Sinhgad Institutes | Date

Name: Pallavi Vijay Patil
Roll No.: COTB028
Subject: DSBDA

Assignment No. 7

- Aim: Test Analytics

1) Extract sample document and apply following document preprocessing methods: Tokenization, Pos Tagging, stop words removal, stemming & lemmitazation.

2) Create representation of document by calculating term frequency and inverse document frequency.

- Theory:

TF-IDF from scratch in Python on real-world dataset.

- Introduction to: TF-IDF

TF-IDF stands for "Term Frequency-Inverse Document Frequency". This is a technique to quantify words in a set of documents. We generally compute a score for each word to signify its importance in document. This method is widely used technique in info" Retrieval and Text Mining.

If I give you a sentence for example "This building is so tall". It's easy for us to understand the sentence as we know semantics of words and sentence. But how can any program (e.g. python) interpret this sentence?

- Term Frequency:
This measures the frequency of word in document. This highly depends on length of document and generality of word, for ex. a very common word such as "was" can appear multiple times in document. But if we take two documents with 100 words & 10,000 words resp., there is high probability that common word "was" is present more in 10,000 worded document.

- Document Frequency:
This measures the importance of documents in whole set of corpus. This is very similar to TF but only difference is that TF is frequency counter for term t in document d, where as DF is count of occurances to term t in document set N. In other words, DF is the no. of documents in which word is present. We consider one ~~occure~~ occurence if term is present in document at least once, we do not need to know number of times the term is present.

df(t) = occurrence of t in N documents.

- Inverse Document Frequency:
IDF is the inverse of the document frequency which measures the informativeness of term t. When we calculate IDF, it will be very low for the most occurring words such as stop words. This finally gives what we want, relative weightage.
idf(t) = N/df.
Now there are few problems with IDF, when we have a large corpus size say N=10000, the IDF value explodes.

- Implementing on real world dataset:
Step 1: Analysing Dataset
Step 2: Extracting Title and Body.
Step 3: Preprocessing.

- Lowercase:
During text processing, each sentence is split into words and each words is considered as token after preprocessing. Programming languages consider textual data as sensitive, which means that ~~The~~ is different from the. Numpy has method that can convert the list of lists to lowercase at once.

np.char.lower(data)

- Stopwords:
Stop words are most commonly occurring words that can don't give any additional value to document vector. In fact removing these will increase computation and space efficiency.

- Punctuation:
Punctuation is set of unnecessary symbols that are in our corpus documents. We should be a little careful with what we are doing with this, there might be few problems such as U.S - u.s. "united state" being converted to "us" after preprocessing.

- Apostrophe:
Note that there is no apostrophe in the punctuation symbols. Because when we remove punctuation first it will convert don't to dont, and it is stop world that won't be removed.

```
return np.char.replace(data, "'", "")
```

- Single Characters:
Single characters are not much useful in knowing the importance of document & few final single characters might be irrelevant symbols, so it is always good to remove single characters.

- Stemming:
This is the final and most important part preprocessing, stemming converts words to their stem.
For ex, playing and played are same type of words that basically indicate an action play.

- Lemmatisation:
Lemmatisation is a way to reduce the word to root synonym of a word. Unlike stemming, lemmatisation make sure that the reduced word is again a dictionary word.

- Calculating TF:
Let us be smart and calculate DF beforehand we need to iterate through all the words in all the documents and store document id's for each word. For this, we will use a dictionary as we can use word as the key and set of documents as value.

- Vectorization:
To compute any of above, the simplest way is to convert anything to vector and then compute the cosine similarity. So, let's convert query and documents to vectors.

- Conclusion:

∴ Thus, we have successfully learnt calculating term frequency and Inverse document Frequency also learnt methods like stemming, lowercase, stop words, etc.