CS5542 Big Data Apps and Analytics

In Class Programming –4 Report
(Jongkook Son)

Project Overview:

Use the same data (that we obtained by in source code in ICP3 `Data = pd.read_csv('https://raw.githubusercontent.com/dD2405/Twitter_Sentiment_Analysis/master/train.csv')`**) and perform the sentiment analysis task on this data using one of the Deep Learning Classifier (Keras Sequantial model) for text.**

Requirements/Task(s):

1) Data cleaning and preprocessing (at minimum have the following: Removing unnecessary columns or data, Removing Twitter Handles(@user), Removing punctuation, numbers, special characters, Removing stop words, Tokenization, and Stemming, TFIDF vectors, POS tagging, checking for missing values , train/test split of data). (40 points)
2) Deep Learning Model building, adding right combination of layers, and successfully executing the model to make prediction. (50 points)
3) Code quality, Pdf Report quality, video explanation (10 points)

What I learned in ICP:

I could have learned the basics of deep learning model and data preprocessing with keras. First of all, I could have learned how to make deep learning model with keras. In this ICP4, I made a sequential model. Also, I could have learned basic structure of the model. I keep thinking about which layer should I add and which activation fuction should I use to maximize accuracy for model. Still I am confused and do not have intuitive about this concept. I can build some basic structure of model. Finally, using the module in keras I learned how to tokenize, which was very meaningful to me.

ICP description what was the task you were performing and Screen shots that shows the successful execution of each required step of your code

1. Data cleaning and preprocessing

Remove unnecessary column in this case id column is not necessary

▾ Removing unnecessary columns or data

```
[4] #Remove id column
    Data = Data.drop("id", axis=1)
```

```
[5] Data
```

	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
...	...	...
31957	0	ate @user isz that youuu?ð□□□ð□□□ð□□□ð□□□ð□□□ð...
31958	0	to see nina turner on the airwaves trying to...
31959	0	listening to sad songs on a monday morning otw...
31960	1	@user #sikh #temple vandalised in in #calgary,...
31961	0	thank you @user for you follow

31962 rows × 2 columns

Remove Twitter handler by using re module

```
#library for data cleaning
import re
import numpy as np


def remove_handle(text, pattern):
  # finds the pattern @ and put it in a list
  words = re.findall(pattern, text)

  for word in words:
    #remove @ and replace it with blank
    text = re.sub(word,"",text)

  return text

#make a new column named cleaned tweet
Data["Cleaned_Tweets"] = np.vectorize(remove_handle)(Data['tweet'], "@[\w]*")

#Remove tweet column
Data = Data.drop("tweet", axis=1)
```

	label	Cleaned_Tweets
0	0	when a father is dysfunctional and is so sel...
1	0	thanks for #lyft credit i can't use cause th...
2	0	bihday your majesty
3	0	#model i love u take with u all the time in ...
4	0	factsguide: society now #motivation
...	...	...
31957	0	ate isz that youuu?ð□□□ð□□□ð□□□ð□□□ð□□□ð□□□ð□...
31958	0	to see nina turner on the airwaves trying to...
31959	0	listening to sad songs on a monday morning otw...
31960	1	#sikh #temple vandalised in in #calgary, #wso...
31961	0	thank you for you follow

31962 rows × 2 columns

Remove punctuation, numbers, special characters simply using Series.str

Removing punctuation, numbers, special characters

```
[8] #replace everything except characters. Series.str can be used to access the values of the series
    Data["Cleaned_Tweets"] = Data["Cleaned_Tweets"].str.replace("[^a-zA-Z]", " ")
```

```
Data
```

	label	Cleaned_Tweets
0	0	when a father is dysfunctional and is so sel...
1	0	thanks for lyft credit i can t use cause th...
2	0	bihday your majesty
3	0	model i love u take with u all the time in ...
4	0	factsguide society now motivation
...	...	...
31957	0	ate isz that youuu ...
31958	0	to see nina turner on the airwaves trying to...
31959	0	listening to sad songs on a monday morning otw...
31960	1	sikh temple vandalised in in calgary wso...
31961	0	thank you for you follow

31962 rows × 2 columns

Remove stopwords which are loaded from nltk

```
[11] #import stopword from nltk
     from nltk.corpus import stopwords
```

```
stopwords = stopwords.words("english")
print(stopwords)
```

```
['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've", "you'll", "you'd", 'your', 'yours', 'your
```

```
[13] #Series.apply Invoke function on values of Series and remove words in stopword
     Data["Cleaned_Tweets"] = Data["Cleaned_Tweets"].apply(lambda x: " ".join([word for word in x.split() if word not in stopwords]))
```

```
[14] Data
```

	label	Cleaned_Tweets
0	0	father dysfunctional selfish drags kids dysfun...
1	0	thanks lyft credit use cause offer wheelchair ...
2	0	bihday majesty
3	0	model love u take u time ur
4	0	factsguide society motivation
...	...	...
31957	0	ate isz youuu
31958	0	see nina turner airwaves trying wrap mantle ge...
31959	0	listening sad songs monday morning otw work sad
31960	1	sikh temple vandalised calgary wso condemns act
31961	0	thank follow

31962 rows × 2 columns

Tokenization, and Stemming

Tokenization, and Stemming

[15]
```
#Tokenize the text by word
Data["Cleaned_Tweets"] = Data["Cleaned_Tweets"].apply(lambda x: word_tokenize(x))
```

[16]
```
Data
```

	label	Cleaned_Tweets
0	0	[father, dysfunctional, selfish, drags, kids, ...
1	0	[thanks, lyft, credit, use, cause, offer, whee...
2	0	[bihday, majesty]
3	0	[model, love, u, take, u, time, ur]
4	0	[factsguide, society, motivation]
...	...	...
31957	0	[ate, isz, youuu]
31958	0	[see, nina, turner, airwaves, trying, wrap, ma...
31959	0	[listening, sad, songs, monday, morning, otw, ...
31960	1	[sikh, temple, vandalised, calgary, wso, conde...
31961	0	[thank, follow]

31962 rows × 2 columns

7]
```
#Import stemming library
from nltk.stem import PorterStemmer
porter = PorterStemmer()
#Stemming for each Series values
Data["Cleaned_Tweets"] = Data["Cleaned_Tweets"].apply(lambda x: [porter.stem(word) for word in x])
```

```
Data
```

	label	Cleaned_Tweets
0	0	[father, dysfunct, selfish, drag, kid, dysfunc...
1	0	[thank, lyft, credit, use, caus, offer, wheelc...
2	0	[bihday, majesti]
3	0	[model, love, u, take, u, time, ur]
4	0	[factsguid, societi, motiv]
...	...	...
31957	0	[ate, isz, youuu]
31958	0	[see, nina, turner, airwav, tri, wrap, mantl, ...
31959	0	[listen, sad, song, monday, morn, otw, work, sad]
31960	1	[sikh, templ, vandalis, calgari, wso, condemn,...
31961	0	[thank, follow]

31962 rows × 2 columns

POS tagging

Visualization of postag(Using yellowbrick)

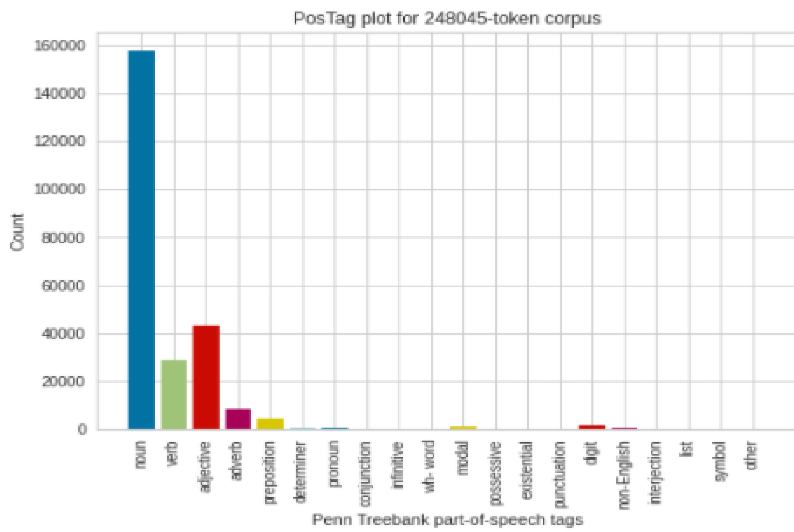
```
[19] # list for contain postagged words
     tagged_words = []

     #Pos tagging eac
     for words in Data["Cleaned_Tweets"]:
       tagged_words.append(nltk.pos_tag(words))

     #Change the form to use in yellowbrick
     tagged_words = [tagged_words]
```

```
#library to visualizae postag
from yellowbrick.text import PosTagVisualizer
```

```
# Create the visualizer, fit, score, and show it
viz = PosTagVisualizer()
viz.fit(tagged_words)
viz.show()
```




```
<matplotlib.axes._subplots.AxesSubplot at 0x7effe85c4ac8>
```

Tokenization by keras

Tokenization using keras

```
[24] from keras.preprocessing.text import Tokenizer
     from keras.preprocessing.sequence import pad_sequences
     from keras.models import Sequential
     from keras import layers
```

```
[25] from sklearn.model_selection import train_test_split
```

```
[26] # Maximum number of words to work with 5000
     tokenizer = Tokenizer(num_words=5000)
```

```
     # Updates internal vocabulary based on a list of texts.
     tokenizer.fit_on_texts(Data["Cleaned_Tweets"].values)
```

```
[29] # Return list of sequences (one per text input).
     X = tokenizer.texts_to_sequences(Data["Cleaned_Tweets"].values)
     X
```

```
[30] maxlen = 100
     # ensure that all sequences in a list have the same length.
     X = pad_sequences(X, padding="post", maxlen=maxlen)
```

```
[31] X

     array([[  16, 2122, 1806, ...,    0,    0,    0],
            [   5, 4548, 1946, ...,    0,    0,    0],
            [  20, 2846,    0, ...,    0,    0,    0],
            ...,
            [ 230,   64,  269, ...,    0,    0,    0],
            [1392, 1096, 1434, ...,    0,    0,    0],
            [   5,   47,    0, ...,    0,    0,    0]], dtype=int32)
```

Train/Test split for data

```
[32] X.shape

     (31962, 100)
```

```
[33] # Target value for the training
     y= Data["label"].values
```

```
[34] # calculate vocab size based on word index of tokenized words
     vocab_size = len(tokenizer.word_index) + 1
```

train/test split of data

```
[ ] #70% training and 30% test
    x_train, x_test, y_train, y_test = train_test_split(X,y,train_size=0.7, test_size=0.3,random_state=15)
```

```
[ ] print(x_test.shape, y_test.shape)

    (9589, 100) (9589,)
```

```
[ ] print(x_train.shape, y_train.shape)

    (22373, 100) (22373,)
```

Simple deep learning model

Simple deeplearning model

```
embedding_dim = 50

model = Sequential()
# embeding layer
model.add(layers.Embedding(input_dim=vocab_size,
                           output_dim=embedding_dim,
                           input_length=maxlen))
# flattening
model.add(layers.Flatten())

# Add a Dense layer with 10 units.
model.add(layers.Dense(10, activation='relu'))
model.add(layers.Dropout(0.2))

# Add a Dense layer with 5 units.
model.add(layers.Dense(5, activation='relu'))
model.add(layers.Dropout(0.2))


model.add(layers.Dense(1, activation='sigmoid'))

# compile the model with chosen parameters
model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])

# print summary of the model
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, 100, 50)           1566300
_________________________________________________________________
flatten (Flatten)            (None, 5000)              0
_________________________________________________________________
dense (Dense)                (None, 10)                50010
_________________________________________________________________
dropout (Dropout)            (None, 10)                0
_________________________________________________________________
dense_1 (Dense)              (None, 5)                 55
_________________________________________________________________
dropout_1 (Dropout)          (None, 5)                 0
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 6
=================================================================
Total params: 1,616,371
Trainable params: 1,616,371
Non-trainable params: 0
_________________________________________________________________
```

```
model.fit(x_train, y_train,
                    epochs=10,
                    verbose=True,
                    validation_data=(x_test, y_test),
                    batch_size=30)
```

```
Epoch 1/10
746/746 [==============================] - 14s 18ms/step - loss: 0.2202 - accuracy: 0.9306 - val_loss: 0.1467 - val_accuracy: 0.9280
Epoch 2/10
746/746 [==============================] - 13s 18ms/step - loss: 0.1219 - accuracy: 0.9361 - val_loss: 0.1472 - val_accuracy: 0.9547
Epoch 3/10
746/746 [==============================] - 13s 18ms/step - loss: 0.0919 - accuracy: 0.9729 - val_loss: 0.1725 - val_accuracy: 0.9567
Epoch 4/10
746/746 [==============================] - 13s 18ms/step - loss: 0.0720 - accuracy: 0.9810 - val_loss: 0.2061 - val_accuracy: 0.9543
Epoch 5/10
746/746 [==============================] - 14s 18ms/step - loss: 0.0528 - accuracy: 0.9864 - val_loss: 0.2318 - val_accuracy: 0.9531
Epoch 6/10
746/746 [==============================] - 13s 18ms/step - loss: 0.0424 - accuracy: 0.9898 - val_loss: 0.3121 - val_accuracy: 0.9563
Epoch 7/10
746/746 [==============================] - 14s 18ms/step - loss: 0.0339 - accuracy: 0.9918 - val_loss: 0.3821 - val_accuracy: 0.9549
Epoch 8/10
746/746 [==============================] - 14s 18ms/step - loss: 0.0291 - accuracy: 0.9929 - val_loss: 0.4712 - val_accuracy: 0.9564
Epoch 9/10
746/746 [==============================] - 14s 19ms/step - loss: 0.0262 - accuracy: 0.9934 - val_loss: 0.5225 - val_accuracy: 0.9563
Epoch 10/10
746/746 [==============================] - 14s 18ms/step - loss: 0.0247 - accuracy: 0.9933 - val_loss: 0.5697 - val_accuracy: 0.9554
<tensorflow.python.keras.callbacks.History at 0x7f66a56cc518>
```

Model evaluation

```
# Model evaluation
scores = model.evaluate(x_test, y_test, verbose=2)
print("Accuracy: %.2f%%" % (scores[1]*100))
```

```
300/300 - 0s - loss: 0.5697 - accuracy: 0.9554
Accuracy: 95.54%
```

RNN model with LSTM layer

Recurrent Neural Networks (RNN) with Keras

```
[35] model2 = Sequential()
```

```
[36] model2.add(layers.Embedding(input_dim=vocab_size,
                              output_dim=128,
                              input_length=X.shape[1]))
```

```
[37] model2.add(layers.SpatialDropout1D(0.5))
```

```
[38] #Add a LSTM layer with 185 internal units.

     model2.add(layers.LSTM(185, dropout=0.4, recurrent_dropout=0.4))
```

```
[39] # Add a Dense layer with 2 units, using softmax activation and using categorical_crossentropy

     model2.add(layers.Dense(2,activation='softmax'))
     model2.compile(loss = 'categorical_crossentropy', optimizer='adam',metrics = ['accuracy'])
```

```
[39]
```

```
# To adjust the shape to the
Y = pd.get_dummies(Data['label']).values

# train test split for the changed value
x_train, x_test, y_train, y_test = train_test_split(X,Y,train_size=0.7, test_size=0.3,random_state=15)
```

```
[41] model2.summary()
```

Model summary and evaluation

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, 100, 128)          4009728
_________________________________________________________________
spatial_dropout1d (SpatialDr (None, 100, 128)          0
_________________________________________________________________
lstm (LSTM)                  (None, 185)               232360
_________________________________________________________________
dense (Dense)                (None, 2)                 372
=================================================================
Total params: 4,242,460
Trainable params: 4,242,460
Non-trainable params: 0
_________________________________________________________________
```

```
[42] print(X_train.shape, Y_train.shape)
```

```
(21414, 100) (21414, 2)
```

```
[43] model2.fit(X_train, Y_train, epochs = 3, batch_size=30, verbose=True)
```

```
Epoch 1/3
714/714 [==============================] - 360s 505ms/step - loss: 0.2600 - accuracy: 0.9290
Epoch 2/3
714/714 [==============================] - 364s 510ms/step - loss: 0.2555 - accuracy: 0.9300
Epoch 3/3
714/714 [==============================] - 361s 505ms/step - loss: 0.2553 - accuracy: 0.9300
<tensorflow.python.keras.callbacks.History at 0x7fc3520a1e10>
```

```
[45] # Model evaluation
     scores = model2.evaluate(X_test, Y_test, verbose=2)
     print("Accuracy: %.2f%%" % (scores[1]*100))
```

```
330/330 - 20s - loss: 0.2549 - accuracy: 0.9297
Accuracy: 92.97%
```

Challenges that I faced:

The most difficult challenge that I faced was that I was not used to building model deep learning. But I overcame this problem by exploring many materials that explain model building. Keras and tensorflow documentation was especcially helpful for me.

Video link

https://www.youtube.com/watch?v=_PJs6zGnpr4