greatlearning
Learning for Life

Statistical NLP

Agenda

- Text Analytics Frameworks
- Stemming
- Lemmatization
- Named Entity Resolution
- POS Tagging
- TF-IDF
- Language Detection
- Case studies
  - Documents Similarity
  - Documents Clustering
  - Yelp Review Sentiment Analysis
  - News Headlines Analysis
- References
- Possible Capstone Projects

Text Analytics Frameworks

- NLTK: The Natural Language Toolkit is a complete platform that contains more than 50 corpora and lexical resources. It also provides the necessary tools, interfaces, and methods to process and analyse text data
- Pattern: pattern provides tools and interfaces for web mining, information retrieval, NLP, machine learning and network analysis. The pattern.en module contains most of the utilities for text analytics
- Gensim: The genism library has a rich set of capabilities for semantic analysis, including topic modelling and similarity analysis. But the best part is that it contains a Python port of Google's popular word2vec model, a neural network model implemented to learn distributed representations of words where similar words(semantic) occur close to each other
- Textblob: Provides several capabilities including text processing, phrase extraction, classification, POS tagging, text translation and sentiment analysis
- Spacy: Claims to provide industrial-strength NLP capabilities by providing the best implementation of each technique and algorithm, making NLP tasks efficient in terms of performance and implementation
- Sklearn

Text preprocessing and wrangling

Objective

Transform unstructured or semi-structured noisy text data into clean text data after removing unnecessary elements

Techniques:

- **Remove HTML and markup tags**
- **Handling accented characters**
- **Expanding contractions**
- **Removing special characters & symbols**
- **Stemming**
- **Lemmatization**
- **Removing Stopwords**

Text preprocessing – removing html tags

Often, unstructured text contains a lot of noise, especially if you use techniques like web or screen scraping. HTML tags are typically one of these components which don't add much value towards understanding and analyzing text.


```
strip_html_tags('<html><h2>Some important text</h2></html>')
```

'Some important text'

Text preprocessing – handling accented characters

Usually in any text corpus, you might be dealing with accented characters/letters, especially if you only want to analyze the English language.

Hence, we need to make sure that these characters are converted and standardized into ASCII characters.

A simple example—converting **é** to **e**.

```
remove_accented_chars('Sómě Áccěntěd těxt')
```

'Some Accented text'

Text preprocessing – Expanding Contractions

Contractions are shortened version of words or syllables. They often exist in either written or spoken forms in the English language. These shortened versions or contractions of words are created by removing specific letters and sounds.

In case of English contractions, they are often created by removing one of the vowels from the word. Examples would be, ***do not*** to ***don't*** and ***I would*** to ***I'd***.

expand_contractions("Y'all can't expand contractions I'd think")

'You all cannot expand contractions I would think'

Text preprocessing – Removing special characters

Special characters and symbols are usually non-alphanumeric characters or even occasionally numeric characters (depending on the problem), which add to the extra noise in unstructured text.

Usually, simple regular expressions (regexes) can be used to remove them.

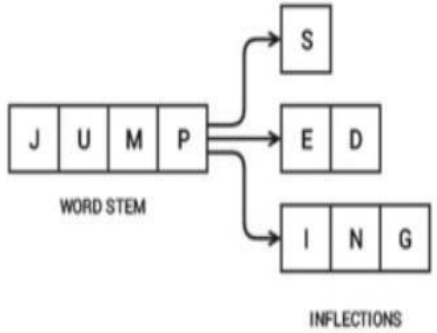
```
remove_special_characters("Well this was fun! What do you
                          think? 123#@!",
                          remove_digits=True)
```


'Well this was fun What do you think '

Text preprocessing – stemming

To understand stemming, you need to gain some perspective on what word stems represent. Word stems are also known as the ***base form*** of a word, and we can create new words by attaching affixes to them in a process known as inflection.

Consider the word ***JUMP***. You can add affixes to it and form new words like ***JUMPS***, ***JUMPED***, and ***JUMPING***. In this case, the base word ***JUMP*** is the word stem.



Word stem and its inflections (Source: Text Analytics with Python, Apress/Springer 2016)

Text preprocessing – stemming

The reverse process of obtaining the base form of a word from its inflected form is known as ***stemming***. (May not be a dictionary\lexicographically correct word)

The Porter stemmer is based on the algorithm developed by its inventor, Dr. Martin Porter. It is said to have had a total of five different phases for reduction of inflections to their stems, where each phase has its own set of rules.

simple_stemmer("My system keeps crashing his crashed yesterday, ours crashes daily")

'My system keep crash hi crash yesterday, our crash daili'

Text preprocessing – lemmatization

Lemmatization is very similar to stemming, where we remove word affixes to get to the base form of a word. However, the base form in this case is known as the root word, but not the root stem.

The difference being that the ***root word is always a lexicographically correct word*** (present in the dictionary), but the root stem may not be so. Thus, root word, also known as the ***lemma***, will always be present in the dictionary.

```
lemmatize_text("My system keeps crashing his
                crashed yesterday, ours crashes daily")
```


'My system keep crash ! his crash yesterday , ours crash daily'

Text preprocessing – removing stopwords

Words which have little or no significance, especially when constructing meaningful features from text, are known as stopwords or stop words.

These are usually words that end up having the maximum frequency if you do a simple term or word frequency in a corpus.

Typically, these can be articles, conjunctions, prepositions and so on. Some examples of stopwords are ***a***, ***an***, ***the***, ***and*** the like.

remove_stopwords("The, and, if are stopwords, computer is not")

', , stopwords , computer'

Beatiful Soup

Library for pulling data out of HTML and XML Files

Beautiful Soup is a Python library designed for quick turnaround projects like screen-scraping. Three features make it powerful:

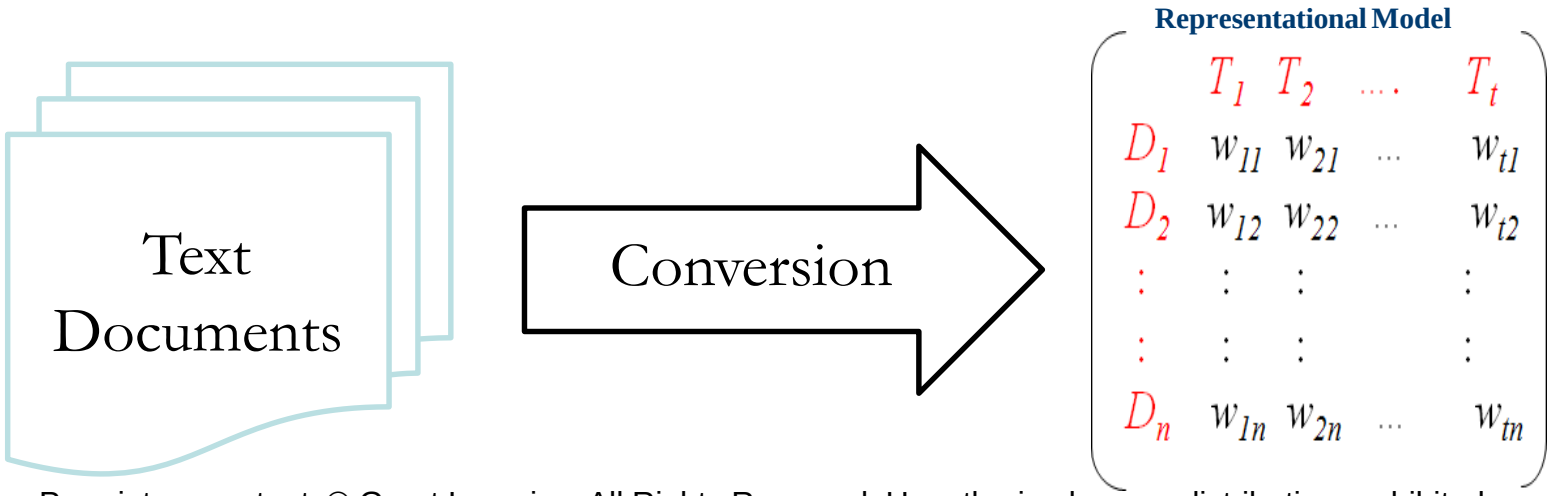
- Beautiful Soup provides a few simple methods and Pythonic idioms for navigating, searching, and modifying a parse tree: a toolkit for dissecting a document and extracting what you need. It doesn't take much code to write an application
- Beautiful Soup sits on top of popular Python parsers like lxml and html5lib, allowing you to try out different parsing strategies or trade speed for flexibility.

Text Preprocessing HandsOn

Hands-on: nlpEDA_v1.ipynb

Text representation model

- Machine Learning models at heart are mathematical functions and cannot understand unstructured text
- Hence we need to convert text into some numeric representations which can be understood by machines
- In information retrieval and text mining, text data of different formats can be represented in a *common representation model*, popularly known as Vector Space Model
- Unstructured text data is converted to the model representation giving structured data representations using feature engineering models (e.g. Bag of Words)



Bag of words

- This is perhaps the most simple vector space representational model for unstructured text.

- A vector space model is simply a mathematical model to represent unstructured text (or any other data) as numeric vectors, such that each dimension of the vector is a specific feature\attribute.

- The bag of words model represents each text document as a numeric vector where each dimension is a specific word from the corpus and the value could be its frequency in the document, occurrence (denoted by 1 or 0) or even weighted values.

- The model's name is such because each document is represented literally as a 'bag' of its own words, disregarding word orders, sequences and grammar.

Bag of words

greatlearning
Learning for Life

	Document
0	The sky is blue and beautiful.
1	Love this blue and beautiful sky!
2	The quick brown fox jumps over the lazy dog.
3	A king's breakfast has sausages, ham, bacon, eggs, toast and beans
4	I love green eggs, ham, sausages and bacon!
5	The brown fox is quick and the blue dog is lazy!
6	The sky is very blue and the sky is very beautiful today
7	The dog is lazy but the brown fox is quick!

	bacon	beans	beautiful	blue	breakfast	brown	dog	eggs	fox	green	ham	jumps	kings	lazy	love	quick	sausages	sky	toast	today
0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0
2	0	0	0	0	0	1	1	0	1	0	0	1	0	1	0	1	0	0	0	0
3	1	1	0	0	1	0	0	1	0	0	1	0	1	0	0	0	1	0	1	0
4	1	0	0	0	0	0	0	1	0	1	1	0	0	0	1	0	1	0	0	0
5	0	0	0	1	0	1	1	0	1	0	0	0	0	1	0	1	0	0	0	0
6	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	1
7	0	0	0	0	0	1	1	0	1	0	0	0	0	1	0	1	0	0	0	0

Bag of N-grams

- A word is just a single token, often known as a unigram or 1-gram. And the Bag of Words model doesn't consider order of words.

- What if we also wanted to take into account phrases or collection of words which occur in a sequence? N-grams help us achieve that.

- An N-gram is basically a collection of word tokens from a text document such that these tokens are contiguous and occur in a sequence. Bi-grams indicate n-grams of order 2 (two words), Tri-grams indicate n-grams of order 3 (three words), and so on.

- The Bag of N-Grams model is hence just an extension of the Bag of Words model so we can also leverage N-gram based features.

Bag of N-grams

greatlearning
Learning for Life

	Document
0	The sky is blue and beautiful.
1	Love this blue and beautiful sky!
2	The quick brown fox jumps over the lazy dog.
3	A king's breakfast has sausages, ham, bacon, eggs, toast and beans
4	I love green eggs, ham, sausages and bacon!
5	The brown fox is quick and the blue dog is lazy!
6	The sky is very blue and the sky is very beautiful today
7	The dog is lazy but the brown fox is quick!

	bacon eggs	beautiful sky	beautiful today	blue beautiful	blue dog	blue sky	breakfast sausages	brown fox	dog lazy	eggs ham	...	lazy dog	love blue	love green	quick blue	quick brown	sausages bacon	sausages ham	sky beautiful
0	0	0	0	1	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0
1	0	1	0	1	0	0	0	0	0	0	...	0	1	0	0	0	0	0	0
2	0	0	0	0	0	0	0	1	0	0	...	1	0	0	0	1	0	0	0
3	1	0	0	0	0	0	1	0	0	0	...	0	0	0	0	0	0	1	0
4	0	0	0	0	0	0	0	0	0	1	...	0	0	1	0	0	1	0	0
5	0	0	0	0	1	0	0	1	1	0	...	0	0	0	1	0	0	0	0
6	0	0	1	0	0	1	0	0	0	0	...	0	0	0	0	0	0	0	1
7	0	0	0	0	0	0	0	1	1	0	...	0	0	0	0	0	0	0	0

Tf-idf model

- There are some potential problems which might arise with the Bag of Words model when it is used on large corpora.
- Since the feature vectors are based on absolute term frequencies, there might be some terms which occur frequently across all documents and these may tend to overshadow other terms in the feature set.
- The TF-IDF model tries to combat this issue by using a scaling or normalizing factor in its computation.
- TF-IDF stands for Term Frequency-Inverse Document Frequency, which uses a combination of two metrics in its computation, namely: ***term frequency (tf)*** and ***inverse document frequency (idf)***.
- This technique was developed for ranking results for queries in search engines and now it is an indispensable model in the world of information retrieval and NLP.

Word Frequencies with TFIDFVectorizer

1. Term Frequency: This summarizes how often a given word appears within a document.
2. Inverse Document Frequency: This downscales words that appear a lot across documents.
3. Without going into the math, TF-IDF scores are word frequency scores that try to highlight words that are more interesting, e.g. frequent in a document but not across documents.

BAG Of Words / Vector Space model

1. **TF:** Term Frequency, which measures how frequently a term occurs in a document.
    a. Since every document is different in length, it is possible that a term would appear much more times in long documents than shorter ones.
    b. Thus, the term frequency is often divided by the document length (aka. the total number of terms in the document) as a way of normalization:
    c. TF(t) = (Number of times term t appears in a document) / (Total number of terms
    in the document)
2. **IDF:** Inverse Document Frequency measures how important a term is.
    a. Certain terms, may appear a lot of times across documents but have little importance.
    b. Weigh down the frequent terms while scale up the rare ones, by computing the following:
        a. IDF(t) = log_e(Total number of documents / Number of documents with term t in it).

Source: http://www.tfidf.com/

BAG Of Words / Vector Space model

$$w_{i,j} = \mathrm{tf}_{i,j} \cdot \log \frac{N}{n_i}$$

$w_{i,j}$ **weight assigned to term *i* in document *j***

$\mathrm{tf}_{i,j}$ **number of occurrence of term *i* in document *j***

N **number of documents in entire collection**

n_i **number of documents with term *i***

- Consider a document containing 1000 words wherein the word ML appears 30 times. The term frequency (i.e., tf) for ML is then (30 / 1000) = 0.03. Now, assume we have 10 million documents and the word ML appears in one thousand of these. Then, the inverse document frequency (i.e., idf) is calculated as log(10,000,000 / 1,000) = 4. Thus, the Tf-idf weight is the product of these quantities: 0.03 * 4 = 0.12.
- This is the coefficient of the given document

Tf-idf model

	Document
0	The sky is blue and beautiful.
1	Love this blue and beautiful sky!
2	The quick brown fox jumps over the lazy dog.
3	A king's breakfast has sausages, ham, bacon, eggs, toast and beans
4	I love green eggs, ham, sausages and bacon!
5	The brown fox is quick and the blue dog is lazy!
6	The sky is very blue and the sky is very beautiful today
7	The dog is lazy but the brown fox is quick!

	bacon	beans	beautiful	blue	breakfast	brown	dog	eggs	fox	green	ham	jumps	kings	lazy	love	quick	sausages	sky	toast	today
0	0.00	0.00	0.60	0.53	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.60	0.00	0.0
1	0.00	0.00	0.49	0.43	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.57	0.00	0.00	0.49	0.00	0.0
2	0.00	0.00	0.00	0.00	0.00	0.38	0.38	0.00	0.38	0.00	0.00	0.53	0.00	0.38	0.00	0.38	0.00	0.00	0.00	0.0
3	0.32	0.38	0.00	0.00	0.38	0.00	0.00	0.32	0.00	0.00	0.32	0.00	0.38	0.00	0.00	0.00	0.32	0.00	0.38	0.0
4	0.39	0.00	0.00	0.00	0.00	0.00	0.00	0.39	0.00	0.47	0.39	0.00	0.00	0.00	0.39	0.00	0.39	0.00	0.00	0.0
5	0.00	0.00	0.00	0.37	0.00	0.42	0.42	0.00	0.42	0.00	0.00	0.00	0.00	0.42	0.00	0.42	0.00	0.00	0.00	0.0
6	0.00	0.00	0.36	0.32	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.72	0.00	0.5
7	0.00	0.00	0.00	0.00	0.00	0.45	0.45	0.00	0.45	0.00	0.00	0.00	0.00	0.45	0.00	0.45	0.00	0.00	0.00	0.0

Word Frequencies with TFIDFVectorizer

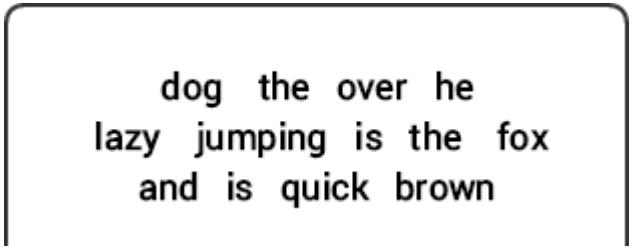
Hands-on: tfIdf_Example_v1.ipynb

Understanding Language Syntax and Structure

- For any language, syntax and structure usually go hand in hand, where a set of specific rules, conventions, and principles govern the way words are combined into phrases; phrases get combines into clauses; and clauses get combined into sentences.

- We will be talking specifically about the English language syntax and structure in this section. In English, words usually combine together to form other constituent units. These constituents include words, phrases, clauses, and sentences.

- Considering a sentence, ***"The brown fox is quick and he is jumping over the lazy dog"***, it is made of a bunch of words and just looking at the words by themselves don't tell us much.



Understanding Language Syntax and Structure

- Knowledge about the structure and syntax of language is helpful in many areas like text processing, annotation, and parsing for further operations such as text classification or summarization.

- Typical parsing techniques for understanding text syntax are mentioned below.
  - **Parts of Speech (POS) Tagging**
  - **Shallow Parsing or Chunking**

Parts of Speech Tagging

- Parts of Speech(POS) are specific lexical categories to which words are assigned, based on their syntactic context and role. Usually, words can fall into one of the following major categories:
  - Noun
  - Verb
  - Adjective
  - Adverb

Shallow Parsing or Chuncking

- Shallow parsing, also known as light parsing or chunking , is a popular natural language processing technique of analyzing the structure of a sentence to break it down into its smallest constituents (which are tokens such as words) and group them together into higher-level phrases. This includes POS tags as well as phrases from a sentence.

Named Entity Recognition

- In any text document, there are particular terms that represent specific entities that are more informative and have a unique context.

- These entities are known as named entities , which more specifically refer to terms that represent real-world objects like people, places, organizations, and so on, which are often denoted by proper names.

- A naive approach could be to find these by looking at the noun phrases in text documents.

- Named entity recognition (NER) , also known as entity chunking/ extraction , is a popular technique used in information extraction to identify and segment the named entities and classify or categorize them under various predefined classes.

Named Entity Recognition

- NER
  - Classifies a text into predefined categories or real world object entities
  - Takes a string of text(sentence or paragraph) as input and identifies relevant nouns(people, places, and organizations) that are mentioned in that string
- Uses
  - Classify or categorize contents by getting the relevant tags
  - Improve search algorithms
  - For content recommendations
  - For info extraction
- Ref:
  - https://spacy.io/api/annotation#named-entities

NER HandsOn

Hands-on: nlpEDA_v1.ipynb

greatlearning
Learning for Life

Case Studies

Document Similarity

- Document similarity is the process of using a distance or similarity based metric that can be used to identify how similar a text document is with any other document(s) based on features extracted from the documents like bag of words or tf-idf.

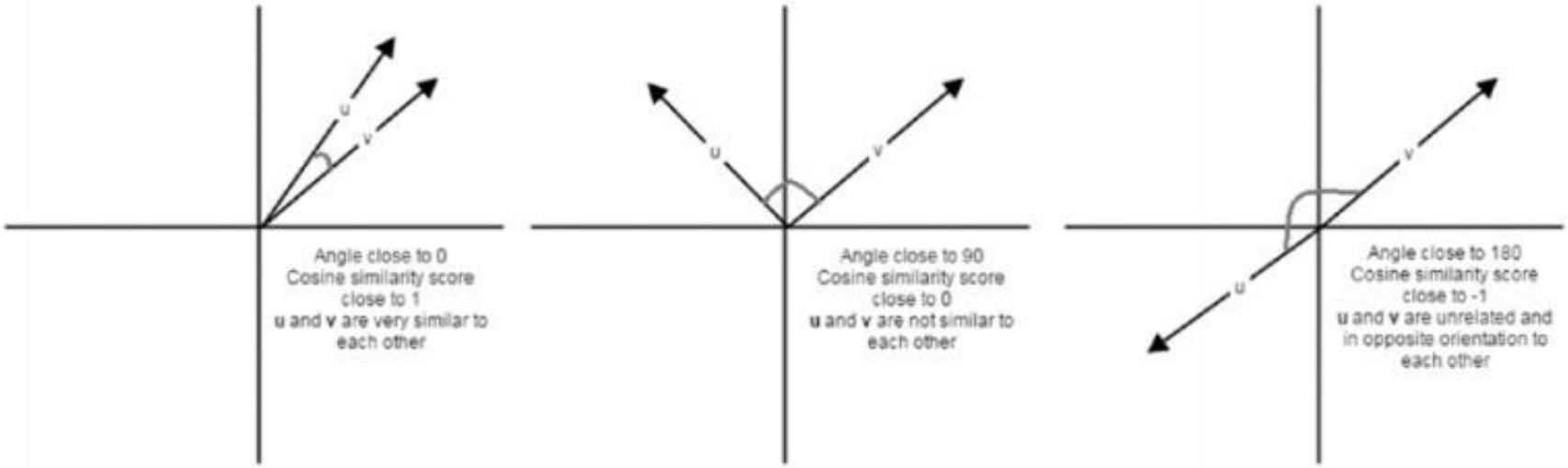
- Pairwise document similarity in a corpus involves computing document similarity for each pair of documents in a corpus.

- Thus if you have ***C*** documents in a corpus, you would end up with a ***C*** **x** ***C*** matrix such that each row and column represents the similarity score for a pair of documents, which represent the indices at the row and column, respectively.

- There are several similarity and distance metrics that are used to compute document similarity. These include cosine distance/similarity, euclidean distance, manhattan distance, BM25

Document Similarity

- Cosine similarity basically gives us a metric representing the cosine of the angle between the feature vector representations of two text documents. Lower the angle between the documents, the closer and more similar they are as depicted in the following figure.



Similarity features

greatlearning
Learning for Life

	Document
0	The sky is blue and beautiful.
1	Love this blue and beautiful sky!
2	The quick brown fox jumps over the lazy dog.
3	A king's breakfast has sausages, ham, bacon, eggs, toast and beans
4	I love green eggs, ham, sausages and bacon!
5	The brown fox is quick and the blue dog is lazy!
6	The sky is very blue and the sky is very beautiful today
7	The dog is lazy but the brown fox is quick!

	0	1	2	3	4	5	6	7
0	1.000000	0.820599	0.000000	0.000000	0.000000	0.192353	0.817246	0.000000
1	0.820599	1.000000	0.000000	0.000000	0.225489	0.157845	0.670631	0.000000
2	0.000000	0.000000	1.000000	0.000000	0.000000	0.791821	0.000000	0.850516
3	0.000000	0.000000	0.000000	1.000000	0.506866	0.000000	0.000000	0.000000
4	0.000000	0.225489	0.000000	0.506866	1.000000	0.000000	0.000000	0.000000
5	0.192353	0.157845	0.791821	0.000000	0.000000	1.000000	0.115488	0.930989
6	0.817246	0.670631	0.000000	0.000000	0.000000	0.115488	1.000000	0.000000
7	0.000000	0.000000	0.850516	0.000000	0.000000	0.930989	0.000000	1.000000

Document Similarity

DocumentSimilarityClustering_v1.ipynb

Document Clustering

DocumentSimilarityClustering_v1.ipynb

Yelp Reviews Sentiment Analysis and Classification: Hands-On

textClassficationWithAdvancedML.ipynb

News HeadLines Analysis: Hands-On

newsHeadlinesAnalysis.ipynb

References

- Text Analytics with Python by Dipanjan Sarkar
- Natural Language Processing by Steven Bird, Ewan Klein and Edward Loper
- Practical Machine Learning with Python by Dipanjan Sarkar

Possible Capstone Projects

- Information retrieval: Find relevant results and similar results
- Information extraction: Structured information from unstructured documents
- Machine translation: One language to another
- Text simplification: Preserve the meaning of text, but simplify the grammar and vocabulary
- Predictive text input: Faster or easier typing
- Sentiment analysis: Attitude of speaker
- Automatic summarization: Extractive or abstractive summarization
- Natural Language Generation: Generate text from data
- Speech recognition and generation: Speech-to-text, text-to-speech
- Question answering: Determine the intent of the question, match query with knowledge base, evaluate hypotheses

Possible Capstone Projects: Specific

- Consumer complaints data analysis and case outcome prediction
  - Collect case status data from district, state and NCDRC
  - Extract important basic info about each case (using NLP techniques) and add to the features
  - Build predictive models for the outcome of the complaints
- Consumer complaints judgments summarization
  - Real estate
  - Healthcare
- Similar consumer complaints judgements identification
- NLP based features engineering for consumer complaints judgements(may need to use sequential models also)

Summary

- Text Analytics Frameworks
- Stemming
- Lemmatization
- Named Entity Resolution
- POS Tagging
- TF-IDF
- Language Detection
- Case studies
  - Documents Similarity
  - Documents Clustering
  - Yelp Reviews Sentiment Analysis and Classification
  - News Headlines Analysis
- References
- Possible Capstone Projects

Questions?