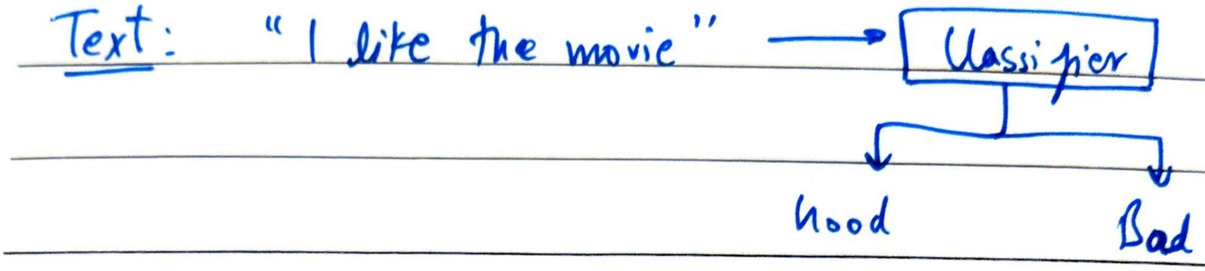
Natural Language Pre-Processing



* Classifier only understands numeric data.

* Cannot consider each word as feature wz, classifiers work with fixed no. of features (some of them) & with numeric data.

* Set up pipeline (called bag of words) to convert words into numbers.

  Numbers are **feature vectors.**

* For this we use **NLTK.**

* Corpa texts mean NLTK provides sample text data for us to work on. ~~Use : nltk.down~~

  Use : nltk.download()

  Corpus : Large collection of texts. Ex: Brown.

Brown corpus will have categories, can see by brown.categories()

Each category will have sentences related to that. brown.sents(categories="adventure")

This will give list of list & each list is a sentence containing words.

Bag of words Pipeline: Way to convert text into numeric data.

Text → Numbers → Classifier

Pipeline:

- Get the Data/ Corpus
- Tokenisation, Stopward Removal
- Stemming / Lemmatization
- Building a Vocab
- Vectorisation
- Classification

Tokenisation: Break down document.

Document → Sentences → Words

Stopward Removal: Not all words like 'was' are meaningful. So we can discard prepositions, determiners, ~~etc.~~ stop words (like a, an, me, you, his), etc.

Stemming: Convert different form of words into a bare word.

like, running, runs converted into run.

Building a common Vocabulary: List of distinct words (unique words) across all documents. All these words combined in a list, this list is called vocabulary & we can assign nos. to these words.

Use ? Suppose you have a sentence "I like ~~to~~ play cricket".

Create frequeny table for each word.

for ex: Say, I is at 50th index in vocab, then increment value of 50th index at a list of size len(vocab) by 1.

This is how you will get a vector for each sentence. This way of converting sentences into vectors is called vectorization.

len(vector) > len(sentence)

Now, these vectors can be fed to the classifier.

Order of occuring of words does not matter. That is why it is called bag of words. Because, in classifier we dont care about order, we just care about what words are occuring.

TF - IDF Normalization:

sent_1 = ["this is good movie"]

sent_2 = ["this was a good movie"]

sent_3 = ["this is not good movie"]

tf. idf

Term freq

$tf(t, d)$

term t, document d

$tf(good, sent\_1) = 1$

Inverse document freq

$$idf(t, d) = \log \frac{\overset{\text{(total documents)}}{N}}{1 + count(t, d)}$$

$$idf(good, all\ 3) = \log \frac{3}{1+3} \approx 0$$

weight = tf* idf

weight (good) = 0 ∈ [0, 1]

Hence, in sent_3 the word 'not' will have higher value.