Text Classification Using LSTM and visualize Word Embeddings: Part-1

Sabber Ahamed Follow
Jan 10, 2018 · 7 min read

Introduction

Yelp round-10 review datasets contain a lot of metadata that can be mined and used to infer meaning, business attributes, and sentiment. In this tutorial, I used the datasets to find out the positive or negative reviews.

For simplicity, I classify the review comments into two classes: either as positive or negative. Reviews that have a star higher than three are regarded as positive while the reviews by star less than or equal to three are negative. Therefore, the problem is a supervised learning.

To build and train the model, I first clean the text and convert them into sequences. Each review comment is limited to 50 words. Short texts less than 50 words are padded with zeros and the long ones are truncated. After processing the review comments, I trained three models in three different ways and obtained three different words-embeddings. Four parts of this tutorial are organized as follows:

- **Part-1**: In this part, I build a neural network with LSTM and word embeddings were learned while fitting the neural network.
- **Part-2**: In in this part, I add an extra 1D convolutional layer on top of the LSTM layer to reduce the training time.
- **Part-3**: In this part-3, I use the same network architecture as part-2, but use the pre-trained glove 100 dimension word embedding as initial input.
- **Part-4**: In part-4, I use word2vec to learn word embedding.

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.

1. Remove numeric and empty texts

First, read the data:

```
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 !="")]
```

2. 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:

- (1) Positive: comments with stars > 3 and
- (2) Negative: comments with stars <= 3

```
labels = df['stars'].map(lambda x : 1 if int(x) > 3 else 0)
```

2. 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**" on "Toward data science". You might find it interesting.

Because of the computational expenses, I use the top 20000 unique words. First, tokenize the comments then convert those into sequences. I keep 50 words to limit the number of words in each comment.

```
### 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.

3. 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.

1. Get embedding weights from the glove

```
word_embds = model.layers[0].get_weights()[
```

2. Get word list

```
ist = []
for word, i in tokenizer.word_index.items():
    word_list.append(word)
```

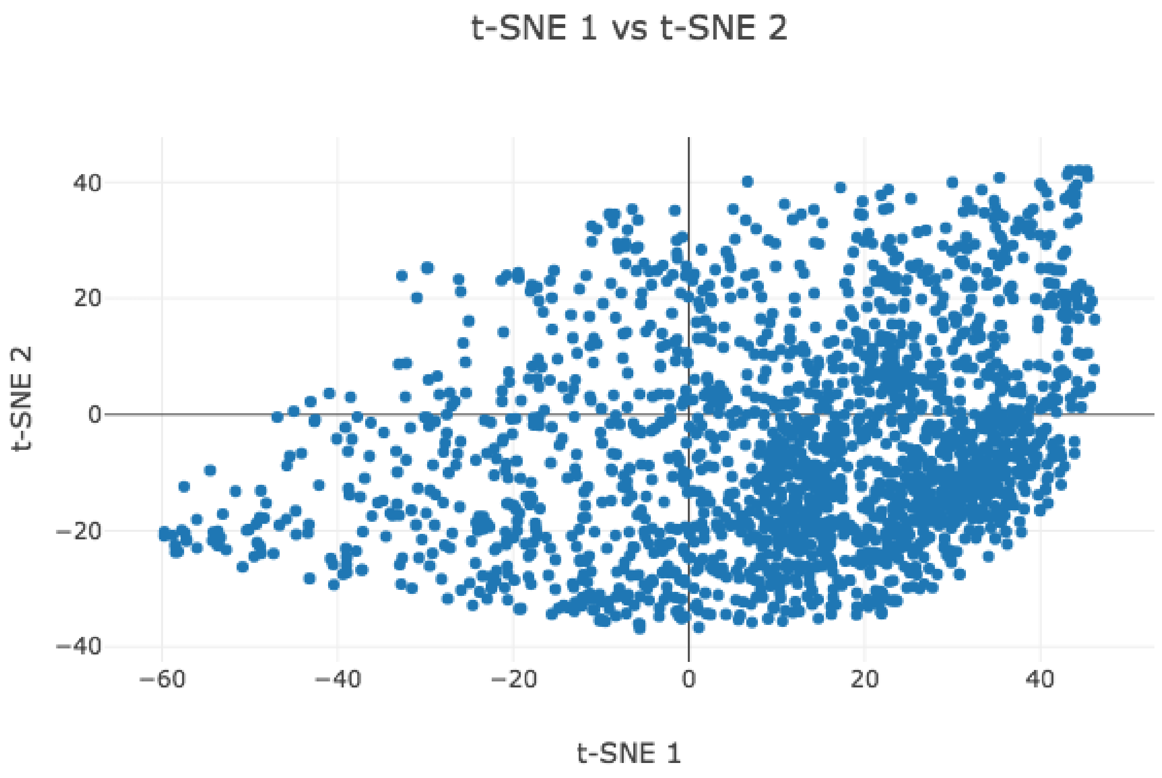
3. Scatter plot of the first two components of TSNE

```
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)
```



Let's discuss

Thank you very much for reading. The full code can be found on Github. As we have done with some necessary processing and cleaning, and build a neural network model with LSTM, in the next tutorial I will discuss how to add an extra 1D Convolutional layer on top of the LSTM layer to reduce the training time. Until then, if you have any questions, feel free to ask. Please make comments if you see any typos, mistakes or you have better suggestions. You can reach out to me:

```
Email: sabbers@gmail.com
LinkedIn: https://www.linkedin.com/in/sabber-ahamed/
Github: https://github.com/msahamed
Medium: https://medium.com/@sabber/
```

Machine Learning   Word Embeddings   NLP   Data Science