Teammates:

Sri Nikhitha Boddapati – sb4dz@umsystem.edu - 16322565

GitHub link: https://github.com/UMKC-APL-BigDataAnalytics/icp-3-Srinikhitha98

Sukumar Bodapati – sb5zh@umsystem.edu - 16326105

GitHub link: https://github.com/UMKC-APL-BigDataAnalytics/icp-3-sukumarbodapati

Video link: https://youtu.be/QZvAzC68Ozg

ICP - Big data App & Analytics – Deep Learning

Outcomes of ICP:

Difference between the Machine learning and Deep learning

Deep learning is the subset of Machine learning. For example, we have designed a AI bot which makes call to a particular person using the phrase “call Name” but if the user says “can you please make call”. The bot never makes a call, this is when deep learning come into picture where it analyses data based on human intervention. Deep learning works in the format of layers where the data is analyzed at each point of time frame.

How does deep learning work?

Deep learning algorithms analyze data with a predetermined logical structure to reach similar conclusions as humans. Deep learning achieves this by employing a multi-layered structure of algorithms known as neural networks.

Objective:

To perform the sentiment analysis task on this data using one of the Deep Learning Classifier (Keras model) for text.

Tasks:

1.Imported pandas library which is used to imported data from different text formats.

2.Imported numpy library which is used to deal with arrays.

```
[1]  #Importing libraries:
     import pandas as pd
     #Pandas is mainly used for data analysis. Pandas allows importing data from
     import numpy as np
     #numpy is a library which is used to work with arrays
     #NLTK is a toolkit build for working with NLP in Python
     import nltk
     from nltk import sent_tokenize
     from nltk import word_tokenize
```

3.Read the input data using pandas.read.csv and display the input.

4.We need to drop unwanted column from the given dataframe.

```
#DRopping unwanted columns
textdata.drop('id', axis=1, inplace=True)
textdata.head()
```

	label	tweet
0	0	@user when a father is dysfunctional and is s...
1	0	@user @user thanks for #lyft credit i can't us...
2	0	bihday your majesty
3	0	#model i love u take with u all the time in ...
4	0	factsguide: society now #motivation

5.Created an empty list and converted the tweet column data into list and removed all the punctuation and appended them to an empty list.

```
[5]  import re
     list_words=[]
     list_tweet = textdata['tweet'].values.tolist()
     for i in range(len(list_tweet)):
         list_words.append(re.sub(re.compile(r'[^a-zA-Z\s]'),"",list_tweet[i]).lower().strip())
     print(list_words)
     print(len(list_words))

     ['user when a father is dysfunctional and is so selfish he drags his kids into his dysfunction   run', 'user user thanks for lyft credit i cant use ca
     31962
```

6.Remove the stop words and remove the user words as well since the frequency is very high.

```
from nltk.corpus import stopwords
stopwords_final = stopwords.words('english')

stopwords_final.append('user')
textdata['tweet_nosw'] = textdata['tweet_nopun
textdata.head()
```

7.Stemming of data: Stemming is nothing but making a short cut code for the word without any meaning.

```
[9]  #Stemming the words
     from nltk.stem import PorterStemmer

     porter = PorterStemmer()

     stem_words_1=[]
     for w in textdata['tweet_nosw']:
         stem_words_1.append(porter.stem(w))

     stem_words_1[:10]
```

8.lemmatization of data:

Lemmatization converts the words into meaningful short words.

```
[10]  lemma = WordNetLemmatizer()

      #creating a new array to store limitized words
      limitized_words=[]

      for w in textdata['tweet_nosw']:
          limitized_words.append(lemma.lemmatize(w))

      limitized_words[:10]
```

9.Vectorization of the data:

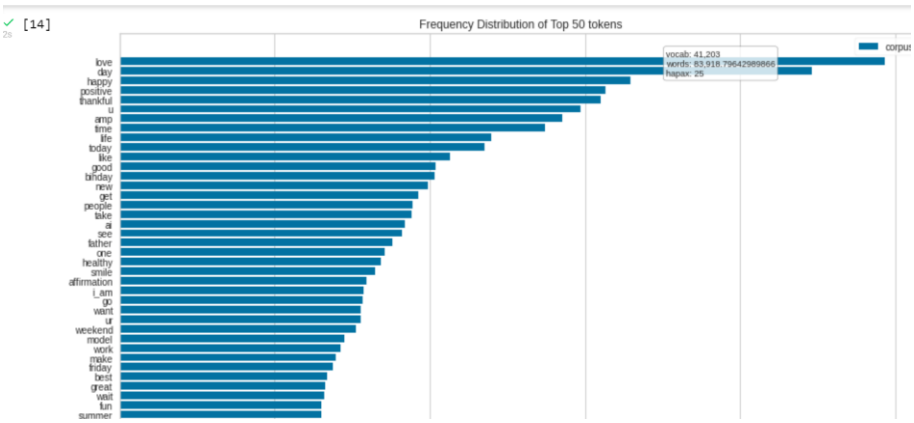
Vectorization makes the data into unique values which will be useful in feature extraction.

```
[14]

     #Total words with their index in model :
     print(tfidf_vectorizer.vocabulary_)
     print("\n")


     #Features :
     key_features_tdidf=tfidf_vectorizer.get_feature_names_out()


     # Displaying the content in a bar chart
     hist_visualizer = FreqDistVisualizer(features=key_features_tdidf, c
     plt.figure(figsize = (15, 10))
     hist_visualizer.fit(words_bunch_tdidf)
     hist_visualizer.show()
```



Model:

1.Import tokenizer from the keras library.

2. Tokenizer turns the text data into vector with coefficients of binary and we can give maximum number of words to keep. (`tokenizer_obj = Tokenizer(num_words=))`

3.Now we will update the vocabulary and convert them into sequence of words.

4.Pad the sequence with the max length of 200.

```
from keras.preprocessing.text import Tokenizer
#tokenizer turns the text data into vector with coefficents of binary
from keras.preprocessing.sequence import pad_sequences
from keras import regularizers
from sklearn.preprocessing import LabelEncoder
# num_words:the maximum number of words to keep, based on word frequency.
tokenizer_obj = Tokenizer(num_words=2000)
#fit_on_texts:Updates internal vocabulary based on a list of texts.
tokenizer_obj.fit_on_texts(limitized_words)
#texts_to_sequences:Transforms each text in texts to a sequence of integers.
sequences = tokenizer_obj.texts_to_sequences(limitized_words)
#this function transforms a list of sequences into a 2D Numpy array of shape.
word_matrix = pad_sequences(sequences, maxlen=150)
print(word_matrix)
print(len(word_matrix))
```

5. For testing and training the data, normalize the label data and assign the value for y.

```
x=word_matrix
#LabelEncoder can be used to normalize labels
encoder=LabelEncoder()
#Transform the  label into normalize them
y=encoder.fit_transform(textdata.label.values)
print(x.shape)
print(y.shape)
```

```
(31962, 150)
(31962,)
```

6.Perform training and testing on the data

```
[17] from sklearn.model_selection import train_test_split
     x_tr, x_te, y_tr, y_te = train_test_split(x,y, test_size = 0.3, stratify
     print(x_tr.shape)
     print(x_te.shape)
     print(y_tr.shape)
     print(y_te.shape)
```

```
(22373, 150)
(9589, 150)
(22373,)
(9589,)
```

7. Layering:

- Embedded: This can be used as only first layer in the model which converts fixed positive integers into dense vector of fixed size.
- LSTM: It uses fast cuDNN implementation. For standard operations like forward and backward convolution, pooling, normalizing, and activation layers, cuDNN provides finely tuned implementations.
- Dense layer:Dense layer makes the dot product with the given input and weight data.It works on the given activation methods.In our model ,we have given Softmax

  Softmax function:

$$\sigma(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

7.Compile the model with the rmspop optimizer with the loss of SparseCategoricalCrossentropy. Instead of considering the learning rate as a hyperparameter, RMSprop uses an adjustable learning rate. SparseCategoricalCrossentropy: The crossentropy measure between the labels and the predictions is calculated.

```
from keras import regularizers
from keras.callbacks import ModelCheckpoint
from keras import layers
from keras.models import Sequential
from keras import backend as K

dl_model = Sequential()
#Turns positive integers (indexes) into dense vectors of fixed size.
dl_model.add(layers.Embedding(2000, 20))
#the layer will use a fast cuDNN implementation
dl_model.add(layers.LSTM(15,dropout=0.5))
# output of dense layer = activation(dot(input, kernel) + bias)
#Activation:Softmax scales numbers/logits into probabilities
dl_model.add(layers.Dense(3,activation='softmax'))

#RMSprop uses an adaptive learning rate instead of treating the learning rate as a hyperparameter.
```

8.After compiling the model, the final output and accuracy of the model.

```
[18]
from tensorflow.keras import utils
# Checking the model outputs for each epoch
model_check = dl_model.fit(x_tr, y_tr, epochs=20,validation_data=(x_te, y_te),callbacks=[checking_point])
```

```
Epoch 1/20
700/700 [==============================] - ETA: 0s - loss: 0.2453 - accuracy: 0.9309
Epoch 00001: saving model to best_model1.hdf5
700/700 [==============================] - 51s 69ms/step - loss: 0.2453 - accuracy: 0.9309 - val_loss: 0.1873 - val_a
Epoch 2/20
700/700 [==============================] - ETA: 0s - loss: 0.1723 - accuracy: 0.9440
Epoch 00002: saving model to best_model1.hdf5
700/700 [==============================] - 49s 70ms/step - loss: 0.1723 - accuracy: 0.9440 - val_loss: 0.1623 - val_a
Epoch 3/20
700/700 [==============================] - ETA: 0s - loss: 0.1611 - accuracy: 0.9464
```

Accuracy of the model:

```
[19] #Evaluating the model to get the final accuracy score
scores=dl_model.evaluate(x_te,y_te,verbose=0)
print("Accuracy : %.2f%%"% (scores[1]*100))
```

```
Accuracy : 95.58%
```

Challenges:

- Building a model in deep learning consumes maximum effort and time, as this topic is completely new.
- Adding different layers in the model is the big challenge but we gained the knowledge on this topic.