Configuration Manual for Crime Category Classification

Jerin Eldho
x21196737

1 Introduction

This configuration manual provides a comprehensive guide for setting up and implementing a research project focused on accurately categorizing crime incidents based on textual descriptions. Various machine learning models, including LSTM, GRU, Logistic Regression, SVM, and Random Forest, are employed to achieve this objective.

2 Environment Setup

Device Specifications:

1. Model Name: MI Notebook Pro
2. Processor: Intel Core i5-11300H @ 3.10GHz (Base) up to 3.11GHz (Turbo Boost), 11th Gen
3. Memory (RAM): 16.0 GB DDR4
4. Graphics: Integrated Graphics (Intel Xe Graphics)
5. Operating System: Windows 11 Home Single Language (64-bit)

Required Software:

1. Development Environment: Google Colab
2. Programming Language: Python 3.8.16
3. Cloud Storage Integration: Google Drive

These software components are essential for the proper functioning of the project. Google Colab provides the computational environment for executing code, while Python 3.8.16 serves as the programming language for implementing the project's algorithms and data processing. Integration with Google Drive enables seamless data access and storage, facilitating efficient collaboration and data management throughout the research endeavor.

3 Python Libraries Essential for Implementing this Project

The below figure shows the python libraries used for implementing this project:

```python
# Linear algebra
import numpy as np
import pandas as pd

# Data processing
import pandas as pd

# Data visualization
import seaborn as sns
%matplotlib inline
from matplotlib import pyplot as plt
from matplotlib import style

# Deep Learning
import tensorflow as tf
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, Dense, Dropout
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.layers import GRU
from tensorflow.keras.layers import Embedding, Dense
from sklearn.preprocessing import LabelEncoder
from keras.layers import Embedding, GRU, Dense, Dropout
from keras.regularizers import l2
# Machine Learning Algorithms
from sklearn.model_selection import train_test_split
from sklearn.linear_model import SGDClassifier, LogisticRegression
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import Perceptron
from sklearn.svm import SVC, LinearSVC
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report


# Preprocessing
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler, OneHotEncoder

# Metrics
from sklearn.metrics import log_loss
from sklearn.model_selection import cross_val_score
from sklearn.metrics import confusion_matrix

# Model Selection & Hyperparameter tuning
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV, StratifiedKFold


# Mathematical Functions
import math

#For input and output handling
import io
```

Figure 1: Importing python libraries

4 Loading the dataset

To access the dataset stored in Google Drive, use the following code snippet to mount Google Drive within the Colab environment and read the CSV data into a DataFrame named "crimedata":

```python
from google.colab import drive

drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```python
[ ]
google_drive_path = '/content/drive/MyDrive/CrimeDataset.csv'
```

```python
[ ] crime_data = pd.read_csv(google_drive_path)
```

Figure 2: Loading dataset

5 Data Preprocessing and Cleaning

The dataset was initially downloaded from the SFPD website, containing records from 2018 to 2023. To ensure data integrity:

Data from 2022 to 2023 was extracted (Figure 3). Missing values were handled by substituting them with appropriate average values (Figure 4). The "Incidentdatetime" column was converted into a datetime format for precise temporal referencing (Figure 5). The "IncidentDayofWeek" column was transformed into numerical categorical values for machine learning compatibility (Figure 6). Categories were mapped to a new unified category for improved modeling (Figure 7).

```
# Extracting data for specific years
start_year = 2022
end_year = 2023

# Filtering the data based on the year column
filtered_df = crime_data[(crime_data['Incident Year'] >= start_year) & (crime_data['Incident Year'] <= end_year)]
```

Figure 3: Extracting data from 2022 till 2023

```
# Replacing null values with average values in integer columns
numeric_columns = filtered_df.select_dtypes(include=['int', 'float']).columns
filtered_df[numeric_columns] = filtered_df[numeric_columns].fillna(filtered_df[numeric_columns].mean())

# Replacing null values with mode values in string columns
string_columns = filtered_df.select_dtypes(include=['object']).columns
filtered_df[string_columns] = filtered_df[string_columns].fillna(filtered_df[string_columns].mode().iloc[0])
```

Figure 4: Code for substituting null values with suitable average values

```
# Converting "Incident_datetime" column to datetime format

date_formats = ['%d-%m-%Y', '%Y/%m/%d %I:%M:%S %p']
for date_format in date_formats:
    try:
        filtered_df['Incident_Datetime'] = pd.to_datetime(filtered_df['Incident_Datetime'], format=date_format)
        break
    except ValueError:
        pass

# Filtering records only for 2021-2023
start_date = pd.to_datetime('2021-01-01')
end_date = pd.to_datetime('2023-12-31')
crime_data1 = filtered_df[(filtered_df['Incident_Datetime'] >= start_date) & (filtered_df['Incident_Datetime'] <= end_date)]

# Printing the DataFrame after preprocessing
print(crime_data1.shape[0])
```

Figure 5: Conversion of Incidentdatetime column into a datetime format

```
# Converting the "Incident_Day_of_Week" column to numerical categorical values
day_of_week_map = {
    'Monday': 1,
    'Tuesday': 2,
    'Wednesday': 3,
    'Thursday': 4,
    'Friday': 5,
    'Saturday': 6,
    'Sunday': 7
}

crime_data1['Incident_Day_of_Week'] = crime_data1['Incident_Day_of_Week'].astype('category').map(day_of_week_map)
```

Figure 6: Conversion of IncidentDayofWeek column into numerical categorical values

```
# Defining the mapping of old categories to the new category
category_mapping = {
    'Drug Violation': 'Drug Offenses',
    'Drug Offense': 'Drug Offenses',
    'Non-Criminal': 'Other Miscellaneous Offenses',
    'Other Offenses': 'Other Miscellaneous Offenses',
    'Other Miscellaneous': 'Other Miscellaneous Offenses',
    'Other': 'Other Miscellaneous Offenses',
    'Human Trafficking, Commercial Sex Acts': 'Human Trafficking for Commercial Sex Acts',
    'Human Trafficking (A), Commercial Sex Acts': 'Human Trafficking for Commercial Sex Acts',
    'Prostitution': 'Sex-related Crimes',
    'Rape': 'Sex-related Crimes',
    'Sex Offense': 'Sex-related Crimes',
    'Suspicious Occ': 'Suspicious Activity',
    'Suspicious': 'Suspicious Activity',
    'Traffic Collision': 'Vehicle-related Offenses',
    'Traffic Violation Arrest': 'Vehicle-related Offenses',
    'Vandalism': 'Vehicle-related Offenses',
    'Vehicle Misplaced': 'Vehicle-related Offenses',
    'Vehicle Impounded': 'Vehicle-related Offenses',
    'Weapons Carrying Etc': 'Weapons-related Offenses',
    'Weapons Offense': 'Weapons-related Offenses',
    'Weapons Offence': 'Weapons-related Offenses',
    'Larceny Theft': 'Property Crimes',
    'Lost Property': 'Property Crimes',
    'Stolen Property': 'Property Crimes',
    'Robbery': 'Property Crimes',
    'Motor Vehicle Theft': 'Property Crimes',
    'Motor Vehicle Theft?': 'Property Crimes',
    'Burglary': 'Property Crimes',
    'Embezzlement': 'Property Crimes',
    'Homicide': 'Fatal Crimes',
    'Suicide': 'Fatal Crimes',
    'Arson': 'Arson',
    'Assault': 'Assault',
    'Case Closure': 'Case Closure',
    'Civil Sidewalks': 'Civil Sidewalks',
    'Courtesy Report': 'Courtesy Report',
    'Disorderly Conduct': 'Disorderly Conduct',
    'Fire Report': 'Fire Report',
    'Forgery And Counterfeiting': 'Forgery And Counterfeiting',
    'Fraud': 'Fraud',
    'Gambling': 'Gambling',
    'Liquor Laws': 'Liquor Laws',
    'Malicious Mischief': 'Malicious Mischief',
    'Miscellaneous Investigation': 'Miscellaneous Investigation',
    'Missing Person': 'Missing Person',
    'Offences Against The Family And Children': 'Offences Against The Family And Children',
    'Recovered Vehicle': 'Recovered Vehicle',
    'Warrant': 'Warrant'

}

# Mapping the categories to the new category called 'Unified Category'
crime_data1['Unified_category'] = crime_data1['Incident_Category'].map(category_mapping)
```

Figure 7: Mapping of old categories to the new category

6 Feature Engineering and Visualization

The code snippet in Figure 8 demonstrates feature extraction from the ”Incidentdatetime” column, including date, year, month, day, hour, minute, hour type, season, and weekend feature. Various plots were generated for data visualization:

```python
# Extracting date,year,month,day,hour,minute etc. from 'Incident_Datetime' column

def extract_datetime_features(df):
    df['Incident_Datetime'] = pd.to_datetime(df['Incident_Datetime'])  # Convert 'Incident_date' column to pandas datetime

    # Extracting year, month, date, day, hour, and minute
    df['Year'] = df['Incident_Datetime'].dt.year
    df['Month'] = df['Incident_Datetime'].dt.month
    df['Day'] = df['Incident_Datetime'].dt.day
    df['Hour'] = df['Incident_Datetime'].dt.hour
    df['Minute'] = df['Incident_Datetime'].dt.minute

    # Extracting hour type (morning, afternoon, evening, night)
    df['Hour_type'] = df['Incident_Datetime'].dt.hour.apply(lambda x: 'Morning' if 6 <= x < 12 else
                                                    'Afternoon' if 12 <= x < 18 else
                                                    'Evening' if 18 <= x < 22 else 'Night')

    # Extracting season (winter, summer, fall, spring)
    df['Season'] = df['Incident_Datetime'].dt.month.apply(lambda m: 'Winter' if m in [12, 1, 2] else
                                                 'Spring' if m in [3, 4, 5] else
                                                 'Summer' if m in [6, 7, 8] else 'Fall')
    # Adding weekend feature (1 for weekend, 0 for weekdays)
    df['Weekend'] = df['Incident_Datetime'].dt.dayofweek.apply(lambda x: 1 if x >= 5 else 0)

    return df

# Callng the function to extract datetime features and update the DataFrame
crime_data1 = extract_datetime_features(crime_data1)
```

Figure 8: Code for extracting different features from 'Incidentdatetime' column

The following figures display the code snippets for plotting various distributions, crime categories distribution, distribution of crimes by year and police district, and occurrences of crimes on a yearly basis:

```python
[ ] # Defining a function of plotting various distributions
    def plot_distribution(data, column_name):
        plt.figure(figsize=(10, 6))
        sns.countplot(x=column_name, data=data)
        plt.title(f'Distribution of {column_name}')
        plt.xlabel(column_name)
        plt.ylabel('Count')
        plt.xticks(rotation=45)
        plt.show()
```

```python
[ ] plot_distribution(crime_df, 'Police_District')
    plot_distribution(crime_df, 'Year')
    plot_distribution(crime_df, 'Month')
    plot_distribution(crime_df, 'Hour_type')
    plot_distribution(crime_df, 'Season')
    plot_distribution(crime_df, 'Weekend')
```

Figure 9: Code for plotting various distributions

```
[ ] # Plotting Crime Categories Distribution
    plt.figure(figsize=(12, 6))
    sns.countplot(x='Unified_category', data=crime_df)
    plt.title('Crime Categories Distribution')
    plt.xlabel('Unified Category')
    plt.ylabel('Count')
    plt.xticks(rotation=90)
    plt.show()
```

Figure 10: Code for plotting Crime Categories Distribution

```
[ ] # Distribution of Crimes by Year and Police District
    plt.figure(figsize=(12, 6))
    sns.countplot(x='Police_District', hue='Year', data=crime_df)
    plt.title('Distribution of Crimes by Year and Police District')
    plt.xlabel('Police District')
    plt.ylabel('Count')
    plt.legend(title='Year', loc='upper right')
    plt.xticks(rotation=45)
    plt.show()
```

Figure 11: Code for plotting Distribution of Crimes by Year and Police District

```
# Ploting the occurrences of crimes that had happened yearly basis.
plt.figure(figsize=(12, 6))
sns.countplot(x='Year', data=crime_df, hue='Unified_category', palette='tab10')
plt.title('Occurrences of Crimes by Year and Crime Category')
plt.xlabel('Year')
plt.ylabel('Count')
plt.legend(title='Crime Category', bbox_to_anchor=(1, 1))
plt.xticks(rotation=45)
plt.show()
```

Figure 12: Code for ploting the occurrences of crimes that had happened yearly basis

7 Implementation of various machine learning models

The implementation of machine learning models, including LSTM, GRU, Logistic Regression, SVM, and Random Forest, is illustrated in the following figures:

LSTM Model

```
X_train = train_data['Incident_Description'].tolist()
y_train = train_data['Unified_category'].tolist()
X_test = test_data['Incident_Description'].tolist()
y_test = test_data['Unified_category'].tolist()

# Tokenizing the text and convert to sequences
tokenizer = Tokenizer(num_words=100000,filters='!"#$%&()*+,-./:;<=>?@[\\]^_`{|}~\t\n')
tokenizer.fit_on_texts(X_train)
sequences_train = tokenizer.texts_to_sequences(X_train)
sequences_test = tokenizer.texts_to_sequences(X_test)

# Padding the sequences to make them of equal length
max_sequence_length = 550
sequences_train_padded = pad_sequences(sequences_train, maxlen=max_sequence_length, padding='post')
sequences_test_padded = pad_sequences(sequences_test, maxlen=max_sequence_length, padding='post')

# Converting the category labels to numerical format
label_encoder = LabelEncoder()
y_train_encoded = label_encoder.fit_transform(y_train)
y_test_encoded = label_encoder.transform(y_test)

# Converting the numerical labels to one-hot encoded format
num_classes = len(label_encoder.classes_)
y_train_onehot = to_categorical(y_train_encoded, num_classes=num_classes)
y_test_onehot = to_categorical(y_test_encoded, num_classes=num_classes)
```

Figure 13: Text Data Preprocessing and Label Encoding of LSTM Model

```
# Creating the LSTM model
embedding_dim = 128
lstm_units = 64

model = Sequential()
model.add(Embedding(input_dim=len(tokenizer.word_index) + 1, output_dim=embedding_dim, input_length=max_sequence_length))
model.add(LSTM(units=256,activation='tanh',return_sequences=True))
model.add(LSTM(units=128,activation='tanh'))
model.add(Dense(num_classes, activation='softmax'))
# Compiling the model
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

# Training the model
batch_size = 64
epochs = 5
model.fit(sequences_train_padded, y_train_onehot, batch_size=batch_size, epochs=epochs, validation_split=0.1)

# Evaluating the model
loss, accuracy = model.evaluate(sequences_test_padded, y_test_onehot)
print(f"Test loss: {loss}, Test accuracy: {accuracy}")

# Generate predictions
y_pred_onehot = model.predict(sequences_test_padded)

# Convert predictions to labels
y_pred_encoded = np.argmax(y_pred_onehot, axis=1)
y_pred_labels = label_encoder.inverse_transform(y_pred_encoded)

# Convert true labels to labels
y_true_encoded = np.argmax(y_test_onehot, axis=1)
y_true_labels = label_encoder.inverse_transform(y_true_encoded)

# Calculate confusion matrix
confusion_mtx = confusion_matrix(y_true_labels, y_pred_labels, labels=label_encoder.classes_)

# Print confusion matrix
print("Confusion Matrix:")
print(confusion_mtx)
```

Figure 14: LSTM Model

GRU Model

```
[ ] X_train = train_data['Incident_Description'].tolist()
    y_train = train_data['Unified_category'].tolist()
    X_test = test_data['Incident_Description'].tolist()
    y_test = test_data['Unified_category'].tolist()

    # Tokenizing the text and convert to sequences
    tokenizer = Tokenizer(num_words=100000,filters='!"#$%&()*+,-./:;<=>?@[\\]^_`{|}~\t\n')
    tokenizer.fit_on_texts(X_train)
    sequences_train = tokenizer.texts_to_sequences(X_train)
    sequences_test = tokenizer.texts_to_sequences(X_test)

    # Padding the sequences to make them of equal length
    max_sequence_length = 550
    sequences_train_padded = pad_sequences(sequences_train, maxlen=max_sequence_length, padding='post')
    sequences_test_padded = pad_sequences(sequences_test, maxlen=max_sequence_length, padding='post')

    # Converting the category labels to numerical format
    label_encoder = LabelEncoder()
    y_train_encoded = label_encoder.fit_transform(y_train)
    y_test_encoded = label_encoder.transform(y_test)

    # Converting the numerical labels to one-hot encoded format
    num_classes = len(label_encoder.classes_)
    y_train_onehot = to_categorical(y_train_encoded, num_classes=num_classes)
    y_test_onehot = to_categorical(y_test_encoded, num_classes=num_classes)
```

Figure 15: Text Data Preprocessing and Label Encoding of GRU Model

```python
# Creating the GRU model
embedding_dim = 128
gru_units = 64

model = Sequential()
model.add(Embedding(input_dim=len(tokenizer.word_index) + 1, output_dim=embedding_dim, input_length=max_sequence_length))
model.add(Dropout(0.2))  # Adding dropout after embedding layer
model.add(GRU(units=750, dropout=0.2, activation='tanh', return_sequences=True, kernel_regularizer=l2(0.01)))
model.add(GRU(units=380, activation='tanh', kernel_regularizer=l2(0.01)))
model.add(Dense(num_classes, activation='softmax'))

# Compiling the model
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

# Training the model
batch_size = 64
epochs = 5
model.fit(sequences_train_padded, y_train_onehot, batch_size=batch_size, epochs=epochs, validation_split=0.1)

# Evaluating the model
loss, accuracy = model.evaluate(sequences_test_padded, y_test_onehot)
print(f"Test loss: {loss}, Test accuracy: {accuracy}")

# Generate predictions
y_pred_onehot = model.predict(sequences_test_padded)

# Convert predictions to labels
y_pred_encoded = np.argmax(y_pred_onehot, axis=1)
y_pred_labels = label_encoder.inverse_transform(y_pred_encoded)

# Convert true labels to labels
y_true_encoded = np.argmax(y_test_onehot, axis=1)
y_true_labels = label_encoder.inverse_transform(y_true_encoded)

# Calculate confusion matrix
confusion_mtx = confusion_matrix(y_true_labels, y_pred_labels, labels=label_encoder.classes_)

# Print confusion matrix
print("Confusion Matrix:")
print(confusion_mtx)
```

Figure 16: GRU Model

Logistic Regression Model

```python
X_train = train_data['Incident_Description'].tolist()
y_train = train_data['Unified_category'].tolist()
X_test = test_data['Incident_Description'].tolist()
y_test = test_data['Unified_category'].tolist()

# Step 1: Converting the category labels to numerical format
label_encoder = LabelEncoder()
y_train_encoded = label_encoder.fit_transform(y_train)
y_test_encoded = label_encoder.transform(y_test)

# Step 2: Vectorizing the text data using TfidfVectorizer
vectorizer = TfidfVectorizer(stop_words='english', max_features=5000)
X_train_vectors = vectorizer.fit_transform(X_train)
X_test_vectors = vectorizer.transform(X_test)

# Step 3: Creating and training the Logistic Regression model
logreg_model = LogisticRegression(max_iter=1000)
logreg_model.fit(X_train_vectors, y_train_encoded)

# Step 4: Making predictions on the test data
y_pred = logreg_model.predict(X_test_vectors)

# Step 5: Converting numerical predictions back to category labels
y_pred_labels = label_encoder.inverse_transform(y_pred)

# Step 6: Evaluating the model
accuracy = accuracy_score(y_test, y_pred_labels)
print("Accuracy:", accuracy)

# Classification report for detailed performance metrics
print(classification_report(y_test, y_pred_labels))
```

Figure 17: Logistic Regression Model

SVM

+ Code

```python
[ ]
X_train = train_data['Incident_Description'].tolist()
y_train = train_data['Unified_category'].tolist()
X_test = test_data['Incident_Description'].tolist()
y_test = test_data['Unified_category'].tolist()

# Step 1: Converting the category labels to numerical format
label_encoder = LabelEncoder()
y_train_encoded = label_encoder.fit_transform(y_train)
y_test_encoded = label_encoder.transform(y_test)

# Step 2: Vectorizing the text data using TfidfVectorizer
vectorizer = TfidfVectorizer(stop_words='english', max_features=5000)
X_train_vectors = vectorizer.fit_transform(X_train)
X_test_vectors = vectorizer.transform(X_test)

# Step 3: Creating and training the Support Vector Machine (SVM) model
svm_model = SVC(kernel='linear')  # You can experiment with different kernels (linear, rbf, etc.)
svm_model.fit(X_train_vectors, y_train_encoded)

# Step 4: Making predictions on the test data
y_pred = svm_model.predict(X_test_vectors)

# Step 5: Converting numerical predictions back to category labels
y_pred_labels = label_encoder.inverse_transform(y_pred)

# Step 6: Evaluating the model
accuracy = accuracy_score(y_test, y_pred_labels)
print("Accuracy:", accuracy)

#Classification report for more detailed performance metrics
print(classification_report(y_test, y_pred_labels))
```

Figure 18: SVM model

Random Forest

```python
[ ]
X_train = train_data['Incident_Description'].tolist()
y_train = train_data['Unified_category'].tolist()
X_test = test_data['Incident_Description'].tolist()
y_test = test_data['Unified_category'].tolist()

# Step 1: Converting the category labels to numerical format
label_encoder = LabelEncoder()
y_train_encoded = label_encoder.fit_transform(y_train)
y_test_encoded = label_encoder.transform(y_test)

# Step 2: Vectorizing the text data using TfidfVectorizer
vectorizer = TfidfVectorizer(stop_words='english', max_features=5000)
X_train_vectors = vectorizer.fit_transform(X_train)
X_test_vectors = vectorizer.transform(X_test)

# Step 3: Creating and training the Random Forest model
random_forest_model = RandomForestClassifier(n_estimators=100, random_state=42)
random_forest_model.fit(X_train_vectors, y_train_encoded)

# Step 4: Making predictions on the test data
y_pred = random_forest_model.predict(X_test_vectors)

# Step 5: Converting numerical predictions back to category labels
y_pred_labels = label_encoder.inverse_transform(y_pred)

# Step 6: Evaluating the model
accuracy = accuracy_score(y_test, y_pred_labels)
print("Accuracy:", accuracy)

# classification report for more detailed performance metrics
print(classification_report(y_test, y_pred_labels))
```

Figure 19: Random Forest