



Search or jump to...

/

Pull requests

Issues

Marketplace

Explore



alstampe / Capstone---NLP-project

Watch

0

Star

0

Fork

0

Code



Issues

0



Pull requests

0



Projects

0



Wiki



Insights



Settings

Branch: master

Capstone---NLP-project / capstone report
template.md

Find file

Copy path

alstampe Report completed

83ab7b1 Nov 29, 2018

1 contributor

737 lines (481 sloc) 42.6 KB

Raw

Blame

History



DNB Data Scientist for Enterprise Nanodegree Program

Capstone Project

Anne Line Stampe, DNB November 30th, 2018

I. Definition

Project Overview

Natural Language Processing (NLP) is a highly relevant topic in many business areas, and specifically so within DNB, Norways largest financial institution. The need for rapidly increasing levels of automation and digitalization spurs a demand for ways to reengineer processes based on manual handling of unstructured data. In addition to reduce costs and risks there is a clearly stated ambition to gain and use data-driven insight from these initiatives.

With a background from participation in an OCR-project piloting recently I chose a NLP-case as my Capstone project, uniting freshly aquired skills in the Nanodegree program and an interest in real-life document analysis.

NLP was briefly introduced in the Nanodegree course, giving a usefull basis for this project.

My focus is to build skills and insights in possible realistic solutions for analysis of text in the form of full-scale documents rather than emails, tweets or other short, informal text objects. The data used in the project is text from 41 full books from the Project Gutenberg. For test purposes an additional unknown book and a separate text document is introduced at the end of the project.

As described in the project proposal my project covers three topics;

- Analysis of words in a full corpus; vocabulary, vector representation, word similarity
- Methods for comparing individual text objects (books) for similarity based on vector representation
- Methods for topic extraction from text objects, on a book level

The project has a high relevance for my job; in DNB we want to be able to understand documents by assessing vocabulary, comparing documents and extracting topics with algorithms as an alternative to a 100% manual reading and understandig the texts. There is also a need to be able to isolate specific data elements from documents (names, account numbers, values etc), but this is not a part of the Capstone scope. Because of this internal relevance, I plan to try out the project code on

internal documents later.

Problem Statement

The underlying problem I want to address is the challenge we face when automating our processes in the bank; some tasks are by nature based on non-numerical data and the understanding of 'real language'. We still have to maintain sizeable staff for digesting large volumes of information manually with related cost and risk of human errors. There is a need to speed up the process and digitize results to be able to make data a part of a digital process. To harvest reusable insight from extracted information would be a valuable by-product. Fundamentally, the initiative can also add traction to the collaborative effort of transforming the bank to a more techno-driven institution and illustrate opportunities in the field of machine learning and AI.

Exploring NLP opportunities does not imply that all human handling of text is expandable, there are numerous situations where we must have skilled people reading and analyzing text based on more non-procedural rules, but it is a start on a journey to a more digital bank.

I go into this project with a moderate ambition of precision from the document analysis, but expect to be able to confirm the hypothesis stated in the three questions. For such a complicated field, a modest proof of concept is a good basis for believing in results from a possible full-scale project later.

Strategy for the work

All text analysis, as other ML tasks, starts with understanding the data, the questions asked and the tools and methods needed to answer the questions with available data.

Building more relevant skills

Before starting to code, I needed some more nlp-specific knowledge. I have used the Kindle book below which I spent some time reading before getting at the coding part.

O'REILLY®



Applied Text Analysis with Python

ENABLING LANGUAGE-AWARE DATA PRODUCTS
WITH MACHINE LEARNING

Benjamin Bengfort,
Rebecca Bilbro & Tony Ojeda

Gathering and loading input data.

The data for this project is solely books from Project Gutenberg, with the exception of a single document to be tested at the very end of the project. All books will be downloaded on my laptop and then uploaded to my Notebook, to be read into dataobjects.

Preprocessing

Preprocessing text data is partly similar to cleaning numerical and categorical data, partly very different. The options and choices are many, hence this step is important and represents a large part of the project's time and effort. Preprocessing code for all three parts of the project scope will be prepared and tested before the data transformation starts.

Transformation, training and producing results

Transforming the text data by establishing models, vectorization, dimensionality reduction, training the model and producing results will be more straightforward and similar to numerical-oriented projects, but steps and methods must be appropriate to the case and results visualized to inspect language-oriented elements.

Inspect and assess

Results from transformation of text can be visualized in addition to list-based output.

Test of an 'unknown' additional book from Gutenberg and a contemporary text document

Finally I will do a test on an ebook and a text document which are both unknown and new. The test will be of the Topic extraction.

Evaluation

Results will be discussed.

Deliverables

Project deliverables will primarily be in form of results described in the report with detailed description of code, models and figures. All results are commented and discussed.

In addition all code, test and comments are present in the project Notebook. All books reside on GitHub.

The Notebook is quite extensive as it includes all code used for testing different ways to preprocess the text, plus some vectorization attempts which I did not utilize: Hence, not every part of the Notebook is used for the final solution but has been important to learn more about the many ways to handle text data.

Metrics

As described in the capstone project proposal, the three parts of my project are all of a nature where sharp metrics are not always present. The individual tasks will produce some comparable values, but more important is the subjective opinion ie:

Word similarity

```
Are the similarity suggestions on single words correct according to my language understanding'?
Does the word pairing examples on the vectorized data make sense?'
```

Book similarity

```
'Does the book similarity look sensible, given my knowledge of the books?'
```

Topic extraction

```
'Is this a readable and good topic extraction from the given book?'
```

II. Analysis

Data Exploration and description

Data for the project is in the form of a 'mini-library' of 41 books collected from the project Gutenberg. Project Gutenberg is a

collection of freely available ebooks on various formats, forming a huge historical library of literature.

'Project Gutenberg offers over 57,000 free eBooks. Choose among free epub books, free kindle books, download them or read them online. You will find the world's great literature here, with focus on older works for which copyright has expired. Thousands of volunteers digitized and diligently proofread the eBooks, for enjoyment and education'. (From the website www.gutenberg.org)

As a balance between informative corpus size and variety - and a volume manageable for computation on a standard laptop - I decided to use 41 books for my mimi-library. The choice of books is based on :

```
The books are all in english
6 books have Norwegian or Danish origin (Ibsens plays and 2 fairytale collections)
Some of the items are plays (Ibsen, Shakespeare)
I have personally read all books - primarily wholly, but for some only partially.
Some authors are represented by several works (Verne, Ibsen, Dumas, Burroughs)
Nearly half of the items can be described as 'adventure journey tales'.
Two of the authors (Shakespeare and Ibsen) are known for their very rich vocabulary(29.000 and 27.000).
```

My list, as named in the file folder:

```
'A_Dolls_house_Ibsen.rtf',
'Alice_in_Wonderland.rtf',
'All_around_the_moon_Verne.rtf',
'An_archtartic_mystery_verne.rtf',
'Anthem_Rand.rtf',
'Around_the_world_in_80_days_Verne.rtf',
'Don_Quixote.rtf',
'Fairytale_H_C_Andersen.rtf',
'Ghosts_Ibsen.rtf',
'Great_Expectations_by_Charles_Dickens.rtf',
'Gullivers_Travels_Swift.rtf',
'Hedda_Gabler_Ibsen .rtf',
'Iliad_Homer.rtf',
'Jungle_tales_of_Tarzan.rtf',
'Little_Eyolf_Ibsen.rtf',
'Martin_Eden_Jack_London.rtf',
'Meditations_Aurelius.rtf',
'Metamorphosis _ Kafka.rtf',
'Norwegian_tales_Asbjornsen_Moe.rtf',
'Peter_Pan.rtf',
'Pride_and_Prejudices .rtf',
'Robinson_Crusoe _ Defoe.rtf',
'Shakespeare.rtf',
'Siddharta _ Hesse.rtf',
'Swanns_Way_Proust.rtf',
'Tale_of_two_cities_dickens.rtf',
'Tarzan_of_the_apes.rtf',
'Tarzan_the_terrible.rtf',
'The_3_musketeers_Dumas.rtf',
'The_Iron_Heel_Jack_london.rtf',
'The_beasts_of_Tarzan.rtf',
'The_black_tulip_Dumas.rtf',
'The_brothers_karamazov_Dostoyevsky.rtf',
'The_importance_of_being_earnest_Wilde.rtf',
'The_jungle_book_Kipling.rtf',
'The_man_in_the_iron_mask_Dumas.rtf',
'The_return_of_tarzan.rtf',
'The_secret_of_the_island_verne.rtf',
'The_trial_Kafka.rtf',
'War_and_peace_tolstoy.rtf',
'Wuthering_Heights_bronte.rtf'
```

Although the filenames indicate .rtf (Rich Text Format), they are in straightforward text format. I started with a few rtf format files, but decided to download the .txt format which I realized is available for all my books. The file naming was kept for the sake of convenience as the code reading the files then was written for the .rtf extension. All books are present in a folder in GitHub.

Comments on the book data

After a few assessments of the data on a word level I noticed some parts of the vocabulary that I did not expect from older works, such as 'email'. I had screened through the first part of most of the books to verify the content and format, which all seemed correct. At closer inspection I realized that all ebooks had a standard, quite long, section at the end, containing legal description of the ebook handling. In addition to the introduction of unwanted irrelevant vocabulary to the corpus this part is the same in all books and would therefore reduce the uniqueness of the items. The section was for this reason removed from each of the books.

The complete volume of words from the books, the 'corpus' was analyzed thoroughly. There are many levels (and combination of levels) to look at the corpus;

```
Corpus level
Book level
Sentence level
Vocabulary level (unique words)
Word level (real words, or 'tokens')
Level of 'words' before removing non-meaningful characters.
Level of 'reduced tokens'; ie words cleaned and/or transformed to a netted minimum.
```

Below is an example of numbers for these levels

```
Corpus_raw: 30776216
Wordlist: 5572533
Sentences: 309218
Vocabulary: 96909
```

As a reflection 96909 comes across as a very rich vocabulary, even with Shakespeare and Ibsen onboard. One reason is probably that names are included in the corpus and are counted as words. Linguists state that if we include all varieties of known english words it will add up to ca 500.000. A normal active vocabulary consists of a modest 5-6000 words in most countries.

One of the first transforms on the corpus was done using the `gensim.utils.simple_preprocess`, which performs this : 'Convert a document into a list of lowercase tokens, ignoring tokens that are too short or too long'. https://programtalk.com/python-examples/gensim.utils.simple_preprocess/.

On inspection this returns a compact list of tokens, the number of words reduced significantly, ie for Alice in wonderland, from 53690 raw words to 9401 tokens. Netting down to unique tokens, in effect the 'token vocabulary' shows a modest token vocabulary of 1504.

Although utilities with options such as the 'simple_preprocess' are useful, I wanted to test effects of the preprocessing in detail, and spent time trying out different functions on sentence and word levels, coding small functions with basic tokenizers and cleaning.

Assessing different word_tokenizers

Tokenizers in common utility libraries offers a variety of processing options for establishing 'tokens'; separate word objects. As an illustrative example, I tested the different outcomes of two often used word tokenizers applied on sentences created from the `punct_sentence_tokenizer`;

```
nltk.tokenize.word_tokenize(sentence) versus nltk.tokenize.wordpunct_tokenize(sentence)
```

For sentence no:99 in the Alice in Wonderland corpus we can see how the word tokenizers differ, respectively:

```
'she was up to her chin in salt-water.',
['she', 'was', 'up', 'to', 'her', 'chin', 'in', 'salt-water', '.'],
['she', 'was', 'up', 'to', 'her', 'chin', 'in', 'salt', '-', 'water', '.']
```

The `word_tokenizer` keeps the hyphenated words as one token, the `wordpunct_tokenize` splits these words into 3 separate tokens. When analyzing text in literary works, it makes sense to keep the original word as written by the author, this sentence is a good example as the words 'salt' and 'water' separately is quite different from 'salt-water'. The first could be a part of a recipe for baking bread, the second indicates seawater - or tears, as is the case in this story.

I choose to use the `word_tokenizer` for this book-based project, although another context could be a case for preferring the `wordpunct_tokenizer`. Use of hyphenation varies between languages, is said to be declining in the English language and is

less common in Norwegian. A discussion on tokenizers is presented here:

https://www.researchgate.net/publication/264157595_A_Comparison_of_13_Tokenizers_on_MEDLINE

Cleaning the sentences for stopwords

When preparing for topic analysis and other content assessments we want to look at the meaningful elements of the text, not the 'filler words'. There are many lists of 'filler words' or 'stop words' as they are called, for several languages. I use a list of english 'stop words' (from the nltk.corpus) and write two small code snippets to remove these words in a sentence and then loop for all sentences in a book. The function 'make-fin-sent' (below) will 'make final sentences' for a given book, to be used for topic extraction where we want to avoid stopwords:

```
def remove_stopwords(words):
    return [word for word in words if word not in stop_words]

def make_fin_sent(sent_in):
    fin_sent=[]
    for sent in sent_in:
        fin_sent.append(remove_stopwords(sent))
    return fin_sent
```

A final sentence will typically look like this (sentence no 80 in Swanns Way):

```
['Come', 'stop', 'husband', 'drinking', 'brandy']
```

Looking at frequent words in a text

Zooming on on the corpus, the need for data cleaning soon becomes obvious. First try of 'most common words' on a raw (uncleaned) corpus returns a rather uninteresting list over the 10 most frequent 'words', using FreqDist (from nltk.probability import FreqDist).


```
{',': 460770, 'the': 270580, '.': 258286, 'and': 160660, 'of': 143300, 'to': 142159, 'a': 95923, 'I': 9
```



Removing punctuations and single characters with a small cleaning function improves a little:

```
cleanlist=[]
def sentence_to_cleanlist(raw):
    cleanlist = re.sub("[^a-zA-Z]", " ", raw)
    return cleanlist

({'the': 270904, 'and': 161240, 'to': 143756, 'of': 143437, 'a': 96535, 'I': 92847, 'in': 79205, 'that':
```



This exercise can be done for the individual books, but across english books the top 20 words pretty much remains the same.

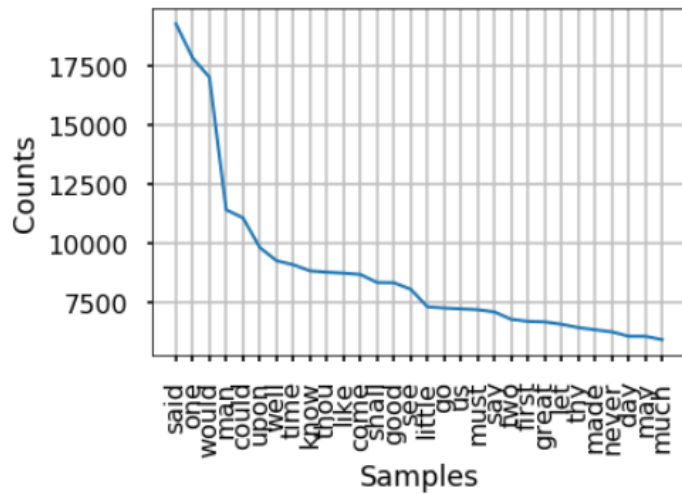
Next step is to remove the stop words, as described above. We can see that all 10 words on the previous list were in fact stopwords.

Using the result from removing stopwords with make_fin_sent, the 10 most frequent words for the full corpus are:

```
('said', 19290),
('one', 17840),
('would', 17020),
('man', 11407),
('could', 11068),
('upon', 9815),
('well', 9252),
('time', 9081),
('know', 8822),
('thou', 8765)
```

Looking at this list, I can recognize the words of literature objects and dialogue-oriented text, with 'said' on top of the lists. The last word is a hint of the book's age and I guess this is a Shakespearian word.

Below is a figure showing the count of the 30 most common words in the corpus, after preprocessing as described.



Applying the word frequency on only Alice in Wonderland, without stopwords, we get this list:

```
('alice', 173),
('said', 144),
('little', 59),
('rabbit', 37),
('one', 35),
('like', 34),
('queen', 30),
('could', 28),
('mouse', 27),
('illustration', 26)
```

For 'The junglebook' , the 10 most frequent words when cleaned for filler words are (using FreqDist again):

```
'said': 430, 'little': 231, 'mowgli': 220, 'man': 177, 'one': 174, 'would': 162, 'jungle': 147, 'head':
```

Recap - data cleaning in nlp

Cleaning the corpus can be done by several small cleaning code-snippets;

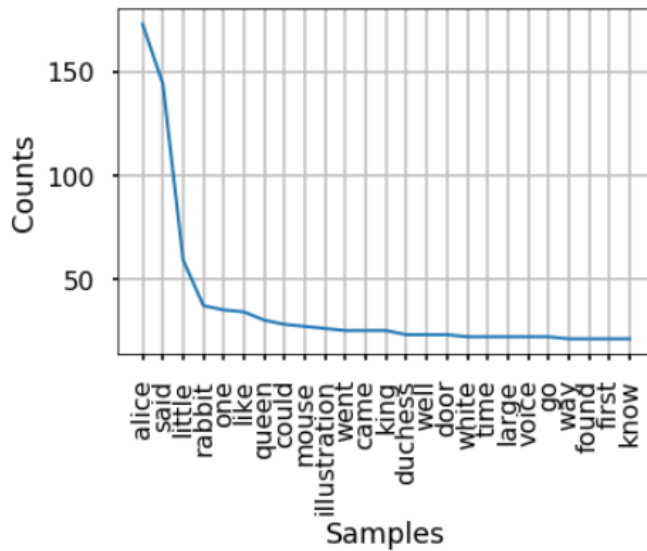
```
sentence tokenization
word tokenizations
removing unwanted characters (often called normalization)
converting all words to lowercase
removing stopwords
lemming and stemming
pos-tagging
```

The 5 first cleaning steps are described. The two latter were tested on the corpus, but not used for later transformation purposes. Code for these tests are still kept in the Notebook.

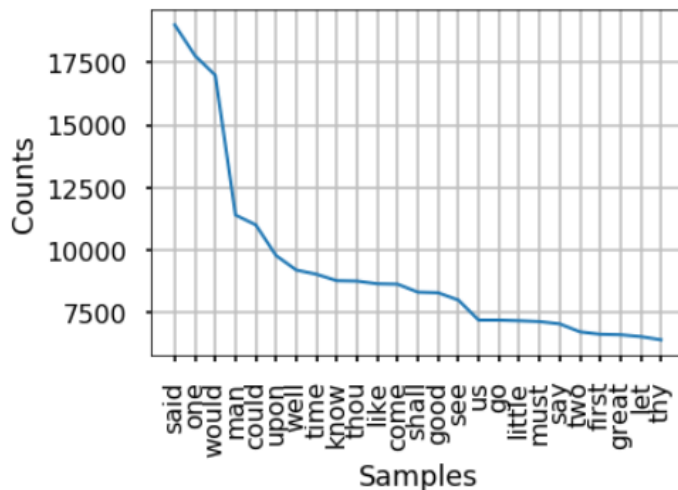
Exploratory Visualization

Word frequency after different word preprocesses is interesting. Below are two more plots describing this, showing number of words in the whole corpus and in one book. Both are cleaned by normalization (removing stray characters), the book is cleaned for stopwords.

```
fdist_fin_alice.plot(25, cumulative=False)
plt.show()
```



```
fdist_all_fin_sent.plot(25, cumulative=False)
plt.show()
```



More visuals are provided later on the word analysis task.

Algorithms and Techniques

Algorithms used for the projects are chosen based on my understanding of useful techniques, based on learning in the Nanodegree, the book I have read and supplementing articles.

As described in the book, there are three commonly used ways to treat text analysis, often by combinations of the three;

```
The Gensim way
The nltk way
The ScikitLearn way
```

My use of algorithms are mainly by using Gensim utilities, but also nltk resources.

The data preprocessing was described in previous part of the report;

```
Use of word- and sentence tokenizers from nltk.tokenize
Use of nltk.probability.FreqDist
Use of nltk.corpus.stopwords
Use of gensim.utils.simple_preprocess
```

Word and document analysis


```
gensims Word2Vec and Doc2Vec for vectorization of corpus
gensim similarities for similarity analysis
sklearn.manifold.TSNE for dimension reduction
```

Topic extraction

```
gensim Dictionary and LDAmodel from gensim.models
```

For all algorithms the initial parameters are set based on examples from the Udacity courses, the book, or articles. Some are set from characteristics of the data (ie vocabulary size). Many of the parameters are changed through the project to understand how and if the models can be optimized.

Benchmark

A benchmark for performance for this project is somewhat challenging as the 'real-life' benchmark starts with time and cost for a person reading and understanding text. Such time-and cost values not often used as benchmark in these experiments. Nevertheless, these figures will be the basis for a business case if the solution is to be evaluated for a project, measured against the project costs.

With an average adult reading speed of 250 words per minute a book with 90.000 words (estimated average length of a novel) will be read in 360 minutes; 6 hours. For my bank case, a relevant comparison would be a document of perhaps 5000 words; estimated reading time being 20 minutes. The act of analysis and documenting the results will represent additional time and effort. Obviously, a machine-based read, classification and topic extract will represent a major performance increase.

If we rather see the benchmark as the 'first state of the solution' , to be compared to the final version the report describes and discusses the model improvements. The most prominent improvements were not by fine-tuning the complex models, but from the level of cleaning in the preprocess stage. A raw corpus can be seen as a very unbalanced dataset, where large volumes of single characters and stopwords skew the corpus and make topic extraction nearly impossible.

III. Methodology

Data Preprocessing

NLP requires several steps of preprocessing just as we do for numerical values and categorical data. The methods are different, but there are several rich libraries of code available. In effect, the first challenge is to understand which packages to apply. Data cleaning for nlp differs from cleaning of datasets for numerical analysis - the choice of preparing must be made based on the purpose of the analysis, we can seldom do an all-purpose text cleaning.

I realized that the three separate questions posed in the project would have to be answered with three separate paths with different data prepping and processing.

1. Data analysis of all words in all books in the mini-library With a focus on the sum of all words in the book collection the preprocessing starts with a 'full' collection of words across the books. The text is then divided into sentences and all words are identified separately. The books are structured into sentences, using a trained process. I also use a word-oriented vectorization and eventually look at the matrix of all words, identifying the most frequent words and how the words are related to each other in the matrix.

In the analysis process I keep separate versions of the result of different text preparations, to be in control of the results and which versions are suitable for the next step of analysis.

2. Data analysis for comparing books in the mini-library. ,
When aiming to compare text objects I need to separate the objects(books) and prepare them individually, building structures for mathematical comparisons. For this purpose I use another version of the package, with a document(ie book) focus. Each book is vectorized and the vectors are compared.
3. Data prepping and analysis for topic extraction Finally, the prepping for topic extraction will reuse some of the prep steps, but with a twist; the text is divided into books again, this time on a corpus level. All analysis is done on book level. In addition to constructing sentences and cleaning for unnecessary characters, the text is made into lower characters and all stopwords are removed explicitly. Stopwords in the text makes it difficult to extract topics and forms unnecessary 'noise'. This was very clearly visible from the code run I performed before and after stopword removal.

Implementation

Transformation of data after preprocessing

Stepping up the analysis after preprocessing is done by vectorizing the corpus. There are numerous different ways to do this and quite tricky to navigate among the options. I chose to use the Gensim functionality for vectorizing. Gensim offers vectorization option on document and word level, with word2Doc and Word2Vec.

1) Word analysis with Word2Vec

Gensim has a reknown Vectorization option, the Word2Vec which, as the name implies, vectorizes the corpus on a word level. Each unique word is assigned a numerical value, giving a matrix where one dimension is the size of the vocabulary.

I start by establishing a word-based Word2Vec model; named 'word_model'.

The word_model is applied on the whole corpus tokenized on a sentence-level, using the version called 'sentences'. In 'sentences', the full corpus is split into sentences and each word is tokenized.

Sentence no 5678 in the corpus is a good example:

```
'when', 'we', 'get', 'to', 'the', 'moon', 'what', 'shall', 'we', 'do', 'there'
```

The word_model is set by :

```
Parameters:
num_features = 300
min_word_count = 15
num_workers = multiprocessing.cpu_count()
context_size = 10
downsampling = 1e-4
seed = 9

word_model = w2v.Word2Vec(parameters)
word_model.build_vocab(sentences)
print("model vocabulary length:", len(word_model.wv.vocab))
model vocabulary length: 14661
```

Confirming the model after building vocabulary:

```
Word2Vec(vocab=14661, size=300, alpha=0.025)
word_model.train(sentences, epochs=5,
total_examples=14661)

all_word_vectors_matrix = word_model.wv.vectors
```

The resulting matrix is huge (14661, 300). To be able to visualize and look into the matrix I use TSNE to reduce the dimensions to 2.

```
tsne = sklearn.manifold.TSNE(n_components = 2,
                             early_exaggeration = 6,
                             learning_rate = 500,
                             n_iter = 300,
                             random_state = 2)

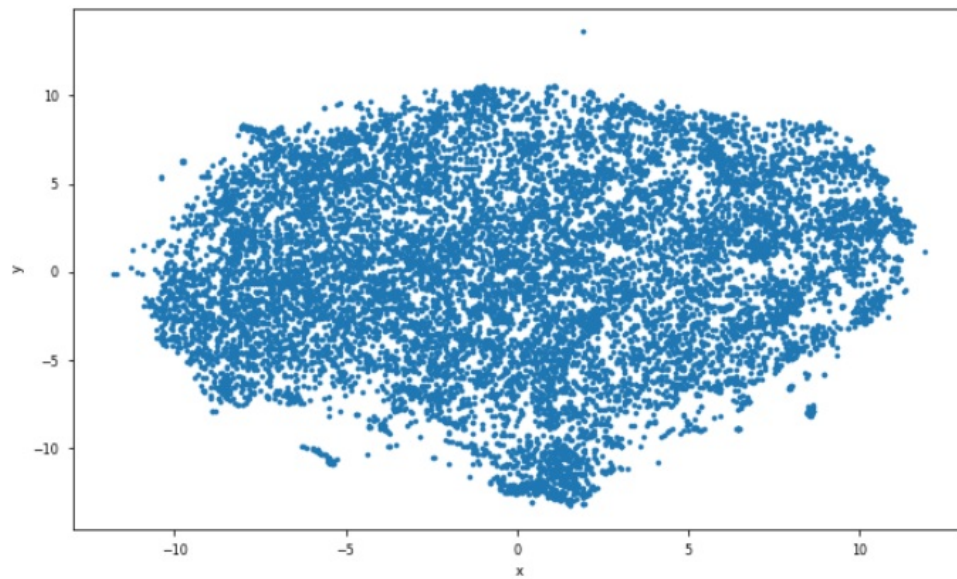
all_word_vectors_matrix_2dim = tsne.fit_transform(all_word_vectors_matrix)
```

The matrix values are set into a dataframe for inspection and visuals. The shape is the vocabulary size (14661, 2). On this dataframe I can 'look at' the word data in a frame with visuals. All the 14661 words are placed along the x- and y-axis, forming a circle-like cloud. I can 'zoom in' on elements in the frame, by navigating along the x- and y-axis.

When picking a word the coordinates are given, showing where the word resides. If I 'zoom in' on the area around the chosen word by picking a slice of the frame by the coordinates, I can see which words are in the close vicinity vector-wise. A few examples:

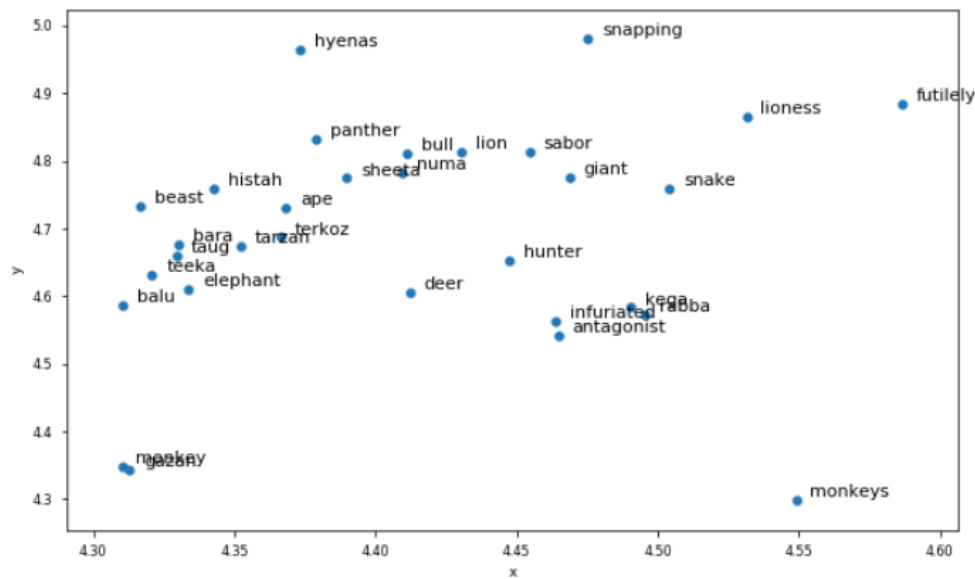
```
# Display the layout of all of the points.
sns.set_context("paper")
points.plot.scatter("x", "y", s=10, figsize=(10, 6))
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x223217ee470>
```



	word	x	y
8840	ape	4.368067	4.730412

```
plot_region(x_bounds=(4.3, 4.6), y_bounds=(4.2, 5))
```



Most similar words - in the vector dimension

Below are two examples, the first on the initial model version, the second on an enhanced model version when upping some of the parameters (number of features, context size) . The results were not dramatically improved for this task after model tweaking.

```
word_model.wv.most_similar('pride')

('envy', 0.7191991806030273),
('ambition', 0.7174361348152161),
('meekness', 0.7124776244163513),
('fortitude', 0.7029587030410767),
('chastisement', 0.7014052271842957),
('disdain', 0.7011118531227112),
('hypocrisy', 0.7005854845046997),
('penitence', 0.6966824531555176),
('cowardice', 0.696437656879425),
('deceitful', 0.6881036758422852)

word_model.wv.most_similar('hunger')

('thirst', 0.8457642197608948),
('pangs', 0.7334113121032715),
('fatigue', 0.7226671576499939),
('gnawing', 0.7220444679260254),
('starved', 0.6977795362472534),
('exhaustion', 0.6882658004760742),
('famine', 0.6871694922447205),
('hungry', 0.6860570907592773),
('sickness', 0.6858627796173096),
('craving', 0.6856997013092041)
```

Word pairing

Finally, I want to try out the classic 'king-to-queen' vector comparison. For this purpose I use the `most_similar_cosmul` functionality, based on a cosine computation.

The base for this is a 'x relates to y as a relates to...' logic, expecting to return the word that has a meaningful relation to a as y has to x. My attempts to do this were not in the first tryouts as clear as the reknown queen-king case. I thought the reason was a limited corpus, but changing the model parameters improved the result, as shown below.

```
def nearest_similarity_cosmul(start1, end1, start2):
    similarities = word_model.wv.most_similar_cosmul(
        positive=[start1, start2],
        negative=[end1])
    end2 = similarities[0][0]
    print("{start1} is related to {end1}, as {start2} is related to {end2}".format(**locals()))
    return end2
```

First version of model returns:

```
'ocean is related to submarine, as jungle is related to forest
swann is related to odette, as nora is related to helmer (OK!)
queen is related to king, as woman is related to girl'
```

After changing parameters the similarity tests are better;

```
nearest_similarity_cosmul("man", "woman", "queen")
nearest_similarity_cosmul("husband", "wife", "man")
nearest_similarity_cosmul("swann", "odette", "artagnan")
nearest_similarity_cosmul("sea", "boat", "city")

'man is related to woman, as queen is related to king
husband is related to wife, as man is related to woman
swann is related to odette, as artagnan is related to porthos
sea is related to boat, as city is related to palaces'
```

2) Book comparisons, using Doc2Vec

Doc2Vec is similar to Word2Vec, but applied on document (here book) level.

A hypothesis was to use tfidf here, but reading articles seems to indicate that LDA does not need TF-IDF and can be used

with bag_of_words only. There are different options on whether TF-idf will improve the results, but without clear conclusions. This is also commented later in the report.

The model is defined and named model_doc. Data input is 'book_corpus', the book-level corpus established earlier. The model uses 3 parameters. All were varied during the project, the results seemed to stay at fairly the same level.

```
model_doc = gensim.models.Doc2Vec(vector_size = 300,
                                   min_count = 3,
                                   epochs = 10)

model_doc.build_vocab(book_corpus)
model's vocabulary length: 31862

model_doc.train(book_corpus, total_examples=31862, epochs=10)
```

Finding the most similar book pairs, sorted by hitrate(similarity)

```
for book in book_filenames:
    most_similar = model_doc.docvecs.most_similar(book)[0][0]
    print("{} - {}".format(book, most_similar))
```

Result list below shows that similarity is found between books from same author, but also of same genre. Folklore tales are paired, so are Gullivers travels and Robinson Crusoe.

```
Books\A_Dolls_house_Ibsen.rtf - Books\Hedda_Gabler_Ibsen .rtf
Books\Alice_in_Wonderland.rtf - Books\Norwegian_tales_Asbjornsen_Moe.rtf
Books\All_around_the_moon_Verne.rtf - Books\The_secret_of_the_island_verne.rtf
Books\An_archtartic_mystery_verne.rtf - Books\The_secret_of_the_island_verne.rtf
Books\Anthem_Rand.rtf - Books\The_jungle_book_Kipling.rtf
Books\Around_the_world_in_80_days_Verne.rtf - Books\The_secret_of_the_island_verne.rtf
Books\Don_Quixote.rtf - Books\Iliad_Homer.rtf
Books\Fairytales_H_C_Andersen.rtf - Books\Norwegian_tales_Asbjornsen_Moe.rtf
Books\Ghosts_Ibsen.rtf - Books\Little_Eyolf_Ibsen.rtf
Books\Great_Expectations_by_Charles_Dickens.rtf - Books\Tale_of_two_cities_dickens.rtf
Books\Gullivers_Travels_Swift.rtf - Books\Robinson_Crusoe _ Defoe.rtf
```

When testing on specific books, I use the 'most_similar':

```
model_doc.docvecs.most_similar('Books\Tarzan_of_the_apes.rtf')
```

This returns the other Tarzan books, followed by adventure books for adolescents; quite a good result.

```
('Books\\The_beasts_of_Tarzan.rtf', 0.7585173845291138),
('Books\\Jungle_tales_of_Tarzan.rtf', 0.7063543796539307),
('Books\\The_return_of_tarzan.rtf', 0.4879530370235443),
('Books\\Tarzan_the_terrible.rtf', 0.4041978716850281),
('Books\\An_archtartic_mystery_verne.rtf', 0.319263219833374),
('Books\\The_secret_of_the_island_verne.rtf', 0.3165384531021118),
('Books\\The_jungle_book_Kipling.rtf', 0.31132781505584717),
('Books\\Peter_Pan.rtf', 0.30049124360084534),
('Books\\Alice_in_Wonderland.rtf', 0.28424006700515747),
('Books\\Robinson_Crusoe _ Defoe.rtf', 0.2718788981437683)

model_doc.docvecs.most_similar(10) (Book 10 = Gullivers Travels by Defoe)

('Books\\Robinson_Crusoe _ Defoe.rtf', 0.47113150358200073),
('Books\\Pride_and_Prejudices .rtf', 0.3537435233592987),
('Books\\Meditations_Aurelius.rtf', 0.3536701798439026),
('Books\\Fairytales_H_C_Andersen.rtf', 0.25862976908683777),
('Books\\The_secret_of_the_island_verne.rtf', 0.25140485167503357)
```

Tsne dimension reduction was done for these vectors as well, and result inspected visually (this is in the Notebook)

```
all_doc_vectors_matrix = model_doc.wv.vectors
all_doc_vectors_matrix_2 = tsne.fit_transform(all_doc_vectors_matrix)
```

3) Topic extraction with doc2bow and LDA

For topic extraction I chose to use a third part of Gensim; doc2bow - using a bag-of-word method (bow: bag-of-words) I also used another algorithm; LDA; often used for nlp. LDA: Latent Dirichlet Allocation. LDA is also from the Gensim package.

Create a Dictionary from the data, then a bag of words.

About LDA : https://en.wikipedia.org/wiki/Latent_Dirichlet_allocation

Using Alice in Wonderland as example, these are the steps for Topic Extraction, starting with the `alice_fin_sent` version of preprocessed corpus, as described earlier. For topic extraction, filler words must be removed. A Lda model is build using the dictionary-based matrix.

```
dictionary_alice = corpora.Dictionary(alice_fin_sent)

doc_term_matrix_alice = [dictionary_alice.doc2bow(doc) for doc in alice_fin_sent]

Lda = gensim.models.ldamodel.LdaModel

ldamodel_alice = Lda(doc_term_matrix_alice, num_topics=3, id2word = dictionary_alice, passes=750)
```

Results from this can be seen when we print the topics from the LDA algorithm

```
alice_topics = ldamodel_alice.print_topics(num_words=5)
for topic in alice_topics:
    print(topic)
```

Presenting the proposed 3 topics using 50 passes:

```
(0, '0.014*"one" + 0.012*"duchess" + 0.009*"alice" + 0.008*"first" + 0.008*"little"')
(1, '0.036*"alice" + 0.018*"little" + 0.016*"said" + 0.012*"rabbit" + 0.009*"could"')
(2, '0.056*"said" + 0.037*"alice" + 0.012*"king" + 0.012*"know" + 0.009*"like"')
```

Setting to 250 passes:

```
(0, '0.031*"alice" + 0.021*"said" + 0.010*"mouse" + 0.009*"thought" + 0.009*"go"')
(1, '0.041*"alice" + 0.037*"said" + 0.010*"rabbit" + 0.007*"came" + 0.007*"moment"')
(2, '0.017*"one" + 0.017*"little" + 0.013*"said" + 0.013*"alice" + 0.012*"way"')
```

Setting to 750 passes:

```
0, '0.066*"said" + 0.042*"alice" + 0.010*"duchess" + 0.009*"oh" + 0.009*"think"')
(1, '0.014*"little" + 0.013*"know" + 0.010*"alice" + 0.009*"said" + 0.009*"queen"')
(2, '0.031*"alice" + 0.015*"rabbit" + 0.011*"little" + 0.009*"white" + 0.008*"one"')
```

The topic extraction must be performed for each book, and on a sentence level. As shown, I varied the no: of passes, and also the no of topics and `num_words`. It is not obvious that the results increase with more passes.

Is "said+alice+duchess+oh+think" really better than "one+duchess+alice+first+little" ?

I can see that Proposed Topics are similar to the book's frequent words, but not 100% identical.

The same topic extraction steps were done for 5 other books, with similar results. As a pattern, topics seem to include the books protagonist, the second most important character (if any) and some typical words

Refinement

NLP enjoys a wide variety of resources for prepping and transforming data, Nevertheless, there is room for coding and adjusting before, during and after package usage.

Several of the improvement measures taken are mentioned above, a recap is as following;

Preprocess : Data gathering and cleaning

```
Legal clausuls at the end of the books were removed completely as I saw they introduced 'noise' in the t
Some sentences in the book's beginnings were removed for same reasons; Internet Urls, email addresses et
Tokenizers on sentence and word level were tested and compared; I chose the one that kept full original
```

1. Word analysis

Fdist results from raw_corpus were poor, I changed the input to sentences with both cleaning of characters and stopwords removed

```
Word2Vec model parameters:
num_features = 300 (increased from 200 )
min_word_count = 20 (increased from 10 to 20)
num_workers = multiprocessing.cpu_count() (unchanged)
context_size = 10 (increased from 5 to 10)
downsampling = 1e-4 (unchanged)
seed = 9 (unchanged)
```

TSNE parameters were tweaked similarly; aiming a balance between results and runtime

2. Book analysis

The book analysis is similar to the word analysis method-wise. I did tryouts with parameters for the model and the tsne dimension reduction. On book-level, the model seems less sensitive for parameter tuning, probably because we operate on a book-level where similarity will be more obvious, the most typical result being similarity by author, followed by genres, picking out adventure stories for adolescents.

3. Topic extraction

The final task turned out to be quite straightforward to perform when I had found the recommended method from books and articles.

LDA parameters to investigate are passes, no of topics and no of words. Increasing number of passes did improve the results slightly.

The method is compact when all data is preprocessed and the steps prepared code-wise. I chose to write small code packages for a 'pipeline' and tested this on several books. The results are of varying quality, but confirm the idea of being able to extract topics automatically from larger texts, such as my books.

One of the useful articles about the topic is the following, by Susan Li <https://towardsdatascience.com/topic-modelling-in-python-with-nltk-and-gensim-4ef03213cd21>

IV. Results

A brief summary of the results for the three main tasks :

Word analysis results

```
Statistics were produced for corpus and books, on different levels
Corpus and vocabulary could be investigated, with a clear visual of vocabulary in a plotted wordcloud
I could zoom in on chosen words, see the vicinity words
Identifying most frequent words was done
Similarity words were delivered for any chosen word
Word pairs were produced on a satisfying level
```

Book similarity

```
Good results after parameter tweaks. Obvious similarity between books by the same author, also similarit
Visuals in form of a wordcloud as for the word analysis
```


Topic extraction

Acceptable good results.
Verification of result, in addition to scoring figures done by book knowledge and a generous interpretat

Test of unknown book Finally, I found a completely unknown book on Gutenberg and performed Topic extraction on the text

The-Chemistry-of-Food-and-Nutrition.txt

The results from this old book about food and chemistry strikes me by being suprisingly relevant for today's view on food and health.

```
'0.017*"food" + 0.016*"protein" + 0.012*"quantity" + 0.011*"matter" '  
'0.007*"salt" + 0.007*"one" + 0.006*"food" + 0.005*"meat" '
```

Extra : Test of a contemporary document

For fun and curiosity, I tested the topic extraction on a random modern document; "Google Chrome Terms of Service"

Spending 5 minutes downloading the file, running it through the code, gave this topic extraction:

```
(0, '0.031*"terms" + 0.030*"adobe" + 0.020*"software" + 0.020*"use" + 0.020*"sublicensee" + 0.019*"shall"  
(1, '0.030*"sublicensee" + 0.028*"terms" + 0.021*"google" + 0.020*"adobe" + 0.018*"rights" + 0.013*"addi"  
(2, '0.035*"adobe" + 0.028*"software" + 0.018*"video" + 0.018*"code" + 0.017*"content" + 0.014*"may" + 0  
(3, '0.071*"google" + 0.032*"terms" + 0.027*"services" + 0.020*"agreement" + 0.018*"adobe" + 0.018*"chro"  
(4, '0.046*"google" + 0.038*"services" + 0.023*"may" + 0.021*"use" + 0.019*"software" + 0.017*"adobe" +
```

It is not a trimmed executive summary, but it indicates the content, ie by the last topic suggestion

"google-services-may-use-software-adobe-sublicensee"

Additional Model Evaluation and Comments

Model usability

in this Kaggle post below (chapter 2 and 3) - and in other similar articles, different options for further work on the word vectors, beyond the bag-of_words (bow), are discussed. Is seems to be a common consensus that the 'bow' method is robust and gives consistently good results compared to other methods.

<https://www.kaggle.com/c/word2vec-nlp-tutorial#part-1-for-beginners-bag-of-words>

Justification

Benchmarking these results with manual handling of text is not easily measured, but I'll provide my reflections.

The important question I started with was; can we compare documents and extract essence in an effective way?

41 books in the mini-library may sound like a lot of words, but in this dicipline it forms a meagre corpus, yet proved enough to do the project and get interesting results.

Comparing documents by applying code to literal pieces of text as the library represents is interesting, but only partly representative for my real-life challenge. The documents in scope for implementation in DNB will be more specific and probably easier to distinguish by context vocabulary, aligning Trade Finance documents and separating those from ie Loan Agreements.

V. Conclusion and Reflection

Diving into NLP has been challenging yet satisfying as this is not a redo of projects but new skills and learnings.

This is not a straight-forward technique but an area where any project will be a series of crossroads where choices must be made of methods, algorithms and so on. Iterations is neccessary, and as metioned earlier there will be separate lanes for the

individual questions.

Topic extraction is the problem I am most eager to try out internally. This is the part of the project where there is the most room for improvement, if looking at the results. Yes, the topics suggested are ok, but the precision and presentetation can be better.

Alltogether, I find the results from a compact Capstone project, perfomed with a very basic understanding of the topic, to be promising. The goal of building insight and understanding has abslutely met my expectation from a personal view.

Improvement

My project did not include establishing a user interface for input of random documents, nor a prepped output in the form of screenbased result presentation or printable report. For my real-life problem this will have to be made, but not neccessarily a glossy user interfase, as this is for internal use in back office environments, nor for customer usage externally.

Topic extraction need to be investigated and tuned more, to find the optimal parameters for a given corpus.

Overall, improvements wil have to be initiated by learning more; reading and trying out more options.

© 2018 GitHub, Inc.

[Terms](#)
[Privacy](#)
[Security](#)
[Status](#)
[Help](#)



[Contact GitHub](#)
[Pricing](#)
[API](#)
[Training](#)
[Blog](#)
[About](#)