



Research Computing

---

# PYTHON SENTIMENT ANALYSIS

Jacalyn Huband  
13 October 2020

# Sentiment Analysis

A computational technique for detecting the polarity (i.e., positive, neutral, or negative) of the meaning associated with a phrase, sentence, paragraph, or document.

# Sentiment Analysis

Sentiment analysis often is used to assess customer service. Are emails, tweets, comments positive or negative?

Sentiment analysis can be used to determine overall user experiences with a product, or (dis)satisfaction of customer service.

# Sentiment Analysis: Types

## Polarity-based:

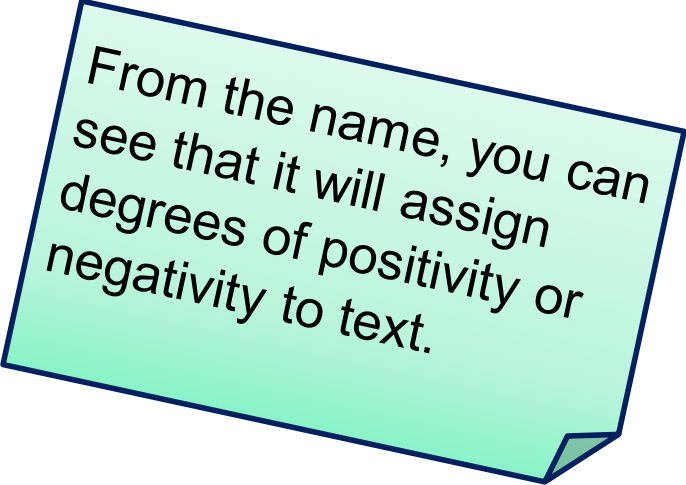
Words are defined simply as positive, negative, or neutral

## Valence-based:

Words are defined by the degree of positivity or negativity (e.g., “the worst” would be more negative than “bad”)

# Sentiment Analysis: VADER

We are going to look at using a Python package, called VADER (**V**alence **A**ware **D**ictionary for s**E**ntiment **R**easoning).



From the name, you can see that it will assign degrees of positivity or negativity to text.

# Sentiment Analysis

CAVEAT: Sentiment analysis is still difficult for a machine to perform. There are no definitive rules for determining the emotion of feeling behind a written expression.

# The Idea behind VADER

To determine the sentiment of a text, VADER looks at individual words.

It uses a built-in lexicon – a dictionary of words where the words have a sentiment rating.

It uses the ratings to determine the percentage of the text that is positive, neutral, or negative.

Finally, it computes an overall score between -1 (most negative) and +1 (most positive)

# Hands-on Activity

- Install the VADER package:

```
pip install vaderSentiment
```

- Or, if you are on Rivanna:

```
pip install --user vaderSentiment
```



# Simple Example of VADER

## Python

```
from vaderSentiment.vaderSentiment import  
SentimentIntensityAnalyzer  
  
#Set up analyzer  
analyzer = SentimentIntensityAnalyzer()  
  
#Feed in a sentence  
sentence = "I hate scary movies."  
score = analyzer.polarity_scores(sentence)  
print(score)
```

This code is available as a Jupyter Notebook at  
[https://github.com/jhuband/Sentiment\\_Analysis.git](https://github.com/jhuband/Sentiment_Analysis.git)

# Simple Example of VADER

## Python

```
from vaderSentiment.vaderSentiment import  
SentimentIntensityAnalyzer
```

```
#Set up analyzer
```

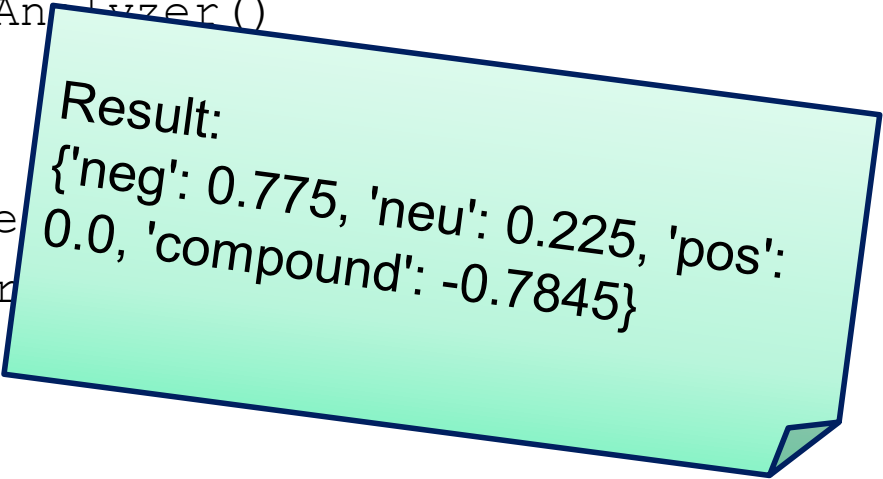
```
analyzer = SentimentIntensityAnalyzer()
```

```
#Feed in a sentence
```

```
sentence = "I hate scary movie"
```

```
score = analyzer.polarity_score(sen
```

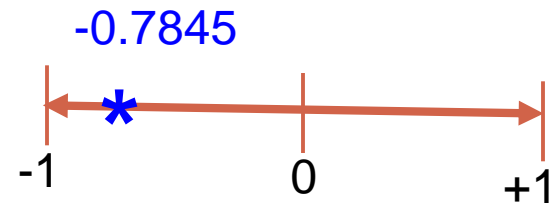
```
print(score)
```



**Result:**  
{'neg': 0.775, 'neu': 0.225, 'pos':  
0.0, 'compound': -0.7845}

# The Results

- The `polarity_score` function returns a dictionary with the percentages of negative, neutral, and positive sentiment, and the compound score.
- In this example, we have
  - negative: 0.775,
  - neutral: 0.225,
  - positive: 0.0,
  - compound: -0.7845



# More power behind VADER

- VADER uses more than just the sentiment ratings of words.
  - It also looks at capitalizations and some punctuations.
  - To determine the sentiment of some text, VADER looks at individual words.

## Hands-on Activity

- In the previous example, change the sentence to the following: “I HATE scary movies!” and rerun the analyzer.
- Did the compound score change?

# LONG SHORT-TERM MEMORY

# What is Long Short-Term Memory?

An example of a recurrent neural network that can handle sequences of data.

The sequences can have dependencies based on time or distance.

These networks can be used to analyze text, videos, and time-series data.

# The Idea behind LSTMs

Often times information builds upon previous information.

Example: What would be the next word in the sentence

“The clouds are in the \_\_\_\_\_”

# The Idea behind LSTMs

The more distant the relevant information, the less likely a recurrent neural network will determine the relationship.

Example: What would be the next word in the sentence

“I grew up in Mexico. It was a wonderful childhood, and I am fluent in \_\_\_\_\_”



# The Idea behind LSTMs

Perhaps a more relevant example:

To fully understand *The Avengers: Endgame*, you would need to watch several of the previous Avenger movies.



Images borrowed from:

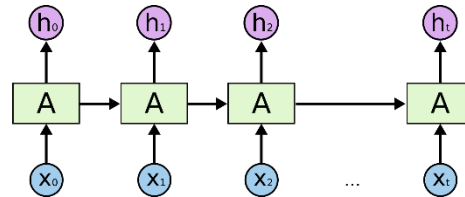
[https://en.wikipedia.org/wiki/Thor\\_\(Marvel\\_Cinematic\\_Universe\)](https://en.wikipedia.org/wiki/Thor_(Marvel_Cinematic_Universe))

<https://alteregocomics.com/avengers-infinity-war-thor-1-6-scale-figure/>

<https://nytimespost.com/avengers-endgame-chris-hemsworth-reveals-the-worst-part-about-fat-thor/>

# The Idea behind LSTMs

A Recurrent Neural Network (RNN) passes on information



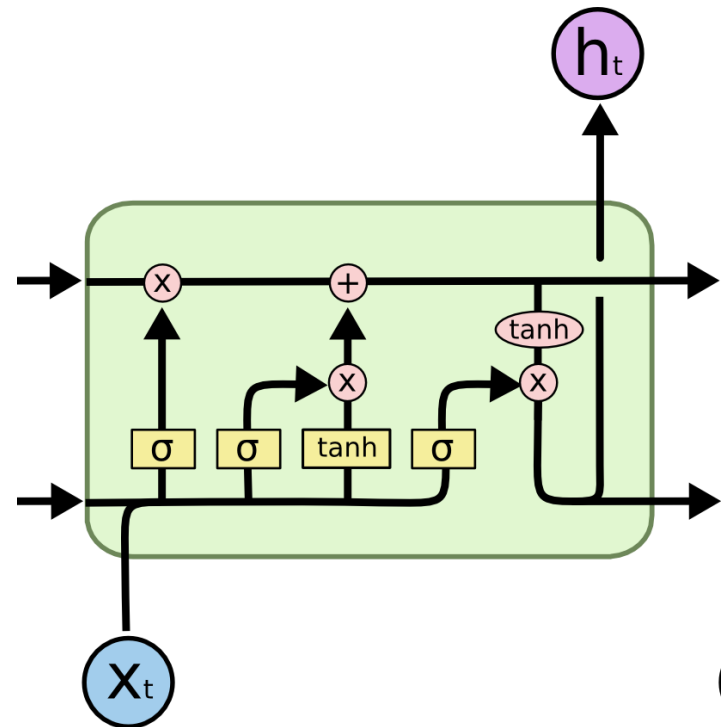
A drawback for RNNs is that the more distant the information, the more likely the information will be forgotten.

LSTMs must have a mechanism to maintain a history of what has gone before each object.

They do this by adding more detailed structures within each node..

# Building Blocks of LSTMs

- The nodes, called cells, process the inputs in 4 ways.
  1. Determine which current information should be thrown away (forget gate)
  2. Determine what new information should be kept (input gate)
  3. Update the current state of the cell
  4. Determine what information will be passed on (output gate)



Images borrowed from

<https://colah.github.io/posts/2015-08-Understanding-LSTMs/>

# Coding LSTM: General Steps

1. Load the tensor flow/keras packages
  2. Read in the data
  3. Pre-process the data
    - a. Simplify to lower case and no punctuation
    - b. Tokenize into words and convert to sequences
    - c. Split into training data & testing data
  4. Define the model
  5. Configure the Learning Process
- 
6. Fit the model to the training data
  7. Apply the model to test data & evaluate results

# 1. Load Keras Packages

## Python

```
import numpy as np
import pandas as pd

from sklearn.feature_extraction.text import
CountVectorizer
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras.models import Sequential
from keras.layers import Dense, Embedding, LSTM,
SpatialDropout1D
from sklearn.model_selection import train_test_split
from keras.utils.np_utils import to_categorical
import re
```

## 2. Read in the Data

### Python

```
data = pd.read_csv('Sentiment.csv')  
  
# Keep only the neccessary columns  
data = data[['text', 'sentiment']]  
print(data.head())
```

## 3a. Pre-process Data: Simplify

### Python

```
# Remove tweets that are neutral
data = data[data.sentiment != "Neutral"]

# Convert text to lower case without punctuation
data['text'] = data['text'].apply(lambda x: x.lower())
data['text'] = data['text'].apply((lambda x:
re.sub('[^a-zA-z0-9\s]', '', x)))

# Display number of positive & negative tweets
print(data[ data['sentiment'] == 'Positive'].size)
print(data[ data['sentiment'] == 'Negative'].size)
```

## 3b. Pre-process Data: Sequences

### Python

```
for idx, row in data.iterrows():
    row[0] = row[0].replace('rt', ' ')

max_fatures = 2000
tokenizer = Tokenizer(num_words=max_fatures, split=' ')
tokenizer.fit_on_texts(data['text'].values)
X = tokenizer.texts_to_sequences(data['text'].values)
X = pad_sequences(X)
```



## 3c. Pre-process Data: Split

### Python

```
Y = pd.get_dummies(data['sentiment']).values
X_train, X_test, Y_train, Y_test =
train_test_split(X, Y, test_size = 0.33, random_state =
42)
print(X_train.shape, Y_train.shape)
print(X_test.shape, Y_test.shape)
```

## 4. Define the Model

### Python

```
embed_dim = 128
lstm_out = 196

model = Sequential()
model.add(Embedding(max_features, embed_dim, input_length
= X.shape[1]))
model.add(SpatialDropout1D(0.4))
model.add(LSTM(lstm_out, dropout=0.2,
recurrent_dropout=0.2))
model.add(Dense(2, activation='softmax'))
```

## 5. Configure the Learning Process

### Python

```
model.compile(loss = 'categorical_crossentropy',  
optimizer='adam',metrics = ['accuracy'])  
print(model.summary())
```

## 6. Fit the Model

### Python

```
batch_size = 32  
model.fit(X_train, Y_train, epochs = 7,  
batch_size=batch_size, verbose = 1)
```

## 7. Apply the Model to Test Data

### Python

```
validation_size = 1500

X_validate = X_test[-validation_size:]
Y_validate = Y_test[-validation_size:]
X_test = X_test[:-validation_size]
Y_test = Y_test[:-validation_size]
```

## 8. Evaluate the Results

### Python

```
# Final evaluation of the model
score, acc = model.evaluate(X_test, Y_test, verbose = 2,
batch_size = batch_size)

print("score: %.2f" % (score))
print("acc: %.2f" % (acc))
```

# Bonus Step: Apply model to new item

## Python

```
twt = ['Meetings: Because none of us is as dumb as all  
of us.']
```

```
# Pre-process tweet
```

```
twt = re.sub('[^a-zA-z0-9\s]', '', twt[0].lower())
```

```
twt = tokenizer.texts_to_sequences(twt)
```

```
twt = pad_sequences(twt, maxlen=28, dtype='int32',  
value=0)
```

# Bonus Step: Apply model to new item

## Python

```
# Run through the model
sentiment = model.predict(twt, batch_size=1, verbose =
2)[0]

#State result
if(np.argmax(sentiment) == 0):
    print("**The tweet is negative.**")
elif (np.argmax(sentiment) == 1):
    print("**The tweet is positive,**")
```



# Activity #7: LSTM Program

- Make sure that you can run the CNN code:

Python

`Py_ex7_LSTM.ipynb`

This Notebook is available at  
[https://github.com/jhuband/Sentiment\\_Analysis.git](https://github.com/jhuband/Sentiment_Analysis.git)

# NEED MORE HELP?

## Office Hours via Zoom

Tuesdays: 3 pm - 5 pm  
Wednesdays: 3 pm – 5 pm  
Thursdays: 10 am - noon

Zoom Links are available at  
<https://www.rc.virginia.edu/support/#office-hours>

---

**Website:**

<https://rc.virginia.edu>

**Email:**

[hpc-support@virginia.edu](mailto:hpc-support@virginia.edu)

# QUESTIONS?

---

