A REPORT ON

CYBERBULLYING DETECTION USING A HYBRID RNN-LSTM MODEL

Introduction

Cyberbullying has become a significant concern in the digital age, with harmful online interactions impacting individuals' mental health and well-being. To address this issue, our project aims to develop an automated cyberbullying detection system using a combination of machine learning and deep learning techniques.

The dataset for this project was collected from multiple sources:

1. Comments were extracted from YouTube videos using the **YouTube Data API v3**, providing a rich dataset of real-world user interactions.
2. An additional dataset was sourced from **Kaggle**, consisting of around **18,000 tweets**. This dataset offered a broader perspective on social media behavior, making the system more robust and applicable across platforms.

The combined dataset underwent rigorous preprocessing, including tokenization, lemmatization, and stemming, followed by labelling each comment or tweet as either cyberbullying or non-cyberbullying.

This report outlines the process from data collection to the development of an interactive Graphical User Interface (GUI) that predicts whether a given comment constitutes cyberbullying.

YouTube Comments Data Collection and Analysis

Introduction

This project aims to collect and analyse YouTube comments from a specified video using the **YouTube Data API v3**. The main goal is to leverage web scraping and data collection techniques to gather valuable information on user interactions. Analysing YouTube comments can provide insights into audience sentiment, engagement, and behaviour, making it valuable for both content creators and researchers.

Web Scraping and Data Collection

Using the **YouTube Data API**, we built a script to programmatically retrieve comments for a specific video. The setup includes importing the necessary libraries (`googleapiclient.discovery` and `pandas`) and configuring API parameters. The **Google API client** is initialized with the `api_service_name`, `api_version`, and an API key (`DEVELOPER_KEY`) to authenticate requests securely.

A request object is created using `youtube.commentThreads().list()`, where:

- `**part="snippet"**` fetches the metadata, including the author, text, and timestamps.

- `**videoId="5EpyN_6dqyk"**` specifies the YouTube video ID.

- `**maxResults=100**` limits the response to the first 100 comments.

This request is executed to obtain a JSON response containing comment threads, with each top-level comment accessible through the `items` attribute. We iterate over `response['items']` to extract relevant details, including the author's name, comment text, publication date, last update, and like count. Each comment's data is stored in a list for easy handling.

Data Structuring and Analysis

Using `pandas`, we create a structured DataFrame to store and organize the collected comment data. This DataFrame has the following columns:

- `**author**`: The name of the user who posted the comment.

- `**published_at**` and `**updated_at**`: Timestamps indicating the comment's publication and last update.

- `**like_count**`: The number of likes received by each comment.

- `**text**`: The actual content of the comment.

This structured data format facilitates further analysis and visualization. The initial preview (`df.head(10)`) of the DataFrame shows the top 10 comments, ensuring data accuracy and enabling preliminary insights.

Preprocessing and Labelling:

Importing the necessary libraries:

Using pandas, the data can be loaded and structured. The nltk library aids in natural language processing tasks: word_tokenize splits text into words, stopwords removes common but uninformative words, and both WordNetLemmatizer and PorterStemmer help standardize words by converting them to their root forms. For sentiment analysis, TextBlob calculates the polarity of text, assigning sentiment scores to classify comments as positive, negative, or neutral. These combined tools effectively prepare textual data for sentiment classification tasks.

Downloading the necessary NLTK data:

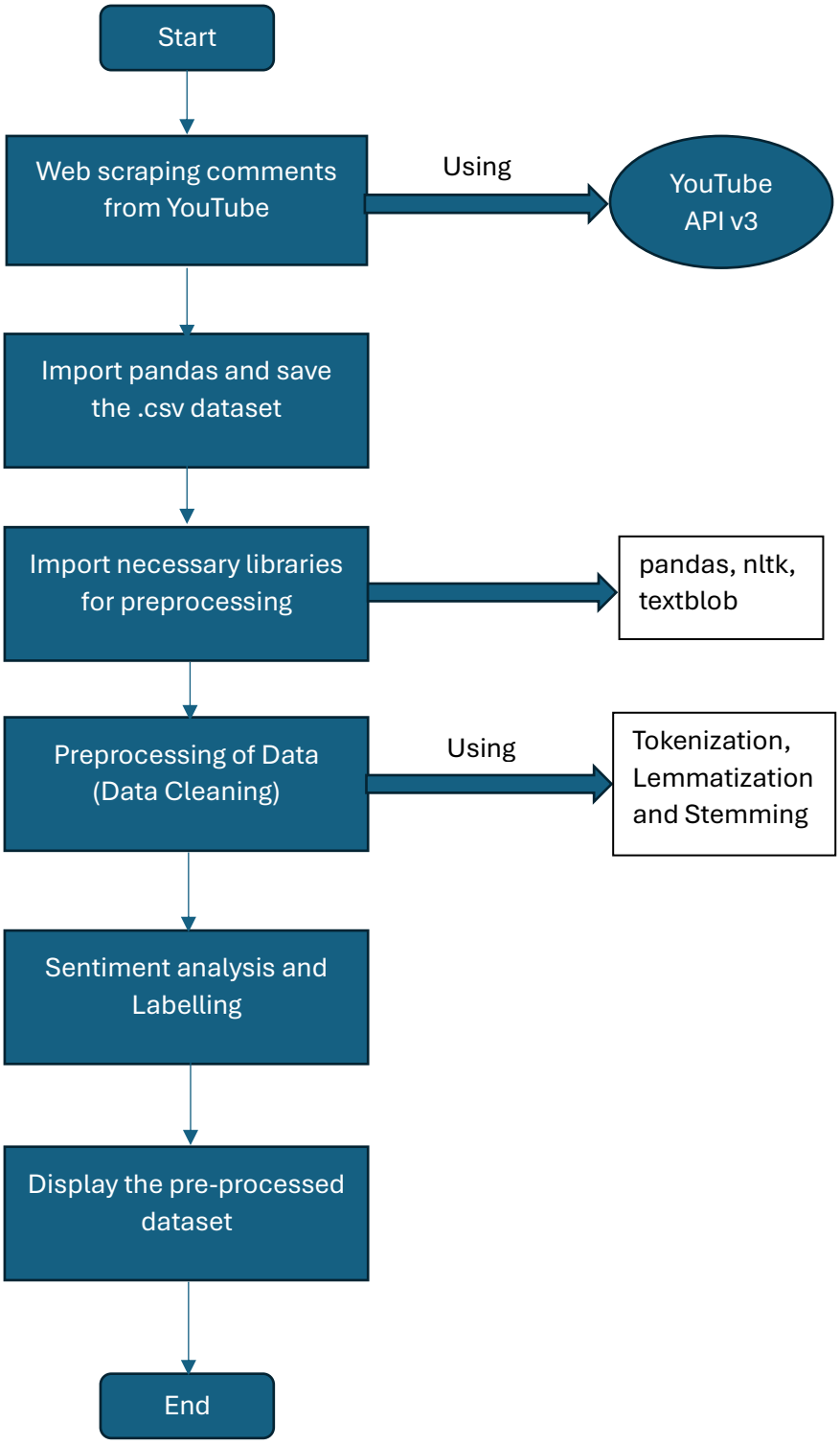
This code block downloads essential data for text preprocessing with the Natural Language Toolkit (NLTK):

- `punkt`: Provides pre-trained data for tokenizing text into sentences and words, enabling `word_tokenize` to split text accurately.

- `wordnet`: Supplies a lexical database for the `WordNetLemmatizer`, which transforms words to their base forms while retaining meaning.

- `stopwords`: Includes a list of common English words (e.g., "the," "and") that can be excluded to focus on meaningful content during text analysis.

Flowchart:



Initializing necessary NLP components:

Running these downloads ensures the necessary resources are available for text tokenization, lemmatization, and stop-word removal.

It looks like you're initializing components for Natural Language Processing (NLP) using Python. Here's a brief overview of what each component does:

1. **Lemmatizer**: The WordNetLemmatizer is used to reduce words to their base or root form (lemma). For example, "running" becomes "run."
2. **Stemmer**: The PorterStemmer reduces words to their base or root form by stripping suffixes. For instance, "running," "runner," and "ran" may all be reduced to "run."
3. **Stop Words**: The stopwords set contains common words in the English language (like "the," "is," "in") that are often filtered out in NLP tasks because they add little meaning to the text.

Function to preprocess text: tokenize, remove stopwords, lemmatize, stem:

1. **Input**: The function takes a single parameter, text, which is a string containing the text to be processed.
2. **Tokenization**:
    - The word_tokenize(text.lower()) function converts the entire text to lowercase and splits it into individual tokens (words).
3. **Remove Stopwords and Punctuation**:
    - The list comprehension [word for word in tokens if word.isalnum() and word not in stop_words] filters out tokens that are not alphanumeric (i.e., it removes punctuation) and excludes common stopwords (like "the," "is," etc.).
4. **Lemmatization**:
    - The line lemmatized_tokens = [lemmatizer.lemmatize(token) for token in tokens] applies lemmatization to each token, reducing words to their base or dictionary form. This helps in standardizing the text.
5. **Stemming**:
    - The line stemmed_tokens = [stemmer.stem(token) for token in lemmatized_tokens] applies stemming to each lemmatized token, further reducing them to their root forms. Stemming often results in words that are not actual dictionary entries but capture the essence of the word (e.g., "running" becomes "run").
6. **Output**:
    - The function returns a single string, ' '.join(stemmed_tokens), which concatenates the processed tokens back into a single string with spaces in between.

Text Preprocessing and Sentiment Labelling Using TextBlob:

Preprocessing and sentiment labelling to a dataset of comments stored in a DataFrame (df). Here's a breakdown of its functionality:

1. **Preprocessing Text**:
    - The line df['processed_text'] = df['text'].apply(preprocess_text) applies a function preprocess_text to the text column of the DataFrame, creating a new column called processed_text that contains the cleaned or transformed text.
2. **Sentiment Analysis**:
    - The label_sentiment function uses the TextBlob library to analyze the sentiment of each comment. It calculates the sentiment polarity (a value between -1 and 1, where values greater than 0 indicate positive sentiment).
    - The function returns a label of 1 for positive sentiments and 0 for negative sentiments.
3. **Labeling Comments**:
    - The commented-out line # df['label'] = df['text'].apply(label_sentiment) would apply the label_sentiment function to the text column, creating a new column called label that categorizes each comment as positive (1) or negative (0).
4. **Displaying Results**:
    - The final line df[['text', 'processed_text', 'label']].head() displays the first few rows of the DataFrame, showing the original text, the processed text, and the sentiment label for each comment.

Overall, this code is used to preprocess text data and label comments based on their sentiment, preparing the dataset for further analysis or modelling.

Steps after Pre-processing and labelling of dataset (18k comments):

1. Loading the Preprocessed and Labelled Dataset

The preprocessed and labelled dataset was saved as a CSV file named final_data18k.csv. This dataset includes:

- **Comments**: Text data representing user comments.
- **Labels**: Binary labels indicating whether a comment is "cyberbullying" (1) or "non-cyberbullying" (0).

The data was loaded into a pandas DataFrame using the following code:

```
import pandas as pd

# Load the dataset
```

```
df = pd.read_csv("final_data18k.csv")

print(df.head())
```

The .info() method was used to ensure the dataset had no missing values, and the distribution of labels was checked using:

```
print("Count of Cyberbullying comments:", sum(df['label'] == 1))

print("Count of Non-Cyberbullying comments:", sum(df['label'] == 0))
```

2. Vectorizing the Data

Machine learning algorithms cannot process textual data directly, so the comments were converted into numerical representations using **TF-IDF Vectorization**:

- **TF (Term Frequency)**: Measures how often a word appears in a document.
- **IDF (Inverse Document Frequency)**: Reduces the weight of common words across multiple documents.

This was implemented using the TfidfVectorizer from scikit-learn:

```
from sklearn.feature_extraction.text import TfidfVectorizer

# Initialize the TF-IDF Vectorizer

tfidf = TfidfVectorizer(max_features=5000)  # Limit to top 5000 words

X = tfidf.fit_transform(df['Comment']).toarray()

y = df['label']
```

The X array contained the numerical representation of comments, and y held the corresponding labels.

3. Splitting the Dataset

The dataset was split into training and testing subsets using an 70-30 split ratio:

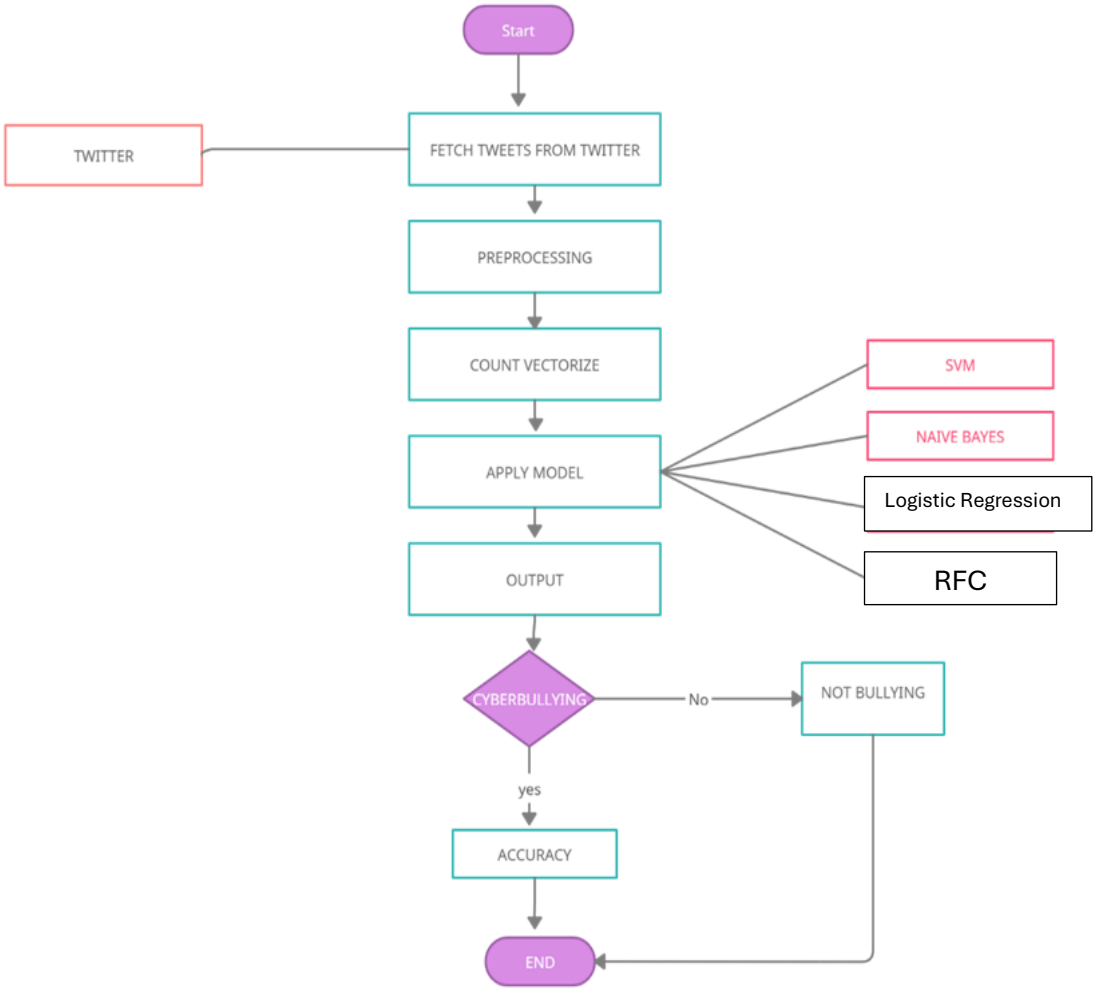
```
from sklearn.model_selection import train_test_split

# Split the dataset

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

- **Training Set**: Used to train the models.
- **Testing Set**: Used to evaluate model performance on unseen data.

This split ensured the models were not overfitted to the training data.



4. Training with Machine Learning Models

Four machine learning models were trained to classify comments:

4.1 Logistic Regression

A logistic regression model was trained using:

```
from sklearn.linear_model import LogisticRegression

logistic_model = LogisticRegression(random_state=42)
logistic_model.fit(X_train, y_train)
```

4.2 Random Forest Classifier

A Random Forest model was trained to make predictions using multiple decision trees:

```
from sklearn.ensemble import RandomForestClassifier

random_forest_model = RandomForestClassifier(random_state=42)
random_forest_model.fit(X_train, y_train)
```

4.3 Multinomial Naive Bayes

A probabilistic approach was applied using Naive Bayes:

```
from sklearn.naive_bayes import MultinomialNB

naive_bayes_model = MultinomialNB()
naive_bayes_model.fit(X_train, y_train)
```

4.4 Support Vector Machine (SVM)

SVM was employed for binary classification with a linear kernel:

```
from sklearn.svm import SVC

svm_model = SVC(kernel='linear', random_state=42)
svm_model.fit(X_train, y_train)
```

Model Evaluation

Each model was evaluated using metrics like accuracy, precision, recall, and F1-score:

```
from sklearn.metrics import accuracy_score, classification_report

# Example for Logistic Regression
y_pred_logistic = logistic_model.predict(X_test)
print("Logistic Regression Accuracy:", accuracy_score(y_test, y_pred_logistic))
print("Classification Report:\n", classification_report(y_test, y_pred_logistic))
```

Confusion matrices were plotted to visualize predictions.

5. Training with Neural Networks

Deep learning models, **LSTM (Long Short-Term Memory)** and **RNN (Recurrent Neural Networks)**, were trained using TensorFlow/Keras. These models are well-suited for sequential data like text.

LSTM Implementation:

```
from keras.models import Sequential
from keras.layers import LSTM, Dense, Embedding

# LSTM Model
model_lstm = Sequential([
    Embedding(input_dim=5000, output_dim=128, input_length=X_train.shape[1]),
    LSTM(128, return_sequences=False),
    Dense(1, activation='sigmoid')
])

model_lstm.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
model_lstm.fit(X_train, y_train, epochs=5, batch_size=64, validation_data=(X_test, y_test))
```

RNN Implementation:

```
from keras.models import Sequential
from keras.layers import RNN, Dense, Embedding

# RNN Model
model_rnn = Sequential([
    Embedding(input_dim=5000, output_dim=128, input_length=X_train.shape[1]),
    RNN(128, return_sequences=False),
    Dense(1, activation='sigmoid')
])

model_rnn.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
model_rnn.fit(X_train, y_train, epochs=5, batch_size=64, validation_data=(X_test, y_test))
```

6. Saving the Model

The best-performing model was saved for deployment:

```
import joblib

# Save the Logistic Regression model
joblib.dump(logistic_model, "logistic_model.pkl")

# Save the LSTM model
model_lstm.save("lstm_model.h5")
```

This allowed reloading and reusing the trained models for predictions without retraining.

7. GUI Development with Flask

A **Flask-based GUI** was developed to make predictions interactively. Users can input comments, and the application predicts whether the comment is cyberbullying or not.

Flask Application Code:

```
from flask import Flask, request, render_template
import joblib

app = Flask(__name__)

# Load the saved model
model = joblib.load("logistic_model.pkl")

@app.route('/')
def home():
    return render_template('index.html')

@app.route('/predict', methods=['POST'])
def predict():
    comment = request.form['comment']
```

```
    vectorized_comment = tfidf.transform([comment]).toarray()
    prediction = model.predict(vectorized_comment)
    result = "Cyberbullying" if prediction[0] == 1 else "Non-Cyberbullying"
    return render_template('result.html', prediction=result)

if __name__ == '__main__':
    app.run(debug=True)
```

Conclusion

This project successfully tackles the problem of **cyberbullying detection** through the following comprehensive steps:

1. **Data Collection and Preprocessing**:
   We began by collecting comments data, ensuring it was cleaned and structured for further analysis. Preprocessing techniques such as tokenization, stopword removal, lemmatization, and stemming were applied to standardize the textual data. Sentiment labelling was then conducted to classify comments into "cyberbullying" or "non-cyberbullying."
2. **Feature Extraction**:
   The textual data was transformed into numerical representations using the **TF-IDF Vectorizer**, allowing machine learning models to process and analyze the comments effectively. This step converted human-readable text into a format that is computationally understandable.
3. **Dataset Splitting**:
   To ensure a fair evaluation of the models, the dataset was split into training and testing sets, with the training data used to build the models and the testing data reserved for performance validation.
4. **Model Training**:
   We employed multiple machine learning algorithms, including **Logistic Regression, Random Forest Classifier, Multinomial Naive Bayes**, and **Support Vector Machine (SVM)**. Each model was trained, evaluated, and compared based on metrics such as accuracy, precision, recall, and F1-score.
5. **Deep Learning Integration**:
   To enhance performance and handle complex sequential data, we trained **Neural Network models** such as **LSTM (Long Short-Term Memory)** and **RNN (Recurrent Neural Networks)**. These models effectively captured the sequential nature of text, offering deeper insights into context and sentiment.
6. **Model Saving**:
   The best-performing models were saved for future use, ensuring they can be deployed

without requiring retraining. This step optimizes efficiency and prepares the models for real-world application.

7. **GUI Development**:
   To make the system accessible and interactive, we developed a user-friendly **Graphical User Interface (GUI)** using Flask. This web application allows users to input comments and receive instant predictions about whether the comment constitutes cyberbullying or not.