NLP Lecture 1

15th July 2021 -- MTECH

About Me

Ekta Shah

- Senior Machine Learning Engineer at Quantiphi
- Data Scientist at Viacom18
- Data Science Intern at General Mills
- Software Developer at BNP Paribas
- Passion for Teaching

Agenda for Today

- Course Overview - Syllabus
- Introduction to NLP
- Machine Learning and NLP
- Argmax function
- Syntactic Collocation
- Data Pipeline

Introduction to NLP

- **Wiki: Natural language processing** (**NLP**) is a field of computer science, artificial intelligence, and computational linguistics concerned with the interactions between computers and human (**natural**) languages.

Go beyond the keyword matching

- Identify the structure and meaning of words, sentences, texts and conversations
- Deep understanding of broad language
- NLP is all around us

Why is NLP interesting?

Languages involve many human activities− Reading, writing, speaking, listening

- Voice can be used as an user interface in many

applications− Remote controls, virtual assistants like siri,...

- NLP is used to acquire insights from massive

amount of textual data− E.g., hypotheses from medical, health reports

- NLP has many applications
- NLP is hard!

Why is NLP hard?

I shot an elephant in my pajamas.

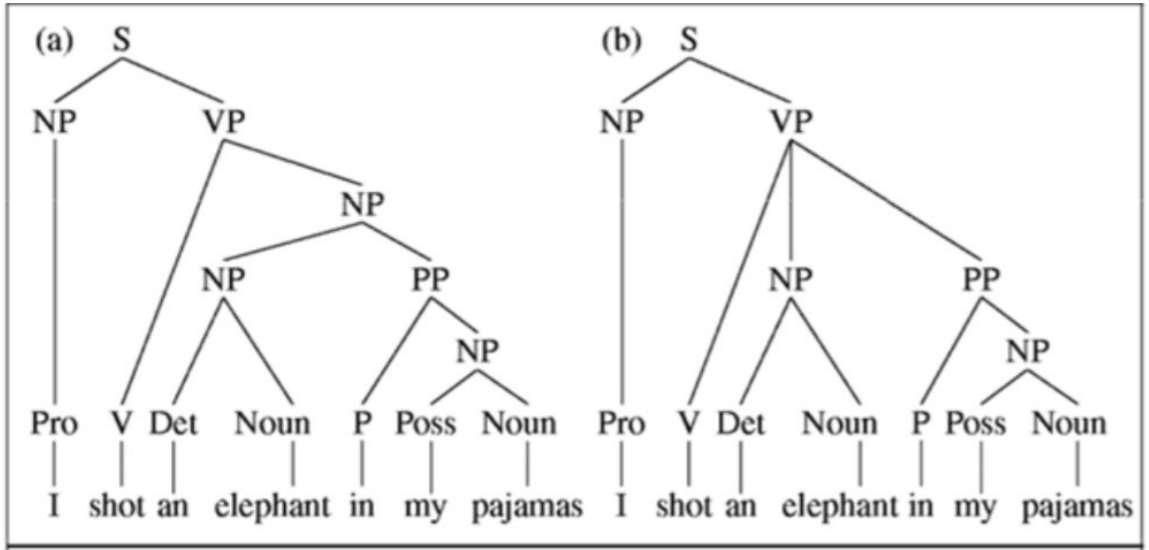


Figure 10.11 Two parse trees for an ambiguous sentence. Parse (a) corresponds to the humorous reading in which the elephant is in the pajamas, parse (b) to the reading in which Captain Spaulding did the shooting in his pajamas.

Why is NLP hard?

Natural languages are highly ambiguous at all levels

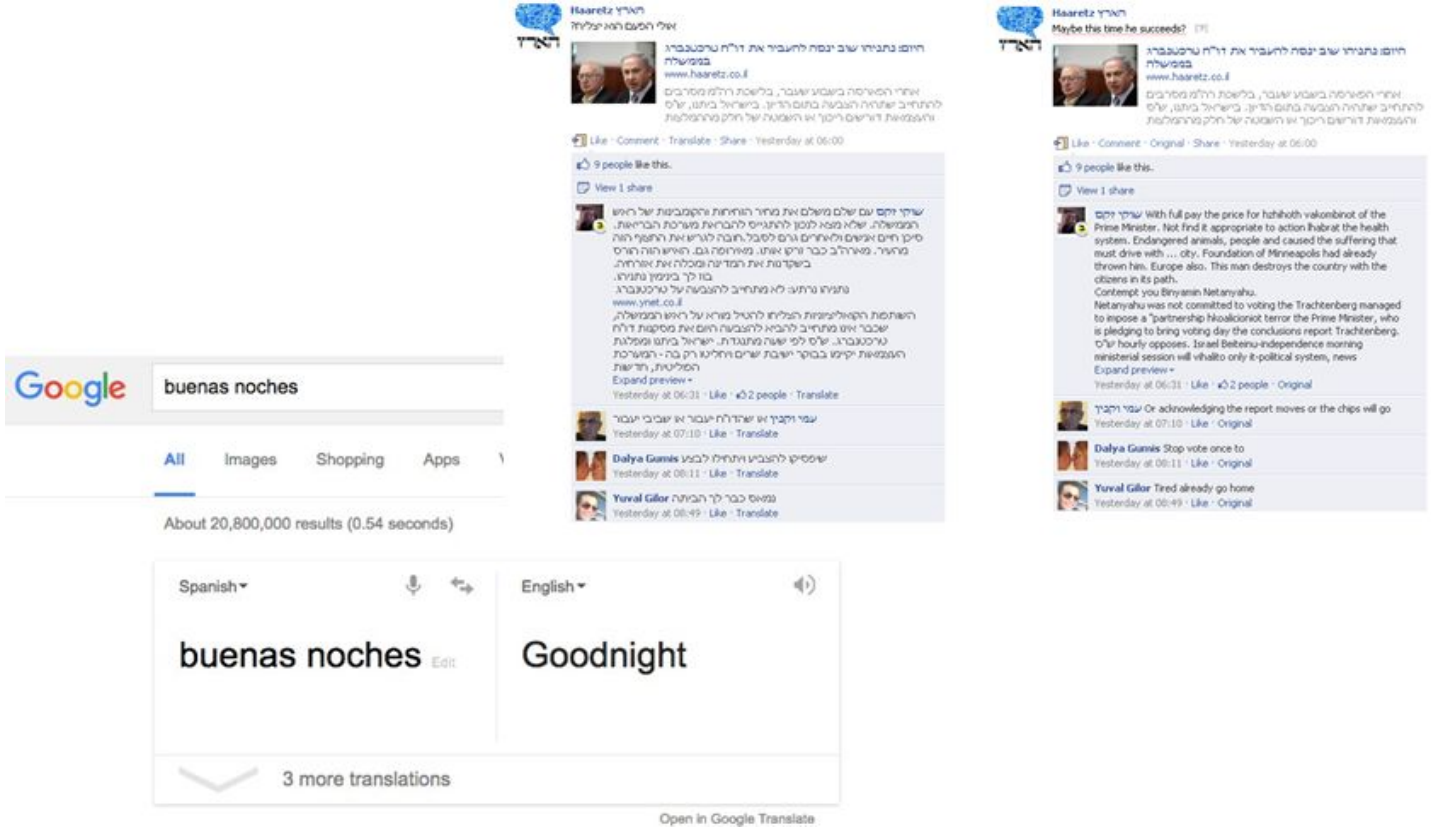
- Lexical (word's meaning)
- Syntactic
- Semantic
- Discourse

- Natural languages are fuzzy
- Natural languages involve reasoning about the world..E.g., It is unlikely that an elephant wears a pajamas

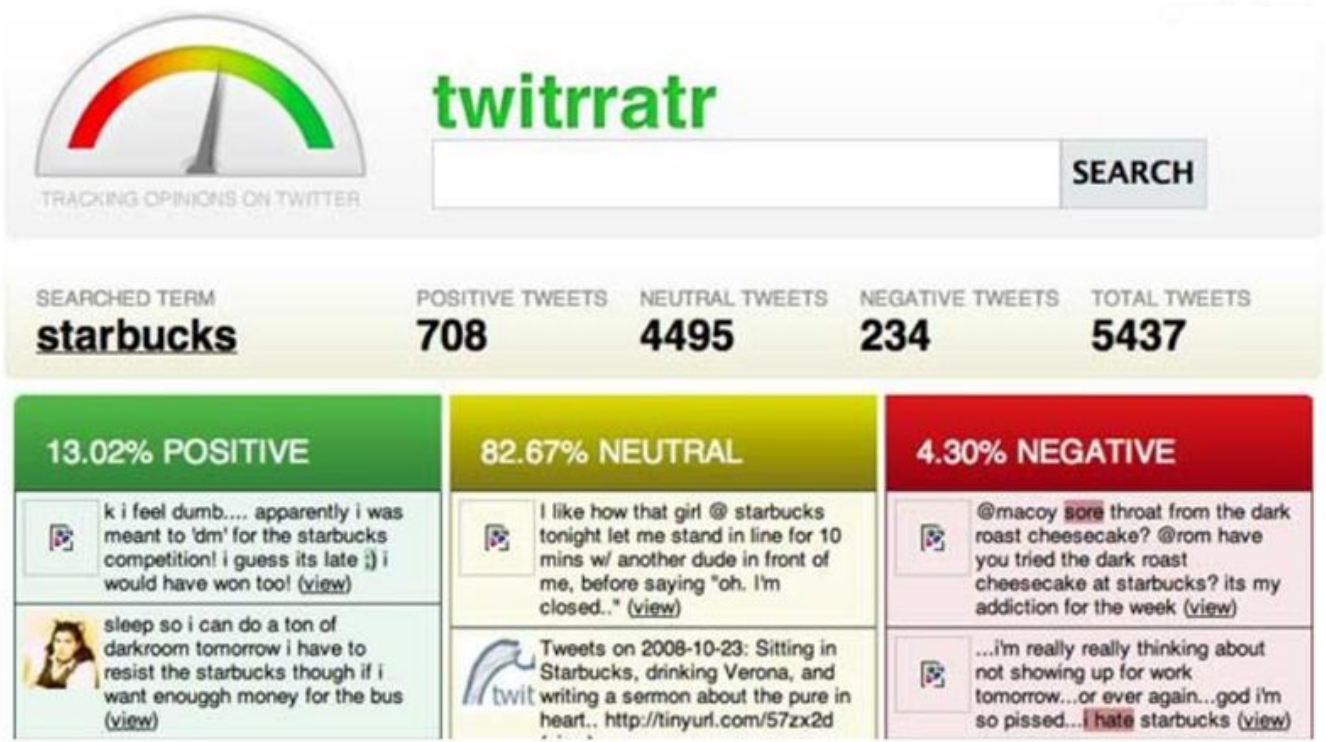
Applications of NLP

Machine

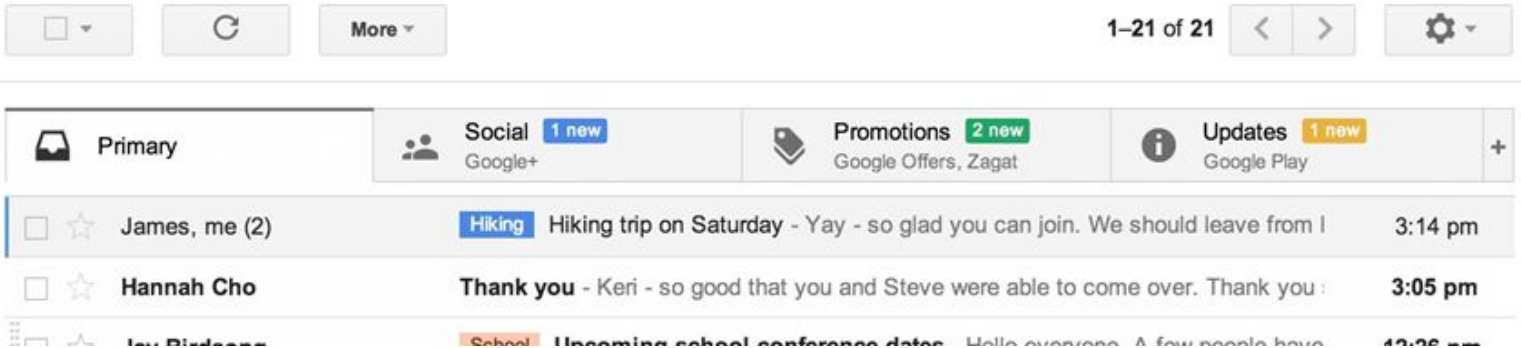
Translation



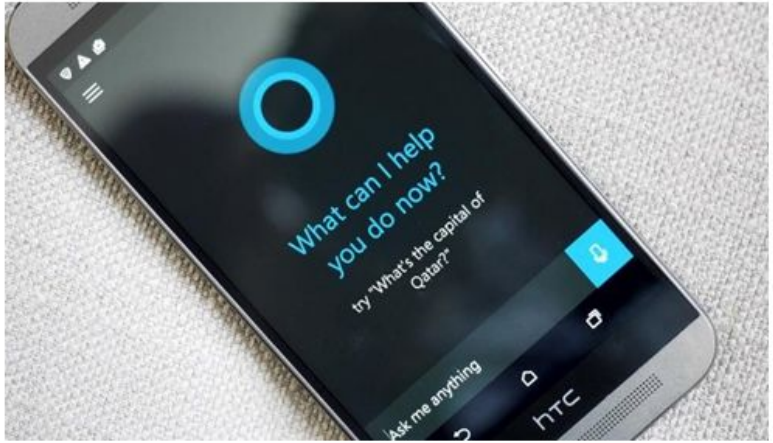
Sentiment/Opinion Analysis



Text Classification



Digital Personal Assistant



- Semantic parsing – understand tasks
- Entity linking – “my wife” = “Kellie” in the phone book

Brief history of NLP

Foundational Insights: 1940s and 1950s

− Two foundational paradigms

● Automaton

● Probabilistic/Information-Theoretic models

● The two camps: 1957-1970

− Symbolic paradigm: the work of Chomsky and others on

formal language theory and generative syntax (1950s ~mid 1960s)

- − Stochastic paradigm In departments of statistics

Brief history of NLP

Four paradigms: 1970-1983, explosion in

research in speech and language processing

- Stochastic paradigm
- Logic-based paradigm
- Natural language understanding
- Discourse modeling paradigm

● Empiricism and Finite State Models Redux: 1983-1993

Brief history of NLP

The Fields Comes Together: 1994-1999

− Probabilistic and data-driven models had become

quite standard

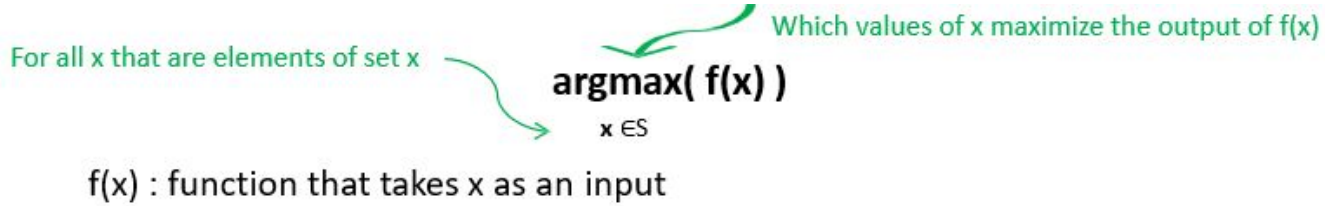
● The Rise of Machine Learning: 2000-now

− Large amount of spoken and textual data become available

− Widespread availability of high-performance computing systems

Argmax function

- The argmax() equation returns the largest values that maximize a function.



- Condition is met when the output of the function for at least one element xj that is greater than or equal to the output of for all x ∈ S.

The output for x_j is greater than or equal to the output for all elements x_i in set S

$$f(x_j) \geq f(x_i),\ x \in S$$

Bayesian Decision Theory

Bayes Theorem : Given the random variables A and B,

$$P(A \mid B) = \frac{P(A)P(B \mid A)}{P(B)}$$

$P(A \mid B)$ Posterior probability

$P(A)$ Prior probability

$P(B \mid A)$ Likelihood

Speech Recognition as a Noisy Channel

Speaker has word sequence W

W is articulated as acoustic sequence A

• This process introduces noise:

– variation in pronunciation

– acoustic variation due to microphone etc.

• Bayes theorem gives us:

$$\begin{aligned}\overline{W} &= \arg\max_W P(W|A) \\ &= \arg\max_W \underbrace{P(A|W)}_{likelihood}\underbrace{P(W)}_{prior}\end{aligned}$$

Collocations

- Collocation or lexical collocation means two or more words co-occur in a sentence more frequently than by chance. A collocation is an expression that forms a specific meaning. It may be noun phrase like large house, verbal phrase like pick up, idioms, cliches or technical terms.
- For example, He is known for his fair and square dealings and everybody trusts his work.
- Here fair and square means honest but if we take the individual words though the word fair gives somewhat closer meaning as it means just the word square confuses us. So instead of taking individual words one should take the collocation fair and square and find meaning. It shows that collocations play a key role in understanding sentences.

Collocations - contd

- It is a phrase consisting of more than one word but these words more commonly co-occur in a given context than its individual word parts. For example, in a set of hospital related documents, the phrase ‘CT scan’ is more likely to co-occur than do ‘CT’ and ‘scan’ individually. ‘CT scan’ is also a meaningful phrase.
- The two most common types of collocation are bigrams and trigrams. Bigrams are two adjacent words, such as ‘CT scan’, ‘machine learning’, or ‘social media’. Trigrams are three adjacent words, such as ‘out of business’, or ‘Proctor and Gamble’.

Uses of Collocation

a) Keyword extraction: identifying the most relevant keywords in documents to assess what aspects are most talked about

b) Bigrams/Trigrams can be concatenated (e.g. social media -> social_media) and counted as one word to improve insights analysis, topic modeling, and create more meaningful features for predictive models in NLP problems

Principal Approaches to Finding Collocations

- Selection of collocations by frequency
- PMI(Pointwise Mutual Information)
- Hypothesis testing (t-test and chi square)

Counting frequencies of adjacent words with part of speech filters

- The simplest method is to rank the most frequent bigrams or trigrams:

```
bigram_freq = bigramFinder.ngram_fd.items()

bigramFreqTable = pd.DataFrame(list(bigram_freq),
columns=['bigram','freq']).sort_values(by='freq', ascending=False)
```

However, a common issue with this is adjacent spaces, stop words, articles, prepositions or pronouns are common and are not meaningful:

	bigram	freq
0	(, -pron-)	29078
1	(, the)	21918
2	(-pron-, be)	18353
3	(the, room)	8898
4	(-pron-, have)	8377

	trigram	freq
0	(, -pron-, be)	6267
1	(the, room, be)	4411
2	(, the, room)	3349
3	(, -pron-, have)	2681
4	(the, staff, be)	2641

To fix this, we filter out for collocations not containing stop words and filter for only the following structures:

Bigrams: (Noun, Noun), (Adjective, Noun)

Trigrams: (Adjective/Noun, Anything, Adjective/Noun)

Pointwise Mutual Information (PMI)

The Pointwise Mutual Information (PMI) score for bigrams is:

$$PMI(w^1, w^2) = log_2 \frac{P(w^1, w^2)}{P(w^1)P(w^2)}$$

For trigrams:

$$PMI(w^1, w^2, w^3) = log_2 \frac{P(w^1, w^2, w^3)}{P(w^1)P(w^2)P(w^3)}$$

PMI

The main intuition is that it measures how much more likely the words co-occur than if they were independent. However, it is very sensitive to rare combination of words. For example, if a random bigram ‘abc xyz’ appears, and neither ‘abc’ nor ‘xyz’ appeared anywhere else in the text, ‘abc xyz’ will be identified as highly significant bigram when it could just be a random misspelling or a phrase too rare to generalize as a bigram. Therefore, this method is often used with a frequency filter.

For further reference on collocations:

https://medium.com/@nicharuch/collocations-identifying-phrases-that-act-like-individual-words-in-nlp-f58a93a2f84a

Terminology

Corpus:

In linguistics and NLP, corpus (literally Latin for body) refers to a collection of texts.

Lexicon:

The definition of a lexicon is a dictionary or the vocabulary of a language, a people or a subject. An example of lexicon is YourDictionary.com. An example of lexicon is a set of medical terms.

Bag of Words

- Bag of Words (BOW) is a method to extract features from text documents. These features can be used for training machine learning algorithms. It creates a vocabulary of all the unique words occurring in all the documents in the training set.

1. "John likes to watch movies. Mary likes movies too."

2. "John also likes to watch football games."

1. ['John', 'likes', 'to', 'watch', 'movies.', 'Mary', 'likes', 'movies', 'too.']

2. ['John', 'also', 'likes', 'to', 'watch', 'football', 'games']

1. {"John":1,"likes":2,"to":1,"watch":1,"movies":2,"Mary":1,"too":1}

2. {"John":1,"also":1,"likes":1,"to":1,"watch":1,"football":1, "games":1}

{"John":2,"likes":3,"to":2,"watch":2,"movies":2,"Mary":1,"too":1,
"also":1,"football":1,"games":1}

The length of the vector will always be equal to vocabulary size. In this case the vector length is 11.In order to represent our original sentences in a vector, each vector is initialized with all zeros — [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

John likes to watch movies. Mary likes movies too.[1, 2, 1, 1, 2, 1, 1, 0, 0, 0]

John also likes to watch football games.[1, 1, 1, 1, 0, 0, 0, 1, 1, 1]

Limitations of BOW

Semantic meaning: the basic BOW approach does not consider the meaning of the word in the document. It completely ignores the context in which it's used. The same word can be used in multiple places based on the context or nearby words.

Vector size: For a large document, the vector size can be huge resulting in a lot of computation and time. You may need to ignore words based on relevance to your use case.

TF-IDF(Term Frequency-Inverse Document Frequency)

- Tf-idf stands for term frequency-inverse document frequency, and the tf-idf weight is a weight often used in information retrieval and text mining.
- This weight is a statistical measure used to evaluate how important a word is to a document in a collection or corpus.
- The importance increases proportionally to the number of times a word appears in the document but is offset by the frequency of the word in the corpus.
- Variations of the tf-idf weighting scheme are often used by search engines as a central tool in scoring and ranking a document's relevance given a user query.

TF

Term Frequency, which measures how frequently a term occurs in a document. Since every document is different in length, it is possible that a term would appear much more times in long documents than shorter ones. 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:

TF(t) = (Number of times term t appears in a document) / (Total number of terms in the document).

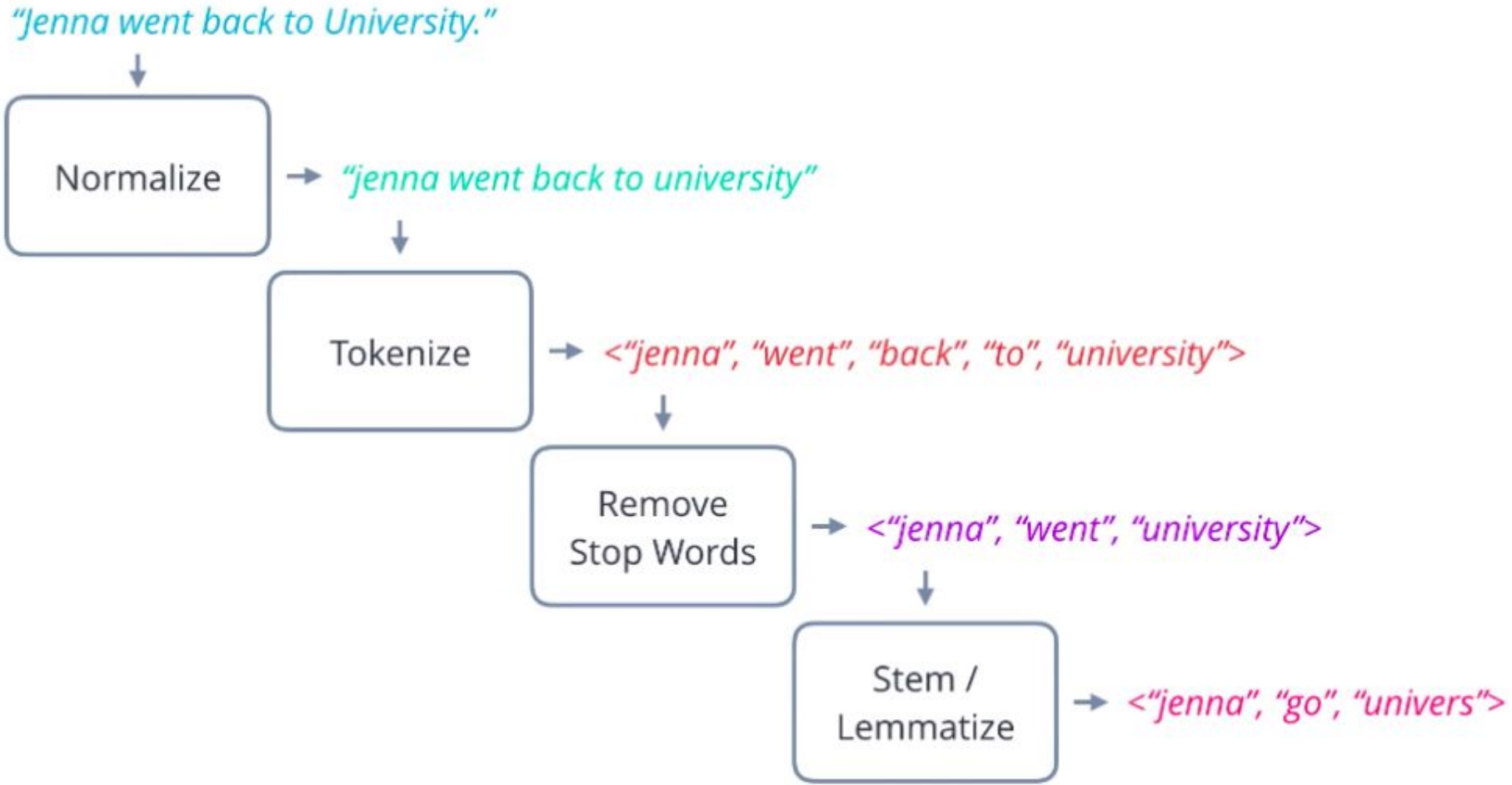
IDF

Inverse Document Frequency, which measures how important a term is. While computing TF, all terms are considered equally important. However it is known that certain terms, such as "is", "of", and "that", may appear a lot of times but have little importance. Thus we need to weigh down the frequent terms while scale up the rare ones, by computing the following:

IDF(t) = log_e(Total number of documents / Number of documents with term t in it).

Example

Consider a document containing 100 words wherein the word cat appears 3 times. The term frequency (i.e., tf) for cat is then (3 / 100) = 0.03. Now, assume we have 10 million documents and the word cat 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.


"Jenna went back to University."
Normalize
"jenna went back to university"
Tokenize
<"jenna", "went", "back", "to", "university">
Remove Stop Words
<"jenna", "went", "university">
Stem / Lemmatize
<"jenna", "go", "univers">"

Tokenization

- *Tokenization is breaking a text chunk in smaller parts. Whether it is breaking Paragraph in sentences, sentence into words or word in characters.*
- *https://medium.com/data-science-in-your-pocket/tokenization-algorithms-in-natural-language-processing-nlp-1fceab8454af*
- *https://analyticsindiamag.com/hands-on-guide-to-different-tokenization-methods-in-nlp/*
-

Stemming

Stemming is the process of producing morphological variants of a root/base word. Stemming programs are commonly referred to as stemming algorithms or stemmers.

A stemming algorithm reduces the words “chocolates”, “chocolatey”, “choco” to the root word, “chocolate” and “retrieval”, “retrieved”, “retrieves” reduce to the stem “retrieve”. Stemming is an important part of the pipelining process in Natural language processing. The input to the stemmer is tokenized words.

https://www.geeksforgeeks.org/introduction-to-stemming/

Lemmatization

Lemmatization is the process of grouping together the different inflected forms of a word so they can be analysed as a single item. Lemmatization is similar to stemming but it brings context to the words. So it links words with similar meaning to one word.

Examples of lemmatization:

-> rocks : rock,corpora : corpus,better : good

Diff b/w stemming and lemmatization

https://towardsdatascience.com/lemmatization-in-natural-language-processing-nlp-and-machine-learning-a4416f69a7b6

NLP Lecture 2

Tokenization

- *Tokenization is breaking a text chunk in smaller parts. Whether it is breaking Paragraph in sentences, sentence into words or word in characters.*
- *https://medium.com/data-science-in-your-pocket/tokenization-algorithms-in-natural-language-processing-nlp-1fceab8454af*
- *https://analyticsindiamag.com/hands-on-guide-to-different-tokenization-methods-in-nlp/*

Stemming

Stemming is the process of producing morphological variants of a root/base word. Stemming programs are commonly referred to as stemming algorithms or stemmers.

A stemming algorithm reduces the words “chocolates”, “chocolatey”, “choco” to the root word, “chocolate” and “retrieval”, “retrieved”, “retrieves” reduce to the stem “retrieve”. Stemming is an important part of the pipelining process in Natural language processing. The input to the stemmer is tokenized words.

https://www.geeksforgeeks.org/introduction-to-stemming/

http://snowball.tartarus.org/algorithms/porter/stemmer.html

Lemmatization

Lemmatization is the process of grouping together the different inflected forms of a word so they can be analysed as a single item. Lemmatization is similar to stemming but it brings context to the words. So it links words with similar meaning to one word.

Examples of lemmatization:

-> rocks : rock,corpora : corpus,better : good

Diff b/w stemming and lemmatization

https://towardsdatascience.com/lemmatization-in-natural-language-processing-nlp-and-machine-learning-a4416f69a7b6

POS Tagging

- Part of Speech (hereby referred to as POS) Tags are useful for building parse trees, which are used in building NERs (most named entities are Nouns) and extracting relations between words.
- POS Tagging is also essential for building lemmatizers which are used to reduce a word to its root form.
- POS tagging is the process of marking up a word in a corpus to a corresponding part of a speech tag, based on its context and definition.
- For example: In the sentence “Give me your answer”, answer is a Noun, but in the sentence “Answer the question”, answer is a verb.

Different POS Tagging Techniques

1. **Lexical Based Methods** — Assigns the POS tag the most frequently occurring with a word in the training corpus.

2. **Rule-Based Methods** — Assigns POS tags based on rules. For example, we can have a rule that says, words ending with “ed” or “ing” must be assigned to a verb. Rule-Based Techniques can be used along with Lexical Based approaches to allow POS Tagging of words that are not present in the training corpus but are there in the testing data.

Different POS Tagging Techniques(contd)

3. **Probabilistic Methods** — This method assigns the POS tags based on the probability of a particular tag sequence occurring. Conditional Random Fields (CRFs) and Hidden Markov Models (HMMs) are probabilistic approaches to assign a POS Tag.

4. **Deep Learning Methods** — Recurrent Neural Networks can also be used for POS tagging.

Conditional Random Fields

A CRF is a Discriminative Probabilistic Classifiers which models conditional probability distribution, i.e., P(y|x).

Logistic Regression, SVM, CRF are Discriminative Classifiers. Naive Bayes, HMMs are Generative Classifiers.

In CRFs, the input is a set of features (real numbers) derived from the input sequence using feature functions, the weights associated with the features (that are learned) and the previous label and the task is to predict the current label. The weights of different feature functions will be determined such that the likelihood of the labels in the training data will be maximised.

In CRF, a set of feature functions are defined to extract features for each word in a sentence. Some examples of feature functions are: is the first letter of the word capitalised, what the suffix and prefix of the word, what is the previous word, is it the first or the last word of the sentence, is it a number etc. These set of features are called **State Features**.

In CRF, we also pass the label of the previous word and the label of the current word to learn the weights. CRF will try to determine the weights of different feature functions that will maximise the likelihood of the labels in the training data. The feature function dependent on the label of the previous word is **Transition Features.**

https://sklearn-crfsuite.readthedocs.io/en/latest/api.html#module-sklearn_crfsuite

https://github.com/AiswaryaSrinivas/DataScienceWithPython/blob/master/CRF%20POS%20Tagging.ipynb

Wordnet

- WordNet is the lexical database i.e. dictionary for the English language, specifically designed for natural language processing.
- Synset is a special kind of a simple interface that is present in NLTK to look up words in WordNet. Synset instances are the groupings of synonymous words that express the same concept.
- Example -

```
from nltk.corpus import wordnet
for syn in wordnet.synsets("good"):

    for l in syn.lemmas():

          synonyms.append(l.name())
```

{'beneficial', 'just', 'upright', 'thoroughly', 'in_force', 'well', 'skilful', 'skillful', 'sound', 'unspoiled', 'expert', 'proficient', 'in_effect', 'honorable', 'adept', 'secure', 'commodity', 'estimable', 'soundly', 'right', 'respectable', 'good', 'serious', 'ripe', 'salutary', 'dear', 'practiced', 'goodness', 'safe', 'effective', 'unspoilt', 'dependable', 'undecomposed', 'honest', 'full', 'near', 'trade_good'}

Word Sense Disambiguation (WSD)

For example, consider the two sentences.

“The bank will not be accepting cash on Saturdays. ”

“The river overflowed the bank.”

The word bank in the first sentence refers to the commercial (finance) banks, while in second sentence, it refers to the river bank.

https://towardsdatascience.com/a-simple-word-sense-disambiguation-application-3ca645c56357

Query Expansion

- Query expansion (QE) is the process of reformulating a given query to improve retrieval performance in information retrieval operations, particularly in the context of query understanding.

https://github.com/ellisa1419/Wordnet-Query-Expansion/blob/master/wordnet.py

NLP Lecture 5

Wordnet Lexical Relations

What's special about WordNet?

WordNet gives information about two fundamental, universal properties of human language:

polysemy and **synonymy**

Polysemy = one:many mapping of form and meaning

Synonymy = one:many mapping of meaning and form

Polysemy

One word form expresses multiple meanings

{*table*, tabular_array}

{*table*, piece_of_furniture}

{*table*, mesa}

{*table*, postpone}

Note: the most frequent word forms are the most polysemous!

Synonymy

One concept is expressed by several different word forms:

{beat, hit, strike}

{car, motorcar, auto, automobile}

Polysemy and synonymy

Understanding and generating language (as for translation) means matching a word form with the intended, context-appropriate meaning

People (fluent speakers of a language) do this very efficiently

Polysemy in WordNet

A word form that appears in n synsets

is n-fold polysemous

{table, tabular_array}

{table, piece_of_furniture}

{table, mesa}

{table, postpone}

table is fourfold polysemous/has four senses

four distinct concepts are associated with the word form table

WN as a lexical resource

Have concept, need words”

--depart from synset, travel in WordNet space

“Have word, need concept”

--query word form, find associated synsets

WordNet as a lexical resource

WN has been incorporated into many dictionaries

Google “define” usually brings up WN entry at the top of the list

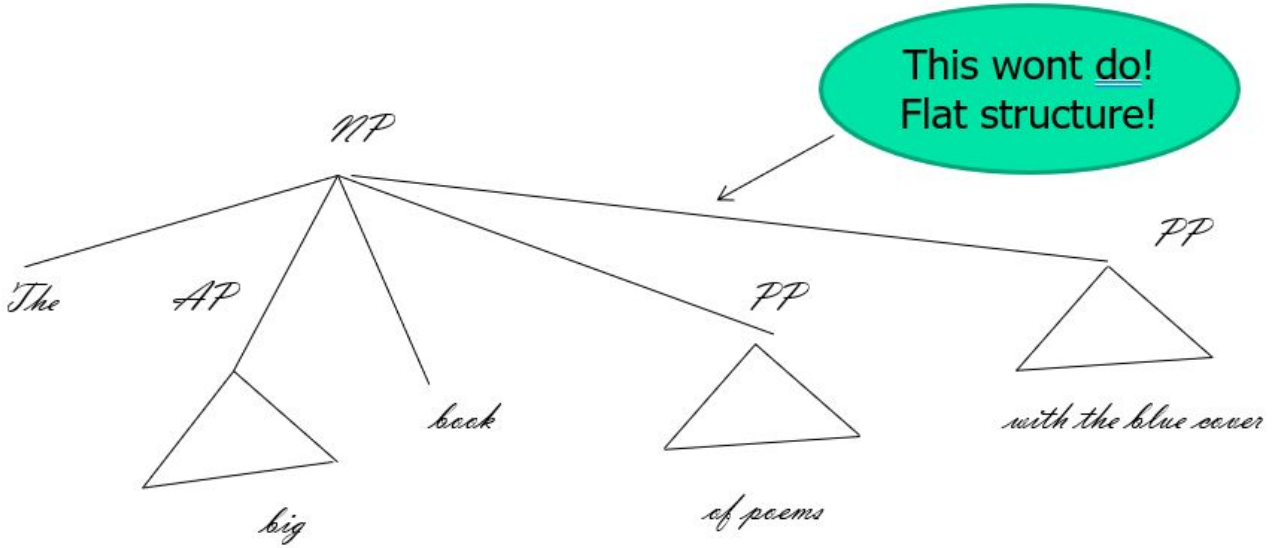
User-created visual interfaces (e.g., visualthesaurus.com)

Keyword Extraction

https://thinkinfi.com/automatic-keyword-extraction-using-rake-in-python/

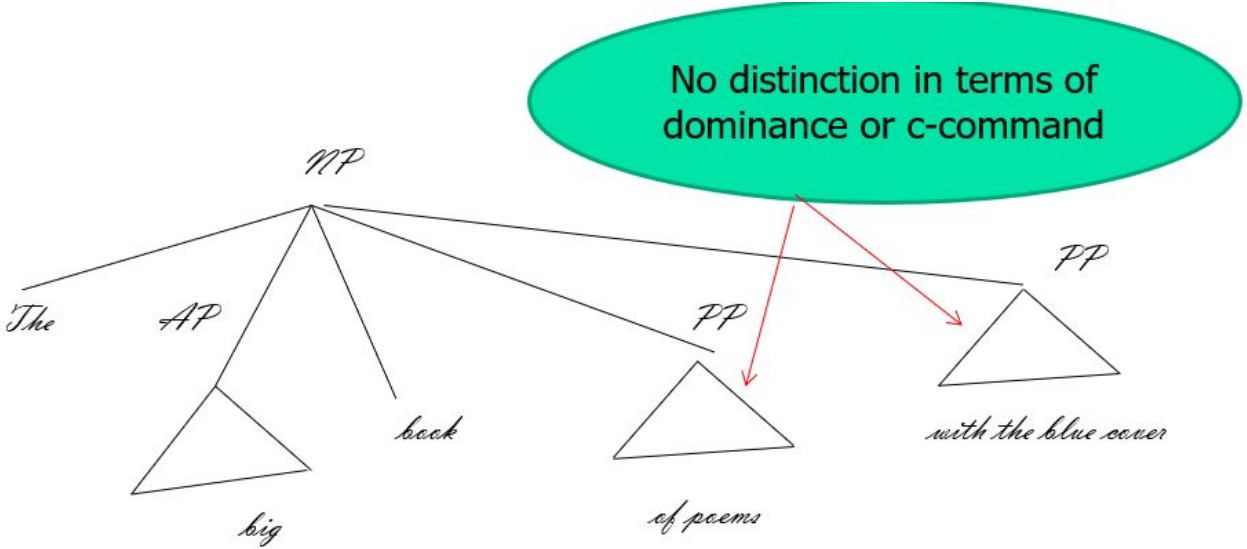
https://www.analyticsvidhya.com/blog/2020/11/words-that-matter-a-simple-guide-to-keyword-extraction-in-python/

Parsing



[The big book of poems with the Blue cover] is on the table.

PPs are at the same level: flat with respect to the head word “book”



[The big book of poems with the Blue cover] is on the table.

Constituency test of Replacement" runs into problems

One-replacement:

I bought the big [book of poems with the blue cover] not the small [one]

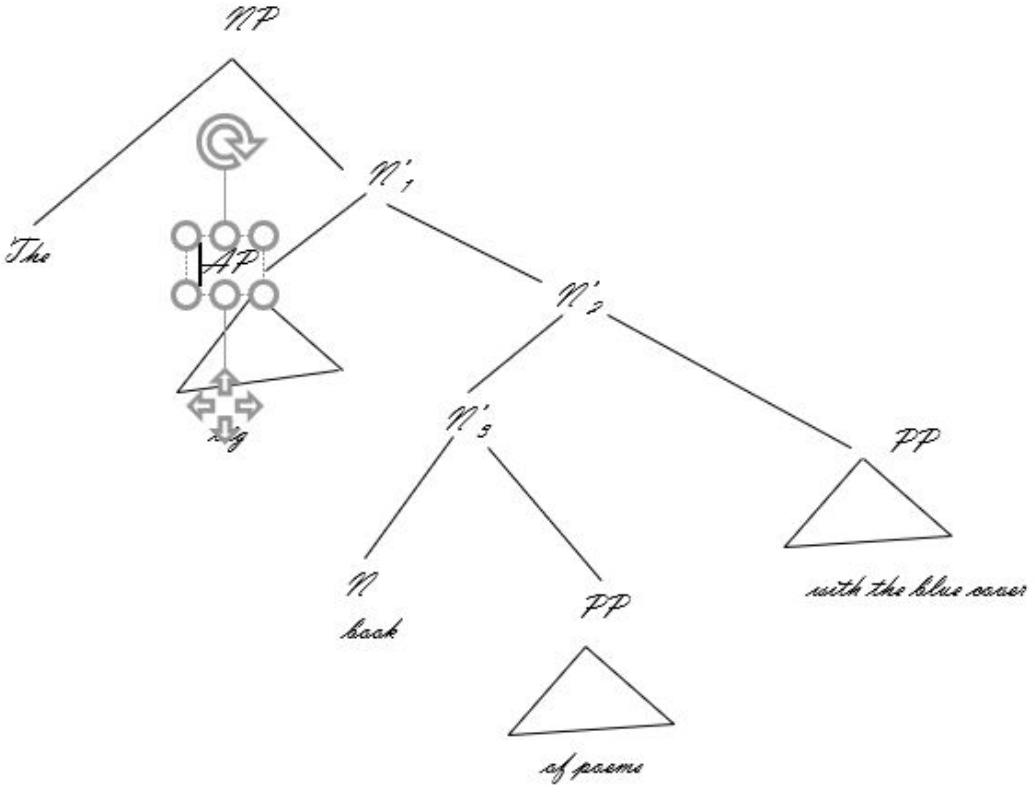
One-replacement targets book of poems with the blue cover

Another one-replacement:

I bought the big [book of poems] with the blue cover not the small [one] with the red cover

One-replacement targets book of poems

Deeply Embedded Structure



To target N1’

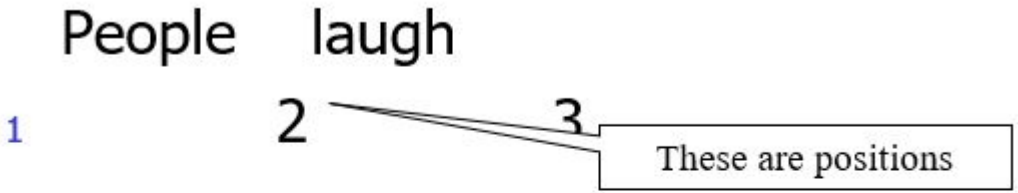
I want [NPthis [N’big book of poems with the red cover] and not [Nthat [None]]

Parsing Algorithms

A simplified grammar

- $S \rightarrow NP\ VP$
- $NP \rightarrow DT\ N \mid N$
- $VP \rightarrow V\ ADV \mid V$

Example Sentence



Lexicon:

People - N, V

Laugh - N, V

This indicate that both Noun and Verb is possible for the word "People"

Top-Down Parsing

	State	Backup State	Action
1.	((S) 1)	-	-
2.	((NP VP)1)	-	-
3a.	((DT N VP)1)	((N VP) 1)	-
3b.	((N VP)1)	-	-
4.	((VP)2)	-	Consume "People"
5a.	((V ADV)2)	((V)2)	-
6.	((ADV)3)	((V)2)	Consume "laugh"
5b.	((V)2)	-	-
6.	((.)3)	-	Consume "laugh"

Termination Condition : All inputs over. No symbols remaining.

Note: Input symbols can be pushed back.

Bottom-Up Parsing

Some conventions:

N_{12}

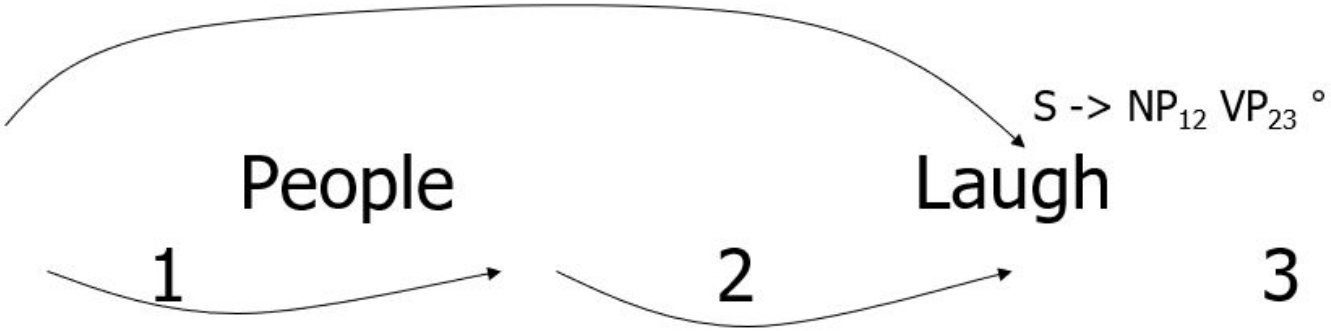
Represents positions

$S_{1?} \rightarrow NP_{12} \circ VP_{2?}$

End position unknown

Work on the LHS done, while the work on RHS remaining

Bottom-Up Parsing (pictorial representation)



N_{12}
V_{12}
NP_{12} -> N_{12} °
VP_{12} -> V_{12} °
$S_{1?}$ -> NP_{12} ° $VP_{2?}$

N_{23}
V_{23}
NP_{23} -> N_{23} °
VP_{23} -> V_{23} °

Problem with Top-Down Parsing

Left Recursion

Suppose you have A-> AB rule.

Then we will have the expansion as follows:

((A)K) -> ((AB)K) -> ((ABB)K) ……..

Top-Down Bottom-Up Chart Parsing

- Combines advantages of top-down & bottom-up parsing.
- Does not work in case of left recursion.
  - *e.g.* – "People laugh"
    - People – noun, verb
    - Laugh – noun, verb
  - Grammar – $S \rightarrow NP\ VP$

    $NP \rightarrow DT\ N \mid N$

    $VP \rightarrow V\ ADV \mid V$

Parse Trees for a Structurally Ambiguous Sentence

Let the grammar be –

$S \rightarrow NP\ VP$

$NP \rightarrow DT\ N \mid DT\ N\ PP$

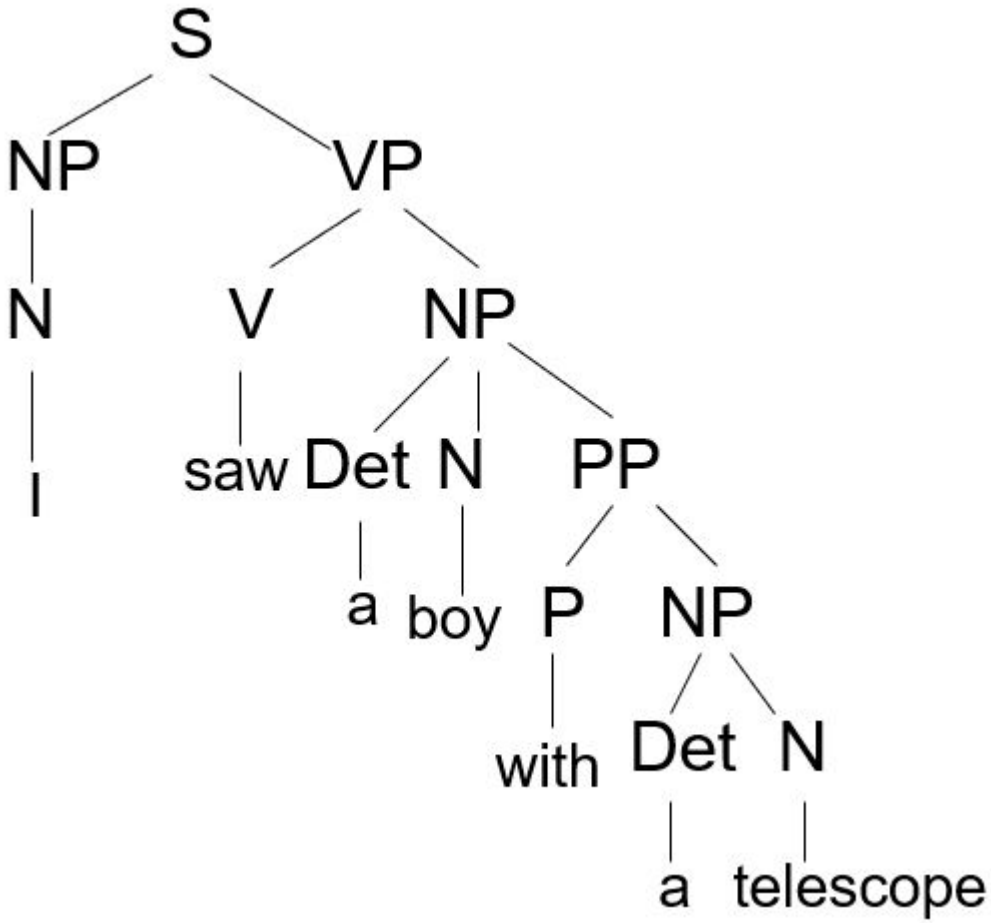
$PP \rightarrow P\ NP$

$VP \rightarrow V\ NP\ PP \mid V\ NP$

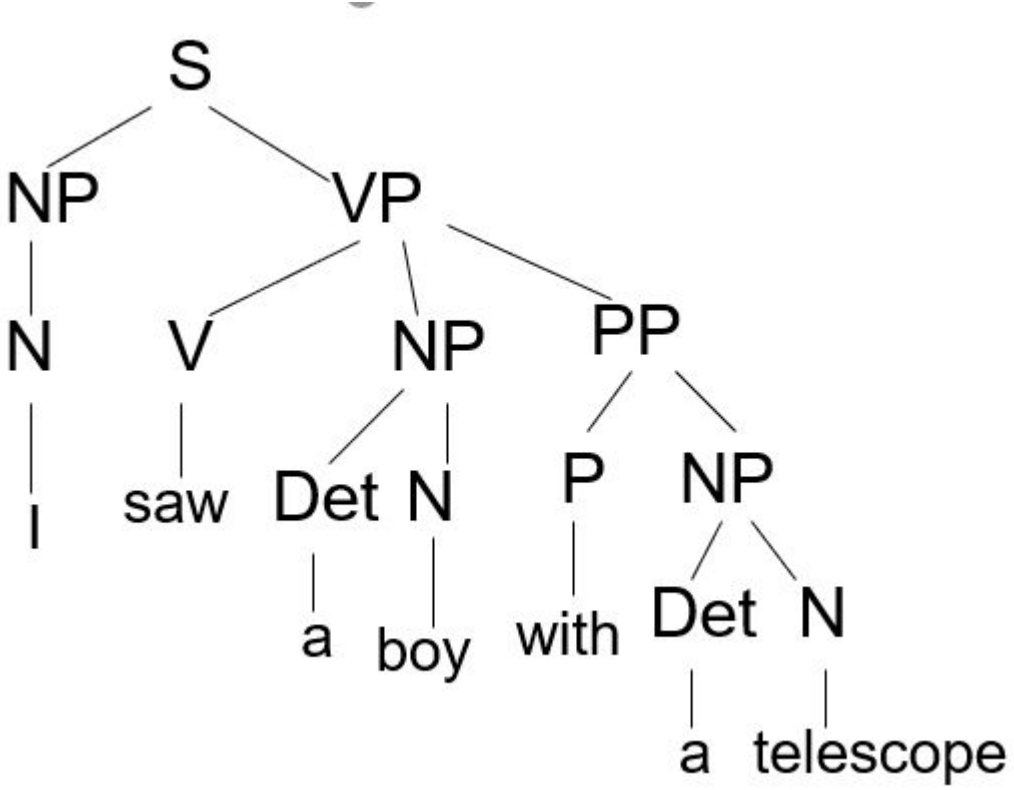
For the sentence,

"I saw a boy with a telescope"

Parse Tree - 1



Parse Tree -2



Lossy Image Compression using SVD Coding Algorithm

- K M Aishwarya

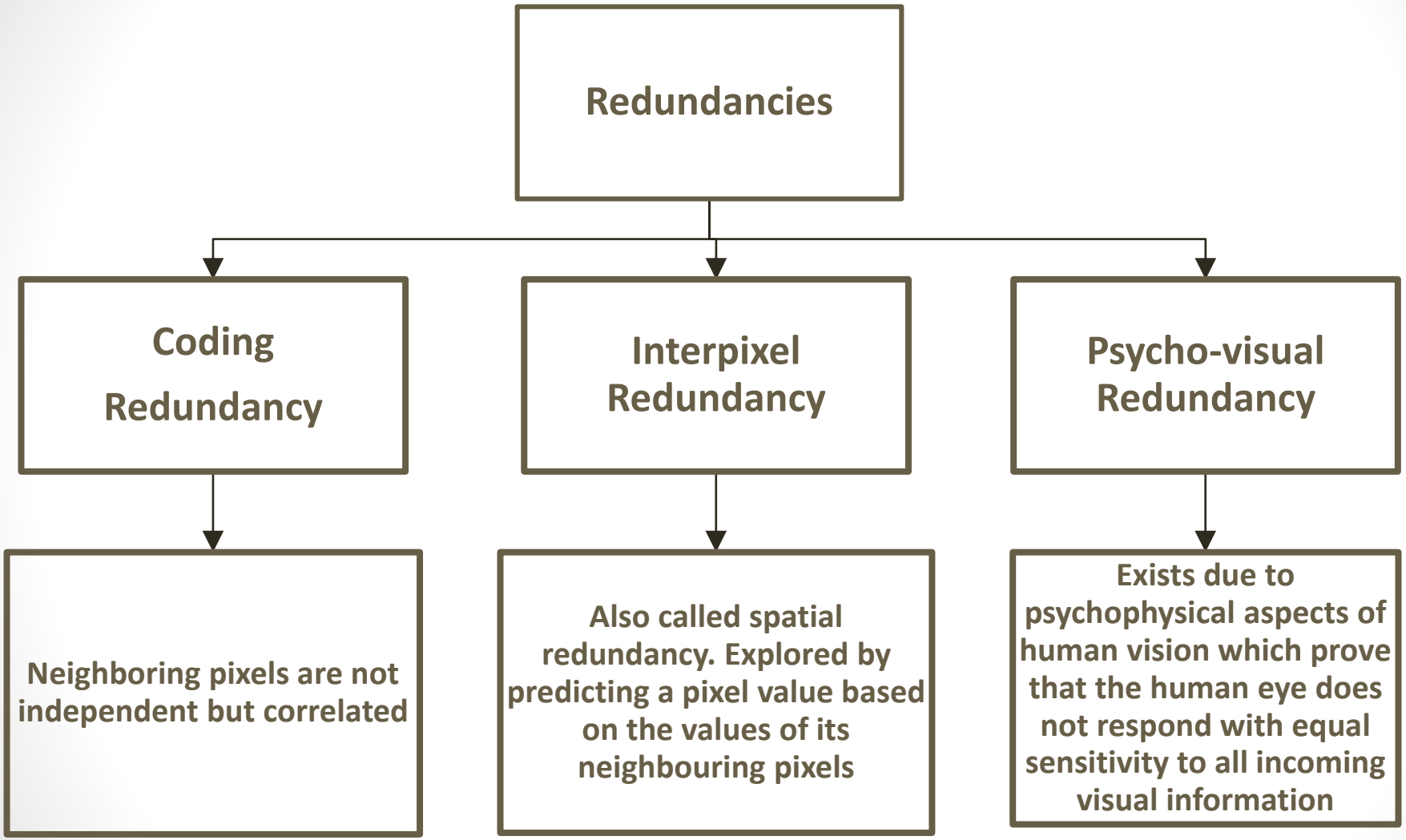
Why is compression needed?

- To store data efficiently
- To transmit data efficiently
- To save:
  - Memory
  - Bandwidth
  - Cost

Why SVD?

- Image compression alone can experience redundancies which can be overcome by Singular Value Decomposition (SVD) of the image matrix.
- Image compression and coding techniques explore 3 types of redundancies:
    - Coding redundancy
    - Interpixel redundancy
    - Psycho-visual redundancy

How to compress?



What is SVD?

- Basic idea:

  Taking a high dimensional, highly variable set of data points and reducing it to a lower dimensional space that exposes the substructure of the original data more clearly and orders it from most variation to the least.

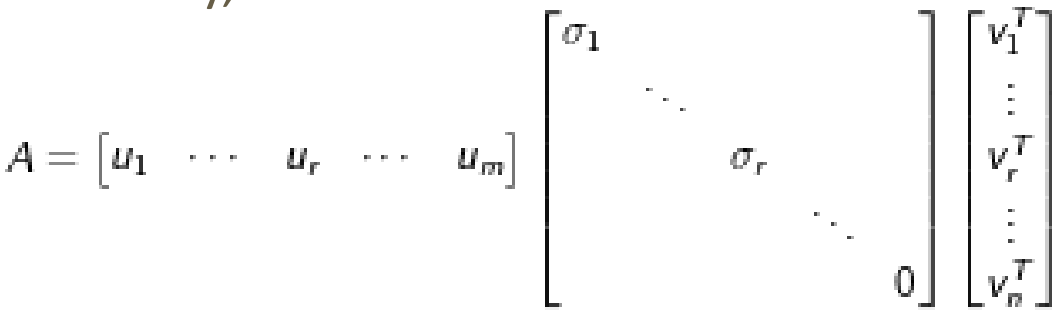
- Definition:

  In linear algebra, the singular value decomposition is a factorization of a real or complex, square or non-square matrix. Consider a matrix A with *m* rows and *n* columns with rank *r*. Then A can be factorized into three matrices:

$$A = USV^T$$

SVD – Overview I

- Diagrammatically,



where,

- U is an $m \times m$ orthogonal matrix
- V^T is the conjugate transpose of the $n \times n$ orthogonal matrix.
- S is an $m \times n$ diagonal matrix with non-negative real numbers on the diagonal which are known as the singular values of A.
- The m columns of U and n columns of V are called the left-singular and right-singular vectors of A respectively.
- The numbers $\sigma_1^2 \geq \ldots \geq \sigma_r^2$ are the eigenvalues of AA^T and A^TA.

SVD – Overview II

- SVD takes a matrix, square or non-square, and divides it into two orthogonal matrices and a diagonal matrix
- Simply applying SVD on an image does not compress it
- To compress an image, after applying SVD:
  - Retain the first few singular values (containing maximum image info)
  - Discard the lower singular values (containing negligible info)
- The singular vectors form orthonormal bases, and the important relation

$$A\, v_i = s_i\, u_i$$

- shows that each right singular vectors is mapped onto the corresponding left singular vector, and the "magnification factor" is the corresponding singular value

Mathematical steps to calculate SVD of a matrix

1. Given an input image matrix A.
2. First, calculate AA^T and A^TA.
3. Use AA^T to find the eigenvalues and eigenvectors to form the columns of U: $(AA^T - \lambda I)\,\vec{x} = 0$.
4. Use A^TA to find the eigenvalues and eigenvectors to form the columns of V: $(A^TA - \lambda I)\,\vec{x} = 0$.
5. Divide each eigenvector by its magnitude to form the columns of U and V.
6. Take the square root of the eigenvalues to find the singular values, and arrange them in the diagonal matrix S in descending order: $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_r \geq 0$

In MATLAB: `[U,W,V]=svd(A,0)`

SVD Compression Measures

- Parameters for quantitative and qualitative compression of image:
  - Compression Ratio (C_R):

$$C_R = \frac{\text{uncompressed image file size}}{\text{compressed image file size}}$$

  - Mean Square Error (MSE):

$$MSE = \frac{1}{mn}\sum_{y=1}^{m}\sum_{x=1}^{n}(f_A(x,y) - f_{A_k}(x,y))$$

  - Signal to Noise Ratio (SNR):

$$SNR = \frac{P_{signal}}{P_{noise}} = \left(\frac{A_{signal}}{A_{noise}}\right)^2$$

  - Peak Signal to Noise Ratio (PSNR):

$$PSNR = 10log_{10}\left(\frac{MAX_I^2}{MSE}\right) = 20log_{10}\left(\frac{MAX_I}{\sqrt{MSE}}\right)$$

Storage Space Calculations

$\Longrightarrow \quad A = USV^T$

$\Longrightarrow \quad A = \sum_{i=1}^{r} \sigma_i u_i v_i^T$

$\Longrightarrow \quad A_k = \sigma_1 u_1 v_1^T + \sigma_2 u_2 v_2^T + \ldots + \sigma_r u_r v_r^T$

- When performing SVD compression, the sum is not performed to the very last Singular Values (SV's); the SV's with small enough values are dropped. The values which fall outside the required rank are equated to zero. The closet matrix of rank *k* is obtained by truncating those sums after the first *k* terms:

  $\Longrightarrow \quad A_k = \sigma_1 u_1 v_1^T + \sigma_2 u_2 v_2^T + \ldots + \sigma_k u_k v_k^T$

- Total storage for A_k will be:

  $\Longrightarrow \quad A_k = k(m + n + 1)$

- The digital image corresponding to A_k will still have very close resemblance to the original image

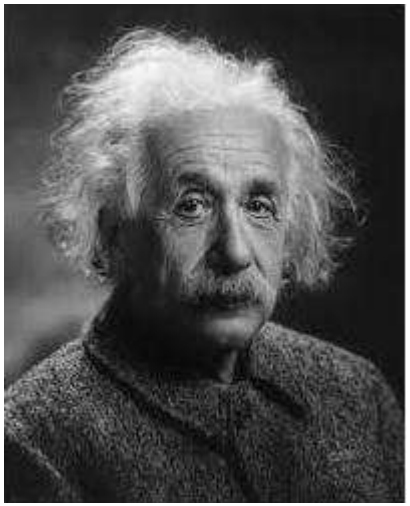
Lowest rank-*k* approximations

- Since $\sigma_1 > \cdots > \sigma_n > 0$, in the singular matrix, the first term of this series will have the largest impact on the total sum, followed by the second term, then the third term, etc.
- This means we can approximate the matrix A by adding only the **first few terms of the series**.
- As *k* increases, the image quality increases, but so too does the amount of memory needed to store the image. This means smaller ranked SVD approximations are preferable.
- Hence *k* is limit bound to ensure SVD compressed image occupies lesser space than original image.

Necessary condition: $\boldsymbol{k < \frac{mn}{m+n+1}}$

Examples of SVD compression in MATLAB

Original Image:

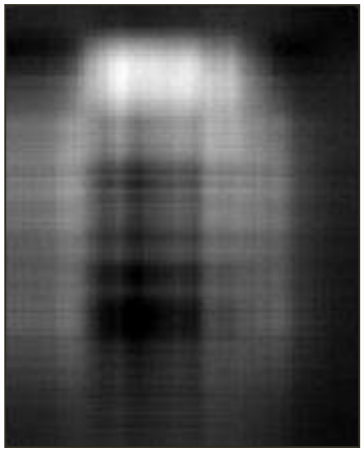


Rank of image = 202

A_k = 31840

C_R = 0.580

SVD compressed images for different values of *k*

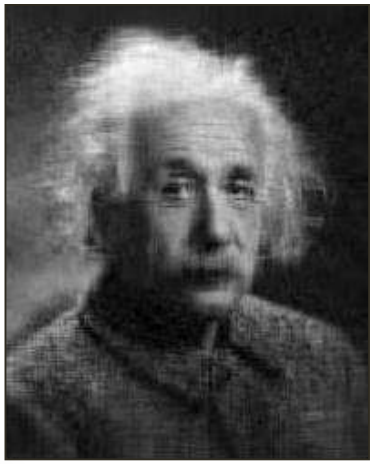


A_k = 30796
C_R = 61.046

k = 2

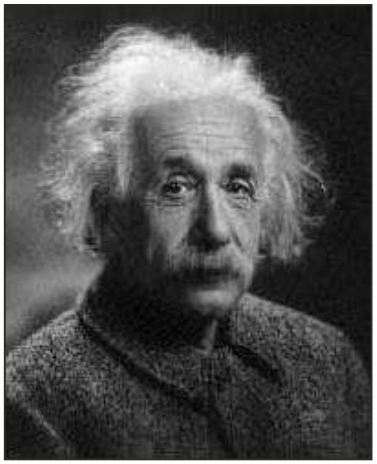
A_k = 31679
C_R = 23.588

k = 12



A_k = 31699
C_R = 12.101

k = 22



A_k = 31781
C_R = 5.72

k = 52

Observation & Inference

- *k* represents the number of Eigen values used in the reconstruction of the compressed image
- Smaller the value of *k*, more is the compression ratio but image quality deteriorates
- As the value of *k* increases, image quality improves (i.e. smaller MSE & larger PSNR) but more storage space is required to store the compressed image
- When *k* is equal to the rank of the image matrix (202 here), the reconstructed image is almost same as the original one

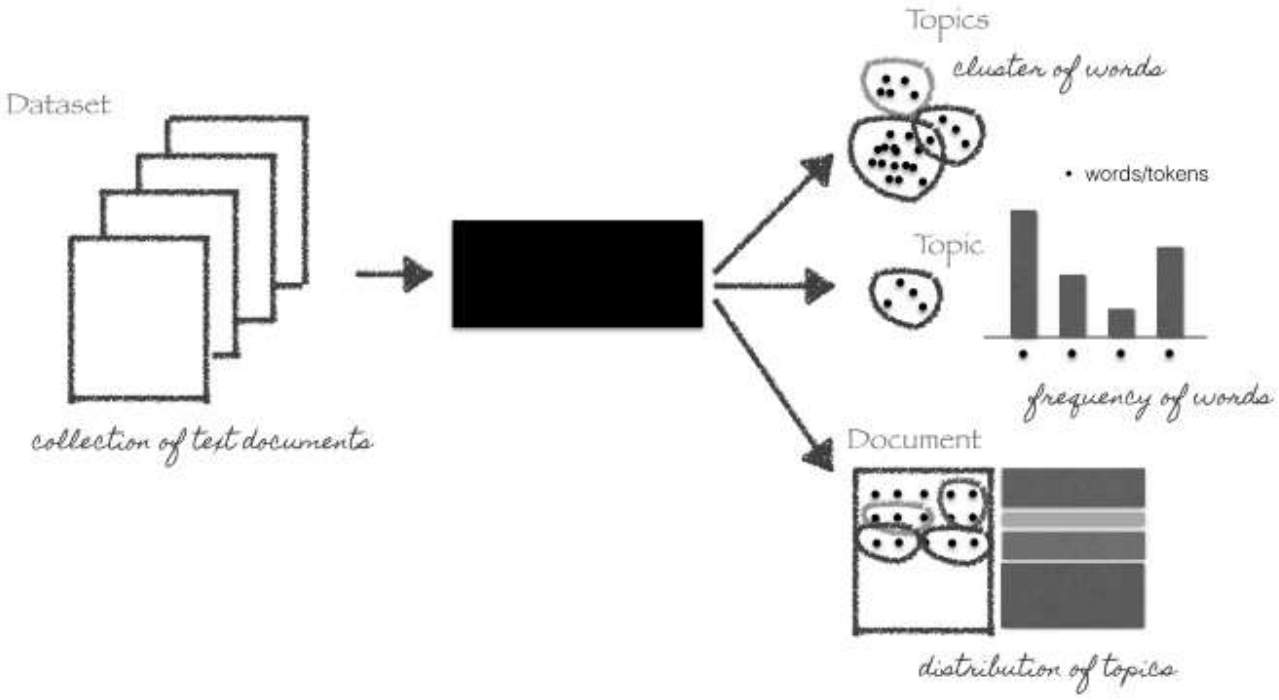
Conclusion

- SVD's applications in world of image and data compression are very useful and resource-saving.
- SVD allows us to arrange the portions of a matrix in order of importance. The most important singular values will produce the most important unit eigenvectors.
- We can eliminate large portions of our matrix without losing quality.
- Therefore, an optimum value for 'k' must be chosen, with an acceptable error, which conveys most of the information contained in the original image, and has an acceptable file size too.

Thank You

Topic modeling using Latent Dirichlet Allocation(LDA) and Gibbs Sampling explained!

Ankur Tomar Follow
Nov 25, 2018 · 9 min read



Credits: Christine Doig

Hi everyone. It's been a long time since I wrote my last blog on the different type of recommender systems. This time, I will be writing about Topic Modelling.

There are numerous sources out there where you can learn what Topic Modelling is, but I think, most of them are somewhat technical and requires a good knowledge of math and other techniques like Markov Chains. In this article, I will try to explain the idea

behind Topic Modelling in a very intuitive way with bare minimum math and technicalities. I will write this article in two parts. In the first part, I will try to explain what topic modeling is and in the second part, I will provide a python tutorial of how to do topic modeling on the real-world dataset.

Before I start, I want to acknowledge that this article is heavily inspired by the lecture of my exploratory data analytics Prof. Edward McFowland, Coursera's course on Natural Language Processing by Higher School of Economics and Jordan Boyd-Graber explanation of Gibbs sampling.

INTRODUCTION

What is topic modeling?

Topic modeling is a branch of unsupervised natural language processing which is used to represent a text document with the help of several topics, that can best explain the underlying information in a particular document. This can be thought in terms of clustering, but with a difference. Now, instead of numerical features, we have a

collection of words that we want to group together in such a way that each group represents a topic in a document.

Why do we need topic modeling?

Okay, so now the question arises why do we need topic modeling? If we look around, we can see a huge amount of textual data lying around us in an unstructured format in the form of news articles, research papers, social media posts etc. and we need a way to understand, organize and label this data to make informed decisions. Topic modeling is used in various applications like finding questions on stack overflow that are similar to each other, news flow aggregation and analysis, recommender systems etc. All of these focus on finding the hidden thematic structure in the text, as it is believed that every text that we write be it a tweet, post or a research paper is composed of themes like sports, physics, aerospace etc.

How to do topic modeling?

Currently, there are many ways to do topic modeling, but in this post, we will be discussing a probabilistic modeling approach called Latent Dirichlet Allocation (LDA) developed by Prof. David M. Blei in 2003. This is an extension of Probabilistic Latent Semantic Analysis (PLSA) developed in 1999 by Thomas Hoffman with a very minute difference in terms of how they treat per-document distribution. So let's jump straight into how LDA works.

Latent Dirichlet Allocation

Let's start by understanding the meaning of each word in the title, as I think it contains everything that we need to know to understand how LDA works.

Latent: This refers to everything that we don't know a priori and are hidden in the data. Here, the themes or topics that document consists of are unknown, but they are believed to be present as the text is generated based on those topics.

Dirichlet: It is a 'distribution of distributions'. Yes, you read it right. But what does this mean? Let's think about this with the help of an example. Let's suppose there is a machine that produces dice and we can control whether the machine will always produce a dice with equal weight to all sides, or will there be any bias for some sides. So,

the machine producing dice is a distribution as it is producing dice of different types. Also, we know that the dice itself is a distribution as we get multiple values when we roll a dice. This is what it means to be a distribution of distributions and this is what Dirichlet is. Here, in the context of topic modeling, the Dirichlet is the distribution of topics in documents and distribution of words in the topic. It might not be very clear at this point of time, but it's fine as we will look at it in more detail in a while.

Allocation: This means that once we have Dirichlet, we will allocate topics to the documents and words of the document to topics.

That's it. This is what LDA is in a nutshell. Now let's understand how this works in topic modeling.

Just to recap, what LDA says is that each word in each document comes from a topic and the topic is selected from a per-document distribution over topics. So we have two matrices:

1. $\theta td = P(t|d)$ which is the probability distribution of topics in documents

2. $\Phi wt = P(w|t)$ which is the probability distribution of words in topics

And, we can say that the probability of a word given document i.e. $P(w|d)$ is equal to:

$$\sum_{t \epsilon T} p(w|t,d)\, p(t|d)$$

where T is the total number of topics. Also, let's assume that there is W number of words in our vocabulary for all the documents.

If we assume conditional independence, we can say that

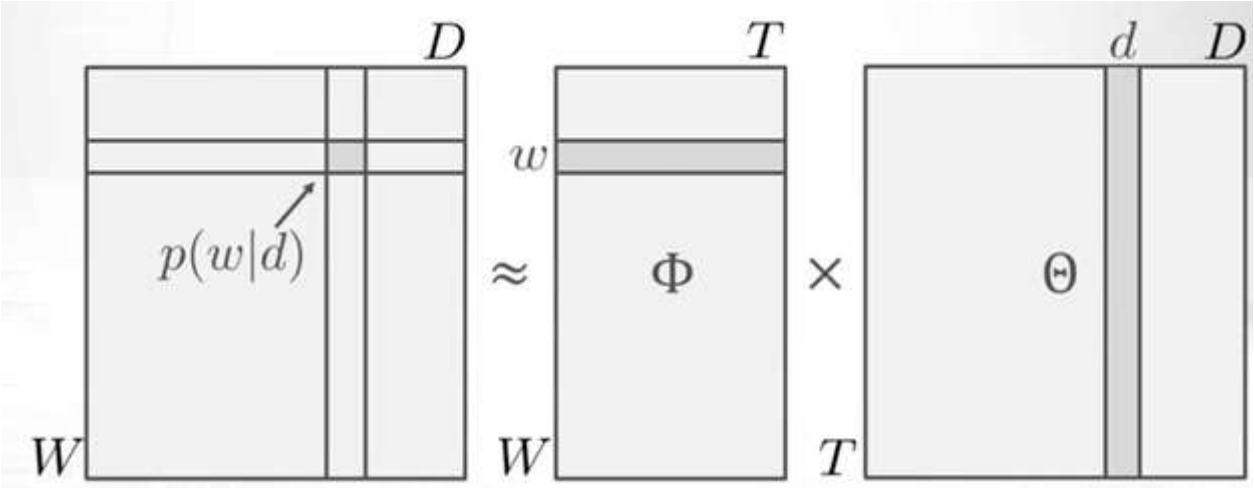
$P(w|t,d) = P(w|t)$

And hence $P(w|d)$ is equal to:

$$\sum_{t=1}^{T} p(w|t)\, p(t|d)$$

that is the dot product of Θtd and Φwt for each topic t.

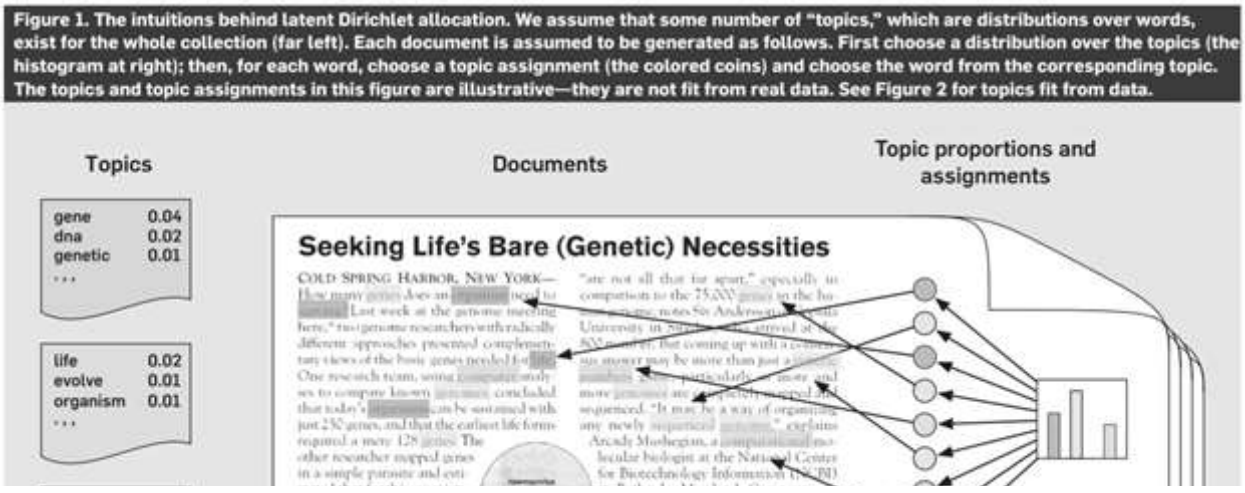
This can be represented in the form of a matrix like this:

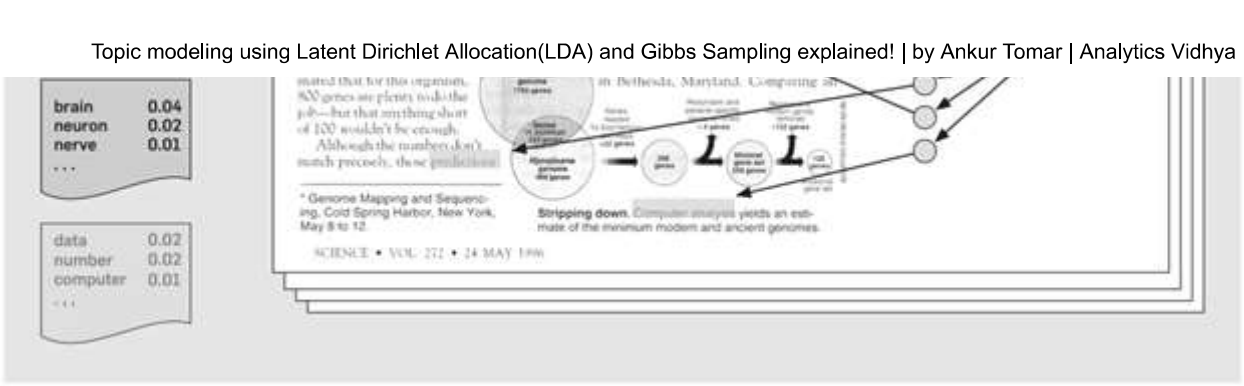


Credit: Higher School of Economics

So, looking at this we can think of LDA similar to that of matrix factorization or SVD, where we decompose the probability distribution matrix of word in document in two matrices consisting of distribution of topic in a document and distribution of words in a topic.

Therefore, what we will get is, for example, this:





Credit: David M. Blei

And to tie back to our example of dice, we can say that each word in the distribution of words in a topic is similar to a side of the dice, and we have Dirichlet parameter to control if all the words have same probability in a topic or will that topic have an extreme bias towards some words. Same intuition is for distribution of topics in a document.

Good. Now comes the important part. How do we learn the weights of these two matrices?

To start with, let's randomly assign weights to both the matrices and assume that our data is generated as per the following steps:

1. Randomly choose a topic from the distribution of topics in a document based on their assigned weights. In the previous example, let's say we chose pink topic

2. Next, based on the distribution of words for the chosen topic, select a word at random and put it in the document

3. Repeat this step for the entire document

In this process, if our guess of the weights is wrong, then the actual data that we observe will be very unlikely under our assumed weights and data generating process. For example, let's say we have document D1 which consists of the following text:

"*Qualcomm® Adreno™ 630 Visual Processing Subsystem featuring room-scale 6DoF with SLAM, Adreno Foveation*"

and let's say we assign high weights to topic T1 which has high weights for words like Spoon, Plates, Onions etc. then we will see that given our assumption of how data is

generated, it is very unlikely that T1 belongs to D1 or these words belongs to T1. Therefore, what we are doing is we are trying to maximize the likelihood of our data given these two matrices.

To identify the correct weights, we will use an algorithm called Gibbs sampling. Let's now understand what Gibbs sampling is and how does it work in LDA.

Gibbs Sampling

Gibbs sampling is an algorithm for successively sampling conditional distributions of variables, whose distribution over states converges to the true distribution in the long run. This is somewhat an abstract concept and needs a good understanding of Monte Carlo Markov Chains and Bayes theorem. These concepts and the math behind them is fairly complex and is out of the scope of this blog. Here, I will try to give an intuition on how Gibbs Sampling work to identify topics in the documents.

As I mentioned earlier, to start with, we will assume that we know Θ and Φ matrices. Now what we will do is, we will slowly change these matrices and get to an answer that maximizes the likelihood of the data that we have. We will do this on word by word basis by changing the topic assignment of one word. We will assume that we don't know the topic assignment of the given word but we know the assignment of all other words in the text and we will try to infer what topic will be assigned to this word.

If we look at this in mathematical manner, what we are doing is we are trying to find conditional probability distribution of a single word's topic assignment conditioned on the rest of the topic assignments. Ignoring all the mathematical calculations, what we will get is a conditional probability equation that looks like this for a single word *w* in document *d* that belongs to topic *k*:

$$p(z_{d,n} = k|\vec{z}_{-d,n}, \vec{w}, \alpha, \lambda) = \frac{n_{d,k} + \alpha_k}{\sum_i^K n_{d,i} + \alpha_i} \frac{v_{k,w_{d,n}} + \lambda_{w_{d,n}}}{\sum_i v_{k,i} + \lambda_i}$$

Credit: Jordan Boyd-Graber

where:

n(d,k): Number of times document d use topic k

v(k,w): Number of times topic k uses the given word

αk: Dirichlet parameter for document to topic distribution

λw: Dirichlet parameter for topic to word distribution

There are two parts two this equation. First part tells us how much each topic is present in a document and the second part tells how much each topic likes a word. Note that for each word, we will get a vector of probabilities that will explain how likely this word belongs to each of the topics. In the above equation, it can be seen that the Dirichlet parameters also acts as smoothing parameters when *n(d,k)* or *v(k,w)* is zero which means that there will still be some chance that the word will choose a topic going forward.

Let's go through an example now:

To start with, suppose we have a document with some random word topic assignment, for example, as shown below:

India	enters	world	cup	final
1	3	1	2	4

We also have our count matrix v(k,w) as shown below:

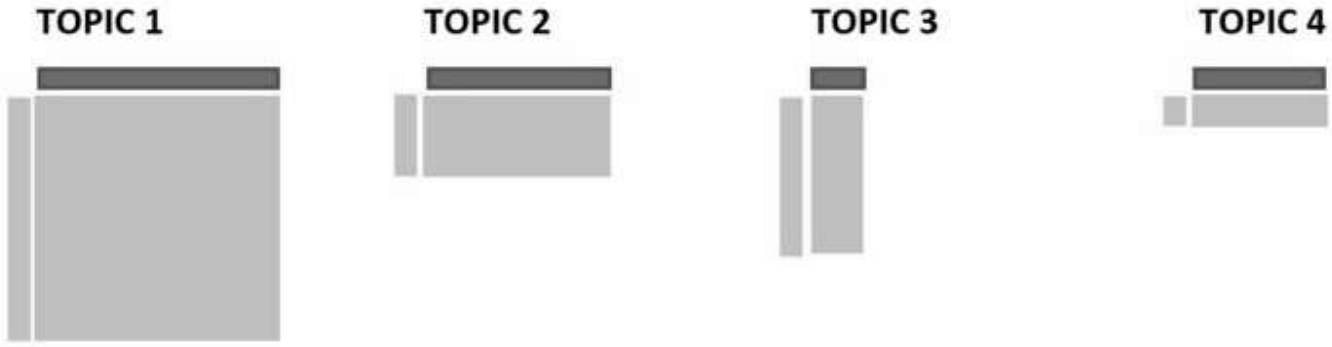
	1	2	3	4
India	70	5	0	8
enters	2	3	15	6
world	28	4	12	1
cup	6	43	6	0
final	7	0	9	31

Now let's change the assignment of word **world** in the document.

- First, we will reduce the count of world in topic 1 from 28 to 27 as we don't know to what topic world belongs.
- Second let's represent the matrix *n(d,k)* in the following way to show how much a document use each topic

· Third, let's represent *v(k,w)* in the following way to show how many times each topic is assigned to this word

- Fourth, we will multiply these two matrices to get our conditional probabilities



- Finally, we will randomly pick any of the topic and will assign that topic to **world** and we will repeat these steps for all other words as well. Intuitively, topic with highest conditional probability should be selected but as we can see other topics also have some chance to get selected

That's it. This what Gibbs sampling algorithm is doing under the hood. Although we skipped some details like hyperparameter tuning, but from an intuition perspective, this is how Gibbs sampling works for topic modeling.

So, this is it for the theory of Latent Dirichlet Allocation. I hope this was helpful in understanding what topic modelling is and how we use LDA with Gibbs for topic modelling. In the next article, I will post the implementation of LDA using Python.

Thanks!