Sentiment Analysis on Amazon Reviews Using Word2Vec + LSTMs

Mahmoud Hossam - 222125
Haidy Haitham - 212539

Table of contents

Table of contents

Sentiment Analysis

An introduction to what Sentiment Analysis is

Sentiment analysis, also referred to as opinion mining, is an approach to natural language processing (NLP) that identifies the emotional tone behind a body of text. This is a popular way for organizations to determine and categorize opinions about a product, service or idea.

Data Preparation

The methods used for preparing the data

The methods used for Data Preparation

01 Word Tokenization

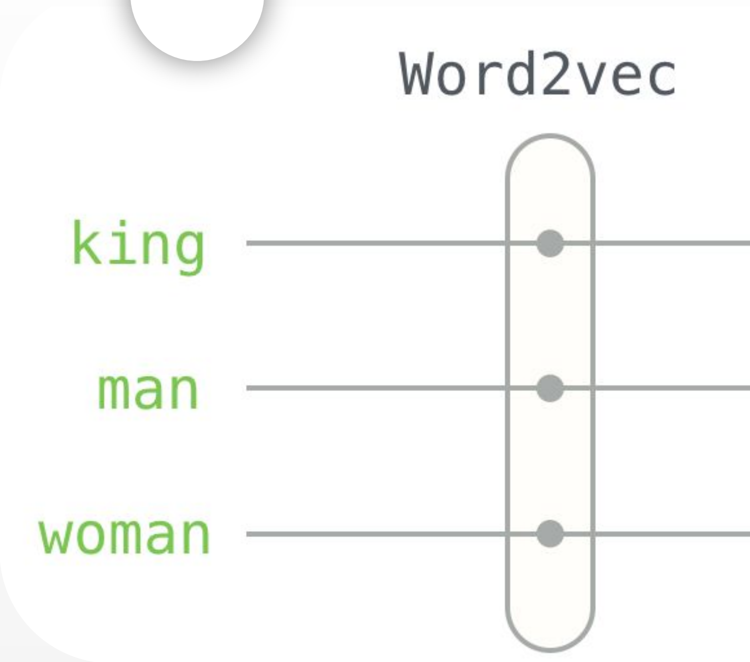
02 Stop Words Removal

03 Converting to Lowercase

04 ETC.

Word2Vec

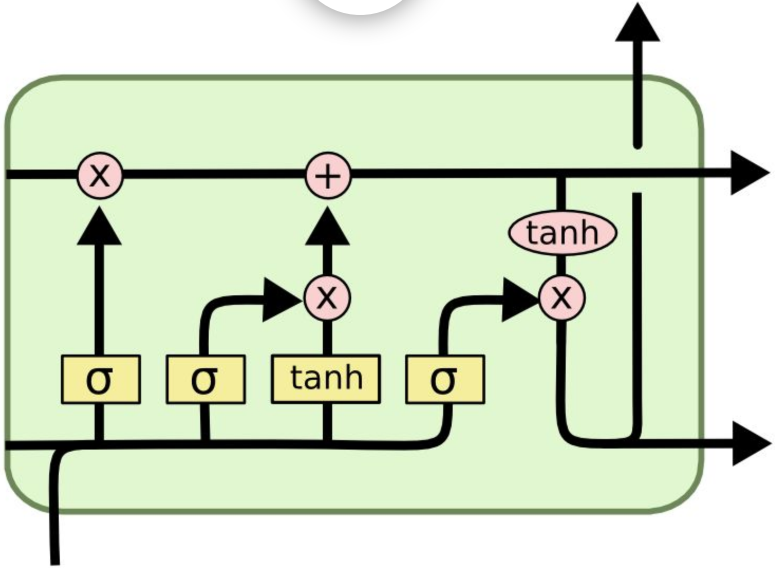
An introduction to the Word2Vec model



Word2vec is a technique for natural language processing (NLP) published in 2013. The word2vec algorithm uses a neural network model to learn word associations from a large corpus of text. Once trained, such a model can detect synonymous words or suggest additional words for a partial sentence.

LSTMs

How LSTMs are used
for Text Classification



There are many classic classification algorithms like Decision trees, RFR, SVM, that can fairly do a good job, then why to use LSTM for classification?
One good reason to use LSTM is that it is effective in memorizing important information.
If we look and other non-neural network classification techniques they are trained on multiple word as separate inputs that are just word having no actual meaning as a sentence, and while predicting the class it will give the output according to statistics and not according to meaning. That means, every single word is classified into one of the categories.
This is not the same in LSTM. In LSTM we can use a multiple word string to find out the class to which it belongs. This is very helpful while working with Natural language processing. If we use appropriate layers of embedding and encoding in LSTM, the model will be able to find out the actual meaning in input string and will give the most accurate output class. The following code will elaborate the idea on how text classification is done using LSTM.

Code Snippets

Data Preparation:

```
In [2]: df.columns

Out[2]: Index(['sentiments', 'cleaned_review', 'cleaned_review_length',
               'review_score'],
              dtype='object')

In [3]: # Dropping the Index colums as it is not relevant
        df = df.drop(['cleaned_review_length'], axis=1, inplace = False)
        df = df.drop(['review_score'], axis=1, inplace = False)
        df = df.reset_index(drop=True)
        df

Out[3]:
```

	sentiments	cleaned_review
0	positive	i wish would have gotten one earlier love it a...
1	neutral	i ve learned this lesson again open the packag...
2	neutral	it is so slow and lags find better option
3	neutral	roller ball stopped working within months of m...
4	neutral	i like the color and size but it few days out ...
...	...	...
17335	positive	i love this speaker and love can take it anywh...
17336	positive	i use it in my house easy to connect and loud ...
17337	positive	the bass is good and the battery is amazing mu...
17338	positive	love it
17339	neutral	mono speaker

Pre-Processing:

```
In [5]: def preprocess_text(text: str, remove_stopwords: bool) -> str:
            """Function that cleans the input text by going to:
            - remove links
            - remove special characters
            - remove numbers
            - remove stopwords
            - convert to lowercase
            - remove excessive white spaces
            Arguments:
                text (str): text to clean
                remove_stopwords (bool): whether to remove stopwords
            Returns:
                str: cleaned text
            """
            # remove links
            text = re.sub(r"http\S+", "", text)
            # remove numbers and special characters
            text = re.sub("[^A-Za-z]+", " ", text)
            # remove stopwords
            if remove_stopwords:
                # 1. create tokens
                tokens = nltk.word_tokenize(text)
                # 2. check if it's a stopword
                tokens = [w.lower().strip() for w in tokens if not w.lower() in stopwords.words("english")]
                # return a list of cleaned tokens
                return tokens

In [6]: df['cleaned'] = df['cleaned_review'].apply(
            lambda myx: preprocess_text(str(myx), remove_stopwords=True)
          )
```

Word2Vec:

```
In [11]: word2vec_model = Word2Vec(sentences=texts, min_count=2, alpha=0.025, vector_size=315,
                                    window=5, epochs=155,  sg=0)
```

```
In [12]: Max_nm_words=50000
         embedding_dim=100
```

```
In [13]: tokenizer = Tokenizer(num_words=50000)
         tokenizer.fit_on_texts(texts)
         sequences = tokenizer.texts_to_sequences(texts)
```

```
In [14]: word_index= tokenizer.word_index
         print('found %s unique tokens' % len(word_index))

         found 9447 unique tokens
```

```
In [15]: max_length = max([len(s) for s in sequences])
         padded_sequences = pad_sequences(sequences, maxlen=max_length)
         padded_sequences.shape
```

Splitting the data:

```
In [17]: y=pd.get_dummies(df['sentiments']).values
         print('shape of label tensor:',y.shape)

         shape of label tensor: (17340, 3)

In [18]: y

Out[18]: array([[0, 0, 1],
                [0, 1, 0],
                [0, 1, 0],
                ...,
                [0, 0, 1],
                [0, 0, 1],
                [0, 1, 0]], dtype=uint8)

In [19]: X_train, X_test, y_train, y_test = train_test_split(padded_sequences, y, test_size=0.25, random_state=42)
```

Model Architecture:

```
n [25]: model = Sequential()
        model.add(Embedding(Max_nm_words, embedding_dim, input_length=padded_sequences.shape[1]))
        model.add(SpatialDropout1D(0.2))
        model.add(LSTM(100,dropout=0.2, recurrent_dropout=0.2))
        model.add(Dense(units=3, activation='softmax'))
```

```
n [26]: model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
        print(model.summary())
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 313, 100)          5000000

 spatial_dropout1d (SpatialD  (None, 313, 100)         0
 ropout1D)

 lstm (LSTM)                 (None, 100)               80400

 dense (Dense)               (None, 3)                 303
```

Model Architecture Cont.

```
In [27]: from tensorflow.keras.callbacks import EarlyStopping

In [28]: history =model.fit(X_train, y_train, validation_data=(X_test,y_test),epochs=10, batch_size=128,callbacks=[EarlyStopping(monitor='

Epoch 1/10
102/102 [==============================] - 148s 1s/step - loss: 0.7147 - accuracy: 0.6815 - val_loss: 0.4944 - val_accuracy: 0.
8062
Epoch 2/10
102/102 [==============================] - 158s 2s/step - loss: 0.4098 - accuracy: 0.8434 - val_loss: 0.3991 - val_accuracy: 0.
8512
Epoch 3/10
102/102 [==============================] - 167s 2s/step - loss: 0.2968 - accuracy: 0.8960 - val_loss: 0.3780 - val_accuracy: 0.
8669
Epoch 4/10
102/102 [==============================] - 167s 2s/step - loss: 0.2257 - accuracy: 0.9212 - val_loss: 0.3936 - val_accuracy: 0.
8685
```

Conclusion

In conclusion, the LSTM model could accurately classify customer reviews' sentiment into one of three categories (positive, negative, or neutral). The model achieved an accuracy of 86.85% on the test data, which was a good result.

Results

Class	Precision	Recall	F1-Score	Support
Positive	0.71	0.72	0.72	398
Neutral	0.85	0.81	0.83	1579
Negative	0.91	0.93	0.92	2358
Overall (Accuracy)	0.87	0.87	0.87	4335

Thanks!

Our Team:
Mahmoud Hossam - 222125
Haidy Haitham - 212539

For Questions:

mahmoud.hossam2@msa.edu.eg
haidy.haitham@msa.edu.eg