Twitter Sentiment Analysis Using an LSTM RNN

By: Zack Godwin

Overview

Brand management is a key factor for a successful business. Understanding how customers feel about a brand, company, or even a specific topic can help predict how they'll respond to future decisions.

Think Nike and the Kaepernick ad.

Twitter gives us the perfect data stream to investigate this.

Project Goals

- ➔ Stream at least 200,000 tweets from Twitter using the Twitter API.
  - Store data in a Postgres database.
- ➔ Use the TextBlob sentiment analysis module to **label** the tweets.
- ➔ Using the labelled data set, build a classification model to predict sentiment.
- ➔ Train a RNN classifier and compare performance.
  - Investigate the impact of different degrees of text preprocessing on the performance of the RNN.

Data Capture Methodology

- Using the Twitter API I wrote a python script to collect Tweets and deposit the data into my Postgres database.
- My Tweet filter was programmed to grab anything with the word "trump".
- I collected over 400,000 tweets.

```python
import tweepy
from tweepy import Stream
from tweepy import OAuthHandler
from tweepy.streaming import StreamListener
import time
import json
```

```python
import psycopg2

# Connect using psycopg2
conn = psycopg2.connect(dbname= "Final Project", user = "postgres",
                        password = "", host="127.0.0.1",
                        port = "5433")

curs = conn.cursor()
```

```python
ckey = ''
csecret = ''
atoken = ''
asecret = ''
```

```python
class listener(StreamListener):

    def on_data(self, data):
            try:
                all_data = json.loads(data)
                tweet = all_data["text"]
                username = all_data["user"]["screen_name"]
                location = all_data["user"]["location"]

                curs.execute("INSERT INTO trump (time, location, username, t
                    (time.time(), location, username, tweet))

                conn.commit()

                print((username,tweet))

                return True
            except KeyError:
                pass

        def on_error(self, status):
            if status_code == 420:
                #returning False in on_data disconnects the stream
                return False


auth = tweepy.OAuthHandler(ckey, csecret)
auth.set_access_token(atoken, asecret)
twitterStream = Stream(auth, listener())
twitterStream.filter(track=['trump'], languages=["en"])
```

The Data

- Dataframe Dimension:
  - **430,228** Rows
- Mean Word Count:
  - **18.7 Words** (+/- 6.1)
- Mean Character Count:
  - **123 Characters** (+/- 33.7)

	tweet	word_count	character_count
0	'RT @SethAbramson: In case you missed it: what...	25	155
1	'RT @RonWyden: Incredible. More luxury travel ...	22	149
2	'President Trump Directed Michael Cohen To Lie...	16	125
3	'RT @TeaPainUSA: Tea would wager that Trump in...	26	147
4	'RT @BruceBartlett: There is one person in Ame...	24	147

	word_count	character_count
count	430228.000000	430228.000000
mean	18.674238	122.972138
std	6.134861	33.739280
min	1.000000	1.000000
25%	16.000000	122.000000
50%	20.000000	140.000000
75%	23.000000	140.000000
max	37.000000	258.000000

Data Cleaning

I compared two methods of data cleaning, the principal difference being the presence of stop-words.

filtered_tweet	clean
case missed fuss tonight retweet think followe...	case you missed what all the fuss about ton...
incredible luxury travel trump administration ...	incredible more luxury travel from the trump a...
president trump directed michael cohen lie con...	president trump directed michael cohen lie c...
tea wager trump instructed folks lie congress ...	tea would wager that trump instructed all his ...
person america trump chose rupert murdoch medi...	there one person america who could somethin...

```
#Second Cleaning Function that keeps stopwords
def extract_text(text):

    # Convert to string
    text = text.astype(str)

    # Remove URLs
    text = text.str.replace('https?://[A-Za-z0-9./]+','')

    # Keep Hashtag text
    text = text.str.replace("[^a-zA-Z]", " ")

    # Make lowercase
    text = text.apply(lambda x: " ".join(x.lower() for x in x.split()))

    # Remove whitespaces
    text = text.apply(lambda x: " ".join(x.strip() for x in x.split()))

    # Remove special characters
    text = text.apply(lambda x: "".join(
        [" " if ord(i) < 32 or ord(i) > 126 else i for i in x]))

    # Remove punctuation
    text = text.str.replace('[^\w\s]', '')

    # Remove numbers
    text = text.str.replace('\d+', '')

    #Remove 1-2 letter clutter remnants
    text = text.apply(lambda x: re.sub(r'\b\w{1,2}\b', '', x))

    return text

# remove  RT: @user
df['clean'] = np.vectorize(remove_pattern)(df['tweet'], "RT @[\w]*")
```

Sentiment Labeling with TextBlob

The sentiment property of TextBlob returns a named tuple of the form Sentiment(polarity, subjectivity).

The polarity score is a float within the range [-1.0, 1.0]. We will use this to label tweets as positive, neutral, or negative.

```
def analyze_sentiment(tweet):
    '''
    Utility function to classify the polarity of a tweet
    using textblob.
    '''
    analysis = TextBlob(tweet)
    if analysis.sentiment.polarity > 0:
        return 1
    elif analysis.sentiment.polarity == 0:
        return 0
    else:
        return -1

# Create a column with the result of the analysis:
df['SA'] = np.array(
    [analyze_sentiment(tweet) for tweet in df['filtered_tweet']])

df['SA2'] = np.array(
    [analyze_sentiment(tweet) for tweet in df['clean']])

df.head()
```

Sentiment Labeling Discrepancy

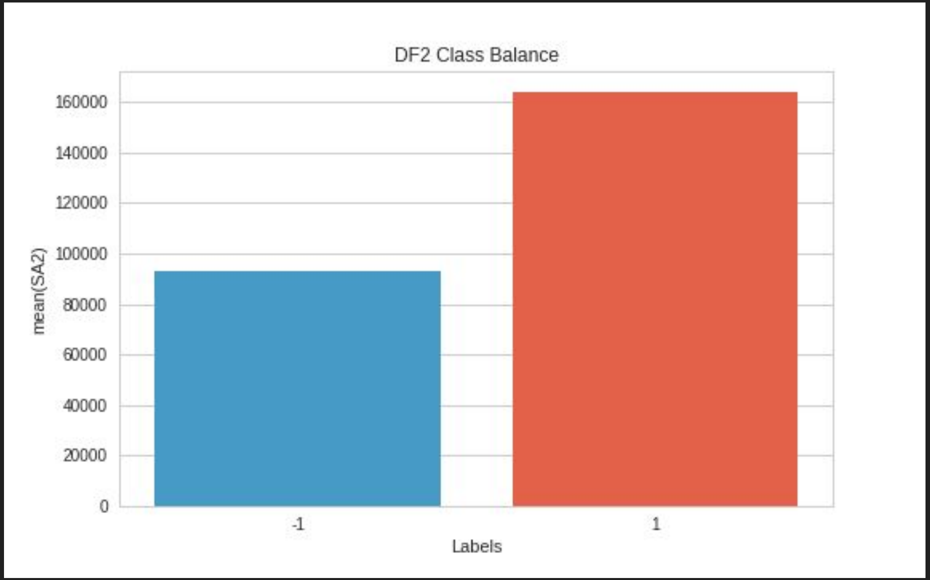
It appears TextBlob performs better on text containing stopwords.

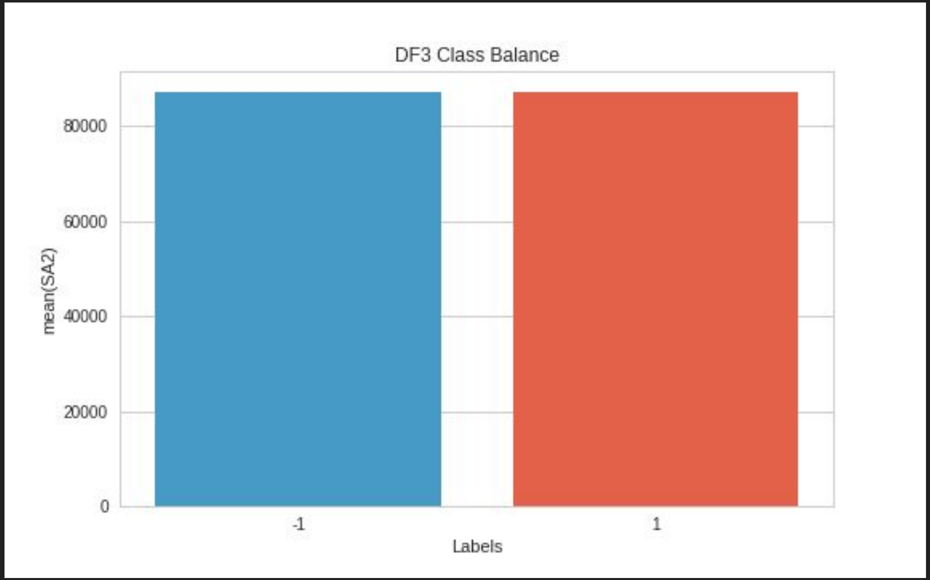
As such, I used ‘clean’ and SA2 as my data and target labels (respectively).

“RT @TeaPainUSA: Tea would wager that Trump instructed all his folks to lie to Congress because he knew Nunes and his other GOP imps and dem\u2026”

tweet	word_count	character_count	filtered_tweet	clean	SA	SA2
'RT @TeaPainUSA: Tea would wager that Trump in...	26	147	tea wager trump instructed folks lie congress ...	tea would wager that trump instructed all his ...	0	-1

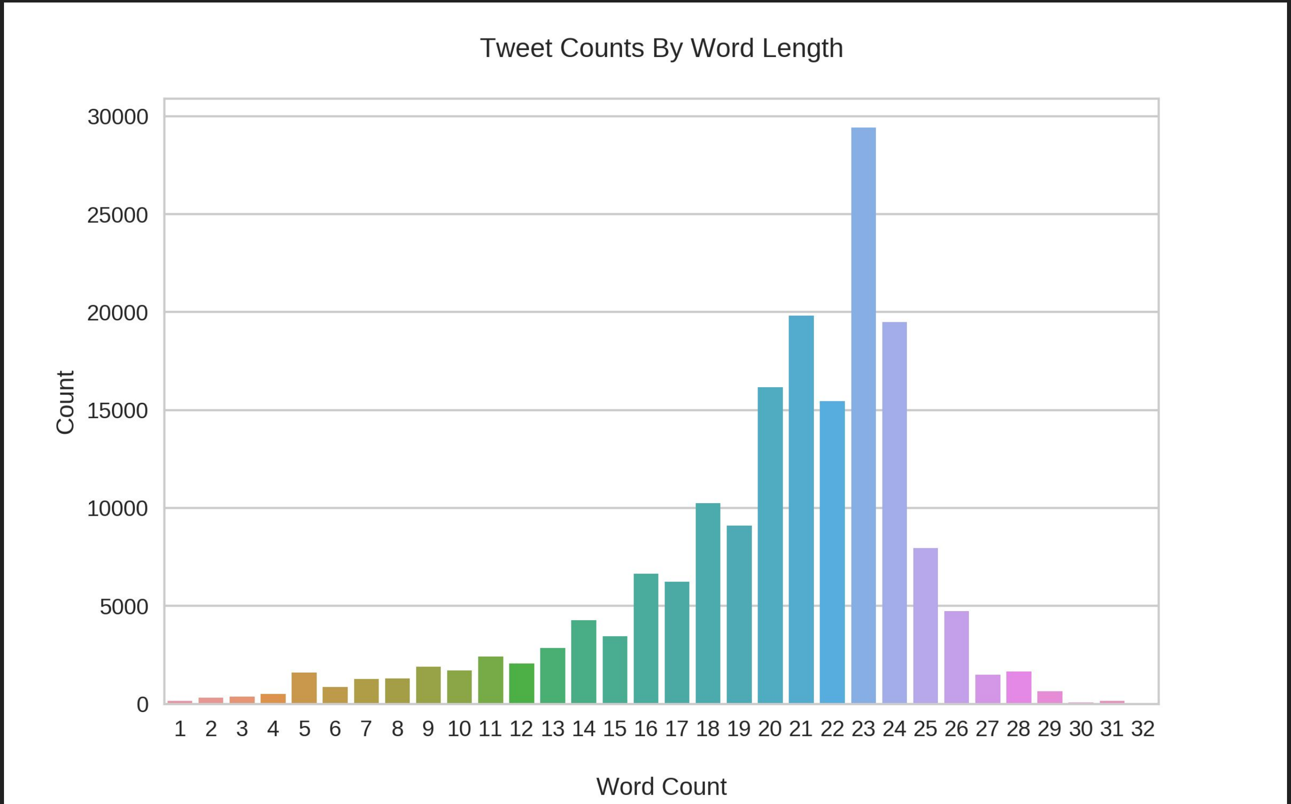
EDA: Class Balance



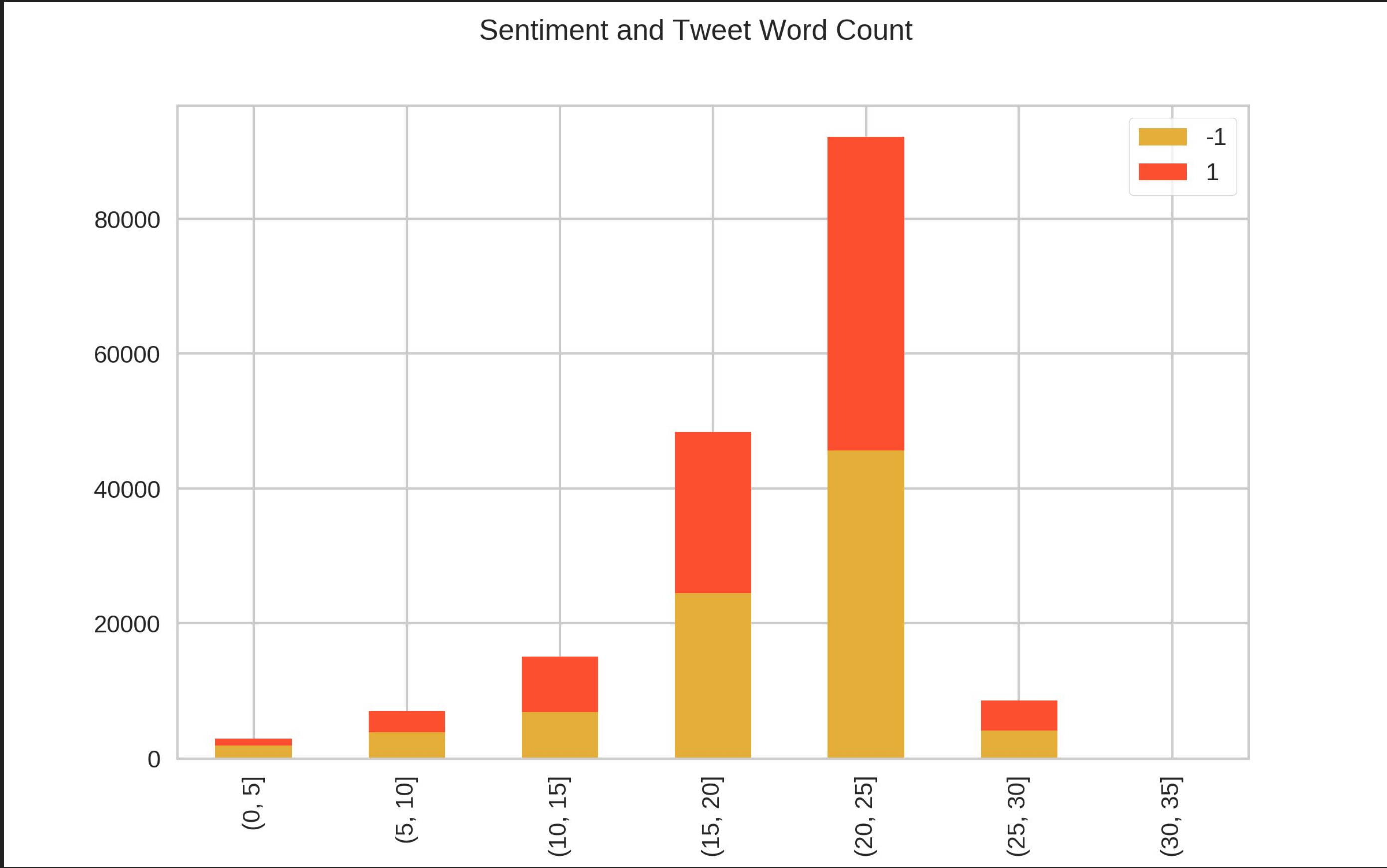


"In a rare move, Mueller's office denies BuzzFeed report that Trump told Cohen to lie about Moscow project"
By Devlin Barrett , Matt Zapotosky and Karoun Demirjian
January 18

EDA:


Tweet Counts By Word Length
Count
30000
25000
20000
15000
10000
5000
0
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32
Word Count

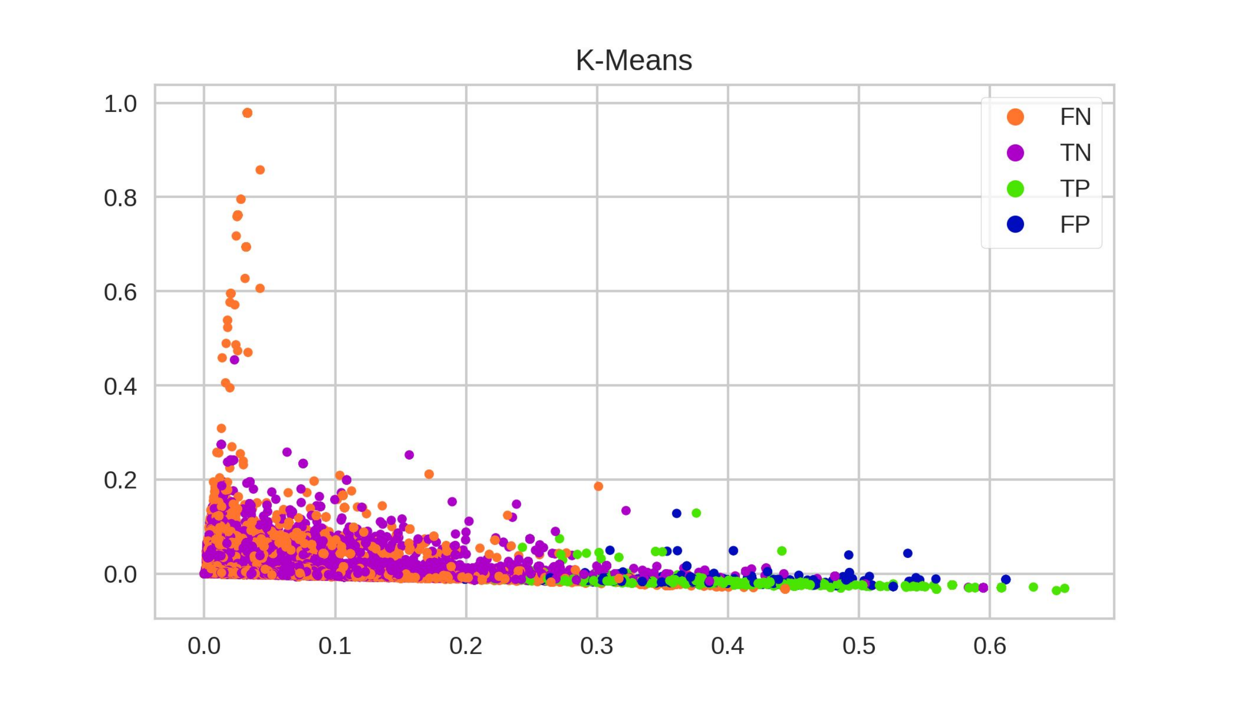
EDA:


Sentiment and Tweet Word Count
-1
1
80000
60000
40000
20000
0
(0, 5]
(5, 10]
(10, 15]
(15, 20]
(20, 25]
(25, 30]
(30, 35]

EDA: K-Means Cluster Analysis

I found the results to vary highly from run to run, this indicates that the data does not cluster well with k-means.

If the data is really suited for k-means clustering, the results would be stable.



EDA: Word2Vec

Word2vec is a two-layer neural net that processes text.

The algorithm takes in data in the shape of a list of lists:
[['combine', 'the', 'military', 'service', 'trump', 'tim', 'moore', 'richard', 'burr', 'thom', 'tillis', 'mark', 'meadows', 'newt', 'gingrich', 'ann', 'coulter'], ['sitting', 'president', 'can', 'solicit', 'donations', 'for', 'his', 'election', 'campaign', 'while', 'spreading', 'propaganda', 'utilize', 'portion']]

The output is a set of vectors: feature vectors for words in that corpus.

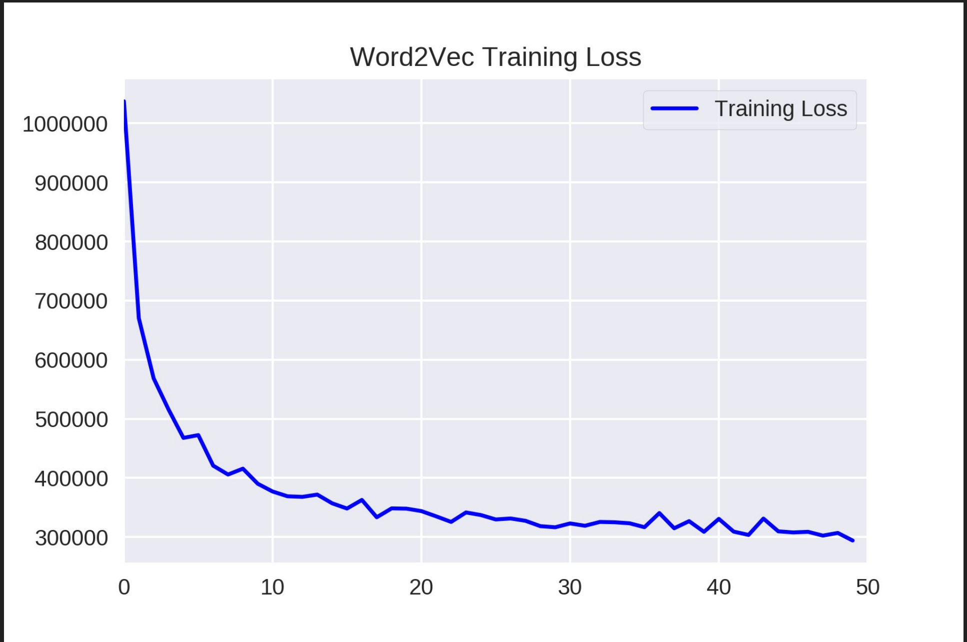
EDA: Word2Vec

```
[19] model.wv.most_similar('good', topn=5)

[('great', 0.5110733509063721),
 ('excellent', 0.46075239777565),
 ('jlg', 0.4202117621898651),
 ('bad', 0.4178803861141205),
 ('plugs', 0.3837891221046448)]

[20] model.similarity('bad', 'good')

0.41788036
```



Not ideal...

EDA: FastText

FastText is an extension to Word2Vec proposed by Facebook in 2016.

Instead of feeding individual words into the Neural Network, FastText breaks words into several n-grams.

```
from gensim.models import FastText
fast_model = FastText(
    sentences, size=300, window=5, min_count=5, workers=4,sg=0)
```

```
fast_model.wv.most_similar("bad")
```

```
[('badly', 0.6813015341758728),
 ('badass', 0.6336409449577332),
 ('vlad', 0.6131807565689087),
 ('load', 0.5794876217842102),
 ('dad', 0.5165011882781982),
 ('knead', 0.503669023513794),
 ('glad', 0.49534744024276733),
 ('bath', 0.48387226462364197),
 ('bags', 0.47988539934158325),
 ('brad', 0.46785059571266174)]
```

EDA: Doc2Vec

Doc2Vec is a small extension to the CBOW Word2Vec model.

Instead of using just words to predict the next word, we also add another feature vector, which is document-unique.

Each tweet is considered a separate document.

EDA: Doc2Vec Classifiers

```
[44]  # Random Forest Classifier
      rfc = RandomForestClassifier(n_estimators=100, n_jobs=-1)
      auto_model(rfc, "Random Forest Classifier",
                 X_train, X_test, y_train, y_test)
```

```
Random Forest Classifier
done in 174.446s
Testing accuracy 0.89
Testing F1 score: 0.89
```

```
[41]  # Gaussian Naive Bayes
      gnb = GaussianNB()
      auto_model(gnb, "Gaussian Naive Bayes",
                 X_train, X_test, y_train, y_test)
```

```
Gaussian Naive Bayes
done in 0.449s
Testing accuracy 0.67
Testing F1 score: 0.67
```

```
[42]  # Logistic Regression
      log = LogisticRegression(solver='lbfgs')
      auto_model(log, "Logistic Regression", X_train, X_test, y_train, y_test)
```

```
Logistic Regression
done in 1.124s
Testing accuracy 0.65
Testing F1 score: 0.65
```

```
[43]  # Linear SVC
      svc = LinearSVC()
      auto_model(svc, "Linear SVC", X_train, X_test, y_train, y_test)
```

```
Linear SVC
done in 10.991s
Testing accuracy 0.68
Testing F1 score: 0.68
```

Random Forest has the highest Accuracy and F1, but it's not exactly fast.

Pytorch: LSTM RNN

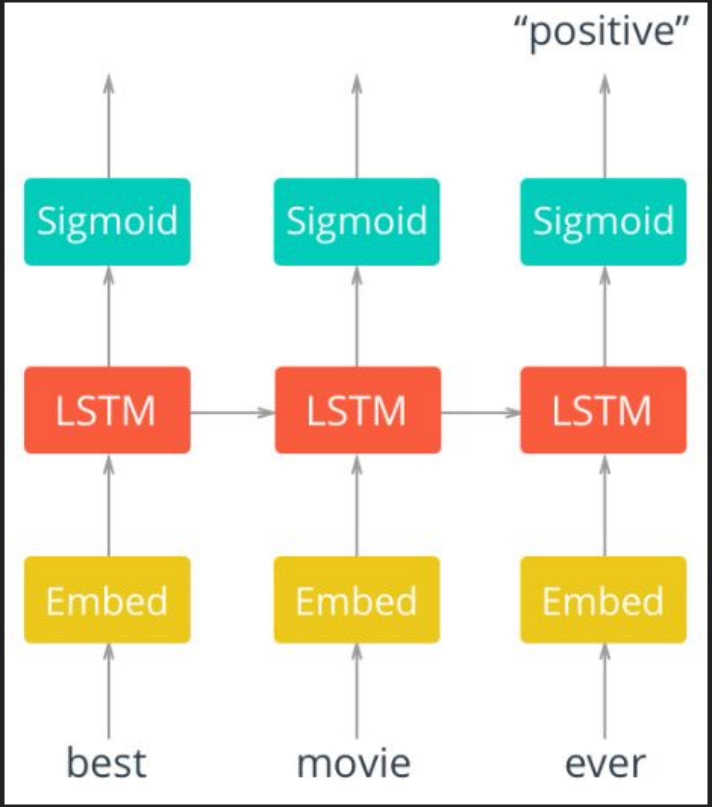
Long Short-Term Memory (LSTM) recurrent neural networks manage to keep contextual information of inputs by integrating a loop that allows information to flow from one step to the next.

Since tweets are composed from a sequence of words, and the specific order of those words provide context to the sentiment of the tweet, it stands to reason this Neural Network should predict tweet sentiment well.

Pytorch: LSTM RNN

<u>The layers are as follows:</u>

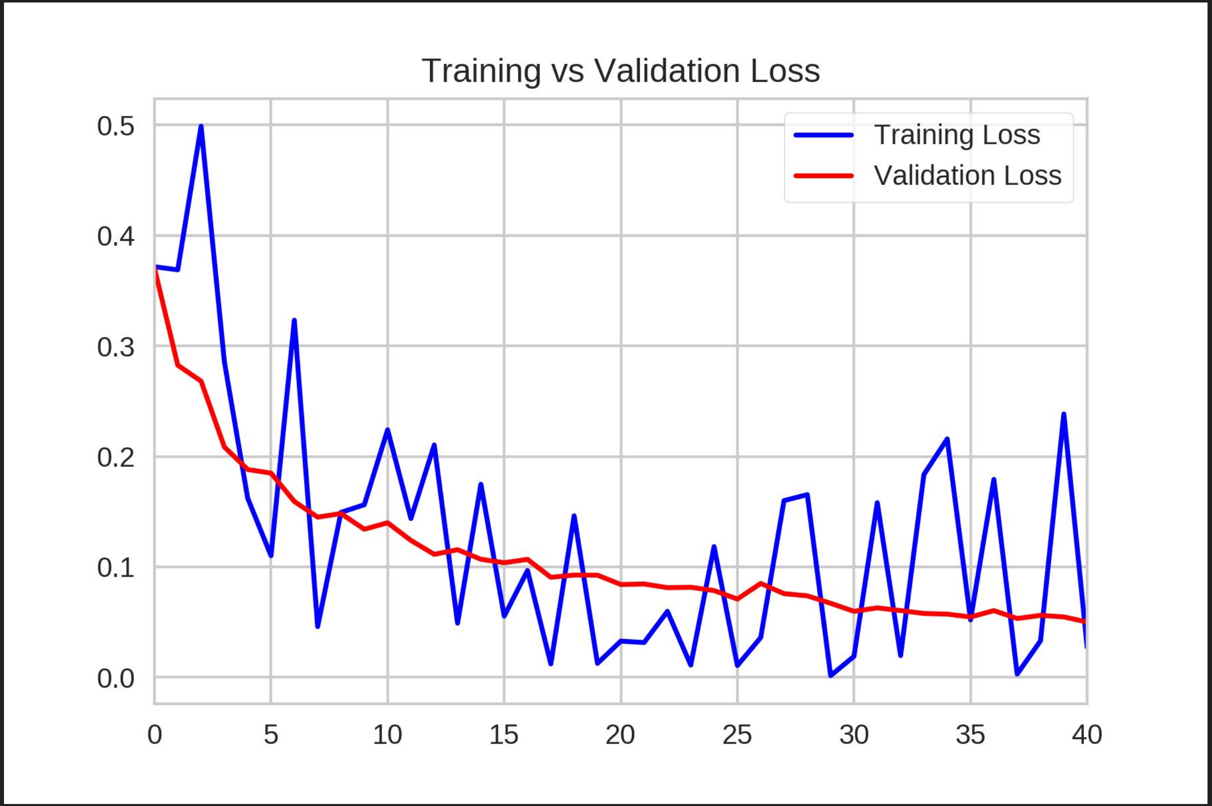
1. An embedding layer that converts our word tokens (integers) into embeddings of a specific size.
2. An LSTM layer defined by a hidden_state size and number of layers
3. A fully-connected output layer that maps the LSTM layer outputs to a desired output_size
4. A sigmoid activation layer which turns all outputs into a value 0-1; return only the last sigmoid output as the output of this network.



Pytorch: Word Embeddings

1. Convert column of cleaned tweets to a giant list of words (words), and also a list of tweets (tweet_split).
2. Build dictionary to pair words to integers.
3. Use the dictionary to tokenize each tweet in tweet_split
   - susansarandon nytimes what wrong with you why are you doing this again you must   closet trump supporter wtf
4. Store the tokenized tweet in list (tweet_ints).
   - [3157, 739, 34, 135, 18, 8, 86, 11, 8, 437, 6, 145, 8, 221, 7626, 2, 991, 519]
5. Pad the tokenized tweets with zeros so all tokenized tweets are equal dimensions.
   - [0,   0,   0,   0,   0,   0,   0,   0, 3157, 739,   34,
     135,   18,   8,   86,   11,   8, 437,   6, 145,   8, 221,
     7626,   2, 991, 519]

Pytorch: Model Training



Pytorch: Model Performance

```
Test loss: 0.050
Test accuracy: 0.983

Confusion Matrix:
 Predicted     0     1    All
True
0           8578   151   8729
1            145  8556   8701
All         8723  8707  17430

                    Classification Report
               precision    recall  f1-score   support

           0       0.98      0.98      0.98      8729
           1       0.98      0.98      0.98      8701

   micro avg       0.98      0.98      0.98     17430
   macro avg       0.98      0.98      0.98     17430
weighted avg       0.98      0.98      0.98     17430
```

Live Demo

Questions?

Extra Slides