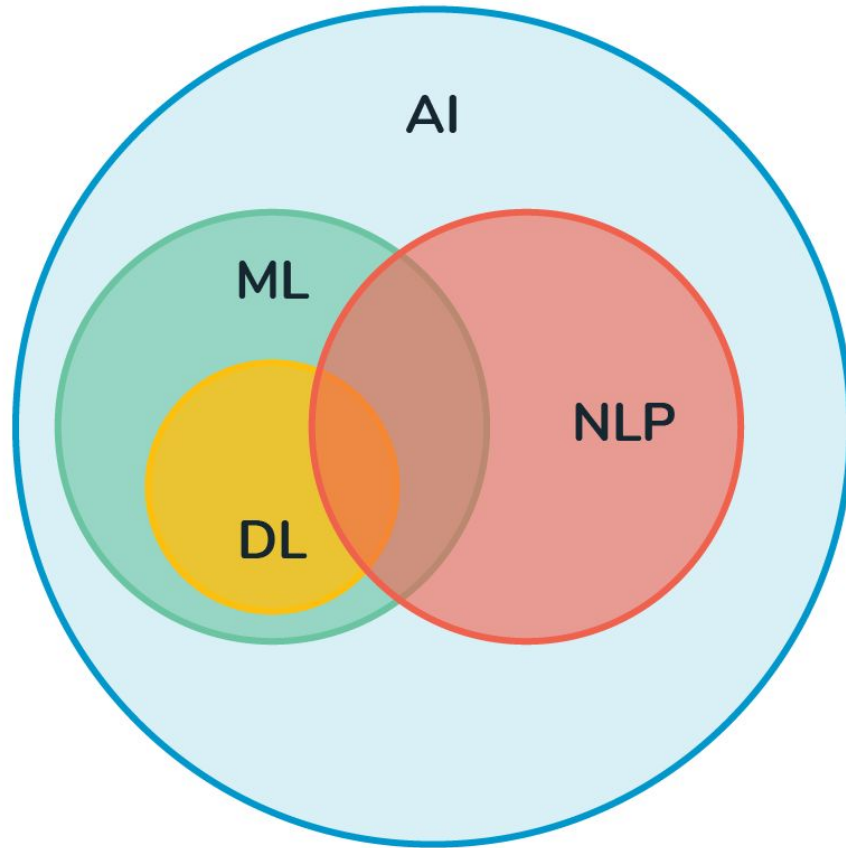





NLP and Topic Modeling

USC CARC Summer Bootcamp 2021
Asya Shklyar

What is NLP (Natural Language Processing)

- Linguistics, Computer Science, Artificial Intelligence/Machine Learning
 - Lemmas
 - Tokenizing
 - Bag of words
 - Stop Words
 - DAGs
- Humans and Language
 - We want to automate highly manual and repetitive work
 - We want to process larger and larger datasets
- What is “understanding” of the language?
 - Context is everything
- Speech Recognition
 - Customer Support Bots and Virtual Humans
 - <https://www.virtualhumans.org/>
- Language Generation
 - Chat bots and automatically generated papers
 - <https://news.mit.edu/2015/how-three-mit-students-fooled-scientific-journals-0414>



-  Artificial intelligence
-  Machine learning
-  Language Processing
-  Deep learning

What is Topic Modeling

Topic model

From Wikipedia, the free encyclopedia

In [machine learning](#) and [natural language processing](#), a **topic model** is a type of [statistical model](#) for discovering the abstract "topics" that occur in a collection of documents. Topic modeling is a frequently used text-mining tool for discovery of hidden semantic structures in a text body. Intuitively, given that a document is about a particular topic, one would expect particular words to appear in the document more or less frequently: "dog" and "bone" will appear more often in documents about dogs, "cat" and "meow" will appear in documents about cats, and "the" and "is" will appear approximately equally in both. A document typically concerns multiple topics in different proportions; thus, in a document that is 10% about cats and 90% about dogs, there would probably be about 9 times more dog words than cat words. The "topics" produced by topic modeling techniques are clusters of similar words. A topic model captures this intuition in a mathematical framework, which allows examining a set of documents and discovering, based on the statistics of the words in each, what the topics might be and what each document's balance of topics is.

Topic models are also referred to as probabilistic topic models, which refers to statistical algorithms for discovering the latent semantic structures of an extensive text body. In the age of information, the amount of the written material we encounter each day is simply beyond our processing capacity. Topic models can help to organize and offer insights for us to understand large collections of unstructured text bodies. Originally developed as a text-mining tool, topic models have been used to detect instructive structures in data such as genetic information, images, and networks. They also have applications in other fields such as [bioinformatics](#)^[1] and [computer vision](#).^[2]

How I learned about NLP - Economics - Job Ads



History

2018: Professor Manisha Goel (Econ) partners with

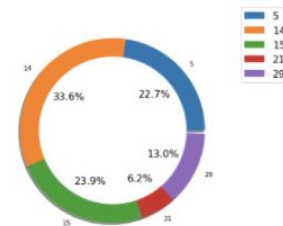
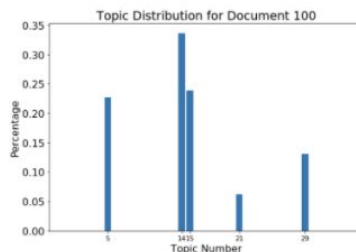
BGT (Burning Glass Technologies - a recruiting company)

Signs a contract to get a large dataset of job ads, descriptions, skills, pay information etc (10 years worth, 2005-2015) 

Has a student write a Python script to do some basic topic modeling using doc2vec and nltk/gensim

Random 50 samples completes just fine but 500 runs out of RAM even on a 512 GB Linux VM (largest available at the time)

Also runs for days instead of hours



Lemma

Lemmatization usually refers to doing things properly with the use of a vocabulary and morphological analysis of words, normally aiming to remove inflectional endings only and to return the base or dictionary form of a word, which is known as the **lemma**

<https://nlp.stanford.edu/IR-book/html/htmledition/stemming-and-lemmatization-1.html#:~:text=Lemmatization%20usually%20refers%20to%20doing,is%20known%20as%20the%20lemma%20.>

Sample text: Such an analysis can reveal features that are not easily visible from the variations in the individual genes and can lead to a picture of expression that is more biologically transparent and accessible to interpretation

Lovins stemmer: such an analysis can reveal features that are not easily visible from the variations in the individual genes and can lead to a picture of expression that is more biologically transparent and accessible to interpretation

Porter stemmer: such an analysis can reveal features that are not easily visible from the variations in the individual genes and can lead to a picture of expression that is more biologically transparent and accessible to interpretation

Paice stemmer: such an analysis can reveal features that are not easily visible from the variations in the individual genes and can lead to a picture of expression that is more biologically transparent and accessible to interpretation

Tokenization

Tokenization is a common task in **Natural Language Processing (NLP)**. ... Tokens are the building blocks of Natural Language. **Tokenization** is a way of separating a piece of text into smaller units called tokens. Here, tokens can be either words, characters, or subwords

[https://www.analyticsvidhya.com/blog/2020/05/what-is-tokenization-nlp/#:~:text=Tokenization%20is%20a%20common%20task%20in%20Natural%20Language%20Processing%20\(NLP\).&text=Tokens%20are%20the%20building%20blocks,words%2C%20characters%2C%20or%20subwords.](https://www.analyticsvidhya.com/blog/2020/05/what-is-tokenization-nlp/#:~:text=Tokenization%20is%20a%20common%20task%20in%20Natural%20Language%20Processing%20(NLP).&text=Tokens%20are%20the%20building%20blocks,words%2C%20characters%2C%20or%20subwords.)

```
Iteration 1 : l o w e s t </w>
Iteration 2 : l o w e s t </w>
Iteration 3 : l o w e s t</w>
Iteration 4 : l o w e s t</w>
Iteration 5 : l o w e s t</w>
```

BPE Completed...

Bigrams and Trigrams oh my

You can think of an N-gram as the sequence of N words, by that notion, a 2-gram (or **bigram**) is a two-word sequence of words like “please turn”, “turn your”, or “your homework”, and a 3-gram (or **trigram**) is a three-word sequence of words like “please turn your”, or “turn your homework”

<https://towardsdatascience.com/introduction-to-language-models-n-gram-e323081503d9>

And so, when you use a bigram model to predict the conditional probability of the next word, you are thus making the following approximation:

$$P(w_n | w_1^{n-1}) \approx P(w_n | w_{n-1})$$

*This assumption that the probability of a word depends only on the previous word is also known as **Markov** assumption.*

Markov models are the class of probabilistic models that assume that we can predict the probability of some future unit without looking too far in the past.

Bag of words!

The **bag-of-words model** is a simplifying representation used in [natural language processing](#) and [information retrieval](#) (IR). In this model, a text (such as a sentence or a document) is represented as the [bag \(multiset\)](#) of its words, disregarding grammar and even word order but keeping [multiplicity](#).

The bag-of-words model is commonly used in methods of [document classification](#) where the (frequency of) occurrence of each word is used as a [feature](#) for training a [classifier](#).

https://en.wikipedia.org/wiki/Bag-of-words_model

n-gram model [\[edit \]](#)

The Bag-of-words model is an orderless document representation — only the counts of words matter. For instance, in the above example "John likes to watch movies. Mary likes movies too", the bag-of-words representation will not reveal that the verb "likes" always follows a person's name in this text. As an alternative, the [n-gram](#) model can store this spatial information. Applying to the same example above, a **bigram** model will parse the text into the following units and store the term frequency of each unit as before.

```
[  
  "John likes",  
  "likes to",  
  "to watch",  
  "watch movies",  
  "Mary likes",  
  "likes movies",  
  "movies too",  
]
```

Conceptually, we can view bag-of-word model as a special case of the *n*-gram model, with *n*=1. For *n*>1 the model is named [w-shingling](#) (where *w* is equivalent to *n* denoting the number of grouped words). See [language model](#) for a more detailed discussion.

Some concepts related to NLP one needs to understand:

- Hadoop is pretty much done (but it used to be really cool - the distributed file system accessible on all nodes similar to a parallel file system)

<https://www.youtube.com/watch?v=QaoJNXW6SQo>

- Scala is a programming language that works similarly to Java (and uses JVM) but is much faster and was designed to be distributed (and is a functional language! vs just OOL like Java)

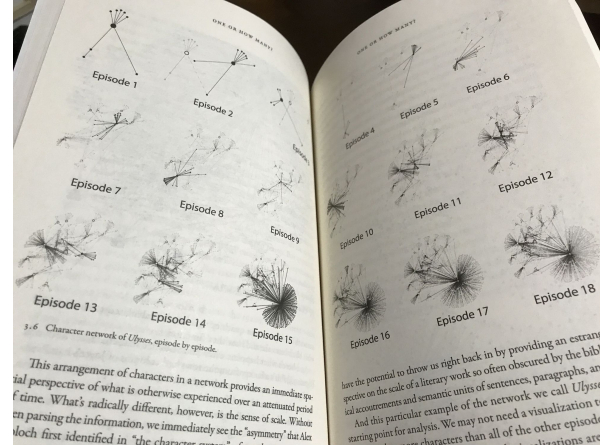
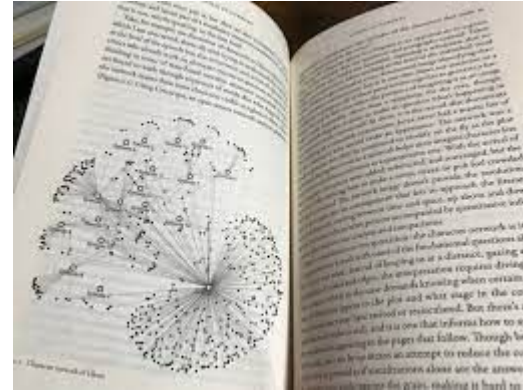
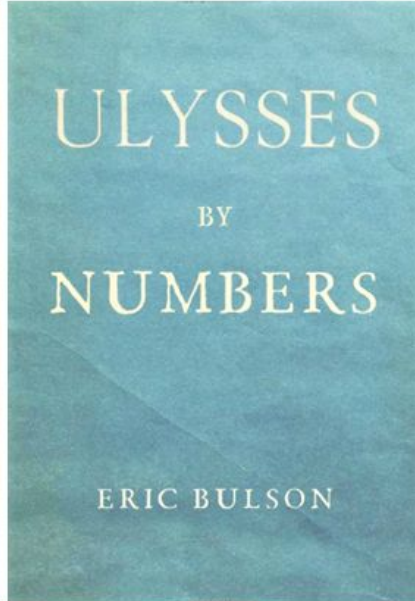
<https://docs.scala-lang.org/tour/tour-of-scala.html#:~:text=Scala%20is%20functional.every%20function%20is%20a%20value>.

- Spark is a distributed framework that uses Scala to send really large datasets to multiple nodes, perform various NLP actions and get the results back
- NLU or Natural Language Understanding is a subset dealing with computers trying to be human :)
- TF-IDF

TF-IDF is a statistical measure that evaluates how relevant a word is to a document in a collection of documents. This is done by multiplying two metrics: how many times a word appears in a document, and the inverse document frequency of the word across a set of documents.

<https://monkeylearn.com/blog/what-is-tf-idf/#:~:text=TF%20IDF%20is%20a%20statistical,across%20a%20set%20of%20documents>.

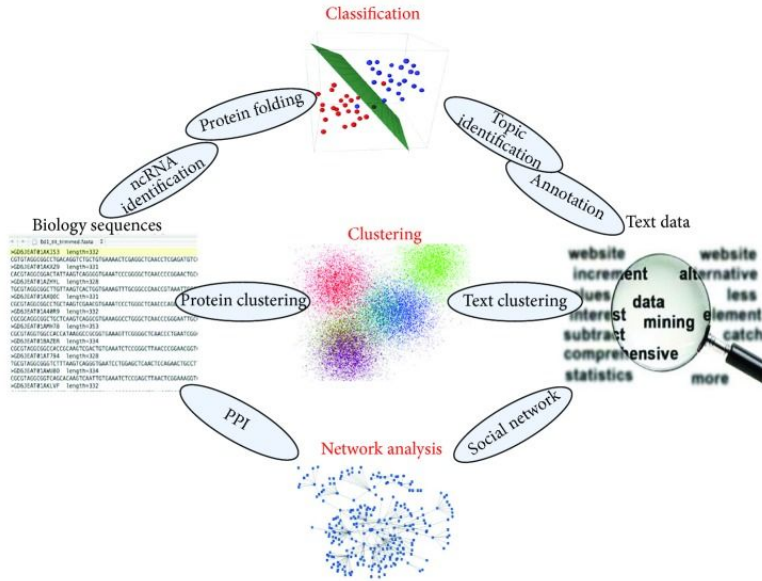
Other notable uses of NLP



<https://www.cgu.edu/news/2021/03/in-bulsons-new-book-understanding-joyce-is-as-easy-as-1-2-3/>

<https://lithub.com/how-to-read-ulysses-by-numbers/>

Other notable uses of NLP



BIOINFORMATICS	BIO-NLP	NLP, IR, IE
CASP	BioCreative	MUC
GASP	TREC Genomics	TREC
CAPRI	KDD cup	SEMEVAL
CAMDA	JNLPBA task	SENSEVAL
DREAM	LLL05 challenge	RTE

<https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4615216/>

<https://genomebiology.biomedcentral.com/articles/10.1186/gb-2008-9-s2-s1>

A whole book on applications of topic models:

Foundations and Trends® in Information Retrieval
Vol. XX, No. XX (2017) 1-154
© 2017
DOI: 10.1561/XXXXXXXXXX

now
the essence of knowledge

Applications of Topic Models

Jordan Boyd-Graber
Department of Computer Science, UMIACS, Language Science
University of Maryland¹
jbg@umiacs.umd.edu

Yuening Hu
Google, Inc.²
ynhu@google.com

David Mimno
Information Science
Cornell University
mimno@cornell.edu

Prep for hands on work:

Mac:

<https://brew.sh/>

```
/bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"
```

<https://formulae.brew.sh/cask/anaconda>

```
brew install --cask anaconda
```

conda install jupyter or conda install jupyterlab (explain the difference as well as JupyterHub)

<https://jupyter.org/install>

Might have to set the path: `export PATH=$PATH:/usr/local/anaconda3/bin`

```
jupyter-lab
```

LDA

Abstract

We describe *latent Dirichlet allocation* (LDA), a generative probabilistic model for collections of discrete data such as text corpora. LDA is a three-level hierarchical Bayesian model, in which each item of a collection is modeled as a finite mixture over an underlying set of topics. Each topic is, in turn, modeled as an infinite mixture over an underlying set of topic probabilities. In the context of text modeling, the topic probabilities provide an explicit representation of a document. We present efficient approximate inference techniques based on variational methods and an EM algorithm for empirical Bayes parameter estimation. We report results in document modeling, text classification, and collaborative filtering, comparing to a mixture of unigrams model and the probabilistic LSI model.

Journal of Machine Learning Research 3 (2003) 993-1022

Submitted 2/02; Published 1/03

Latent Dirichlet Allocation

David M. Blei

Computer Science Division
University of California
Berkeley, CA 94720, USA

Andrew Y. Ng

Computer Science Department
Stanford University
Stanford, CA 94305, USA

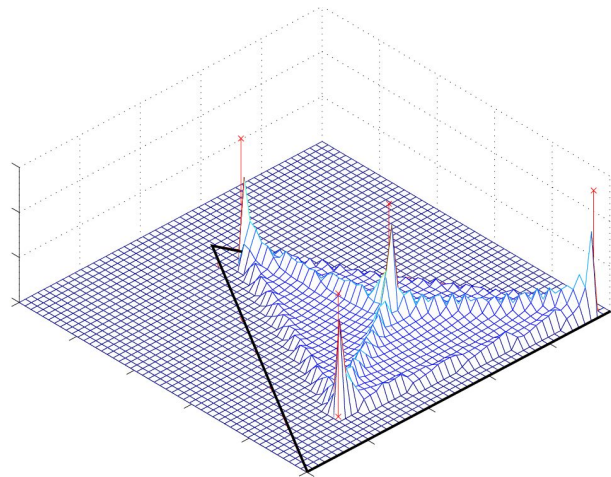
Michael I. Jordan

Computer Science Division and Department of Statistics
University of California
Berkeley, CA 94720, USA

BLEI@CS.BERKELEY.EDU

ANG@CS.STANFORD.EDU

JORDAN@CS.BERKELEY.EDU

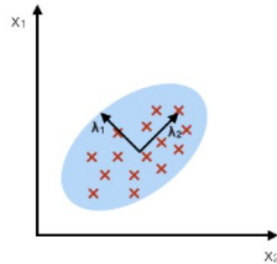


Editor: John Lafferty

Other Algorithms

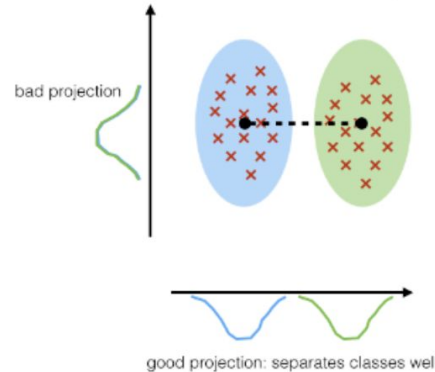
PCA:

component axes that maximize the variance



LDA:

maximizing the component axes for class-separation



LDA vs PCA

Other algorithms - tSNE and EM

- Word2Vec
- Doc2Vec
- gensim
- nltk
- sklearn

Building lexicons, vocabularies, word lists and dictionaries

Finance Example:

https://www3.nd.edu/~mcdonald/Word_Lists_files/Documentation/Documentation_LoughranMcDonald_MasterDictionary.pdf

The role of lexicons:

https://dl.acm.org/doi/pdf/10.1145/234173.234204?casa_token=Ur8QkKDweNwAAAAA:xFM2GuRydAI7BE9ZLBzeUXr33PCniVfsMiD5p_v_-YeVgjQ0zZGLE281i4iDOhniPa4mZQkzEsw5rNo

Sentiment:

<https://nlp.stanford.edu/projects/socialsent/>

Stop Words

In computing, **stop words** are **words** that are filtered out before or after the natural language data (text) are processed. While “**stop words**” typically refers to the most common **words** in a language, all-natural language processing tools don't use a single universal list of **stop words**.

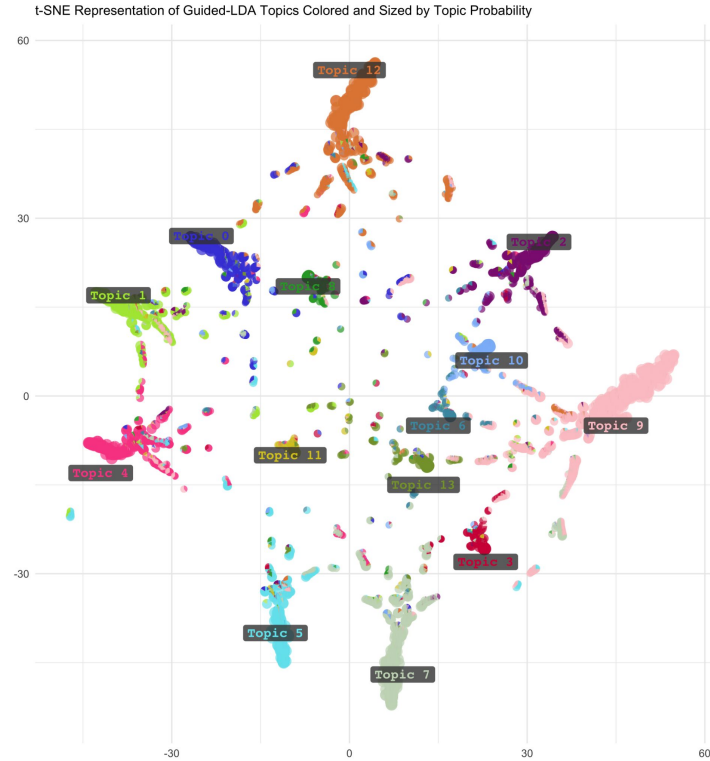
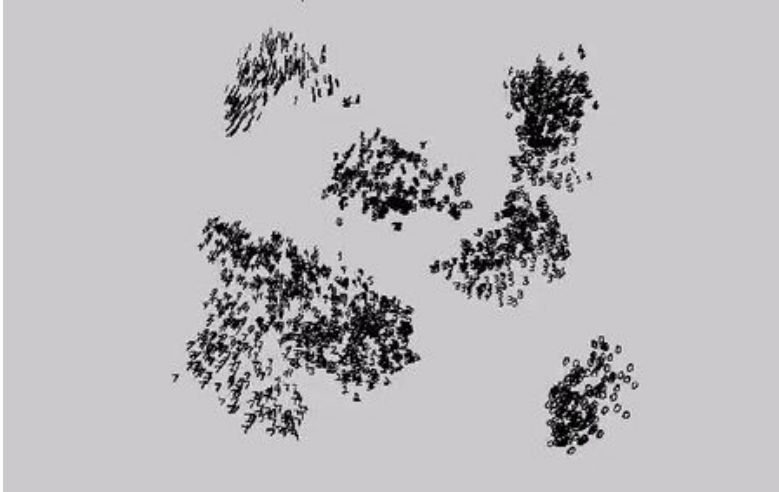
<https://medium.com/@saitejaponugoti/stop-words-in-nlp-5b248dadad47#:~:text=In%20computing%2C%20stop%20words%20are,universal%20list%20of%20stop%20words.>

We can observe that words like ‘this’, ‘is’, ‘will’, ‘do’, ‘more’, ‘such’ are removed from the tokenized vector as they are part of NLTK’s stopwords set. We can have a look at all such stop words for English by printing the stopwords.

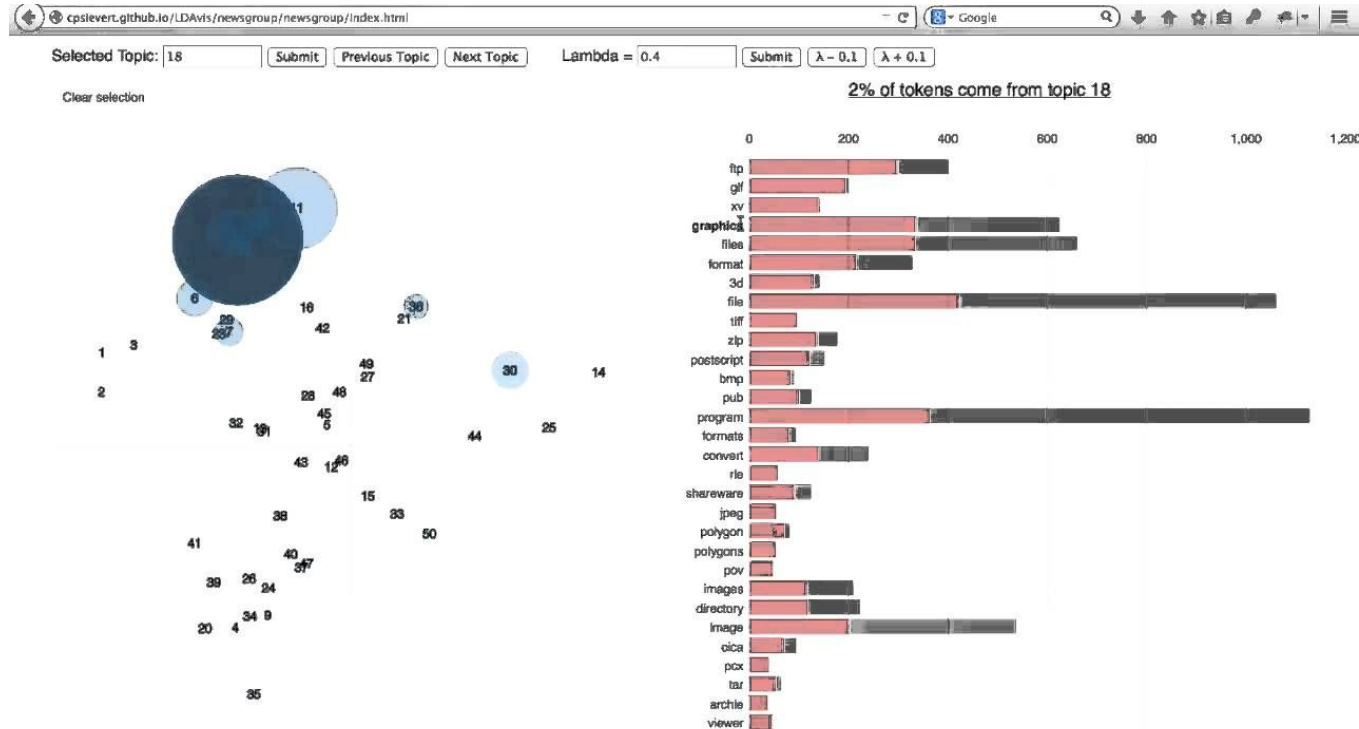
```
['I', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', 'you're', 'you've', 'you'll', 'you'd', 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his', 'himself', 'she', 'she's', 'her', 'hers', 'herself', 'it', 'it's', 'its', 'itself', 'they', 'them', 'their', 'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', 'that'll', 'these', 'those', 'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having', 'do', 'does', 'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', 'while', 'of', 'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'during', 'before', 'after', 'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under', 'again', 'further', 'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'each', 'few', 'more', 'most', 'other', 'some', 'such', 'no', 'nor', 'not', 'only', 'own', 'same', 'so', 'than', 'too', 'very', 's', 't', 'can', 'will', 'just', 'don', 'don't', 'should', 'should've', 'now', 'd', 'll', 'm', 'o', 're', 've', 'y', 'ain', 'aren', 'aren't', 'couldn', 'couldn't', 'didn', 'didn't', 'doesn', 'doesn't', 'hadn', 'hadn't', 'hasn', 'hasn't', 'haven', 'haven't', 'isn', 'isn't', 'ma', 'mightn', 'mightn't', 'mustn', 'mustn't', 'needn', 'needn't', 'shan', 'shan't', 'shouldn', 'shouldn't', 'wasn', 'wasn't', 'weren', 'weren't', 'won', 'won't', 'wouldn', 'wouldn't']
```

List of 179 NLTK stop words

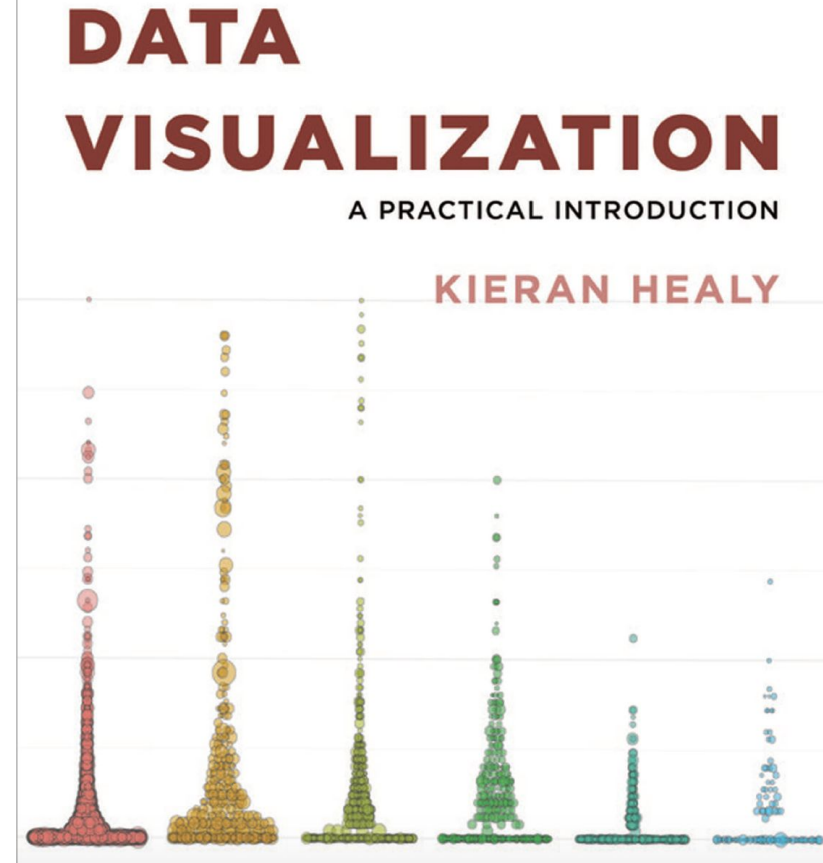
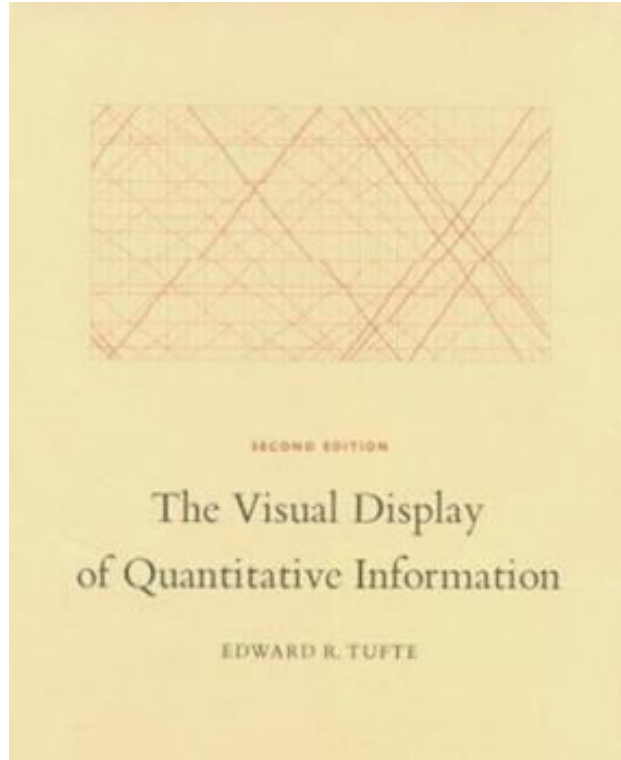
Visualization



LDAViz - Java-based visualization tool



Books about Visualization



<https://www.amazon.com/Data-Visualization-Introduction-Kieran-Healy/dp/0691181624>

Difference between Mallet and Spark

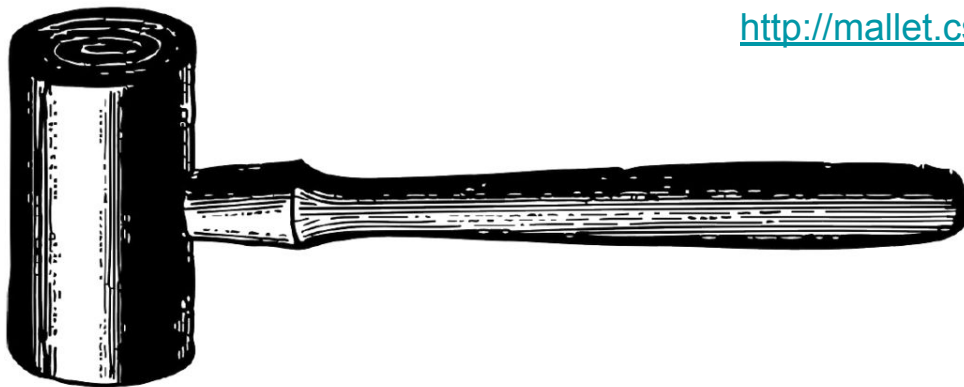
Mallet - single computer/small(ish) dataset

Spark - scales to dozens or hundreds of computers depending on the size of the dataset

1.5 TB, 130 million documents, 20,000 words each - > Spark

A book or a collection of news articles or a Twitter scrape - Mallet

Both Java/Scala-based though...



Getting started with MALLET

Xanda Schofield, xanda@cs.hmc.edu

About MALLET

MAchine Learning for LanguagE Toolkit: <http://mallet.cs.umass.edu/index.php>

Open-source library released under the Common Public License

Supports classification, sequence tagging, numerical optimization, and (most popularly) LDA topic modeling.

Reference: McCallum, Andrew Kachites. "MALLET: A Machine Learning for Language Toolkit." <http://mallet.cs.umass.edu>. 2002.

Some pros and cons

PROS

- Can be run as a standalone application
- Supports lots of standard pre-processing operations internally, including a good default tokenizer and stoplist for English
- Automatically supports several common topic model evaluations (held-out likelihood, topic coherence)
- Prints topics as it trains
- Uses MCMC for good-quality models
- Supports multiprocessing and sampling optimizations out-of-the-box to run fast

CONS

- Doesn't run on more than one machine
- Uses its own "sequence" data storage format which doesn't transfer
- Usually requires external scripting to pre-process the corpus before training (this is usually the case for these tools)
- Codebase is somewhat brittle due to optimizations for inference
- While GUIs exist for MALLET, none at time of writing work super well
- Sometimes produces surprising bugs on Windows systems

Installation

In general, you should install from Github (maintained by David Mimno), not the MALLET homepage. From the command line,

```
$ git clone https://github.com/mimno/Mallet.git
```

To build, you'll need to install Java and Apache Ant (for Java \geq 1.8):

Java: https://java.com/en/download/help/download_options.xml

Ant: <https://ant.apache.org/manual/install.html>

Installation

In the command line, navigate to the root directory of Mallet:

```
$ cd Mallet
```

Then, run ant (no arguments are necessary):

```
$ ant
```

If it outputs BUILD SUCCESSFUL, it should work.

If you run into subsequent memory issues, you can edit the execution script bin/mallet, which has variables limiting memory and number of processes.

```
12:17 [xanda@knuth:~]
13 % cd Mallet
12:17 [xanda@knuth:~/Mallet]
14 % ant
Picked up JAVA_TOOL_OPTIONS: -Xmx24g -Xms4g
Buildfile: /mnt/home/xanda/Mallet/build.xml

init:
[mkdir] Created dir: /mnt/home/xanda/Mallet/class
[copy] Copying 1 file to /mnt/home/xanda/Mallet/class
[mkdir] Created dir: /mnt/home/xanda/Mallet/dist
[copy] Copying 1 file to /mnt/home/xanda/Mallet/dist

compile:
[javac] Compiling 617 source files to /mnt/home/xanda/Mallet/class
[javac] Note: Some input files use or override a deprecated API.
[javac] Note: Recompile with -Xlint:deprecation for details.
[javac] Note: Some input files use unchecked or unsafe operations.
[javac] Note: Recompile with -Xlint:unchecked for details.

BUILD SUCCESSFUL
Total time: 12 seconds
```

Installation

Initially, your operating system might not know where MALLET is installed. To run Mallet, you would use the whole path, e.g. (assuming you installed in your home directory):

```
$ ~/Mallet/bin/mallet
```

If you want to just be able to run `mallet` without the path, you'll want to either add the `Mallet/bin` subdirectory to your operating system's `PATH` variable or put in an alias to the executable. If you're running a bash terminal, this will work:

```
$ echo 'alias mallet="~/Mallet/bin/mallet"' >> ~/.bashrc
```

```
$ source ~/.zshrc
```

Topic Model Steps

Importing data:

\$ `mallet import-file` or \$ `mallet import-dir`

(Optional) Filtering corpus:

\$ `mallet prune`

Training topic model:

\$ `mallet train-topics`

Evaluating topic model:

\$ `mallet evaluate-topics`

Importing Data

Mallet imports documents into a special format for efficient processing. This format stores each document with a unique identifier and, optionally, a label (e.g. year, author). To import your data, you should either store one document per file or have one shared tab-separated value file with all of your documents.

If your documents are in fairly well-processed existing units with one document in each file already, then importing by directory is likely a good option.

`mallet import-dir --input [directory]` expects every document to be in a separate file in directory without any stored metadata. Each document's identifier as its filename.

Gotchas: If there is metadata stored in each document, it will be treated as text!

Importing Data

If you want to do nontrivial pre-processing (tokenizing, splitting documents into pieces, adding labels, joining ngrams), it may be easier to use the single-file import option.

`mallet import-file --input [file].tsv` expects every document to be represented as a row in a tab-separated value (tsv) file with the following format:
[document identifier]<tab>[label]<tab>[document text]<newline>

Gotchas: It's definitely important that no newlines exist in the document text. It's also a good idea to avoid having spaces in your document identifier or label. Make sure document identifiers are unique (e.g. if you split a document into pieces, each should have its own name.) If you don't care about labels, put whatever you want there.

Importing Data

Flags useful for import:

- `--keep-sequence`: Even though LDA topic models are bag-of-words models, MALLET uses the sequence of tokens as part of its processing. You'll need to use `--keep-sequence` to store that information, or it will just store word count vectors.
- `--remove-stopwords`: Automatically removes a standard list of English stopwords (stored in `stoplists/en.txt`).
- `--stoplist-file [filename]`: remove a custom list of stopwords from the file. Expects a `.txt`-format file with one word per line.
- `--token-regex [regex]`: Specifies a Java-style regular expression for parsing tokens (separating words in the file). Default is `\p{L}[\p{L}\p{P}]+\p{L}` (words with three or more letter characters with optional punctuation inside).

Importing Data

Sample command:

```
$ mallet import-file --input segmented_documents.tsv --output  
segmented_documents_special.seq --keep-sequence  
--stoplist-file special_stops.txt
```

This imports the tab-separated file `segmented_documents.tsv` while removing the stopwords listed in `special_stops.txt`. The file Mallet needs will be saved in `segmented_documents_special.seq`. (Tip: filenames should indicate any unusual processing steps. To be safe, keep a log of the commands you run.) If you want more fancy functionality (regex-based or extended stoplists), run `mallet import-file --help` for additional options.

Filtering corpus

There are two principal kinds of filtering you might be interested in doing in your corpus: reducing the list of words in your vocabulary, and splitting text into training and test texts. `mallet prune` supports both of these.

To filter less useful words from a vocabulary, useful flags to set (with reasonable values) are:

- `--prune-document-freq 10`: remove words in fewer than 10 distinct documents
- `--prune-count 20`: remove words that appear fewer than 20 times
- `--min-idf 3`: remove words that show up in more than 1 out of 3 documents.

To split the corpus into multiple files for training, validation, and testing, you can use flags to specify proportions of documents and where to write the new sequence files, e.g.

```
--training-portion 0.9 --validation-portion 0.1 --training-file  
[file-train].seq --testing-file [file-test].seq
```

Running a topic model

To run (or more formally, “infer”) a topic model, use the train-topics command.

Let’s take apart a chunky train-topics command:

```
mallet train-topics --input [filename].seq --num-topics 50 --optimize-interval 10  
--num-iterations 5000 --output-state [filename].txt.gz --evaluator-filename  
[filename].evaluator --output-topic-keys [filename].keys --num-top-words 50  
--diagnostics-file [filename]_diag.xml --topic-word-weights-file  
[filename].wordweights.txt --output-doc-topics [filename].doctopics.txt
```

Running a topic model

To run (or more formally, “infer”) a topic model, use the train-topics command.

Let’s take apart a chunky train-topics command:

```
mallet train-topics --input [filename].seq --num-topics 50 --optimize-interval 10
--num-iterations 5000 --output-state [filename].txt.gz --evaluator-filename
[filename].evaluator --output-topic-keys [filename].keys --num-top-words 50
--diagnostics-file [filename]_diag.xml --topic-word-weights-file
[filename].wordweights.txt --output-doc-topics [filename].doctopics.txt
--input [filename].seq: source documents are stored in filename.seq
--num-topics 50: 50 topics will be inferred in the model.
```

Running a topic model

To run (or more formally, “infer”) a topic model, use the train-topics command.

Let’s take apart a chunky train-topics command:

```
mallet train-topics --input [filename].seq --num-topics 50 --optimize-interval 10
--num-iterations 5000 --output-state [filename].txt.gz --evaluator-filename
[filename].evaluator --output-topic-keys [filename].keys --num-top-words 50
--diagnostics-file [filename]_diag.xml --topic-word-weights-file
[filename].wordweights.txt --output-doc-topics [filename].doctopics.txt
```

--optimize-interval 10: let topic sizes adjust with respect to each other. Every 10 iterations after the first 100, the topic prior (alpha) will infer how much to adjust.

--num-iterations 5000: run for 5000 total MCMC iterations (usually 1000-5000 is good).

Running a topic model

To run (or more formally, “infer”) a topic model, use the train-topics command.

Let’s take apart a chunky train-topics command:

```
mallet train-topics --input [filename].seq --num-topics 50 --optimize-interval 10
```

```
--num-iterations 5000 --output-state [filename].txt.gz --evaluator-filename
```

```
[filename].evaluator --output-topic-keys [filename].keys --num-top-words 50
```

```
--diagnostics-file [filename]_diag.xml --topic-word-weights-file
```

```
[filename].wordweights.txt --output-doc-topics [filename].doctopics.txt
```

--output-state...: save a gzip-compressed file mapping each token in the corpus with its final topic assignment. This file & the .seq file is enough to reconstruct the entire model.

--evaluator-filename...: save a special file to help compute held-out likelihood.

Running a topic model

To run (or more formally, “infer”) a topic model, use the train-topics command.

Let’s take apart a chunky train-topics command:

```
mallet train-topics --input [filename].seq --num-topics 50 --optimize-interval 10
```

```
--num-iterations 5000 --output-state [filename].txt.gz --evaluator-filename
```

```
[filename].evaluator --output-topic-keys [filename].keys --num-top-words 50
```

```
--diagnostics-file [filename]_diag.xml --topic-word-weights-file
```

```
[filename].wordweights.txt --output-doc-topics [filename].doctopics.txt
```

`--output-topic-keys...:` save the top probability keywords (in decreasing order of probability) from each topic into a text file

`--num-top-words 50` save a special file to help compute held-out likelihood.

Running a topic model

To run (or more formally, “infer”) a topic model, use the train-topics command.

Let’s take apart a chunky train-topics command:

```
mallet train-topics --input [filename].seq --num-topics 50 --optimize-interval 10  
--num-iterations 5000 --output-state [filename].txt.gz --evaluator-filename  
[filename].evaluator --output-topic-keys [filename].keys --num-top-words 50  
--diagnostics-file [filename]_diag.xml --topic-word-weights-file  
[filename].wordweights.txt --output-doc-topics [filename].doctopics.txt
```

`--diagnostics-file ...`: saves an XML file with various metrics and information per topic.

Other XML evaluations can be output; this one reports topic coherence (from Mimno et al. 2011, “Optimizing semantic coherence...”)

Running a topic model

To run (or more formally, “infer”) a topic model, use the train-topics command.

Let’s take apart a chunky train-topics command:

```
mallet train-topics --input [filename].seq --num-topics 50 --optimize-interval 10  
--num-iterations 5000 --output-state [filename].txt.gz --evaluator-filename  
[filename].evaluator --output-topic-keys [filename].keys --num-top-words 50  
--diagnostics-file [filename]_diag.xml --topic-word-weights-file  
[filename].wordweights.txt --output-doc-topics [filename].doctopics.txt
```

--topic-word-weights-file...: lists the “weight” for each topic-word pair. These are neither counts nor proportions of words, but can be normalized by words or topics.

--output-doc-topics: lists the topic proportions of each document.

Running a topic model

If you ever train a model but forget to write out some outputs you need, you can use the state file and the `--no-inference` flag to reload the topic, run 0 iterations (i.e., not change the model) and then use other flags to write out other outputs:

```
mallet train-topics --input [filename].seq --num-topics 50 --no-inference  
--input-state [filename].txt.gz...
```

For information on other flags (especially for other kinds of output files), run

```
$ mallet train-topics --help
```

Evaluating topics

If you use held-out documents to evaluate how good a topic model is, you can use an evaluator (from the `--evaluator-filename` flag) to estimate this quantity.

(This isn't an exact process; see Wallach et al. 2009, "Evaluation methods for topic models"). This will write the estimated log probability of the whole corpus (the sum of the log probabilities of each token) into `[file].prob`.

```
mallet evaluate-topics --input [test-file].seq --evaluator  
[file].evaluator --output-prob [file].prob
```

More resources

On GitHub:

<https://github.com/mimno/Mallet/tree/master/stoplists>

Getting Started with Topic Modeling and MALLET:

<https://programminghistorian.org/en/lessons/topic-modeling-and-mallet>

Optimizing Semantic Coherence in Topic Models:

<http://dirichlet.net/pdf/mimno11optimizing.pdf>

Evaluation Methods for Topic Models:

<https://mimno.infosci.cornell.edu/papers/wallach09evaluation.pdf>

Sp **03: Intro Spark Apps**

Spark Essentials

lecture/lab: 45 min

Scraping

- Well structured URLS vs Not
- OCR artifacts
- HTML/CSS junk - BeautifulSoup etc
- PDF to TXT conversion
- Cloud limitations (5,000 docs)
- Data viz tools: Gradient, Alteryx, John Snow Labs, Databricks etc



The screenshot shows the U.S. Securities and Exchange Commission (SEC) Edgar Company Filings search interface. At the top, there is a search bar labeled "Search SEC.gov" and a navigation menu with links: ABOUT, DIVISIONS, ENFORCEMENT, REGULATION, EDUCATION, FILINGS, and NEWS. Below the navigation bar, the page is titled "EDGAR | Company Filings". On the left, there is a sidebar with links: "EDGAR Search and Access", "Latest Filings", "Company Filings" (highlighted), "Mutual Funds", "Variable Insurance Products", "Daily Filings by Type", "Boolean Archive Search", "EDGAR Full Text Search", "CIK Lookup", and "Confidential Treatment". The main content area features a "Company and Person Lookup" section with a search input field labeled "Name, ticker symbol, or CIK" and a "SEARCH" button. Below the input field is a "More Options" link. To the right of the search input is a "How to Use this Search?" section with instructions: "Enter name, ticker or CIK into the single search field. Suggestions as you type link directly to filings." Below the search input is a "Guides" section with links: "How to Research Public Companies", "Form Types", and "Investor.gov". To the right of the guides is a "Search Tools" section with links: "EDGAR Full Text Search", "CIK Lookup", and "Save Your Search".

<https://www.sec.gov/edgar/searchedgar/companysearch.html>

Data Cleaning

receipt-home-depot.jpg
receipt-mels.jpg
receipt-sunshine.jpg
receipt-tadish.jpg

Welcome to Mel's

Check #: 0001 12/20/11
Server: Josh F 4:38 PM
Table: 7/1 Guests: 2

2 Beef Burgr (@9.95/ea) 19.90
SIDE: Fries
1 Bud Light 3.79
1 Bud 4.50

Sub-total 28.19
Sales Tax 2.50
TOTAL 30.69

Balance Due 30.69

Thank you for your patronage!

1425788830...depot.jpg.txt
1425788831...t-mels.jpg.txt
1425788834...shine.jpg.txt
1425788834...tadish.jpg.txt

welcome to MeT's

Check #: 0001 12/20/11
Server: Josh F 4:38 PM
Table: 7/1 Guests: 2
2 Beef Burgr (@9.95/ea) 19.90
SIDE: Fries
1 Bud Light 3.79
1 Bud 4.50
Sub-total 28.19
SaTes Tax ____mg.\$g
TOTAL 30.69
eai;r};;"13;;
"Tédfee

Thank you for your patronage!

<https://medium.com/nanonets/a-comprehensive-guide-to-ocr-with-tesseract-opencv-and-python-fd42f69e8ca8>

<https://towardsdatascience.com/a-gentle-introduction-to-ocr-ee1469a201aa>

Frameworks

Databricks <https://databricks.com/>

Snow Labs NLP <https://www.johnsnowlabs.com/>

Gradient <https://gradient.paperspace.com/>

Alteryx <https://www.alteryx.com/>

<https://medium.com/@ODSC/10-notable-frameworks-for-nlp-ce8c4196bfd6>

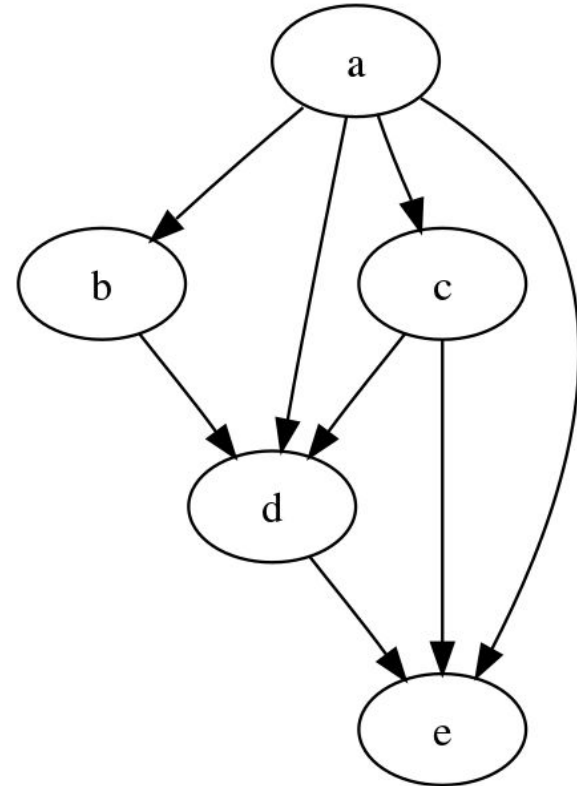
Jupyter Doc Demo

If there is time...

DAGs - Directed Acyclical Graphs

<https://towardsdatascience.com/understanding-nlp-how-ai-understands-our-languages-77601002cffc>

https://en.wikipedia.org/wiki/Directed_acyclic_graph



NeuraLink

<https://www.iflscience.com/technology/elon-musk-claims-neuralink-could-render-human-language-obsolete-in-five-to-ten-years/>

Speaking on the [Joe Rogan Experience](#) podcast – yes, the same one he famously smoked weed on, which resulted in a [\\$5 million NASA investigation](#) into the suitability and professionalism of SpaceX as a government contractor – the SpaceX boss claimed that his company Neuralink could allow brain-to-brain communication in as little as five years, thereby overcoming the need to go to the trouble of actually talking.

It's worth pointing out that the primary objective of [Neuralink](#) is not to eliminate speech. The company is developing a device that links people's brains to a computer in order to treat brain injuries and traumas, using tiny threads that are implanted directly into the relevant parts of the brain.