|  |  |
| --- | --- |
| Name Of The Student | Aman Rai |
| Internship Project Topic | Automate Detection of different emotions from textual comments and feedback |
| Name of the Organization | TCS iON |
| Name of the Industry Mentor | Mr. Debashis Roy |
| Name of the Institute | Institute of Engineering & Management Kolkata |

|  |  |  |
| --- | --- | --- |
| Date | Day # | Hours Spent |
| 27 August 2020 | 8 | 6+ |
| Activities done during the day: Day 8 I am going to use word embedding and LSTM to build my Model.  **Text Classification Using LSTM and visualize Word Embeddings**  **Import Libraries**  In the following code snippets, I port all the necessary libraries. Kears the library to make the models. I used the ‘Plotly’ for plotting interactive word visualizations. The Bokeh can also be used for this purpose. ‘nltk’ library was used to tokenize the sentence and remove stop-words.  **# Keras** from keras.preprocessing.text import Tokenizer from keras.preprocessing.sequence import pad\_sequences from keras.models import Sequential from keras.layers import Dense, Flatten, LSTM, Conv1D, MaxPooling1D, Dropout, Activation from keras.layers.embeddings import Embedding  **## Plotly** import plotly.offline as py import plotly.graph\_objs as go py.init\_notebook\_mode(connected=True)  **# Others** import nltk import string import numpy as np import pandas as pd from nltk.corpus import stopwords  from sklearn.manifold import TSNE  **Data Processing**  Data processing is one of the vital and must to do a step in exploratory data analysis in any data science project. If the project is related to raw text data the cleaning and processing are musts. In the following subsections, I describe step by step how to clean unnecessary information from raw comments.  **Remove numeric and empty texts**  df = pd.read\_csv(‘train.csv’, sep = ‘|’, names = [‘stars’, ‘text’], error\_bad\_lines=False)  After reading the data, I drop all the null values using pandas ‘dropna’ function. Then filter out the rows with non-numeric characters in the star column. Similarly, I also filtered out all the rows with empty comments.  df= df.dropna() df = df[df.stars.apply(lambda x: x.isnumeric())] df = df[df.stars.apply(lambda x: x !="")] df = df[df.text.apply(lambda x: x !="")]  **Convert ratings into classes (positive = 1 and negative = 0)**  Since the main idea is to identify comments being positive or negative and for simplicity, I convert rating stars into two classes like as below:   * Positive: comments with stars > 3 and * Negative: comments with stars <= 3   labels = df['stars'].map(lambda x : 1 if int(x) > 3 else 0)  **Clean unnecessary text**  In text mining, preprocessing and cleaning is must to do steps. Regex becomes the vital part of this step. Regex can find a pattern in the raw, messy text and perform actions accordingly. Recently I have published an article on the usages of regex on the command line as “[**Text mining on the command line**](https://towardsdatascience.com/text-mining-on-the-command-line-8ee88648476f)” on “Toward data science”. You might find it interesting.  ### Text Normalizing function. Part of the following function was taken from this link.  def clean\_text(text):    ## Remove puncuation  text = text.translate(string.punctuation)    ## Convert words to lower case and split them  text = text.lower().split()    ## Remove stop words  stops = set(stopwords.words("english"))  text = [w for w in text if not w in stops and len(w) >= 3]    text = " ".join(text)  ## Clean the text  text = re.sub(r"[^A-Za-z0-9^,!.\/'+-=]", " ", text)  text = re.sub(r"what's", "what is ", text)  text = re.sub(r"\'s", " ", text)  text = re.sub(r"\'ve", " have ", text)  text = re.sub(r"n't", " not ", text)  text = re.sub(r"i'm", "i am ", text)  text = re.sub(r"\'re", " are ", text)  text = re.sub(r"\'d", " would ", text)  text = re.sub(r"\'ll", " will ", text)  text = re.sub(r",", " ", text)  text = re.sub(r"\.", " ", text)  text = re.sub(r"!", " ! ", text)  text = re.sub(r"\/", " ", text)  text = re.sub(r"\^", " ^ ", text)  text = re.sub(r"\+", " + ", text)  text = re.sub(r"\-", " - ", text)  text = re.sub(r"\=", " = ", text)  text = re.sub(r"'", " ", text)  text = re.sub(r"(\d+)(k)", r"\g<1>000", text)  text = re.sub(r":", " : ", text)  text = re.sub(r" e g ", " eg ", text)  text = re.sub(r" b g ", " bg ", text)  text = re.sub(r" u s ", " american ", text)  text = re.sub(r"\0s", "0", text)  text = re.sub(r" 9 11 ", "911", text)  text = re.sub(r"e - mail", "email", text)  text = re.sub(r"j k", "jk", text)  text = re.sub(r"\s{2,}", " ", text)  ## Stemming  text = text.split()  stemmer = SnowballStemmer('english')  stemmed\_words = [stemmer.stem(word) for word in text]  text = " ".join(stemmed\_words)  return text  # apply the above function to df['text']  df['text'] = df['text'].map(lambda x: clean\_text(x))  In the above code snippet, I used pandas one of the efficient built-in function ‘map’ to be used on pandas Series (single column). ‘Map’ used an external function that takes a string argument and performs some cleaning steps. First, the function removes all the punctuations, then converts all the words in lower case. I used the ‘nltk’ stop-word list to remove them from the text. Later, the function performs some regex operations to clean the unnecessary part of the text. Finally, I used ‘SnowballStemmer’ to stem the words. Stemming is also another important part of NLP.  **Tokenize and Create Sequence**  Tokenization of sentences is one of the essential parts in natural language processing. Tokenization simply divides a sentence into a list of words. I used Keras tokenizer function to tokenize the strings and the used another important function ‘texts\_to\_sequences’ to make sequences of words. More details can be found on the Kears website.  ### Create sequence vocabulary\_size = 20000 tokenizer = Tokenizer(num\_words= vocabulary\_size) tokenizer.fit\_on\_texts(df['text'])sequences = tokenizer.texts\_to\_sequences(df['text']) data = pad\_sequences(sequences, maxlen=50)  **Build a neural network with LSTM**  In the following code snippet, I used Keras library to build a neural network classifier. The network starts with an embedding layer. The layer lets the system expand each token to a more massive vector, allowing the network to represent a word in a meaningful way. The layer takes 20000 as the first argument, which is the size of our vocabulary, and 100 as the second input parameter, which is the dimension of the embedding. The third parameter is the input\_length of 50, which is the length of each comment sequence.  ***## Network architecture***model = Sequential() model.add(Embedding(20000, 100, input\_length=50)) model.add(LSTM(100, dropout=0.2, recurrent\_dropout=0.2)) model.add(Dense(1, activation='sigmoid')) model.compile(loss='binary\_crossentropy', optimizer='adam', metrics=['accuracy'])***## Fit the model***model.fit(data, np.array(labels), validation\_split=0.4, epochs=3)***### Training output***Train on 1004322 samples, validate on 669548 samples Epoch 1/3 1004322/1004322 [==============================] - 7913s - loss: 0.2875 - acc: 0.8776 - val\_loss: 0.2553 - val\_acc: 0.8934 Epoch 2/3 1004322/1004322 [==============================] - 7931s - loss: 0.2454 - acc: 0.8978 - val\_loss: 0.2469 - val\_acc: 0.8975 Epoch 3/3 1004322/1004322 [==============================] - 11974s - loss: 0.2291 - acc: 0.9057 - val\_loss: 0.2530 - val\_acc: 0.8977  **Word embedding visualization**  In this subsection, I want to visualize word embedding weights obtained from trained models. Word embeddings with 100 dimensions are first reduced to 2 dimensions using t-SNE. TensorFlow has an excellent tool to visualize the embeddings in a great way, but here in this tutorial, I just used Plotly to visualize the word in 2D space.  word\_embds = model.layers[0].get\_weights()  ist = [] for word, i in tokenizer.word\_index.items(): word\_list.append(word)  X\_embedded = TSNE(n\_components=2).fit\_transform(word\_weights)number\_of\_words = 1000 trace = go.Scatter(  x = X\_embedded[0:number\_of\_words,0],   y = X\_embedded[0:number\_of\_words, 1],  mode = 'markers',  text= word\_list[0:number\_of\_words] )layout = dict(title= 't-SNE 1 vs t-SNE 2 for sirst 1000 words ',  yaxis = dict(title='t-SNE 2'),  xaxis = dict(title='t-SNE 1'),  hovermode= 'closest')fig = dict(data = [trace], layout= layout) py.iplot(fig)    As we have done with some necessary processing and cleaning, and build a neural network model with LSTM  References:  <https://medium.com/@sabber/classifying-yelp-review-comments-using-lstm-and-word-embeddings-part-1-eb2275e4066b#:~:text=Build%20a%20neural%20network%20with%20LSTM,-In%20the%20following&text=The%20network%20starts%20with%20an,word%20in%20a%20meaningful%20way.>  <https://github.com/krishnaik06/Word-Embedding/blob/master/Untitled2.ipynb>  <https://github.com/krishnaik06/Natural-Language-Processing/blob/master/Toeknization.py>  <https://github.com/krishnaik06/Natural-Language-Processing/blob/master/Toeknization.py>  <https://ieeexplore.ieee.org/abstract/document/8614159> | | |