Appendix A

Extra Information

A.1 Graphs for the Component Detection Model

Here we present a comparison between training accuracy and validation accuracy of the LSTM Model.

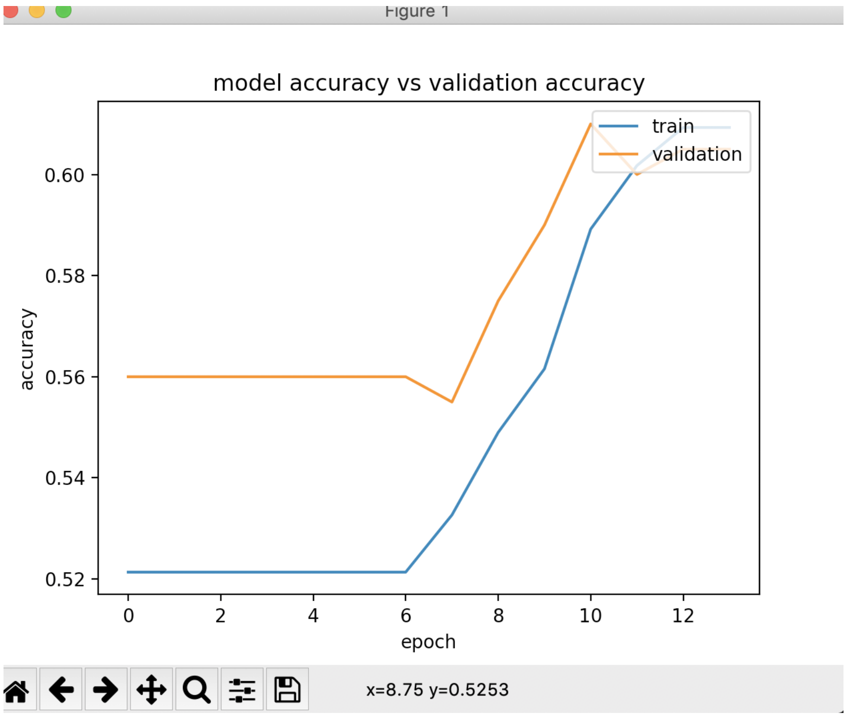

Figure A.1: **Accuracy Comparison**

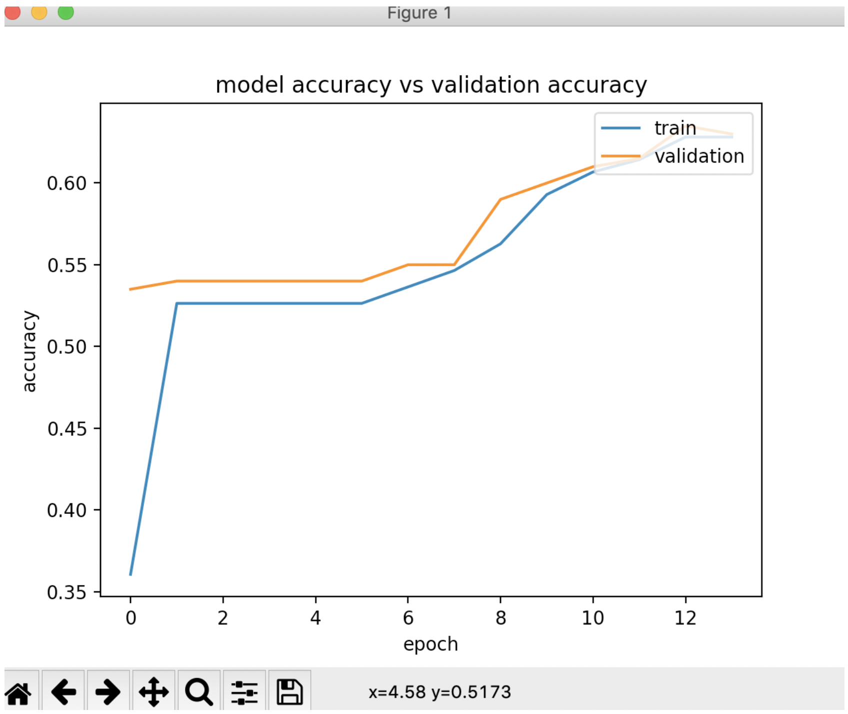

Figure A.2: **Accuracy Comparison**

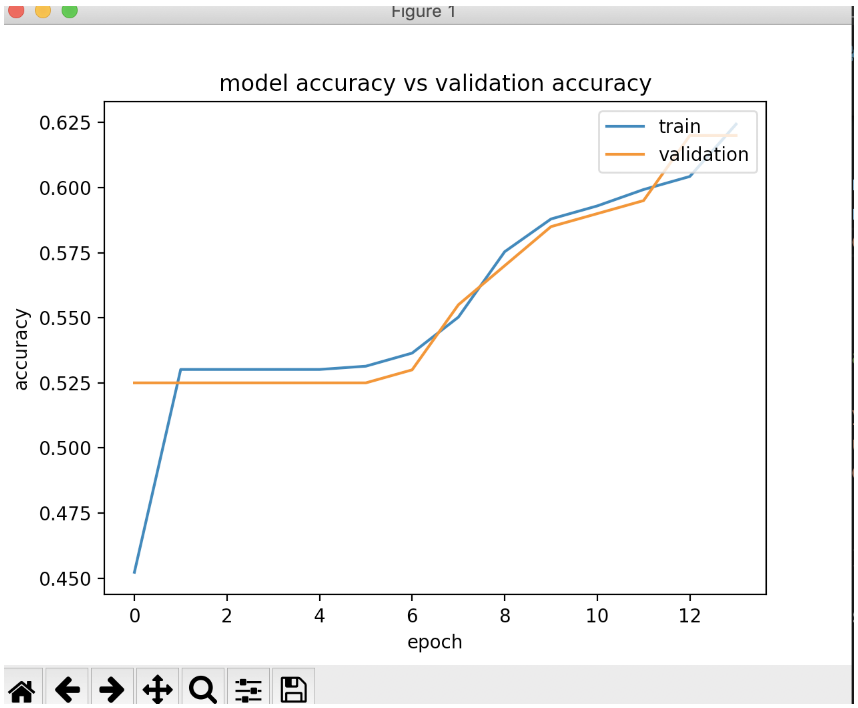

Figure A.3: **Accuracy Comparison**

Appendix B

User Guide

B.1 Instructions

The following instructions will help in running the source code for the models built in this project. The source code is written in Python, and requires the installation of a few libraries.

- Setup an environment that can run Python3 code (Possibly an IDE).
- Before running the code, the libraries used in this project need to be downloaded. The libraries used in this project along are mentioned here. To install a library enter the associated command into a terminal window.
    - scikit-learn : `pip3 install -U scikit-learn`
    - nltk: `pip3 install -U nltk`
    - keras: `pip3 install -U keras`
    - tensorflow: `pip3 install tensorflow`
    - tweepy: `pip3 install tweepy`
    - demoji: `pip3 install demoji`
    - cleantext: `pip3 install cleantext`
    - pandas: `pip3 install pandas`
    - numpy: `pip3 install numpy`
- Once all the libraries are installed, ensure you are in the code folder, and to download the GloVe embedding which is used here enter:

`curl -O http://nlp.stanford.edu/data/glove.twitter.27B.zip`.

This should download a zip file which then needs to be opened. Please ensure this file is placed in the source code folder.

- Once the GloVe embeddings have been downloaded, we can proceed to running the source code.

- To run the tweet extraction file, in the source code folder, enter `python3 tweet_extraction.py` in the terminal. The code should execute, and display the extraction of data happening in the terminal. The final extracted data will then be found in the `tweets_ext.csv` file.

- To run the argument identification model built in this project, in the source code folder, enter `python3 classify.py` in the terminal. The code should execute, and display all results mentioned in the previous chapters.

- To run the argument component detection model built in this project, in the source code folder, enter `python3 arg_component_det.py` in the terminal. The code should execute, and display all results mentioned in the previous chapters.

Appendix C

Source Code

C.1 Instructions

I verify that I am the sole author of the programs contained in this folder, except where explicitly stated to the contrary. Your (typed) signature and the date should follow this statement.

No.	File	Description
1	`tweet_extraction.py`	Python code to interact with the Twitter API and extract tweets
2	`classify.py`	Python code that contains the argument identification model
3	`arg_component_det.py`	Python code that contains the argument component detection model
4	`tweets_ext.csv`	`csv` This file stores the extracted tweets
5	`training1.csv`	`csv` File containing training data
6	`FINAL_RNN.csv`	`csv` File containing training data for the RNN
8	`labelled_twitter1`	File that contains the annotated data set from Twitter
9	`labelled_twitter2`	File that contains the annotated data set from Twitter
10	`credentials.json`	File containing the authentication credentials for Twitter API

Table C.1: The list of files in the project

```
import tweepy
import json
import csv
import demoji
auth_data = json.load(open('credentials.json'))

demoji.download_codes()

#Authentication setup for Twitter API
auth = tweepy.AppAuthHandler(auth_data['CONSUMER_KEY'], auth_data['CONSUMER_SECRET'])
#auth = tweepy.OAuthHandler(auth_data['CONSUMER_KEY'], auth_data['CONSUMER_SECRET'])
#auth.set_access_token(auth_data['ACCESS_KEY'], auth_data['ACCESS_SECRET'])

api = tweepy.API(auth)


i = 0
tweepy.debug(True)

#Using cursoring to join pages of extracted tweets

r = tweepy.Cursor(api.search, q = 'vaccine', count = 100, lang ='en', tweet_mode = 'extended').items()

#Storing the extracted in a csv file

with open('tweets_ext.csv','w') as f1:
 writer=csv.writer(f1, lineterminator="\n")
 for tweet in r:
  i = i+1
  try:
   res = tweet.retweeted_status.full_text.encode('utf-8', 'ignore')
  except AttributeError:  # Not a Retweet
   res= tweet.full_text.encode('utf-8', 'ignore')
  label = (str(i))
  res = demoji.replace(string=((res).decode('utf-8')), repl = '')
  writer.writerow([label, res]),
print(i)


tweepy.debug(True)
```

```
import nltk
from nltk import word_tokenize, pos_tag, ngrams,classify, bigrams
from nltk.classify import MaxentClassifier, NaiveBayesClassifier
from nltk.corpus import movie_reviews, stopwords
from cleantext import clean
import collections
import csv
import pandas as pd
from sklearn.ensemble import VotingClassifier, RandomForestClassifier
from sklearn.metrics import f1_score
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split, KFold, cross_val_score
from sklearn.naive_bayes import GaussianNB
from sklearn.dummy import DummyClassifier
from sklearn import svm, metrics
from sklearn.metrics import accuracy_score
from tensorflow import keras
from tensorflow.keras import layers
from keras.layers import LSTM
from keras.layers.embeddings import Embedding
from keras.preprocessing import sequence

# Extracting the training dataset and storing as a pandas dataframe


dataset = pd.read_csv('training1.csv', encoding='ISO-8859-1')
dataset = dataset[["TEXT", "LABEL"]]

# Testing the model on the Twitter data
test_df = pd.read_csv('labelled_twitter1.csv', encoding = 'ISO-8859-1')
test_df = test_df.append(pd.read_csv('labelled_twitter2.csv', encoding = 'ISO-8859-1'))
test_df = test_df.astype({'LABEL': 'bool'})
test_df = test_df.dropna()
print("test-df:", test_df.head(10))

# Extracting the text from the pandas dataframe

data= dataset.iloc[:,0]

# Tokenizing a piece of raw text, returns a list of tokens.

def tokenize(text):
    tokens = word_tokenize(text)
    return tokens

#Part-Of-Speech Tagging  for a stream of tokens, returns a list of tuples.

def tagger(tokens):
    tags = pos_tag(tokens)
    return tags

# Pre-processing raw data.

def clean_text(words):
    wnl = nltk.stem.WordNetLemmatizer()
    stwords = stopwords.words('english')
    refined = [wnl.lemmatize(word) for word in words if word not in stwords]
    return refined


data = clean_text(data)


# POS Tagging of the dataset using the functions defined above.
```

```python

data_pos = []
for dp in data:
    tags = tagger(tokenize(dp))
    n = 1
    y_2 = [b[n] for b in tags]
    data_pos.append(' '.join(y_2))


#Implementation of the TF-IDF Model, using unigrams and bigrams as features.

matrix2 = TfidfVectorizer( ngram_range=(1,2), lowercase=True)
X = matrix2.fit_transform(data).toarray()


# Display the vocabulary created by the model.
#print("VOCAB:", matrix2.vocabulary_)


# Splitting of the dataset into suitable sizes for training and testing phase.

y= dataset.iloc[:,1]
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.20, train_size=0.80)


# Determining baseline score using a Dummy Classifier
base1 = DummyClassifier(strategy='most_frequent', random_state=0)
base1.fit(X_train, y_train)
base2 = DummyClassifier(strategy='stratified', random_state=0)
base2.fit(X_train, y_train)

print("---------- BASELINE SCORES -------------")
print("BASELINE-1:", base1.score(X_test, y_test))
print("BASELINE-2:", base2.score(X_test, y_test))
print("----------------------------------------")

# Training a Support Vector Machine(SVM) for classification

sv = svm.SVC()
sv.fit(X_train,y_train)


# Testing the SVM classifier on our test data, and further evaluating on different metrics
#
# Metrics used:
# Accuracy -
# F1_Macro Score -
# 5-Cross Validation -
#
#


y_pred = []
for x in X_test:
   y_pred.append(sv.predict(x.reshape(1,-1)))

accuracy = accuracy_score(y_test, y_pred)

scores = cross_val_score(sv, X, y, cv = 5, scoring='f1_macro')
sum = 0
for s in scores:
    sum += s
avg_score = sum/5
print("-----SVM--------------")
print('10-CROSS VALIDATION SCORE: ', avg_score)
print('F1_MACRO: ', f1_score(y_test, y_pred, average='macro'))
print("ACCURACY: ", accuracy)
print("-----------------------")
print("")
print("")



### Training an ensemble classifier of an SVM, a Random Forest Classifier a Naive Bayes Model to compare results

print("-----ENSEMBLE CLASSIFIER------")
clf2 = RandomForestClassifier(n_estimators=50, random_state=1)
clf3 = GaussianNB()
sv2 = svm.SVC()
eclf = VotingClassifier(estimators=[('svm',sv2), ('gnb',clf3), ('rf', clf2)], voting = 'hard')
eclf.fit(X_train, y_train)


y_pred_ensemble = []
for x in X_test:
    y_pred_ensemble.append(eclf.predict(x.reshape(1,-1)))

print("ACCURACY:",eclf.score(X_test, y_test))
print('F1_MACRO: ', f1_score(y_test, y_pred_ensemble, average='macro'))
print("--------------------------------")

#Test on Twitter Data

t_data= test_df.iloc[:,0]
t_data = clean_text(t_data)
t_X = matrix2.transform(t_data).toarray()
t_y= test_df.iloc[:,1]

# Baseline results for twitter data

print("---------- BASELINE SCORES -------------")
print("BASELINE-1:", base1.score(t_X, t_y))
print("BASELINE-2:", base2.score(t_X, t_y))
print("----------------------------------------")

ty_pred = []
for x in t_X:
    ty_pred.append(sv.predict(x.reshape(1,-1)))

accuracy = accuracy_score(t_y, ty_pred)

scores = cross_val_score(sv, t_X, t_y, cv = 5, scoring='f1_macro')
sum = 0
for s in scores:
     sum += s
avg_score = sum/5
print("-----TWITTER SVM------")
print('10-CROSS VALIDATION SCORE: ', avg_score)
print('F1_MACRO: ', f1_score(t_y, ty_pred, average='macro'))
print("ACCURACY: ", accuracy)
```

```python
import numpy as np
import tensorflow
import tensorflow_addons as tfa
import matplotlib.pyplot as pyplot
from tensorflow import keras
from tensorflow.keras import layers
from keras.layers import LSTM, Bidirectional, Input, SpatialDropout1D
from keras.models import Sequential
from keras.layers import Dense, Dropout
from keras.datasets import imdb
from keras.layers.embeddings import Embedding
from keras.utils import to_categorical
from keras.preprocessing import sequence
import csv
from sklearn.dummy import DummyClassifier
from sklearn import svm, metrics
from sklearn.metrics import f1_score
from sklearn.metrics import precision_recall_fscore_support as score
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import LabelEncoder
import nltk
from nltk import word_tokenize, pos_tag, ngrams,classify, bigrams
from nltk.classify import MaxentClassifier, NaiveBayesClassifier
from nltk.corpus import movie_reviews, stopwords
from cleantext import clean
import collections
import pandas as pd
from sklearn.model_selection import train_test_split, KFold, cross_val_score

dataset  = pd.read_csv('FINAL_RNN.csv', encoding='ISO-8859-1')

dataset[dataset.columns[0]] = dataset[dataset.columns[0]].str.replace('\d+','')
y = dataset.iloc[:,1]
data = dataset.iloc[:,0]
sample = dataset.iloc[1,0]

# Encoding the labels into acceptable numeric vectors

le = LabelEncoder()
le.fit(["CLAIM", "BACKING","REBUTTAL","REFUTATION", "PREMISE"])

y = le.transform(y)
y = to_categorical(y)



# Tokenizing a piece of raw text, returns a list of tokens.

def tokenize(text):
    tokens = word_tokenize(text)
    return tokens

#Part-Of-Speech Tagging  for a stream of tokens, returns a list of tuples.

def tagger(tokens):
    tags = pos_tag(tokens)
    return tags

# Pre-processing raw data.

def clean_text(words):
    wnl = nltk.stem.WordNetLemmatizer()
    stwords = stopwords.words('english')
    refined = [wnl.lemmatize(word) for word in words if word not in stwords]
    return refined
```

```python


data = clean_text(data)


# Creating the GloVe Embeddings Dictionary

embedding_dict = {}
#glove2word2vec('glove.twitter.27B.25d.txt', 'word.txt')
with open('glove.twitter.27B.100d.txt', 'r') as glove:
    for line in glove:
        values = line.split()
        word = values[0]
        vectors = np.asarray(values[1:], 'float32')
        embedding_dict[word] = vectors


glove.close()



# Creating a GloVe Matrix



def Glove_matrix(data):
    matrix = np.zeros( (len(data), 100) )
    n = 0
    for dp in data:
        ab = []
        ab = tokenize(dp)
        y_2 = []
        for b in ab:
            if b in embedding_dict:
                y_2.append((embedding_dict[b]))
            else:
                y_2.append( (np.zeros([100,], dtype = np.float32) ))
        y_3 = [list(i) for i in y_2]
        y_3 = np.array(y_3)
        col_mean = y_3.mean(axis=0)
        matrix[n] = col_mean
        n= n+1

    return matrix




#Glove Matrix for predictions
def Glove2(dp):
    matrix2 = np.zeros( (len(data), 100) )
    ab = []
    ab = tokenize(dp)
    y_2 = []
    for b in ab:
        if b in embedding_dict:
            y_2.append((embedding_dict[b]))
        else:
            y_2.append( (np.zeros([100,], dtype = np.float32) ))
    y_3 = [list(i) for i in y_2]
    y_3 = np.array(y_3)
    col_mean = y_3.mean(axis=0)
    matrix2[0] = col_mean
    return matrix2


# Converting the training data into the GloVe matrix
```

```
matrix =(Glove_matrix(data))
mat_arr = np.array(matrix)
X = matrix

# Reshaping to an appropriate shape for the LSTM model

X = X.reshape(len(X), 1, 100)

# Splitting of data into training and test data.

X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.20, train_size=0.80)

# Defining baseline classifiers to measure the performance of the model

base1 = DummyClassifier(strategy='most_frequent', random_state=0)
base1.fit(X_train, y_train)
base2 = DummyClassifier(strategy='stratified', random_state=0)
base2.fit(X_train, y_train)

print("---------- BASELINE SCORES -------------")
print("BASELINE-1:", base1.score(X_test, y_test))
print("BASELINE-2:", base2.score(X_test, y_test))
print("---------------------------------------")




#The RNN Model
# Testing the RNN classifier on our test data, and further evaluating on different metrics
#
# Metrics used:
# Accuracy -
# F1_Macro Score -
# Loss -
#


model = Sequential()
model.add(Input(shape = (1,100)))
model.add(LSTM(units = 64, activation='relu'))
model.add(Dense(units = 5, activation = 'softmax'))
model.compile(loss = 'categorical_crossentropy', optimizer = 'adam', metrics = ['accuracy'])
print(model.summary())
history = model.fit(X_train, y_train, epochs=14 ,validation_data = (X_test, y_test), batch_size = 64)

# Creating a graph to compare the training and validation loss

pyplot.plot(history.history['accuracy'])
pyplot.plot(history.history['val_accuracy'])
pyplot.title('model accuracy vs validation accuracy')
pyplot.ylabel('accuracy')
pyplot.xlabel('epoch')
pyplot.legend(['train', 'validation'], loc='upper right')
pyplot.show()
scores = model.evaluate(X_test, y_test, verbose = 0)
print("ACCURACY:" , (scores[1]))

y_pred = []
for x  in X_test:
    x = x.reshape(len(x),1,100)
    y_pred.append(np.argmax(model.predict(x)))

y_check =[]
for a in y_test:
    y_check.append(np.argmax(a))


f1 = f1_score(y_check, y_pred, average="weighted")
```

```
print("F1 SCORE:", f1)
```