Norges teknisk–naturvitenskapelige universitet
Institutt for datateknologi og informatikk

TDT 4310
Spring 2023

Lab 2

1 Text normalization

Text normalization deals with a large set of techniques, but most commonly:

- Dealing with unwanted characters and symbols
- Lower/uppercasing
- Stopword filtering
- Stemming and lemmatization

The last point is the task of converting text to its standard form. While stemming is a simpler process of reducin words to a common core (*stem*), often by removing suffixes, lemmatization will use dictionaries and definitions to convert words to their base form. The differences can be illustrated as follows:

Word	Stemming	Lemmatization
information	inform	information
informative	inform	informative
computers	comput	computer
feet	feet	foot

1) *Given the current state of your smart keyboard (from Lab 1), how do you think you could use the above to improve upon the prediction task? Also discuss why it may fail to give us the words we expect.*
- I did some removal of punctuation symbols and lowercasing for the last lab (however I did not focus on it for this one). I think lowercasing is more helpful when the symbols are removed, since it can be helpful to see that capitalization should come when a new sentence starts (which is often after a '.').
However, these measures will come with drawbacks. For instance will capitalization of names, countries, etc. not be suggested correctly, unless you make extra measures in order for that to happen. Also, sometimes you want symbols to be suggested, in order to help the user to write with "correctpunctuation. When it comes to stopwords I think they are very valuable for the smart keyboard to suggest, and wouldn't want to remove them. They are more important to remove when we're trying to deduce what the semantics of the text are, and I think our keyboard is way dumber than that at this point. Also, for stemming and lemmatization, doing them could give us a better data set, but it would again require extra measures to make sure that the full form"of the word is what is suggested back - we don't want to only predict the simplest word forms to our user.

2 TF-IDF

TF-IDF (Term Frequency-Inverse Document Frequency) is a way to measure the importance of a word in a document.
Consider the following documents:

- *I love cats*
- *I love dogs*
- *I love cats, but I also like dogs*

1) *Calculate the TF-IDF for '**love**' and '**like**' for the last document.*

- *Tip: use NLTK's FreqDist to get the bag-of-words (as seen in the last part of the lab 1 handout). Apply preprocessing you think is necessary.*

- For preprocessing, I lowercased, removed stopwords and lemmatized. This left me with:

- $['love', 'cat']$
- $['love', 'dog']$
- $['love', 'cat', 'also', 'like', 'dog']$

$TF = \frac{\text{number of times the term appears in the document}}{\text{total number of terms in the document}}$
$IDF = log(\frac{\text{number of documents in the corpus}}{\text{number of documents that contain the term}})$
'love': $\frac{1}{5} * log(\frac{3}{3}) = 0$
'like': $\frac{1}{5} * log(\frac{3}{1}) = 0.0954$

2) *Would you replace words with their TF-IDF values? Why or why not?*
- If we're doing tasks like text searches, we can use the values to determine what words are more likely to help us - say the input is a bunch of words with a low TF-IDF, but one has a high one, we can assert that looking for texts with this particular word is more likely to give us good results than looking for the other (more insignificant) words.

3) *Why should you use the logarithm of the inverse document frequency?*
- If we have a high number of documents in the corpus, the IDF value is likely to get very high. We use log to dampen its effect on the value.

3 Part-of-speech tagging

Both NLTK and spaCy supports POS-tagging. Let's compare a few different approaches.
Consider the sentence:
I saw her duck
And print out the POS tags by...

1) *Training a UnigramTagger with a DefaultTagger as backoff (defaulting to nouns).*

- *See the nltk.tag module. The training data should be the Brown corpus.*

- PPSS | VBD | PP$ | VB

2) *Using spaCy built in tagger*
- PRON | VERB | PRON | NOUN

3) *Discuss why you think the results differ*
- They are trained on different data. I tested the nltk UnigramTagger with only

categories='news', and it went with the default. They also use a different tag set. The UnigramTagger was trained on the brown corpus, and it has a larger tagset than the spacy 'en_core_web_sm' tagger does. So they aren't necessarily giving different tags for 'I'; PPSS is just a more narrow/precise tag than PRON.

4 Implementation

For Lab 2, I want you to get more familiar with spaCy, while reusing the models created in Lab 1. As usual, the details are found in TODOs in the python file (lab2.py).

A running version of Lab 1 will also be published in case you had trouble with your implementation.

1) *As a final part of the lab, I want you to briefly discuss your approach to the implementation task. This could be things you had to learn, difficult parts of the lab, etc.*

- Once I had an idea of what I wanted to do, the task was pretty okay. I was however stuck for a long time figuring out a sensible way to use POS to strengthen last lab's implementation. The ideas I came up with either felt like they were overkill, or that they were transitioning to much into later parts of the subject. I understand, and to a certain extent agree, that the labs shouldn't be too close-ended. I would however have liked a couple of suggestions of what direction to go with, more so than in the simplified lab2 version. I ended up looking at the previous version of the file for inspiration.