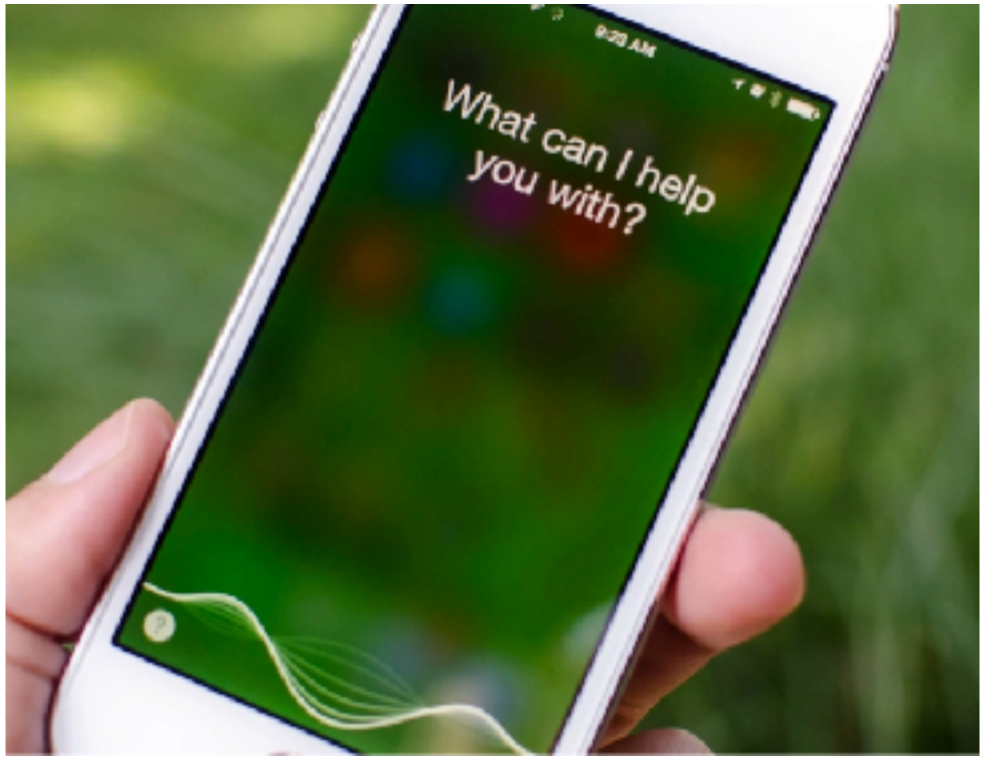
Natural Language Processing with Python

by: Sarah Nooravi

Examples


What can I help you with?


Sent Mail
Spam (372)
Trash


SEARCH
ENGINE
RANKING


SENTIMENT
100 %
50%
0 %

Libraries

- Gensim
- Scikit-Learn

Algorithms

- Word2Vec
- TF-IDF (Term Frequency - Inverse Document Frequency)
- LDA (Latent Dirichlet Allocation) - Supervised
- HDP (Hierarchical Dirichlet Process) - Unsupervised
- many others…

TF-IDF (Term Frequency - Inverse Document Frequency)


CAUTION
MATH
AHEAD

What is TF-IDF?

TF-IDF stands for "Term Frequency, Inverse Document Frequency". It is a way to score the importance of words (or "terms") in a document based on how frequently they appear across multiple documents.

Intuitively...

- **If a word appears frequently in a document, it's important. Give the word a high score.**
- **But if a word appears in many documents, it's not a unique identifier. Give the word a low score.**

Therefore, common words like "the" and "for", which appear in many documents, will be scaled down. Words that appear frequently in a *single* document will be scaled up.

For instance, 83% of text-based recommender systems in the domain of digital libraries use tf-idf

http://blog.christianperone.com/2013/09/machine-learning-cosine-similarity-for-vector-space-models-part-iii/

Motivation

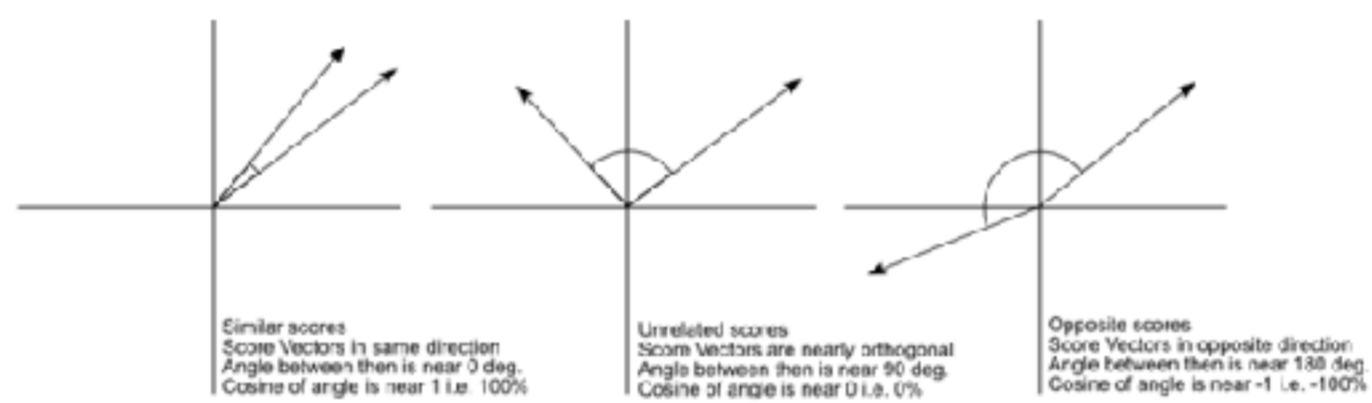
"… we're living in a world where big data is growing at a rate equivalent to a company the size of Google being created every day. It's staggering. And it's only going to get bigger. Many data analysts are suggesting the digital universe will be 40 times bigger by 2020! This is largely due to the vast increase of dark data, meaning all the unstructured data from the Internet, social media, voice and information from connected devices."

Jeremy Waite, Marketing Evangelist, Watson Marketing EMEA

Motivation

- Helps to draw insights from freely written text (ex. emails)
- Reasons why we might want to implement TF-IDF:
    - Categorize documents
    - To find the similarity between documents (i.e. cosine similarity)
    - To extract keywords
    - Stop-word filtering

$$\cos\theta = \frac{\vec{a} \cdot \vec{b}}{\|\vec{a}\| \|\vec{b}\|}$$



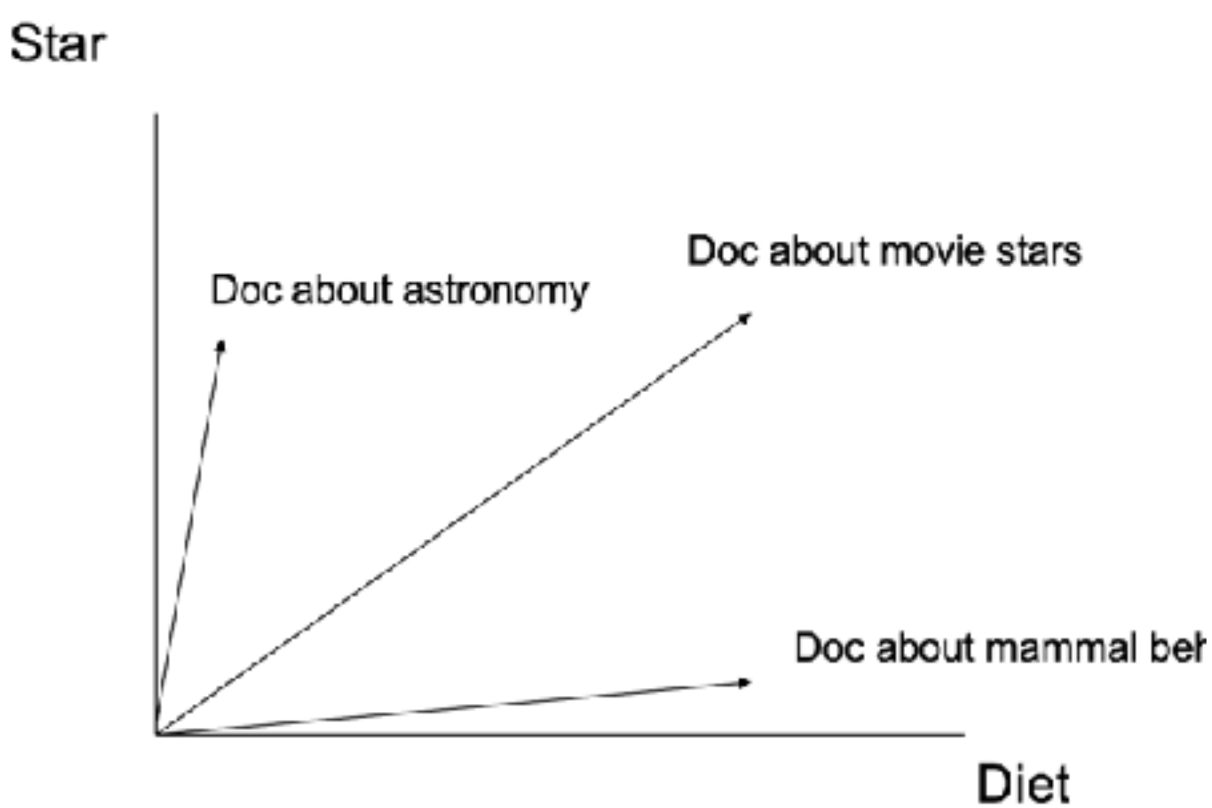
The Cosine Similarity values for different documents, 1 (same direction), 0 (90 deg.), -1 (opposite directions).

http://blog.christianperone.com/2013/09/machine-learning-cosine-similarity-for-vector-space-models-part-iii/

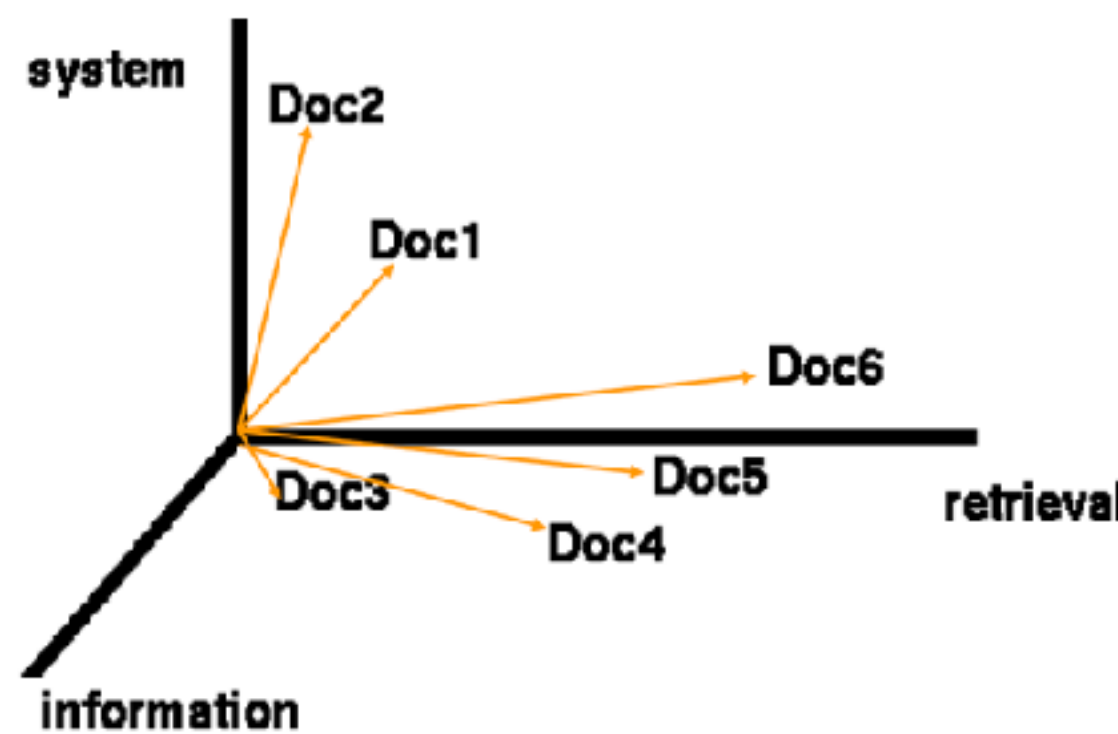
How it works

- Documents are represented as vectors in term space.
- Primary assumption of the Vector Space Model: Documents that are close together in space are similar in meaning.

2-D Space



3-D Space



How it works

- TF-IDF computes a weight which represents the importance of a term inside a document. It does this by comparing the frequency of usage inside an individual document as opposed to the entire dataset (a collection of documents).

Each row is a document

$$\begin{array}{c|cccc} & T_1 & T_2 & \cdots & T_t \\ \hline D_1 & w_{11} & w_{21} & \cdots & w_{t1} \\ D_2 & w_{12} & w_{22} & \cdots & w_{t2} \\ \vdots & \vdots & \vdots & & \vdots \\ \vdots & \vdots & \vdots & & \vdots \\ D_n & w_{1n} & w_{2n} & \cdots & w_{tn} \end{array}$$

Each column is a term

Each term is a feature

https://blogs.msdn.microsoft.com/spt/2008/03/05/information-retrieval-amp-search-basic-ir-models/

Step 1: Turning Text into Numbers

Each word is a number under the hood

Example:

Document 1: I like Facebook

Document 2: I updated Facebook

Document 3: Don't use Facebook

Turning Text into Numbers as **Frequency Matrix**

Pool of unique words: "I", "like", "Facebook", "updated", "Don't", "use" (features)

	I	like	facebook	updated	Don’t	use
Doc 1	1	1	1	0	0	0
Doc 2	1	0	1	1	0	0
Doc 3	0	0	1	0	1	1

Order of words don't matter. We only focus on frequency of words

Step 1 Frequency Matrix

TF (Term Frequency, Part 1): measures how frequently a term occurs in a document. Since every document is different in length, it is possible that a term would appear more often in a long document than a shorter one. Thus, the term frequency is often divided by the document length as a way of normalization.

Step 2: How to calculate weights?

TF (Term Frequency, Part 1): measures how frequently a term occurs in a document. Since every document is different in length, it is possible that a term would appear more often in a long document than a shorter one. Thus, the term frequency is often divided by the document length as a way of normalization.

Variants of term frequency (TF) weight

weighting scheme	TF weight
binary	$0, 1$
raw count	$f_{t,d}$
term frequency	$f_{t,d} \Big/ \sum_{t' \in d} f_{t',d}$
log normalization	$1 + \log(f_{t,d})$
double normalization 0.5	$0.5 + 0.5 \cdot \frac{f_{t,d}}{\max_{\{t' \in d\}} f_{t',d}}$
double normalization K	$K + (1 - K)\frac{f_{t,d}}{\max_{\{t' \in d\}} f_{t',d}}$

Step 2: How to calculate weights?

IDF (Inverse Document Frequency, Part 2): measure of how unique a word is i.e. how infrequently the word occurs across all documents.

Step 2: How to calculate weights?

IDF (Inverse Document Frequency, Part 2): measure of how unique a word is i.e. how infrequently the word occurs across all documents.

Variants of inverse document frequency (IDF) weight

weighting scheme	IDF weight ($n_t = \|\{d \in D : t \in d\}\|$)
unary	1
inverse document frequency	$\log \frac{N}{n_t} = -\log \frac{n_t}{N}$
inverse document frequency smooth	$\log \left(1 + \frac{N}{n_t}\right)$
inverse document frequency max	$\log \left(\frac{\max_{\{t' \in d\}} n_{t'}}{1 + n_t}\right)$
probabilistic inverse document frequency	$\log \frac{N - n_t}{n_t}$

Step 2: How to calculate weights?

- IDF provides high values for rare words and low values for common words

For a collection of 10000 documents (N = 10000)

$$\log\left(\frac{10000}{10000}\right) = 0$$

$$\log\left(\frac{10000}{5000}\right) = 0.301$$

$$\log\left(\frac{10000}{20}\right) = 2.698$$

$$\log\left(\frac{10000}{1}\right) = 4$$

Putting it all together

$$w_{ik} = tf_{ik} * \log(N / n_k)$$

T_k = term k in document D_i

tf_{ik} = frequency of term T_k in document D_i

idf_k = inverse document frequency of term T_k in C

N = total number of documents in the collection C

n_k = the number of documents in C that contain T_k

$$idf_k = \log\left(\frac{N}{n_k}\right)$$

Example

Example: Tagging Blog Posts

Step 1: Generate Scores for Each Document

Let's say you have a 100 word blog post with the word "JavaScript" in it 5 times. The calculation for the Term Frequency would be:

Next, assume your entire collection of blog posts has 10,000 documents and the word "JavaScript" appears at least once in 100 of these. The Inverse Document Frequency calculation would look like this:

To calculate the TF-IDF, we multiply the previous two values. This gives us the final score:

Example

Example: Tagging Blog Posts

Step 1: Generate Scores for Each Document

Let's say you have a 100 word blog post with the word "JavaScript" in it 5 times. The calculation for the Term Frequency would be:

```
TF = 5/100 = 0.05
```

Next, assume your entire collection of blog posts has 10,000 documents and the word "JavaScript" appears at least once in 100 of these. The Inverse Document Frequency calculation would look like this:

[redacted]

To calculate the TF-IDF, we multiply the previous two values. This gives us the final score:

[redacted]

Example

Example: Tagging Blog Posts

Step 1: Generate Scores for Each Document

Let's say you have a 100 word blog post with the word "JavaScript" in it 5 times. The calculation for the Term Frequency would be:

```
TF = 5/100 = 0.05
```

Next, assume your entire collection of blog posts has 10,000 documents and the word "JavaScript" appears at least once in 100 of these. The Inverse Document Frequency calculation would look like this:

```
IDF = log(10,000/100) = 2
```

To calculate the TF-IDF, we multiply the previous two values. This gives us the final score:

[redacted]

Example

Example: Tagging Blog Posts

Step 1: Generate Scores for Each Document

Let's say you have a 100 word blog post with the word "JavaScript" in it 5 times. The calculation for the Term Frequency would be:

```
TF = 5/100 = 0.05
```

Next, assume your entire collection of blog posts has 10,000 documents and the word "JavaScript" appears at least once in 100 of these. The Inverse Document Frequency calculation would look like this:

```
IDF = log(10,000/100) = 2
```

To calculate the TF-IDF, we multiply the previous two values. This gives us the final score:

```
TF-IDF = 0.05 * 2 = 0.1
```

Implementation in Scikit Learn

Models

- Import
- Instantiate
- Fit
- Predict

Vectorizers

- Import
- Instantiate
- Fit
- Transform

Implementation of Vectorization in Scikit Learn

Import and Instantiate

```
# import and instantiate CountVectorizer (with the default parameters)
from sklearn.feature_extraction.text import CountVectorizer
vect = CountVectorizer()
```

Fit

```
# learn the 'vocabulary' of the training data (occurs in-place)
vect.fit(simple_train)
```

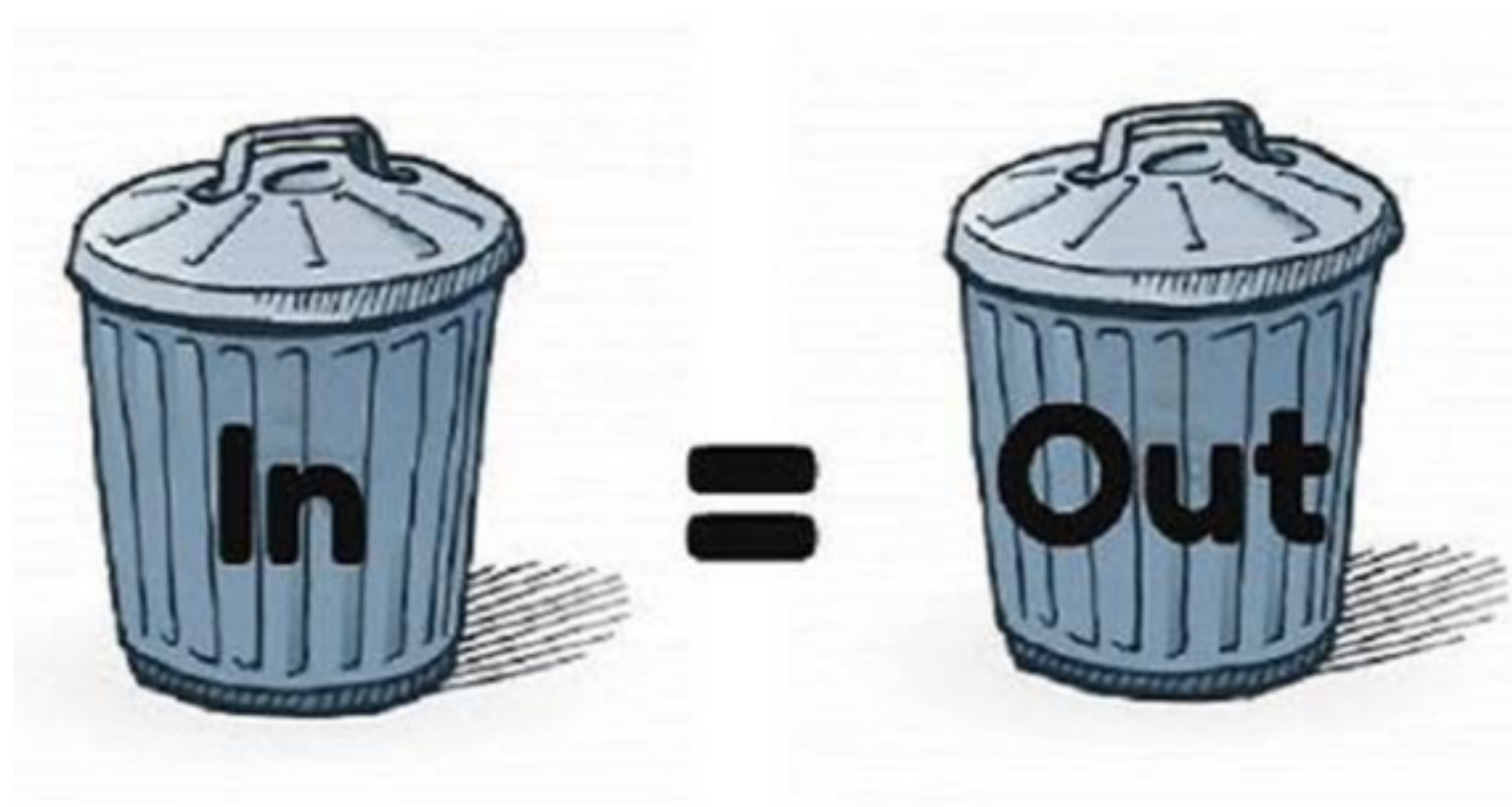
Transform

```
# transform training data into a 'document-term matrix'
simple_train_dtm = vect.transform(simple_train)
simple_train_dtm
```

Pretty Easy…huh?

Pre-Processing

- Stop words
- N-grams
- Punctuation
- Parts of speech
- Short text
- Be a little smart with the data you have



Tuning Parameters - (Pre-Processing)

lowercase : boolean, default True

Convert all characters to lowercase before tokenizing.

stop_words : string {'english'}, list, or None (default)

If a string, it is passed to _check_stop_list and the appropriate stop list is returned. 'english' is currently the only supported string value.

If a list, that list is assumed to contain stop words, all of which will be removed from the resulting tokens. Only applies if `analyzer == 'word'`.

If None, no stop words will be used. max_df can be set to a value in the range [0.7, 1.0) to automatically detect and filter stop words based on intra corpus document frequency of terms.

http://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html

Tuning Parameters

analyzer : string, {'word', 'char'} or callable

Whether the feature should be made of word or character n-grams.

If a callable is passed it is used to extract the sequence of features out of the raw, unprocessed input.

ngram_range : tuple (min_n, max_n)

The lower and upper boundary of the range of n-values for different n-grams to be extracted. All values of n such that min_n <= n <= max_n will be used.

max_df : float in range [0.0, 1.0] or int, default=1.0

When building the vocabulary ignore terms that have a document frequency strictly higher than the given threshold (corpus-specific stop words). If float, the parameter represents a proportion of documents, integer absolute counts. This parameter is ignored if vocabulary is not None.

min_df : float in range [0.0, 1.0] or int, default=1

When building the vocabulary ignore terms that have a document frequency strictly lower than the given threshold. This value is also called cut-off in the literature. If float, the parameter represents a proportion of documents, integer absolute counts. This parameter is ignored if vocabulary is not None.

max_features : int or None, default=None

If not None, build a vocabulary that only consider the top max_features ordered by term frequency across the corpus.

This parameter is ignored if vocabulary is not None.

L1/L2 Norm: https://stackoverflow.com/questions/35114292/tfidvectorizer-l2-normalized-vector

http://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html

How does it do this? Part Two

The most frequent words are not the most descriptive.

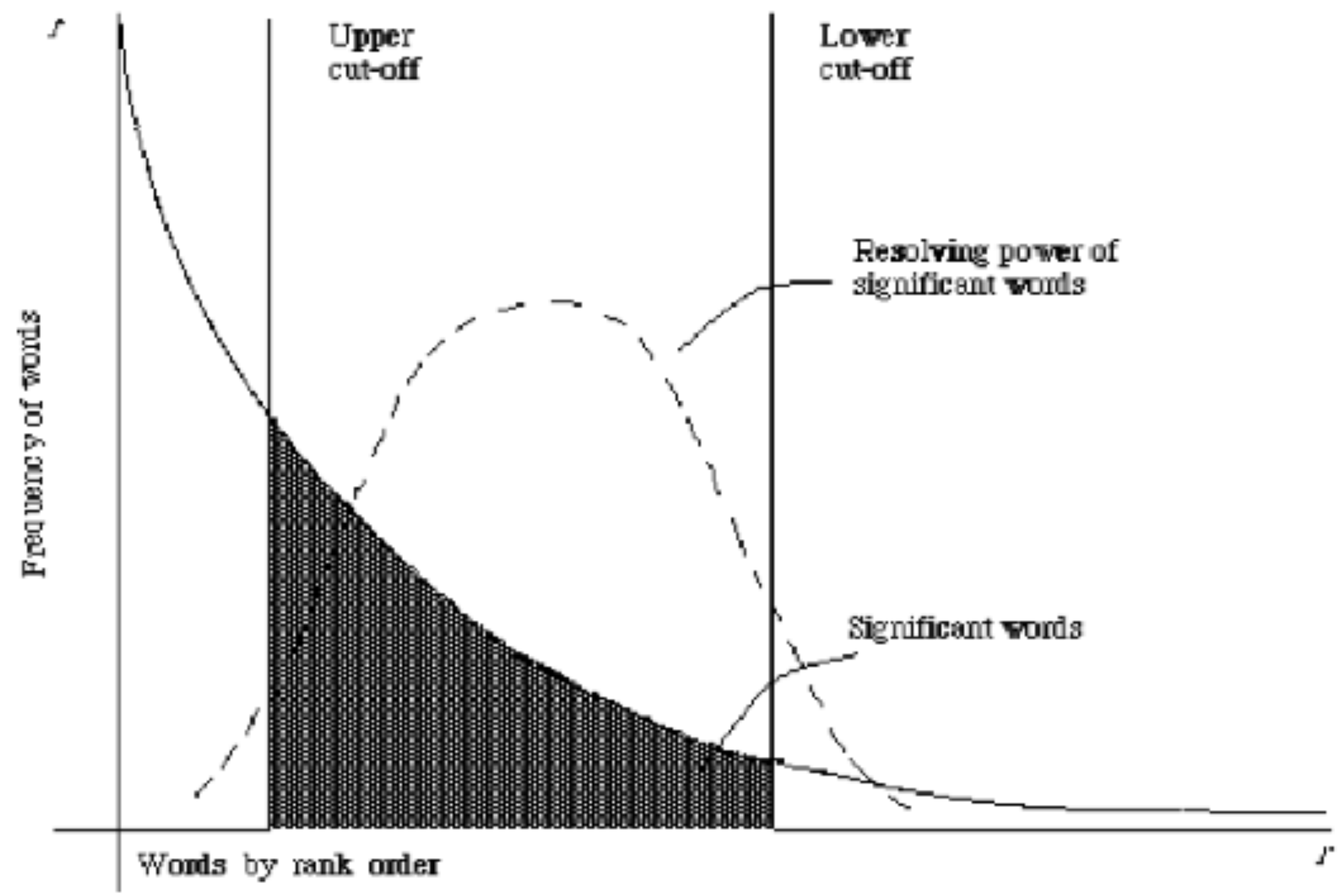


Figure 2.1. A plot of the hyperbolic curve relating f, the frequency of occurrence and r, the rank order (Adaped from Schultz[44] page 120)

(from van Rijsbergen 79)

Practice

1. Go to: https://github.com/snooravi/meetups
2. Clone/download the directory to your local computer
3. Open jupyter notebook (run ‘jupyter notebook’ in CLI)
4. Navigate to: tf_idf_example/NLP on Enron Data.ipynb

More NLP Resources

1. Go to: https://github.com/justmarkham/pycon-2016-tutorial
2. Clone/download the directory to your local computer
3. Open jupyter notebook (run ‘jupyter notebook’ in CLI)
4. Navigate to: pycon-2016-tutorial/tutorial.ipynb
5. Related Video: here