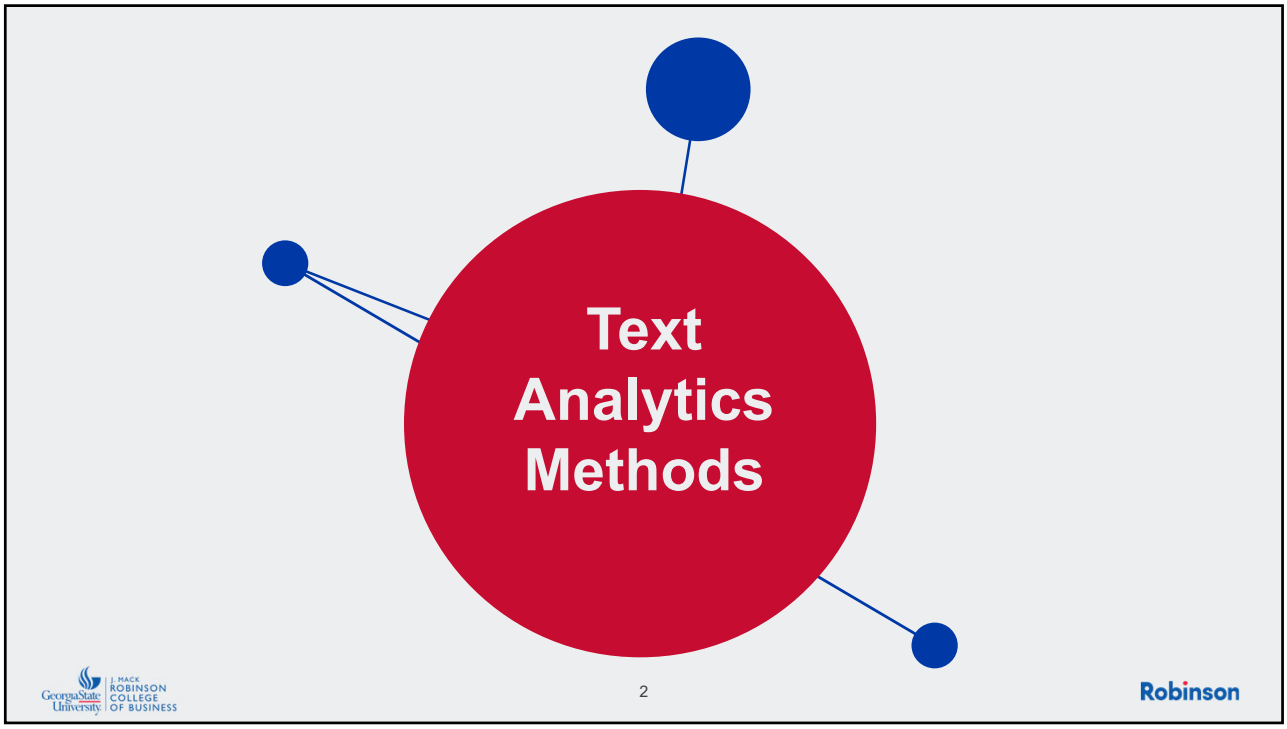
Georgia State University | J. MACK ROBINSON COLLEGE OF BUSINESS

MSA 8650 Deep Learning

fall Semester, 2018

1



Text Documents

Black Ink

869 N Swarthmore Ave, Pacific Palisades, CA

Write a review

4.7 ★★★★½ 15 reviews

Sort by: Most helpful

Martin Jay
8 reviews
★★★★★ a month ago
My wife and I were out and about last weekend and decided to do a little window shopping. We were instantly drawn in by the theme and decor of this boutique. I would only recommend Black Ink if you are searching for a great variety of ... More

Like

Angie Morris
4 reviews
★★★★★ 2 weeks ago
What a hidden gem this place is! I'm so grateful that I found this place on Facebook. The ambiance was extremely heartwarming and cozy. I love everything about this place. I highly recommend this store. Compared to a stationary I recently ... More

Like

Text Documents

☆☆☆☆☆ **nothing fits**
By M. Stelmach on May 31, 2014
Verified Purchase
I bought this to host a tiki bbq so my guest could bar tend themselves- BUT NONE OF THE BOTTLES FIT- the few I got to fit leaked! I must have tried 20 different bottle styles! What a waste!

Comment | 12 people found this helpful. Was this review helpful to you? Yes No Report abuse

☆☆☆☆☆ **it looks good but pass it up.**
By jared belgrave on October 27, 2013
Verified Purchase
It looks great but sometimes the dispenser Leaks so you are losing liquor and that's not good . And it doesn't hold a bottle of Grey goose its to tall , and the holder is plastic.

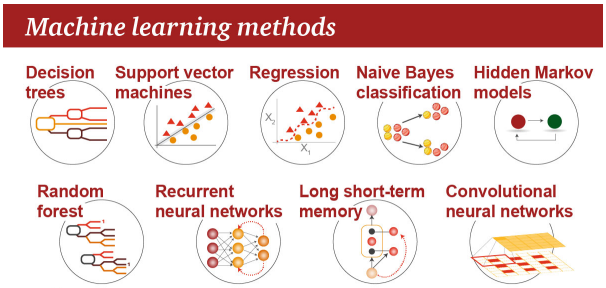
Comment | 25 people found this helpful. Was this review helpful to you? Yes No Report abuse

☆☆☆☆☆ **Piece of trash**
By sexy momma on April 12, 2016
Verified Purchase
Piece of trash, with in the first 30 minutes of the first time using this all three of the taps where leaking. I double checked the seals on the bottles and they where tight, it was the cheaply made plastic taps.

Comment | 4 people found this helpful. Was this review helpful to you? Yes No Report abuse

☆☆☆☆☆ **Leaks right away**
By GeorgeVegas on November 4, 2015
Verified Purchase
I didn't know that Goose, Belvedere bottles don't fit on this or anything with a long neck. But the worst part is I finally used it and after pouring 1 shot 2 of the 3 nozzles started leaking. What a waste of liquor. I don't write bad reviews unless I'm really upset. I tried to contact the seller first but didn't find anywhere to contact

Text Analytics Methods



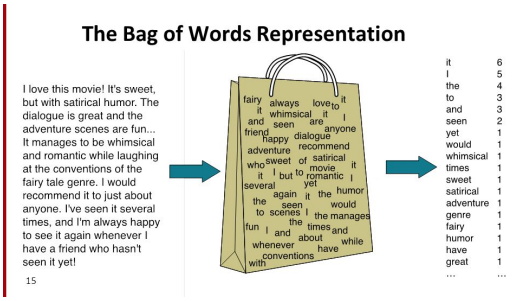
Source: pwc.com/nextintech

Word Representation and Feature Extraction

- **How to represent words and get features for textual daata?**
  - The mapping from textual data to real valued numbers/vectors is called feature extraction or feature representation.
  - When we consider a sentence, a paragraph, or a document, ***the observable features are the counts and the order of letters and words within the text***.

Word Representation and Feature Extraction
Bag of Words

- A very common feature extraction procedure for sentences and documents is the bag-of-words approach (BOW). In this approach, we look at the histogram of the words within the text, i.e., considering *each word count as a feature*.



Major assumption of the BOW:

Answer: Order does not matter.

John is quicker than Mary and *Mary is quicker than John* have the same vectors

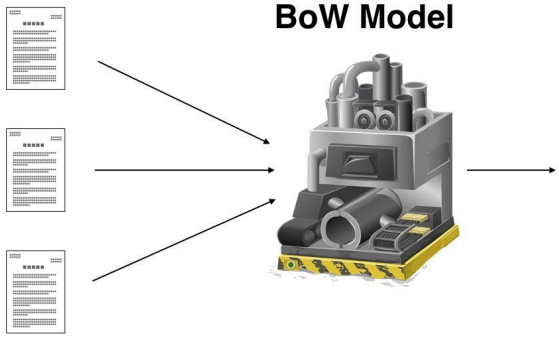
Word Representation and Feature Extraction
Bag of Words

Binary term-document incidence matrix

1. **I love dogs.**
2. **I hate dogs and knitting.**
3. **Knitting is my hobby and my passion.**

Each document is represented by a binary vector $\in \{0,1\}^{|V|}$

BoW Model



	I	love	dogs	hate	and	knittin	is	my	hobby	passi
Doc 1	1	1	1							
Doc 2	1		1	1	1	1				
Doc 3					1	1	1	2	1	1

http://datameetsmedia.com/bag-of-words-tf-idf-explained/

Word Representation and Feature Extraction
Bag of Words

Term-document count matrices

1. **I love dogs.**
2. **I hate dogs and knitting.**
3. **Knitting is my hobby and my passion.**

Consider the number of occurrences of a term in a document

	I	love	dogs	hate	and	knitting	is	my	hobby	passion
Doc 1	1	1	1							
Doc 2	1		1	1	1	1				
Doc 3					1	1	1	2	1	1

Question: Can we know the signature/important words in each document?

http://datameetsmedia.com/bag-of-words-tf-idf-explained/

Word Representation and Feature Extraction
Bag of Words

Tf-idf Matrix: **term frequency** – inverse document frequency

- Term frequency (tf) is basically the output of the BOW model.
- Term frequency measures the local importance of the word. If a word appears a lot of times, then the word must be important.
- For example, if our document is "***I am a cat lover. I have a cat. I feed the cat outside my room regularly***," we see that the words with the highest frequency are **I** and **cat**. This agrees with our intuition that high term frequency = higher importance since the document is all about my fascination with cats.

http://datameetsmedia.com/bag-of-words-tf-idf-explained/

Word Representation and Feature Extraction
Bag of Words

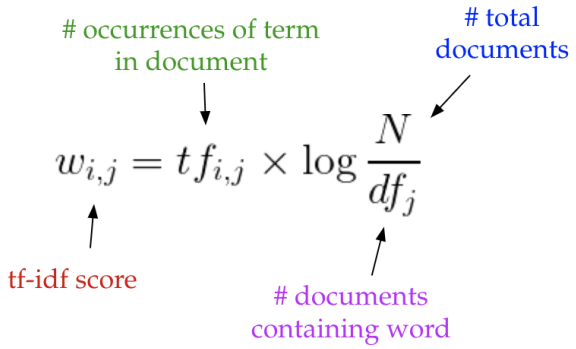
Tf-idf Matrix: term frequency – **inverse document frequency**

- The second component of tf-idf is inverse document frequency (idf). For a word to be considered a signature word of a document, it shouldn't appear that often in the other documents.
- Thus, a signature word's document frequency must be low, meaning its inverse document frequency must be high.
- The idf is calculated as $\text{idf(W)} = \log \frac{\#(\text{documents})}{\#(\text{documents containing word W})}$

http://datameetsmedia.com/bag-of-words-tf-idf-explained/

Word Representation and Feature Extraction
Bag of Words

Tf-idf assigns a weight for term *j* in document *i* as follows:



Intuitively, a term has a large weight when it occurs frequently across the *document* but infrequently across the *corpus*.

Consider two extreme cases:

1. A term/word appears in every docu
2. A term/word only appears once in one document in total

What are the values for tf-idf for 1 and 2?

Word Representation and Feature Extraction
Bag of Words

Tf-idf Matrix: **Binary → count → weight matrix**

- For a word to have high tf-idf in a document, it must appear a lot of times in said document and must be absent in the other documents. It must be a signature word of the document.

	I	love	dogs	hate	and	knitting	is	my	hobby	passion
Doc 1	0.18	**0.48**	0.18							
Doc 2	0.18		0.18	**0.48**	0.18	0.18				
Doc 3					0.18	0.18	**0.48**	**0.95**	**0.48**	**0.48**

We can see here that tf-idf highlights **love** as a signature word in the first document, **hate** in the second, and **is**, **my**, **hobby** and **passion** in the third.

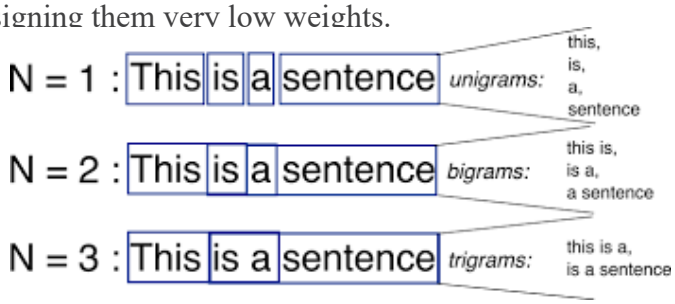
Robinson
http://datameetsmedia.com/bag-of-words-tf-idf-explained/

Word Representation and Feature Extraction
Bag of Words

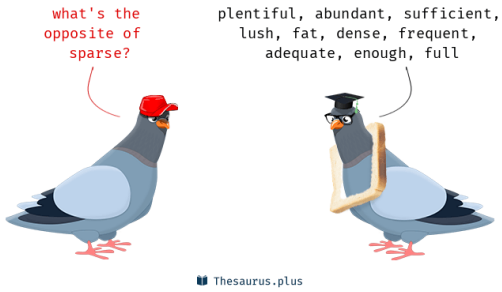
Ngram Features

- Consecutive word sequences of a given length
- Bag of bigrams representation is much more powerful than bag of words; for example, New York, not bad, etc.
- It is very hard to know a-priori which ngrams will be useful for a given task. The common solution is to include all ngrams up to a given length, and let the model regularization discard of the less interesting ones by assigning them very low weights.

Setting up a fixed window size, multi-layer perceptions can infer ngrams.



Potential Issues of Bag of Words Approach

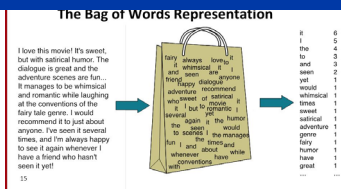


Word Representation and Feature Extraction

Bag of Words

Documents as vectors

- So we have a |V|-dimensional vector space
- Very high-dimensional: tens of millions of dimensions when you apply this to a web search engine
- These are very sparse vectors - most entries are zero.




```
> inspect(tdm[20:30, 1:30])
<<TermDocumentMatrix (terms: 11, documents: 30)>>
Non-/sparse entries: 1/329
Sparsity           : 100%
Maximal term length: 7
Weighting          : term frequency (tf)

         Docs
Terms     1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
  abs     0 0 0 0 0 0 0 0 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
  absb    0 0 0 0 0 0 0 0 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
  absenc  0 0 0 0 0 0 0 0 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
  absolut 0 0 0 0 0 0 0 0 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
  absorb  0 0 0 0 0 0 0 0 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
  abu     0 0 0 0 0 0 0 0 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
  abus    0 0 0 0 0 0 0 0 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
  abut    0 0 0 0 0 0 0 0 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
  academi 0 0 0 0 0 0 0 0 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
  acceler 0 0 0 0 0 0 0 0 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
  accept  0 0 0 0 0 0 0 0 0  0  0  0  0  0  0  0  0  1  0  0  0  0  0  0  0  0  0  0  0  0
>
```

Word Representations: Word Relationships

- I need to stay in a hotel for my travel.
- I am looking for a motel for my travel.

In the above two sentences, the relationship between hotel and motel is____.

When people search for hotel, then motel info needs to be displayed as well.

https://web.stanford.edu/~jurafsky/li15/lec3.vector.pdf

Word Representations: Word2Vec

Distributional Features

- The distributional hypothesis of language states that the ***meaning of a word can be inferred from the contexts in which it is used.***
- By observing co-occurance patterns of words across a large body of text, we can discover that the contexts in which burger occur are quite similar to those in which pizza occurs, less similar to those in which ice cream occurs, and very different from those in which chair occurs.

Text representation

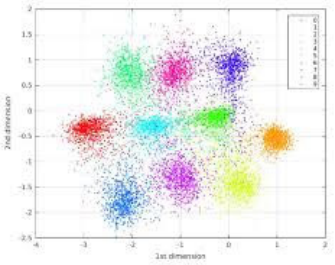
- A standard approach: Bag-Of-Words.
  - A document as a list of words it contains.
- Much more sophisticated method: distributional clusters.
  - A word is represented as a distribution over the categories.
  - The words are then clustered to k clusters.
  - Details will go later on.

Word Representations: Word2Vec

Distributional Features

- Algorithms that learn generalizations of words based on the contents they occur are broadly categorized two methods:
  - ❖ Clustering-based methods: assign similar words to the same cluster and represent each word by its cluster index
  - ❖ Embedding-based methods: represent each word as a vector such that similar words (words having a similar distribution) have similar vectors.
- The Matrix is generally dense
- Distributed features are extremely useful for the NER task.

Figure 1: An example of NER application on an example text



Word Representations: Word2Vec

- **Singular Value Decomposition (SVD) – A special case of this is called LSA (Latent Semantic Analysis)**
- **Neural Language Model**
  - ❖ Skip-grams
  - ❖ CBOW
- **Brown clustering:**
  - ❖ Clustering words based on which words precede or follow them
  - ❖ These word clusters can be turned into a kind of vector

Word Representations: Word2Vec

Other names:

- distributional semantic model since the underlying semantic theory is called distributional semantics in computational linguistics;
- Distributed representation
- Semantic vector space
- Word space

The Eye

Source: A. Efros

Find the Word Embeddings

To find these distributed word representations, we need to design algorithm to train the machine learning task, question is what are the objectives/loss/costs functions?

Language Modelling

The Language Modelling Task

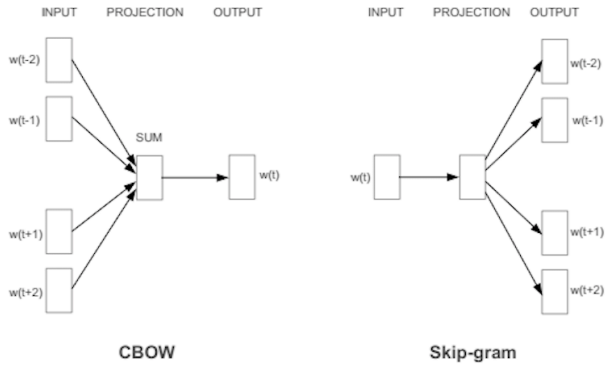
- Language modeling is the task of assigning a probability to sentences in a language.
  - For example, what is the probability of seeing the sentence "*the lazy dog barked loudly?*"

- The language models also assign a probability for the likelihood of a given word (or a sequence of words) to follow a sequence of words.
  - For example, what is the probability of seeing the word *barked* after the seeing sequence *the lazy dog*?

Word Representations: Word2Vec

Skip-Gram Versus CBOW

- Use the current word to predict its neighbors (its context)---"Skip-Gram"
- Uses each of contexts/neighbors to predict the current word -- "Continuous Bag Of Words" (CBOW)
- The limit on the number of words in each context is determined by a parameter called "window size".



https://towardsdatascience.com/word2vec-skip-gram-model-part-1-intuition-78614e4d6e0b

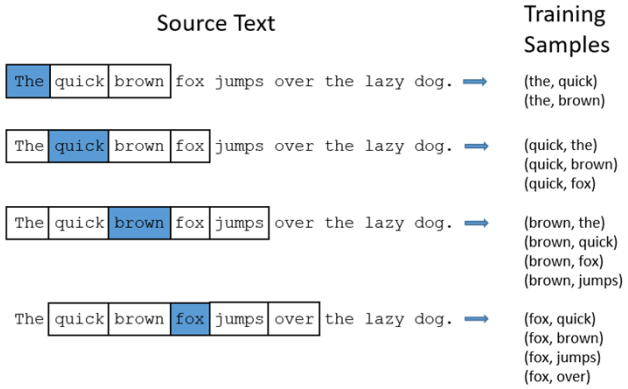
Word Representations: Skip-gram

- The skip-gram framework uses a simple neural network model with a single hidden layer.
- The goal for the neural network is to learn the weights of the hidden layer–these weights are actually the "word vectors" that we're trying to learn.
- The output probabilities are going to relate to how likely it is to find each vocabulary word nearby our input word. For example, if you gave the trained network the input word "artificial", the output probabilities are going to be much higher for words like "intelligence" than for unrelated words like "watermelon" and "kangaroo".

https://towardsdatascience.com/word2vec-skip-gram-model-part-1-intuition-78614e4d6e0b

Word Representations: Skip-gram

Training Input



- train the neural network by feeding word pairs found in our training documents.
- The left example shows some of the training samples (word pairs) we would take from the sentence
- The network is going to learn the statistics from the number of times each pairing shows up

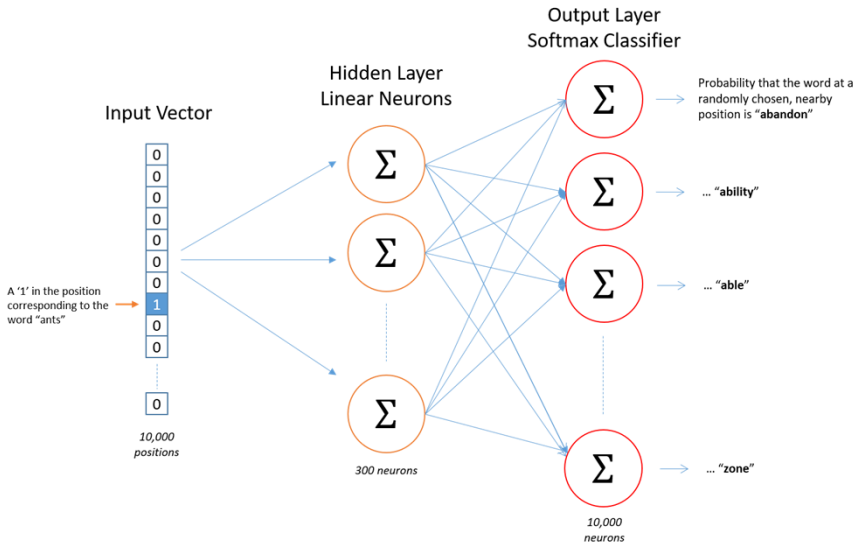
The window size in this example = 5, focal plus two in the left and two in the right

https://towardsdatascience.com/word2vec-skip-gram-model-part-1-intuition-78614e4d6e0b

Word Representations: Word2Vec

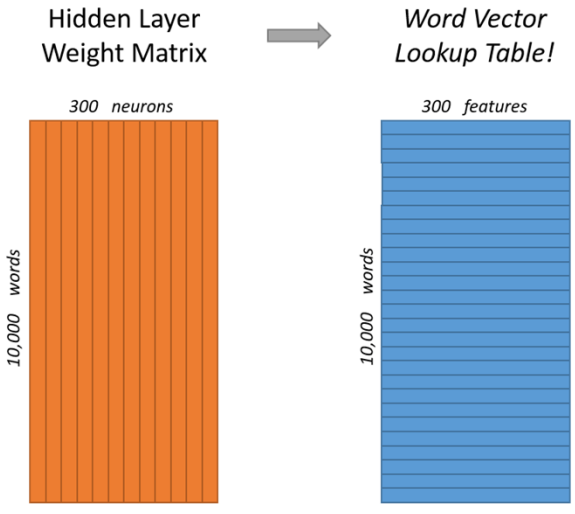
One-Hot Encodings

- Consider a vocabulary of 40,000 items, it will be a 40,000 dimensional vector for each word, where there is only 1 in this vector and the rest of 39,999 values are all 0
- Dimensionality of one-hot vector the same as the number of distant features/words/terms



https://towardsdatascience.com/word2vec-skip-gram-model-part-1-intuition-78614e4d6e0b

Word2vec Training



For our example, we're going to say that we're learning word vectors with 300 features. So the hidden layer is going to be represented by a weight matrix with 10,000 rows (one for every word in our vocabulary) and 300 columns (one for every hidden neuron).

If you look at the *rows* of this weight matrix, these are actually what will be our word vectors!

http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/

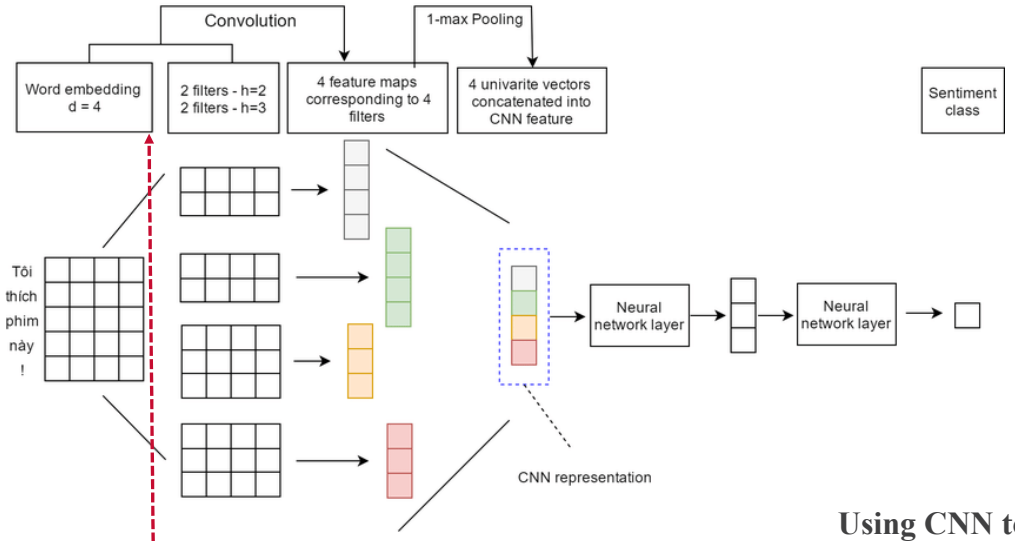
Word Embeddings

Dense Encodings/Feature Embeddings – Another way to get word2vec

- Through non-linear models such as RNN, we stop representing each feature as a unique dimension in a one-hot vector, and instead represent features as dense vectors;
- Each core feature is *embedded* into a d dimensional space and being represented as a vector in that space.
- d is normally much smaller than the number of features.
- The embeddings (the vector representation of each core feature) are treated as parameters of the network, and are trained.
- Model training will cause similar features to have similar vectors.
- Vector models are also called "embeddings"

First layer in deep learning models

Word Embeddings

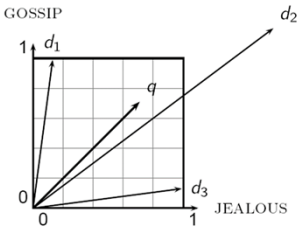


Using CNN to predict sentiment

Word Representations

- The key idea of word/feature embeddings is that similar words have similar vectors. But the question is how to define similarity.
- Distributed hypothesis states that words are similar if they appear in similar contexts. Further, if A and B have almost identical environments we say that they are **synonyms**.
- A popular and effective measure is the **cosine similarity**, measuring the cosine of the angle between the vectors:

$$\text{sim}(\mathbf{u},\mathbf{v}) = \frac{\mathbf{u}\cdot\mathbf{v}}{\|\mathbf{u}\|_2 \cdot \|\mathbf{v}\|_2} = \frac{\sum_i u_i \cdot v_i}{\sqrt{\sum_i u_i^2} \cdot \sqrt{\sum_i v_i^2}}$$



Vector distance?

Word Representations

$$\cos(\vec{v},\vec{w}) = \frac{\vec{v}\bullet\vec{w}}{|\vec{v}||\vec{w}|} = \frac{\vec{v}}{|\vec{v}|}\bullet\frac{\vec{w}}{|\vec{w}|} = \frac{\sum_{i-1}^{N} v_i w_i}{\sqrt{\sum_{i=1}^{N} v_i^2}\sqrt{\sum_{i=1}^{N} w_i^2}}$$

	large	data	computer
Apricot	2	0	0
Digital	0	1	2
information	1	6	1

Which pair of words is more similar?

cosine(apricot, information) = $\frac{2+0+0}{\sqrt{2+0+0}\sqrt{1+36+1}} = \frac{2}{\sqrt{2}\sqrt{38}} = 0.23$

cosine(digital, information) = $\frac{0+6+2}{\sqrt{0+1+4}\sqrt{1+36+1}} = \frac{8}{\sqrt{5}\sqrt{38}} = 0.58$

cosine(apricot, digital) = $\frac{0+0+0}{\sqrt{4+0+0}\sqrt{0+1+4}} = \frac{0}{\sqrt{4}\sqrt{5}} = 0.00$

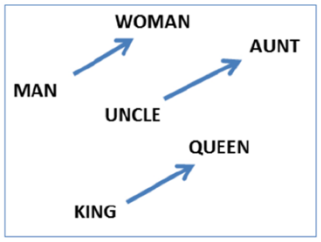
https://web.stanford.edu/~jurafsky/li15/lec3.vector.pdf

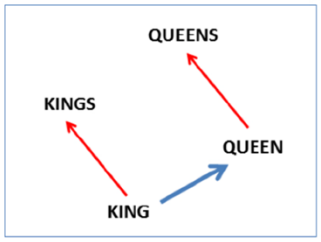
Word Embeddings

Word Analogies:

Embeddings capture relational meaning!

vector('*king*') - vector('*man*') + vector('*woman*') ≈ vector('queen')
vector('*Paris*') - vector('*France*') + vector('*Italy*') ≈ vector('Rome')



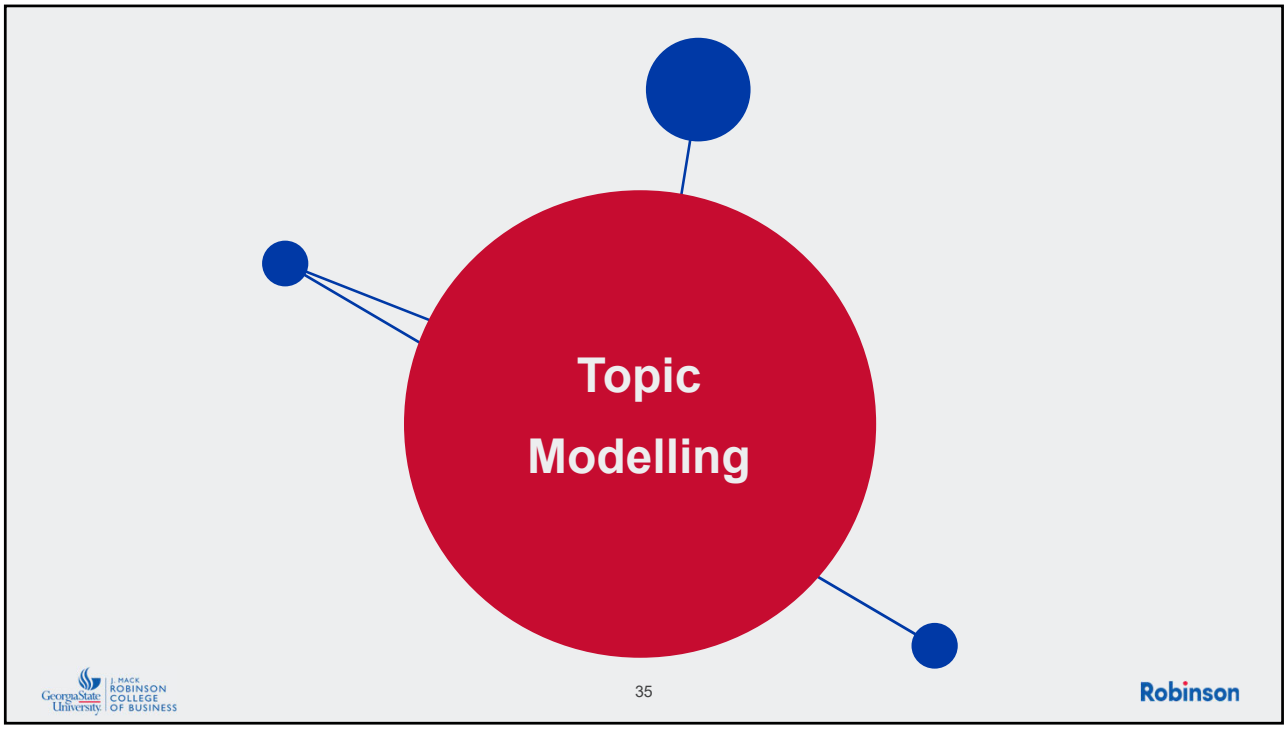


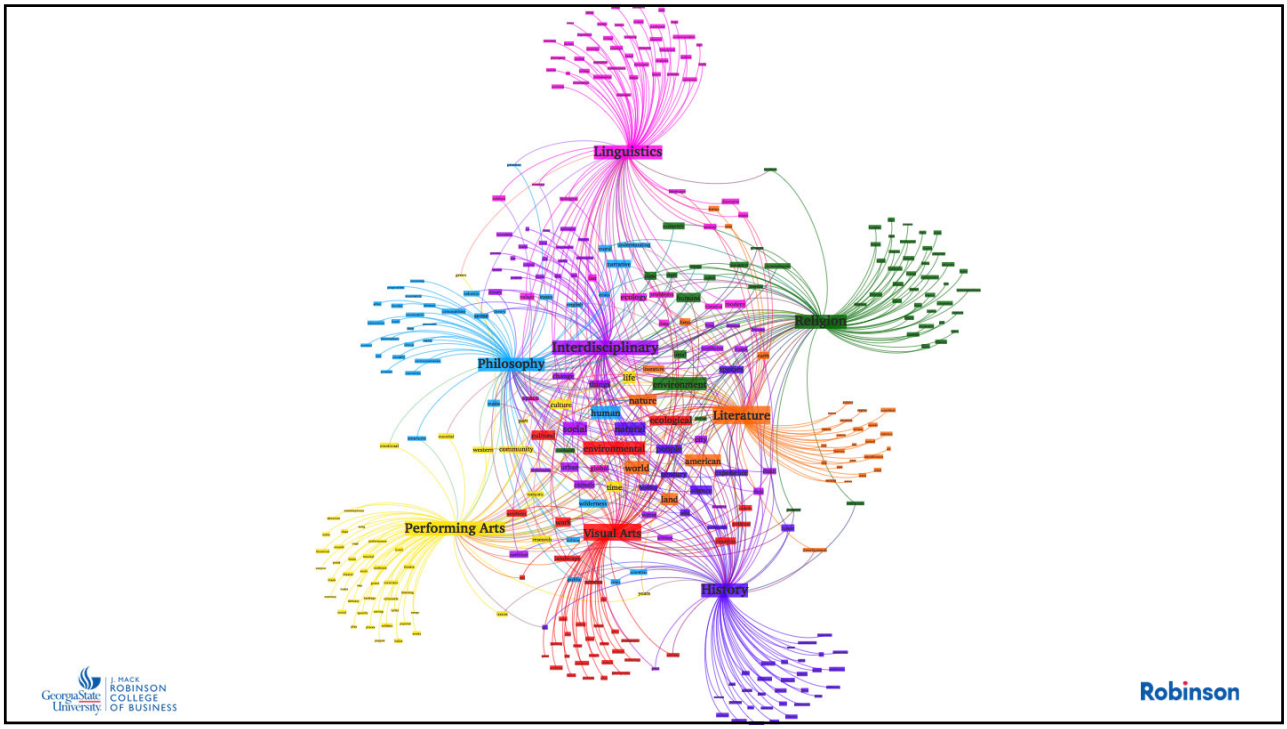
Word Embeddings

Similar Documents:

- To find the similarity between two documents, we represent each document as a bag-of-words, and define the similarity between the documents to be the sum of the pairwise similarities between the words in the documents.
- In particular, consider two documents $D_1 = w_1^1, w_2^1, \cdots, w_m^1$ and $D_2 = w_1^2, w_2^2, \cdots, w_n^2$. The document similarity is

$$\text{Sim}_{\text{doc}}(D_1, D_2) = \sum_{i=1}^{m} \sum_{j=1}^{n} \cos\left(\mathbf{w}_i^1, \mathbf{w}_j^2\right)$$


Topic
Modelling
35
Robinson


Linguistics
Religion
Interdisciplinary
Philosophy
Literature
Performing Arts
Visual Arts
History
Robinson

Topic Model

Real world example: The New York Times

LDA analysis of 1.8M New York Times articles:

music band songs rock album jazz pop song singer night	book life novel story books man stories love children family	art museum show exhibition artist artists paintings painting century works	game knicks nets points team season play games night coach	show film television movie series says life man character know
theater play production show stage street broadway director musical directed	clinton bush campaign gore political republican dole presidential senator house	stock market percent fund investors funds companies stocks investment trading	restaurant sauce menu food dishes street dining dinner chicken served	budget tax governor county mayor billion taxes plan legislature fiscal

Topic Modeling

- Roughly speaking, a topic is the main idea discussed in text data
  - A theme or subject of a discussion or a conversation
  - Different granularities: the topic of a sentence, the topic of a paragraph, the topic of an article, the topic of all the research articles in a library.

Topic Modeling

Many applications require discovery and analysis of topics in text

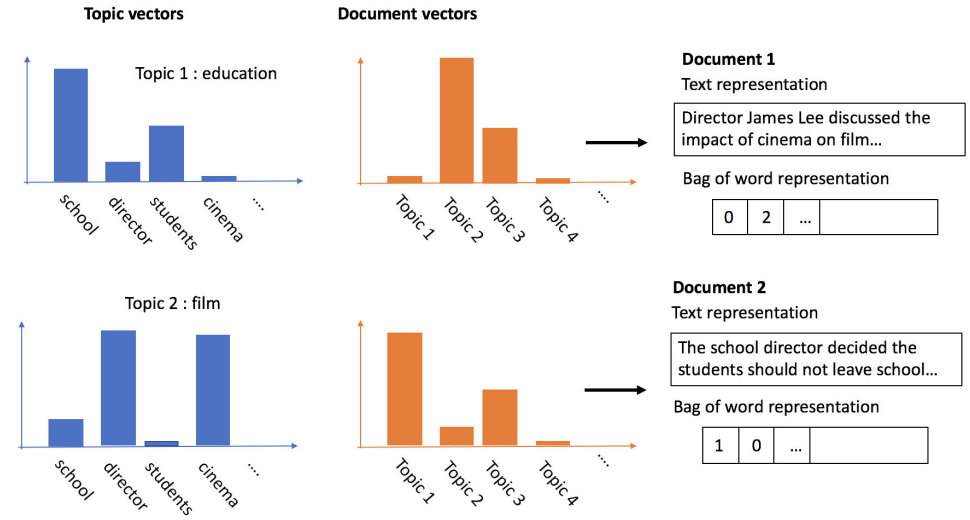
- What Twitter users are talking about today? NBA sports, international events, #UnitedAirlines
- What people like about some products, such as smart phones.
  - Topics in both positive reviews and negative reviews

Topic Modeling

- Tasks of topic modeling and mining
  - Discover the k topics from a collection of text
  - Figure out which documents covers which topic to what extent
- All topic models are based on the same basic assumption:
  - Each document consists of a mixture of topics, and
  - Each topic consists of a collection of words.

Topic Model

- The image illustrates the LDA model visually.



Georgia State University J. Mack Robinson College of Business

Robinson

Inferring the Topic from Keywords

Georgia State University J. Mack Robinson College of Business

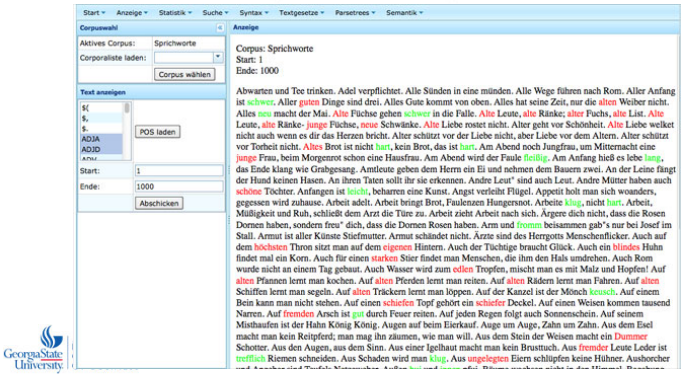
42

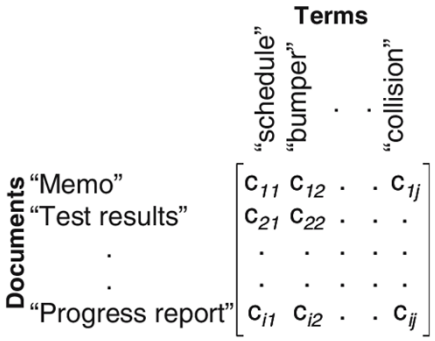
Robinson

Topic Model

Text Mining – Some Definitions

- Document – a sequence of words and punctuation, following the grammatical rules of the language.
- Term – usually a word, but can be a word-pair or phrase
- Corpus – a collection of documents
- Lexicon – set of all unique words in corpus





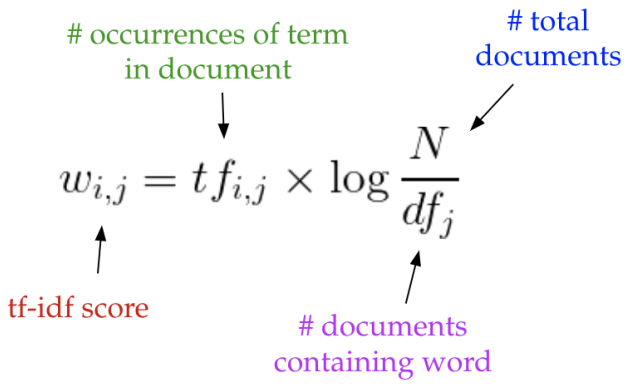
LSA

- Latent Semantic Analysis, or LSA, is one of the foundational techniques in topic modeling.
- The core idea is to take a matrix of documents and terms, and decompose it into a separate document-topic matrix and a topic-term matrix.
- The first step is to generate our document-term matrix. Given ***m*** documents and ***n*** words in our vocabulary, we can construct an m × n matrix A in which each row represents a document and each column represents a word.
- In the simplest version of LSA, each entry can simply be a raw count of the number of times the j-th word appeared in the i-th document. In practice, however, raw counts do not work particularly well because they do not account for the significance of each word in the document. For example, the word "nuclear" probably informs us more about the topic(s) of a given document than the word "test."

https://medium.com/nanonets/topic-modeling-with-lsa-psla-lda-and-lda2vec-555ff65b0b05

LSA

Consequently, LSA models typically replace raw counts in the document-term matrix with a tf-idf score. Tf-idf, or term frequency-inverse document frequency, assigns a weight for term j in document i as follows:



https://medium.com/nanonets/topic-modeling-with-lsa-psla-lda-and-lda2vec-555ff65b0b05

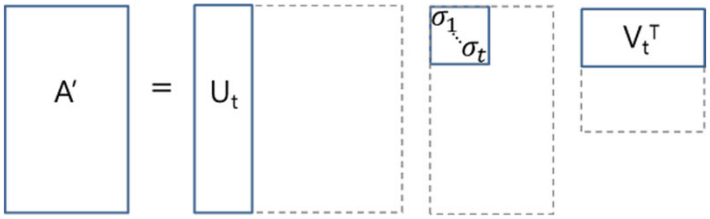
LSA

Since the term-document matrix is sparse, we do truncated ***SVD*** (singular value decomposition), which is a technique in linear algebra that factorizes any matrix M into the product of 3 separate matrices: $M = U \times S \times V$, where S is a diagonal matrix of the singular values of M. Critically, truncated SVD reduces dimensionality by selecting only the t largest singular values, and only keeping the first t columns of U and V. In this case, t is a hyperparameter we can select and adjust to reflect the number of topics we want to find.

$$A \approx U_t S_t V_t^T$$

https://medium.com/nanonets/topic-modeling-with-lsa-psla-lda-and-lda2vec-555ff65b0b05

LSA



In this case, $U \in \mathbb{R}$^(m × t) emerges as our document-topic matrix, and $V \in \mathbb{R}$^(n × t) becomes our term-topic matrix. In both U and V, the columns correspond to one of our t topics. In U, rows represent document vectors expressed in terms of topics; in V, rows represent term vectors expressed in terms of topics.

With these document vectors and term vectors, we can apply measures such as cosine similarity to evaluate 1) the similarity of different documents; 2) the similarity of different words; 3) the similarity of terms (or "queries") and documents.

https://medium.com/nanonets/topic-modeling-with-lsa-psla-lda-and-lda2vec-555ff65b0b05

LSA

LSA is quick and efficient to use, but it does have a few primary drawbacks:

- lack of interpretable embeddings (we don't know what the topics are, and the components may be arbitrarily positive/negative)
- need for really large set of documents and vocabulary to get accurate results
- less efficient representation

https://medium.com/nanonets/topic-modeling-with-lsa-psla-lda-and-lda2vec-555ff65b0b05

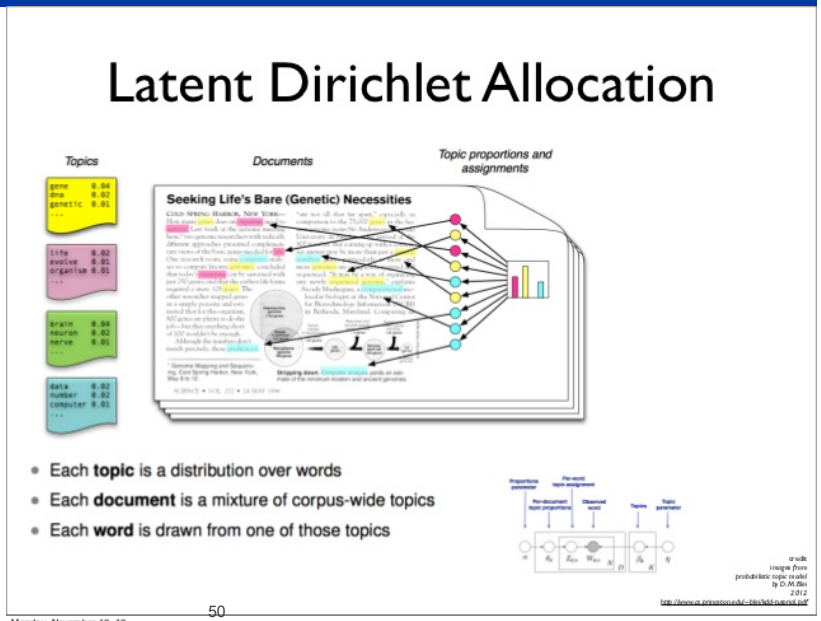
LSA

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.decomposition import TruncatedSVD
from sklearn.pipeline import Pipeline

documents = ["doc1.txt", "doc2.txt", "doc3.txt"]

# raw documents to tf-idf matrix:
vectorizer = TfidfVectorizer(stop_words='english', use_idf=True,  smooth_idf=True)

# SVD to reduce dimensionality:
svd_model = TruncatedSVD(n_components=100,        // num dimensions
                         algorithm='randomized',
                         n_iter=10)

# pipeline of tf-idf + SVD, fit to and applied to documents:
svd_transformer = Pipeline([('tfidf', vectorizer),  ('svd', svd_model)])

svd_matrix = svd_transformer.fit_transform(documents)

# svd_matrix can later be used to compare documents, compare words, or compare queries with documents
```


https://medium.com/nanonets/topic-modeling-with-lsa-psla-lda-and-lda2vec-555ff65b0b05

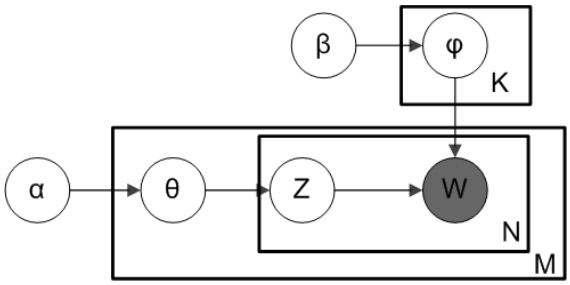
LDA

- LDA stands for Latent Dirichlet Allocation. LDA uses Dirichlet as Bayesian priors for the document-topic and word-topic distributions, lending itself to better generalization.



LDA

For the model of LDA, from a dirichlet distribution Dir(α), we draw a random sample representing the *topic distribution*, or topic mixture, of a particular document. This topic distribution is θ. From θ, we select a particular topic Z based on the distribution.



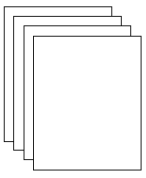
Word Distribution

Topic Distribution

from another dirichlet distribution Dir(β), we select a random sample representing the *word distribution* of the topic Z. This word distribution is φ. From φ, we choose the word w.

Latent Dirichlet Allocation

- When training an LDA model, you start with a collection of documents and each of these is represented by a fixed-length vector (bag-of-words).
- To implement an LDA model, you first start by defining the number of 'topics' that are present in your collection of documents
- N documents, M topics, V total number of words in all documents

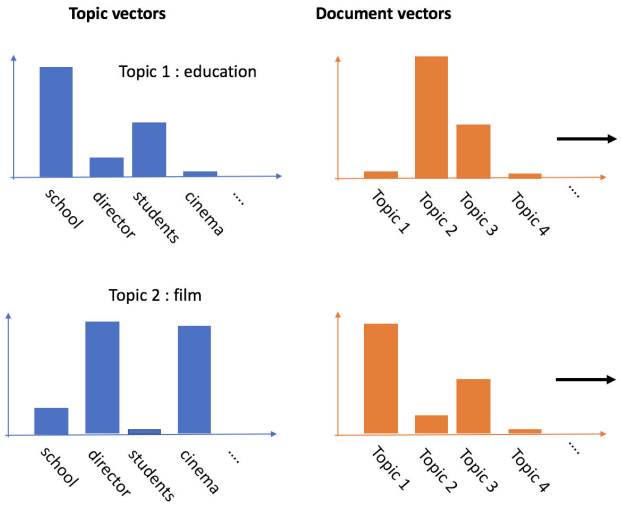


N documents constructed from a vocabulary of V words

Represented as N V-dimensional vectors (bag-of-word)

Goal: Find the document and topic vectors which explain the observed data

Latent Dirichlet Allocation



- The image illustrates the LDA model visually.
- ***The goal of the model is to find the topic and document vectors that explain the original bag-of-word representation of the different documents.***

Latent Dirichlet Allocation

LDA assumes the following generative process for a corpus D consisting of M documents each of length N_i:

1. Choose $\theta_i \sim \text{Dir}(\alpha)$ where $i \in \{1, \cdots, M\}$ and $\text{Dir}(\alpha)$ is a Dirichlet distribution with a symmetric parameter α ($\alpha < 1$)

2. Choose $\psi_i \sim \text{Dir}(\beta)$ where $k \in \{1, \cdots, K\}$ and $\text{Dir}(\alpha)$ is a Dirichlet distribution with a symmetric parameter α ($\alpha < 1$)

3. For each of the word positions i, j where $i \in \{1, \cdots, M\}$ and $j \in \{1, \cdots, N_i\}$

   a) Choose a topic $z_{i,j} \sim Multinomial(\theta_i)$

   b) Choose a word $w_{i,j} \sim Multinomial(\psi_{i,j})$

Latent Dirichlet Allocation

$$P(\boldsymbol{W}, \boldsymbol{Z}, \boldsymbol{\Theta}, \boldsymbol{\Psi}; \alpha, \beta) = \prod_{i=1}^{K} P(\psi_i; \beta) \prod_{j=1}^{M} P(\theta_j; \alpha) \prod_{t=1}^{N} P(Z_{j,t}|\theta_j) P(W_{j,t}|\psi_{j,t})$$

Word dist. — Topic dist. — topics — words

Dirichlet **Dirichlet** ***Multinomial*** ***Multinomial***

Latent Dirichlet Allocation

- It is important to notice that you are relying on the assumption that the topic vectors will be interpretable, otherwise the output of the model is pretty much garbage.
- Essentially, you are assuming that the model, given enough data, will figure out which words tend to co-occur, and will cluster them into distinct 'topics'.
- You have to determine a good estimate of the number of topics that occur in the collection of the documents.
- In addition, you have to manually assign a distinct nominator/'topic' to the different topic vector.

LDA

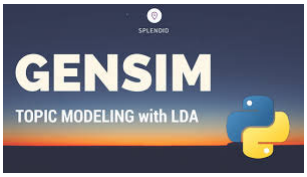
- You have to determine a good estimate of the number of topics that occur in the collection of the documents.
- In addition, you have to manually assign a distinct nominator/'topic' to the different topic vector.

Too many topics: → heterogeneous set of words

Too few topics: → diffuse the information with the same words shared across many topics

https://medium.com/@alexisperrier/topic-modeling-of-twitter-timelines-in-python-bb91fa90d98d

LDA Model



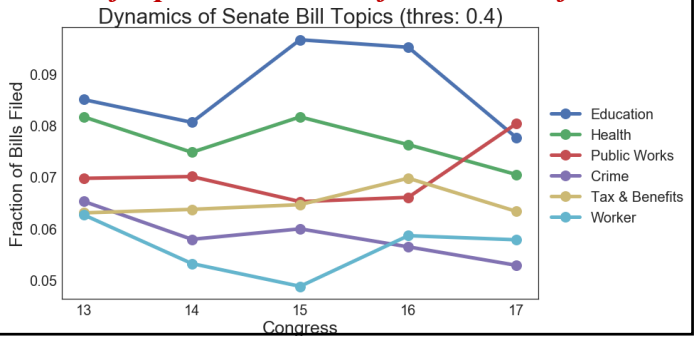
https://www.machinelearningplus.com/nlp/topic-modeling-gensim-python/
https://www.machinelearningplus.com/nlp/topic-modeling-visualization-how-to-present-results-lda-models/

https://www.machinelearningplus.com/nlp/topic-modeling-python-sklearn-examples/

Dynamic Topic Model (STM)

- **Dynamic topic models** are generative models that can be used to analyze the evolution of (unobserved) topics of a collection of documents over time.
- By David Blei and John Lafferty and is an extension of Latent Dirichlet Allocation (LDA) that can handle sequential documents.

In a dynamic topic model the order of the documents plays a fundamental role. More precisely, the documents are grouped by time slice (e.g.: years) and it is assumed ***that the documents of each group come from a set of topics that evolved from the set of the previous slice***.



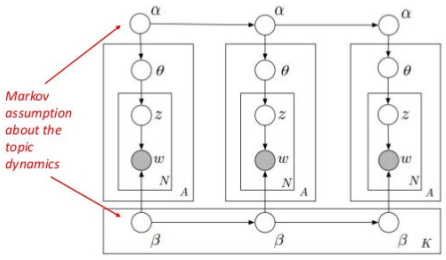
Dynamic Topic Model

Similarly to LDA, in a dynamic topic model, each document is viewed as a mixture of unobserved topics. Furthermore, each topic defines a multinomial distribution over a set of terms. Thus, for each word of each document, a topic is drawn from the mixture and a term is subsequently drawn from the multinomial distribution corresponding to that topic.

Dynamic Topic Model [Blei ICML'06]

The topics, however, evolve over time

- Capture the evolving topics over time

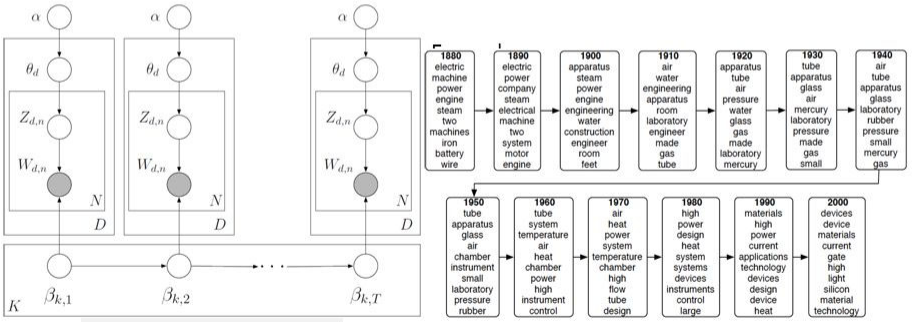


Dynamic LDA

https://towardsdatascience.com/exploring-the-un-general-debates-with-dynamic-topic-models-72dc0e307696

https://github.com/llefebure/un-general-debates/tree/master/notebooks

- Motivation:
  - LDA assumes the order of documents does not matter (Not appropriate for sequential corpora)
  - We want to capture language change over time.
- In Dynamic LDA, topic evolves over time.
  - Dynamic LDA uses a logistic normal distribution to model topics evolving over time.



Topics drift through time

Blei, D. M., and Lafferty, J. D. 2006. "Dynamic topic models," in *Proceedings of the 23rd international conference on Machine learning (ICML 2006)*, pp. 113–120 (doi: 10.1145/1143844.1143859).

Dynamic Topic Model

https://markroxor.github.io/gensim/static/notebooks/ldaseqmodel.html

https://towardsdatascience.com/exploring-the-un-general-debates-with-dynamic-topic-models-72dc0e307696

LDA2vec: Word Embeddings in Topic Models

- The general goal of a topic model is to produce interpretable document representations which can be used to discover the topics or structure in a collection of unlabelled documents.
- An example of such an interpretable document representation is: document X is 20% topic a, 40% topic b and 40% topic c.
- LDA is a probabilistic topic model and it treats documents as a bag-of-words
- lda2vec builds document representations on top of word embeddings

Topic Models with Word2vec: LDA2Vec

- The word2vec model learns a word vector that predicts context words across different documents. As a result, document-specific information is mixed together in the word embeddings.
- Inspired by Latent Dirichlet Allocation (LDA), the word2vec model is expanded to simultaneously learn word, document and topic vectors.
- Lda2vec is obtained by modifying the skip-gram word2vec variant. In the original skip-gram method, the model is trained to predict context words based on a pivot word.
- In lda2vec, the pivot word vector and a document vector are added to obtain a context vector. This context vector is then used to predict context words.

Robinson

LDA2VEC

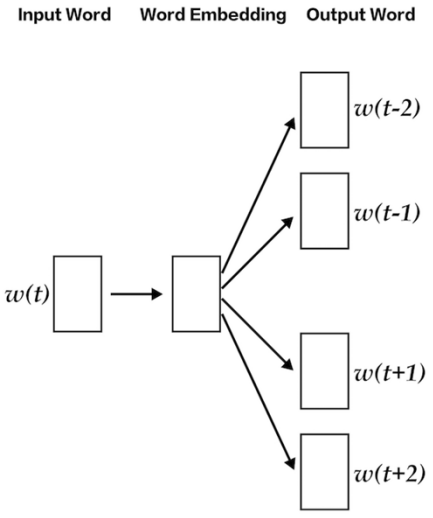
- Consider the following example: in the original word2vec model, if the pivot word is 'French', then possible context words might be 'German', 'Dutch', 'English'. Without any global (document-related) information, these would be the most plausible guesses.
- By providing an additional context vector in the lda2vec model, it is possible to make better guesses of context words.
- If the document vector is a combination of the ‘food’ and ‘drinks’ topics, then 'baguette', 'cheese' and 'wine' might be more suitable. If the document vector is similar to the 'city' and ‘geography’ topics, then 'Paris', 'Lyon' and 'Grenoble' might be more suitable.

https://lda2vec.readthedocs.io/en/latest/?badge=latest https://github.com/meereeum/lda2vec-tf

https://www.datacamp.com/community/tutorials/lda2vec-topic-model

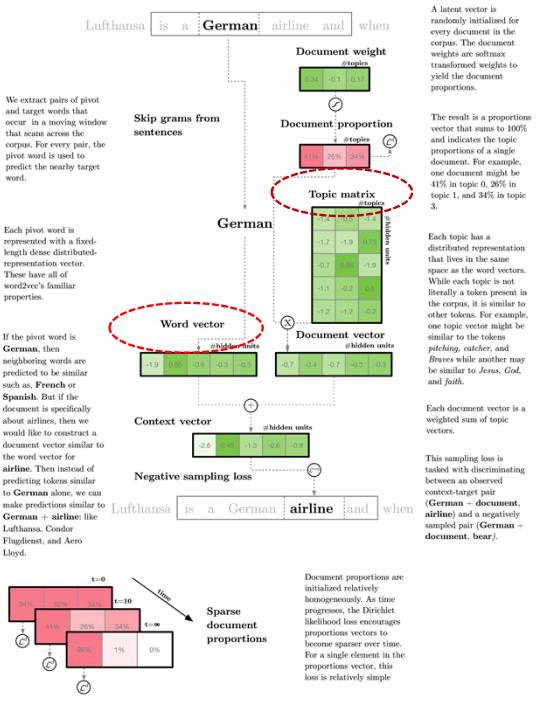
LDA2VEC

- lda2vec is an extension of word2vec and LDA that jointly learns word, document, and topic vectors.
- lda2vec specifically builds on top of the skip-gram model of word2vec to generate word vectors.
- With lda2vec, instead of using the word vector directly to predict context words, we leverage a *context vector* to make the predictions. This context vector is created as the sum of two other vectors: the word vector and the document vector.



https://medium.com/nanonets/topic-modeling-with-lsa-psla-lda-and-lda2vec-555ff65b0b05

LDA2VEC



https://multithreaded.stitchfix.com/assets/posts/2016-05-27-lda2vec/lda2vec_network_publish_text_header.gif

LDA2VEC

The following link talks about technical details:

https://multithreaded.stitchfix.com/blog/2016/05/27/lda2vec/#topic=38&lambda=1&term=

The following link provides a Tensorflow implementation:

https://github.com/meereeum/lda2vec-tf

An overview of the lda2vec Python module

https://lda2vec.readthedocs.io/en/latest/?badge=latest

Topic Model Evaluation

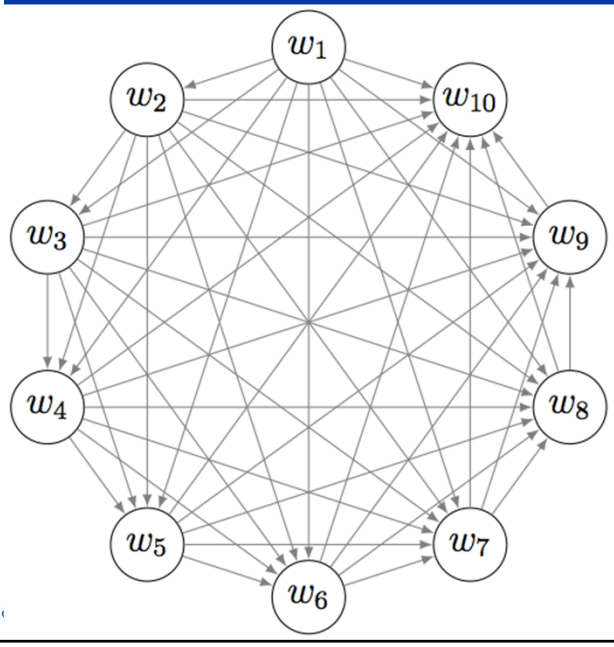
"Trust me on this."

Topic Model Evaluation

- Look at collections of words that are associated with topics.
- Examine actual documents that are estimated to be highly associated with each topic.

https://towardsdatascience.com/evaluate-topic-model-in-python-latent-dirichlet-allocation-lda-7d57484bb5d0

Topic Model Evaluation: Topic Coherence



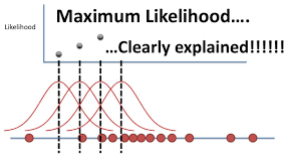
Coherence= $\sum_{i<j} \mathbf{score}(w_i, w_j)$

Which is equal to the sum of pairwise scores on the words $w_1 \cdots, w_n$, used to describe the topic, usually the top n words by frequency. This measure can be seen as the sum of all edges on complete graph.

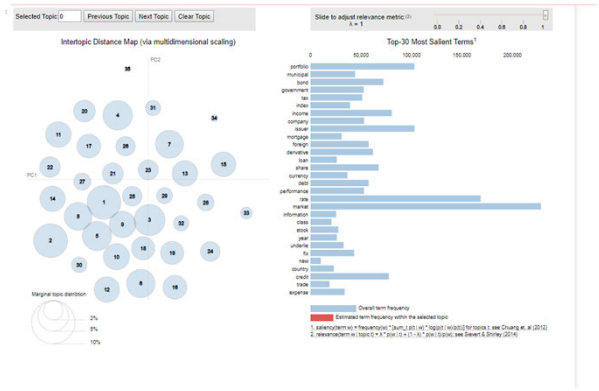
Robinson
http://qpleple.com/topic-coherence-to-evaluate-topic-models/

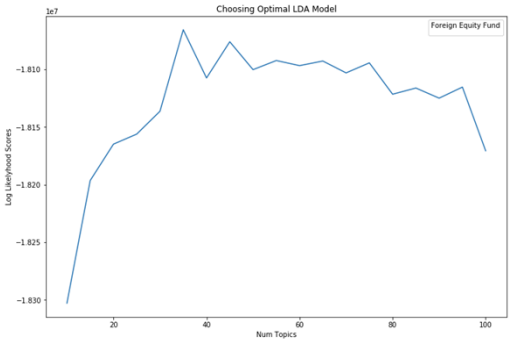
Topic Model Evaluation: Log-likelihood Comparison

$$P(\boldsymbol{W}, \boldsymbol{Z}, \boldsymbol{\Theta}, \boldsymbol{\Psi}; \alpha, \beta) = \prod_{i=1}^{K} P(\psi_i; \beta) \prod_{j=1}^{M} P(\theta_j; \alpha) \prod_{t=1}^{N} P(Z_{j,t}|\theta_j) P(W_{j,t}|\psi_{j,t})$$

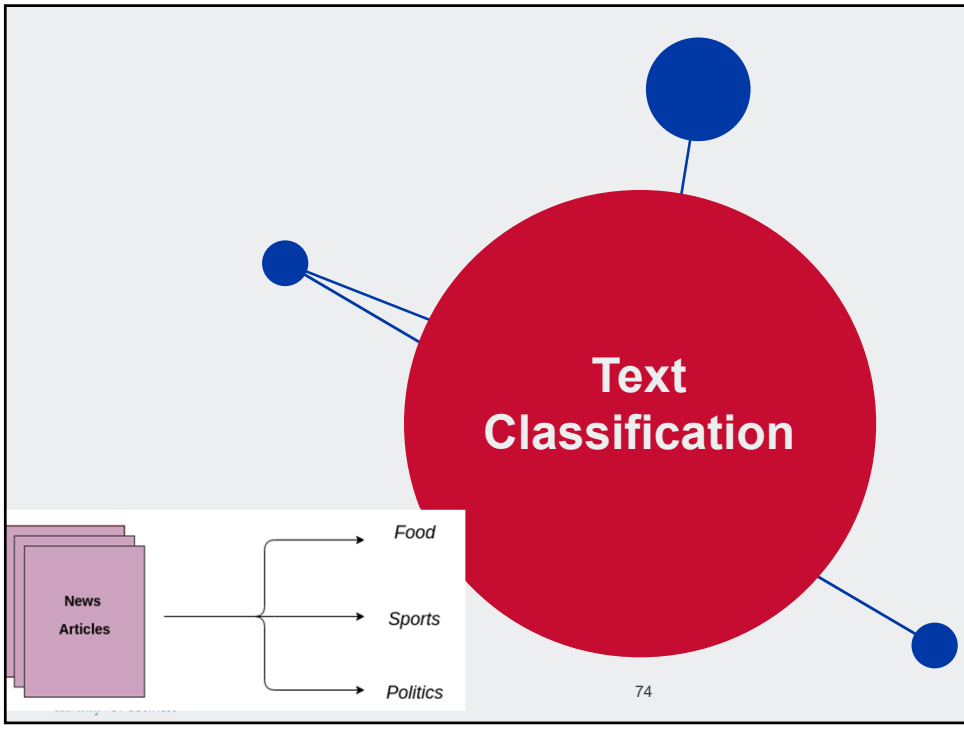


Topic Model Evaluation





Question: What is a good LDA model output?



Text Classification

Text classification is to automatically classify the text documents into one or more defined categories. Some examples of text classification are:

1. Understanding audience sentiment from social media,
2. Detection of spam and non-spam emails,
3. Auto tagging of customer queries, and
4. Categorization of news articles into defined topics.

https://www.analyticsvidhya.com/blog/2018/04/a-comprehensive-guide-to-understand-and-implement-text-classification-in-python/

Text Classification

Text Classification is an example of supervised machine learning task since a labelled dataset containing text documents and their labels is used for training a classifier. An end-to-end text classification pipeline is composed of several main components:

1. Dataset Preparation:
2. Feature Engineering:
3. Model Training:
4. Improve Performance of Text Classifier:

https://www.analyticsvidhya.com/blog/2018/04/a-comprehensive-guide-to-understand-and-implement-text-classification-in-python/

Text Classification

Dataset Preparation: the process of loading a dataset and performing basic pre-processing. The dataset is then splited into train and validation sets.

```
# load the dataset
data = open('data/corpus').read()
labels, texts = [], []
for i, line in enumerate(data.split("\n")):
    content = line.split()
    labels.append(content[0])
    texts.append(" ".join(content[1:]))

# create a dataframe using texts and lables
trainDF = pandas.DataFrame()
trainDF['text'] = texts
trainDF['label'] = labels
```


https://www.analyticsvidhya.com/blog/2018/04/a-comprehensive-guide-to-understand-and-implement-text-classification-in-python/

Text Classification

Feature Engineering: the raw dataset is transformed into flat features which can be used in a machine learning model. This step also includes the process of creating new features from the existing data.

2.1 Count Vectors as features
2.2 TF-IDF Vectors as features

- Word level
- N-Gram level
- Character level

2.3 Word Embeddings as features
2.4 Text / NLP based features
2.5 Topic Models as features

https://www.analyticsvidhya.com/blog/2018/04/a-comprehensive-guide-to-understand-and-implement-text-classification-in-python/

Text Classification

Count Vectors as features: Count Vector is a matrix notation of the dataset in which every row represents a document from the corpus, every column represents a term from the corpus, and every cell represents the frequency count of a particular term in a particular document.

```
# create a count vectorizer object
count_vect = CountVectorizer(analyzer='word', token_pattern=r'\w{1,}')
count_vect.fit(trainDF['text'])

# transform the training and validation data using count vectorizer object
xtrain_count =  count_vect.transform(train_x)
xvalid_count =  count_vect.transform(valid_x)
```


https://www.analyticsvidhya.com/blog/2018/04/a-comprehensive-guide-to-understand-and-implement-text-classification-in-python/

Text Classification

TF-IDF Vectors as features: TF-IDF score represents the relative importance of a term in the document and the entire corpus. TF-IDF score is composed by two terms: the first computes the normalized Term Frequency (TF), the second term is the Inverse Document Frequency (IDF), computed as the logarithm of the number of the documents in the corpus divided by the number of documents where the specific term appears.

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

$$\text{IDF}(w) = \ln \frac{\text{Total number of documents}}{\text{Number of documents with term } w \text{ in it}}$$

https://www.analyticsvidhya.com/blog/2018/04/a-comprehensive-guide-to-understand-and-implement-text-classification-in-python/

Text Classification

TF-IDF Vectors can be generated at different levels of input tokens (words, characters, n-grams)

Word Level TF-IDF : Matrix representing tf-idf scores of every term/word in different documents

```
# word level tf-idf
tfidf_vect = TfidfVectorizer(analyzer='word', token_pattern=r'\w{1,}', max_features=5000)
tfidf_vect.fit(trainDF['text'])
xtrain_tfidf =  tfidf_vect.transform(train_x)
xvalid_tfidf =  tfidf_vect.transform(valid_x)
```


Text Classification

TF-IDF Vectors can be generated at different levels of input tokens (words, characters, n-grams)

N-gram Level TF-IDF : N-grams are the combination of N terms together. Matrix is tf-idf scores of N-grams

```
# ngram level tf-idf
tfidf_vect_ngram = TfidfVectorizer(analyzer='word', token_pattern=r'\w{1,}', ngram_range=(2,3), max_features=5000)
tfidf_vect_ngram.fit(trainDF['text'])
xtrain_tfidf_ngram =  tfidf_vect_ngram.transform(train_x)
xvalid_tfidf_ngram =  tfidf_vect_ngram.transform(valid_x)
```

Text Classification

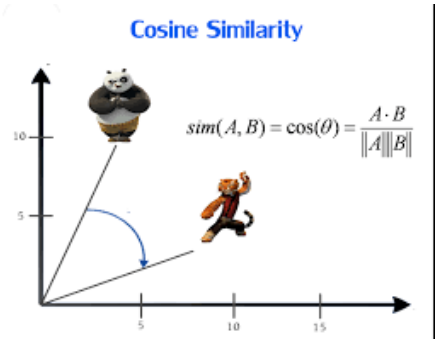
TF-IDF Vectors can be generated at different levels of input tokens (words, characters, n-grams)

Character Level TF-IDF: Matrix representing tf-idf scores of character level n-grams in the corpus

```
# characters level tf-idf
tfidf_vect_ngram_chars = TfidfVectorizer(analyzer='char', token_pattern=r'\w{1,}', ngram_range=(2,3), max_features=5000)
tfidf_vect_ngram_chars.fit(trainDF['text'])
xtrain_tfidf_ngram_chars =  tfidf_vect_ngram_chars.transform(train_x)
xvalid_tfidf_ngram_chars =  tfidf_vect_ngram_chars.transform(valid_x)
```


Text Classification

Word Embeddings: A word embedding is a form of representing words and documents using a dense vector representation. The position of a word within the vector space is learned from text and is based on the words that surround the word when it is used.



Text Classification

Text / NLP based features:

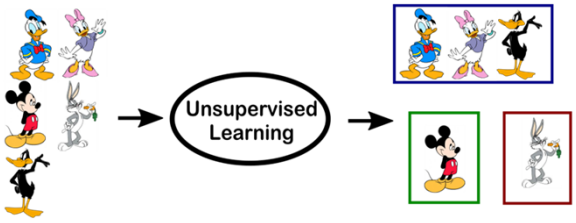
1) Word count of the documents – total number of words in the documents
2) Character count of the documents – total number of characters in the documents
3) Average word density of the documents – average length of the words used in the documents
4) Punctuation count in the complete essay – total number of punctuation marks in the documents
5) Upper case count in the complete essay – total number of upper count words in the documents
6) Title word count in the complete essay – total number of proper case (title) words in the documents
7) Frequency distribution of part of speech tags: counts of Noun, Verb, Adjective, Adverb, Pronoun

https://www.analyticsvidhya.com/blog/2018/04/a-comprehensive-guide-to-understand-and-implement-text-classification-in-python/

```
trainDF['char_count'] = trainDF['text'].apply(len)
trainDF['word_count'] = trainDF['text'].apply(lambda x: len(x.split()))
trainDF['word_density'] = trainDF['char_count'] / (trainDF['word_count']+1)
trainDF['punctuation_count'] = trainDF['text'].apply(lambda x: len("".join(_ for _ in x if _ in string.punctuation)))
trainDF['title_word_count'] = trainDF['text'].apply(lambda x: len([wrd for wrd in x.split() if wrd.istitle()]))
trainDF['upper_case_word_count'] = trainDF['text'].apply(lambda x: len([wrd for wrd in x.split() if wrd.isupper()]))
```

Text Classification

Topic Models as features:



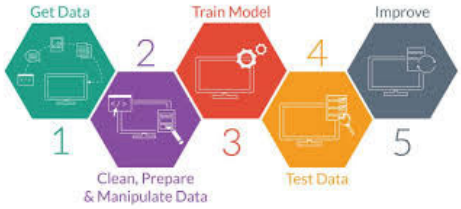
https://www.analyticsvidhya.com/blog/2018/04/a-comprehensive-guide-to-understand-and-implement-text-classification-in-python/

Text Classification

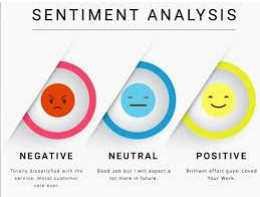
Model Building

The final step in the text classification framework is to train a classifier using the features created in the previous step. There are many different choices of machine learning models which can be used to train a final model. A few examples include:

1) Naive Bayes Classifier
2) Linear Classifier
3) Support Vector Machine
4) Bagging Models
5) Boosting Models
6) Shallow Neural Networks
7) Deep Neural Networks



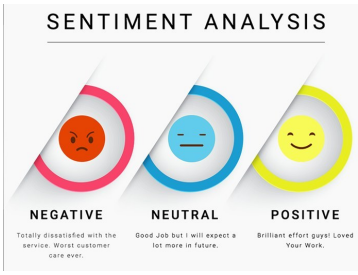
Sentiment Analysis



Classification Example: Sentiment Analysis

The process of computationally identifying and categorizing **opinions** expressed **in** a piece of **text**, especially in order to determine whether the writer's **attitude** towards a particular topic, product, etc. is positive, negative, or neutral.

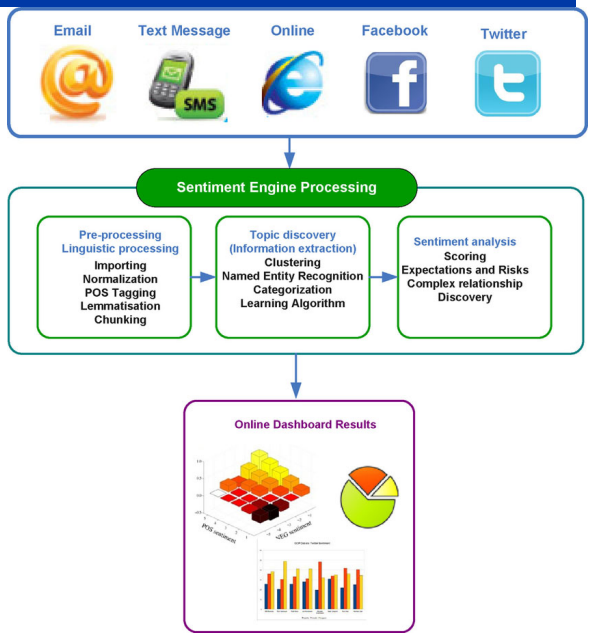
"companies have key lessons to learn about harnessing the power of social media and sentiment analysis"



Sentiment Analysis

Sentiment analysis has many other names:

- Opinion extraction
- Opinion mining
- Sentiment mining
- Subjectivity analysis



Sentiment Analysis

Why sentiment analysis?

• *Movie*: is this review positive or negative?

• *Products*: what do people think about the new iPhone?

• *Public sentiment*: how is consumer confidence? Is despair increasing?

• *Politics*: what do people think about this candidate or issue?

• *Prediction*: predict election outcomes or market trends from sentiment; predict stock return/price together with twitter or other types of text data

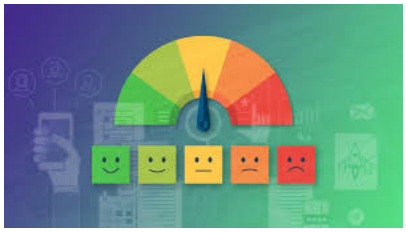
Sentiment Analysis

https://www.northeastern.edu/levelblog/2018/08/02/companies-use-sentiment-analysis/

https://medium.com/analytics-vidhya/simplifying-social-media-sentiment-analysis-using-vader-in-python-f9e6ec6fc52f

https://medium.com/@datamonsters/sentiment-analysis-tools-overview-part-2-7f3a75c262a3

https://www.kdnuggets.com/2018/08/emotion-sentiment-analysis-practitioners-guide-nlp-5.html



Sentiment Analysis

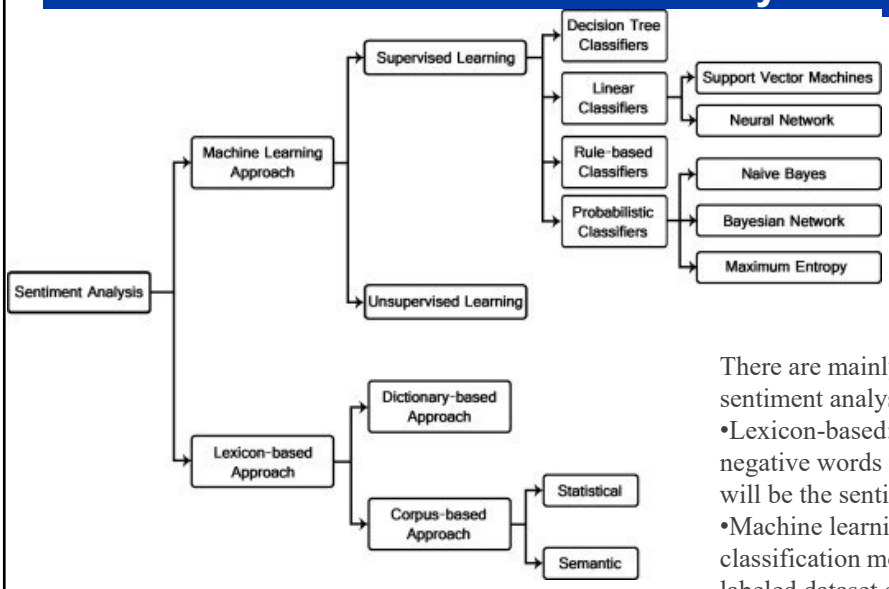
Why is Sentiment Analysis Hard?

Sentiment Analysis is actually a tricky subject. There are various reasons for that:

- Understanding emotions through text are not always easy. For instance,
  - ✓ "The best I can say about the movie is that it was interesting."
- A text may contain multiple sentiments all at once. For instance,
  - ✓ "The intent behind the movie was great, but it could have been better".
- Heavy use of emoticons and slangs with sentiment values in social media texts like that of Twitter and Facebook also makes text analysis difficult. For example a " :)" denotes a smiley and generally refers to positive sentiment while ":(" denotes a negative sentiment on the other hand.

https://medium.com/analytics-vidhya/simplifying-social-media-sentiment-analysis-using-vader-in-python-f9e6ec6fc52f

Sentiment Analysis



There are mainly two approaches for performing sentiment analysis.

- Lexicon-based: count number of positive and negative words in given text and the larger count will be the sentiment of text.
- Machine learning based approach: Develop a classification model, which is trained using the pre-labeled dataset of positive, negative, and neutral

https://www.kdnuggets.com/2018/03/5-things-sentiment-analysis-classification.html

Sentiment Analysis

Lexicon Based: TextBlob Sentiment

The TextBlob package for Python is a convenient way to do a lot of Natural Language Processing (NLP) tasks.

```
from textblob import TextBlob

TextBlob("not a very great calculation").sentiment
```

Output is:

Sentiment(polarity=-0.3076923076923077, subjectivity=0.5769230769230769)

Question: ***where do these numbers come from?***

https://planspace.org/20150607-textblob_sentiment/

Sentiment Analysis

Lexicon Based: TextBlob Sentiment

Each word in the lexicon has scores for:

1) polarity: negative vs. positive (-1.0 => +1.0)

2) subjectivity: objective vs. subjective (+0.0 => +1.0)

3) intensity: modifies next word? (x0.5 => x2.0)

Polarity in sentiment analysis refers to identifying *sentiment orientation* (positive, neutral, and negative) in written or spoken language

A sentence could be stating a fact(*objective*) or expressing an opinion(*subjective*).

Sentiment Analysis

Lexicon Based: TextBlob Sentiment

Negation multiplies the polarity by -0.5, and doesn't affect subjectivity.

Add NOT_ to every word between negation and following punctuation:

```
didn't like this movie , but I
```

```
didn't NOT_like NOT_this NOT_movie but I
```

Sentiment Analysis

Lexicon Based: VADER

- VADER (Valence Aware Dictionary and sEntiment Reasoner) is a lexicon and rule-based sentiment analysis tool that is specifically attuned to sentiments expressed in social media..
- VADER has been found to be quite successful when dealing with social media texts, NY Times editorials, movie reviews, and product reviews.
- VADER not only tells about the Positivity and Negativity score but also tells us about how positive or negative a sentiment is.
- It is fully open-sourced under the MIT License. The developers of VADER have used Amazon's Mechanical Turk to get most of their ratings,

https://medium.com/analytics-vidhya/simplifying-social-media-sentiment-analysis-using-vader-in-python-f9e6ec6fc52f

Sentiment Analysis

Lexicon Based: VADER

- pip install vaderSentiment
- from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer

  analyser = SentimentIntensityAnalyzer()
- def sentiment_analyzer_scores(sentence):

    score = analyser.polarity_scores(sentence)

    print("{:-<40} {}".format(sentence, str(score)))

https://medium.com/analytics-vidhya/simplifying-social-media-sentiment-analysis-using-vader-in-python-f9e6ec6fc52f

Sentiment Analysis

Lexicon Based: VADER

- sentiment_analyzer_scores("The phone is super cool.")

The phone is super cool---------- {'neg': 0.0, 'neu': 0.326, 'pos': 0.674, 'compound': 0.7351}

Sentiment Metric	Score
Positive	0.674
Neutral	0.326
Negative	0.0
Compound	0.735

- The Positive, Negative and Neutral scores represent the proportion of text that falls in these categories. This means our sentence was rated as 67% Positive, 33% Neutral and 0% Negative. Hence all these should add up to 1.
- The compound score is computed by summing the valence scores of each word in the lexicon, adjusted according to the rules, and then normalized to be between -1 (most extreme negative) and +1 (most extreme positive).
- Typical threshold values (used in the literature cited on this page) are:
  - positive sentiment: compound score >= 0.05
  - neutral sentiment: (compound score is (-0.05, 0.05)
  - negative sentiment: compound score <= -0.05

https://medium.com/analytics-vidhya/simplifying-social-media-sentiment-analysis-using-vader-in-python-f9e6ec6fc52f

Sentiment Analysis

Sentiment Analysis (Opinion Mining) lexicons

- MPQA Subjectivity Lexicon
- Bing Liu and Minqing Hu Sentiment Lexicon
- SentiWordNet (Included in NLTK)
- VADER Sentiment Lexicon
- SenticNet
- LIWC (not free)
- Harvard Inquirer
- ANEW
- Loughran-McDonald Master Dictionary

Sentiment Symposium Tutorial by Christopher Potts

Sentiment Analysis

When there is a training data set, ….

https://www.datacamp.com/community/tutorials/simplifying-sentiment-analysis-python

Sentiment Analysis

Pre-processing

An initial step in text and sentiment classification is pre-processing. A significant amount of techniques is applied to data in order to reduce the noise of text, reduce dimensionality, and assist in the improvement of classification effectiveness. These include:

- Remove numbers
- Stemming
- Part of speech tagging
- Remove punctuation
- Lowercase
- Remove stopwords: https://gist.github.com/sebleier/554280

Sentiment Analysis

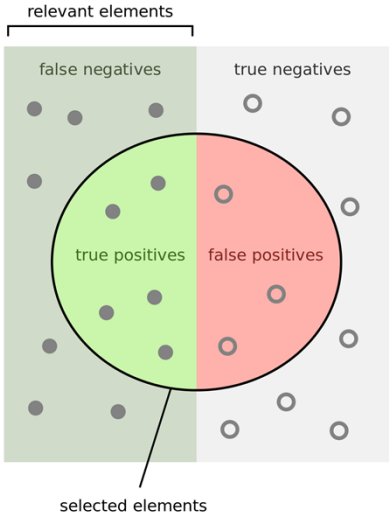
Some evaluations and labeled sentiment datasets are:

- Stanford Twitter Sentiment
- Sentiment Strength Twitter Dataset
- Amazon Reviews for Sentiment Analysis
- Large Movie Review Dataset
- Sanders Corpus
- SemEval (Semantic Evaluation) dataset

Sentiment Analysis

Evaluation metrics

Sentiment Analysis uses the evaluation metrics of Precision, Recall, F-score, and Accuracy.



How many selected items are relevant?

$$\text{Precision} = \frac{\text{true positives}}{\text{true positives} + \text{false positives}}$$

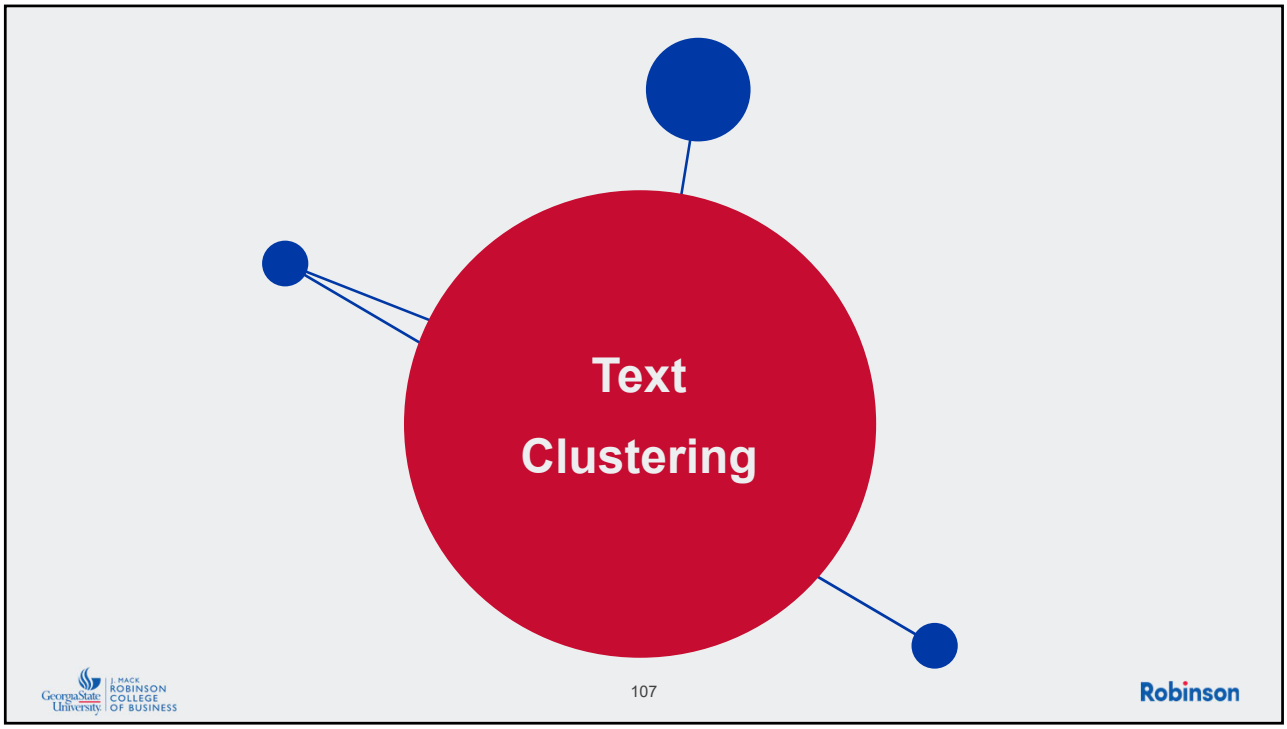
How many relevant items are selected?

$$\text{Recall} = \frac{\text{true positives}}{\text{true positives} + \text{false negatives}}$$

$$\text{Accuracy} = \frac{\text{Number of correct prediction}}{\text{Total number of predictions}}$$

The F score is the Harmonic mean of precision and recall:

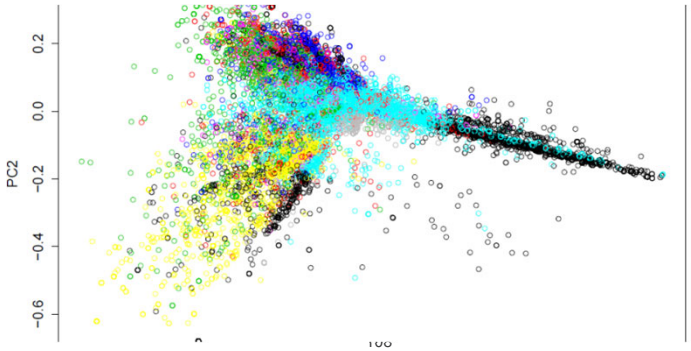
$$F = \frac{2}{\frac{1}{\text{Recall}} + \frac{1}{\text{Precision}}} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$



Text Clustering

Text clustering is the application of cluster analysis to text-based documents.

- the task of grouping a set of texts in such a way that texts in the same group (called a cluster) are more similar to each other than to those in other clusters.
- uses machine learning and natural language processing (NLP) to understand and categorize unstructured, textual data.



https://www.kdnuggets.com/2017/06/text-clustering-unstructured-data.html

Text Clustering

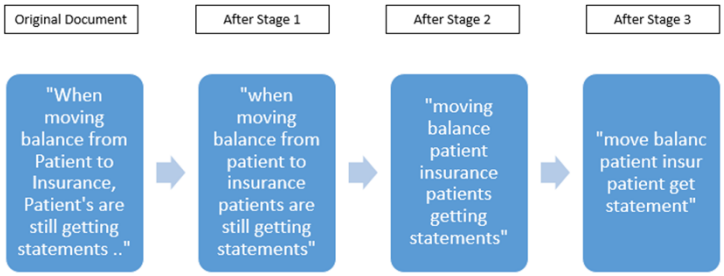
Text clustering algorithm:

1) Transformations on raw stream of free flow text

   Stage 1: removing punctuations, transforming to lower case, grammatically tagging sentences and removing pre-identified stop phrases, removing numbers from the document, stripping any excess white spaces

   Stage 2: Removing generic words of the English language viz. determiners, articles, conjunctions and other parts of speech.

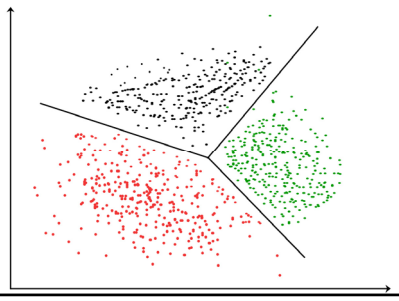
   Stage 3: Document Stemming which reduces each word to its root using Porter's stemming algorithm.



Text Clustering

Text clustering algorithm:

1) …
2) Creation of Term Document Matrix
3) TF-IDF (Term Frequency – Inverse Document Frequency) Normalization
4) K-Means Clustering using Euclidean Distances: computes the Euclidean Distances amongst these documents and clusters nearby documents together.
5) Auto-Tagging based on Cluster Centers

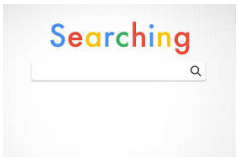


Text Clustering

Text Clustering Applications:

Media monitoring and analysis (social and traditional)

Detection of duplicate content, identification of plagiarism, related news.



Information retrieval and recommendation systems

Grouping of search results, aid to navigation, suggestion of related information, recommendation of contents and products.

Feedback analysis and opinion mining

Detection of not predefined and unforeseen subjects in surveys and claims (which enable a more proactive management and a more effective response); analysis of the voice of the customer, employee, citizen, etc.; idea management.

Document organization

Structuring of collections of documents and records according to the implicit subjects that naturally emerge from the contents themselves and not from external taxonomies.

https://www.meaningcloud.com/products/text-clustering

Assignment #2

Four components:

1) LDA model: best output (log-likelihood, coherence store, bubble visualization)
2) Dynamic topic model (topics change over time)
3) LDA2VEC (how does that compare to the LDA model, improvement methods such as using word2vec pre-trained data)
4) LSTM generate new texts: can refer http://karpathy.github.io/2015/05/21/rnn-effectiveness/

Deliverables:

1. Code: notebook and html,
2. Excel file: topic-key words distributions output,
3. Presentation slides