DocAssist (Building Intelligent Medical Decision Support System)

Welcome to Medical Decision Support System Notebook!

Greeting and welcome to this comprehensive notebook that walks you through the preprocessing, training, evaluation, etc. of our Intelligent Medical model.

Overview

This code represents a comprehensive workflow for training and evaluating a machine learning model using Amazon SageMaker. The aim is to demonstrate a structured approach towards building and assessing predictive models, particularly in the context of healthcare data analysis. Here's an overview of the tasks covered in this workflow:

1. **Data Preprocessing**: The initial step involves preparing the data for analysis by cleaning it, performing feature engineering, and handling missing values. This ensures that the data is in a suitable format for model training.
2. **Model Training and Evaluation**: We utilize the Random Forest Classifier as our predictive model and explore different hyperparameters, such as the number of estimators (`n_estimators`). The model is trained on a portion of the data and evaluated on separate validation and test sets to assess its performance.
3. **Data Saving and Uploading to S3**: Once the model is trained and evaluated, the datasets are saved as CSV files and uploaded to Amazon S3 for efficient storage and future access. This facilitates seamless integration with other AWS services and enables scalability.
4. **Prediction and Displaying Results**: Finally, the trained model is used to make predictions on unseen data, and the results are displayed, including predicted labels and probability estimates. This step provides insights into the model's predictive capabilities and confidence levels.

Throughout this workflow, we aim to illustrate best practices in machine learning model development and evaluation, emphasizing the importance of data preprocessing, model tuning, and result interpretation in achieving accurate and reliable predictions.

Dependencies and Imports

```
pip install --upgrade pandas
pip install --upgrade fsspec
pip install --upgrade xgboost
pip install -U sagemaker
pip install tornado==6.4
pip install --upgrade typing-extensions
```

```
pip install scikit-learn
pip install pydantic
pip install pydantic-settings
pip install matplotlib reportlab
pip install --upgrade scikit-learn

import pandas as pd
import numpy as np
import boto3
import sagemaker
import json
import joblib
from sagemaker.xgboost.estimator import XGBoost
from sagemaker.tuner import (
    IntegerParameter,
    ContinuousParameter,
    HyperparameterTuner
)
from sagemaker.inputs import TrainingInput
from sagemaker.image_uris import retrieve
from sagemaker.serializers import CSVSerializer
from sagemaker.deserializers import CSVDeserializer

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, roc_auc_score,
roc_curve, auc
import matplotlib.pyplot as plt
import seaborn as sns

sagemaker.config INFO - Not applying SDK defaults from location:
/etc/xdg/sagemaker/config.yaml
sagemaker.config INFO - Not applying SDK defaults from location:
/root/.config/sagemaker/config.yaml

Exception ignored in: <bound method
IPythonKernel._clean_thread_parent_frames of
<ipykernel.ipkernel.IPythonKernel object at 0x7f3dec90fd90>>
Traceback (most recent call last):
  File
"/opt/conda/lib/python3.10/site-packages/ipykernel/ipkernel.py", line
770, in _clean_thread_parent_frames
    def _clean_thread_parent_frames(
KeyboardInterrupt:
```

Exploratory Data Analysis

This code is for exploratory data analysis (EDA). Here, we are loading a dataset from Amazon S3 and performing initial exploration.

```
# Load Dataset from S3
df = pd.read_csv('s3://medicaldata01/Medical dataset.csv')
print(df)
```

Data Preprocessing

This section of the code focuses on data preprocessing, including data cleaning and feature engineering.

Data Cleaning and Feature Engineering

```
# Fill Missing Values with Mean
numeric_columns =
df.select_dtypes(include=[np.number]).columns.tolist()
df[numeric_columns] =
df[numeric_columns].fillna(df[numeric_columns].mean())
```

One-Hot Encoding for 'SEX' Column

The 'SEX' column is one-hot encoded to convert categorical data into a numerical format.

```
encoder = OneHotEncoder(drop='first')
sex_encoded = encoder.fit_transform(df[['SEX']])
df_encoded = pd.concat([df.drop('SEX', axis=1),
pd.DataFrame(sex_encoded.toarray(), columns=['SEX_encoded'])], axis=1)
```

Histograms

Histograms are plotted to visualize the distribution of numerical features.

```
df.hist(figsize=(15, 10))
plt.show()
```

Correlation Matrix

A correlation matrix is plotted to examine the relationships between variables.

```
corr_matrix = df_encoded.corr()
plt.figure(figsize=(12, 8))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.show()
```

Skewness and Kurtosis

Skewness and kurtosis of the features are calculated to understand the distribution characteristics.

```
print(df_encoded.skew())
print(df_encoded.kurt())
```

Train-Test-Validation Split

The dataset is split into training, testing, and validation sets.

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

X_train, X_temp, y_train, y_temp = train_test_split(X, y,
test_size=0.2, random_state=42)
X_test, X_val, y_test, y_val = train_test_split(X_temp, y_temp,
test_size=0.5, random_state=42)
```

One-Hot Encoding for 'SEX' Column (Again)

The 'SEX' column in the training and testing sets is one-hot encoded.

```
encoder = OneHotEncoder(drop='first')
X_train_encoded = encoder.fit_transform(X_train[['SEX']]).toarray()
X_test_encoded = encoder.transform(X_test[['SEX']]).toarray()
```

Drop Original 'SEX' Column and Concatenate Encoded Columns

The original 'SEX' column is dropped, and the encoded columns are concatenated with the feature matrices.

```
X_train.drop('SEX', axis=1, inplace=True)
X_test.drop('SEX', axis=1, inplace=True)
X_train = np.hstack((X_train.values, X_train_encoded))
X_test = np.hstack((X_test.values, X_test_encoded))
```

Feature Scaling

Feature scaling is applied to standardize the features.

```
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

Random Forest Classifier Training and Validation

This section of the code trains a Random Forest Classifier with different values of `n_estimators` and evaluates its performance on both the training and validation sets.

```
# Initialize lists to store training and validation scores
train_scores = []
val_scores = []

# Define a range of values for n_estimators
n_estimators_values = range(10, 201, 10)

# Train Random Forest Classifier with different n_estimators values
for n in n_estimators_values:
    clf = RandomForestClassifier(n_estimators=n, random_state=42)
    clf.fit(X_train_scaled, y_train)

    # Calculate training and validation scores
    train_score = clf.score(X_train_scaled, y_train)
    val_score = clf.score(X_test_scaled, y_test)

    train_scores.append(train_score)
    val_scores.append(val_score)

    # Plotting the training and validation scores
plt.figure(figsize=(10, 6))
plt.plot(n_estimators_values, train_scores, label='Training Accuracy',
marker='o')
plt.plot(n_estimators_values, val_scores, label='Validation Accuracy',
marker='x')
plt.xlabel('n_estimators')
plt.ylabel('Accuracy')
plt.title('Training vs Validation Accuracy for Random Forest')
plt.legend()
plt.grid(True)
plt.show()
```

Model Evaluation: Random Forest Classifier

This section of the code evaluates the trained Random Forest Classifier on the test set using various metrics, including the classification report and ROC curve.

```
clf = RandomForestClassifier(random_state=42)
clf.fit(X_train_scaled, y_train)

y_pred = clf.predict(X_test_scaled)

print("Classification Report:")
print(classification_report(y_test, y_pred))
print("ROC AUC Score:", roc_auc_score(y_test, y_pred))

# ROC curve and ROC area for each class
fpr, tpr, _ = roc_curve(y_test, clf.predict_proba(X_test_scaled)[:,
1])
```

```
roc_auc = auc(fpr, tpr)

# Plotting
plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area =
%0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend(loc="lower right")
plt.show()
```

Data Saving and Uploading to S3

This section of the code involves saving the datasets to CSV files and uploading them to Amazon S3 for storage and future use.

```
# Train/test split for dataset export
train_data, test_data = train_test_split(df[numeric_columns],
test_size=0.2, random_state=42)

# Further split the train_data to create validation_data
train_data, validation_data = train_test_split(train_data,
test_size=0.2, random_state=42)

# Save datasets to CSV files
train_data.to_csv('train.csv', index=False)
test_data.to_csv('test.csv', index=False)
validation_data.to_csv('validation.csv', index=False)

# Upload datasets to S3
s3 = boto3.client('s3')
bucket_name = 'medicaldata01'
s3.upload_file('train.csv', 'medicaldata01',
'Medical_data/dataset/train.csv')
s3.upload_file('test.csv', 'medicaldata01',
'Medical_data/dataset/test.csv')
s3.upload_file('validation.csv', 'medicaldata01',
'Medical_data/dataset/validation.csv')
```

Prediction and Displaying Results

This section of the code involves making predictions using the trained classifier on the test set and displaying the results, including predicted labels and probability estimates.

```
# Prediction
y_pred = clf.predict(X_test_scaled)

# Displaying predictions
print("Predicted Labels:", y_pred)

# Probability Estimates
y_prob = clf.predict_proba(X_test_scaled)

# Displaying probability estimates
print("Probability Estimates:", y_prob)
```

Conclusion

In this code, we performed various steps for training and evaluating a machine learning model using Amazon SageMaker. Here's a summary of the key tasks accomplished:

1. **Data Preprocessing**: We cleaned the data, performed feature engineering, handled missing values, and encoded categorical variables. Additionally, we split the dataset into training, testing, and validation sets.
2. **Model Training and Evaluation**: We trained a Random Forest Classifier with different values of `n_estimators` and evaluated its performance using training and validation accuracy scores. We also evaluated the trained classifier on the test set using metrics such as the classification report and ROC curve.
3. **Data Saving and Uploading to S3**: Finally, we saved the datasets to CSV files and uploaded them to Amazon S3 for storage and future use.

Overall, this code demonstrates the end-to-end process of training a machine learning model, evaluating its performance, and saving/uploading datasets for further analysis or deployment. The results obtained from the model evaluation provide insights into its effectiveness and can guide future improvements or decisions.

Thank You

Riddhi Sharma