Part 1 - Vector Models and Text Processing

27 April 2023 23:36

- Sentence - sequence of words begins with capitalized word, end with punctuation.
- Token - Token can be words, punctuation, sub-unit of words
- Characters - letters, punctuations, whitespace
- Vocabulary - the set of "all" words
- Corpus - A large collection of writings of a specific kind or on a specific subject.
- N-Gram - refers to N consecutive items(eg -words, subwords,characters)
  - Data -> 1-Gram
  - Great Day -> 2-Gram
  - I am fine -> 3-Gram
  - Nice to meet you -> 4-Gram

- Bag of Words
  - Count Vectorizer is a bag of words approach .
    - determine the size of vocabulary (unique tokens in training corpus) - Size V
    - Each document will be converted into a vector of size V

      - V (vocabulary size) = 6

      Each row is a vector of size 6

Original Documents
I like eggs
I hate cats
I like eggs and I like cats

and	cats	eggs	hate	I	like
0	0	1	0	1	1
0	1	0	1	1	0
1	1	1	0	2	2

    - CountVectorizer in sklearn doesn't have normalization, but tf-idf has.

- Level of Tokenization:
  - word based tokenization
  - character based tokenization
  - subword-based tokenization

Subword-Based Tokenization

- What if we *didn't* split "walking" into "walk" + "ing"?
- Each vector component (count) is separate, so "walk" is no closer to "walking" than it is to "tree"
- We can only hope our model learns the similarity through the data
- Do we want our model to learn "walk", "walks", "walking", "walked", etc. independently? Or should we connect them via a shared representation?
- I make a strong case for subword tokenization, but we won't see it again until we study Transformers / deep learning

- Tokenization
  - Punctuations
  - Case
  - accent
    - sklearn countvectorizer , make CountVectorizer(strip_accents=True)

- Removing stopwords
  - High dimensionality is bad, so better to not include stopwords
  - use:
    - CountVectorizer(stop_words="english")
    - CountVectorizer(stop_words=list_of_user_defined_terms) #helpful when you are working in a niche industry and english stopwords don't have them
    - Or use, stopwords from nltk
      - nltk.downloads('stopwords')
      - from nltk.corpus import stopwords
      - stopwords.words('german')

Stemming and Lemmatization:

Stemming vs. Lemmatization

- Stemming is very **crude** - it just chops off the end of the word
- The result is not necessarily a real word

- Lemmatization is more sophisticated, uses actual rules of language
- The true root word will be returned

Lemmatization

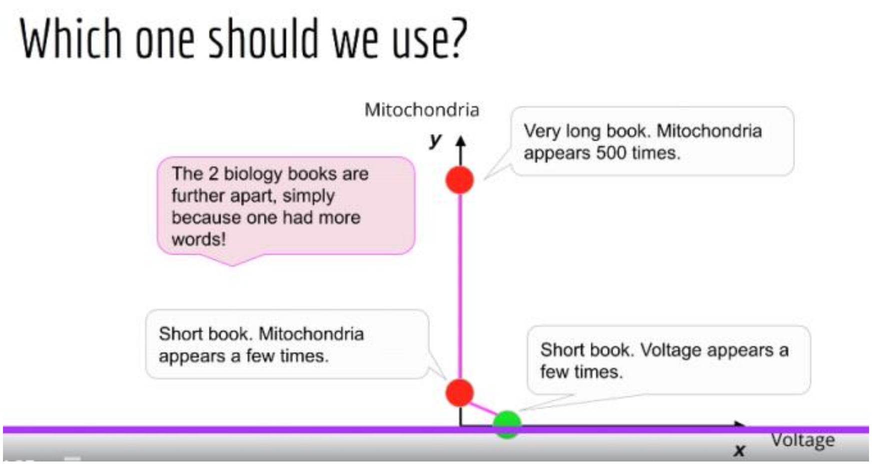
- Think of it as a lookup table / table of rules
- Stemming: "Better" → "Better"
- Lemmatization: "Better" → "Good"
- Note: "Was" is the past-tense of "Is", both are derivatives of "Be"
- Stemming: "Was" → "Wa"
- Lemmatization: "Was" / "Is" → "Be"
- Stemming: "Mice" → "Mice"
- Lemmatization: "Mice" → "Mouse"

- Lemmatization using nltk:
  - Appears in NLTK, spaCy, and others
  - ```
    from nltk.stem import WordNetLemmatizer
    from nltk.corpus import wordnet
    nltk.download("wordnet") # only need to do once

    lemmatizer = WordNetLemmatizer()
    lemmatizer.lemmatize("mice") # returns 'mouse'

    lemmatizer.lemmatize("going") # returns 'going'
    lemmatizer.lemmatize("going", pos=wordnet.VERB) # returns 'go'
    ```
  - Use pos(part of speech tagging) with lemmatizer - as default is always noun.

- Vector Similarity:
  - Euclidean distance
  - Cosine distance
  - When we have to deal with vector of different sizes , cosine distance may be of more use than euclidean distance.
    ```

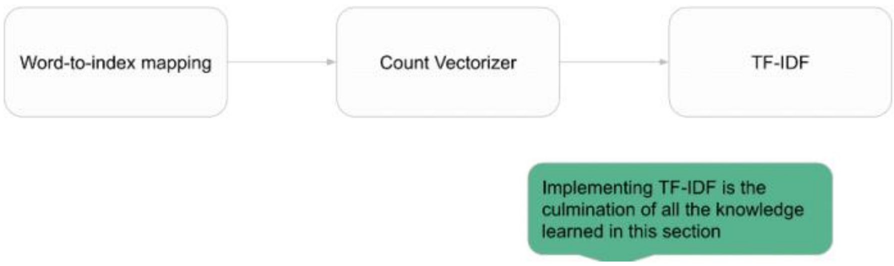


Cosine Similarity does not take into account the magnitude of the vectors. Row-normalised have a magnitude of 1 and so the Linear Kernel is sufficient to calculate the similarity values.

From <https://stackoverflow.com/questions/12118720/python-tf-idf-cosine-to-find-document-similarity>

- So , tf-idf can be used to find document similarity as well.

TF-IDF from scratch:

- 

- Neural Word Embedding:
  - Word2vec
  - glove

- Text Summarization types:
  - Extractive
    - easy to generate
  - Abstractive
    - complex

- Text Summariaztion Using tfidf:

Text Summarization with TF-IDF

- High-level outline
- Split the document into sentences
- Score each sentence
- Rank each sentence by those scores
- Summary = top scoring sentences

More Details - Scoring Each Sentence

- Score = Average(non-zero TF-IDF values)
- E.g. if row = [0, 1, 0, 0, 0, 2, 3, 0, 0, 0, ...] then score = avg(1,2,3) = 2
- Why does it work?
- Each TF-IDF component tells us how often a word appears (TF)
- But if a word appears across many sentences, it will shrink (IDF)
- Important words will have a larger score
- Why mean and not sum?
- The sum would be biased toward longer sentences
- Why only the non-zero values?
- TF-IDF matrix is sparse (don't want to choose based on variety of words)

More Details - What To Do With The Scores

- Idea: sort the scores, pick the sentences with the highest scores
- How? There are multiple options: you choose what works best
- Simple: top N sentences (e.g. top 5, top 10)
- Also simple: top N words, top N characters (e.g. if limited by space)
- Top X% of sentences, top X% of words / characters
- Sentences with score > threshold (e.g. threshold = average score)
- Or threshold = average score * factor

Text Summarization Exercise Prompt

- Dataset: use any article you like (we'll be using BBC News again)
- Try it on multiple articles
- Split the article into sentences (nltk.sent_tokenize)
- Compute TF-IDF matrix from list of sentences
- Score each sentence by taking the average of non-zero TF-IDF values
- Sort each sentence by score
- Print the top scoring sentences as the summary

- for word similarity using libraries, some points:
  - NLTK:
    - we can train using our own text corpora, then find similar word using context based similarity
      - nltk.text.similar
    - Or,
      `we can use `from nltk.corpus import wordnet` and use pre-existing synsets to get similar words`
  - Spacy:
    - You can use pretrained word2vec model
  - gensim:
    - you can use both pretrained or you can train on your data too
  - Glove

```
#### training our own Word2Vec model using gensim Word2Vec
Steps:
* use sentence tokenizer to tokenize document into sentence
* tokenize each sentence into word
* preprocess and remove stopwords,punctuation
* so something like this we would have : document -> tokenized into sentence ->
each sentence tokenized into words
* now use gensim.models Word2Vec to create model
```