2695 Introduction to Machine Learning
Masters Program in Economics, Finance and Management

NOVA SCHOOL OF BUSINESS & ECONOMICS

9a Video Lecture

TEXT PREPROCESSING AND REPRESENTATION

Text is everywhere

- Internet contains vast amount of text:
  web pages, social media feeds, email, emails and chats, digitized libraries
- Many applications still produce or record text:
  customer support tickets, surveys and feedback forms, online product reviews

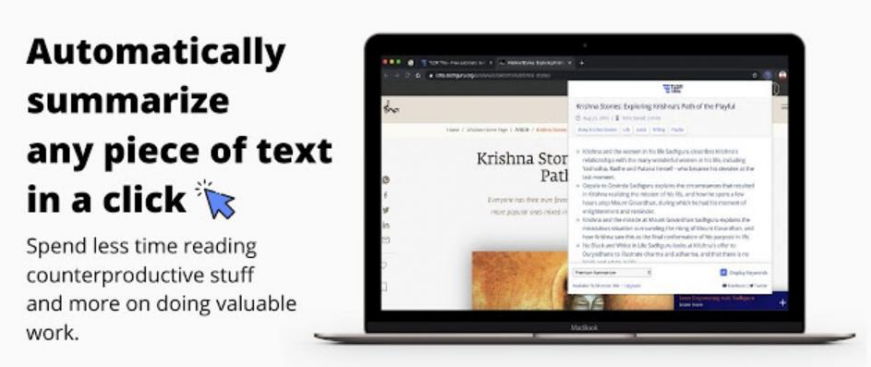
Text Preprocessing: transforms raw, unstructured text into a format that is suitable for automated analysis algorithm

Text mining: discovers patterns, insights, and knowledge from unstructured text data

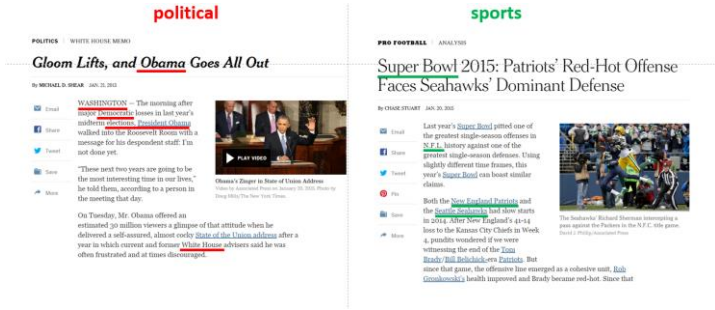
Natural Language Processing: focused on enabling computers to understand, interpret, and generate human language, both text and speech, in a meaningful way

Text mining tasks

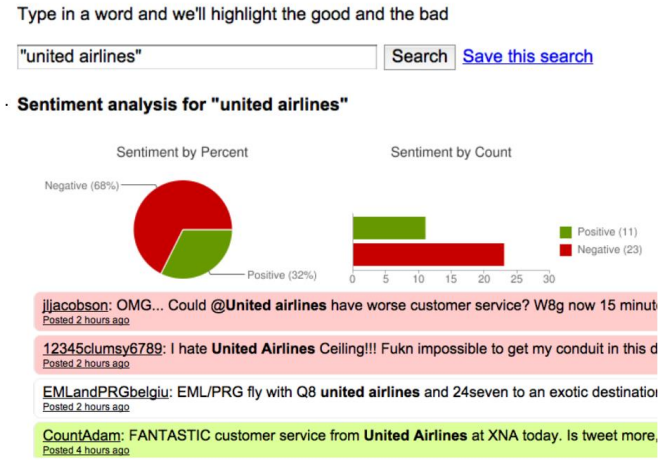
Text Summarization: Generating concise and coherent summaries



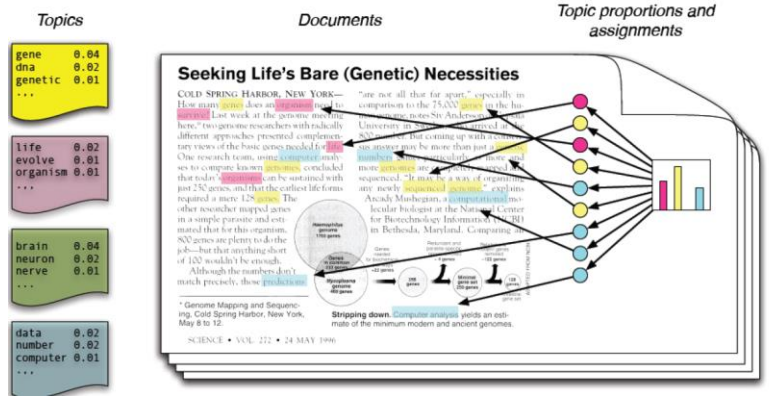
Document classification: labeling documents using pre-defined categories, depending on their content.



Sentiment Analysis: Determining the sentiment (positive, negative, neutral), opinions, expressed in text towards specific topics



Topic modeling: automatically extracting meaning from texts by identifying recurrent themes or topics



Why is text mining challenging?

- Unstructured data:
  - Documents have varying number of words, words can have varying lengths.
  - Sometimes word order matters, sometimes not.

- As data, text is relatively dirty:
  People write ungrammatically, they misspell words, they abbreviate and punctuate randomly.

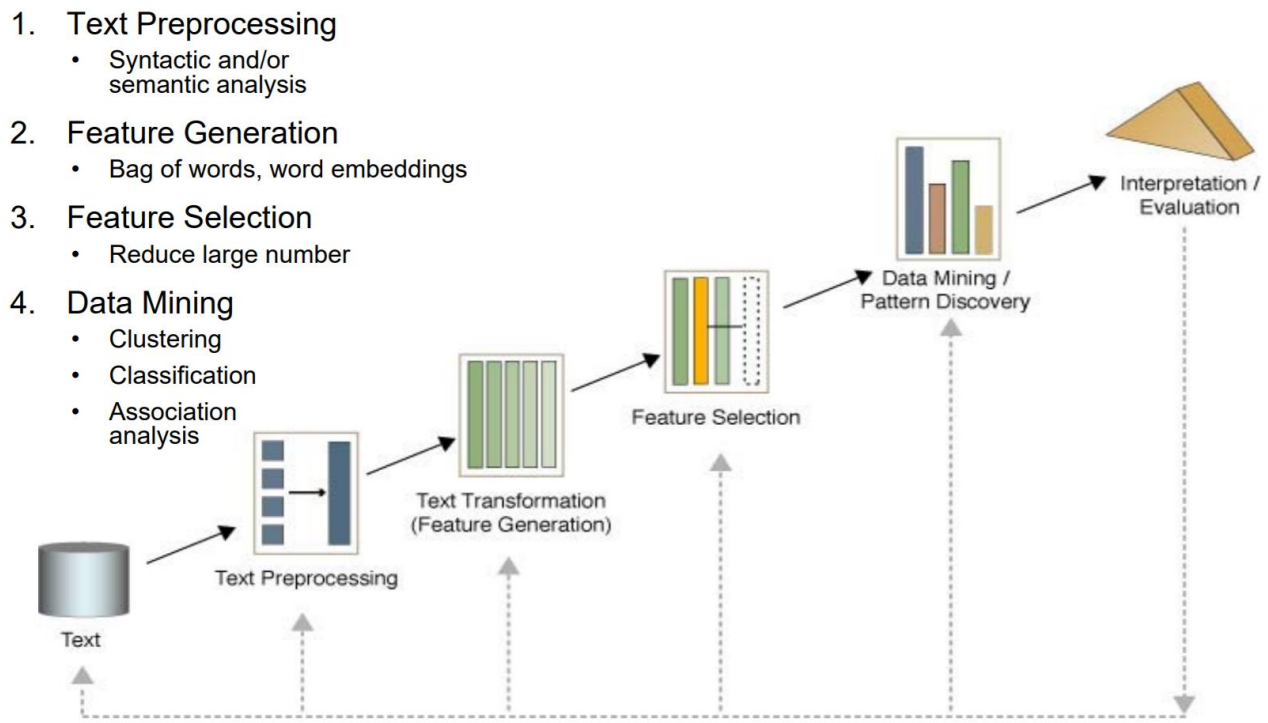
- Text may contain synonyms (multiple words with the same meaning) and homographs (one spelling shared among multiple words with different meanings).

- Terminology and abbreviations in one domain might be meaningless in another domain.

- Context is important

Text mining basic terminology and basic pipeline

- **Document***:* one piece of text, no matter how large or small, could be a single sentence or a 100-page report, or anything in between, such as a YouTube comment or a blog posting.
- **Tokens** or **terms:** for example, a word, sentence
- **Corpus**: collection of documents



Text preprocessing

Goal: to clean and transform raw, unstructured text data into a standardized, structured, and noise-free format. The exact steps and their order can vary based on the specific task and data:

- Lowercasing (case normalization)
- Cleaning the text & normalizing
- Tokenization
- Stop word removal
- Stemming or Lemmatization

Text preprocessing: Lowercasing

- **Lowercasing** all text: one of the simplest and most effective form of text preprocessing. It is applicable to most text mining problems and significantly helps with consistency of expected output.

- Case variations are so common that case normalization is usually necessary: example iPhone, iphone, and IPHONE.

- Lowercasing not always helpful: sentiment analysis, Information extraction: distinguishing US and us, company Apple from fruit apple

Text preprocessing: Cleaning the text & normalizing

- Removing HTML tags

- Converting accented characters to ASCII characters

"latté" and "café" can be converted and standardized to just "latte" and "cafe"

- Expanding contractions

don't and can't: expanding such words to "do not" and "can not"

- Standardizing different spelling & abbreviations

- Removing extra whitespaces

- Removing special characters and punctation (depending on the task)

(matching USA and U.S.A.)

- Converting number words to numeric form, removing numbers

Text preprocessing: Tokenization

- **Tokenization** is a step which splits longer strings of text into smaller pieces, or **tokens** (sub word, word, phrase…)
- How are sentences identified within larger bodies of text? Using "sentence-ending punctuation," is ambiguous.

> The quick brown fox jumps over the lazy dog.

But what about this one:

> Dr. Ford did not ask Col. Mustard the name of Mr. Smith's dog.

Or this one:

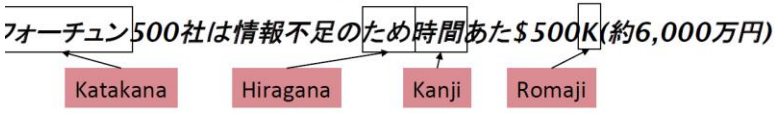
> "What is all the fuss about?" asked Mr. Peters.

And that's just sentences. What about words? Easy, right? *Right?*

> This full-time student isn't living in on-campus housing, and she's not wanting to visit Hawai'i.

- Challenges for different languages: linguistic structures vary significantly across the globe

- Chinese and Japanese no spaces between words:
  - 莎拉波娃现在居住在美国东南部的佛罗里达。
  - 莎拉波娃 现在 居住 在 美国 东南部 的 佛罗里达
  - Sharapova now lives in US southeastern Florida
- Further complicated in Japanese, with multiple alphabets intermingled
  - Dates/amounts in multiple formats



End-user can express query entirely in hiragana!

- German noun compounds are not segmented
  - ***Lebensversicherungsgesellschaftsangestellter***
  - 'life insurance company employee'
  - German information retrieval needs **compound splitter**

- LLMs typically use sub-word tokenization: break words down into frequently occurring smaller units based purely on the statistics of a large training corpus.

Tokenization: N-grams

- Whitespace/ unigram tokenization: text is split into words by splitting them from whitespaces. Word order is discarded.

- Include *sequences* of adjacent words as terms.
  - For example, we could include pairs of adjacent words so that if a document contained the sentence *"Quick brown fox jumps."* it would be transformed into
    - set of its constituent words {*quick, brown, fox, jumps*},
    - tokens *quick_brown, brown_fox, and fox_jumps*.

- Adjacent pairs are commonly called **bi-grams**.

- This general representation tactic is called ***n-grams***.

- N-grams: Easy to generate, and they require no linguistic knowledge or complex parsing algorithm but greatly increase the size of the feature set. (followed by feature selection)

- LLMs use different techniques to capture word order.

Text preprocessing: Stemming

- **Stemming:** reducing inflected words to their base or root form: **word stem**.

  Stems: The core meaning-bearing units (WALK)
  Affixes: Parts that adhere to stems, often with grammatical functions (-ED)

  The inflection *-ed* is often used to indicate the past tense, changing *walk* to *walked* and *listen* to *listened*.
- Stemming: process of eliminating affixes (suffixes, prefixes, infixes, circumfixes) from a word in order to obtain a word stem.
- Porter's algorithm: the most common English stemmer
- It is language dependent. In some languages, it is more useful than in others

  - e.g., ***automate(s), automatic, automation*** all reduced to ***automat***.

for example compressed and compression are both accepted as equivalent to compress.

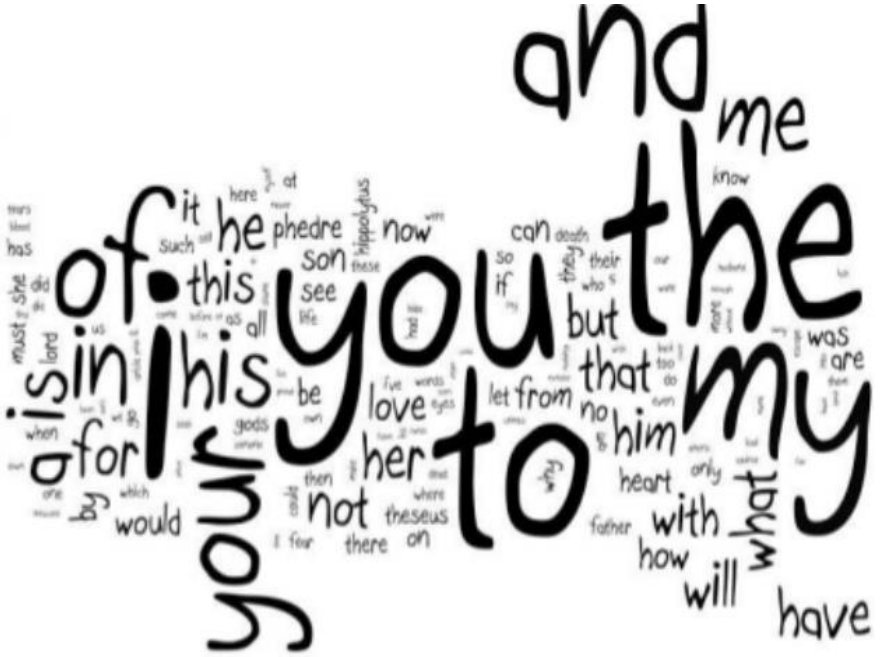
for exampl compress and compress ar both accept as equival to compress

Text preprocessing: Lemmatization

- **Lemmatization** is the process of converting a word to its base form (correct dictionary base form).
- The output of lemmatization is a root word called a **lemma**. For example:
  - *am, are, is* will be converted to *be*
  - *running, runs, ran* will be replaced by *run*
- Stemming and Lemmatization: generate the foundation of the inflected words with the difference being that **stem may not be an actual word** whereas, lemma is an actual language word.
- Stemming: algorithm with steps to perform on the words which makes it faster.
- Lemmatization, corpus to supply lemma, slower than stemming

Text preprocessing: Stopwords removal

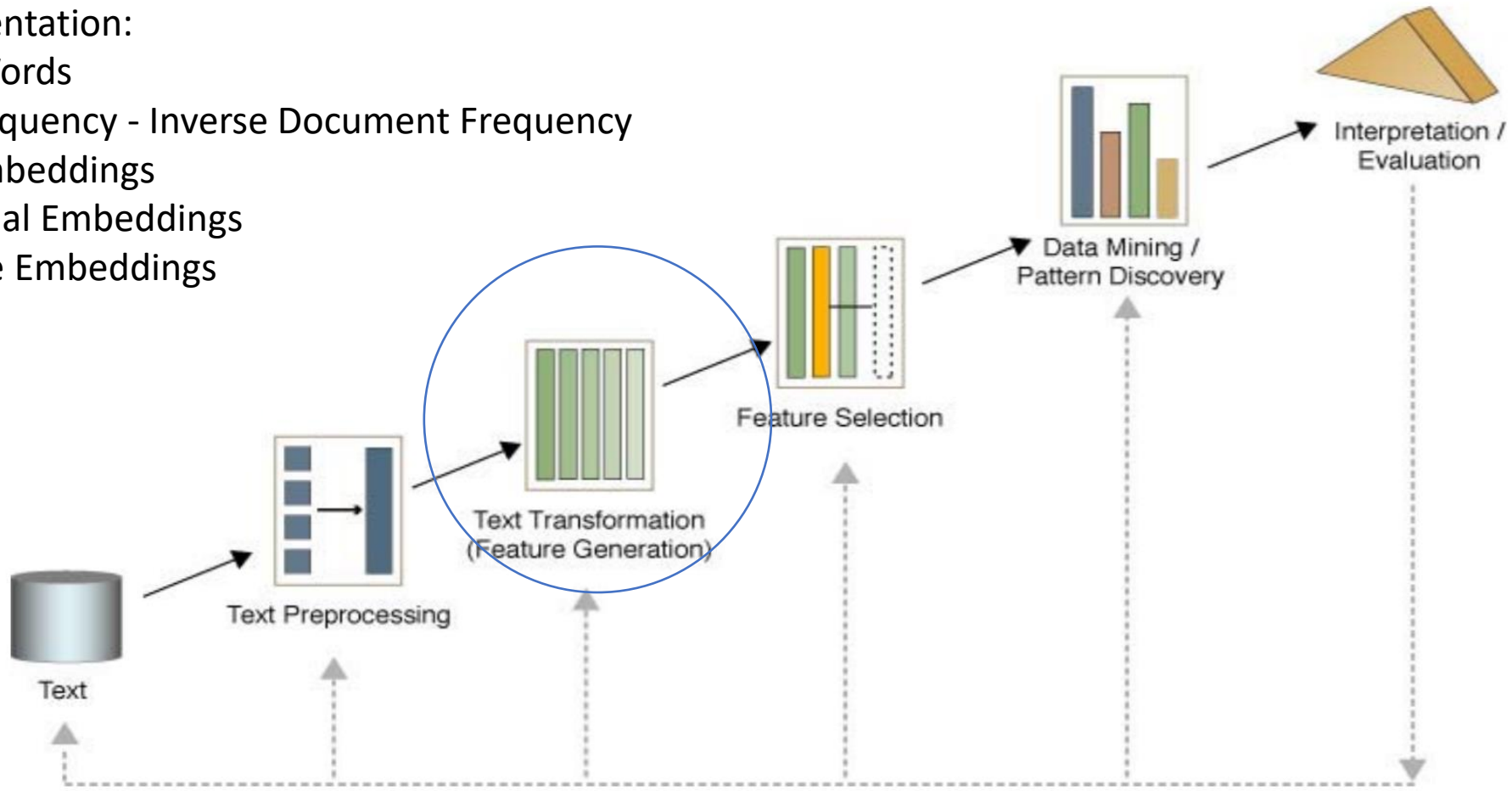
- Stopwords are very common words.
- Intuition:
  - They have little semantic content: words like *a*, *the*, *we* probably do not help in many text mining tasks.
  - There are a lot of them.
- Remove the stopwords to save computing time and efforts in processing large volumes of text.
- Stop word lists can come from pre-established sets, or we can create a custom one for our domain.
- But sometimes, we should not remove them:
  - Relational queries: *flights to London vs flights from London*
  - *Not* is a stopword, but very important for sentiment analysis.



Feature generation

Creating numerical features from cleaned and tokenized text.

Text representation:
- Bag of Words
- Term Frequency - Inverse Document Frequency
- Word Embeddings
- Contextual Embeddings
- Sentence Embeddings



Bag of words (BoW)

- Treats every document as just a collection of individual words.
- Ignores grammar, word order, sentence structure, and (typically) punctuation.
- Representation is straightforward and inexpensive to generate.
- Starting point for many tasks.
- Encodes each document as a vector based on some dictionary.

  Example: binary vector (presence / absence of each word) or discrete (word counts).

Document vectors: binary vectors

Example of three documents:

- Review 1: *This movie is very scary and long*
- Review 2: *This movie is not scary and is slow*
- Review 3: *This movie is spooky and good*

Vocabulary (11 unique words):

- this
- movie
- is
- very
- scary
- and
- long
- not
- slow
- spooky
- good

Binary vector: 1 denotes the presence of a word, 0 absence

	this	movie	is	very	scary	and	long	not	slow	spooky	good	Total words
Document 1	1	1	1	1	1	1	1	0	0	0	0	7
Document 2	1	1	1	0	1	1	0	1	1	0	0	8
Document 3	1	1	1	0	0	1	0	0	0	1	1	6

Number of features in the dataset= number of unique words (size of vocabulary)

Document feature vector: binary vector of word presence/absence for the document, based on the corpus vocabulary

Term frequency TF

- Instead of just representing presence or absence of a word, count words in the document (word frequency).
  In some applications, the importance of a term in a document should increase with the number of times that term occurs.

Term count: each entry is a count how many times a word occurs in a document

	this	movie	is	very	scary	and	long	not	slow	spooky	good	Total words
Document 1	1	1	1	1	1	1	1	0	0	0	0	7
Document 2	1	1	**2**	0	1	1	0	1	1	0	0	8
Document 3	1	1	1	0	0	1	0	0	0	1	1	6

$$tf_{t,d} = \frac{n_{t,d}}{number\ of\ terms\ in\ a\ document}$$

$n_{t,d}$ number of times a term t is present in a document d

Term frequency: each entry is a frequency of a word occurs in a document

	this	movie	is	very	scary	and	long	not	slow	spooky	good	Total words
Document 1	1/7	1/7	1/7	1/7	1/7	1/7	1/7	0	0	0	0	7
Document 2	1/8	1/8	**2/8**	0	1/8	1/8	0	1/8	1/8	0	0	8
Document 3	1/6	1/6	1/6	1/6	0	1/6	0	0	0	1/6	1/6	6

Document feature vector: vector of word counts/frequencies for the document, based on the corpus vocabulary

Inverse Document Frequency IDF

- Term *frequency* measures how prevalent a term is in a single document. But how common it is in the entire corpus we're mining?
- A term should not be too *rare*.
- A term should not be too *common*.
- The fewer documents in which a term occurs, the more significant it is likely to be to the documents is does occur in.
- **Inverse document frequency (IDF)**:

$$\text{IDF}(t) = 1 + \log\left(\frac{\text{Total number of documents}}{\text{Number of documents containing } t}\right)$$

Note: There are variations for the IDF expression, often "1+" term is omitted, and IDF equals only the log, or 1+ appears within the log.

Inverse Document Frequency example

Term	Review 1	Review 2	Review 3
This	1	1	1
movie	1	1	1
is	1	2	1
very	1	0	0
scary	1	1	0
and	1	1	1
long	1	0	0
not	0	1	0
slow	0	1	0
spooky	0	0	1
good	0	0	1

- Review 1: *This movie is very scary and long*
- Review 2: *This movie is not scary and is slow*
- Review 3: *This movie is spooky and good*

- The words like *is*, *this*, *and*, etc., are reduced to 1 and have little importance.
- The words like *scary*, *long*, *good*, etc. are words with more importance and thus have a higher IDF value.

	this	movie	is	very	scary	and	long	not	slow	spooky	good
IDF	1	1	1	1.48	1.18	1	1.48	1.48	1.48	1.48	1.48

Note: here we use log base 10, though other bases are sometimes used.

Term Frequency Inverse Document Frequency TF-IDF

- We can combine them as:

$$\mathrm{TFIDF}(t, d) = \mathrm{TF}(t, d) \times \mathrm{IDF}(t)$$

- TF depends on the specific document, IDF depends on the entire corpus, TF-IDF value is specific to a single document.

- Document feature vector: vector of TF-IDF for the document. Same as in BOW, each feature corresponds to a unique term from the corpus vocabulary. These vectors can then be used in a data mining algorithm for classification, clustering, or retrieval.

NOVA SCHOOL OF BUSINESS & ECONOMICS

9b Video Lecture

TEXT CLASSIFICATION
SENTIMENT ANALYSIS
TOPIC MODELING

Text Classification: definition

Input:
- Document d
- A fixed set of classes C={C_1, C_2, …, C_N}
- Training set: documents with class labels

Output:
- Trained classifier that maps a document to a class

Text Representation:
- BoW
- TF-IDF

Classifier:
- Logistic Regression
- Naïve Bayes
- Neural Networks
- …

Bayes rule for classification

Fruit features:

- color =red
- shape =round
- size =10cm

- p (C = apple) is the „prior“ probability of the class:

probability of a fruit being apple before even knowing its features, depends how common is this type of fruit

- p ($E|C = c$) is the likelihood of seeing the evidence E given a class

Likelihood seeing these feature values in an apple

- p (E) is the likelihood of the evidence

Likelihood seeing these feature values in any fruit

- p ($C = c|E$) as an estimate of class probability

Probability of a fruit being apple if it is red, round and 10 cm in size

$$p(C = c|E) = \frac{p(E|C = c)p(C = c)}{p(E)}$$

Conditional Independence

- $p\,(E|C = c)$= $p(e_1\ e_2\ \cdots e_k|c)$.

P(color =*red*, shape =*round*, size=*10* | class=*apple*)

- ***E*** feature vector: [e_1, e_2, …, e_k], where each e_i is a feature, and *k* is large.
- We may never see a specific example in the training data that exactly matches a given ***E*** in our testing data.
- Simplifying assumption: conditional independence - assume that the features are conditionally independent, given the class.
- In $p(e_1\ e_2 \cdots e_k|c)$, each e_i is independent of every other e_j given the class *c*.

P(color =*red*, shape =*round*, size=*10* | *apple)* = P(color =*red*| *apple)* P(shape =*round* | *apple)* P(size=*10* | *apple)*

$$p(E|c) = p(e_1|c)p(e_2|c)\ldots p(e_k|c)$$

Naïve Bayes

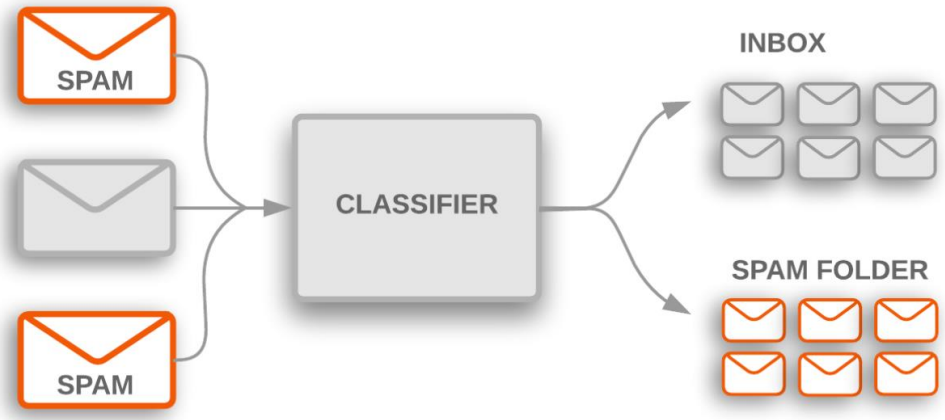
- Choose the class with the highest probability given feature values $p\ (C = c|E)$:

$$p\ (C = c|E) = \frac{p(e_1|c = class)\ldots p(e_k|c = class)\ p(c=class)}{p(e_1 \ldots e_k)}$$

Denominator *p(E)* does not depend on the class

Select class that gives the highest value: $p(e_1|c = class)\ \ldots p(e_k|c = class)\ p(c = class)$

Naïve Bayes for Text Classification



- **Bag of Words**: Assume word order does not matter, features are words, vectors of size k (number of unique words):
  - Binary vector
  - TF vector

- $p(e_k|class = c\ )$

Probability that a particular word occurs in a document of class c:

- $p(e_k|class = c\ ) = \frac{n_{kc}}{n_c}$

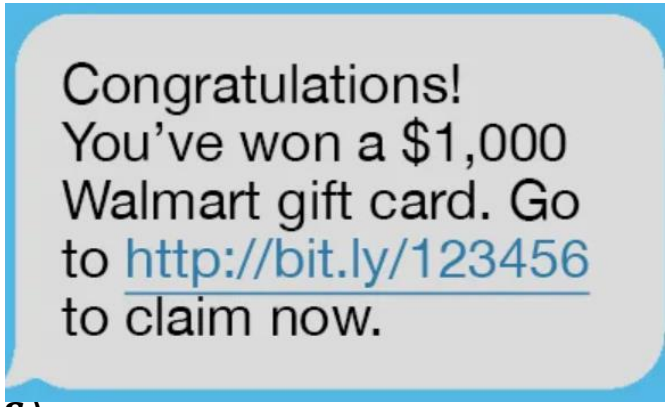
number of documents in the training dataset that contain this word and belong to this class c

number of documents in the training dataset that belong to this class c

- $p(e_k|class = c\ ) = \frac{n_{kc}+1}{n_c+2}$

Image from: https://developers.google.com/machine-learning/guides/text-classification

Example: Using Naïve Bayes for Text Classification



- We have two classes={*SPAM, NOT SPAM*}

Choose class c that gives the highest value: $\boldsymbol{p(e_1|c = class)} \ldots \boldsymbol{p(e_k|c = class)}\ \boldsymbol{p(c = class)}$

- For each class we need to learn the probabilities *p(word$_i$ | class)* and *p(class)*

- *p(spam)=number of spam emails in the training set/number of all emails in the training set*
- *p(not spam)=number of not spam emails in the training set/number of all emails in the training set*

- *p(word=congratulations| spam):* the fraction of emails in the training set of class spam with the word *congratulations*
- *p(word=congratulations| not spam):* the fraction of emails in the training set of class not spam with the word *congratulations*

- *P(spam|E)* ~ *p(word=congratulations|spam) * p(word=you|spam) * p(word=won|spam) * … * p(spam)= 0.5*
- *P(not spam| E)* ~ *p(word=congratulations|not spam) * p(word=you|not spam) * p(word=won|not spam) * … * p(not spam)= 0.02*

Sentiment Analysis

- Business applications
  - Social Media Monitoring
  - Customer Service
  - Market Research

- Tasks:
  - Simplest
    Is the attitude of this text positive or negative? (polarity detection)
  - More complex
    Rank the attitude of this text 1 to 5.
  - Advanced
    Detect the source (holder), aspect (target), or the actual emotion
    The food was great, but the service was awful

Sentiment Analysis challenges

- Context

  What did you like about the event?/ What did you DISlike about the event?

  - *Absolutely nothing!*
- Irony and sarcasm
  - *Did you enjoy your shopping experience with us?*

    Yeah, sure. So smooth!
- “Correct” labels for training data

Sentiment Lexicons

- A **sentiment lexicon** is a dictionary where words annotated with their sentiment orientation: positive or negative.
  - Positive sentiment words: *wonderful, beautiful* and *amazing*.
  - Negative sentiment words: *awful, poor* and *bad*.
- The overall sentiment of the text can be calculated by aggregating the scores of the words present, potentially considering intensifiers ("very") or negations ("not").
- Use lexicons in Naïve Bayes by adding features:
  - *'this word occurs in the positive lexicon'*
  - *'this word occurs in the negative lexicon'*
- Can be domain specific and customized for a certain task. Sentiment of a word may change over time.

Traditional ML for Sentiment Analysis

- Preprocess the data
- Extract features
  - Which words to use, for example only adjectives or all the words?
  - Word occurrence matters more than its frequency
  - Important to handle negation: *I liked the movie* vs *I didn't like the movie*

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

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

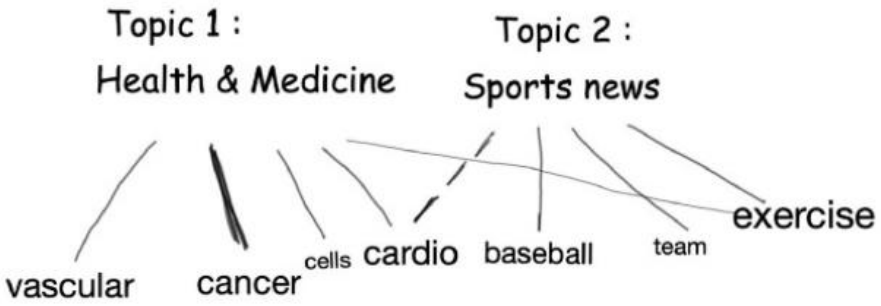
- Apply a classifier (positive, negative) or regressor (level of attitude)
  - Naïve Bayes as a baseline
  - More advanced approaches (neural networks)

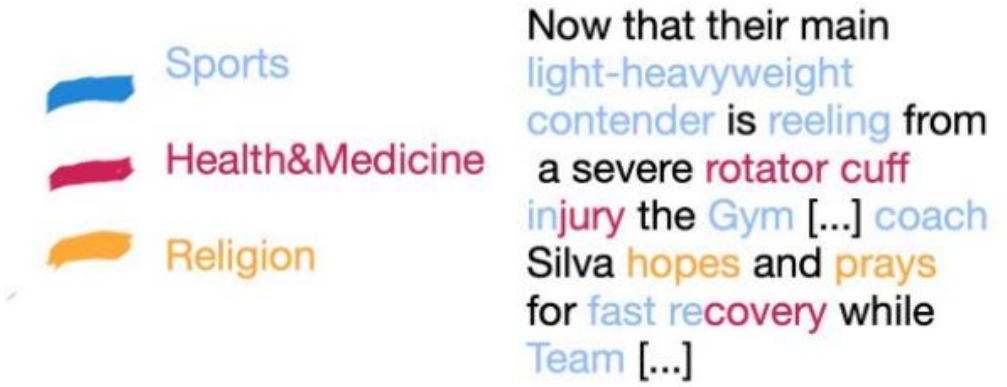
Topic Modeling

- Collection of documents (corpus): identify topics.
- A topic consists of a cluster of words that frequently occur together.
- We do not want to do supervised topic classification.
- Approach which automatically discovers the topics.
  - clustering problem –both words and documents being clustered.
- There are many techniques that are used to obtain topic models.
  - **Latent Dirichlet Allocation (LDA)**

Latent Dirichlet Allocation (LDA) key assumptions

- Key assumptions:
  - words that appear together frequently are likely to be close in meaning
  - each topic is a mixture of different words
  - each document is a mixture of different topics (but typically not many, and typically one is dominant)
  - number of topics is known in advance





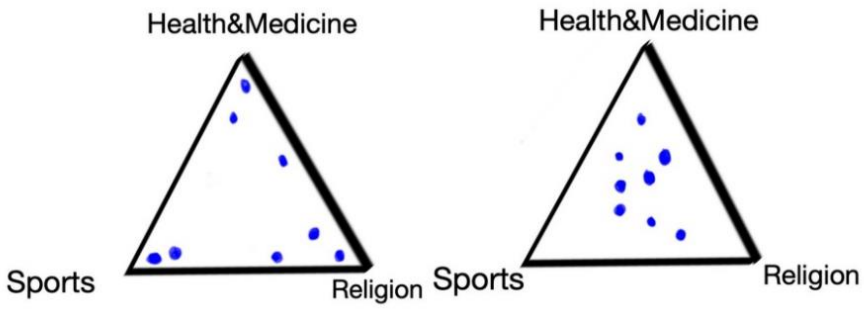
A single document can contain multiple topics (as color-coded on the left).

https://towardsdatascience.com/latent-dirichlet-allocation-intuition-math-implementation-and-visualisation-63ccb616e094

Latent Dirichlet Allocation (LDA) hyperparameters

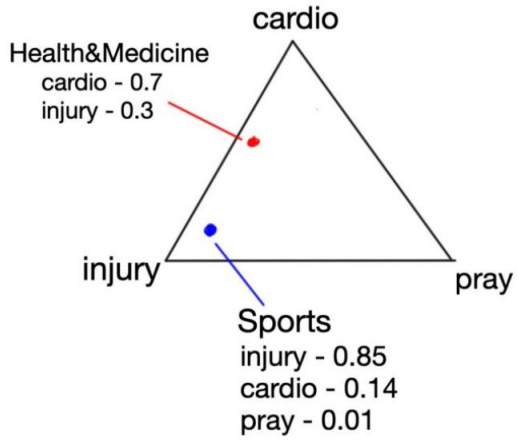
- **Dirichlet prior of the topics**:

represents how *sparse* or how *mixed* up the topics are



- **Dirichlet prior of the terms**:

represents how the **topics** are distributed amongst the **terms**.

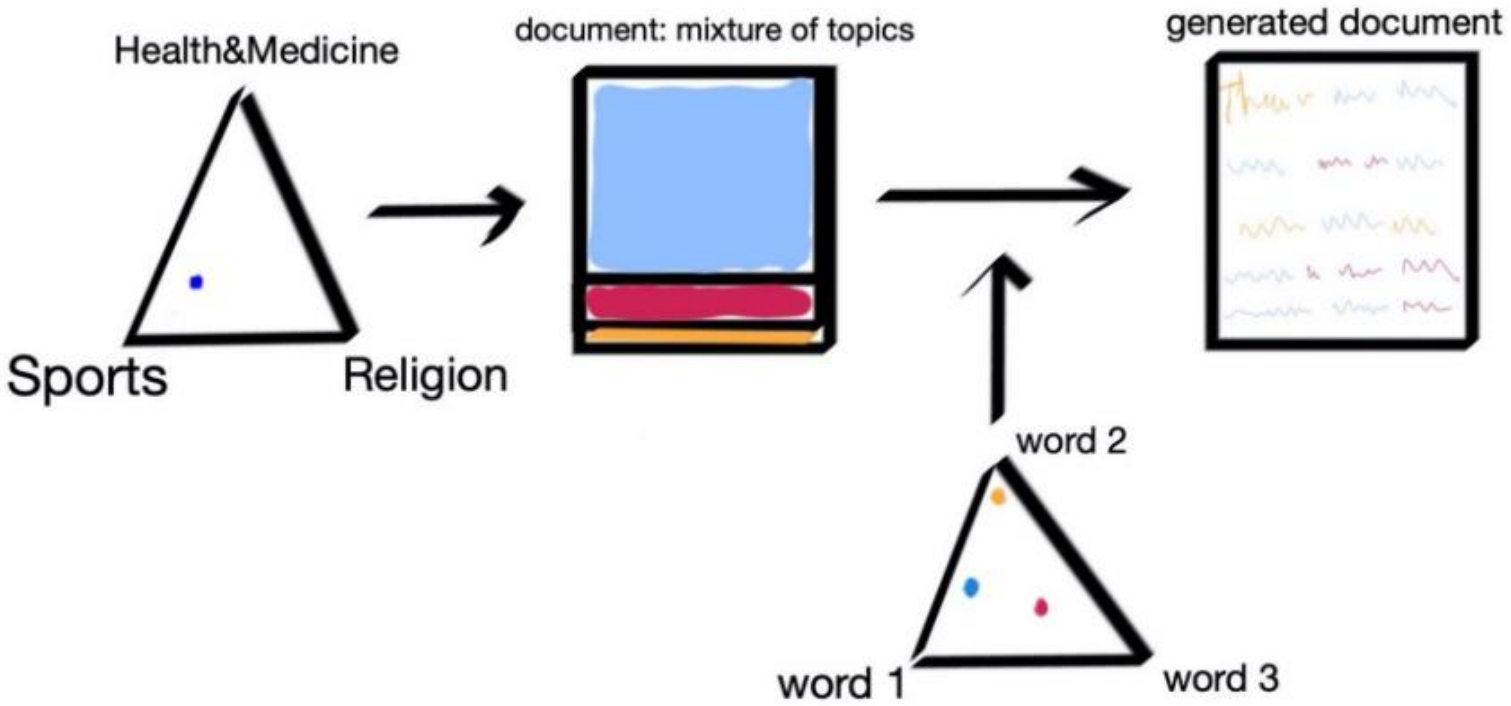


https://towardsdatascience.com/latent-dirichlet-allocation-intuition-math-implementation-and-visualisation-63ccb616e094

Latent Dirichlet Allocation (LDA): intuition

Assume:

- We have a fixed number of topics.
- Each document is a probability distribution over these k topics.
- Each topic is a probability distribution over all the different terms.



https://towardsdatascience.com/latent-dirichlet-allocation-intuition-math-implementation-and-visualisation-63ccb616e094

Choosing the number of topics and interpreting them

Choosing the number topics:

- Run the algorithm with different values of the number of topics and select the best according to some metric:
  - perplexity: measures how probable some new unseen data is given the model that was learned
  - coherence: measures the degree of similarity between high scoring words in the topic

Topic interpretability:

- examine the ranked list of the most probable terms in that topic
- analyze word relevance

9c Video Lecture

WORD EMBEDDINGS

Simple word representation

Vocabulary		one-hot encodings
dog	1	[1, 0, 0, 0, 0, 0, 0, 0, 0, 0]
cat	2	[0, 1, 0, 0, 0, 0, 0, 0, 0, 0]
person	3	[0, 0, 1, 0, 0, 0, 0, 0, 0, 0]
holding	4	[0, 0, 0, 1, 0, 0, 0, 0, 0, 0]
tree	5	[0, 0, 0, 0, 1, 0, 0, 0, 0, 0]
computer	6	[0, 0, 0, 0, 0, 1, 0, 0, 0, 0]
using	7	[0, 0, 0, 0, 0, 0, 1, 0, 0, 0]

- Feature vector can be large (size of the corpus vocabulary).
- Word order is lost.

Co-occurrence matrix

1. I enjoy flying.
2. I like NLP.
3. I like deep learning.

The resulting counts matrix will then be:

$$X = \begin{array}{c} \\ I \\ like \\ enjoy \\ deep \\ learning \\ NLP \\ flying \\ . \end{array} \begin{array}{c} \begin{array}{cccccccc} I & like & enjoy & deep & learning & NLP & flying & . \end{array} \\ \begin{bmatrix} 0 & 2 & 1 & 0 & 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \end{bmatrix} \end{array}$$

Example of three documents and the corresponding co-occurrence matrix for a window of size 1.

- Co-occurrence matrix: look at the window of fixed size of words (typically size 5 to 10) and count how many time words show up in the same window.
- Very high dimensional: requires a lot of storage.
- How to reduce dimensionality?

Word2Vec: Low dimensional word representation

- Neural networks used to directly learn low-dimensional word vectors.
- Goal: map a word as a vector in a lower-dimensional space, representation in this space often has a semantic meaning.
- **Word2Vec** (Google): Instead of capturing co-occurrence counts directly, predict surrounding words of every word in a window.
- Different flavors:
  - **Continuous Bag of Words** (CBOW): use context words in a window to predict the middle word.
  - **Skip-gram**: use the middle word to predict surrounding ones in a window.

Creating labels for CBoW

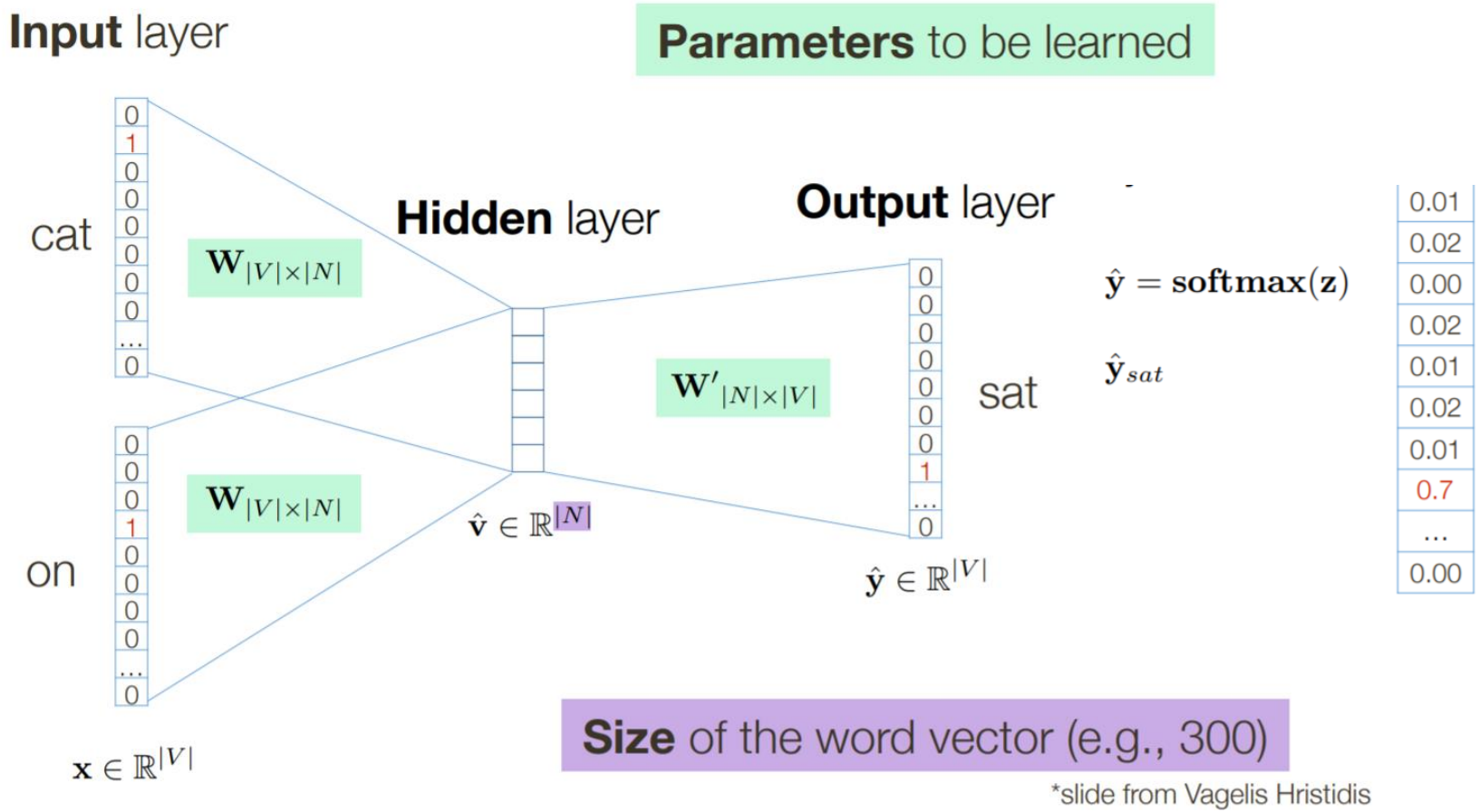
Continuous Bag of Words (CBOW): use context words in a window to predict middle word.

- Example of a window of size 1 word around the word of interest, in both directions.
- Use ‘I’ and ‘natural’ to predict ‘like’
- Use ‘like’ and ‘language’ to predict ‘natural’
- …

CBoW architecture

Continuous Bag of Words (CBOW): use context words in a window to predict middle word.

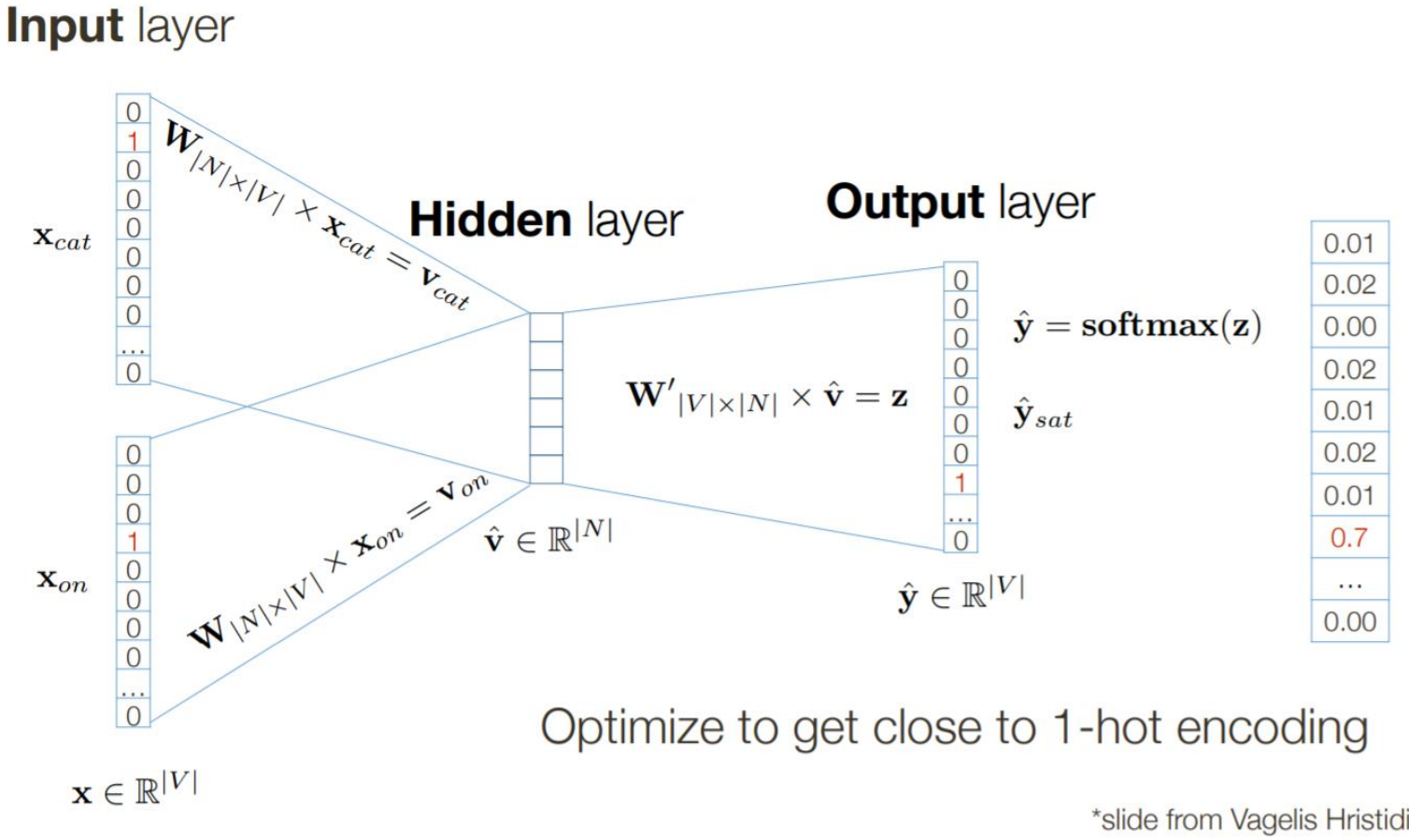
Document:
the **cat** **sat** **on** …

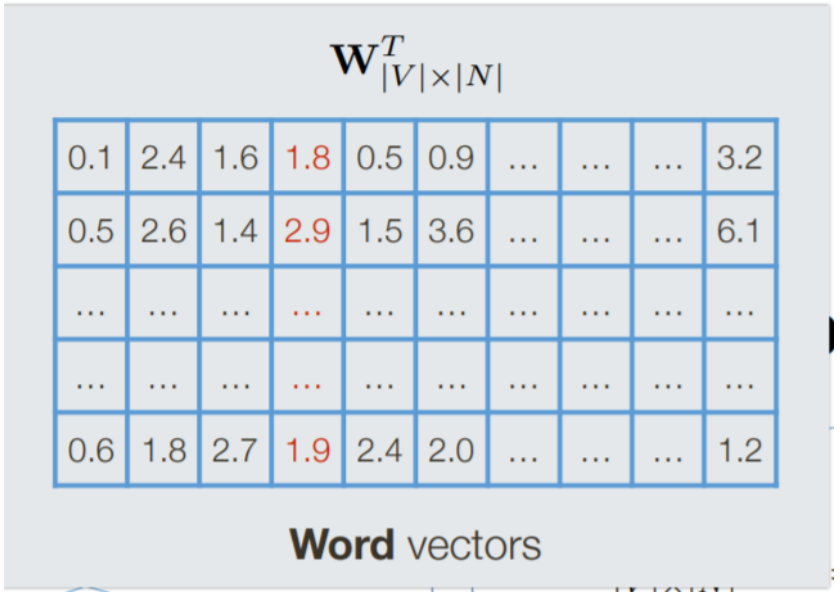


*slide from Vagelis Hristidis

Word embeddings

- We are not really interested in the prediction output, prediction is a “fake task”.
- We will not use the network for prediction. We will only use the learnt weights.
- Row j of the weight matrix is a **word embedding** for our word j.

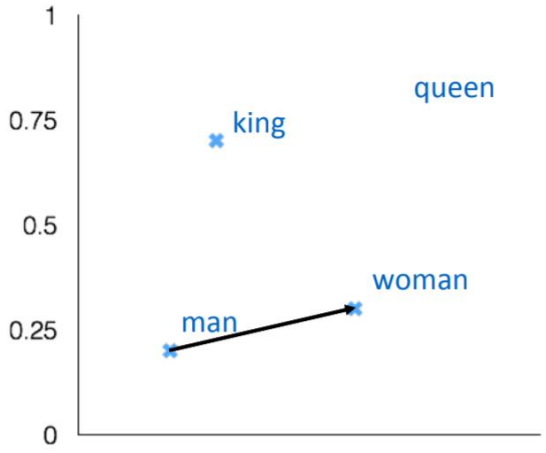




*slide from Vagelis Hristidis

Properties of word embeddings

- Word embeddings capture meaning and relationship between words.
- They encode word similarity and can capture complex patterns: discover words with similar context, word analogies.
- The resulting word embedding can be used as features in many natural language processing and machine learning applications.
- Learning embeddings this way is applicable to other sequential data, not only words (for example: genes).



man:woman :: king:?

+	king	[0.30 0.70]
-	man	[0.20 0.20]
+	woman	[0.60 0.30]
	queen	[0.70 0.80]

vector('king') - vector('man') + vector('woman') is close to **vector('queen')**

Visualization of word embeddings

http://suriyadeepan.github.io/