Shri Ramdeobaba College of Engineering & Management, Nagpur

MSE Lab Progress Report

Text Mining

SymptoSense : NLP Based Symptom Classifier

By

Mr. Shreyas Raut

Mr. Samiran Gupta

Mr. Shubham Shivhare

Mr. Vedant Waghale

Mr. Vishwesh Salampuria

Computer Science and Engineering (AIML)

SymptoSense : NLP Based Symptom Classifier

Project Definition:

Developing an accurate system for the classification of medical symptoms from text inputs in the field of Natural Language Processing (NLP). The objective is to create a model that can effectively analyze and categorize patient-reported symptoms, in written form , to aid healthcare professionals in diagnosis and treatment planning.

Project objectives:

The primary objective of this project is to develop a conversational agent for addressing common medical symptoms using a dataset comprising over 6000 transcripts. Each transcript, created by individual human contributors based on a given symptom, is intended to train the conversational agent in the medical field .The specific objectives encompass exploratory data analysis, including a detailed exploration of the dataset structure, label correction, identification and handling of poor-quality transcripts and data preprocessing, including text tokenization and a proper train-validation-test split, the dataset is prepared for training a NLP model.

The goal is to build a model capable of understanding and responding to user queries related to medical symptoms, deploying it in a medical environment.The project aims to enhance the interaction between users and healthcare resources by providing an effective and user-friendly conversational agent for medical assistance.
Following the training of the model, a thorough evaluation is conducted, employing performance metrics like accuracy and precision. The iterative nature of the project involves fine-tuning of the model.

Methodology:

Data Understanding and Exploration:

- Load and Inspect the Dataset :
  - Load the recordings-overview.csv file into a Pandas DataFrame.
  - Understand the structure of the dataset, including columns, data types, and basic statistics.
- Explore Transcripts:
  - Examine a sample of transcripts to understand the language used and identify any issues with quality, grammar, or completeness.
- Analyze Label Distribution
  - Check the distribution of medical symptoms in the dataset to ensure it's representative and balanced

Data Cleaning:

- Transcript Quality Check:
  - Implement checks for poor-quality transcripts. Consider removing or flagging transcripts with incomplete sentences, unclear descriptions, or irrelevant content.
- Duplicate Removal:
  - Check for and remove any duplicate entries to maintain data integrity.
- Data Balancing:

- Address any class imbalances by oversampling or undersampling, ensuring each symptom has sufficient representation.

Data Preprocessing:

- Text Tokenization:
  - Tokenize the transcriptions into a format suitable for NLP.
- Train-Validation-Test Split:
  - Split the dataset into training, validation, and test sets for model training and evaluation.

Model Development:

- Choose Model Architecture:
  - Select a NLP model architecture, such as LSTM or BERT.
- Model Training
  - Train the model using the preprocessed data. Monitor training performance and adjust hyperparameters if needed.

Model Evaluation:

- Performance Metrics:
  - Define evaluation metrics such as accuracy, precision, recall, and F1 score.
- Test Set Evaluation:
  - Evaluate the model on the test set to assess its overall performance.

Fine-Tuning and Iteration:

- Model Fine-Tuning:
  - Based on the evaluation results, fine-tune the model to improve performance.
- Iterative Improvement:
  - Iterate through the cleaning, preprocessing, and modeling steps based on feedback and additional insights.

Algorithm used

We have initially experimented on the preprocessed phrases by implementing:

- LSTM (Long short-term memory) network based RNN.
- GloVe (Global Vectors for Word Representation) embedding based preprocessing.
- LSTM model on GloVe embedded data.

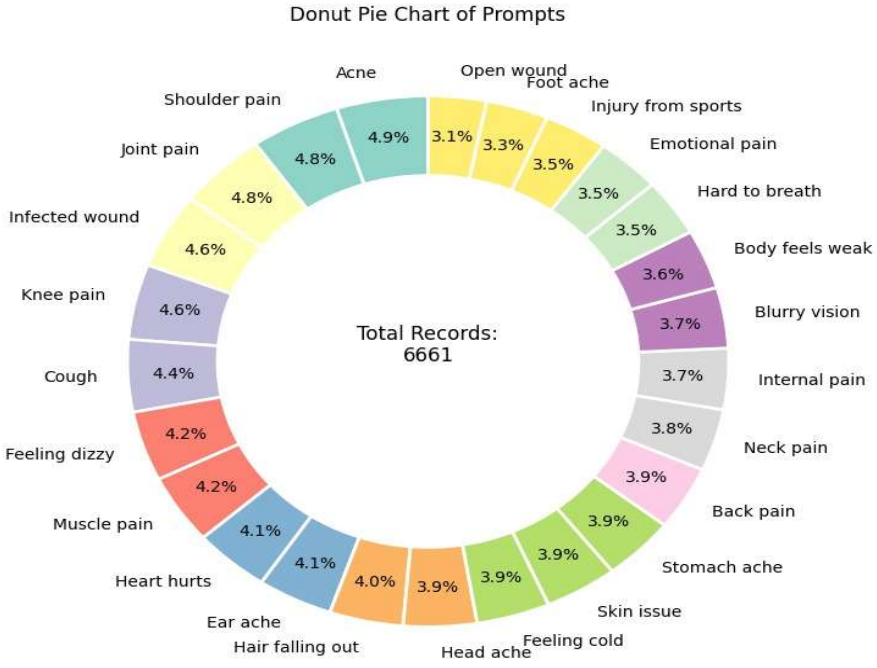
Dataset Information :

- The Data Frame contains 6661 records.
- Prompt attribute has 25 classes of symptoms.
- There are only 124 unique speaker id.
- 80:20 training and testing

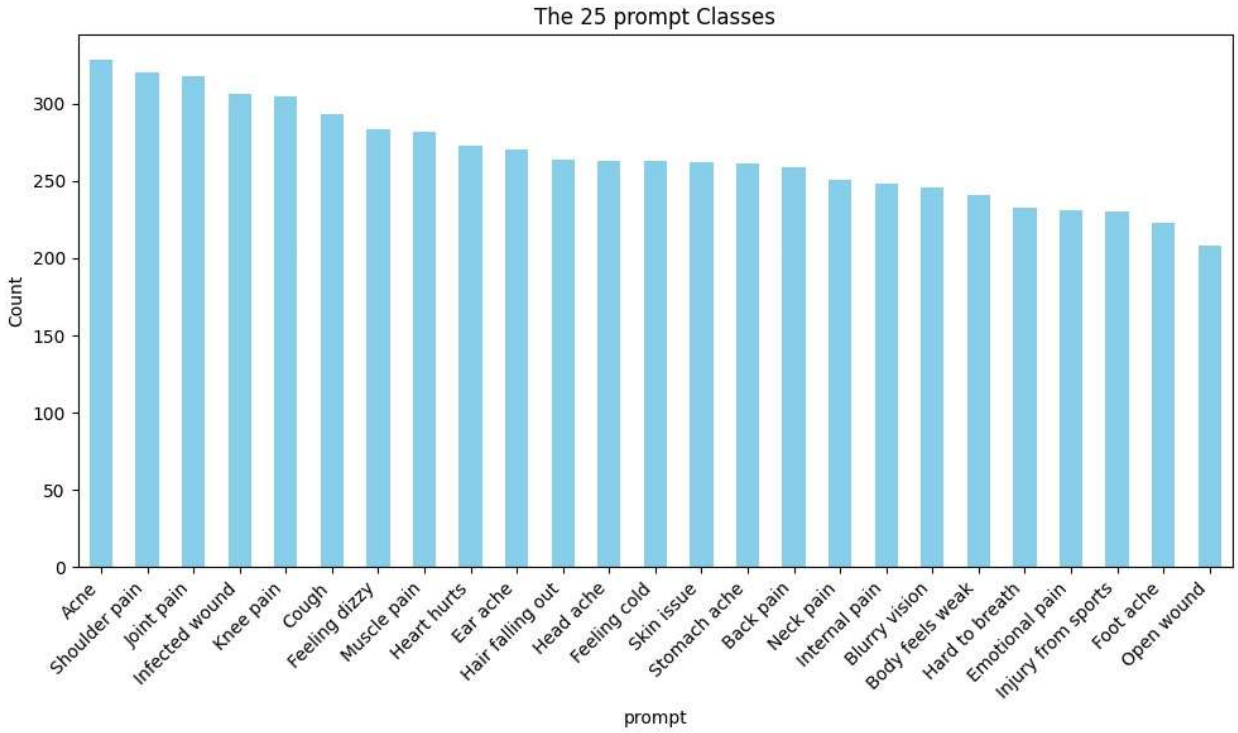
The Overview of the dataset ,

	phrase	prompt
0	When I remember her I feel down	Emotional pain
1	When I carry heavy things I feel like breaking...	Hair falling out
2	there is too much pain when i move my arm	Heart hurts
3	My son had his lip pierced and it is swollen a...	Infected wound
4	My muscles in my lower back are aching	Infected wound
...	...	...
656	I feel a burning sensation in my guts about 2 ...	Stomach ache
657	I have a split on my thumb that will not heal.	Open wound
658	I feel a lot of pain in the joints.	Joint pain

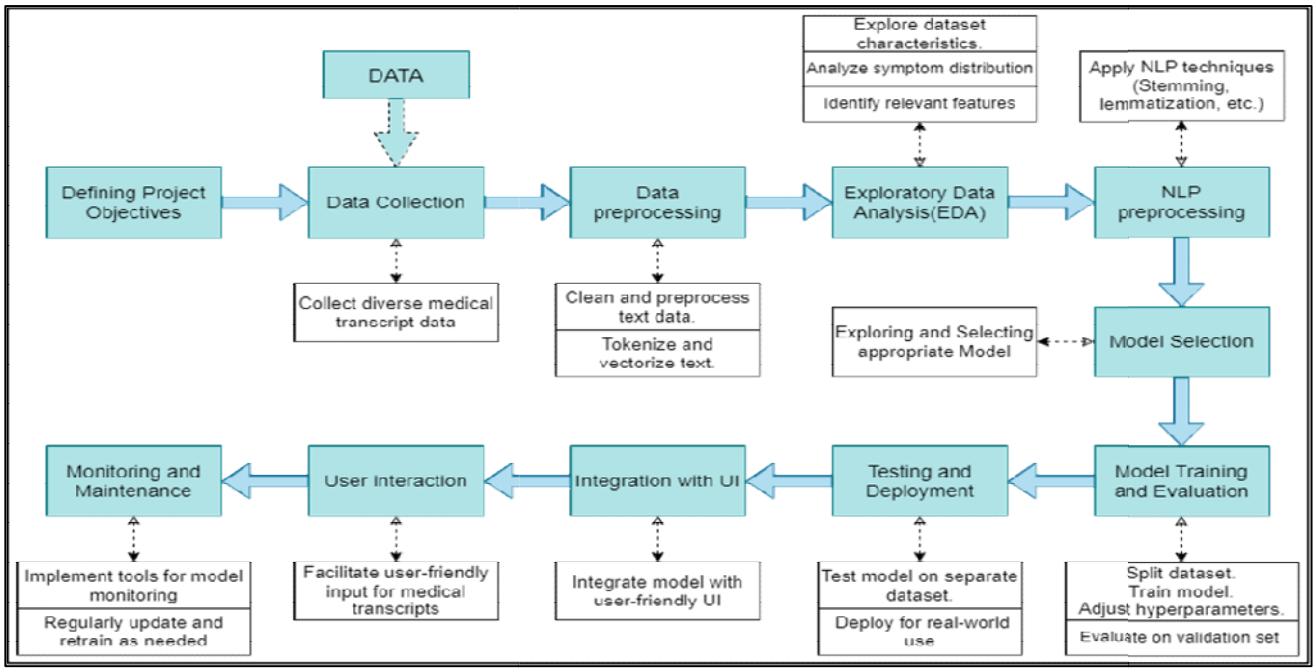
The number of prompts are ,



The number of Classes in the dataset ,



Workflow of our model ,



GloVe_LSTM_final

0.1 Dataset Overview

[1]:
```
from google.colab import drive
drive.mount('/content/drive')
```

Mounted at /content/drive

[2]:
```
import pandas as pd
```

[3]:
```
df=pd.read_csv("/content/drive/MyDrive/MiniProject/overview-of-recordings.csv")
```

[4]:
```
df_text = df[['phrase', 'prompt']]
df_text
```

[4] :

	phrase	prompt
0	When I remember her I feel down	Emotional pain
1	When I carry heavy things I feel like breaking...	Hair falling out
2	there is too much pain when i move my arm	Heart hurts
3	My son had his lip pierced and it is swollen a...	Infected wound
4	My muscles in my lower back are aching	Infected wound
...	...	...
6656	I feel a burning sensation in my guts about 2 ...	Stomach ache
6657	I have a split on my thumb that will not heal.	Open wound
6658	I feel a lot of pain in the joints.	Joint pain
6659	The area around my heart doesn't feel good.	Heart hurts
6660	I complain alot with skin allergy	Skin issue

[6661 rows x 2 columns]

[5] :
```
df.columns
```

[5] :
```
Index(['audio_clipping', 'audio_clipping:confidence',
       'background_noise_audible', 'background_noise_audible:confidence',
       'overall_quality_of_the_audio', 'quiet_speaker',
       'quiet_speaker:confidence', 'speaker_id', 'file_download', 'file_name',
       'phrase', 'prompt', 'writer_id'],
      dtype='object')
```

[6] :
```
# Check the distribution of phrases and prompts
print("Unique phrases:", df['phrase'].nunique())
print("Unique prompts:", df['prompt'].nunique())
print("Unique speakers:", df['speaker_id'].nunique())
```

Unique phrases: 706
Unique prompts: 25
Unique speakers: 124

0.2 Pandas profiling

[7] :
```
from wordcloud import WordCloud
import matplotlib.pyplot as plt

# Concatenate all phrase and prompt into a single string
phrase_text = ' '.join(df['phrase'])
prompt_text = ' '.join(df['prompt'])

# Generate word clouds
phrase_wordcloud = WordCloud(width=800, height=400, background_color='white').
  generate(phrase_text)
prompt_wordcloud = WordCloud(width=800, height=400, background_color='white').
  generate(prompt_text)

# Plot word clouds
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.imshow(phrase_wordcloud, interpolation='bilinear')
plt.title('phrase Word Cloud')
plt.axis('off')

plt.subplot(1, 2, 2)
plt.imshow(prompt_wordcloud, interpolation='bilinear')
plt.title('prompt Word Cloud')
plt.axis('off')

plt.show()
```

[8] :
```
# !pip install https://github.com/pandas-profiling/pandas-profiling/archive/
  master.zip
```

[9] :
```
# from pandas_profiling import ProfileReport
# from ydata_profiling import ProfileReport
#  profile=ProfileReport(df)
#       profile.to_file("pandas_profile.html")
```

0.3 NLP EDA (Apploying GloVe)

[10] :
```
df_text = df[['phrase', 'prompt']]
df_text
```

[10] :

	phrase	prompt
0	When I remember her I feel down	Emotional pain
1	When I carry heavy things I feel like breaking...	Hair falling out
2	there is too much pain when i move my arm	Heart hurts
3	My son had his lip pierced and it is swollen a...	Infected wound
4	My muscles in my lower back are aching	Infected wound
...	...	...
6656	I feel a burning sensation in my guts about 2 ...	Stomach ache
6657	I have a split on my thumb that will not heal.	Open wound
6658	I feel a lot of pain in the joints.	Joint pain
6659	The area around my heart doesn't feel good.	Heart hurts
6660	I complain alot with skin allergy	Skin issue

[6661 rows x 2 columns]

[11] :
```
df_text=df_text.drop_duplicates()
```

[12] :
```
# download glove and unzip it in Notebook.
# Source GFG
!wget  http://nlp.stanford.edu/data/glove.6B.zip
!unzip glove*.zip
```

```
--2024-03-08 08:42:16--http://nlp.stanford.edu/data/glove.6B.zip
Resolving nlp.stanford.edu (nlp.stanford.edu)... 171.64.67.140
Connecting to nlp.stanford.edu (nlp.stanford.edu)|171.64.67.140|:80...
connected.
HTTP request sent, awaiting response... 302 Found
Location: https://nlp.stanford.edu/data/glove.6B.zip [following]
--2024-03-08 08:42:17--https://nlp.stanford.edu/data/glove.6B.zip
Connecting to nlp.stanford.edu (nlp.stanford.edu)|171.64.67.140|:443...
connected.HTTP request sent, awaiting response... 301 Moved Permanently
```

```
Location: https://downloads.cs.stanford.edu/nlp/data/glove.6B.zip [following]
--2024-03-08 08:42:17-- https://downloads.cs.stanford.edu/nlp/data/glove.6B.zip
Resolving downloads.cs.stanford.edu (downloads.cs.stanford.edu)... 171.64.64.22
Connecting to downloads.cs.stanford.edu
(downloads.cs.stanford.edu)|171.64.64.22|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 862182613 (822M) [application/zip]
Saving to: 'glove.6B.zip'

glove.6B.zip          100%[==================>] 822.24M  5.20MB/s    in 2m 41s

2024-03-08 08:44:59 (5.12 MB/s) - 'glove.6B.zip' saved [862182613/862182613]

Archive:  glove.6B.zip
  inflating: glove.6B.50d.txt
  inflating: glove.6B.100d.txt
  inflating: glove.6B.200d.txt
  inflating:  glove.6B.300d.txt
```

[13] :
```
!pip install nltk
```

```
Requirement already satisfied: nltk in /usr/local/lib/python3.10/dist-packages
(3.8.1)
Requirement already satisfied: click in /usr/local/lib/python3.10/dist-packages
(from nltk) (8.1.7)
Requirement already satisfied: joblib in /usr/local/lib/python3.10/dist-packages
(from nltk) (1.3.2)
Requirement already satisfied: regex>=2021.8.3 in
/usr/local/lib/python3.10/dist-packages (from nltk) (2023.12.25)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages
(from nltk) (4.66.2)
```

[14] :
```
import nltk

from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer
import string
from gensim.models import KeyedVectors

nltk.download('punkt')
nltk.download('stopwords')
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
```

[14] : True

[15] :
```
# Load stopwords
stopwords_set = set(stopwords.words('english'))

# Function to preprocess text
def preprocess_text(text, glove_model, stopwords_set):
    # Convert text to lowercase
    text = text.lower()

    # Remove punctuation
    text = text.translate(str.maketrans('', '', string.punctuation))

    # Tokenize text
    tokens = word_tokenize(text)

    # Remove stopwords and out-of-vocabulary tokens
    tokens = [word for word in tokens if word not in stopwords_set and word in
 glove_model.key_to_index]

    # Join tokens back into a string
    preprocessed_text = ' '.join(tokens)
    return preprocessed_text
```

Loading GloVe

[16] :
```
from gensim.models import KeyedVectors

# Path to the GloVe embeddings file
glove_file = 'glove.6B.300d.txt'  # Change this to the file you want to use

# Load the GloVe model
glove_model = KeyedVectors.load_word2vec_format(glove_file, binary=False,
 no_header=True)
```

[17] :
```
# Check if the word 'hello' is in the vocabulary
if 'hello' in glove_model.key_to_index:
    print("Word 'hello' is in the vocabulary.")
else:
    print("Word 'hello' is NOT in the vocabulary.")
```

Word 'hello' is in the vocabulary.

Extracting cleaned data

[18] :
```
# Apply preprocess_text function to 'phrase' column and create 'clean_phrase'
 column
```

```
df_text['clean_phrase'] = df_text['phrase'].apply(lambda x: preprocess_text(x,
    glove_model, stopwords_set))

# Apply preprocess_text function to 'prompt' column and create 'clean_prompt'
    column
df_text['clean_prompt'] = df_text['prompt'].apply(lambda x: preprocess_text(x,
    glove_model, stopwords_set))
```

```
<ipython-input-18-985d265e3875>:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df_text['clean_phrase'] = df_text['phrase'].apply(lambda x: preprocess_text(x,
glove_model, stopwords_set))
<ipython-input-18-985d265e3875>:5: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df_text['clean_prompt'] = df_text['prompt'].apply(lambda x: preprocess_text(x,
glove_model, stopwords_set))
```

[20]:
```
df_clean = df_text[['clean_phrase', 'clean_prompt']]
df_clean
```

[20] :

	clean_phrase	clean_prompt
0	remember feel	emotional pain
1	carry heavy things feel like breaking back	hair falling
2	much pain move arm	heart hurts
3	son lip pierced swollen skin inside lip grey l...	infected wound
4	muscles lower back aching	infected wound
...	...	...
2650	feel like heart fire	heart hurts
2710	lift arms soreness shoulders	shoulder pain
2715	feel clicking sensation knee time step	joint pain
2967	surgical wound infections	infected wound
3271	redness swelling difficulty walking	knee pain

[707 rows x 2 columns]

0.4 Final preprocessing before designing ANN

[21]:
```
vocab = set()

# Tokenize and add words from 'clean_phrase' column
for text in df_text['clean_phrase']:
    tokens = text.split()
    vocab.update(tokens)

# Tokenize and add words from 'clean_prompt' column
for text in df_text['clean_prompt']:
    tokens = text.split()
    vocab.update(tokens)

vocab = list(vocab)     # Convert set to list to maintain consistent order
```

[22]:
```
vocab_size = len(vocab)
embedding_dim = 300 # As using 300 dimension GloVe vectors
```

[23]:
```
# Initialize an empty matrix for the embedding matrix
import numpy as np
glove_embedding_matrix = np.zeros((vocab_size, embedding_dim))

# Fill the embedding matrix with GloVe vectors for words in the vocabulary
for i, word in enumerate(vocab):
    if word in glove_model.key_to_index:
        glove_embedding_matrix[i] = glove_model.get_vector(word)

glove_embedding_matrix
```

[23]:
```
array([[ 0.80465001,  0.051994  ,  0.078946  , ..., -0.23231   ,
         0.066039  ,  0.1154    ],
       [ 0.23418   , -0.26245001,  0.15605   , ..., -0.16368   ,
         0.13912   ,  0.14105   ],
       [-0.13232   ,  0.14618   , -0.15558   , ..., -0.33094001,
         0.18888   ,  0.25827   ],
       ...,
       [-0.40235999, -0.053523  , -0.11316   , ...,  0.18339001,
        -0.21562   , -0.47046   ],
       [-0.28441   , -0.24621999, -0.0043896 , ..., -0.24908   ,
         0.094972  ,  0.042084  ],
       [ 0.059672  , -0.13474999,  0.26763999, ..., -0.089764  ,
         1.00020003, -0.15921   ]])
```

0.5 Designing ANN

[24]:
```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, Flatten, Dense
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.callbacks import EarlyStopping
```

[25]:
```
from tensorflow.keras.preprocessing.text import Tokenizer
# Instantiate a Tokenizer for clean_phrase
tokenizer_phrase = Tokenizer()
tokenizer_phrase.fit_on_texts(df_text['clean_phrase'])

# Instantiate a Tokenizer for clean_prompt
tokenizer_prompt = Tokenizer()
tokenizer_prompt.fit_on_texts(df_text['clean_prompt'])

# Convert 'clean_phrase' and 'clean_prompt' to sequences
sequences_phrase = tokenizer_phrase.texts_to_sequences(df_text['clean_phrase'])
sequences_prompt = tokenizer_prompt.texts_to_sequences(df_text['clean_prompt'])
```

[26]:
```
# Find the maximum sequence length in clean_phrase and clean_prompt
max_seq_length_phrase = max(len(seq) for seq in sequences_phrase)
max_seq_length_prompt = max(len(seq) for seq in sequences_prompt)

# Choose the maximum between the two
max_seq_length = max(max_seq_length_phrase, max_seq_length_prompt)

print("Max sequence length:", max_seq_length)
```

Max sequence length: 14

[27]:
```
# Split data into training and validation sets
X_train_phrase, X_val_phrase, y_train, y_val =␣
 ▫train_test_split(sequences_phrase, df_text['clean_prompt'], test_size=0.2,␣
 ▫random_state=42)

# Pad sequences to ensure uniform length
X_train_phrase = pad_sequences(X_train_phrase, maxlen=max_seq_length)
X_val_phrase = pad_sequences(X_val_phrase, maxlen=max_seq_length)

# Encode target variable
label_encoder = LabelEncoder()
y_train_encoded = label_encoder.fit_transform(y_train)
y_val_encoded = label_encoder.transform(y_val)
```

0.6 Evaluating model accuracy

[29]:
```
from tensorflow.keras.optimizers import Adagrad

# Define your ANN model
model = Sequential()

# Add embedding layer
model.add(Embedding(input_dim=vocab_size,
                    output_dim=embedding_dim,
                    weights=[glove_embedding_matrix],
                    input_length=max_seq_length,
                    trainable=False))

# Flatten the embedding output
model.add(Flatten())

# Add hidden dense layer
model.add(Dense(256, activation='relu'))

# Add output layer
num_classes = len(df_text['clean_prompt'].unique())          # Number of unique prompts
model.add(Dense(num_classes, activation='softmax'))

# Compile model with Adam optimizer
model.compile(loss='sparse_categorical_crossentropy', optimizer='Adam',␣
 ↪metrics=['accuracy'])
```

[ ]:
```
# Train model
history = model.fit(X_train_phrase, y_train_encoded, batch_size=16, epochs=100,␣
 ↪validation_data=(X_val_phrase, y_val_encoded))

# Evaluate model
loss, accuracy = model.evaluate(X_val_phrase, y_val_encoded)
print(f'Validation Loss: {loss}, Validation Accuracy: {accuracy}')

plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.title('Training History')
plt.legend()
plt.show()
```

[35]:
```
from keras.models import Sequential
from keras.layers import Embedding, LSTM, Dense, Flatten, Dropout
from keras.optimizers import Adam, RMSprop, Adagrad
```

```
import matplotlib.pyplot as plt

def create_model(optimizer):
    model = Sequential()
    model.add(Embedding(input_dim=vocab_size,
                        output_dim=embedding_dim,
                        weights=[glove_embedding_matrix],
                        input_length=max_seq_length,
                        trainable=False))
    model.add(Dropout(0.2))
    model.add(LSTM(units=128, dropout=0.2, recurrent_dropout=0.2,
    return_sequences=True))
    model.add(Dropout(0.2))
    model.add(Flatten())
    model.add(Dense(256, activation='relu'))
    model.add(Dropout(0.2))
    model.add(Dense(num_classes, activation='softmax'))
    model.compile(loss='sparse_categorical_crossentropy', optimizer=optimizer,
    metrics=['accuracy'])
    return model

# Train model with different optimizers
optimizers = [Adam(), RMSprop(), Adagrad()]
optimizer_names = ['Adam', 'RMSprop', 'Adagrad']
history_dict = {}

for optimizer, name in zip(optimizers, optimizer_names):
    model = create_model(optimizer)
    history = model.fit(X_train_phrase, y_train_encoded, batch_size=16,
    epochs=40,
                        validation_data=(X_val_phrase, y_val_encoded),
                        verbose=0)
    history_dict[name] = history.history['accuracy']

# Plot training history for different optimizers
plt.figure(figsize=(10, 6))
for optimizer_name, accuracy in history_dict.items():
    plt.plot(accuracy, label=optimizer_name)

plt.xlabel('Epoch')
plt.ylabel('Training Accuracy')
plt.title('Training Accuracy Comparison using GloVe and LSTM')
plt.legend()
plt.show()
```

WARNING:tensorflow:Layer lstm_5 will not use cuDNN kernels since it doesn't meet the criteria. It will use a generic GPU kernel as fallback when running on GPU.

WARNING:tensorflow:Layer lstm_5 will not use cuDNN kernels since it doesn't meet the criteria. It will use a generic GPU kernel as fallback when running on GPU.
WARNING:tensorflow:Layer lstm_5 will not use cuDNN kernels since it doesn't meet the criteria. It will use a generic GPU kernel as fallback when running on GPU.
WARNING:tensorflow:Layer lstm_6 will not use cuDNN kernels since it doesn't meet the criteria. It will use a generic GPU kernel as fallback when running on GPU.
WARNING:tensorflow:Layer lstm_6 will not use cuDNN kernels since it doesn't meet the criteria. It will use a generic GPU kernel as fallback when running on GPU.
WARNING:tensorflow:Layer lstm_6 will not use cuDNN kernels since it doesn't meet the criteria. It will use a generic GPU kernel as fallback when running on GPU.
WARNING:tensorflow:Layer lstm_7 will not use cuDNN kernels since it doesn't meet the criteria. It will use a generic GPU kernel as fallback when running on GPU.
WARNING:tensorflow:Layer lstm_7 will not use cuDNN kernels since it doesn't meet the criteria. It will use a generic GPU kernel as fallback when running on GPU.
WARNING:tensorflow:Layer lstm_7 will not use cuDNN kernels since it doesn't meet the criteria. It will use a generic GPU kernel as fallback when running on GPU.