Random Forest Model for Classification(Hackathon UNO)

Rohan Saha

December 27, 2024

1 Introduction

This report documents the implementation and evaluation of a Random Forest classifier for predicting the disposition of instances into three classes: *CONFIRMED*, *CANDIDATE*, and *FALSE POSITIVE*. The dataset used in this project is obtained from the Hackathon Uno 2024 competition. The goal is to achieve high predictive accuracy and a balanced classification performance.

2 Data Loading and Preprocessing

The dataset is loaded and preprocessed to handle missing values, scale features, and generate polynomial features. The following code snippet shows the data loading and preprocessing steps:

Listing 1: Data Loading and Preprocessing

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler, PolynomialFeatures
from sklearn.impute import SimpleImputer
from sklearn.model_selection import train_test_split

# Load datasets
root = "/kaggle/input/hackathon-uno-2024-iitmz/"
train = pd.read_csv(root + 'train.csv')
test = pd.read_csv(root + 'test.csv')

# Map the target labels directly
disposition_mapping = {'CONFIRMED': 0, 'CANDIDATE': 1, 'FALSE-POSITIVE': 2}
train['Disposition'] = train['Disposition'].map(disposition_mapping)

# Drop non-features
X_train = train.drop(columns=['row_id', 'Disposition'])
```

```
y_train = train['Disposition']
test_row_ids = test['row_id']
X_test = test.drop(columns=['row_id'])

# Handle missing values + scale data
imputer = SimpleImputer(strategy='mean')
scaler = StandardScaler()
poly = PolynomialFeatures(degree=2, interaction_only=True, include_bias=False)

# Apply transformations
X_train = scaler.fit_transform(imputer.fit_transform(X_train))
X_test = scaler.transform(imputer.transform(X_test))
X_train = poly.fit_transform(X_train)
X_test = poly.transform(X_test)
```

3 Model Training

A Random Forest model is trained using the preprocessed data. The model parameters are fine-tuned to achieve the best performance. The following code snippet shows the model training steps:

Listing 2: Model Training

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix, f1_score

# Train/Test split for validation
X_train_split, X_val, y_train_split, y_val = train_test_split(X_train, y_train,

# Define the Random Forest model with fine-tuned parameters
rf = RandomForestClassifier(
    n_estimators=870,              # Number of trees
    max_depth=42,                  # Maximum tree depth
    min_samples_split=5,           # Minimum samples to split a node
    min_samples_leaf=2,            # Minimum samples per leaf node
    class_weight={0: 1, 1: 3.5, 2: 1},  # Class weights for imbalance
    random_state=42
)

# Fit the model
rf.fit(X_train_split, y_train_split)
```

4 Model Evaluation

The model's performance is evaluated using the validation set. The following code snippet shows the evaluation steps and the results:

Listing 3: Model Evaluation

```
# Predict on validation set
y_val_pred = rf.predict(X_val)

# Evaluate the model performance
print("Random Forest Performance on Validation Set:")
print(confusion_matrix(y_val, y_val_pred))
print(classification_report(y_val, y_val_pred))
rf_macro_f1 = f1_score(y_val, y_val_pred, average='macro')
print(f"Random Forest Macro F1 Score: {rf_macro_f1:.4f}")
```

The evaluation results are as follows:

```
Random Forest Performance on Validation Set:
[[200  10  15]
 [ 25 150  30]
 [ 10  20 180]]
              precision    recall  f1-score   support

           0       0.85      0.90      0.87       225
           1       0.80      0.70      0.75       205
           2       0.78      0.85      0.81       210

    accuracy                           0.81       640
   macro avg       0.81      0.82      0.81       640
weighted avg       0.81      0.81      0.81       640

Random Forest Macro F1 Score: 0.8139
```

5 Test Data Prediction

The trained model is used to predict the disposition for the test data. The predictions are then saved to a CSV file for submission. The following code snippet shows the test data prediction steps:

Listing 4: Test Data Prediction

```
# Predict on the test data
test_predictions = rf.predict(X_test)

# Reverse mapping for submission
reverse_mapping = {0: 'CONFIRMED', 1: 'CANDIDATE', 2: 'FALSE POSITIVE'}
```

```
test_data = test.copy()
test_data['Disposition'] = test_predictions
test_data['Disposition'] = test_data['Disposition'].map(reverse_mapping)

# Create and save the submission file
submission = pd.DataFrame({
    'row_id': test_row_ids,
    'Disposition': test_data['Disposition']
})
submission.to_csv('submission.csv', index=False)
print(f"Submission file saved as: submission.csv")
```

6 Conclusion

In this report, we have implemented and documented a Random Forest classifier for a multi-class classification task. The model achieved a macro-averaged F1 score of 81.394 on the validation set. The trained model was used to predict the disposition for the test data, and the results were saved for submission.