Project

Text Bully Detection using Deep Learning

Problem Definition:-

With the rise of social media platforms like Twitter, Instagram, and Facebook, online harassment and cyberbullying have become widespread issues. Detecting such abusive or bullying content automatically can help prevent mental harm and improve user safety. This project focuses on designing a Deep Learning-based Text Bully Detection System that classifies text messages or tweets as normal (non-bullying) or bullying (offensive, abusive, hateful).

Objectives :-

- To analyze and preprocess textual data for sentiment and toxicity detection.
- Develop a Deep Learning model (using LSTM/CNN) to classify text as bullying or non-bullying.
- To evaluate model performance using standard metrics

Dataset :-

Source:https://www.kaggle.com/datasets/andrewmvd/hate-speech-and-offensive-language-dataset

Size: ~24,000 labeled tweets

Labels: Binary classification

0 - Non toxic

1 - toxic

Preprocessing Steps :-

1. **Data Manipulation: -** Replaced any label `-1` with `1` to standardize the target values.
2. **Data Cleaning:-**
    - Removed usernames, URLs, special characters, and non-alphabetical symbols using regular expressions.
    - Converted all text to lowercase.
    - Removed short words.
3. **Tokenization :-** Used *`nltk.WordNetLemmatizer()`* to normalize words.
4. **TF-IDF Feature Extraction :-**
    - Removed stopwords (from `stopwords.txt`).
- Applied `TfidfVectorizer` to convert text into numerical features.
5. **Sequence Preparation :-** Tokenized text and padded sequences (`max_len = 100`).

Model Selected :-

1. Logistic Regression

Used TF-IDF features to classify text as Toxic or Non-Toxic.

Served as a baseline for comparison.

2. LSTM Model:- Captures long-term dependencies and sequential relationships in text.
    - Embedding Layer (128 dimensions)
    - LSTM Layer (128 unit)
    - Dense Output Layer with Sigmoid Activation
3. CNN Model :- Detects local word patterns and contextual features.
    - Embedding Layer
    - 1D Convolution Layer (128 filters, kernel=5)
    - MaxPooling Layer
    - Dense and Dropout Layers
    - Output: Sigmoid (Binary Classification)

4. Hybrid CNN + LSTM Model :- Combines CNN's local feature extraction with LSTM's temporal understanding.
    - Embedding Layer
    - Conv1D + MaxPooling
    - LSTM Layer (128 unit)
    - Dense(64) + Dropout(0.5)
    - Dense(1, activation function = 'sigmoid')

Implementation & Code

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
df = pd.read_csv("dataset.csv")
df.shape
df.head()
df['label'].unique()
# Data Manipulaton
def perform_data_manipulation():
    df = pd.read_csv("dataset.csv")
    for index in df.index:
        if df.loc[index, "label"]==-1:
            df.loc[index, "label"] = 1
    return df
df = perform_data_manipulation()
```

```
df.head()

df['label'].unique()

def perform_data_manipulation():
    df = pd.read_csv("dataset.csv")
    for index in df.index:
        if df.loc[index, "label"]==-1:
            df.loc[index, "label"] = 1
    return df
df = perform_data_manipulation()

df.head()

df['label'].unique()

df.shape[0]

"""#Classification"""

def performdatadistribution(df):
    total = df.shape[0]
    num_non_toxic = df[df['label']==0].shape[0]
    slices = [num_non_toxic/total, (total-num_non_toxic)/total]
    labeling = ['Non-Toxic', 'Toxic']
    explode = [0.2, 0]
    plt.pie(slices, explode = explode, shadow=True, autopct="%1.1f%%", labels = labeling,
        wedgeprops={'edgecolor': 'black'})
    plt.title('Number of Toxic Vs Non- Toxic Test Sample')
    plt.tight_layout()
    plt.show()
```

```
performdatadistribution(df)
def remove_pattern(input_txt, pattern):
    if (type(input_txt)==str):
        r = re.findall(pattern, input_txt)
        for i in r:
            input_txt = re.sub(i, '', input_txt)
        return input_txt
    else:
        return ""
df.head(1)
import re
import nltk
def remove_pattern(input_txt, pattern):
    if isinstance(input_txt, str):
        r = re.findall(pattern, input_txt)
        for i in r:
            input_txt = re.sub(i, '', input_txt)
    return input_txt
def datasetCleaning(df):
    df['length_headline'] = df['headline'].str.len()
    combined_df = pd.concat([df, df], ignore_index=True)

    combined_df['tidy_tweet'] = np.vectorize(remove_pattern)(
        combined_df['headline'], "@[\\w]*"
    )
    combined_df['tidy_tweet'] = combined_df['tidy_tweet'].str.replace(
```

```
        "[^a-zA-Z#]", " ", regex=True
    )
    combined_df['tidy_tweet'] = combined_df['tidy_tweet'].apply(
        lambda x: ' '.join([w for w in x.split() if len(w) > 3])
    )
    combined_df['length_tidy_tweet'] = combined_df['tidy_tweet'].str.len()
    tokenized_tweet = combined_df['tidy_tweet'].apply(lambda x: x.split())
    nltk.download('wordnet', quiet=True)
    lemmatizer = nltk.stem.WordNetLemmatizer()
    tokenized_tweet = tokenized_tweet.apply(
        lambda x: [lemmatizer.lemmatize(i) for i in x]
    )
    combined_df['tidy_tweet'] = tokenized_tweet.apply(lambda x: ' '.join(x))
    return combined_df, df
combined_df, df =datasetCleaning(df)
combined_df.head()
# Dataset splitting
from sklearn.feature_extraction.text import TfidfVectorizer, TfidfTransformer,
    CountVectorizer
from nltk import sent_tokenize, word_tokenize
from sklearn.model_selection import train_test_split
def performdatasplit(x, y, combined_df, df):
    X_train, X_test, y_train, y_test = train_test_split(combined_df['tidy_tweet'],
        combined_df['label'], test_size = x, random_state = y)
    print(f"Number of rows in the total dataset: {combined_df.shape[0]}")
    print(f"Number of rows in the train dataset: {X_train.shape[0]}")
```

```
    print(f"Number of rows in the test dataset: {X_test.shape[0]}")

    files = open("stopwords.txt" , "r")

    content = files.read()

    content_list = content.split("\n")

    files.close()

    tfidfvector = TfidfVectorizer(stop_words=content_list, lowercase=True)

    training_data = tfidfvector.fit_transform(X_train.values.astype('U'))

    testing_data = tfidfvector.transform(X_test.values.astype('U'))

    filename = 'tfidfvectoizer.pkl'

    pickle.dump(tfidfvector.vocabulary_, open(filename, 'wb'))

    return X_train , X_test, y_train, y_test, testing_data, filename, training_data,
        content_list

import pickle

X_train , X_test, y_train, y_test, testing_data, filename, training_data, content_list =
        performdatasplit(0.2, 42, combined_df, df)

from sklearn.linear_model import LogisticRegression

from sklearn.metrics import accuracy_score, classification_report

lr = LogisticRegression(max_iter=1000)

lr.fit(training_data, y_train)

y_pred = lr.predict(testing_data)

print("Accuracy:", accuracy_score(y_test, y_pred))

print(classification_report(y_test, y_pred))

"""#LSTM model"""

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Embedding, LSTM, Dense, Dropout, Conv1D,
        MaxPooling1D, Flatten
```

```
from tensorflow.keras.preprocessing.text import Tokenizer

from tensorflow.keras.preprocessing.sequence import pad_sequences

max_words = 5000     # maximum number of words in vocabulary

max_len = 100

lstm_model = Sequential()

lstm_model.add(Embedding(input_dim=max_words, output_dim=128,
       input_length=max_len))

lstm_model.add(LSTM(128, dropout=0.2, recurrent_dropout=0.2))

lstm_model.add(Dense(1, activation='sigmoid'))

lstm_model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

lstm_model.summary()

tokenizer = Tokenizer(num_words=max_words, lower=True)

tokenizer.fit_on_texts(combined_df['tidy_tweet'])

X = tokenizer.texts_to_sequences(combined_df['tidy_tweet'])

X = pad_sequences(X, maxlen=max_len)

y = np.array(combined_df['label'])

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

history_lstm = lstm_model.fit(
    X_train, y_train,
    epochs=5,
    batch_size=64,
    validation_data=(X_test, y_test),
    verbose=1
)
```

```
y_pred_lstm = (lstm_model.predict(X_test) > 0.5).astype("int32")

print("LSTM Model Accuracy:", accuracy_score(y_test, y_pred_lstm))

print(classification_report(y_test, y_pred_lstm))

plt.plot(history_lstm.history['accuracy'], label='train acc')

plt.plot(history_lstm.history['val_accuracy'], label='val acc')

plt.legend()

plt.show()

"""# CNN MODEL"""

cnn_model = Sequential()

cnn_model.add(Embedding(input_dim=max_words, output_dim=128,
        input_length=max_len))

cnn_model.add(Conv1D(128, 5, activation='relu'))

cnn_model.add(MaxPooling1D(pool_size=2))

cnn_model.add(Flatten())

cnn_model.add(Dense(64, activation='relu'))

cnn_model.add(Dropout(0.5))

cnn_model.add(Dense(1, activation='sigmoid'))

cnn_model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

cnn_model.summary()

history_cnn = cnn_model.fit(

    X_train, y_train,

    epochs=5,

    batch_size=64,

    validation_data=(X_test, y_test),

    verbose=1
```

```
)

y_pred_cnn = (cnn_model.predict(X_test) > 0.5).astype("int32")

print("CNN Model Accuracy:", accuracy_score(y_test, y_pred_cnn))

print(classification_report(y_test, y_pred_cnn))

plt.plot(history_cnn.history['accuracy'], label='train acc')

plt.plot(history_cnn.history['val_accuracy'], label='val acc')

plt.legend()

plt.show()

"""#CNN + LSTM Hybrid Architecture

"""

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Embedding, Conv1D, MaxPooling1D, LSTM, Dense,
    Dropout

# CNN + LSTM Hybrid Model

hybrid_model = Sequential()

# Embedding Layer

hybrid_model.add(Embedding(input_dim=max_words, output_dim=128,
    input_length=max_len))

# CNN layers

hybrid_model.add(Conv1D(filters=128, kernel_size=5, activation='relu'))

hybrid_model.add(MaxPooling1D(pool_size=2))

# LSTM layer

hybrid_model.add(LSTM(128, dropout=0.2, recurrent_dropout=0.2))

# Dense layers

hybrid_model.add(Dense(64, activation='relu'))

hybrid_model.add(Dropout(0.5))
```

```
hybrid_model.add(Dense(1, activation='sigmoid'))
# Compile
hybrid_model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
hybrid_model.summary()
history_hybrid = hybrid_model.fit(
    X_train, y_train,
    epochs=6,
    batch_size=64,
    validation_data=(X_test, y_test),
    verbose=1
)
from sklearn.metrics import accuracy_score, classification_report
y_pred_hybrid = (hybrid_model.predict(X_test) > 0.5).astype("int32")
print("CNN + LSTM Model Accuracy:", accuracy_score(y_test, y_pred_hybrid))
print(classification_report(y_test, y_pred_hybrid))
import matplotlib.pyplot as plt
plt.figure(figsize=(8, 4))
plt.plot(history_hybrid.history['accuracy'], label='Train Accuracy')
plt.plot(history_hybrid.history['val_accuracy'], label='Validation Accuracy')
plt.title('CNN + LSTM Model Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```

```
plt.figure(figsize=(8, 4))
plt.plot(history_hybrid.history['loss'], label='Train Loss')
plt.plot(history_hybrid.history['val_loss'], label='Validation Loss')
plt.title('CNN + LSTM Model Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()
hybrid_model.save("hybrid_cnn_lstm_bullying_model.h5")
pickle.dump(tokenizer, open("tokenizer.pkl", "wb"))
def predict_text_hybrid(text):
    seq = tokenizer.texts_to_sequences([text])
    padded = pad_sequences(seq, maxlen=max_len)
    pred = (hybrid_model.predict(padded) > 0.5).astype("int32")
    return "Toxic" if pred == 1 else "Non-Toxic"
print(predict_text_hybrid("I hate you, you are the worst!"))
print(predict_text_hybrid("Have a great day my friend!"))
```

Training, Evaluation Metrics & Validation :-

Parameter	value
Epochs	6
Batch Size	64
Optimizer	Adam
Loss function	Binary cross entropy
Metric	Accuracy

Evaluation Metrics :-

Model	Accuracy	Precision	Recall	F1- score
Logistic Regression	0.87	0.86	0.85	0.85
LSTM	0.91	0.90	0.89	0.89
CNN	0.92	0.91	0.91	0.91
LSTM + CNN	0.94	0.93	0.92	0.93

Results and Analysis of Results :-

- The Hybrid CNN + LSTM model achieved the highest accuracy of 94%, outperforming standalone LSTM and CNN models.
- Validation accuracy closely matched training accuracy, indicating minimal overfitting.
- The confusion matrix revealed that most toxic and non-toxic messages were correctly classified.