



Sentiment Analysis

Capstone Project

V R S S Suryavamsi Tenneti

23rd November, 2018.

Overview

Opinion Mining or Sentiment Analysis, refers to the use of natural language processing, text analysis, computational linguistics, and biometrics to systematically identify, extract, quantify, and study affective states and subjective information. Sentiment analysis is widely applied to voice of the customer materials such as reviews and survey responses, online and social media, and healthcare materials for applications that range from marketing to customer service to clinical medicine.

Each industry nowadays are very much interested in getting to know about the opinion of customers on their products and by that way want to improve their products. But it is not that easy to derive the sentiment of a statement. Many challenges like sarcasm make the sentiment detection very difficult.

This project is an application of Natural Language Processing, which deals with the extraction of useful information and patterns from a piece of text.

Problem statement

Given a sentence, the model identifies whether the sentiment of the sentence is either positive or negative based on the information learned using Supervised Learning technique.

My approach

In this project, we try to train a model using a set of tweets labelled either as positive or negative. Based on the training done, we try to use the trained model to determine the sentiment of a piece of text.

This is an implementation of “Supervised Learning” technique. Also, in this project I have implemented different word embedding algorithms to find out which type of word embeddings help in training the model better. As word embeddings play an important role in representing relation between words thereby representing relation between sentences, it will help us to consider all the sentences with more similarity into the same category. The word embedding algorithms used in this project are:

1. Word2Vec
2. fastText
3. Universal Sentence Encoder

And initially some preprocessing is done to the tweets to decrease the amount of noise in the dataset. Some types of noise in the dataset are:

1. Emoji
2. Hashtags
3. @ Mention
4. HTML encodings (Eg: &, ")

Previous Work (Literature Review)

Lot of research work has been done on this topic as this poses certain challenges like sarcasm which highly misleads the conventional models. Below are few works carried out which inspired me.

1. Zhang, Lei, Shuai Wang, and Bing Liu. "Deep Learning for Sentiment Analysis: A Survey." arXiv preprint arXiv:1801.07883 (2018).

2. Sosa, Pedro M. "Twitter Sentiment Analysis Using Combined LSTM-CNN Models". Konukoii.Com, 2018,

<http://konukoii.com/blog/2018/02/19/twitter-sentimentanalysis-using-combined-lstm-cnn-models/>. Accessed 22 Feb 2018.

Metrics

The dataset which I have chosen contains equal number of positive and negative tweets, so as the dataset is completely balanced, accuracy and validation loss are the best metrics to measure the performance of the model. More the accuracy of the model, more is the model's efficiency to predict the sentiment of the text (positive or negative).

Why accuracy is sufficient for this model? This is the most common question. For this model, accuracy is best because, we know that accuracy is the ratio of the total number of correctly classified items to the total number of items. In case of an unbalanced dataset, if the major class is correctly identified and minor class is wrongly classified, it doesn't show great effect on accuracy but that doesn't mean it is a good model as the minor class is mostly wrongly classified. But in this case as both positive and negative class items are equal in number, accuracy is the best metric.

Milestones

Initially, I used Word2Vec algorithm for generating word embeddings to input to the CNN-LSTM neural network but then came to know of different word embedding generator algorithms and wanted to try those to find out the performance changes.

Also, the CNN-LSTM model was not the initial model. First, I tried to use a simple SVM to identify the sentiment but later on found that there are certain patterns in the text which can't be identified using a simple Machine Learning algorithm so, I shifted to deep learning algorithms. In deep learning algorithms also, the first model was the Recurrent Neural Networks, but there is a problem with the way they try to identify the patterns and store context information which is referred to as "Bottle Neck" problem. So, shifted to a memory based network LSTM (Long Short Term Memory) network.

Also, after performing some image processing tasks, I came to know that Convolutional Neural Networks have the ability to learn patterns. As, text is a 1D data, thought of using a 1D Convolutional filter. In this way, I came to this stage where I thought to use a CNN-LSTM model along with different word embedding algorithms to predict the sentiment of a text.

Data Exploration

The dataset used for this project is the "Sentiment140" dataset. This dataset consists of tweets which are collected and labelled as 0/2/4, where

0 → Negative

2 → Neutral

4 → Positive

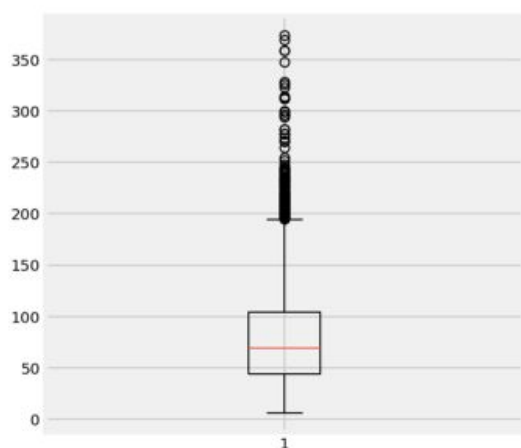
But the dataset does not have any tweets labelled as Neutral, and the dataset is completely balanced as it consists a total of 1.6 million tweets where 0.8 million are labelled positive and the remaining 0.8 million labelled as negative.

The dataset description is as follows:

1. Polarity of tweet → 0/2/4 (0 → negative, 2 → neutral, 4 → positive)
2. Id → User id of the tweet
3. Date → date of the tweet
4. Query → query used to retrieve that particular tweet (If nothing then its value is NO_QUERY)
5. Tweet → Text content of the tweet

We need only the Polarity and the tweet and rest all are not important for our sentiment analysis. So, I decided to drop the remaining columns.

Also, the below is a boxplot of the tweets with respect to the length of the tweets before data preprocessing.



Though the length of tweet has been changed from 140 to 280 recently, this dataset was from a time where the length of the tweet was only 140. But we can see that some tweets have length more than 200 also. So, we need to process the data.

The dataset is collected by taking advantage of some properties of a tweet like:

Username: Users often include Twitter usernames in their tweets in order to direct their messages.

Usage of links : Users very often include links in their tweets. An equivalence class is used for all URLs. That is, we convert a URL like “<http://tinyurl.com/cvvg9a>” to the token “URL.”

Repeated letters : Tweets contain very casual language. For example, if you search “hungry” with an arbitrary number of u’s in the middle (e.g. huuuungry, huuuuuuungry, huuuuuuuuungry) on Twitter, there will most likely be a nonempty result set. We use preprocessing so that any letter occurring more than two times in a row is replaced with two occurrences. In the samples above, these words would be converted into the token hungry.

What is a word embedding?

A very basic definition of a word embedding is a real number, vector representation of a word. Typically, these days, words with similar meaning will have vector representations that are close together in the embedding space (though this hasn't always been the case).

When constructing a word embedding space, typically the goal is to capture some sort of relationship in that space, be it meaning, morphology, context, or some other kind of relationship.

By encoding word embeddings in a densely populated space, we can represent words numerically in a way that captures them in vectors that have tens or hundreds of dimensions instead of millions (like one-hot encoded vectors).

A lot of word embeddings are created based on the notion introduced by Zellig Harris' "distributional hypothesis" which boils down to a simple idea that words that are used close to one another typically have the same meaning.

Words aren't things that computers naturally understand. By encoding them in a numeric form, we can apply mathematical rules and do matrix operations to them. This makes them amazing in the world of machine learning, especially.

Algorithms and techniques

1. Word2Vec

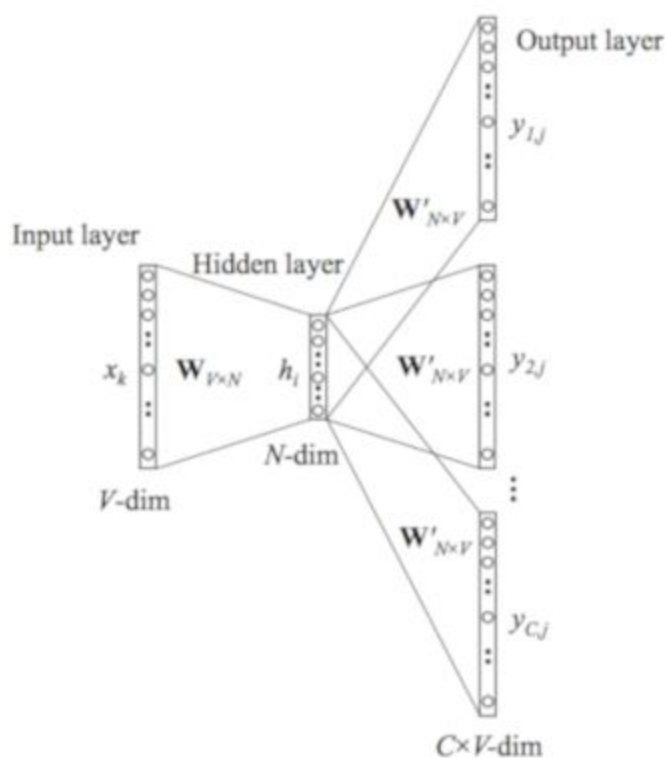
Word2vec is a group of related models that are used to produce word embeddings. These models are shallow, two-layer neural networks that are trained to reconstruct linguistic contexts of words. Word2vec takes as its input a large corpus of text and produces a vector space, typically of several hundred dimensions, with each unique word in the corpus being assigned a corresponding vector in the space. Word vectors are positioned in the vector space such that words that share common

contexts in the corpus are located in close proximity to one another in the space.
[Source: Wikipedia]

Docs in Gensim: [models.word2vec](#)

Word2Vec has 2 important models inside: Skip-Grams and Continuous Bag-of-Words(CBOW)

Skip-Grams:

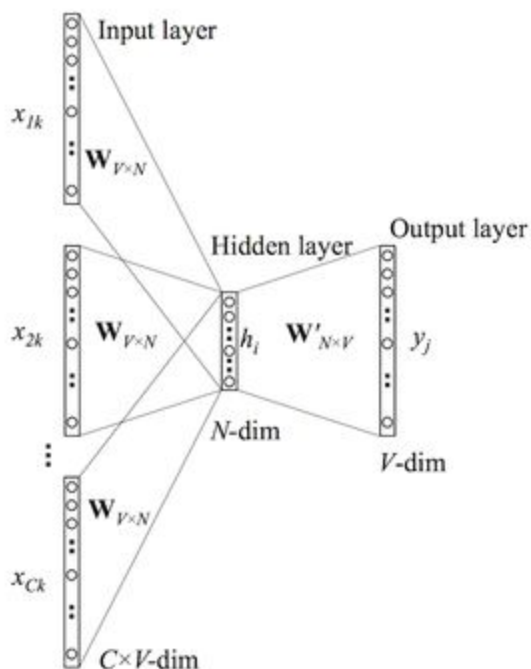


In Skip-Gram model, we take a centre word and a window of context words or neighbors within the context window and we try to predict context words for each centre word. The model generates a probability distribution i.e., probability of a word appearing in context given centre word and the task here is to choose the vector representation to maximize the probability.

Source Text	Training Samples						
<table><tr><td>The</td><td>quick</td><td>brown</td></tr></table> fox jumps over the lazy dog. ➡	The	quick	brown	(the, quick) (the, brown)			
The	quick	brown					
<table><tr><td>The</td><td>quick</td><td>brown</td><td>fox</td></tr></table> jumps over the lazy dog. ➡	The	quick	brown	fox	(quick, the) (quick, brown) (quick, fox)		
The	quick	brown	fox				
<table><tr><td>The</td><td>quick</td><td>brown</td><td>fox</td><td>jumps</td></tr></table> over the lazy dog. ➡	The	quick	brown	fox	jumps	(brown, the) (brown, quick) (brown, fox) (brown, jumps)	
The	quick	brown	fox	jumps			
<table><tr><td>The</td><td>quick</td><td>brown</td><td>fox</td><td>jumps</td><td>over</td></tr></table> the lazy dog. ➡	The	quick	brown	fox	jumps	over	(fox, quick) (fox, brown) (fox, jumps) (fox, over)
The	quick	brown	fox	jumps	over		

Continuous Bag-of-Words (CBOW) :

CBOW is opposite of Skip-Grams. We attempt to predict the centre word from the given context i.e., we try to predict the centre word by summing vectors of surrounding words.



2. fastText

fastText is a library for learning of word embeddings and text classification created by Facebook's AI Research (FAIR) lab. The model is an unsupervised learning algorithm for obtaining vector representations for words. Facebook makes available pretrained models for 294 languages. fastText uses Neural network for word embedding.

Docs on Gensim: [models.fastText](https://www.gensim.org/docs/models/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 (sub-words). For instance, the tri-grams for the word apple is app, ppl, and ple (ignoring the starting and ending of boundaries of words). The word embedding vector for apple will be the sum of all these n-grams. After training the Neural Network, we will have word embeddings for all the n-grams given the training dataset. Rare words can now be properly represented since it is highly likely that some of their n-grams also appears in other words. I will show you how to use FastText with Gensim in the following section.

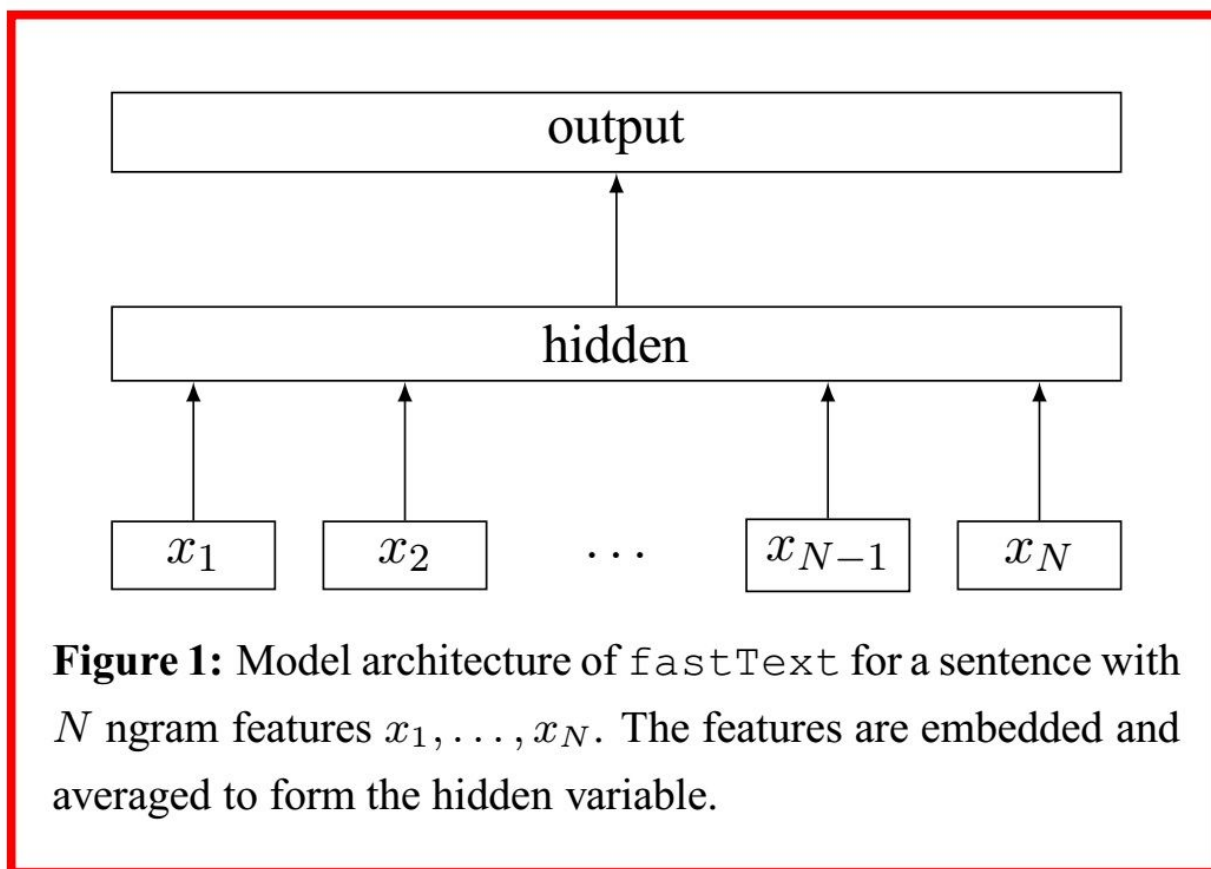



Figure 1: Model architecture of `fastText` for a sentence with N ngram features x_1, \dots, x_N . The features are embedded and averaged to form the hidden variable.

3. Universal Sentence Encoder

Released in 2018, The Universal Sentence Encoder encodes text into high dimensional vectors that can be used for text classification, semantic similarity, clustering and other natural language tasks.

The model is trained and optimized for greater-than-word length text, such as sentences, phrases or short paragraphs. It is trained on a variety of data sources and a variety of tasks with the aim of dynamically accommodating a wide variety of natural language understanding tasks. The input is variable length English text and the output is a 512 dimensional vector. We apply this model to the STS benchmark for semantic similarity, and the results can be seen in the example notebook made available. The universal-sentence-encoder model is trained with a deep averaging network (DAN) encoder.

In my experience with all the three models, I observed that `word2vec` takes a lot more time to generate Vectors from all the three models. `FastText` and `Universal Sentence Encoder`



take relatively same time. For word2vec and fastText, pre-processing of data is required which takes some amount of time.

When it comes to training, fastText takes a lot less time than Universal Sentence Encoder and as same time as word2vec model.

Benchmark model

Sentiment140, a standalone tool on Tech test bed achieved 67.82% accuracy and on Telco test bed achieved 71.79% accuracy on the Sentiment140 dataset and the EC2 backend implementation model achieved an accuracy of 83%. My model performance is compared with this tool's performance to check how far I could reach and try to fine tune my model.

Methodology

Data PreProcessing

The main issue with tweet text is that it contains lot many non-ASCII characters along with hashtags, mentions etc. So, we need to process those to build a good model.

Step-1:

I used 'twitter-preprocessor' to handle basic preprocessing which includes removal of URLs and @ mentions.

Step-2:

Even after step-1, there are some HTML encodings which are not decoded like & and ". So, to fix these issues, used 'BeautifulSoup' package.


Step-3:

After step-2, we can see meaningful text. Now, there are several words in the tweet which may state the same or few words which have the same root words. So, if all words are represented in their root word form it will improve the training of the model. So, used lemmatization technique to achieve this.

After all the above preprocessing steps, the dataset is sufficiently shuffled so that there is randomness in the dataset and hence will help to build a good model. I applied 3 different word embedding algorithms to achieve the results and compare their performance on applied their output as input to CNN-LSTM network.

Conv1D -> Conv1D -> Conv1D -> Max Pooling1D -> Bidirectional LSTM -> Dense -> Dropout -> Dense -> Dropout -> Dense -> Dropout -> Output

This is the architecture of the network. The reason to use CNNs is that though they are widely used in image-related tasks, they are able to detect patterns in the data. So, a 1D convolutional filter would help to identify the patterns which help us to distinguish the



negative tweets from the positive ones more easily. Long-Term Short Term Memory (LSTMs) are a type of network that has a memory that "remembers" previous data from the input and makes decisions based on that knowledge. These networks are more directly suited for written data inputs, since each word in a sentence has meaning based on the surrounding words (previous and upcoming words). In our particular case, it is possible that an LSTM could allow us to capture changing sentiment in a tweet.

All the algorithms are implemented using Gensim package or the Tensor_Hub pretrained models. More implementation details can be seen in the IPynb notebooks.

The Universal Sentence Encoder model's word embeddings are better than the rest two as they helped to achieve an accuracy of approximately 77%.

Model Evaluation and Validation

The dataset is split into train and validation (test) sets as 90% and 10% respectively. The model is trained on the train set and the validation set is used to evaluate the accuracy and loss of the model. Here accuracy is a good metric as the dataset is balanced.

The highest accuracy achieved is approximately 77% by the model whose word embeddings are calculated using "Universal Sentence Encoder".

Comparison with the benchmark model

The best accuracy achieved by the models I implemented is only 77% whereas the benchmark model stated above was able to achieve 83% accuracy. Some factors that might affect are number of epochs and batch size.

Also, the backend GPU I used was comparatively less powerful and also if the model was trained for some more number of times, the model could be able to learn some more complex patterns.

Robustness of the model

My model was able to classify even a random unseen tweet as positive or negative to a maximum extent. I have tried with different tweets from recent times to verify if my model is working fine. Though I need to preprocess the tweet, it performs fairly well with unseen data.

Conclusion

This is one of the application of Natural Language Processing. On later reading different research papers, I came to know that there are several factors affecting the accuracy of the model like sarcasm. Sarcasm is generally not easily identified by machines. So, we need to use some other algorithms to efficiently find out the sentiment out of a sarcastic sentence. Also, this dataset is not too much noisy, and generally in real world we will be facing a more noisy datasets and we need more preprocessing techniques to be able to work with real world datasets.

Project summary

Initially, the tweets are made free from emoji, @ mentions and also removed '#' symbol from hashtags.

Afterwards, we reduced the words to their root words so that words which are in different sentences but express the same, makes the training of the model more efficient.

Then applied Word2Vec, fastText and Universal Sentence Encoder to compute the word embeddings which are then input to the CNN-LSTM neural network and training is done on the train set and validation is done on the validation set. The dataset splits were as follows:

Train set → 90% of the data

Validation set → 10% of the data

Improvements in implementation

The negation statements are not handled properly. The negative or positiveness is based on the subject we consider. So, for that we need to do work on some more semantics. Due to this there may be a considerable increase in the accuracy of the model.

Also, the emoticons are stripped off from the dataset, but sometimes emoticons considerably help in identifying the sentiment. So, we need to do some type of mapping.

Another thing is the slang words. We did not do much processing of slang words and they are so much common in social networking sites like Twitter, so handling them must really boost the performance.