- Answers to the exercises provided above.

References

- Scikit-learn Documentation: https://scikit-learn.org/stable/
- Kaggle Titanic Dataset: https://www.kaggle.com/c/titanic