Things to check

* Precision and Recall
* Sequence labelling
* Hype
* Word2vec/distributional semantics
* Word alignment
* Bag of words
* Morphology
* Syntax
* Semantics
* Pragmatics
* NLTK
* Stanford Parser
* spaCy
* Gensim
* Mallet
* Wordnet, BabelNet
  + Hyponym
  + Hypernym
  + Meronmys
* Coreference
* DAG-LSTM – Dynamic Acyclic Graph
* Syntax dependency trees
  + Constituency trees
* Sentiment analysis
* Treebank word tokenizer
* Stemming
  + <https://nlp.stanford.edu/IR-book/html/htmledition/stemming-and-lemmatization-1.html>
* Lemmatization
  + Porter’s stemmer
* Bag of words
  + <https://towardsdatascience.com/hacking-scikit-learns-vectorizers-9ef26a7170af>

We lose the word order!!!

* Stop words
* L2 norms
* Hashing – hash values (hashing reduces the number of features??)
* Wowpal wabbit
* F1 score

# TF – IDF (term frequency – inverse document frequency)

The weight increases as the word frequency in a document increases. (weight increases the more times a term occurs). This is offset by how many times a word appears in the entire dataset or corpus. This offset removes the importance of words such as the, a (stopwords).

TF x IDF (TF multiplied by IDF)

Log is used to dampen the effects of IDF function

**EXAMPLE**

D1 The sky is blue.  
D2 The sky is not blue.

The only word that is different between D1 and D2 is “not”. So the word “not” is now very important. The TF-IDF for “not” should reflect this in the calculations.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | TF | TF | IDF | TF-IDF | TF-IDF |
| 8 | A | B |  | A | B |
| The | 1 | 1 | Log(2/2) | 0 | 0 |
| Sky | 1 | 1 | Log(2/2) | 0 | 0 |
| Is | 1 | 1 | Log(2/2) | 0 | 0 |
| Blue | 1 | 1 | Log(2/2) | 0 | 0 |
| Not | 0 | 1 | Log(2/1) | 0 | 1x(log(2)) =0.301 |

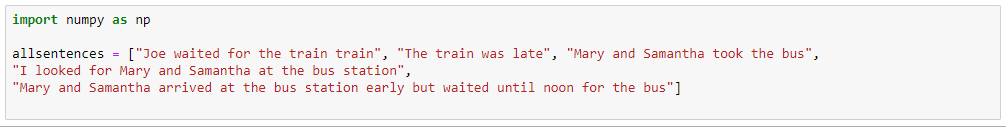
**TF-A:** How many times the word appears on document A.  
**IDF:** how many documents are there in total over how many documents contain the words in a log base.

**TF-IDF:** Multiply the TF-A x IDF column to find TF-IDF A

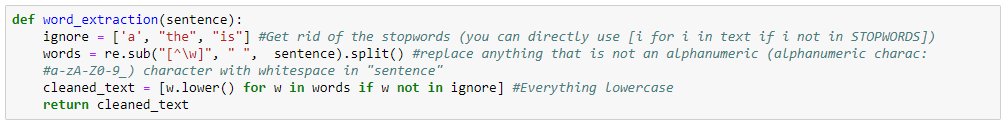
**Interpretation**: The higher the numerical weight value, the rarer the term.

The smaller the weight, the more common the term.

# Bag of Words Implementation



Pre-processing:



Tokenizer:  
  
Count Vectorizer:



Word is gets each and word from “vocab”:

---Word List for Document

['and', 'arrived', 'at', 'bus', 'but', 'early', 'for', 'i', 'joe', 'late', 'looked', 'mary', 'noon', 'samantha', 'station', 'the', 'took', 'train', 'until', 'waited', 'was']---

Then compares this with the tokenized version (output of tokenize) of every sentence – “sentences”:

['joe', 'waited', 'for', 'train', 'train']

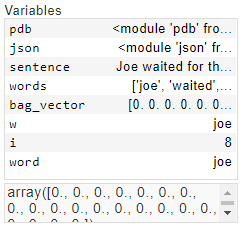
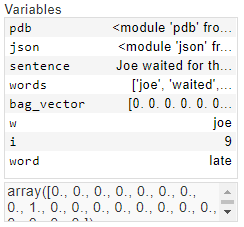
['the', 'train', 'was', 'late']

['mary', 'and', 'samantha', 'took', 'bus']

['i', 'looked', 'for', 'mary', 'and', 'samantha', 'at', 'bus', 'statio']

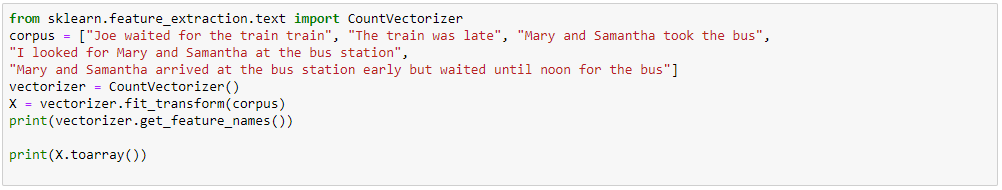
['mary', 'and', 'samantha', 'arrived', 'at', 'bus', 'station', 'early', 'but', 'waited', 'until', 'noon', 'for', 'bus']

In essence we are trying to find each and every words (that are in “vocab) index from “sentences”

🡪 🡪 🡪  
 Figure -1 Figure -2

Match is made in Figure -1 (both w and word is joe), and joe’s index in vocab is 8 (8+1) will be increased by one. Check array on Figure-2

# SK Learn Library of Count Vectorizer



Word2vec

Convolutional filter -1D convolutions

Maximum pooling over time