Introduction to Natural Language Processing

Applied Machine Learning Days
Lausanne 2018

Maria Han Veiga
PhD student @ ICS & I-Math
hmaria@physik.uzh.ch

What is Natural Language Processing (NLP)?

- Question answering

Computer Wins on ‘Jeopardy!’: Trivial, It’s Not

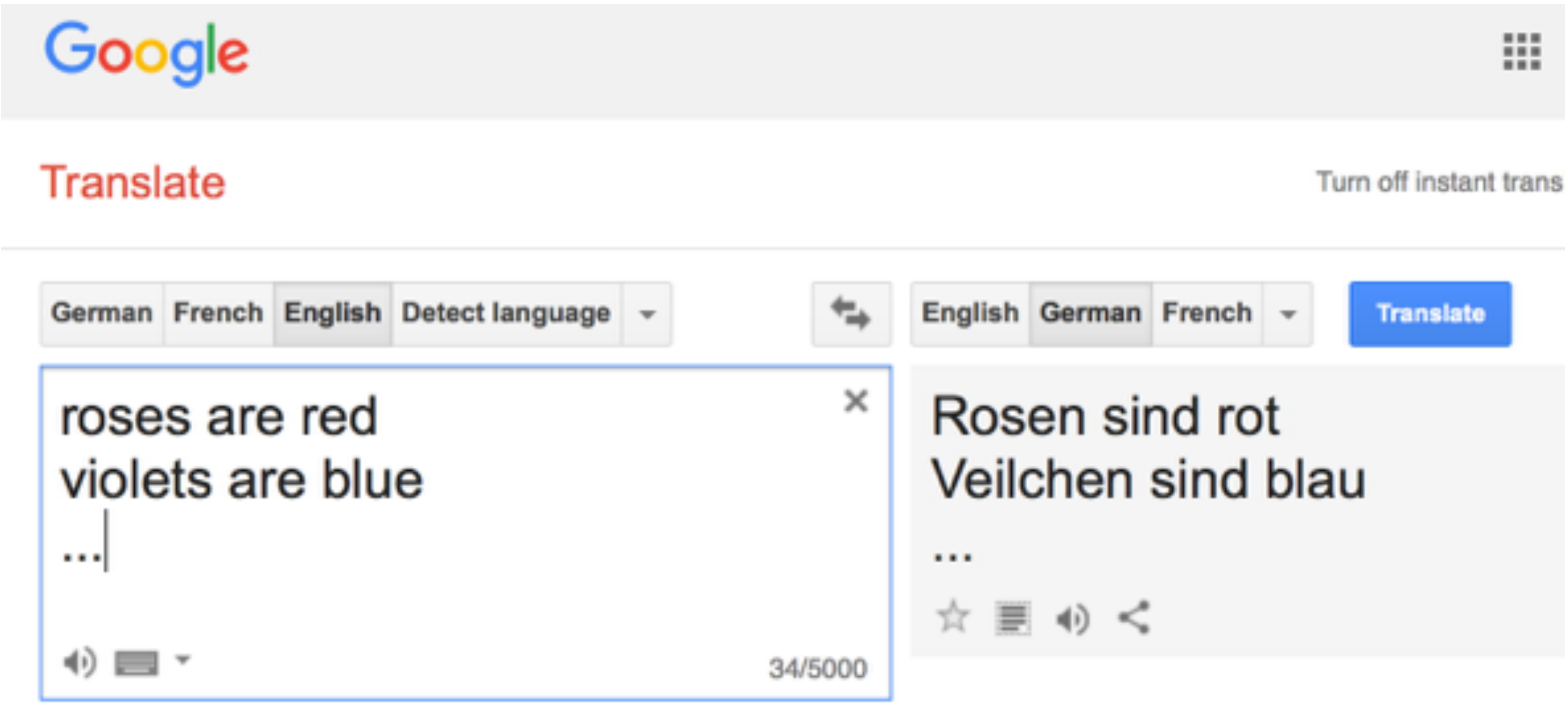
Carol Kaelson/Jeopardy Productions Inc., via Associated Press

Two "Jeopardy!" champions, Ken Jennings, left, and Brad Rutter, competed against a computer named Watson, which proved adept at buzzing in quickly.

http://www.nytimes.com/2011/02/17/science/17jeopardy-watson.html?pagewanted=all

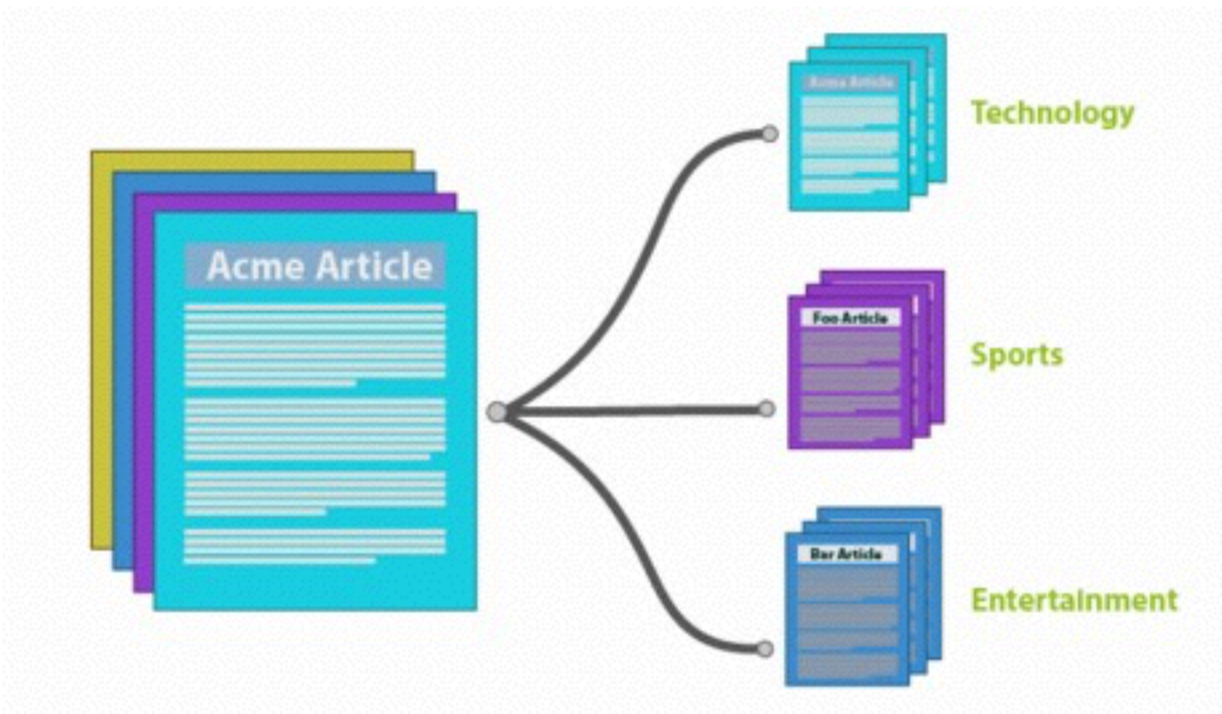
What is Natural Language Processing (NLP)?

- Machine translation



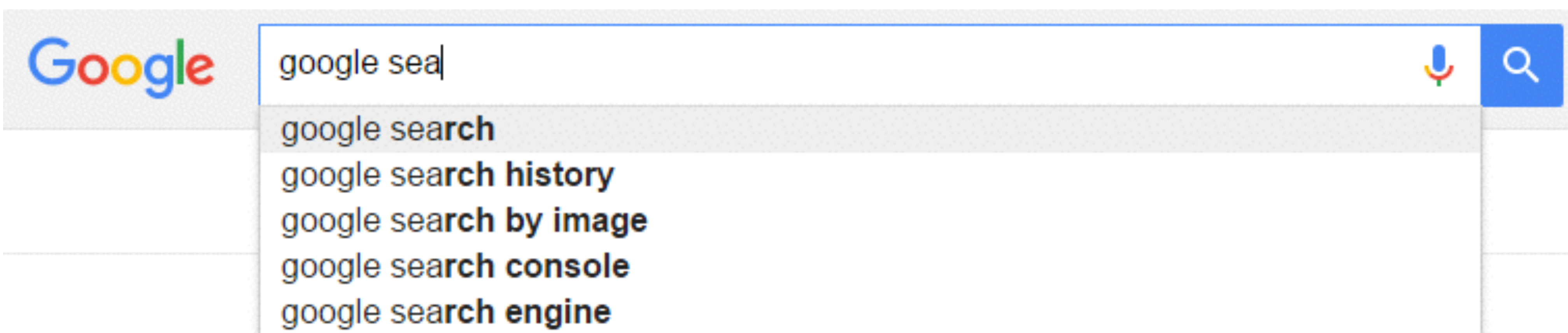
What is Natural Language Processing (NLP)?

- Document categorisation



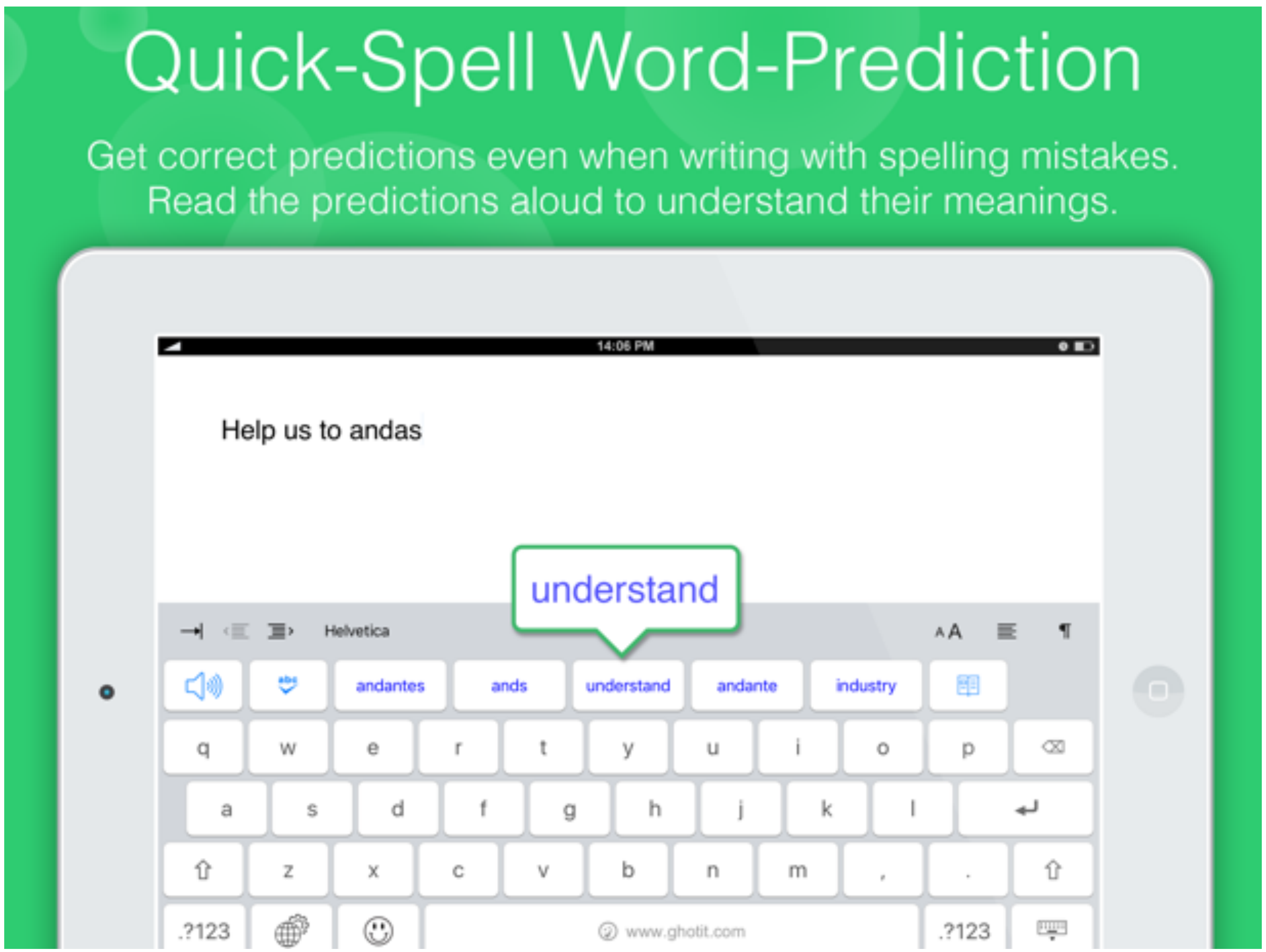
What is Natural Language Processing (NLP)?

- Search (information retrieval)



What is Natural Language Processing (NLP)?

- Spelling correction



Problem types

- Supervised learning:

  e.g: Document categorisation, sentiment analysis, spam detection, part-of-speech tagging

- Unsupervised learning:

  e.g: Document clustering, text summarization…

- Information retrieval:

  e.g: Search engines, question answering…

Why is it hard?

“Money is coined liberty, and so it is ten times dearer to the man who is deprived of freedom. If money is jingling in his pocket, he is half consoled, even though he cannot spend it. But money can always and everywhere be spent, and, moreover, forbidden fruit is sweetest of all.”

Source: “*The Old Soul*”, Fyodor Dostoyevsky

Why is it hard?

ambiguity
times - mathematical operation vs
plural of time

"Money is coined liberty, and so it is ten **times** dearer to the man who is deprived of freedom. If money is jingling in his pocket, he is half consoled, even though he cannot spend it. But money can always and everywhere be spent, and, moreover, forbidden fruit is sweetest of all."

Why is it hard?

Synonyms

“Money is coined **liberty**, and so it is ten times dearer to the man who is deprived of **freedom**. If money is jingling in his pocket, he is half consoled, even though he cannot spend it. But money can always and everywhere be spent, and, moreover, forbidden fruit is sweetest of all.”

Why is it hard?

"Money is coined liberty, and so it is ten times dearer to the man who is deprived of freedom. If money is jingling in his pocket, he is half consoled, even though he **cannot** spend it. But money can always and everywhere be spent, and, moreover, forbidden fruit is sweetest of all."

abbreviations/contractions
non-standardised text

Why is it hard?

“Money is coined liberty, and so it is ten times dearer to the man who is deprived of freedom. If money is jingling in his pocket, he is half consoled, even though he cannot spend it. But money can always and everywhere be spent, and, moreover, **forbidden fruit is sweetest of all**.”

idioms

Why is it hard?

“Money is coined liberty, and so it is ten times dearer to the man who is deprived of freedom. If money is jingling in his pocket, he is half consoled, even though he cannot spend **it** [the money]. But money can always and everywhere be spent, and, moreover, forbidden fruit is sweetest of all.”

Implicit meaning / references

Why is it hard?

“Money is coined liberty, and so it is ten times dearer to the man who is deprived of freedom. If money is jingling in his pocket, he is half consoled, even though he cannot spend it. But money can always and everywhere be spent, and, moreover, forbidden fruit is sweetest of all.”

In general: how to represent text? How to segment? How to deal with sequential dependence?

Why is it hard?

- Ambiguity
- Synonyms
- Abbreviations, typos
- Word segmentation: “New York”→“New” “York” or “New York”?
- Non-locality of text: **Money** is coined liberty (*…more than 20 words in between…*), he cannot spend **it.**
- Implicit meaning **e.g:** pronouns, idioms, etc…

Outline

- Basic concepts of NLP
- Feature representation
  - Vectorisation
  - Word embeddings
- Supervised learning
- Unsupervised learning

Basic concepts

Text normalisation:

- Tokenisation
- Normalisation
- Stemming and lemmatisation

Tokenisation

Given a sentence/text, tokenisation is the task of chopping it up into pieces, called *tokens.* (Perhaps at the same time throwing away certain characters, such as punctuation.)

e.g:

Input: “Friends, Romans, Countrymen, lend me your ears;”

Output: “Friends”, “Romans”, “Countrymen”, “lend”, “me”, “your”, “ears”

Normalisation

After tokenising a document/text, we might want some tokens to be "*merged*"

e.g:

"antidiscrimination", "anti-discrimination" → "antidiscrimination"

"Today", "today" → "today"

But this is not always obvious!

e.g: "C.A.T" should not be mapped to "cat"

Stemming/Lemmatisation

The goal of stemming and lemmatisation is to reduce inflectional forms and transform a word to a common base form.

e.g: *organise*, *organises*, and *organising* → *organise*

Stemming: chops off the ends of words in hopes to achieve this common base form.

e.g: car, cars, car's, cars' → car

Lemmatisation: uses a vocabulary and morphological analysis of words and returns the base or dictionary form of a word (knows as the lemma).

e.g: saw, seeing → see

To learn more, check out **Porter's algorithm** for stemming and **WordNet** for lemmatisation!

Feature representation

Ok, so now we know how to go from text:

"So much Fake News is being reported."

@realdonaldtrump

To tokens:

"So" "much" "Fake" "News" "is" "being" "reported" "."

Normalisation and lemmatisation:

"so" "much" "fake" "news" "is" "be" "report" "."

But still, how to perform computation on these objects?

Feature representation

Let’s say we have a set of Tweets that we want to do some computation on (e.g: identifying the author, from a list of authors)

T1: “We will continue to bring people together. We will not allow the Donald Trumps of the world to divide us up.”

T2: “I got into politics not to figure out how to become President. I got into politics because I give a damn.”

T3: “Does everyone see that the Democrats and President Obama are now, because of me, starting to deport people who are here illegally. Politics!”

T4: “As President, I WILL fix this rigged system and only answer to YOU, the American people!”

Feature representation

Preprocessing (e.g: tokens, normalisation, removing common words [1] and punctuation)

T1: “will” “continue” “bring” “people” “together” “will” “allow” “Donald” “Trumps” “world” “divide” “us” “up”

T2: “got” “politics” “figure” “become” “president” “got” “politics” “give” “damn”

T3: “everyone” “see” “democrats” “president” “Obama” “now” “starting” “deport” “people” “illegally” “politics”

T4: “president” “will” “fix” “rigged” “system” “answer” “American” “people”

[1] eg: https://www.ranks.nl/stopwords

Feature representation

Count occurrence of tokens

e.g. Author identification

Tokens / Tweets	politics	president	people	become	together	deport	...
T1	0	0	1	1	1	0	...
T2	2	1	0	0	0	0	...
T3	1	1	1	0	0	1	...
T4	0	1	1	0	0	0	...

T1: [0 0 1 1 1 0 . . .]

Feature representation

- This representation is called a **bag of words** model.

- Set of all tokens is called a **dictionary**.

- **Vectorising** a sentence means expressing the sentence in a vector.

    - We saw how to express a sentence in terms of a dictionary, by counting the occurrences.

Feature representation

Instead of a raw count of word appearance, we can have a better way to represent the words.

Term frequency-inverse document frequency is a numerical statistic that intends to reflect how important a word is to a document class in a collection of documents.

Feature representation

Term frequency (TF):

$$tf(f,d) = \frac{f_{t,d}}{\sum_{t' \in d} f_{t',d}}$$

Inverse document frequency (IDF):

$$idf(t,D) = \log \frac{N}{|\{d \in D : t \in d\}|}$$

TF-IDF measure:

$$tf(f,d) \times idf(t,D)$$

Feature representation

Term frequency (TF):

term frequency in document d

$$tf(f,d) = \frac{f_{t,d}}{\sum_{t' \in d} f_{t',d}}$$

all terms in document d

Inverse document frequency (IDF):

$$idf(t,D) = \log \frac{N}{|\{d \in D : t \in d\}|}$$

TF-IDF measure:

$$tf(f,d) \times idf(t,D)$$

Feature representation

Term frequency (TF):

$$tf(f, d) = \frac{f_{t,d}}{\sum_{t' \in d} f_{t',d}}$$

Inverse document frequency (IDF):

total number of documents

$$idf(t, D) = \log \frac{N}{|\{d \in D : t \in d\}|}$$

number of documents in which term t appears

TF-IDF measure:

$$tf(f, d) \times idf(t, D)$$

Feature representation

- Big limitations of this vector representation:

  1. Sequential nature is lost

     e.g: “is this true” - “this is true” have the exact same representation

  2. potential scalability issues (each word occupies a dimension, all words are orthogonal)

- Can improve 1. with using N-grams instead of 1-gram, at cost of increasing again the number of dimensions

  e.g: 2-gram

  “is this true” → “is this” “this true”

  “this is true” → “this is” “is true”

Feature representation

Word embeddings:

- Words are mapped to vectors: embed from a space with one dimension per word to a continuous vector space with lower dimension.

- Semantically similar words are mapped to nearby points

e.g: using word2vec[1]

vector('**king**') - vector('**man**') + vector('**woman**') $\approx$ vector('**queen**')

[1]https://code.google.com/archive/p/word2vec/

Feature representation

How to generate a word embedding?

Assumption: words occurring in similar context have similar meanings.

A language model uses previous words to predict the distribution of the next word.

i.e: to get the next word, we maximize the log likelihood: $P(w_n|w_{n-1}...w_1)$

which word n is most likely to occur after n-1, … 1 words?

Feature representation

We can write a co-occurence matrix:

e.g: "I like deep learning", "I like NLP", "I enjoy flying"

	I	like	deep	learning	NLP	enjoy	flying
I	0	2	0	0	0	1	0
like	2	0	1	0	1	0	0
deep	0	1	0	1	0	0	0
learning	0	0	1	0	0	0	0
NLP	0	1	0	0	0	0	0
enjoy	0	0	0	0	0	0	1
flying	0	0	0	0	0	1	0

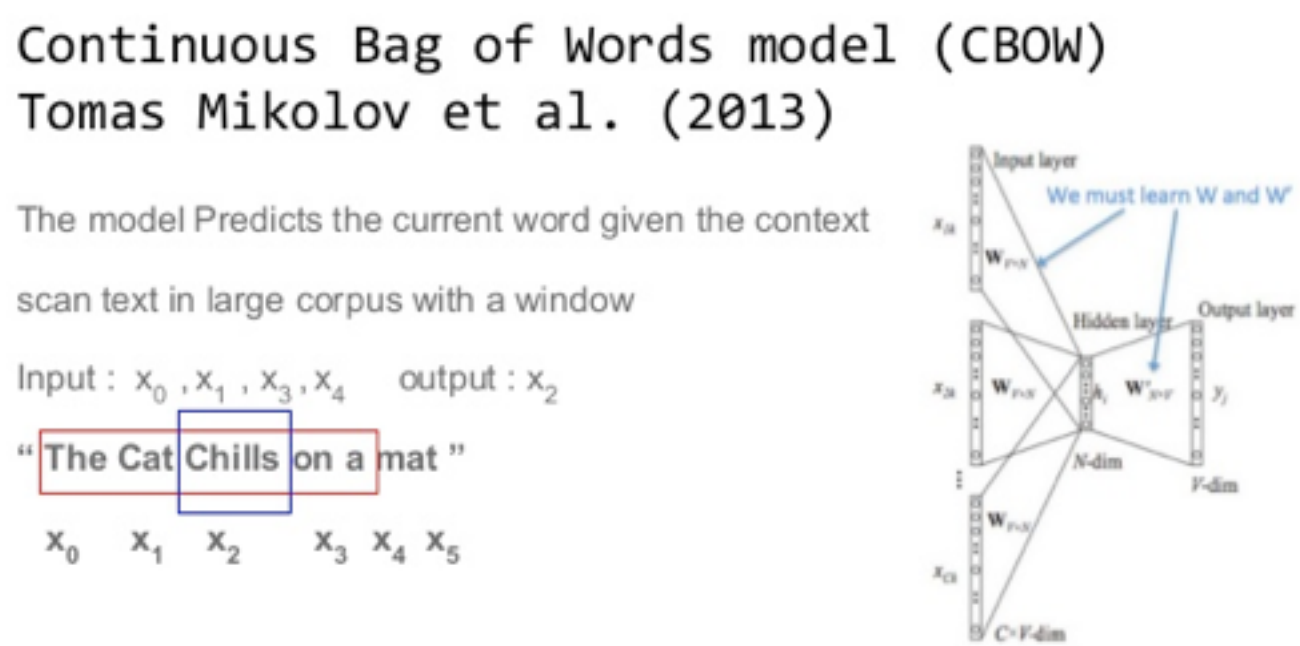
reduce matrix dimensionality, e.g: PCA

The intuition is that words which occur a similar context might be similar

[1] https://www.youtube.com/watch?v=ASn7ExxLZws

Feature representation

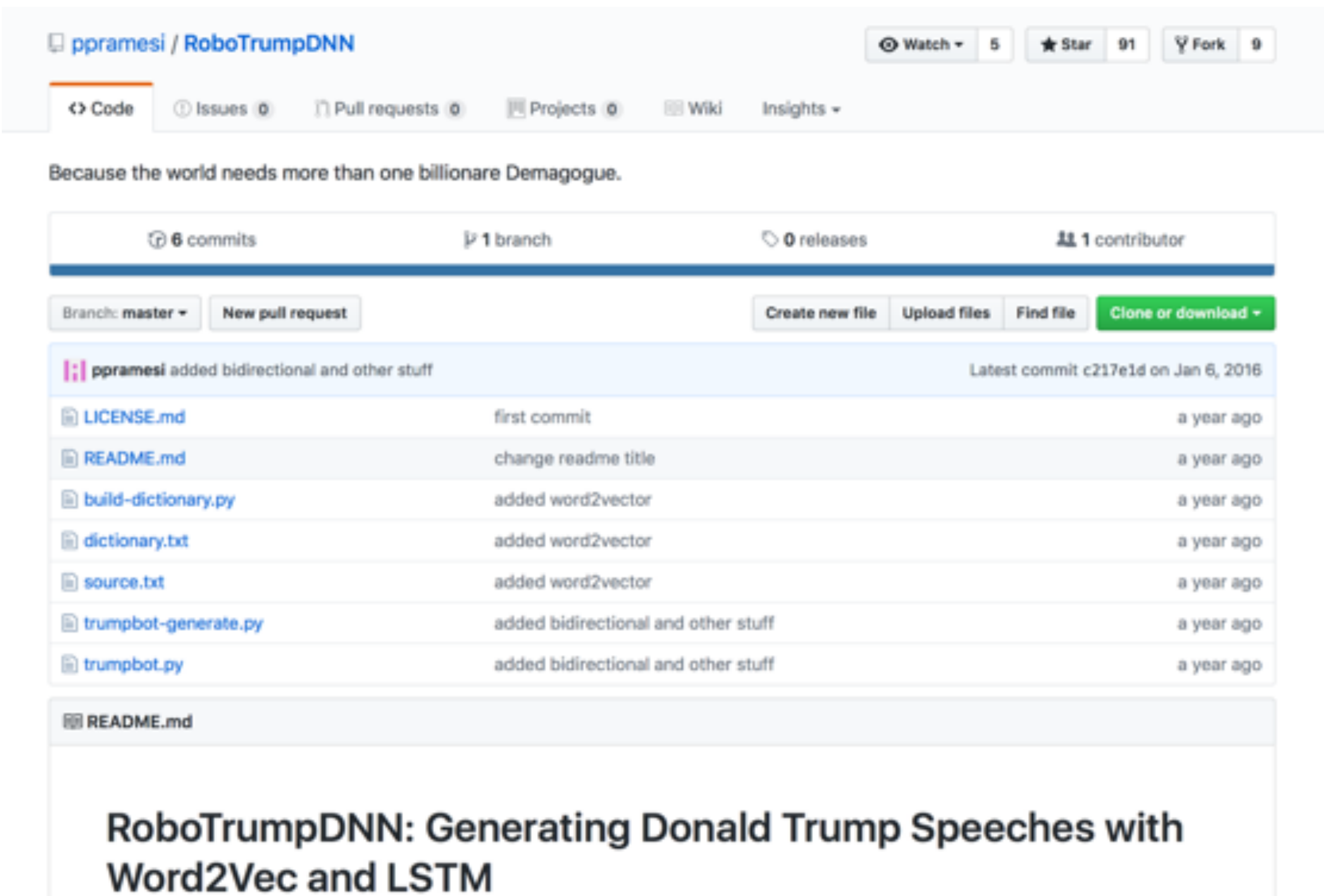
Another way to generate a word embedding is by using neural networks:



Here we have a language model, given some context, returns the most likely word. Between the input and output there is a hidden layer of smaller dimension which generates a representation of each word in the context. This is the word embedding.

[1] https://www.slideshare.net/hadyelsahar/word-embedings-why-the-hype-55769273

Feature representation



Output:
"rich. it was terrible. but saudi arabia, they make a billion dollars a day. i was the king. i was the king. i was the smartest person in yemen, we have to get to business. i have to say, but he was an early starter. and we have to get to business. i have to say, donald, i can't believe it."

https://github.com/ppramesi/RoboTrumpDNN

What we talked about so far…

1. Traditional ways in Natural Language processing to split up a sentence (tokenise) and normalise lemmatise and find the stem of words

2. Representing sentences as vectors (Count or TFIDF) or collection of vectors (word embeddings)

Supervised Learning

Let’s imagine the task of identifying authors of Tweets.

T1: “We will continue to bring people together. We will not allow the Donald Trumps of the world to divide us up.” (Sanders)

T2: “I got into politics not to figure out how to become President. I got into politics because I give a damn.” (Sanders)

T3: “Does everyone see that the Democrats and President Obama are now, because of me, starting to deport people who are here illegally. Politics!” (Trump)

T4: “As President, I WILL fix this rigged system and only answer to YOU, the American people!” (Trump)

T5: “Funny to hear the Democrats talking about the National Debt when President Obama doubled it in only 8 years!” (?)

Supervised Learning

Text classification:

Input:

- document **d**
- fixed set of classes **C = {c1, c2,..., cJ}**
- A training set of **m** hand-labeled documents **(d1,c1),....,(dm,cm)**

Output:

- a learner classifier $\gamma : d \to c$

Supervised Learning

Text classification:

Input:

tweet A

- document **d**

classes:
Trump
Sanders

- fixed set of classes **C = {c1, c2,..., cJ}**
- A training set of **m** hand-labeled documents **(d1,c1),....,(dm,cm)**

A set of tweets from Trump
and a set of tweets from Sanders

Output:

- a learner classifier $\gamma : d \rightarrow c$

Text classification

Classifier: **Naive bayes**

- Simple classification method based on Bayes rule
- Relies on very simple representation of document: **Bag of words**

  e.g: “John eats an apple” -> [0 0 0 0 1 0 0 0 …. 1]

- For a document **d** and a class **c:**

$$P(c|d) = \frac{P(d|c)P(c)}{P(d)}$$

Text classification - Probabilistic models

finds class c which maximises this conditional probability

$$
\begin{aligned}
c_{MAP} &= \arg\max_{c \in C} P(c|d) \\
&= \arg\max_{c \in C} \frac{P(d|c)P(c)}{P(d)} \\
&\propto \arg\max_{c \in C} P(d|c)P(c)
\end{aligned}
$$

Bayes rule

Text classification - Probabilistic models

$$
\begin{aligned}
c_{MAP} &\propto \arg\max_{c \in C} P(d|c)P(c) \\
&= \arg\max_{c \in C} P(x_1, x_2, ..., x_n|c)P(c)
\end{aligned}
$$

tweet is represented by
a vector

Text classification - Probabilistic models

$$
\begin{aligned}
c_{MAP} &\propto \arg\max_{c \in C} P(d|c)P(c) \\
&= \arg\max_{c \in C} P(x_1, x_2, ..., x_n|c)P(c)
\end{aligned}
$$

Assuming **conditional independence between words**

$$P(x_1, x_2, ..., x_n|c) = P(x_1|c) \cdot P(x_2|c) \cdot ... \cdot P(x_n|c)$$

$$c_{NB} = \arg\max_{c \in C} P(c) \prod_{x \in X} P(x|c)$$

Text classification - Probabilistic models

$$c_{NB} = \arg\max_{c \in C} P(c) \prod_{x \in X} P(x|c)$$

We need to estimate the probabilities P(c) and P(x|c)

Maximum likelihood estimates:

$$\hat{P}(c_j) = \frac{|\{\text{d s.t. label(d)} = c_j\}|}{N}$$

proportion of documents in class c_j in the training set

$$\hat{P}(x_i|c_j) = \frac{count(x_i, c_j)}{\sum_{x \in V} count(w, c_j)}$$

fraction of times word x_i appears among all words in documents of topic c_j

Text classification - Probabilistic models

$$\hat{P}(x_i|c_j) = \frac{count(x_i, c_j)}{\sum_{x \in V} count(w, c_j)}$$

fraction of times word x_i appears among all words in documents of topic c_j

What if x_i never appears in a document of class c_j ?

$$\hat{P}(x_i|c_j) = \frac{count(x_i, c_j) + 1}{(\sum_{x \in V} count(x, c_j) + 1)}$$

Laplace smoothing

Supervised Learning

Let’s imagine the task of identifying authors of Tweets.

T1: “We will continue to bring people together. We will not allow the Donald Trumps of the world to divide us up.” (Sanders)

T2: “I got into politics not to figure out how to become President. I got into politics because I give a damn.” (Sanders)

T3: “Does everyone see that the Democrats and President Obama are now, because of me, starting to deport people who are here illegally. Politics!” (Trump)

T4: “As President, I WILL fix this rigged system and only answer to YOU, the American people!” (Trump)

T5: “Funny to hear the Democrats talking about the National Debt when President Obama doubled it in only 8 years!” (?)

Text classification - Probabilistic models

	Doc	Words	Class
Training	1	“will” “continue” “bring” “people” “together” “will” “allow” “Donald” “Trumps” “world” “divide” “us” “up”	Sanders
	2	“got” “politics” “figure” “become” “president” “got” “politics” “give” “damn”	Sanders
	3	“everyone” “see” “democrats” “president” “Obama” “now” “starting” “deport” “people” “illegally” “politics”	Trump
	4	“president” “will” “fix” “rigged” “system” “answer” “American” “people”	Trump
Test	5	"funny" "hear" "democrats" "talking" "national" "debt" "president" "Obama" "doubled" "8" "years"	?

$$\hat{P}(c) = \frac{N_c}{N}$$

$$\hat{P}(x|c) = \frac{count(x,c) + 1}{count(c) + |V|}$$

Text classification - Probabilistic models

	Doc	Words	Class
Training	1	“people” “together” “Donald” “Trumps” “divide”	Sanders
	2	“politics” “president” “politics” “damn”	Sanders
	3	“democrats” “president” “Obama” “people” “illegally” “politics”	Trump
	4	“president” “American” “people”	Trump
Test	5	“democrats” "president" “Obama"	?

$$\hat{P}(c) = \frac{N_c}{N}$$

$$\hat{P}(x|c) = \frac{count(x,c)+1}{count(c)+|V|}$$

$$\hat{P}("Sanders") = 0.5$$

$$\hat{P}("Trump") = 0.5$$

$$\hat{P}("democrats"|"Sanders") = \frac{0+1}{9+12}$$

$$\hat{P}("democrats"|"Trump") = \frac{1+1}{9+12}$$

Text classification - Probabilistic models

	Doc	Words	Class
Training	1	“people” “together” “Donald” “Trumps” “divide”	Sanders
	2	“politics” “president” “politics” “damn”	Sanders
	3	“democrats” “president” “Obama” “people” “illegally” “politics”	Trump
	4	“president” “American” “people”	Trump
Test	5	“democrats” "president" “Obama"	?

$$c_{NB} = \arg\max_{c \in C} P(c) \prod_{x \in X} P(x|c)$$

$$\hat{P}(''Sanders'') = 0.5$$

$$\hat{P}(''Trump'') = 0.5$$

$$\hat{P}(''democrats''|''Sanders'') = \frac{0+1}{9+12}$$

$$\hat{P}(''democrats''|''Trump'') = \frac{1+1}{9+12}$$

$$\hat{P}(T5|''Trump'') \approx 0.00087$$

$$\hat{P}(T5|''Sanders'') \approx 0.00013$$

Naive Bayes

- Naive Bayes classifiers can use any sort of feature:

  e.g: URL, email address, dictionaries, network features…

- But only word features

Naive Bayes

- Very Fast, low storage requirements
- Good in domains with many equally important features
  - Decision Trees suffer from fragmentation in such cases – especially if not much data
- In general, a good dependable baseline for text classification

Other classifiers

- SVM
- Random Forest
- Adaboost
- Neural networks

Text classification - Pipeline

1. From training corpus, extract **dictionary** (set of unique words).
2. Create **vectoriser** to transform documents into vectors
3. Using Training data {(d,c)}, **train a classifier of choice**
    1. To avoid overfitting a dataset (learning a function which performs well on the dataset but generalises poorly), do **cross-validation**. The training procedure is repeated on a subset of the training data and evaluated on the excluded training data.
4. Given a test set {d'}
    1. vectorise documents using **vectoriser**
    2. use trained classifier to predict class:
        1. Find the label c which maximises probability P(c|d)

Text classification

1. Document classification
2. Sentiment analysis (good, bad, neutral)
3. Fake news detection (true, false, unverified)
4. Spam detection
5. Part of speech tagging (e.g. John → noun, eats → verb . . .)

and more!

Unsupervised learning

Essentially: there are no explicit labels

Examples of unsupervised learning are:

- clustering
- learning word embeddings (as seen before!)
- topic modelling
- summarisation
- query generation
- retrieval

Document Similarity

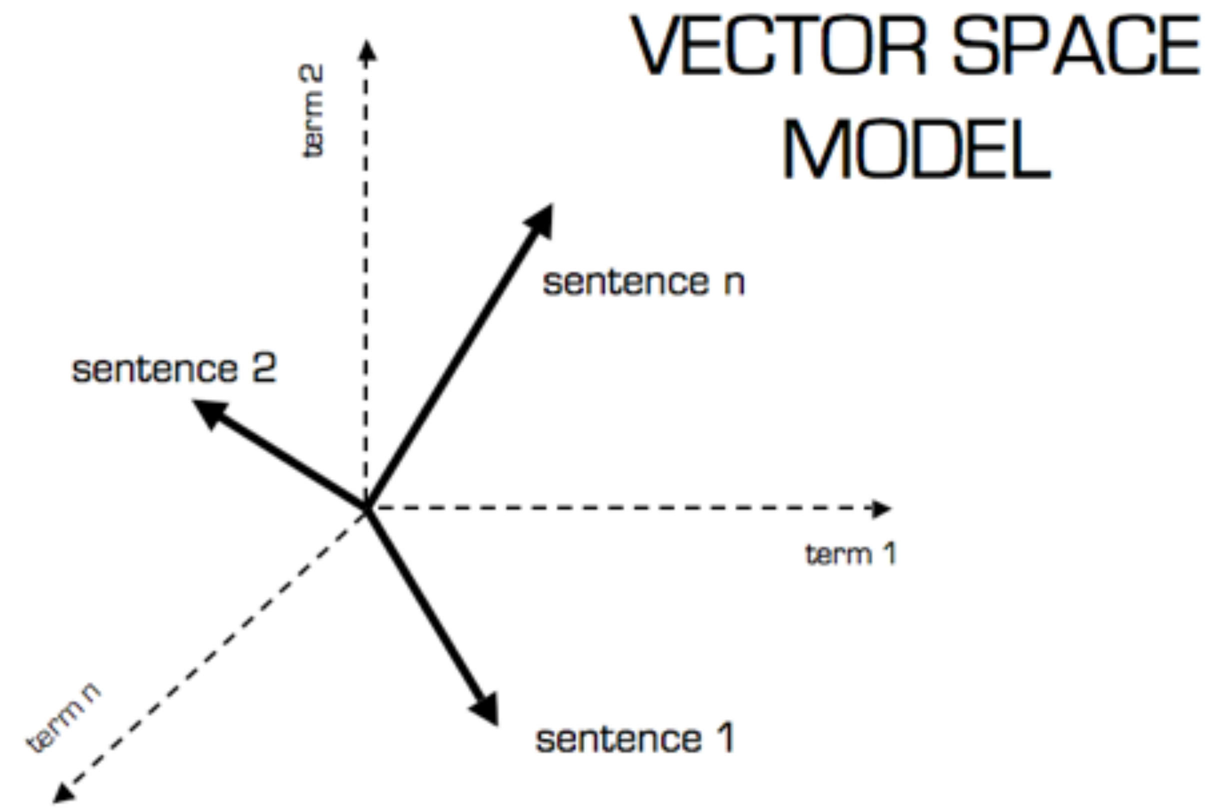
- We want to find documents which are similar to each other (common in search) but we don't necessarily have labelled data

e.g:

- Information retrieval problem:
  - Issue search query **Q**
  - Find set of documents {d} which are relevant to query **Q**

Document Similarity

- Suppose we have a vector representation of **Q** and documents {**d**}.
    - Q = [1 0 0 0 0 1 .. . .]
    - d = [1 1 1 0 1 0 1 …]
- These live in a vector space.
- We can compute the distance between two vectors using the cosine distance



$$\vec{a} \cdot \vec{b} = \|\vec{a}\| \|\vec{b}\| \cos\theta$$

$$\cos\theta = \frac{\vec{a} \cdot \vec{b}}{\|\vec{a}\| \|\vec{b}\|}$$

intuition: how well does Q ‘represent’ d?

Topic modelling

- **Problem to solve:** Given a set of documents, can we extract ‘topics’ from them?
- LDA (Latent Dirichlet Analysis):
    - A document is a mixture of various topics
    - Each word in a document belongs to a topic
    - A Bayesian inference model that associates each document with a probability distribution over topics, where topics are probability distributions over words

Performance metrics

- Accuracy:

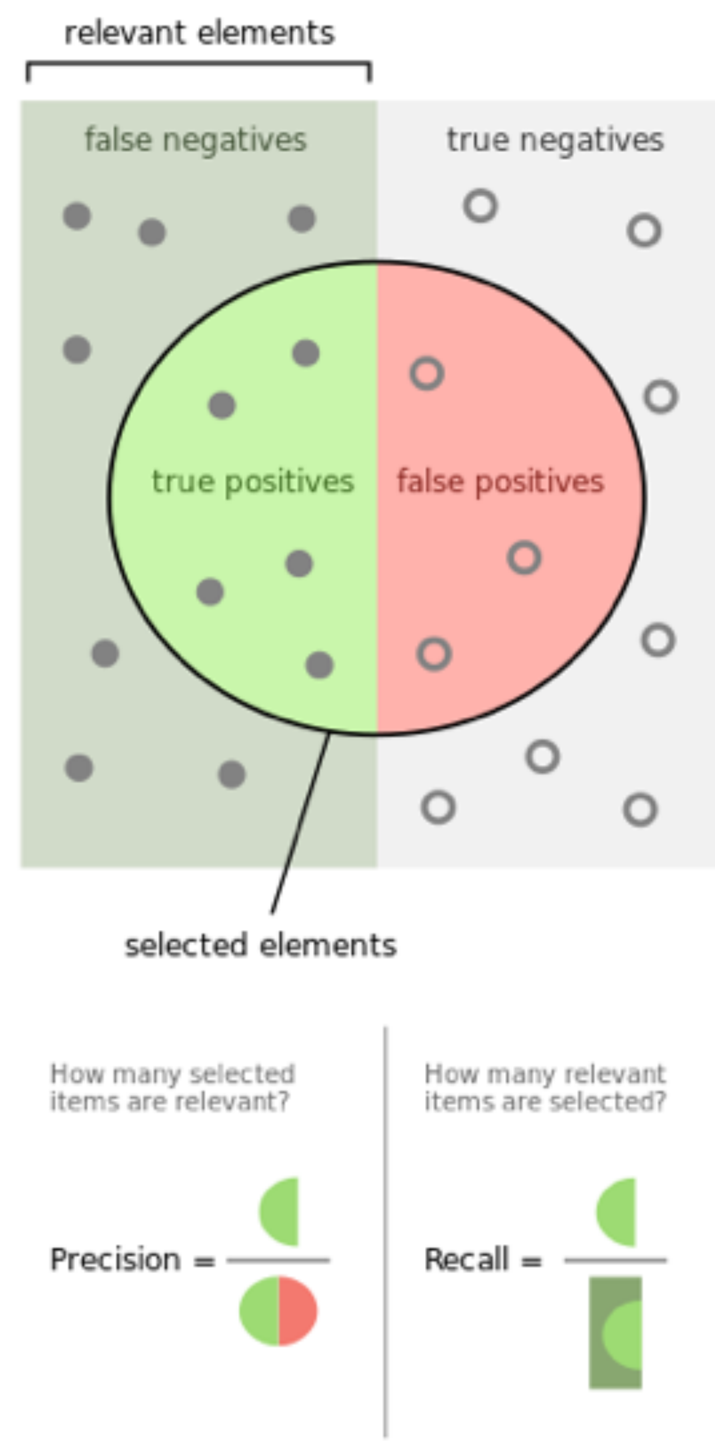
$$\frac{TP + TN}{total}$$

- Precision:

$$\frac{TP}{TP + FP}$$

- Recall:

$$\frac{TP}{TP + FN}$$



Performance metrics

- Accuracy:

$$\frac{TP + TN}{total}$$

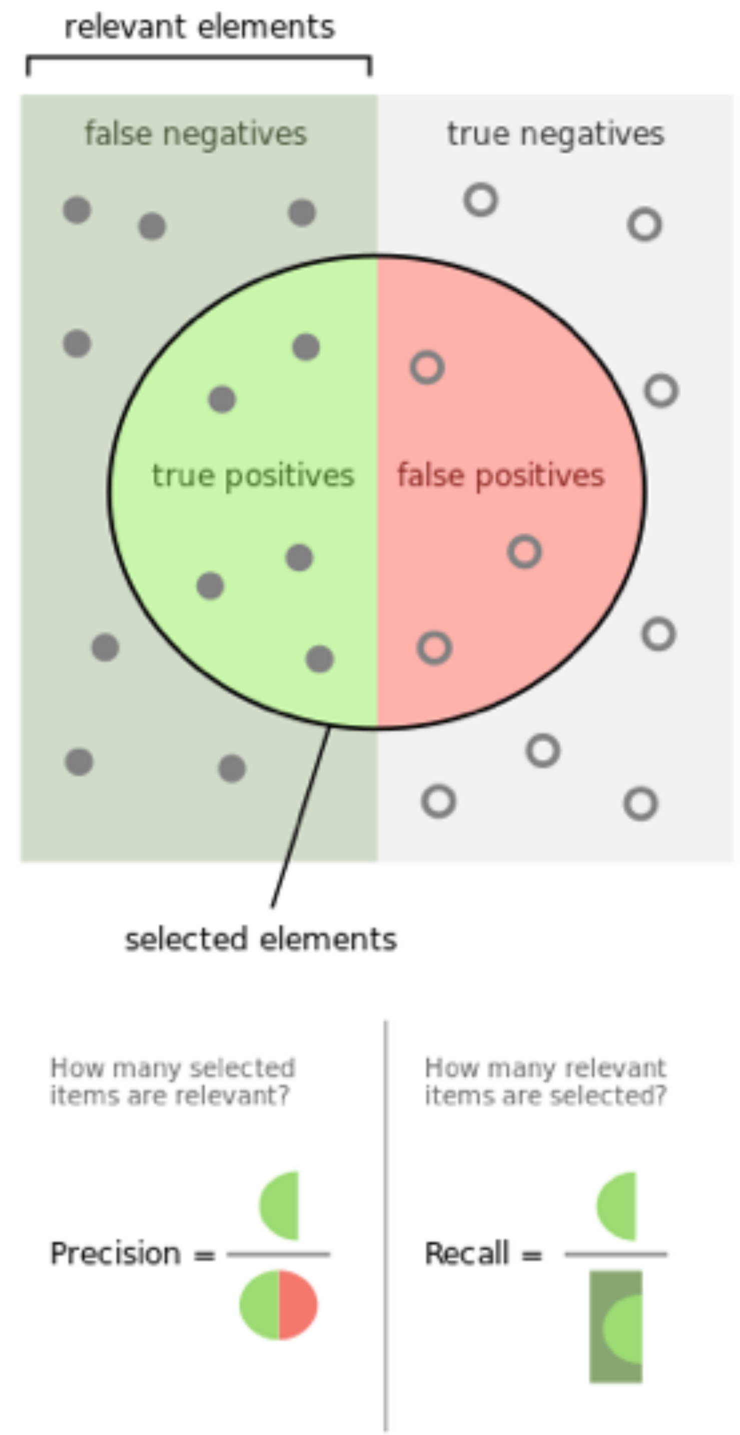
- Precision:

$$\frac{TP}{TP + FP}$$

out of the things we said were positive,
how many are actually positive?

- Recall:

$$\frac{TP}{TP + FN}$$



Performance metrics

- Accuracy:

$$\frac{TP + TN}{total}$$

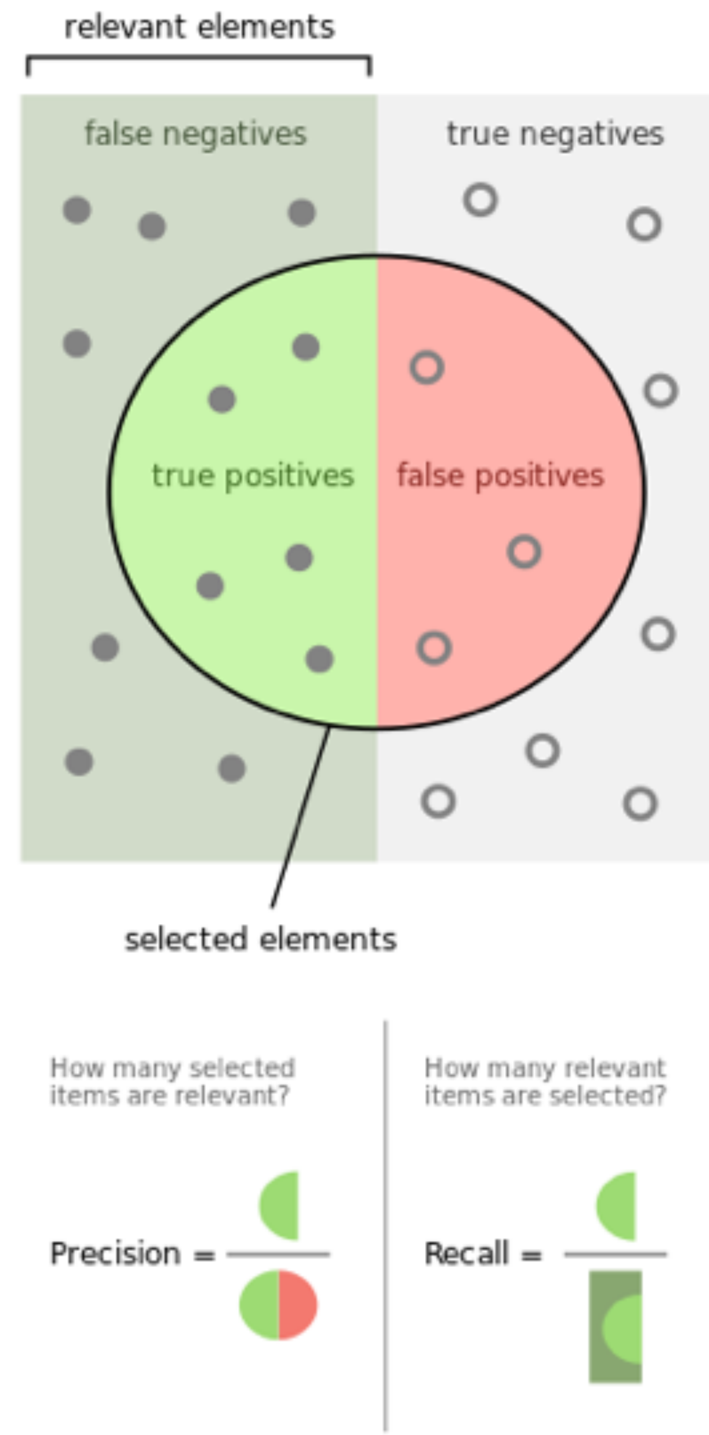
- Precision:

$$\frac{TP}{TP + FP}$$

- Recall:

$$\frac{TP}{TP + FN}$$

how many positives did we measure out of all the positives that exist?



Performance metrics

- Accuracy:

$$\frac{TP + TN}{total}$$

- Precision:

$$\frac{TP}{TP + FP}$$

- Recall:

$$\frac{TP}{TP + FN}$$

True labels:
99% false
1% true

If I always predict false:
Accuracy: 99%

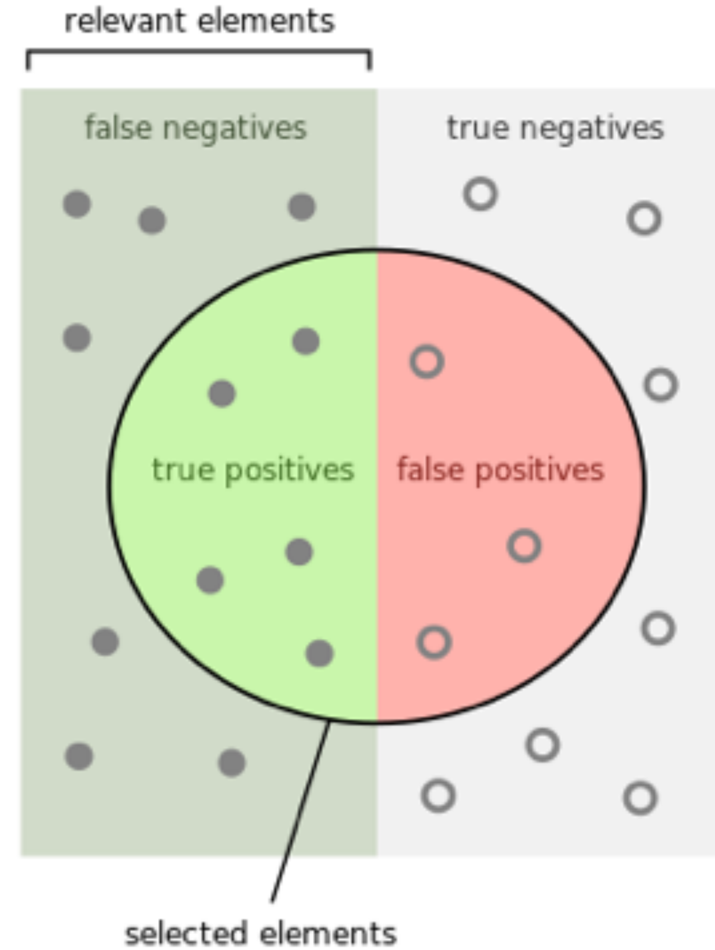
Precision:
0%

Recall:
0%

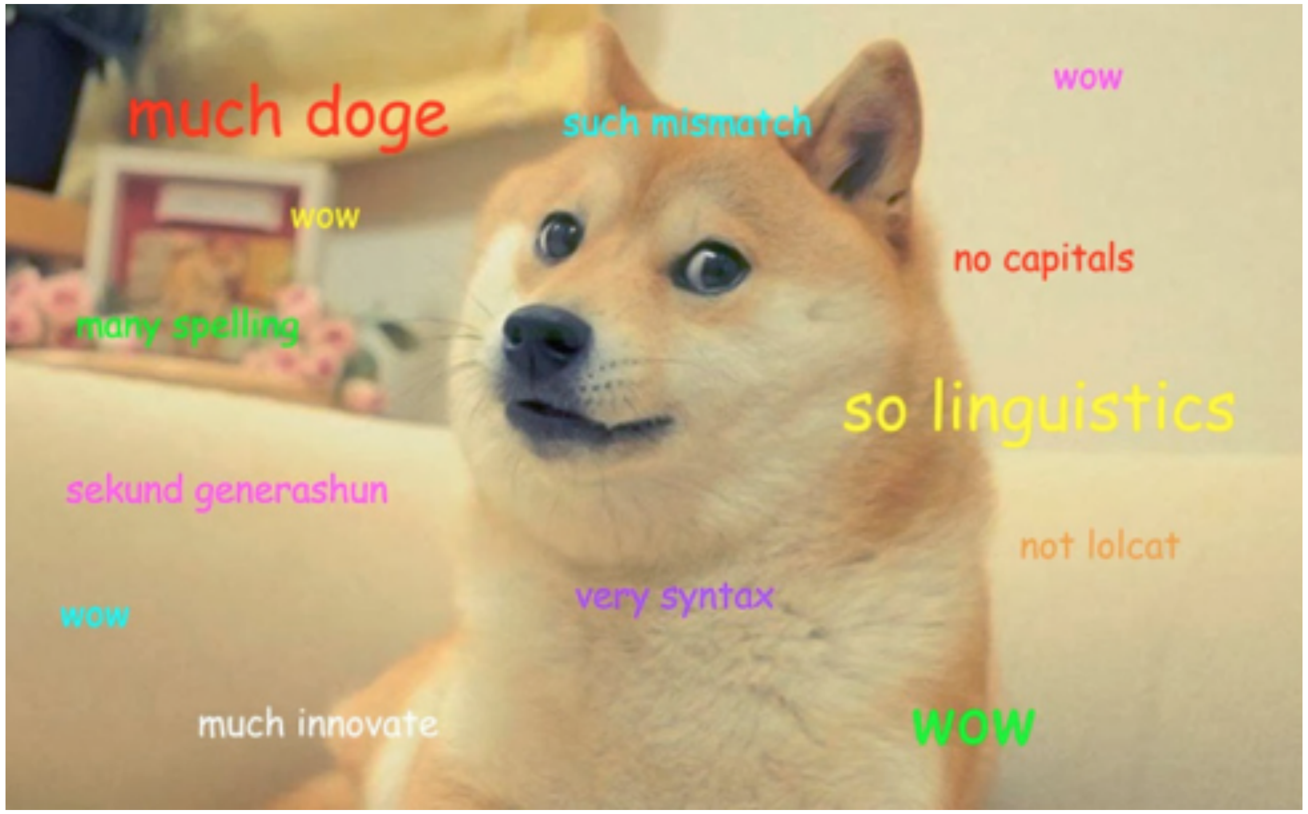
If I over predict the true label:
Accuracy: <99%

Precision:
>0%

Recall:
>0%



Thank you for your attention! :3



Connect over twitter or github @hanveiga