Text preprocessing level -2:

**Bag of words:**

Here it will create a unique id or unique vector of numerical with 0’s and 1’s as one hot encoding does for categorical data, but the difference is It will create it for the entire dictionary of words that are present in the input data taken after eliminating the stopwords

Eg:

Sent 1 = He is a good boy

Sent 2 = She is a good girl

Sent 3 = Boy and girl are good.

* Lower the sentences first to get out of the duplicates eg: “he” and “He” and “HE” etc.
* But in some cases like country name “US” and ‘us’ general word refers us are both different so it can be handled in preprocessing( take out the county name with index number in particular sentence and place them after lowering the sentence)

Stop keywords using stopwords.

After analysis i.e stopwords and lowering

Sent 1 = good boy

Sent 2 = good girl

Sent 3 = boy girl good

Histogram of above sentences: (sort them in descending order).

|  |  |
| --- | --- |
| Word | Frequency |
| Good | 3 |
| Boy | 2 |
| Girl | 2 |

Vectors: with bag of words

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Good | Boy | Girl | Output |
| Sent 1 | 1 | 1 | 0 |  |
| Sent 2 | 1 | 0 | 1 |  |
| Sent 3 | 1 | 1 | 1 |  |

Bag of words:

Binary: as above (like using binary numbers 0’s and 1’s ) (1 for existing word and 0 for non existing)

Normal: increment the count and replace it in place of 1 as shown below.

Eg:

Sent 1 = good boy boy

Sent 2 = good girl good

Sent 3 = boy girl good

Histogram of above sentences: (sort them in descending order).

|  |  |
| --- | --- |
| Word | Frequency |
| Good | 4 |
| Boy | 3 |
| Girl | 2 |

Normal: Vectors: with bag of words

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | F1 | F2 | F3 |  |
|  | Good | Boy | Girl | Output |
| Sent 1 | 1 | 2 | 0 |  |
| Sent 2 | 2 | 0 | 1 |  |
| Sent 3 | 1 | 1 | 1 |  |

Here F1, F2, F3 are independent features and output column can be anything that is directly taken from the input data as a class to classify the vectors in ML model.

Disadvantages:

Values are either 1 and 0’s. with this there is no more importance or weightage like ‘good ‘ is more important but with bag of words it is all equal with 1’s and 0’s .

Overcome: TFIDF (Term Frequency Inverse Document Frequency)

\*\* bag of words 🡪 for small data and some simple sentiment analysis or so. But for big data or large data use ‘word2vec’ .

Code:

‘import re’ to clean the text data with only alphabets and remove punctuation marks, numericals etc.

‘review = re.sub(‘[^a-zA-Z]', ' ', sentences[i])

To lower

‘review = review.lower()’

To split in to words.

review = review.split()

to use unique words and remove stop words using list comprehension to get that data using lemmatization

review = [wordnet.lemmatize(word) for word in review if not word in set(stopwords.words('english'))]

review = ' '.join(review)

corpus.append(review)

Convert to Bag of words:

“””

from sklearn.feature\_extraction.text import CountVectorizer

cv = CountVectorizer(max\_features = 1500)

x = cv.fit\_transform(corpus).toarray()

“””

The output is an array.

array([[0, 0, 0, ..., 0, 0, 0],

[0, 0, 0, ..., 1, 1, 0],

[0, 1, 0, ..., 0, 0, 0],

...,

[0, 0, 0, ..., 0, 0, 0],

[0, 0, 0, ..., 0, 0, 0],

[0, 0, 0, ..., 0, 0, 0]], dtype=int64)

Shape of it is 31, 114 -🡪 31 sentences, and 114 unique words on the whole data paragraph taken in preprocessing task level 1

**TFIDF :**

Term Frequency and Inverse Document frequency

Term Freq : no of rep of words in sentence/no of words in sentence

IDF : log (no of sentences/no of sentences containing words)

* Convert sentences in to vectors,

finally: TF \* IDF (for each word)

eg;

sent 1 = good boy

sent 2 = good girl

sent 3 = boy girl good

histogram of each word

|  |  |
| --- | --- |
| Word | Frequency |
| Good | 3 |
| Boy | 2 |
| Girl | 2 |

TF:

|  |  |  |  |
| --- | --- | --- | --- |
|  | Sent 1 | Sent 2 | Sent 3 |
| Good | ½ | ½ | 1/3 |
| Boy | ½ | 0 | 1/3 |
| Girl | 0 | ½ | 1/3 |

IDF:

|  |  |
| --- | --- |
| Words | IDF |
| Good | Log(3/3) = 0 |
| Boy | Log(3/2) |
| Girl | Log(3/2) |

TF\*IDF:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | F1 | F2 | F3 | O/p |
|  | Good | Boy | Girl |  |
| Sent 1 | 0 | ½ \* log(3/2) | 0 |  |
| Sent 2 | 0 | 0 | ½\*log(3/2) |  |
| Sent 3 | 0 | 1/3\*log(3/2) | 1/3\*log(3/2) |  |

Here the weightage is given to the words as shown above which is a drawback in bag of words with either normal or binary.

The weightage is given to those words which are unique in a particular sentence and assigning 0 or approx. to 0 value as the word occurs in all sentences and it doesn’t make any sense to include that in the classification problem.

**Word2Vec:**



**Drawbacks of Bag of Words and TF-IDF:**

Both BOW and TF-IDF approach semantic information is not stored.

TF-IDF gives importance to uncommon words

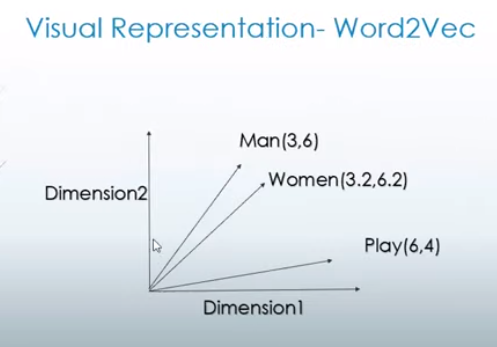
There is definitely chance of over fitting

**Overcome: Word2Vec models**

Instead of assigning a value to the word and getting the whole sentence as a vector, here in word2vec representing each word with a vector

Each word is basically represented as a vector of 32 or more dimension instead of a single number

Here the semantic information and relation between different words is also preserved,



Let’s take an example of 2 dimensional;

Each word is represented by 2-dimensional vector

Here in the above example man and women are with little or less distance to each other because there semantic information is similar (human). And play has the different vector which is far away from the remaining as it has different meaning.

Overall conclusion the word2vec will arrange the vectors of each word is with semantic information.

Eg: to approach the similar words there is some kind of calculation as below.

King – man + woman = Queen (king and queen are same with same power or with same sematic information).

Steps to create word2vec:

* Tokenization of the sentences
* Create histogram
* Take most frequent words
* Create a matrix with all unique words. It also represents the occurrence of relation between the words.

**Gensim library has word2vec module to perform.**

Note: can create a 100 dimensional vectors also in genism library word2vec modules.

Git hub:

<https://github.com/RaRe-Technologies/gensim/wiki/Migrating-from-Gensim-3.x-to-4>

Check the notebook to know how to do the word2vec in genism library.

Created vectors to each word, and condition that min\_count of that word is atleast 1 but in practical 2 is minimum.

Code is as below.

‘model = Word2Vec(sentences, min\_count=1)’

And model.wv gives the vocab and list of all words considered In here.

Len(model.wv) gives the number of words

And to get more about the usage of word2vec module check out the above link

<https://github.com/RaRe-Technologies/gensim/wiki/Migrating-from-Gensim-3.x-to-4>