CaseNext(single case)

April 20, 2025

[4]:
```python
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
import warnings
import sys

warnings.filterwarnings('ignore')

# --- 1. Load Data and Define Features ---
try:
    import openpyxl
    df = pd.read_excel(r"C:
 ↪\Users\dell\Downloads\Indian_Court_Cases_Dataset_Updated.xlsx")
    df['Priority_Label'] = df['Priority_Label'].astype(str).str.strip()
    df.dropna(subset=['Priority_Label'], inplace=True)
except (FileNotFoundError, ImportError) as e:
    print(f"Error: {e}")
    sys.exit(1)
except Exception as e:
    print(f"An error occurred while loading or cleaning the file: {e}")
    sys.exit(1)

FEATURE_DEFINITIONS = {
    'Court_Name': {'type': 'categorical'},
    'Case_Type': {'type': 'categorical'},
    'Urgency_Tag': {'type': 'categorical'},
    'Advocate_Names': {'type': 'categorical'},
    'Legal_Sections': {'type': 'categorical'},
    'Past_History': {'type': 'categorical'},
    'Estimated_Impact': {'type': 'categorical'},
    'Media_Coverage': {'type': 'categorical'},
}

TARGET_COLUMN = 'Priority_Label'

# --- Helper Functions ---
def get_feature_lists(definitions, df):
    return [name for name, details in definitions.items()
            if details['type'] == 'categorical' and name in df.columns]

def create_preprocessor(categorical_features):
    categorical_transformer = Pipeline(steps=[
        ('imputer', SimpleImputer(strategy='most_frequent')),
        ('onehot', OneHotEncoder(handle_unknown='ignore', sparse_output=False))
    ])
    return ColumnTransformer(
        transformers=[('cat', categorical_transformer, categorical_features)],
        remainder='drop'
    )

def train_models(X_train, y_train, preprocessor):
    rf_model = Pipeline([
        ('preprocessor', preprocessor),
        ('classifier', RandomForestClassifier(n_estimators=100,␣
 ↪random_state=42))
    ])
    lr_model = Pipeline([
        ('preprocessor', preprocessor),
        ('classifier', LogisticRegression(random_state=42, max_iter=1000,␣
 ↪class_weight='balanced'))
    ])
    rf_model.fit(X_train, y_train)
    lr_model.fit(X_train, y_train)
    return rf_model, lr_model

def get_user_input(definitions, df):
    user_data = {}
    for feature, details in definitions.items():
        if feature not in df.columns:
            user_data[feature] = input(f"Enter value for {feature}: ")
            continue
        options = df[feature].dropna().unique()
        if options.size > 0:
            print(f"\nFeature: {feature}\nOptions:")
            for i, opt in enumerate(options[:20]):
                print(f"{i+1}. {opt[:50]}")
            if len(options) > 20: print("...")
            while True:
                choice = input(f"Enter choice number (1-{len(options)}), 's' to␣
↪skip, or type custom value: ").strip()
                if choice.lower() == 's':
                    user_data[feature] = np.nan; break
                try:
                    choice_index = int(choice) -1
                    if 0 <= choice_index < len(options):
                        user_data[feature] = options[choice_index]; break
                    print(f"Invalid choice. Enter 1-{len(options)}.")
                except ValueError: print("Invalid input. Try again.")
        else: user_data[feature] = input(f"Enter value for {feature}: ")
    return pd.DataFrame([user_data])

def predict_priority(input_df, rf_model, lr_model):
    try:
        rf_proba = rf_model.predict_proba(input_df)[0]
        rf_proba_dict = {class_label: f"{prob:.1%}" for class_label, prob in␣
↪zip(rf_model.classes_, rf_proba)}

        lr_proba = lr_model.predict_proba(input_df)[0]
        lr_proba_dict = {class_label: f"{prob:.1%}" for class_label, prob in␣
↪zip(lr_model.classes_, lr_proba)}

        averaged_probabilities = {}
        for label in rf_model.classes_:
            rf_prob = rf_proba_dict.get(label, 0.0)
            lr_prob = lr_proba_dict.get(label, 0.0)
            try:
                averaged_prob = (float(rf_prob.strip('%')) + float(lr_prob.
↪strip('%')))/2
                averaged_probabilities[label] = f"{averaged_prob:.1f}%"
            except (ValueError, TypeError):
                averaged_probabilities[label] = "N/A"

        final_prediction = max(averaged_probabilities,
                               key=lambda k: float(averaged_probabilities[k].
↪strip('%')) if averaged_probabilities[k] != "N/A" else -1)

        print("\n=== Prediction Results ===")
        print(f"Final Priority Prediction: {final_prediction}")
        print("\nConfidence Levels:")
        for label, prob in sorted(averaged_probabilities.items(),
                                  key=lambda x: float(x[1].rstrip('%')) if x[1] !
↪= "N/A" else -1,
                                  reverse=True):
            print(f"  {label}: {prob}")
```

```python
    except Exception as e:
        print(f"\nError during prediction: {str(e)}")

# --- 2. Preprocessing and Model Training ---
X = df[list(FEATURE_DEFINITIONS.keys())]
y = df[TARGET_COLUMN]
categorical_features = get_feature_lists(FEATURE_DEFINITIONS, df)
preprocessor = create_preprocessor(categorical_features)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42, stratify=y)
rf_model, lr_model = train_models(X_train, y_train, preprocessor)

# --- 3. Get User Input and Predict ---
while True:
    try:
        user_input_df = get_user_input(FEATURE_DEFINITIONS, df)
        predict_priority(user_input_df, rf_model, lr_model)

        another = input("\nMake another prediction? (y/n): ").strip().lower()
        if another != 'y':
            break

    except Exception as e:
        print(f"Error: {str(e)}")
        print("Please try again with different inputs.")

print("\nProgram completed.")
```

```
Feature: Court_Name
Options:
1. Supreme Court
2. High Court
3. District Court
Enter choice number (1-3), 's' to skip, or type custom value: 2

Feature: Case_Type
Options:
1. Family
2. Civil
3. Criminal
4. PIL
Enter choice number (1-4), 's' to skip, or type custom value: 1

Feature: Urgency_Tag
Options:
1. Emergency
```

```
2. Regular
3. High-profile
Enter choice number (1-3), 's' to skip, or type custom value: 2

Feature: Advocate_Names
Options:
1. A. Singh, R. Mehta
2. L. Verma, L. Verma
3. L. Verma, R. Mehta
4. R. Mehta, A. Singh
5. R. Mehta, R. Mehta
6. K. Sharma, L. Verma
7. K. Sharma, R. Mehta
8. L. Verma, A. Singh
9. K. Sharma, K. Sharma
10. L. Verma, K. Sharma
11. R. Mehta, K. Sharma
12. A. Singh, L. Verma
13. A. Singh, A. Singh
14. A. Singh, K. Sharma
15. R. Mehta, L. Verma
Enter choice number (1-15), 's' to skip, or type custom value: 12

Feature: Legal_Sections
Options:
1. Article 32
2. CRPC 125
3. IPC 302
4. Family Act 1955
5. IPC 420, 467
Enter choice number (1-5), 's' to skip, or type custom value: 2

Feature: Past_History
Options:
1. No
2. Yes
Enter choice number (1-2), 's' to skip, or type custom value: 2

Feature: Estimated_Impact
Options:
1. Medium
2. High
3. Low
Enter choice number (1-3), 's' to skip, or type custom value: 1

Feature: Media_Coverage
Options:
1. No
```

```
2. Yes
Enter choice number (1-2), 's' to skip, or type custom value: 1

=== Prediction Results ===
Final Priority Prediction:  Low

Confidence Levels:
   Low: 38.4%
   Medium: 31.6%
   High: 30.0%

Make another prediction? (y/n): n

Program completed.
```

[ ]: