UNIVERSITY SCHOOL OF AUTOMATION & ROBOTICS (USAR)

Guru Gobind Singh Indraprastha University, East Delhi Campus, Surajmal Vihar, Delhi 110092

NATUAL LANGUAGE PROCESSING LAB

(ARD 352)

Submitted to:

Ms. Ritu Kalonia

Asst. Professor, USAR

Submitted By:

Name: Ujjawal Kumar Singh

Batch: AI-DS B1

Enrollment No: 06519011921

INDEX

S.No	Lab	Date	Teacher Remark
1.	Introduction to NLTK		
2.	Implement morphological parser to accept andreject given string.		
3.	Implement stemming and lemmatization for acorpus.		
4.	Perform and analyse POS tagging -HMM		
5.	Implement the viterbi algorithm Using pythonor NLTK		
6.	Implement a bigram model using 3 sentences inpython orNLTK		
7.	Text classification using Naive Bayes Classifier		
8.	Sentiment analysis using SVM		
9.	Design of ANN for email spam classification.		
10.	Mini Project Based Application		

LAB1 Introduction to NLTK

NLTK is Python's API library for performing an array of tasks in human language. It can perform a variety of operations on textual data, such as classification, tokenization, stemming, tagging, Leparsing, semantic reasoning, etc.

Installation: NLTK can be installed simply using pip or by running the following code.

```
! 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) (1.4.0)
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)
```

Accessing Additional Resources:

To incorporate the usage of additional resources, such as recourses of languages other than English – you can run the following in a python script. It has to be done only once when you are running it for the first time in your system.

```
import nltk
nltk.download('all')
```

```
[nltk_data]   |   Unzipping corpora/
stopwords.zip. [nltk_data]  | Downloading
package subjectivity to [nltk_data]  |
/root/nltk_data...
[nltk_data]   |   Unzipping corpora/subjectivity.zip.
[nltk_data]   | Downloading package swadesh to /root/
nltk_data... [nltk_data]    |       Unzipping corpora/
swadesh.zip.
[nltk_data]   | Downloading package switchboard to /root/
nltk_data... [nltk_data]    |       Unzipping corpora/
switchboard.zip.
[nltk_data]   | Downloading package tagsets to /root/
nltk_data... [nltk_data]    |       Unzipping help/
tagsets.zip.
[nltk_data]   | Downloading package timit to /root/nltk_data...
[nltk_data]   |   Unzipping corpora/timit.zip.
[nltk_data]   | Downloading package toolbox to /root/
nltk_data... [nltk_data]    |       Unzipping corpora/
toolbox.zip.
[nltk_data]   | Downloading package treebank to /root/
nltk_data... [nltk_data]    |       Unzipping corpora/
treebank.zip.
[nltk_data]   | Downloading package twitter_samples to
[nltk_data]   |     /root/nltk_data...
[nltk_data]   |   Unzipping corpora/twitter_samples.zip.
[nltk_data]   | Downloading package udhr to /root/
nltk_data... [nltk_data]    |       Unzipping corpora/
udhr.zip.
[nltk_data]   | Downloading package udhr2 to /root/nltk_data...
[nltk_data]   |   Unzipping corpora/udhr2.zip.
[nltk_data]   | Downloading package unicode_samples
to [nltk_data]      |       /root/nltk_data...
[nltk_data]   |   Unzipping corpora/
unicode_samples.zip. [nltk_data]    | Downloading
package universal_tagset to [nltk_data]      |
              /root/nltk_data...
[nltk_data]   |   Unzipping taggers/universal_tagset.zip.
[nltk_data]   | Downloading package
universal_treebanks_v20 to [nltk_data]       |       /
root/nltk_data...
[nltk_data]   | Downloading package vader_lexicon
to [nltk_data]      |       /root/nltk_data...
[nltk_data]   | Downloading package verbnet to /root/
nltk_data... [nltk_data]    |       Unzipping corpora/
verbnet.zip.
[nltk_data]   | Downloading package verbnet3 to /root/nltk_data...
[nltk_data]   |   Unzipping corpora/verbnet3.zip.
[nltk_data]   | Downloading package webtext to /root/
nltk_data... [nltk_data]    |       Unzipping corpora/
webtext.zip.
[nltk_data]   | Downloading package wmt15_eval to /root/
nltk_data... [nltk_data]    |       Unzipping models/
wmt15_eval.zip.
[nltk_data]   | Downloading package word2vec_sample to
[nltk_data]   |     /root/nltk_data...
[nltk_data]   |   Unzipping models/word2vec_sample.zip.
[nltk_data]   | Downloading package wordnet to /root/nltk_data...
[nltk_data]    |  Downloading  package  wordnet2021  to  /root/
nltk_data... [nltk_data]  | Downloading package wordnet2022 to /
```

```
root/nltk_data...    [nltk_data]        |     Unzipping   corpora/
wordnet2022.zip.
[nltk_data]   | Downloading package wordnet31 to /root/
nltk_data... [nltk_data]    | Downloading package wordnet_ic
to /root/nltk_data... [nltk_data]    |       Unzipping corpora/
wordnet_ic.zip.
[nltk_data]   | Downloading package words to /root/
nltk_data... [nltk_data]    |       Unzipping corpora/
words.zip.
[nltk_data]   | Downloading package ycoe to /root/
nltk_data... [nltk_data]    |       Unzipping corpora/
ycoe.zip.
[nltk_data]   |
[nltk_data] Done downloading collection
all True
```

Tokenization:

Tokenization is the process of breaking text into individual words or sentences. NLTK provides various tokenizers for this purpose.

```
import nltk
from nltk.tokenize import word_tokenize, sent_tokenize
text = "NLTK is a leading platform for building Python programs to work with human
language data." words = word_tokenize(text)
sentences =
sent_tokenize(text)
print("Word tokens:", words)
print("Sentence tokens:", sentences)
```

```
Word tokens: ['NLTK', 'is', 'a', 'leading', 'platform', 'for', 'building', 'Python', 'programs', 'to', 'work', 'with',
'human', 'la Sentence tokens: ['NLTK is a leading platform for building Python programs to work with human language
data.']
```

Part-of-speech Tagging:

Part-of-speech tagging assigns a grammatical category (such as noun, verb, adjective, etc.) to each word in a sentence.

```
from nltk import pos_tag
pos_tags = pos_tag(words)
print("Part-of-speech tags:", pos_tags)
```

```
Part-of-speech tags: [('NLTK', 'NNP'), ('is', 'VBZ'), ('a', 'DT'), ('leading', 'VBG'), ('platform', 'NN'), ('for', 'IN'),
('buildin
```

Lemmatization:

Lemmatization is similar to stemming but aims to return the base or dictionary form of a word, which is known as the lemma.

```
from nltk.stem import WordNetLemmatizer
lemmatizer = WordNetLemmatizer()
lemmatized_words = [lemmatizer.lemmatize(word) for word in words]
print("Lemmatized words:", lemmatized_words)
```

```
Lemmatized words: ['NLTK', 'is', 'a', 'leading', 'platform', 'for', 'building', 'Python', 'program', 'to', 'work', 'with',
'human',
```

Named Entity Recognition (NER):

NER identifies named entities such as persons, organizations, and locations in text.

```
from nltk import ne_chunk
from nltk.tokenize import word_tokenize
text = "Steve Jobs was the CEO of Apple Inc. located in
California." words = word_tokenize(text)
pos_tags = pos_tag(words)
named_entities = ne_chunk(pos_tags)
print("Named entities:", named_entities)
```

```
Named entities: (S
  (PERSON Steve/NNP)
  (PERSON Jobs/NNP)
  was/
  VBD
  the/DT
  (ORGANIZATION CEO/NNP)
  of/IN
  (ORGANIZATION Apple/NNP Inc./NNP)
  located/VBD
  in/IN
  (GPE California/NNP)
  ./.)
```

LAB2 Implement morphological parser to accept and reject given string.

```
import nltk
from       nltk.tokenize       import
word_tokenize  from nltk.stem  import
WordNetLemmatizer  from  nltk.corpus
import words
from nltk.metrics.distance import edit_distance

class MorphologicalParser:
    def init(self):
        self.accepted_suffixes = ["ing", "ed",
        "s","ies"] self.accepted_prefixes = ["un",
        "re"]
        self.lexicon = set(words.words())  # English word lexicon
        self.lemmatizer = WordNetLemmatizer()

    def parse_word(self,
        word): prefix = ""
        suffix = ""

        # Tokenize the word
        tokens = word_tokenize(word)

        # Check if the word has an accepted
        prefix for p in
        self.accepted_prefixes:
            if tokens[0].startswith(p):
                prefix = p
                tokens[0] = tokens[0][len(p):]
                break

        # Check if the word has an accepted
        suffix for s in
        self.accepted_suffixes:
            if tokens[-1].endswith(s):
                suffix = s
                tokens[-1] = tokens[-1][:-
                len(s)] break

        # Lemmatize the remaining word
        lemmatized_word = self.lemmatizer.lemmatize(tokens[-1])

        # Check if the lemmatized word is in the
        lexicon if lemmatized_word.lower() in
        self.lexicon:
            # Apply spelling rules
            if
                self._is_spelled_correctly(lemmatized_w
                ord): # Apply orthographic rules
                lemmatized_word = self._apply_capitalization(lemmatized_word, word)

                # Check if the remaining word is
                acceptable if len(lemmatized_word) > 1:
                    return f"Accepted: {prefix}{lemmatized_word}

        {suffix}" return "Rejected"

    def _is_spelled_correctly(self, word):
        # Check if the word is spelled
        correctly return word.lower() in
        self.lexicon

    def _apply_capitalization(self, lemmatized_word,
        original_word): # Apply capitalization based on the
        original word
        if original_word.islower():
            return
        lemmatized_word.lower() elif
        original_word.isupper():
            return
        lemmatized_word.upper() elif
        original_word.istitle():
            return lemmatized_word.capitalize()
        else:
            return lemmatized_word

if name == "main":
    nltk.download('punkt')
    nltk.download('wordnet')
    nltk.download('words')
    parser = MorphologicalParser()

    # Test some example words
    words_to_test = ["running", "unhappy", "restarted", "JUMPED", "sit", "ed",
    "unseen","ladys"] for word in words_to_test:
        result = parser.parse_word(word)
        print(f"{word}: {result}")
```

```
[nltk_data] Downloading package punkt to /root/
nltk_data... [nltk_data]    Package punkt is already
up-to-date!
[nltk_data] Downloading package wordnet to /root/
nltk_data... [nltk_data]    Package wordnet is already
up-to-date!
[nltk_data] Downloading package words to /root/nltk_data...
```

```
[nltk_data]  Package words is already up-to-
date! running: Rejected
unhappy: Accepted: unhappy
restarted: Accepted: restarted
JUMPED: Rejected
sit: Accepted: sit
ed: Rejected
unseen: Accepted: unseen
ladys: Accepted: ladys
```

LAB3 Implement stemming and lemmatization for a corpus.

```
import nltk
from nltk.corpus import reuters
from nltk.tokenize import word_tokenize
from nltk.stem import PorterStemmer, WordNetLemmatizer

# Download WordNet resource if not already downloaded
nltk.download('wordnet')
# Initialize stemmer and lemmatizer
stemmer = PorterStemmer()
lemmatizer = WordNetLemmatizer()

# Sample text from the Reuters corpus
sample_text = reuters.raw('test/14826')

# Tokenize the text
words = word_tokenize(sample_text)

# Perform stemming
stemmed_words = [stemmer.stem(word) for word in words]

# Perform lemmatization
lemmatized_words = [lemmatizer.lemmatize(word) for word in words]

# Print results
print("Original words:", words)
print("Stemmed words:", stemmed_words)
print("Lemmatized words:", lemmatized_words)
```

```
[nltk_data] Downloading package wordnet to /root/
nltk_data... [nltk_data]    Package wordnet is already
up-to-date!
Original words: ['ASIAN', 'EXPORTERS', 'FEAR', 'DAMAGE', 'FROM', 'U.S.-JAPAN', 'RIFT', 'Mounting', 'trade', 'friction',
'between',
Stemmed words: ['asian', 'export', 'fear', 'damag', 'from', 'u.s.-japan', 'rift', 'mount', 'trade', 'friction', 'between',
'the', '
Lemmatized words: ['ASIAN', 'EXPORTERS', 'FEAR', 'DAMAGE', 'FROM', 'U.S.-JAPAN', 'RIFT', 'Mounting', 'trade', 'friction',
'between'
```

LAB4 Perform and analyse POS tagging -HMM

```
import nltk
from nltk.corpus  import
brown # Step 1: Prepare
Training Data
tagged_sentences =
brown.tagged_sents(tagset='universal') # Split the
data into training and testing sets
train_data = tagged_sentences[:int(0.8 *
len(tagged_sentences))] test_data =
tagged_sentences[int(0.8 * len(tagged_sentences)):] # Step
2: Train the HMM
hmm_tagger =
nltk.HiddenMarkovModelTagger.train(train_data) # Step
3: POS Tagging
sentence = "The quick brown fox jumps over the lazy
dog." tokenized_sentence =
nltk.word_tokenize(sentence)
pos_tags = hmm_tagger.tag(tokenized_sentence)
print("Predicted POS tags:")
print(pos_tags)
# Step 4: Evaluation
accuracy = hmm_tagger.evaluate(test_data)
print("Accuracy:", accuracy)
```

```
Predicted POS tags:
[('The', 'DET'), ('quick', 'ADJ'), ('brown', 'ADJ'), ('fox', 'NOUN'), ('jumps', 'VERB'), ('over', 'ADP'), ('the', 'DET'),
('lazy',
```

```
<ipython-input-11-62f2ff4fad9e>:17: DeprecationWarning:
  Function evaluate() has been deprecated.  Use
  accuracy(gold) instead.
  accuracy =
hmm_tagger.evaluate(test_data) Accuracy:
0.9363852687473148
```

LAB5 Implement a Viterbi algorithm using python or nltk

```
import numpy as np

class HMM_POS_Tagging:
    def init(self, states, observations, transition_probs, emission_probs, start_probs):
        self.states = states
        self.observations = observations
        self.transition_probs = transition_probs
        self.emission_probs = emission_probs
        self.start_probs = start_probs

    def viterbi(self,
        sequence): T =
        len(sequence)
        N = len(self.states)

        # Initialize the Viterbi matrix
        viterbi = np.zeros((N, T))
        backpointer = np.zeros((N, T), dtype=int)

        # Initialization step
        for s in range(N):
            viterbi[s, 0] = self.start_probs[s] * self.emission_probs[s, self.observations.index(sequence[0])]

        # Recursion step
        for t in range(1, T):
            for s in range(N):
                prob_list = [viterbi[s_prev, t-1] * self.transition_probs[s_prev, s] * self.emission_probs[s,
                self.observations.index(s viterbi[s, t] = max(prob_list)
                backpointer[s, t] = np.argmax(prob_list)

        # Termination step
        best_path_prob = np.max(viterbi[:, T-1])
        best_final_state = np.argmax(viterbi[:, T-1])

        # Backtrace to get the best path
        best_path = [best_final_state]
        for t in range(T-1, 0, -1):
            best_final_state = backpointer[best_final_state, t]
            best_path.append(best_final_state)
        best_path.reverse()
        return best_path,

best_path_prob # Example usage

states = ['Noun', 'Verb', 'Adjective']
observations = ['I', 'am', 'happy']
transition_probs = np.array([[0.4, 0.3, 0.3],
                             [0.2, 0.5, 0.3],
                             [0.1, 0.2, 0.7]])
   emission_probs = np.array([[0.1, 0.9,
                                      0.0],
                         [0.8, 0.2, 0.0],
                         [0.0, 0.0, 1.0]])
start_probs = np.array([0.2, 0.3, 0.5])

hmm = HMM_POS_Tagging(states, observations, transition_probs, emission_probs, start_probs)
best_path, best_path_prob = hmm.viterbi(observations)
print("POS Tags:", [states[i] for i in best_path], "with probability:", best_path_prob)
```

```
     POS Tags: ['Verb', 'Noun', 'Adjective'] with probability: 0.012960000000000001
```

Start coding or ge_nerate with AI.

LAB6 Implement a bigram model using 3 sentences in python orNLTK

```
import nltk
from nltk import bigrams
from nltk.tokenize import word_tokenize

# Sample sentences
sentences = [
    "This is the first sentence.",
    "Here comes the second
    sentence.", "And finally, the
    third sentence."
]
```

```
# Tokenize the sentences
```

```
tokenized_sentences = [word_tokenize(sentence.lower()) for sentence in sentences]

# Compute bigrams for each sentence
bigram_list = [list(bigrams(sentence)) for sentence in tokenized_sentences]

# Count occurrences of each bigram
bigram_freq = {}
for bigrams_in_sentence in
    bigram_list: for bigram in
    bigrams_in_sentence:
        if bigram in bigram_freq:
            bigram_freq[bigram] += 1
        else:
            bigram_freq[bigram] = 1

# Print the bigrams and their
frequencies print("Bigrams and their
frequencies:") for bigram, freq in
bigram_freq.items():
    print(bigram, ":", freq)
```

```
Bigrams and their frequencies:
('this', 'is') : 1
('is', 'the') : 1
('the', 'first') : 1
('first', 'sentence') : 1
('sentence', '.') : 3
('here', 'comes') : 1
('comes', 'the') : 1
('the', 'second') : 1
('second', 'sentence') : 1
('and', 'finally') : 1
('finally', ',') : 1
(',', 'the') : 1
('the', 'third') : 1
('third', 'sentence') : 1
```

LAB7 Text classification using Naïve Bayes Classifier

```
import nltk
from nltk.corpus import movie_reviews
from nltk.tokenize import
word_tokenize
from nltk.classify import
NaiveBayesClassifier from nltk.classify.util
import accuracy

# Prepare the dataset (Movie Reviews corpus in NLTK)
documents = [(list(movie_reviews.words(fileid)),
    category) for category in
    movie_reviews.categories()
    for fileid in movie_reviews.fileids(category)]

import random
random.shuffle(documents)

# Define feature extractor
function def
document_features(document):
    words = set(document)
    features = {}
    for word in word_features:
      features[word] = (word in words)
    return features

all_words = nltk.FreqDist(w.lower() for w in movie_reviews.words())

word_features = list(all_words.keys())[:2000] # Select top 2000 words as features
featuresets = [(document_features(d), c) for (d,c) in documents]
train_set, test_set = featuresets[:1600], featuresets[1600:]
classifier = NaiveBayesClassifier.train(train_set)

print("Accuracy:", accuracy(classifier, test_set))
new_text = "This movie is great!"
new_text_features = document_features(word_tokenize(new_text))
print("Classification:", classifier.classify(new_text_features))
```

```
Accuracy: 0.7925
Classification: neg
```

LAB8 Sentiment analysis using SVM

```
import pandas as
pd # Train Data
trainData = pd.read_csv("https://raw.githubusercontent.com/Vasistareddy/sentiment_analysis/master/data/
train.csv") # Test Data
testData = pd.read_csv("https://raw.githubusercontent.com/Vasistareddy/sentiment_analysis/master/data/test.csv")

from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(min_df=5, max_df=0.8, sublinear_tf=True, use_idf=True)
train_vectors = vectorizer.fit_transform(trainData['Content'])
test_vectors = vectorizer.transform(testData['Content'])

import time
from sklearn import svm
from sklearn.metrics import classification_report

# Perform classification with SVM, kernel=linear
classifier_linear = svm.SVC(kernel='linear')
t0 = time.time()
classifier_linear.fit(train_vectors,
trainData['Label']) t1 = time.time()
prediction_linear =
classifier_linear.predict(test_vectors) t2 =
time.time()
time_linear_train = t1 - t0
time_linear_predict = t2 - t1

print("Training time: %fs; Prediction time: %fs" % (time_linear_train,
time_linear_predict)) report = classification_report(testData['Label'],
prediction_linear, output_dict=True)
print('positive: ', report['pos'])
print('negative: ', report['neg'])
```

```
 Training time: 13.019269s; Prediction time: 1.140995s
positive:  {'precision': 0.9191919191919192, 'recall': 0.91, 'f1-score': 0.9145728643216081, 'support': 100}
negative:  {'precision': 0.9108910891089109, 'recall': 0.92, 'f1-score': 0.9154228855721394, 'support': 100}
```

```
review = """SUPERB, I AM IN LOVE IN THIS PHONE"""
review_vector = vectorizer.transform([review]) # Vectorizing
print(classifier_linear.predict(review_vector))
```

```
['pos']
```

```
review = """Do not purchase this product. My cell phone blasted when I switched the charger"""
review_vector = vectorizer.transform([review]) # Vectorizing
print(classifier_linear.predict(review_vector))
```

```
['neg']
```

Double-click (or enter) to edit

```python
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

import os
for dirname, _, filenames in os.walk('/kaggle/input'): for filename in
        filenames:
                print(os.path.join(dirname, filename))
```

/kaggle/input/spam-email-classification/email.csv

```python
data =
pd.read_csv("/kaggle/input/spam-email-classification/email.csv")

data.head()
```

	Category	Message
1.	ham	Go until jurong point, crazy.. Available only ...
2.	ham	Ok lar... Joking wif u oni...
3.	spam	Free entry in 2 a wkly comp to win FA Cup fina...
4.	ham	U dun say so early hor... U c already then say...
5.	ham	Nah I don't think he goes to usf, he lives aro...

```python
data['Message'][2]
```

"Free entry in 2 a wkly comp to win FA Cup final tkts 21st May 2005. Text FA to 87121 to receive entry question(std txt rate)T&C's apply 08452810075over18's"

```python
data.isna().sum()
```

```
Category  0
Message  0
dtype: int64
```

```python
data.tail()
```

	Category	Message
5568	ham	Will ü b going to esplanade fr home?
5569	ham	Pity, * was in mood for that. So...any other s...
5570	ham	The guy did some bitching but I acted like i'd...
5571	ham	Rofl. Its true to its
5572	{"mode":"full" isActive:false}	

```python
data.drop(data.tail(1).index,  inplace=True)
```

```
data.tail()

data['Category'].value_counts()/len(data) # Inbalanced Dataset
```

Data Preprocessing

```
ham = data[data['Category']=='ham'] spam
= data[data['Category']=='spam']

ham.shape,spam.shape

ham = ham.sample(spam.shape[0])

ham.shape,spam.shape
```

((747, 2), (747, 2))

```
data = pd.concat([ham, spam], ignore_index=True) data.shape
```

(1494, 2)

```
data['Category'].value_counts()#Balanced Dataset
```

Category ham
747 spam 747
Name: count, dtype: int64

```
data.head()
```

	Category	Message
0	ham	Yep, by the pretty sculpture
1	ham	Thankyou so much for the call. I appreciate yo...
2	ham	I'm at work. Please call
3	ham	Yar lor... How u noe? U used dat route too?
4	ham	Yesterday its with me only . Now am going home.

Text Preprocessing

```
#Convert to lowercase data['Message']=data['Message'].apply(lambda
x:str(x).lower()) data['Message'][0]
```

'yep, by the pretty sculpture'

```
#Remove stopwords
import nltk
```

```
from nltk.corpus import stopwords
stop_words = set(stopwords.words('english')) def
remove_stopwords(text):
        words = str(text).split() new_text=[]
        for word in words:

                if word not in stop_words: new_text.append(word)
        return " ".join(new_text)

data['Message']=data['Message'].apply(lambda x:remove_stopwords(x)) data['Message'][0]
```

```
'yep, pretty sculpture'
```

```
#Remove Punctuation
from string import punctuation def
clean_punctuation(text):
        translator = str.maketrans('','',punctuation) return
        text.translate(translator)
data['Message']=data['Message'].apply(lambda x:clean_punctuation(x)) data['Message'][0]
```

```
'yep pretty sculpture'
```

```
X = data['Message'] Y
= data['Category']

from sklearn.feature_extraction.text import TfidfVectorizer tf =
TfidfVectorizer()
X = tf.fit_transform(X)
```

Split into train and test

```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test =
train_test_split(X,Y,test_size=0.3,random_state=42,
                                                            shuffle=True)

x_train.shape,y_train.shape,x_test.shape ((1045, 4849),

(1045,), (449, 4849))
```

Model Building

```
import tensorflow
from tensorflow import keras
from keras.models import Sequential from
keras.layers import Dense,Dropout
```

```
2024-05-04 11:50:28.665760: E
external/local_xla/xla/stream_executor/cuda/cuda_dnn.cc:9261] Unable to register
cuDNN factory: Attempting to register factory for plugin cuDNN when one has already
been registered
2024-05-04 11:50:28.665879: E
external/local_xla/xla/stream_executor/cuda/cuda_fft.cc:607] Unable to register
cuFFT factory: Attempting to register factory for plugin cuFFT when one has already
been registered
2024-05-04 11:50:28.792478: E
external/local_xla/xla/stream_executor/cuda/cuda_blas.cc:1515] Unable to register
cuBLAS factory: Attempting to register factory for plugin cuBLAS when one has already
been registered
```

```
model = Sequential()
```

```
x_train.shape[1:] (4849,)
```

```
model.add(Dense(units=20,activation='relu',input_shape=x_train.shape[1
:]))
model.add(Dropout(0.5)) model.add(Dense(30,
activation='relu')) model.add(Dropout(0.5))
model.add(Dense(1, activation='sigmoid'))
```

```
model.summary()
```

```
Model: "sequential"
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━┓
┃ ɔe) ┃ Output Shape
Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━┩
│ (Dense) │ (None, 20)
97,000 │
├──────────────────────────────────────────────────────────────────────────────────┼───────────────────────────────────────────────────────────────┼────────────────┤
│ ropout (Dropout) │ (None, 20)
0 │
├──────────────────────────────────────────────────────────────────────────────────┼───────────────────────────────────────────────────────────────┼────────────────┤
```

```
| dense_1 (Dense)                    | (None, 30)
630 |
├──────────────────────────────────────────────────────┼────────────
────────────────────────────┼─────────────────
────────┤
   ropout_1 (Dropout)                | (None, 30)
0 |
├──────────────────────────────────────────────────────┼────────────
────────────────────────────┼─────────────────
────────┤
    nse_2 (Dense)                    | (None, 1)
31 |
└──────────────────────────────────────────────────────┴────────────
────────────────────────────┴─────────────────
────────┘

 Total params: 97,661 (381.49 KB)

 Trainable params: 97,661 (381.49 KB)

 Non-trainable params: 0 (0.00 B)
```

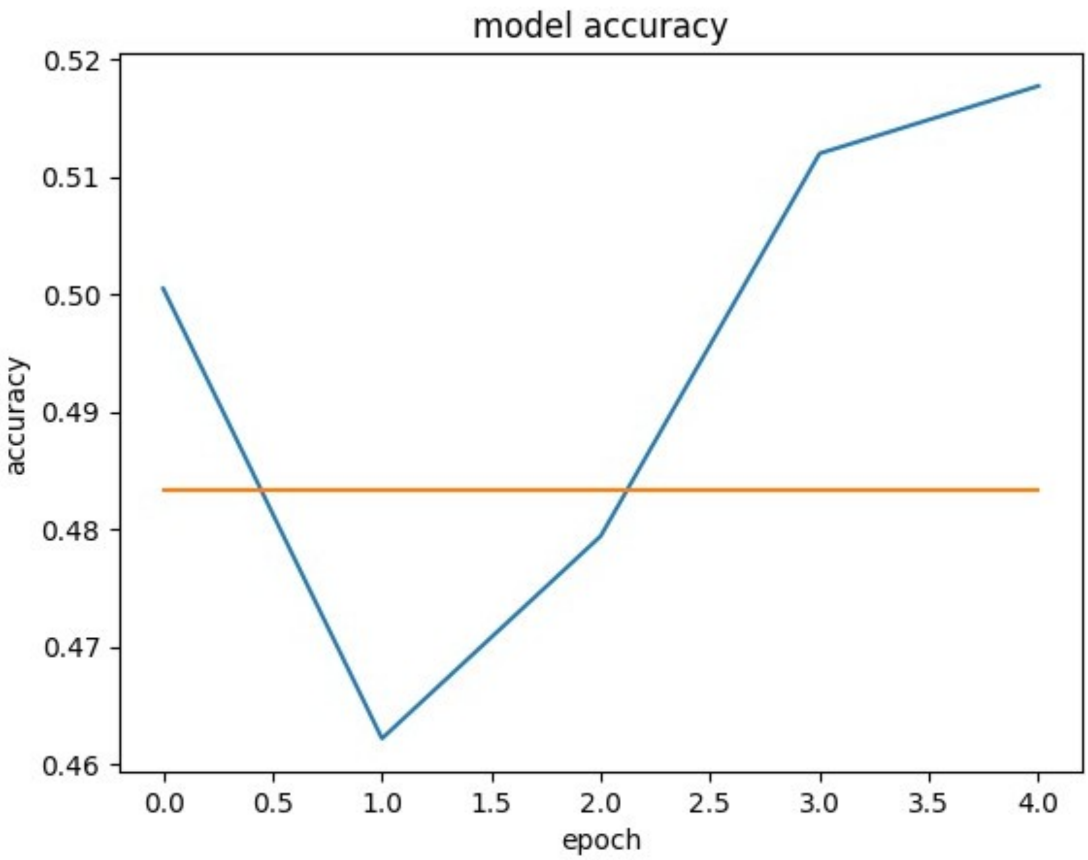
```
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
from sklearn.preprocessing import LabelEncoder

# Initialize LabelEncoder
label_encoder = LabelEncoder()

# Fit label encoder and transform target variables
y_train_encoded = label_encoder.fit_transform(y_train)
y_test_encoded = label_encoder.transform(y_test)

from tensorflow.keras.callbacks import EarlyStopping early_stopping =
EarlyStopping(monitor='val_loss', patience=3, verbose=1)
history = model.fit(x_train, y_train_encoded, epochs=100, validation_data=(x_test,
y_test_encoded), callbacks=[early_stopping])
```

```
Epoch 1/100
33/33 ━━━━━━━━━━━━━━━━━━━━ 3s 16ms/step - accuracy: 0.5061
- loss: 0.6973 - val_accuracy: 0.4833 - val_loss: 0.6933
Epoch 2/100
33/33 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.4641 -
loss: 0.6987 - val_accuracy: 0.4833 - val_loss: 0.6932
Epoch 3/100
33/33 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.4930 -
loss: 0.6944 - val_accuracy: 0.4833 - val_loss: 0.6933
Epoch 4/100
33/33 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.5117 -
loss: 0.6918 - val_accuracy: 0.4833 - val_loss: 0.6934
Epoch 5/100
33/33 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.5291 -
loss:
```

0.6928 - val_accuracy: 0.4833 - val_loss: 0.6932 Epoch 5: early stopping

```
import matplotlib.pyplot as plt
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy') plt.ylabel('accuracy')
plt.xlabel('epoch')
```

Text(0.5, 0, 'epoch')




```
y_train
```

```
1459        spam
1126        spam
Name: Category, Length: 1045, dtype: object
```

```
y_train_encoded #spam-1 ham-0
```

```
array([0, 0, 0, ..., 1, 1, 1])
```

```
x_test
```

```
<449x4849 sparse matrix of type '<class 'numpy.float64'>'
        with 5332 stored elements in Compressed Sparse Row format>
```

```
y_pred = model.predict(x_test)
```

```
15/15
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s
8ms/step np.argmax(y_pred, axis=1)
```

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
           0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
           0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
           0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
           0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
           0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
           0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
           0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
           0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
           0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
           0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
           0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
           0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
           0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
           0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
           0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
```

```
              0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
              0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
              0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
              0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
              0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
```

```
model.evaluate(x_test,y_test_encoded)
```

```
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.4962 -
loss: 0.6932
[0.6932110786437988, 0.4832962155342102]
```

```
predict_msg = ["Free entry in 2 a wkly comp to win FA Cup final tkts 21st May 2005.
Text FA to 87121 to receive entry question(std txt rate)T&C's apply
08452810075over18's"]
text = tf.fit_transform(predict_msg) if

(model.predict(text))>0.5:

        print("Spam")
else:
        print("Not Spam")
```

```
1/1
━━━━━━━━━━━━━━━━━━━━ 0s
46ms/step Not Spam
```

10. Mini Project based on NLP Applications.

Chatbot

```
import pickle import
numpy as np

with open("train_qa.txt",'rb') as f: train_data=pickle.load(f)

with open("test_qa.txt",'rb') as f: test_data=pickle.load(f)

for story,question,answer in all_data: vocab=vocab.union(set(story))
    vocab=vocab.union(set(question))

vocab.add('no')
vocab.add('yes')


all_story_lens=[len(data[0]) for data in all_data] max_story_len=max(all_story_lens)

max_question_len=max([len(data[1]) for data in all_data]) max_question_len

6

# We have Reserve 0 for keras pad_sequences
vocab_size = len(vocab) + 1

from keras.preprocessing.sequence import pad_sequences from
keras.preprocessing.text import Tokenizer tokenizer =
Tokenizer(filters=[]) tokenizer.fit_on_texts(vocab)
tokenizer.word_index

{'in': 1,
 'left': 2,
 'discarded': 3,
 'to': 4,
 'daniel': 5,
 'down': 6,
 'went': 7,
 'office': 8,
 'is': 9,
 'football': 10,
 'moved': 11,
```

```
'up': 12,
'travelled': 13,
'kitchen': 14,
'no': 15,
'picked': 16,
'?': 17,
'put': 18,
'grabbed': 19,
'yes': 20,
'took': 21,
'bathroom': 22,
'got': 23,
'back': 24,
'dropped': 25,
'garden': 26,
'hallway': 27,
'.': 28,
'there': 29,
'bedroom': 30,
'john': 31,
'apple': 32,
'the': 33,
'mary': 34,
'sandra': 35,
'milk': 36,
'journeyed': 37}

train_story_text = [] train_question_text
= [] train_answers = []

for story,question,answer in train_data:
        train_story_text.append(story)
        train_question_text.append(question)

train_story_seq = tokenizer.texts_to_sequences(train_story_text) len(train_story_text)

10000

len(train_story_seq)

10000

def vectorize_stories(data, word_index=tokenizer.word_index,
max_story_len=max_story_len,max_question_len=max_question_len):



        X = []
```

```
    Xq = []
    Y = []

    for story, query, answer in data:

        x = [word_index[word.lower()] for word in story] xq =
        [word_index[word.lower()] for word in query]

        y = np.zeros(len(word_index) + 1)

        y[word_index[answer]] = 1

        X.append(x)

        Xq.append(xq)
        Y.append(y)

    return (pad_sequences(X, maxlen=max_story_len),pad_sequences(Xq,
maxlen=max_question_len), np.array(Y))

inputs_train, queries_train, answers_train =
vectorize_stories(train_data)
inputs_test, queries_test, answers_test = vectorize_stories(test_data) inputs_test
```

```
    array([[ 0,   0,   0,   ...,   33,  30,  28],
           [ 0,   0,   0,   ...,   33,  26,  28],
           [ 0,   0,   0,   ...,   33,  26,  28],
           ...,
           [ 0,   0,   0,   ...,   33,  32,  28],
           [ 0,   0,   0,   ...,   33,  26,  28],
           [ 0,   0,   0, ..., 32, 29, 28]], dtype=int32)
```

```
queries_test
```

```
    array([[ 9,  31,  1,  33,  14,  17],
           [ 9,  31,  1,  33,  14,  17],
           [ 9,  31,  1,  33,  26,  17],
           ...,
           [ 9,  34,  1,  33,  30,  17],
           [ 9,  35,  1,  33,  26,  17],
           [ 9,  34,   1, 33, 26, 17]], dtype=int32)
```

```
answers_test

array([[0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       ...,
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.]])

sum(answers_test)

array([   0.,    0.,    0.,    0.,    0.,    0.,    0.,    0.,    0.,    0.,
0.,
          0.,    0.,    0.,    0., 503.,    0.,    0.,    0.,    0., 497.,
0.,
          0.,    0.,    0.,    0.,    0.,    0.,    0.,    0.,    0.,    0.,
0.,
          0.,    0.,    0.,    0.,    0.])

tokenizer.word_index['yes'] 20

tokenizer.word_index['no'] 15

from keras.models import Sequential, Model from
keras.layers import Embedding
from keras.layers import Input, Activation, Dense, Permute, Dropout from keras.layers
import add, dot, concatenate
from keras.layers import LSTM

input_sequence = Input((max_story_len,)) question =
Input((max_question_len,))


input_encoder_m = Sequential()
input_encoder_m.add(Embedding(input_dim=vocab_size,output_dim=64))
input_encoder_m.add(Dropout(0.3))


input_encoder_c = Sequential()
input_encoder_c.add(Embedding(input_dim=vocab_size,output_dim=max_ques        tion_len))
input_encoder_c.add(Dropout(0.3))


question_encoder = Sequential() question_encoder.add(Embedding(input_dim=vocab_size,
                                                output_dim=64,
```

```
                                                 input_length=max_question_len))
question_encoder.add(Dropout(0.3))


input_encoded_m  =  input_encoder_m(input_sequence)
input_encoded_c  =  input_encoder_c(input_sequence)
question_encoded = question_encoder(question)


match = dot([input_encoded_m, question_encoded], axes=(2, 2)) match =
Activation('softmax')(match)


response = add([match, input_encoded_c]) response =
Permute((2, 1))(response)


answer = concatenate([response, question_encoded]) answer

<KerasTensor: shape=(None, 6, 220) dtype=float32 (created by layer 'concatenate')>

answer=concatenate([response,question_encode])  answer =
LSTM(32)(answer)


answer = Dropout(0.5)(answer) answer =
Dense(vocab_size)(answer)

answer    =    Activation('softmax')(answer)


model = Model([input_sequence, question], answer) model.compile(optimizer='rmsprop',
loss='categorical_crossentropy',
                      metrics=['accuracy'])

model.summary()

Model: "model"


 Layer (type)                        Output Shape
Param # Connected to
==================================================================
=
  input_4 (InputLayer)               [(None, 156)]           0           []


  input_5 (InputLayer)               [(None, 6)]             0           []
```

```
 sequential_2 (Sequential)          (None, None, 64)
['input_4[0][0]']

 sequential_4 (Sequential)          (None,  6, 64)
2432 ['input_5[0][0]']

 dot (Dot)                          (None,  156,  6)                 0
['sequential_2[0][0]',

'sequential_4[0][0]']

 activation (Activation)            (None, 156, 6)
0['dot[0][0]']

 sequential_3 (Sequential)          (None, None, 6)
228 ['input_4[0][0]']

 add (Add)                          (None,  156,  6)                 0
['activation[0][0]',

'sequential_3[0][0]']

 permute  (Permute)                 (None,  6,  156)                 0
['add[0][0]']

 concatenate (Concatenate)          (None, 6, 220)
0['permute[0][0]',

'sequential_4[0][0]']

 lstm (LSTM)                        (None, 32)                       32384
['concatenate[0][0]']

 dropout_5 (Dropout)                (None, 32)                       0
['lstm[0][0]']

 dense (Dense)                      (None,  38)
['dropout_5[0][0]']
```

```
 activation_1 (Activation)          (None, 38)                  0
['dense[0][0]']

==========================================================================
=
==============================
Total params: 38730 (151.29 KB)
Trainable params: 38730 (151.29 KB) Non-
trainable params: 0 (0.00 Byte)
__________________________________________________________________________
______________________________


history = model.fit([inputs_train, queries_train],
answers_train,batch_size=32,epochs=200,validation_data=([inputs_test, queries_test],
answers_test))
Epoch 1/120
313/313 [==============================] - 11s 24ms/step - loss:
0.9020 - accuracy: 0.4920 - val_loss: 0.6967 - val_accuracy: 0.5030 Epoch 2/120
313/313 [==============================] - 6s 18ms/step - loss: 0.7112
- accuracy: 0.4903 - val_loss: 0.6968 - val_accuracy: 0.4970 Epoch 3/120
313/313 [==============================] - 7s 23ms/step - loss: 0.6996
- accuracy: 0.4962 - val_loss: 0.6949 - val_accuracy: 0.5030 Epoch 4/120
313/313 [==============================] - 6s 18ms/step - loss: 0.6969
- accuracy: 0.4974 - val_loss: 0.6935 - val_accuracy: 0.5030 Epoch 5/120
313/313 [==============================] - 7s 22ms/step - loss: 0.6963
- accuracy: 0.4914 - val_loss: 0.6942 - val_accuracy: 0.4970 Epoch 6/120
313/313 [==============================] - 6s 18ms/step - loss: 0.6959
- accuracy: 0.4919 - val_loss: 0.6932 - val_accuracy: 0.5030 Epoch 7/120
313/313 [==============================] - 7s 22ms/step - loss: 0.6950
- accuracy: 0.5091 - val_loss: 0.6932 - val_accuracy: 0.4970 Epoch 8/120
313/313 [==============================] - 6s 20ms/step - loss: 0.6954
- accuracy: 0.5007 - val_loss: 0.6938 - val_accuracy: 0.4970 Epoch 9/120
313/313 [==============================] - 7s 23ms/step - loss: 0.6957
- accuracy: 0.4931 - val_loss: 0.6935 - val_accuracy: 0.4970 Epoch
 10/120
313/313 [==============================] - 6s 19ms/step - loss: 0.6951
- accuracy: 0.5006 - val_loss: 0.6954 - val_accuracy: 0.4970 Epoch
 11/120
313/313 [==============================] - 7s 21ms/step - loss: 0.6947
```

```
-   accuracy: 0.5016 - val_loss: 0.6944 - val_accuracy: 0.5030 Epoch 12/120
313/313 [==============================] - 6s 18ms/step - loss: 0.6956
-   accuracy: 0.4963 - val_loss: 0.6951 - val_accuracy: 0.4970 Epoch
13/120
313/313 [==============================] - 7s 21ms/step - loss: 0.6957
-   accuracy: 0.4977 - val_loss: 0.6934 - val_accuracy: 0.4970 Epoch
14/120
313/313 [==============================] - 6s 20ms/step - loss: 0.6948
-   accuracy: 0.4997 - val_loss: 0.6957 - val_accuracy: 0.4970 Epoch
15/120
313/313 [==============================] - 6s 18ms/step - loss: 0.6950
-   accuracy: 0.5011 - val_loss: 0.6953 - val_accuracy: 0.4970 Epoch 16/120
313/313 [==============================] - 7s 21ms/step - loss: 0.6950
-   accuracy: 0.5003 - val_loss: 0.6957 - val_accuracy: 0.5030 Epoch
17/120
313/313 [==============================] - 6s 19ms/step - loss: 0.6951
-   accuracy: 0.4977 - val_loss: 0.6932 - val_accuracy: 0.5030 Epoch
18/120
313/313 [==============================] - 7s 22ms/step - loss: 0.6943
-   accuracy: 0.5026 - val_loss: 0.6961 - val_accuracy: 0.4970 Epoch
19/120
313/313 [==============================] - 6s 18ms/step - loss: 0.6956
-   accuracy: 0.4945 - val_loss: 0.6961 - val_accuracy: 0.4970 Epoch
20/120
313/313 [==============================] - 6s 20ms/step - loss: 0.6953
-   accuracy: 0.4877 - val_loss: 0.6932 - val_accuracy: 0.5030 Epoch
21/120
313/313 [==============================] - 5s 16ms/step - loss: 0.6944
-   accuracy: 0.5080 - val_loss: 0.6934 - val_accuracy: 0.4970 Epoch
22/120
313/313 [==============================] - 6s 20ms/step - loss: 0.6947
-   accuracy: 0.5031 - val_loss: 0.6947 - val_accuracy: 0.5030 Epoch
23/120
313/313 [==============================] - 5s 17ms/step - loss: 0.6953
-   accuracy: 0.4995 - val_loss: 0.6931 - val_accuracy: 0.5030 Epoch
24/120
313/313 [==============================] - 6s 20ms/step - loss: 0.6950
-   accuracy: 0.5004 - val_loss: 0.6932 - val_accuracy: 0.4970 Epoch
25/120
313/313 [==============================] - 6s 18ms/step - loss: 0.6953
-   accuracy: 0.4968 - val_loss: 0.6942 - val_accuracy: 0.5030 Epoch
26/120
313/313 [==============================] - 6s 19ms/step - loss: 0.6954
-   accuracy: 0.4921 - val_loss: 0.6968 - val_accuracy: 0.5030 Epoch
27/120
313/313 [==============================] - 6s 19ms/step - loss: 0.6954
-   accuracy: 0.4963 - val_loss: 0.6934 - val_accuracy: 0.4970
```

```
Epoch 28/120
313/313 [==============================] - 5s 17ms/step - loss: 0.6949
- accuracy: 0.4988 - val_loss: 0.6932 - val_accuracy: 0.4910 Epoch
29/120
313/313 [==============================] - 7s 21ms/step - loss: 0.6947
- accuracy: 0.5040 - val_loss: 0.6932 - val_accuracy: 0.5040 Epoch
30/120
313/313 [==============================] - 6s 18ms/step - loss: 0.6952
- accuracy: 0.4960 - val_loss: 0.6942 - val_accuracy: 0.4970 Epoch
31/120
313/313 [==============================] - 7s 21ms/step - loss: 0.6937
- accuracy: 0.5100 - val_loss: 0.6934 - val_accuracy: 0.5030 Epoch
32/120
313/313 [==============================] - 6s 19ms/step - loss: 0.6956
- accuracy: 0.4945 - val_loss: 0.6932 - val_accuracy: 0.5030 Epoch
33/120
313/313 [==============================] - 7s 22ms/step - loss: 0.6949
- accuracy: 0.4933 - val_loss: 0.6933 - val_accuracy: 0.4970 Epoch
34/120
313/313 [==============================] - 6s 19ms/step - loss: 0.6951
- accuracy: 0.4961 - val_loss: 0.6943 - val_accuracy: 0.5030 Epoch
35/120
313/313 [==============================] - 7s 22ms/step - loss: 0.6953
- accuracy: 0.4966 - val_loss: 0.6932 - val_accuracy: 0.5030 Epoch
36/120
313/313 [==============================] - 6s 18ms/step - loss: 0.6950
- accuracy: 0.5075 - val_loss: 0.6936 - val_accuracy: 0.5030 Epoch
37/120
313/313 [==============================] - 6s 21ms/step - loss: 0.6948
- accuracy: 0.4950 - val_loss: 0.6952 - val_accuracy: 0.4970 Epoch
38/120
313/313 [==============================] - 5s 17ms/step - loss: 0.6952
- accuracy: 0.4912 - val_loss: 0.6932 - val_accuracy: 0.5030 Epoch
39/120
313/313 [==============================] - 6s 20ms/step - loss: 0.6949
- accuracy: 0.5005 - val_loss: 0.6941 - val_accuracy: 0.4970 Epoch
40/120
313/313 [==============================] - 6s 20ms/step - loss: 0.6949
- accuracy: 0.5015 - val_loss: 0.6932 - val_accuracy: 0.5060 Epoch 41/120
313/313 [==============================] - 7s 21ms/step - loss: 0.6950
- accuracy: 0.4987 - val_loss: 0.6938 - val_accuracy: 0.5030 Epoch
42/120
313/313 [==============================] - 6s 18ms/step - loss: 0.6942
- accuracy: 0.5043 - val_loss: 0.6934 - val_accuracy: 0.4940 Epoch
43/120
313/313 [==============================] - 7s 22ms/step - loss: 0.6945
- accuracy: 0.5068 - val_loss: 0.6941 - val_accuracy: 0.5030 Epoch
44/120
```

```
313/313 [==============================] - 6s 18ms/step - loss: 0.6942
- accuracy: 0.5014 - val_loss: 0.6939 - val_accuracy: 0.4970 Epoch
45/120
313/313 [==============================] - 6s 20ms/step - loss: 0.6935
- accuracy: 0.5172 - val_loss: 0.6941 - val_accuracy: 0.4870 Epoch 46/120
313/313 [==============================] - 6s 20ms/step - loss: 0.6937
- accuracy: 0.5110 - val_loss: 0.6969 - val_accuracy: 0.4930 Epoch 47/120
313/313 [==============================] - 6s 20ms/step - loss: 0.6936
- accuracy: 0.5198 - val_loss: 0.6974 - val_accuracy: 0.4930 Epoch
48/120
313/313 [==============================] - 6s 19ms/step - loss: 0.6924
- accuracy: 0.5254 - val_loss: 0.6977 - val_accuracy: 0.4690 Epoch
49/120
313/313 [==============================] - 6s 19ms/step - loss: 0.6931
- accuracy: 0.5191 - val_loss: 0.6964 - val_accuracy: 0.4950 Epoch 50/120
313/313 [==============================] - 6s 20ms/step - loss: 0.6924
- accuracy: 0.5222 - val_loss: 0.6969 - val_accuracy: 0.4740 Epoch
51/120
313/313 [==============================] - 6s 19ms/step - loss: 0.6929
- accuracy: 0.5265 - val_loss: 0.6966 - val_accuracy: 0.5090 Epoch
52/120
313/313 [==============================] - 6s 20ms/step - loss: 0.6932
- accuracy: 0.5178 - val_loss: 0.6973 - val_accuracy: 0.5000 Epoch 53/120
313/313 [==============================] - 6s 18ms/step - loss: 0.6906
- accuracy: 0.5351 - val_loss: 0.6975 - val_accuracy: 0.4940 Epoch
54/120
313/313 [==============================] - 7s 22ms/step - loss: 0.6889
- accuracy: 0.5349 - val_loss: 0.6967 - val_accuracy: 0.5030 Epoch
55/120
313/313 [==============================] - 6s 18ms/step - loss: 0.6855
- accuracy: 0.5422 - val_loss: 0.7049 - val_accuracy: 0.5130 Epoch
56/120
313/313 [==============================] - 6s 21ms/step - loss: 0.6792
- accuracy: 0.5621 - val_loss: 0.6825 - val_accuracy: 0.5470 Epoch
57/120
313/313 [==============================] - 6s 21ms/step - loss: 0.6618
- accuracy: 0.5927 - val_loss: 0.6582 - val_accuracy: 0.6110 Epoch 58/120
313/313 [==============================] - 6s 20ms/step - loss: 0.6435
- accuracy: 0.6290 - val_loss: 0.6430 - val_accuracy: 0.6420 Epoch
59/120
313/313 [==============================] - 6s 18ms/step - loss: 0.6281
- accuracy: 0.6524 - val_loss: 0.6213 - val_accuracy: 0.6580 Epoch
60/120
313/313 [==============================] - 6s 20ms/step - loss: 0.6145
```

```
- accuracy: 0.6662 - val_loss: 0.6022 - val_accuracy: 0.6820 Epoch
61/120
313/313 [==============================] - 6s 18ms/step - loss: 0.5956
- accuracy: 0.6884 - val_loss: 0.5810 - val_accuracy: 0.6990 Epoch
62/120
313/313 [==============================] - 7s 21ms/step - loss: 0.5804
- accuracy: 0.7051 - val_loss: 0.5606 - val_accuracy: 0.7140 Epoch 63/120
313/313 [==============================] - 5s 17ms/step - loss: 0.5641
- accuracy: 0.7190 - val_loss: 0.5354 - val_accuracy: 0.7360 Epoch
64/120
313/313 [==============================] - 6s 20ms/step - loss: 0.5448
- accuracy: 0.7365 - val_loss: 0.5150 - val_accuracy: 0.7590 Epoch 65/120
313/313 [==============================] - 6s 18ms/step - loss: 0.5257
- accuracy: 0.7525 - val_loss: 0.5036 - val_accuracy: 0.7650 Epoch
66/120
313/313 [==============================] - 6s 20ms/step - loss: 0.5193
- accuracy: 0.7562 - val_loss: 0.4746 - val_accuracy: 0.7840 Epoch
67/120
313/313 [==============================] - 6s 18ms/step - loss: 0.5004
- accuracy: 0.7749 - val_loss: 0.4525 - val_accuracy: 0.7970 Epoch
68/120
313/313 [==============================] - 6s 18ms/step - loss: 0.4726
- accuracy: 0.7893 - val_loss: 0.4656 - val_accuracy: 0.8120 Epoch
69/120
313/313 [==============================] - 7s 21ms/step - loss: 0.4685
- accuracy: 0.7896 - val_loss: 0.4345 - val_accuracy: 0.8120 Epoch
70/120
313/313 [==============================] - 5s 17ms/step - loss: 0.4534
- accuracy: 0.8051 - val_loss: 0.4415 - val_accuracy: 0.8200 Epoch 71/120
313/313 [==============================] - 6s 21ms/step - loss: 0.4447
- accuracy: 0.8101 - val_loss: 0.4231 - val_accuracy: 0.8250 Epoch
72/120
313/313 [==============================] - 5s 17ms/step - loss: 0.4391
- accuracy: 0.8152 - val_loss: 0.4165 - val_accuracy: 0.8180 Epoch 73/120
313/313 [==============================] - 7s 21ms/step - loss: 0.4189
- accuracy: 0.8212 - val_loss: 0.4315 - val_accuracy: 0.8170 Epoch 74/120
313/313 [==============================] - 5s 17ms/step - loss: 0.4147
- accuracy: 0.8264 - val_loss: 0.4175 - val_accuracy: 0.8300 Epoch 75/120
313/313 [==============================] - 6s 19ms/step - loss: 0.4179
- accuracy: 0.8269 - val_loss: 0.4134 - val_accuracy: 0.8250 Epoch
76/120
313/313 [==============================] - 6s 18ms/step - loss: 0.4068
- accuracy: 0.8287 - val_loss: 0.4021 - val_accuracy: 0.8270
```

```
Epoch 77/120
313/313 [==============================] - 5s 17ms/step - loss: 0.3972
- accuracy: 0.8350 - val_loss: 0.4127 - val_accuracy: 0.8260 Epoch
78/120
313/313 [==============================] - 6s 20ms/step - loss: 0.3946
- accuracy: 0.8411 - val_loss: 0.4141 - val_accuracy: 0.8270 Epoch 79/120
313/313 [==============================] - 5s 17ms/step - loss: 0.3929
- accuracy: 0.8361 - val_loss: 0.3945 - val_accuracy: 0.8330 Epoch
80/120
313/313 [==============================] - 6s 19ms/step - loss: 0.3822
- accuracy: 0.8404 - val_loss: 0.4228 - val_accuracy: 0.8300 Epoch
81/120
313/313 [==============================] - 5s 16ms/step - loss: 0.3798
- accuracy: 0.8423 - val_loss: 0.4088 - val_accuracy: 0.8210 Epoch
82/120
313/313 [==============================] - 6s 20ms/step - loss: 0.3792
- accuracy: 0.8410 - val_loss: 0.3963 - val_accuracy: 0.8280 Epoch
83/120
313/313 [==============================] - 5s 16ms/step - loss: 0.3769
- accuracy: 0.8472 - val_loss: 0.3924 - val_accuracy: 0.8290 Epoch
84/120
313/313 [==============================] - 6s 18ms/step - loss: 0.3727
- accuracy: 0.8484 - val_loss: 0.3919 - val_accuracy: 0.8380 Epoch
85/120
313/313 [==============================] - 6s 19ms/step - loss: 0.3666
- accuracy: 0.8500 - val_loss: 0.4163 - val_accuracy: 0.8350 Epoch
86/120
313/313 [==============================] - 5s 17ms/step - loss: 0.3681
- accuracy: 0.8497 - val_loss: 0.3892 - val_accuracy: 0.8380 Epoch
87/120
313/313 [==============================] - 6s 21ms/step - loss: 0.3591
- accuracy: 0.8487 - val_loss: 0.4074 - val_accuracy: 0.8380 Epoch
88/120
313/313 [==============================] - 5s 17ms/step - loss: 0.3575
- accuracy: 0.8522 - val_loss: 0.3900 - val_accuracy: 0.8320 Epoch
89/120
313/313 [==============================] - 6s 20ms/step - loss: 0.3543
- accuracy: 0.8548 - val_loss: 0.3891 - val_accuracy: 0.8360 Epoch
90/120
313/313 [==============================] - 5s 16ms/step - loss: 0.3507
- accuracy: 0.8575 - val_loss: 0.3935 - val_accuracy: 0.8340 Epoch
91/120
313/313 [==============================] - 6s 19ms/step - loss: 0.3415
- accuracy: 0.8585 - val_loss: 0.3983 - val_accuracy: 0.8300 Epoch
92/120
313/313 [==============================] - 5s 17ms/step - loss: 0.3456
- accuracy: 0.8574 - val_loss: 0.3980 - val_accuracy: 0.8380 Epoch
93/120
```

```
313/313 [==============================] - 6s 18ms/step - loss: 0.3410
- accuracy: 0.8609 - val_loss: 0.4123 - val_accuracy: 0.8280 Epoch
94/120
313/313 [==============================] - 6s 19ms/step - loss: 0.3317
- accuracy: 0.8642 - val_loss: 0.4209 - val_accuracy: 0.8240 Epoch
95/120
313/313 [==============================] - 5s 16ms/step - loss: 0.3375
- accuracy: 0.8638 - val_loss: 0.3953 - val_accuracy: 0.8130 Epoch
96/120
313/313 [==============================] - 6s 19ms/step - loss: 0.3293
- accuracy: 0.8638 - val_loss: 0.4247 - val_accuracy: 0.8200 Epoch
97/120
313/313 [==============================] - 5s 17ms/step - loss: 0.3288
- accuracy: 0.8634 - val_loss: 0.4075 - val_accuracy: 0.8290 Epoch
98/120
313/313 [==============================] - 6s 20ms/step - loss: 0.3287
- accuracy: 0.8667 - val_loss: 0.4028 - val_accuracy: 0.8240 Epoch
99/120
313/313 [==============================] - 5s 17ms/step - loss: 0.3284
- accuracy: 0.8667 - val_loss: 0.4184 - val_accuracy: 0.8300 Epoch
100/120
313/313 [==============================] - 6s 20ms/step - loss: 0.3286
- accuracy: 0.8675 - val_loss: 0.3950 - val_accuracy: 0.8200 Epoch
101/120
313/313 [==============================] - 5s 16ms/step - loss: 0.3210
- accuracy: 0.8713 - val_loss: 0.4068 - val_accuracy: 0.8100 Epoch
102/120
313/313 [==============================] - 5s 16ms/step - loss: 0.3175
- accuracy: 0.8744 - val_loss: 0.4121 - val_accuracy: 0.8330 Epoch
103/120
313/313 [==============================] - 6s 19ms/step - loss: 0.3167
- accuracy: 0.8711 - val_loss: 0.4059 - val_accuracy: 0.8160 Epoch
104/120
313/313 [==============================] - 6s 20ms/step - loss: 0.3120
- accuracy: 0.8714 - val_loss: 0.4106 - val_accuracy: 0.8200 Epoch
105/120
313/313 [==============================] - 6s 20ms/step - loss: 0.3132
- accuracy: 0.8739 - val_loss: 0.4081 - val_accuracy: 0.8160 Epoch
106/120
313/313 [==============================] - 5s 16ms/step - loss: 0.3055
- accuracy: 0.8742 - val_loss: 0.4080 - val_accuracy: 0.8240 Epoch
107/120
313/313 [==============================] - 6s 19ms/step - loss: 0.3109
- accuracy: 0.8665 - val_loss: 0.4148 - val_accuracy: 0.8210 Epoch
108/120
313/313 [==============================] - 6s 18ms/step - loss: 0.3009
- accuracy: 0.8729 - val_loss: 0.4053 - val_accuracy: 0.8230 Epoch
109/120
313/313 [==============================] - 7s 21ms/step - loss: 0.3046
```

```
- accuracy: 0.8800 - val_loss: 0.4276 - val_accuracy: 0.8260 Epoch 110/120
313/313 [==============================] - 5s 16ms/step - loss: 0.3050
- accuracy: 0.8771 - val_loss: 0.4535 - val_accuracy: 0.8210 Epoch 111/120
313/313 [==============================] - 5s 16ms/step - loss: 0.3037
- accuracy: 0.8776 - val_loss: 0.4194 - val_accuracy: 0.8190 Epoch 112/120
313/313 [==============================] - 6s 19ms/step - loss: 0.2978
- accuracy: 0.8788 - val_loss: 0.4362 - val_accuracy: 0.8210 Epoch 113/120
313/313 [==============================] - 5s 17ms/step - loss: 0.2957
- accuracy: 0.8752 - val_loss: 0.4241 - val_accuracy: 0.8170 Epoch 114/120
313/313 [==============================] - 6s 20ms/step - loss: 0.3037
- accuracy: 0.8748 - val_loss: 0.4211 - val_accuracy: 0.8000 Epoch 115/120
313/313 [==============================] - 6s 18ms/step - loss: 0.2882
- accuracy: 0.8804 - val_loss: 0.4322 - val_accuracy: 0.8110 Epoch 116/120
313/313 [==============================] - 6s 20ms/step - loss: 0.2786
- accuracy: 0.8842 - val_loss: 0.4380 - val_accuracy: 0.8200 Epoch 117/120
313/313 [==============================] - 5s 17ms/step - loss: 0.2886
- accuracy: 0.8861 - val_loss: 0.4436 - val_accuracy: 0.8190 Epoch 118/120
313/313 [==============================] - 6s 20ms/step - loss: 0.2885
- accuracy: 0.8804 - val_loss: 0.4553 - val_accuracy: 0.8060 Epoch 119/120
313/313 [==============================] - 5s 16ms/step - loss: 0.2877
- accuracy: 0.8822 - val_loss: 0.4356 - val_accuracy: 0.8190 Epoch 120/120
313/313 [==============================] - 6s 18ms/step - loss: 0.2837
- accuracy: 0.8837 - val_loss: 0.4348 - val_accuracy: 0.8170
```

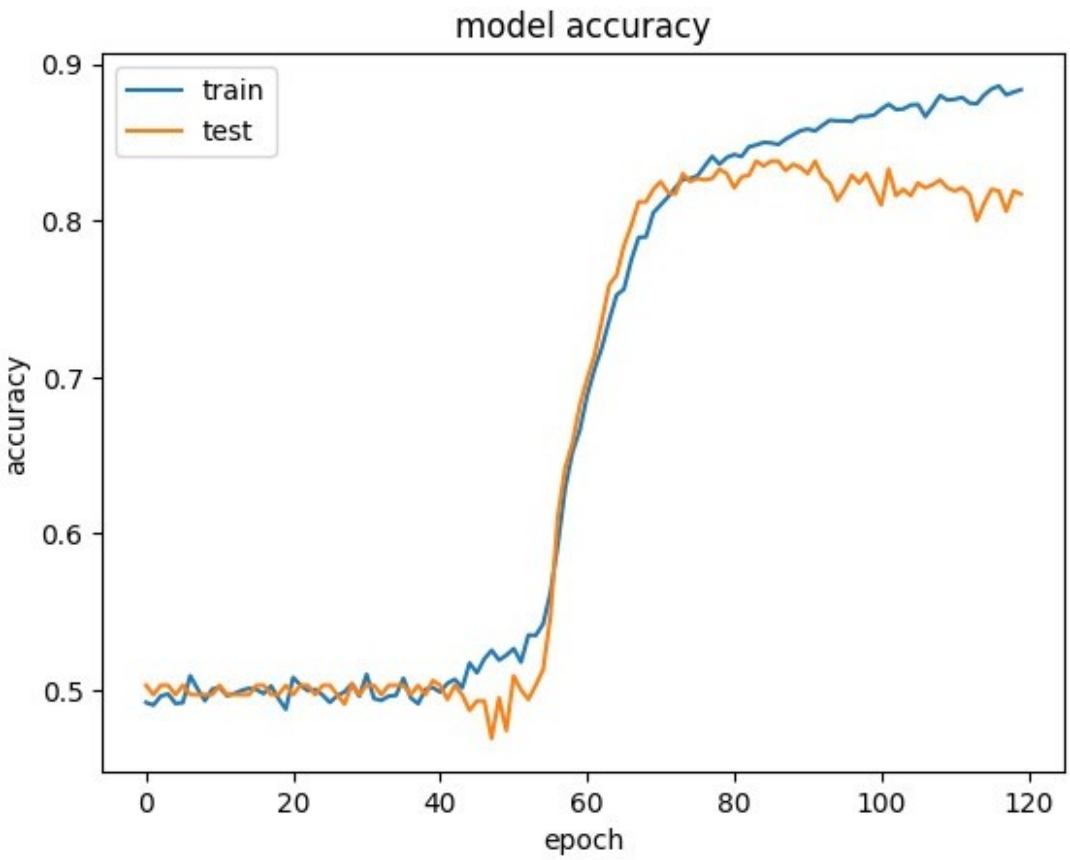
```
filename = 'chatbot.h5'
model.save(filename)
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/engine/ training.py:3103: UserWarning: You are saving your model as an HDF5 file via `model.save()`. This file format is considered legacy. We recommend using instead the native Keras format, e.g. `model.save('my_model.keras')`.
  saving_api.save_model(
```

```
import matplotlib.pyplot as plt
%matplotlib inline print(history.history.keys())
```

```
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy']) plt.title('model
accuracy') plt.ylabel('accuracy') plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left') plt.show()
```

```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```




```
model.load_weights(filename)
pred_results = model.predict(([inputs_test, queries_test]))
```

```
32/32 [==============================] - 1s 9ms/step
```

```
test_data[0][0] ['Mary',
 'got',
 'the',
 'milk',
 'there',
 '.',
```

```
 'John',
 'moved',
 'to',
 'the',
 'bedroom',
 '.']
```

```
story =' '.join(word for word in test_data[0][0]) print(story)
```

Mary got the milk there . John moved to the bedroom .

```
query = ' '.join(word for word in test_data[0][1]) print(query)
```

Is John in the kitchen ?

```
print("True Test Answer from Data is:",test_data[0][2])
```

True Test Answer from Data is: no