(http://play.google.com/store/apps/details?id=com.analyticsvidhya.android)

(http://play.google.com/store/apps/details?id=com.analyticsvidhya.android)

LOGIN / REGISTER (HTTPS://ID.ANALYTICSVIDHYA.COM/ACCOUNTS/LOGIN/?

NEXT=HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/2018/02/THE-DIFFERENT-METHODS-DEAL-TEXT-DATA-PREDICTIVE-

PYTHON/) LOGIN / REGISTER (HTTPS://ID.ANALYTICSVIDHYA.COM/ACCOUNTS/LOGIN/?

NEXT=HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/2018/02/THE-DIFFERENT-METHODS-DEAL-TEXT-DATA-PREDICTIVE-

PYTHON/)





# **Applied Machine Learning**



#### 16 - 17TH FEB, 2019 | DELHI NCR

(https://trainings.analyticsvidhya.com/courses/course-

v1:AnalyticsVidhya+AMLW101+AMLW101 Jan2019/about?utm source=AVtopBanner)

MACHINE LEARNING (HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/CATEGORY/MACHINE-LEARNING/).

NLP (HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/CATEGORY/NLP/)

PYTHON (HTTPS://WWW ANALYTICSVIDHYA COM/BLOG/CATEGORY/PYTHON-2/

# Ultimate guide to deal with Text Data (using Python) – for Data Scientists & Engineers

#### Introduction

One of the biggest breakthroughs required for achieving any level of artificial intelligence is to have machines which can process text data. Thankfully, the amount of text data being generated in this universe has exploded exponentially in the last few years.

It has become imperative for an organization to have a structure in place to mine actionable insights from the text being generated. From social media analytics to risk management and cybercrime protection, dealing with text data has never been more important.

In this article we will discuss different feature extraction methods, starting with some basic techniques which will lead into advanced Natural Language Processing techniques. We will also learn about preprocessing of the text data in order to extract better features from clean data.

In addition, if you want to dive deeper, we also have a <u>video course</u> (<a href="https://trainings.analyticsvidhya.com/courses/course-v1:AnalyticsVidhya+NLP101+2018\_T1/about?">https://trainings.analyticsvidhya.com/courses/course-v1:AnalyticsVidhya+NLP101+2018\_T1/about?</a>
<a href="https://trainings.analyticsvidhya.com/courses/course-v1:AnalyticsVidhya+NLP101+2018\_T1/about?">https://trainings.analyticsvidhya.com/courses/course-v1:AnalyticsVidhya+NLP101+2018\_T1/about?</a>
<a href="https://trainings.analyticsvidhya.com/courses/course-v1:AnalyticsVidhya+NLP101+2018\_T1/about?">https://trainings.analyticsvidhya.com/courses/course-v1:AnalyticsVidhya+NLP101+2018\_T1/about?</a>
<a href="https://trainings.analyticsvidhya.com/courses/course-v1:AnalyticsVidhya+NLP101+2018\_T1/about?">https://trainings.analyticsvidhya.com/courses/course-v1:AnalyticsVidhya+NLP101+2018\_T1/about?</a>
<a href="https://trainings.analyticsvidhya.com/courses/course-v1:AnalyticsVidhya+NLP101+2018\_T1/about?">https://trainings.analyticsvidhya.com/courses/course-v1:AnalyticsVidhya+NLP101+2018\_T1/about?</a>
<a href="https://trainings.analyticsvidhya.com/courses/course-v1:AnalyticsVidhya+NLP101+2018\_T1/about?">https://trainings.analyticsvidhya.com/courses/course-v1:AnalyticsVidhya+NLP101+2018\_T1/about?</a>
<a href="https://trainings.analyticsvidhya.com/course-v1:AnalyticsVidhya+NLP101+2018\_T1/about?">https://trainings.analyticsvidhya.com/course-v1:AnalyticsVidhya+NLP101+2018\_T1/about?</a>
<a href="https://trainings.analyticsvidhya.com/course-v1:AnalyticsVidhya+NLP101+2018\_T1/about?">https://trainings.analyticsvidhya.com/course-v1:AnalyticsVidhya+NLP101+2018\_T1/about?</a>
<a href="https://trainings.analyticsvidhya.com/course-v1:Analyticsvidhya.com/course-v1:Analyticsvidhya.com/course-v1:Analyticsvidhya.com/course-v1:Analyticsvidhya.com/course-v1:Analyticsvidhya.com/course-v1:Analyticsvidhya.com/course-v1:Analyticsvidhya.com/course-v1:Analyticsvidhya.com/course-v1:Analyticsvidhya.com/course-v1:Analyticsvidhya.com/course-v1:Analyticsvidhya.com/course-v1:Ana

By the end of this article, you will be able to perform text operations by yourself. Let's get started!

#### Table of Contents:

- 1. Basic feature extraction using text data
  - Number of words
  - Number of characters
  - Average word length
  - Number of stopwords
  - Number of special characters
  - Number of numerics
  - Number of uppercase words

#### 2. Basic Text Pre-processing of text data

- Lower casing
- Punctuation removal
- Stopwords removal
- Frequent words removal
- Rare words removal
- Spelling correction
- Tokenization
- Stemming
- Lemmatization

#### 3. Advance Text Processing

- N-grams
- Term Frequency
- Inverse Document Frequency
- Term Frequency-Inverse Document Frequency (TF-IDF)
- Bag of Words
- Sentiment Analysis
- Word Embedding

#### 1. Basic Feature Extraction

We can use text data to extract a number of features even if we don't have sufficient knowledge of Natural Language Processing. So let's discuss some of them in this section.

Before starting, let's quickly read the training file from the dataset in order to perform different tasks on it. In the entire article, we will use the twitter sentiment <u>dataset</u> (<a href="https://datahack.analyticsvidhya.com/contest/practice-problem-twitter-sentiment-analysis/">https://datahack.analyticsvidhya.com/contest/practice-problem-twitter-sentiment-analysis/</a>) from the

(https://datahack.analyticsvidhya.com/contest/practice-problem-twitter-sentiment-analysis/) from the datahack platform.

```
train = pd.read_csv('train_E6oV3lV.csv')
```

Note that here we are only working with textual data, but we can also use the below methods when numerical features are also present along with the text.

## 1.1 Number of Words

One of the most basic features we can extract is the number of words in each tweet. The basic intuition behind this is that generally, the negative sentiments contain a lesser amount of words than the positive ones.

To do this, we simply use the split function in python:

```
train['word_count'] = train['tweet'].apply(lambda x: len(str(x).split(" ")))
train[['tweet','word_count']].head()
```

#### 1.2 Number of characters

This feature is also based on the previous feature intuition. Here, we calculate the number of characters in each tweet. This is done by calculating the length of the tweet.

```
train['char_count'] = train['tweet'].str.len() ## this also includes spaces
train[['tweet','char_count']].head()
```

Note that the calculation will also include the number of spaces, which you can remove, if required.

# 1.3 Average Word Length

We will also extract another feature which will calculate the average word length of each tweet. This can also potentially help us in improving our model.

Here, we simply take the sum of the length of all the words and divide it by the total length of the tweet:

```
def avg_word(sentence):
    words = sentence.split()
    return (sum(len(word) for word in words)/len(words))

train['avg_word'] = train['tweet'].apply(lambda x: avg_word(x))

train[['tweet','avg_word']].head()
```

## 1.4 Number of stopwords

Generally, while solving an NLP problem, the first thing we do is to remove the stopwords. But sometimes calculating the number of stopwords can also give us some extra information which we might have been losing before.

Here, we have imported stopwords from *NLTK*, which is a basic NLP library in python.

```
from nltk.corpus import stopwords
stop = stopwords.words('english')

train['stopwords'] = train['tweet'].apply(lambda x: len([x for x in x.split() if x in stop]))

train[['tweet','stopwords']].head()
```

## 1.5 Number of special characters

One more interesting feature which we can extract from a tweet is calculating the number of hashtags or mentions present in it. This also helps in extracting extra information from our text data.

Here, we make use of the 'starts with' function because hashtags (or mentions) always appear at the beginning of a word.

```
train['hastags'] = train['tweet'].apply(lambda x: len([x for x in x.split() if x.star
tswith('#')]))
train[['tweet','hastags']].head()
```

#### 1.6 Number of numerics

Just like we calculated the number of words, we can also calculate the number of numerics which are present in the tweets. It does not have a lot of use in our example, but this is still a useful feature that should be run while doing similar exercises. For example,

```
train['numerics'] = train['tweet'].apply(lambda x: len([x for x in x.split() if x.isd
    igit()]))
    train[['tweet', 'numerics']].head()
```

## 1.7 Number of Uppercase words

Anger or rage is quite often expressed by writing in UPPERCASE words which makes this a necessary operation to identify those words.

```
train['upper'] = train['tweet'].apply(lambda x: len([x for x in x.split() if x.isuppe
r()]))
train[['tweet','upper']].head()
```

## 2. Basic Pre-processing

So far, we have learned how to extract basic features from text data. Before diving into text and feature extraction, our first step should be cleaning the data in order to obtain better features. We will achieve this by doing some of the basic pre-processing steps on our training data.

So, let's get into it.

#### 2.1 Lower case

The first pre-processing step which we will do is transform our tweets into lower case. This avoids having multiple copies of the same words. For example, while calculating the word count, 'Analytics' and 'analytics' will be taken as different words.

```
train['tweet'] = train['tweet'].apply(lambda x: " ".join(x.lower() for x in x.split()
))
train['tweet'].head()
```

#### 2.2 Removing Punctuation

The next step is to remove punctuation, as it doesn't add any extra information while treating text data. Therefore removing all instances of it will help us reduce the size of the training data.

```
train['tweet'] = train['tweet'].str.replace('[^\w\s]','')
train['tweet'].head()
```

As you can see in the above output, all the punctuation, including '#' and '@', has been removed from the training data.

## 2.3 Removal of Stop Words

As we discussed earlier, stop words (or commonly occurring words) should be removed from the text data. For this purpose, we can either create a list of stopwords ourselves or we can use predefined libraries.

```
from nltk.corpus import stopwords
stop = stopwords.words('english')
train['tweet'] = train['tweet'].apply(lambda x: " ".join(x for x in x.split() if x no
t in stop))
train['tweet'].head()
```

#### 2.4 Common word removal

Previously, we just removed commonly occurring words in a general sense. We can also remove commonly occurring words from our text data First, let's check the 10 most frequently occurring words in our text data then take call to remove or retain.

Now, let's remove these words as their presence will not of any use in classification of our text data.

```
freq = list(freq.index)
train['tweet'] = train['tweet'].apply(lambda x: " ".join(x for x in x.split() if x no
t in freq))
train['tweet'].head()
```

#### 2.5 Rare words removal

Similarly, just as we removed the most common words, this time let's remove rarely occurring words from the text. Because they're so rare, the association between them and other words is dominated by noise. You can replace rare words with a more general form and then this will have higher counts

```
freq = list(freq.index)
train['tweet'] = train['tweet'].apply(lambda x: " ".join(x for x in x.split() if x no
t in freq))
train['tweet'].head()
```

All these pre-processing steps are essential and help us in reducing our vocabulary clutter so that the features produced in the end are more effective.

## 2.6 Spelling correction

We've all seen tweets with a plethora of spelling mistakes. Our timelines are often filled with hastly sent tweets that are barely legible at times.

In that regard, spelling correction is a useful pre-processing step because this also will help us in reducing multiple copies of words. For example, "Analytics" and "analytics" will be treated as different words even if they are used in the same sense.

To achieve this we will use the *textblob* library. If you are not familiar with it, you can check my previous <u>article (https://www.analyticsvidhya.com/blog/2018/02/natural-language-processing-for-beginners-using-textblob/)</u> on 'NLP for beginners using *textblob*'.

```
from textblob import TextBlob
train['tweet'][:5].apply(lambda x: str(TextBlob(x).correct()))
```

Note that it will actually take a lot of time to make these corrections. Therefore, just for the purposes of learning, I have shown this technique by applying it on only the first 5 rows. Moreover, we cannot always expect it to be accurate so some care should be taken before applying it.

We should also keep in mind that words are often used in their abbreviated form. For instance, 'your' is used as 'ur'. We should treat this before the spelling correction step, otherwise these words might be transformed into any other word like the one shown below:

#### 2.7 Tokenization

Tokenization refers to dividing the text into a sequence of words or sentences. In our example, we have used the *textblob* library to first transform our tweets into a blob and then converted them into a series of words.

```
TextBlob(train['tweet'][1]).words
> WordList(['thanks', 'lyft', 'credit', 'cant', 'use', 'cause', 'dont', 'offer', 'whe
elchair', 'vans', 'pdx', 'disapointed', 'getthanked'])
```

## 2.8 Stemming

Stemming refers to the removal of suffices, like "ing", "ly", "s", etc. by a simple rule-based approach. For this purpose, we will use *PorterStemmer* from the NLTK library.

```
from nltk.stem import PorterStemmer

st = PorterStemmer()

train['tweet'][:5].apply(lambda x: " ".join([st.stem(word) for word in x.split()]))

0 father dysfunct selfish drag kid dysfunct run

1 thank lyft credit cant use caus dont offer whe...

2 bihday majesti

3 model take urð ðððð ððð

4 factsguid societi motiv

Name: tweet, dtype: object
```

In the above output, dysfunctional has been transformed into dysfunct, among other changes.

#### 2.9 Lemmatization

Lemmatization is a more effective option than stemming because it converts the word into its root word, rather than just stripping the suffices. It makes use of the vocabulary and does a morphological analysis to obtain the root word. Therefore, we usually prefer using lemmatization over stemming.

```
from textblob import Word
train['tweet'] = train['tweet'].apply(lambda x: " ".join([Word(word).lemmatize() for
word in x.split()]))
train['tweet'].head()
```

## 3. Advance Text Processing

Up to this point, we have done all the basic pre-processing steps in order to clean our data. Now, we can finally move on to extracting features using NLP techniques.

## 3.1 N-grams

N-grams are the combination of multiple words used together. Ngrams with N=1 are called unigrams. Similarly, bigrams (N=2), trigrams (N=3) and so on can also be used.

Unigrams do not usually contain as much information as compared to bigrams and trigrams. The basic principle behind n-grams is that they capture the language structure, like what letter or word is likely to follow the given one. The longer the n-gram (the higher the *n*), the more context you have to work with. Optimum length really depends on the application – if your n-grams are too short, you may fail to capture important differences. On the other hand, if they are too long, you may fail to capture the "general knowledge" and only stick to particular cases.

So, let's quickly extract bigrams from our tweets using the ngrams function of the textblob library.

```
TextBlob(train['tweet'][0]).ngrams(2)
WordList(['a', 'father']),
WordList(['father', 'is']),
WordList(['is', 'dysfunctional']),
WordList(['dysfunctional', 'and']),
WordList(['is', 'so']),
WordList(['so', 'selfish']),
WordList(['selfish', 'he']),
WordList(['his', 'dysfunction']),
WordList(['dysfunction', 'run'])]
```

## 3.2 Term frequency

Term frequency is simply the ratio of the count of a word present in a sentence, to the length of the sentence.

Therefore, we can generalize term frequency as:

TF = (Number of times term T appears in the particular row) / (number of terms in that row)

To understand more about Term Frequency, have a look at <a href="mailto:this.article.">this article.</a>
<a href="mailto:(https://www.analyticsvidhya.com/blog/2015/04/information-retrieval-system-explained/">https://www.analyticsvidhya.com/blog/2015/04/information-retrieval-system-explained/</a>

Below, I have tried to show you the term frequency table of a tweet.

```
tf1 = (train['tweet'][1:2]).apply(lambda x: pd.value_counts(x.split(" "))).sum(axis =
0).reset_index()
tf1.columns = ['words','tf']
tf1
```

You can read more about term frequency in this <u>article</u> (https://www.analyticsvidhya.com/blog/2017/06/word-embeddings-count-word2veec/)

## 3.3 Inverse Document Frequency

The intuition behind inverse document frequency (IDF) is that a word is not of much use to us if it's appearing in all the documents.

Therefore, the IDF of each word is the log of the ratio of the total number of rows to the number of rows in which that word is present.

IDF = log(N/n), where, N is the total number of rows and n is the number of rows in which the word was present.

So, let's calculate IDF for the same tweets for which we calculated the term frequency

```
for i,word in enumerate(tf1['words']):
   tf1.loc[i, 'idf'] = np.log(train.shape[0]/(len(train[train['tweet'].str.contains(wo
rd)])))

tf1
```

The more the value of IDF, the more unique is the word.

## 3.4 Term Frequency – Inverse Document Frequency (TF-IDF)

TF-IDF is the multiplication of the TF and IDF which we calculated above.

```
tf1['tfidf'] = tf1['tf'] * tf1['idf']
tf1
```

We can see that the TF-IDF has penalized words like 'don't', 'can't', and 'use' because they are commonly occurring words. However, it has given a high weight to "disappointed" since that will be very useful in determining the sentiment of the tweet.

We don't have to calculate TF and IDF every time beforehand and then multiply it to obtain TF-IDF. Instead, *sklearn* has a separate function to directly obtain it:

```
from sklearn.feature_extraction.text import TfidfVectorizer

tfidf = TfidfVectorizer(max_features=1000, lowercase=True, analyzer='word',
   stop_words= 'english',ngram_range=(1,1))

train_vect = tfidf.fit_transform(train['tweet'])

train_vect

<31962x1000 sparse matrix of type '<class 'numpy.float64'>'
   with 114033 stored elements in Compressed Sparse Row format>
```

We can also perform basic pre-processing steps like lower-casing and removal of stopwords, if we haven't done them earlier.

## 3.5 Bag of Words

Bag of Words (BoW) refers to the representation of text which describes the presence of words within the text data. The intuition behind this is that two similar text fields will contain similar kind of words, and will therefore have a similar bag of words. Further, that from the text alone we can learn something about the meaning of the document.

For implementation, *sklearn* provides a separate function for it as shown below:

```
from sklearn.feature_extraction.text import CountVectorizer
bow = CountVectorizer(max_features=1000, lowercase=True, ngram_range=(1,1),analyzer =
"word")
train_bow = bow.fit_transform(train['tweet'])
train_bow
> <31962x1000 sparse matrix of type '<class 'numpy.int64'>'
    with 128380 stored elements in Compressed Sparse Row format>
```

To gain a better understanding of this, you can refer to this <u>article</u> (https://www.analyticsvidhya.com/blog/2017/06/word-embeddings-count-word2veec/)

## 3.6 Sentiment Analysis

If you recall, our problem was to detect the sentiment of the tweet. So, before applying any ML/DL models (which can have a separate feature detecting the sentiment using the textblob library), let's check the sentiment of the first few tweets.

Above, you can see that it returns a tuple representing polarity and subjectivity of each tweet. Here, we only extract polarity as it indicates the sentiment as value nearer to 1 means a positive sentiment and values nearer to -1 means a negative sentiment. This can also work as a feature for building a machine learning model.

```
train['sentiment'] = train['tweet'].apply(lambda x: TextBlob(x).sentiment[0] )
train[['tweet','sentiment']].head()
```

## 3.7 Word Embeddings

Word Embedding is the representation of text in the form of vectors. The underlying idea here is that similar words will have a minimum distance between their vectors

Word2Vec models require a lot of text, so either we can train it on our training data or we can use the pre-trained word vectors developed by Google, Wiki, etc.

Here, we will use pre-trained word vectors which can be downloaded from the <u>glove</u> (<a href="https://nlp.stanford.edu/projects/glove/">https://nlp.stanford.edu/projects/glove/</a>) website. There are different dimensions (50,100, 200, 300) vectors trained on wiki data. For this example, I have downloaded the 100-dimensional version of the model

You can refer an article <a href="https://www.analyticsvidhya.com/blog/2017/06/word-embeddings-count-word2veec/">https://www.analyticsvidhya.com/blog/2017/06/word-embeddings-count-word2veec/</a>) to understand different form of word embeddings.

The first step here is to convert it into the *word2vec* format.

```
from gensim.scripts.glove2word2vec import glove2word2vec
glove_input_file = 'glove.6B.100d.txt'
word2vec_output_file = 'glove.6B.100d.txt.word2vec'
glove2word2vec(glove_input_file, word2vec_output_file)
>(400000, 100)
```

Now, we can load the above *word2vec* file as a model.

```
from gensim.models import KeyedVectors # load the Stanford GloVe model
filename = 'glove.6B.100d.txt.word2vec'
model = KeyedVectors.load_word2vec_format(filename, binary=False)
```

Let's say our tweet contains a text saying 'go away'. We can easily obtain it's word vector using the above model:

```
model['go']

model['away']
```

We then take the average to represent the string 'go away' in the form of vectors having 100 dimensions

```
(model['go'] + model['away'])/2
      -0.47944498, -0.431528 , -0.20091 , -0.55664998, -0.32982001,
      -0.088548 , -0.28038502, 0.219725 , 0.090537 , -0.67012
      -0.13080999, 0.92779499, -0.37028 , 0.18854649, 0.66197997,
```

We have converted the entire string into a vector which can now be used as a feature in any modelling technique.

#### **End Notes**

I hope that now you have a basic understanding of how to deal with text data in predictive modeling. These methods will help in extracting more information which in return will help you in building better models.

I would recommend practising these methods by applying them in machine learning/deep learning competitions. You can also start with the Twitter sentiment problem we covered in this article (the dataset is available on the <u>datahack (https://datahack.analyticsvidhya.com/contest/all/)</u> platform of AV).

Did you find this article helpful? Please share your opinions/thoughts in the comments section below.

You can also read this article on Analytics Vidhya's Android APP



<u>(//play.google.com/store/apps/details?</u>

<u>id=com.analyticsvidhya.android&utm\_source=blog\_article&utm\_campaign=blog&pcampaignid=MKT-Other-global-all-co-prtnr-py-PartBadge-Mar2515-1)</u>

#### Share this:

- (https://www.analyticsvidhya.com/blog/2018/02/the-different-methods-deal-text-data-predictive-python/?share=linkedin&nb=1&nb=1)
- (https://www.analyticsvidhya.com/blog/2018/02/the-different-methods-deal-text-data-predictive-python/?share=facebook&nb=1&nb=1)

**G+** (https://www.analyticsvidhya.com/blog/2018/02/the-different-methods-deal-text-data-predictive-python/?share=google-plus-1&nb=1&nb=1)

- (https://www.analyticsvidhya.com/blog/2018/02/the-different-methods-deal-text-data-predictive-python/?share=twitter&nb=1&nb=1)
- (https://www.analyticsvidhya.com/blog/2018/02/the-different-methods-deal-text-data-predictive-python/? share=pocket&nb=1&nb=1)
- (https://www.analyticsvidhya.com/blog/2018/02/the-different-methods-deal-text-data-predictive-python/share=reddit&nb=1&nb=1)

Like this:

l Q Q

Loading...

 ${\tt TAGS:} \ \underline{{\tt BAG}\ OF\ WORDS\ (HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/TAG/BAG-OF-WORDS/)},\ \underline{{\tt FEATURE}\ EXTRACTION}$ 

(HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/TAG/FEATURE-EXTRACTION/), GENSIM

(HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/TAG/GENSIM/), NATURAL LANGUAGE PROCESSING

(HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/TAG/NATURAL-LANGUAGE-PROCESSING/), NLP

(HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/TAG/NLP/), PYTHON

(HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/TAG/PYTHON/), SENTIMENT ANALYSIS

(HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/TAG/SENTIMENT-ANALYSIS/), TEXT DATA MODELING

(HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/TAG/TEXT-DATA-MODELING/), TEXT FEATURE EXTRACTION

(HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/TAG/TEXT-FEATURE-EXTRACTION/), TEXT PREPROCESSING

(HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/TAG/TEXT-PREPROCESSING/), TEXTBLOB

(HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/TAG/TEXTBLOB/). TFIDF

(HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/TAG/TFIDF/). WORD EMBEDDING

(HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/TAG/WORD-EMBEDDING/). WORD VECTORS

(HTTPS://WWW ANALYTICSVIDHYA COM/BLOG/TAG/WORD-VECTORS/)

#### **NEXT ARTICLE**

## **Gartner's 2018 Magic Quadrant Ranks Elite Machine Learning Tools**

(https://www.analyticsvidhya.com/blog/2018/02/gartners-magic-quadrant-data-science-machine-learning/)

•••

#### PREVIOUS ARTICLE

## **Mabl Uses Machine Learning to Automate Functional Testing for Developers**

(https://www.analyticsvidhya.com/blog/2018/02/mabl-automate-functional-testing-machine-learning/)



(https://www.analyticsvidhya.com/blog/author/shubhamjain/)

## **Shubham Jain**

(Https://Www.analyticsvidhya.com/Blog/Author/Shubham-Jain/)

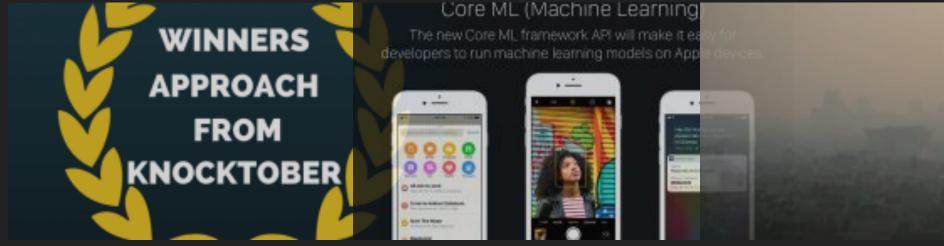
I am currently pursing my B.Tech in Ceramic Engineering from IIT (B.H.U) Varanasi. I am an aspiring data scientist and a ML enthusiast. I am really passionate about changing the world by using artificial intelligence.

(mailto:shubham.jain.cer14@itbhu.ac.in)

in (https://www.linkedin.com/in/shubham-jain-25a104108/)

## **RELATED ARTICLES**

KUNAL JAIN (HTTPS://WWW.ANALYTICSVIDHYA.CIOIM/BEQCIAD ZAKI RIZVI (HTTPS://WWW.ANALYTIOSSIDBING (HTTPS://WWW.ANA



(https://www.analyticsvidhya.com/blog/**2fbttps**1///wwww.analyticsvidhya.com/blog/**2fbttps**9///www.analyticsvidhya approach-codes-from-knocktober-xgbooshachine-learning-iphone-apple-corerst/Jdy-of-factors-contribut dominates/)

How to build your first Machine Learnin@ompdete Study of Factors (

Winners Approach & Codes from Knocktoitehone (Intro to Apple's CoreML) Pollution

Itòsablaboutafeature Estagine ening lytics with the control of the control

xqboost-dominates/)



(https://www.analyticsvidhya.com/blog/名的ttps://wsew.analyticsvidhya.com/blog/名的ttps://www.analyticsvidhya.analyticsvidhya.analyticsvidhya.analyticsvidhya.analyticsvidhya.com/blog/名的ttps://www.analyticsvidhya.analyticsvidhya.com/blog/名的ttps://www.analyticsvidhya.com/blog/analyticsvidhya.com/blog/analyticsvidhya.com/blog/analyticsvidhya.com/blog/analyticsvidhya.com/blog/analyticsvidhya.com/blog/analyticsvidhya.com/blog/analyticsvidhya.com/blog/analyticsvidhya.com/blog/analyticsvidhya.com/blog/analyticsvidhya.com/blog/analyticsvidhya.com/blog/analyticsvidhya.com/blog/analyticsvidhya.com/blog/analyticsvidhya.com/blog/analyticsvidhya.co

This article is quite old and you might not get a prompt response from the author. We request you self-tipes to breaking and you might not get a prompt response from the author. We request you self-tipes to breaking and you might not get a prompt response from the author. We request you self-tipes to breaking and you might not get a prompt response from the author. We request you self-tipes to breaking and you might not get a prompt response from the author. We request you self-tipes to breaking and you self-tipes to be a prompt and you self-tipes to breaking and you self-tipes to break the author. We request you self-tipes to be a self-tipes to be a self-tipes to be author. We request you self-tipes to be author. We request you self-tipes to be a self-tipes to be



YONGDUEK SEO

Reply

<u>February 27, 2018 at 12:48 pm (https://www.analyticsvidhya.com/blog/2018/02/the-different-methods-deal-text-data-predictive-python/#comment-151603)</u>

str(x).split() instead produces better result without empty words.



#### **SOURAV MAHARANA**

Reply

February 27, 2018 at 2:55 pm (https://www.analyticsvidhya.com/blog/2018/02/the-different-methods-deal-text-data-predictive-python/#comment-151606)

Regarding your last section. You used glove model to find similarity between words or find a similar word to the target word. If i want to find a similar document to my target document, then can I achieve this by word embedding? how?



#### **SHUBHAM JAIN**

Renly

May 14, 2018 at 11:09 am (https://www.analyticsvidhya.com/blog/2018/02/the-different-methods-deal-text-data-predictive-python/#comment-153236)

For finding similarity between documents, you can try with help of building document vector using doc2vec.



GIANNI Reply

February 27, 2018 at 4:08 pm (https://www.analyticsvidhya.com/blog/2018/02/the-different-methods-deal-text-data-predictive-python/#comment-151610)

Great job Shubham!

Every Time I peek in AV I got mesmerized  $\stackrel{\ \ \cup}{\ }$  thank you all folks!



SHUBHAM JAIN Reply

March 1, 2018 at 12:19 am (https://www.analyticsvidhya.com/blog/2018/02/the-different-methods-deal-text-data-predictive-pvthon/#comment-151656)

Glad you liked the article.  $\stackrel{ }{ }$ 



JEFF Reply

<u>February 28, 2018 at 2:14 am (https://www.analyticsvidhya.com/blog/2018/02/the-different-methods-deal-text-data-predictive-python/#comment-151627)</u>

Excellent write-up. Keep up the good work. Thank you so much.



#### PRADYUT DASGUPTA

Reply

<u>February 28, 2018 at 8:34 am (https://www.analyticsvidhya.com/blog/2018/02/the-different-methods-deal-text-data-predictive-python/#comment-151631)</u>

Hi, I am not able to find the data set. Kindly help.!



SHUBHAM JAIN

<u>Reply</u>

March 1, 2018 at 12:18 am (https://www.analyticsvidhya.com/blog/2018/02/the-different-methods-deal-text-data-predictive-python/#comment-151655)

You can find the dataset from here.

https://datahack.analyticsvidhya.com/contest/practice-problem-twitter-sentiment-analysis/ (https://datahack.analyticsvidhya.com/contest/practice-problem-twitter-sentiment-analysis/)



MAHESH

<u>Reply</u>

July 25, 2018 at 2:38 pm (https://www.analyticsvidhya.com/blog/2018/02/the-different-methods-deal-text-data-predictive-python/#comment-154246)

I'm not able to find the dataset in the above link. "Data" link present in that page doesn't perform any action at all so, I guess it's removed from that link. Can you pls check once and provide the link witch which I can directly download the dataset?



**PULKIT SHARMA** 

Reply

<u>July 25, 2018 at 8:03 pm (https://www.analyticsvidhya.com/blog/2018/02/the-different-methods-deal-text-data-predictive-python/#comment-154260)</u>

Hi Mahesh,

You can find the dataset from here.

https://datahack.analyticsvidhya.com/contest/practice-problem-twitter-sentiment-analysis/ (https://datahack.analyticsvidhya.com/contest/practice-problem-twitter-sentiment-analysis/)



**VIVEK** 

Reply

February 28, 2018 at 11:35 am (https://www.analyticsvidhya.com/blog/2018/02/the-different-methods-deal-text-data-predictive-python/#comment-151637)

Excellent article, easy to understand.



**SUSANT** 

Repl

March 6, 2018 at 8:02 pm (https://www.analyticsvidhya.com/blog/2018/02/the-different-methods-deal-text-datapredictive-python/#comment-151727)

Ultimate guide ,Shubham..very well written..



**PRAKASH** 

Repl

March 9, 2018 at 6:22 am (https://www.analyticsvidhya.com/blog/2018/02/the-different-methods-deal-text-datapredictive-pvthon/#comment-151787)

Very useful article.



**MARCEL** 

<u>nepry</u>

March 12, 2018 at 1:39 pm (https://www.analyticsvidhya.com/blog/2018/02/the-different-methods-deal-text-data predictive-python/#comment-151845)

My compliments for this nice article



SATISH Reply

March 16, 2018 at 6:32 pm (https://www.analyticsvidhya.com/blog/2018/02/the-different-methods-deal-text-data-predictive-python/#comment-151943)

Can you please elaborate on N-grams.. what the use of n-grams and what happens if we choose high n values



**SHUBHAM JAIN** 

Reply

May 14, 2018 at 11:20 am (https://www.analyticsvidhya.com/blog/2018/02/the-different-methods-deal-text-data-predictive-pvthon/#comment-153237)

N-grams are generally preferred to learn some sequential order in our model. We prefer small values of N because otherwise our model will become very slow and will also require higher computational power. So, instead of using higher values of N, we generally prefer using sequential modeling techniques like RNN, LSTM.

Hope this helps

Shubham



VANI

Reply

March 19, 2018 at 2:11 pm (https://www.analyticsvidhya.com/blog/2018/02/the-different-methods-deal-text-datapredictive-python/#comment-151996)

can u suggest some topic related to textdata for research



**MEL** 

Reply

<u>April 2, 2018 at 3:45 pm (https://www.analyticsvidhya.com/blog/2018/02/the-different-methods-deal-text-data-predictive-python/#comment-152314)</u>

Good day - Thank you for the example. It provides good guidelines to newbies like me.

I was able to follow your example right up til 3.3 Inverse Document Frequency, but sample code does not seem to work, Additionally, the output provided seems to come from another dataset or rather a copy /paste from a previous article ?

Finally, the numerical sections following are not labeled correctly. Jumping from 3.3 to 3.34 then 4.5, 4.6



#### **SHUBHAM JAIN**

Reply

<u>April 3, 2018 at 8:11 pm (https://www.analyticsvidhya.com/blog/2018/02/the-different-methods-deal-text-data-predictive-python/#comment-152334)</u>

The code seems to be fine with me. And the output is also correct. Try to follow the preprocessing steps properly and then run it again.

As far as the numbering of sections is concerned, they were just mistakenly put by me. Not a big issue though since it is clear from the table of content. Still, I have updated it.

Regards,



**RUBEN** 

Reply

<u>April 22, 2018 at 9:59 pm (https://www.analyticsvidhya.com/blog/2018/02/the-different-methods-deal-text-data predictive-python/#comment-152768)</u>

Hi Shubham, great tutorial!

Only thing is that I'm getting stuck at the same point (3.3 ITF):

[CODE]

NameError Traceback (most recent call last)

in ()

1 for i, word in enumerate(tf1['words']):

 $--> 2 \ tf1.loc[i, \ 'idf'] = np.log(train.shape[0]/(len(train[train['tweet'].str.contains(word)])))$ 

4 tf1

NameError: name 'np' is not defined [/CODE]

I've cleared the notebook output multiple times, but it keeps giving me the same error. I'll appreciate any help, thanks!



**GANESH** 

<u>Reply</u>

June 19, 2018 at 12:55 pm (https://www.analyticsvidhya.com/blog/2018/02/the-different-methods-deal-text-data-predictive-python/#comment-153863)

use below one and proceed

import numpy as np



**DENNIS**Reply

<u>April 18, 2018 at 8:04 pm (https://www.analyticsvidhya.com/blog/2018/02/the-different-methods-deal-text-data-predictive-python/#comment-152659)</u>

how do you now use the trained model



HAKAN Reply

<u>April 25, 2018 at 2:38 am (https://www.analyticsvidhya.com/blog/2018/02/the-different-methods-deal-text-data-predictive-python/#comment-152821)</u>

Hi Shubham,

Thank you for the article. It is really helpful for text analysis. One thing I cannot quite understand is how can I use features I extracted from text such as number of numerics, number of uppercase with TFIDF vector. I couldn't find an intuitive explanation or example of this. Could you be able to make an example of it?

Thanks again.



DATTA SAI Repl

May 18, 2018 at 12:46 pm (https://www.analyticsvidhya.com/blog/2018/02/the-different-methods-deal-text-datapredictive-python/#comment-153367)

what is the pd there in:

freq = pd.Series(' '.join(train['tweet']).split()).value\_counts()[-10:]
thanks in advance



#### **AISHWARYA SINGH**

Reply

May 18, 2018 at 1:51 pm (https://www.analyticsvidhya.com/blog/2018/02/the-different-methods-deal-text-data-predictive-pvthon/#comment-153370)

Hi Datta,

pd here represents pandas. The library pandas is imported as pd.



SRIRAM Reply

July 4, 2018 at 2:19 am (https://www.analyticsvidhya.com/blog/2018/02/the-different-methods-deal-text-datapredictive-python/#comment-154050)

Hi Shubham, great article, thanks.

# **TOP ANALYTICS VIDHYA USERS**

Rank	Name		Points
1		SRK (https://datahack.analyticsvidhya.com/user/profile/SRK)	9688
2	9	Rohan Rao (https://datahack.analyticsvidhya.com/user/profile/Rohan_Rao)	9200
3		aayushmnit (https://datahack.analyticsvidhya.com/user/profile/aayushmnit)	7779
4		mark12 (https://datahack.analyticsvidhya.com/user/profile/mark12)	7212



More Rankings (http://datahack.analyticsvidhya.com/users)

## **POPULAR POSTS**

- 24 Ultimate Data Science Projects To Boost Your Knowledge and Skills (& can be accessed freely)
   (https://www.analyticsvidhya.com/blog/2018/05/24-ultimate-data-science-projects-to-boost-your-knowledge-and-skills/)
- A Complete Tutorial to Learn Data Science with Python from Scratch

(https://www.analyticsvidhya.com/blog/2016/01/complete-tutorial-learn-data-science-python-scratch-2/)

- Essentials of Machine Learning Algorithms (with Python and R Codes)
   (https://www.analyticsvidhya.com/blog/2017/09/common-machine-learning-algorithms/)
- The 25 Best Data Science and Machine Learning GitHub Repositories from 2018
   (https://www.analyticsvidhya.com/blog/2018/12/best-data-science-machine-learning-projects-github/j
- 7 Types of Regression Techniques you should know!
   (https://www.analyticsvidhya.com/blog/2015/08/comprehensive-guide-regression/
- Understanding Support Vector Machine algorithm from examples (along with code)
   (https://www.analyticsvidhya.com/blog/2017/09/understaing-support-vector-machine-example-code/)
- 6 Easy Steps to Learn Naive Bayes Algorithm (with codes in Python and R)
   (https://www.analyticsvidhya.com/blog/2017/09/naive-bayes-explained/)
- The 15 Most Popular Data Science and Machine Learning Articles on Analytics Vidhya in 2018 (https://www.analyticsvidhya.com/blog/2018/12/most-popular-articles-analytics-vidhya-2018/)

#### **RECENT POSTS**

DataHack Radio #16: Kaggle Grandmaster SRK's Journey and Advice for Data Science Competitions (https://www.analyticsvidhya.com/blog/2019/01/datahack-radio-tips-crack-data-science-competitions-kaggle-grandmaster/)

JANUARY 24, 2019

Introduction to Monte Carlo Tree Search: The Game-Changing Algorithm behind DeepMind's AlphaGo (https://www.analyticsvidhya.com/blog/2019/01/monte-carlo-tree-search-introduction-algorithm-deepmind-alphago/)

**JANUARY 24, 2019** 

Top 10 Presentations from rstudio::conf 2019 – The Best R Conference of the Year! (https://www.analyticsvidhya.com/blog/2019/01/top-highlights-rstudioconf-2019-best-r-conference/)

**JANUARY 22, 2019** 

Must-Read Tutorial to Learn Sequence Modeling (deeplearning.ai Course #5) (https://www.analyticsvidhya.com/blog/2019/01/sequence-models-deeplearning/)

**JANUARY 21, 2019** 



http://www.edvancer.in/certified-data-scientist-with-

pvthon-course?

utm\_source=AV&utm\_medium=AVads&utm\_campaign=AVadsnonfc&utm\_content=pythonavad



# Learn Python, Statisti



(https://trainings.analyticsvidhya.com/courses/course-

v1:AnalyticsVidhya+DS101+2018T2/about?utm\_source=Blog&utm\_medium=Sticky\_banner1)



Almost 1.5 Billion Seconds of Videos Are Being Uploaded To Youtube Every Day!

(https://trainings.analyticsvidhya.com/courses/course-

v1:AnalyticsVidhya+CVDL101+CVDL101\_T1/about?
utm\_source=Blog&utm\_medium=Sticky\_banner2&utm\_campaign=cv10percent



**JOIN OUR COMMUNITY: ANALYTICS DATA COMPANIES VIDHYA SCIENTISTS** 

(https://www.analyticsvidhya.com/corporate/)

(http://www.analytittssidhyavcamatytigsya.com/bittss://www.facebtos://bittssidhyavcamatytigsya.com/bittss://www.facebtos://bittssidhyavcamatytigsya.com/bittss://www.facebtos://

(https://datahack-analyticsvidhya.com/) (https://www.fac**elotops://tovit/Anabytics/** 

(https://www.analyticsvjchya.com/sheenleing (https://www.facebbbok://twittAnabyticsrv analytics-

(https://plus.googhetps://hi/a.Amlaeydiosx.od

(https://plus.googhetps://hi/a.Amlaeydiosx.od

Subscribe to emailer

