NLP

Wednesday, October 10, 2018 3:57 PM

https://blog.insightdatascience.com/how-to-solve-90-of-nlp-problems-a-step-by-step-guide-fda605278e4e

1. Remove all irrelevant characters such as any non alphanumeric characters
2. Tokenize your text by separating it into individual words
3. Remove words that are not relevant, such as "@" twitter mentions or urls
4. Convert all characters to lowercase, in order to treat words such as "hello", "Hello", and "HELLO" the same
5. Consider combining misspelled or alternately spelled words to a single representation (e.g. "cool"/"kewl"/"cooool")
6. Consider lemmatization (reduce words such as "am", "are", and "is" to a common form such as "be")

From <https://blog.insightdatascience.com/how-to-solve-90-of-nlp-problems-a-step-by-step-guide-fda605278e4e>

Word2Vec creation :-https://machinelearningmastery.com/develop-word-embeddings-python-gensim/

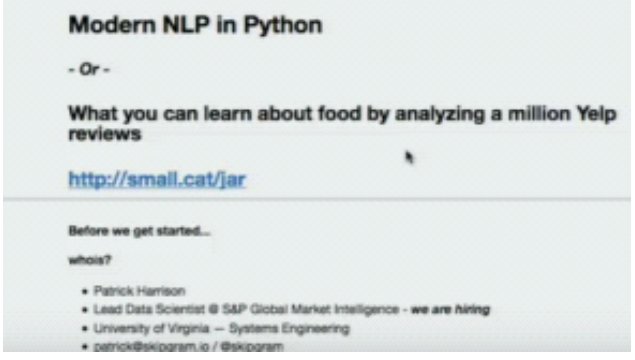
Gensim :--https://www.youtube.com/watch?v=lfqW46u0UKc

Topic Modelling :- unlabeled textual data

https://github.com/bhargavvader/personal/tree/master/notebooks/text_analysis_tutorial

Lsi
LDA
Hdp model

PyDavis:--for visualisation of the Topics and how god topics are coming out



Tf-idf is a scoring scheme for words - that is a measure of how important a word is to a document.

Glove and Word2vec are both unsupervised models for generating word vectors. The difference between them is the mechanism of generating word vectors. The word vectors generated by either of these models can be used for a wide variety of tasks ranging such as

- finding words that are semantically similar to a word,
- representing a word when it is being input to a downstream model. A word embedding representation of a word captures more information about a word than just a one-hot representation of the word, since the former captures semantic similarity of that word to other words whereas the latter representation of the word is equidistant from all other words.

Data Cleaning

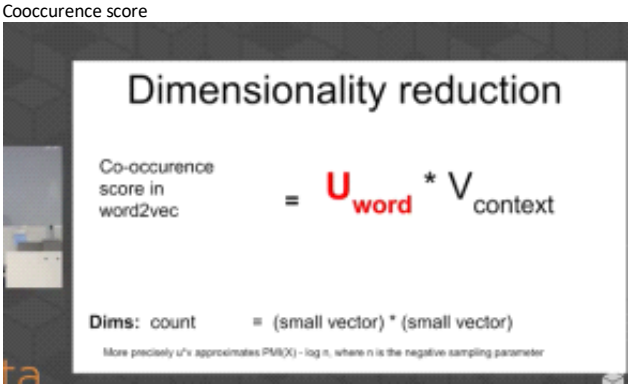
```
import re from nltk.corpus import stopwords import pandas as pd
def preprocess(raw_text): # keep only words letters_only_text =
re.sub("[^a-zA-Z]", " ", raw_text) # convert to lower case and
split words = letters_only_text.lower().split()
# remove stopwords stopword_set = set(stopwords.words("english"))
meaningful_words = [w for w in words if w not in stopword_set] #
join the cleaned words in a list cleaned_word_list = "
".join(meaningful_words) return cleaned_word_list def
process_data(dataset): tweets_df =
pd.read_csv(dataset,delimiter='|',header=None) num_tweets =
tweets_df.shape[0] print("Total tweets: " + str(num_tweets))
cleaned_tweets = []
print("Beginning processing of tweets at: " + str(datetime.now()))
for i in range(num_tweets): cleaned_tweet =
preprocess(tweets_df.iloc[i][1])
cleaned_tweets.append(cleaned_tweet) if(i % 10000 == 0):
print(str(i) + " tweets processed") print("Finished processing of
tweets at: " + str(datetime.now())) return cleaned_tweets
cleaned_data = process_data("tweets.csv)

>
```

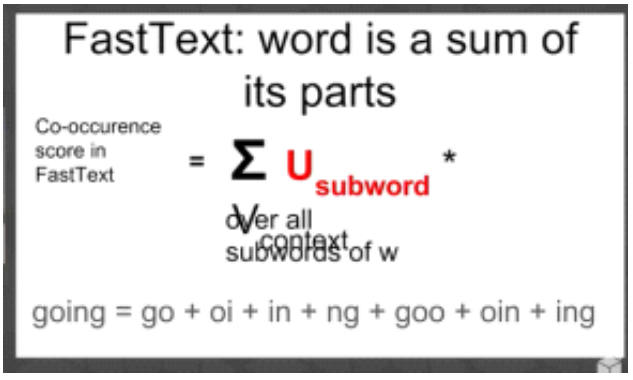
From a practical usage standpoint, while tf-idf is a simple scoring scheme and that is its key advantage, word embeddings may be a better choice for most tasks

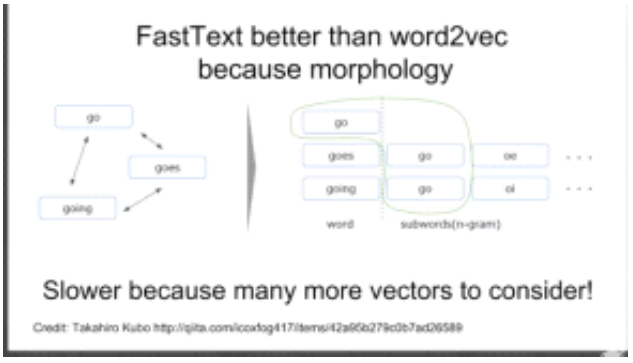
where tf-idf is used, particularly when the task can benefit from the semantic similarity captured by word embeddings *(e.g. in information retrieval tasks)*

https://www.youtube.com/watch?v=7530Tn2J0Mc
WordVec
Cooccurence score



Fast text (sub words





FastText Gensim Wrapper

Same API as word2vec.
Out-of-vocabulary words can also be used, provided they have at least one character n-gram present in the training data.

```
In [7]: print("nights" in model.wv.vocab)
        print("night" in model.wv.vocab)
        model.similarity("night", "nights")

        False
        True
Out[7]: 0.97944545147919504
```

RARE

Many ways to get a vector for a word

- **Word2vec**
- **FastText**
- **WordRank**
- Factorise the co-occurence matrix: SVD/LSI
- GLoVe
- EigenWords
- VarEmbed

WordRank is a Ranking Algorithm

Word2vec

Input: Context **Cute**

Output: Word **Kitten**

Classification problem

WordRank

Input: Context **Cute**

Output: *Ranking*

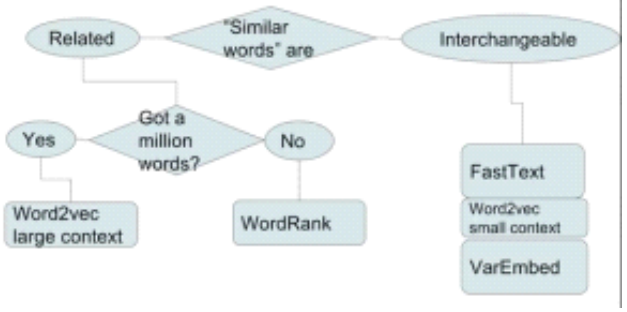
1. Kitten

2. Cat

3. Dog

Robust: Mistake at the top of the rank costs more than mistake at the bottom.

How to get the similarity you need



RARE

What is a word embedding?

"Word embedding" = "word vectors" = "distributed representations"

It is a **dense** representation of words in a **low-dimensional vector space**.

One-hot representation:

king = [1 0 0 0.. 0 0 0 0 0]

queen = [0 1 0 0 0 0 0 0 0]

book = [0 0 1 0 0 0 0 0 0]

Distributed representation:

king = [0.9457, 0.5774, 0.2224]