Computational Linguistics

Statistical NLP

Aurélie Herbelot

2018

Centre for Mind/Brain Sciences
University of Trento

Table of Contents

Probabilities in NLP

The probability of a word

- Most introductions to probabilities start with coin and dice examples:
  - The probability P(H) of a fair coing falling heads is 0.5.
  - The probability P(2) of rolling a 2 with a fair six-sided die is $\frac{1}{6}$.
- Let's think of a word example:
  - The probability P(the) of a speaker uttering *the* is...?

Words and dice

- The occurrence of a word is like a throw of a loaded dice...
- except that we don’t know how many sides the dice has (what is the vocabulary of a speaker?)
- and we don’t know how many times the dice has been thrown (how much the speaker has spoken).

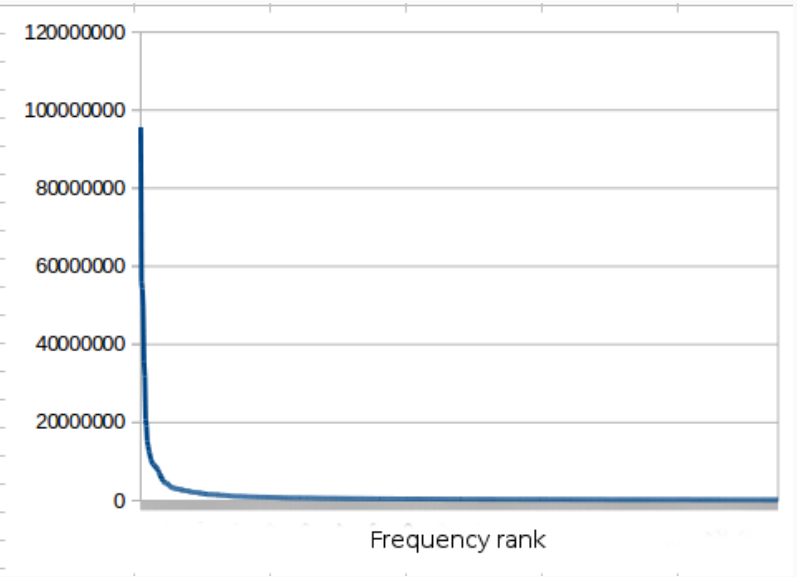
Using corpora

- There is actually little work done on individual speakers in NLP.
- Mostly, we will do machine learning from a *corpus*: a large body of text, which may or may not be representative of what an individual might be exposed to.
- We can imagine a corpus as the concatenation of what many people have said.
- **But individual subjects are not retrievable from the data.**

Zipf Law

- From corpora, we can get some general idea of the likelihood of a word by observing its frequency in a large corpus.

95630829	the
56069188	of
54204048	and
49798143	to
35772166	a
31703195	in
20870074	is
19143807	for
15288542	that
14115656	on
12745509	with
12171873	be
10768897	are
10347955	as
9586761	by
9485045	it
9260275	you
8794676	was
8778097	at
8366020	from



Corpora vs individual speakers

Machine exposed to:

100M words (BNC)

2B words (ukWaC)

100B words (Google News)

3-year old child exposed to:

25M words (US)

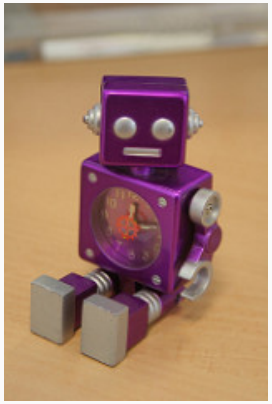
20M words (Dutch)

5M words (Mayan)

(*Cristia et al 2017)*

A unigram language model

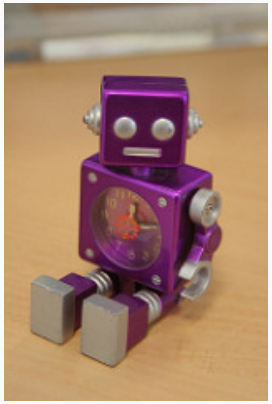
- A language model (LM) is a model that computes the probability of a sequence of words, given some previously observed data.
- A unigram LM assumes that the probability of each word can be calculated in isolation.



A robot with two words: ‘o’ and ‘a’. The robot says: *o a a*.
What might it say next? How confident are you in your answer?

A unigram language model

- A language model (LM) is a model that computes the probability of a sequence of words, given some previously observed data.
- A unigram LM assumes that the probability of each word can be calculated in isolation.



Now the robot says:
o a a o o o o o o o o o o o o o o a o o o o.
What might it say next? How confident are you in your answer?

A unigram language model

- $P(A)$: the frequency of event A, relative to all other possible events, given an experiment repeated an *infinite* number of times.
- The estimated probabilities are approximations:
  - *o a a*:
    $P(a) = \frac{2}{3}$ with low confidence.
  - *o a a o o o o o o o o o o o o o o o a o o o o*:
    $P(a) = \frac{3}{22}$ with somewhat better confidence.
- So more data is better data...

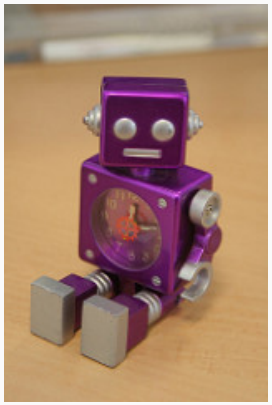
Example unigram model

- We can generate sentences with a language model, by sampling words out of the calculated probability distribution.
- Example sentences generated with a unigram model (taken from Dan Jurasky):
  - *fifth an of futures the an incorporated a a the inflation most dollars quarter in is mass*
  - *thrift did eighty said hard 'm july bullish*
  - *that or limited the*

Conditional probability and bigram language models

$P(A|B)$: the probability of A given B.

$$P(A|B) = \frac{P(A \cap B)}{P(B)}$$



The robot now knows three words. It says:

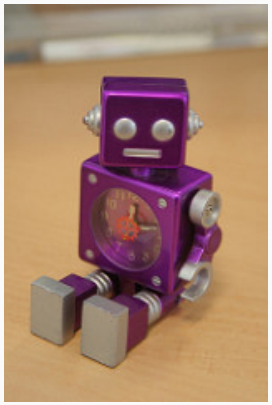
o o o o o a i o o a o o o a i o o o a i o o a

What is it likely to say next?

Conditional probability and bigram language models

$P(A|B)$: the probability of A given B.

$$P(A|B) = \frac{P(A \cap B)}{P(B)}$$



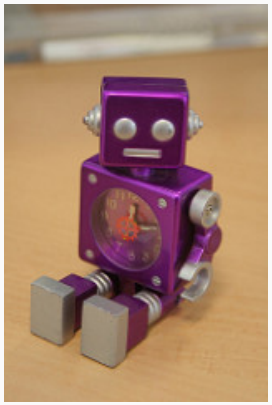
o o o o o a i o o a o o o a i o o o a i o o a

$$P(a|a) = \frac{c(a,a)}{c(a)} = \frac{0}{4}$$

Conditional probability and bigram language models

$P(A|B)$: the probability of A given B.

$$P(A|B) = \frac{P(A \cap B)}{P(B)}$$



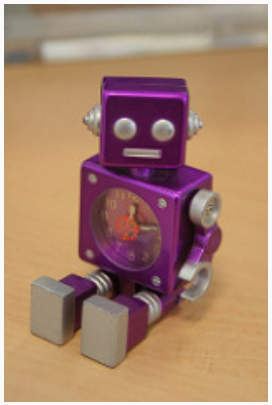
o o o o o a i o o a o o o a i o o o a i o o a

$$P(o|a) = \frac{c(o,a)}{c(a)} = \frac{1}{4}$$

Conditional probability and bigram language models

$P(A|B)$: the probability of A given B.

$$P(A|B) = \frac{P(A \cap B)}{P(B)}$$



o o o o o a i o o a o o o a i o o o a i o o a

$$P(i|a) = \frac{c(i,a)}{c(a)} = \frac{3}{4}$$

Example bigram model

- Example sentences generated with a bigram model (taken from Dan Jurasky):
  - *texaco rose one in this issue is pursuing growth in a boiler house said mr. gurria mexico 's motion control proposal without permission from five hundred fifty five yen*
  - *outside new car parking lot of the agreement reached*
  - *this would be a record november*

Example bigram model

- Example sentences generated with a bigram model (taken from Dan Jurasky):
  - *texaco rose one in this issue is pursuing growth in a boiler house said mr. gurria mexico 's motion control proposal without permission from five hundred fifty five yen*
  - *outside new car parking lot of the agreement reached*
  - *this would be a record november*
- **Btw, what do you think the model was trained on?**

The Markov assumption

- Why are those sentences so weird?
- We are estimating the probability of a word without taking into account the broader context of the sentence.

The Markov assumption

- Let's assume the following sentence:

  The robot is talkative.

- We are going to use the chain rule for calculating its probability:
  $\mathrm{P}(A_n, \ldots, A_1) = \mathrm{P}(A_n | A_{n-1}, \ldots, A_1) \cdot \mathrm{P}(A_{n-1}, \ldots, A_1)$
- For our example:
  $\mathrm{P}(\textit{the}, \textit{robot}, \textit{is}, \textit{talkative}) = \mathrm{P}(\textit{talkative} \mid$
  $\textit{is}, \textit{robot}, \textit{the}) \cdot \mathrm{P}(\textit{is} \mid \textit{robot}, \textit{the}) \cdot \mathrm{P}(\textit{robot} \mid \textit{the}) \cdot \mathrm{P}(\textit{the})$

The Markov assumption

- The problem is, we cannot easily estimate the probability of a word in a long sequence.
- There are too many possible sequences that are not observable in our data or have very low frequency: $\mathrm{P}(\textit{talkative} \mid \textit{is}, \textit{robot}, \textit{the})$
- So we make a simplifying Markov assumption: $\mathrm{P}(\textit{talkative} \mid \textit{is}, \textit{robot}, \textit{the}) \approx \mathrm{P}(\textit{talkative} \mid \textit{is})$ (bigram)
  or
  $\mathrm{P}(\textit{talkative} \mid \textit{is}, \textit{robot}, \textit{the}) \approx \mathrm{P}(\textit{talkative} \mid \textit{is}, \textit{robot})$ (trigram)

The Markov assumption

- Coming back to our example:
  $\mathrm{P}(\textit{the}, \textit{robot}, \textit{is}, \textit{talkative}) = \mathrm{P}(\textit{talkative} \mid \textit{is}, \textit{robot}, \textit{the}) \cdot \mathrm{P}(\textit{is} \mid \textit{robot}, \textit{the}) \cdot \mathrm{P}(\textit{robot} \mid \textit{the}) \cdot \mathrm{P}(\textit{the})$
- A bigram model simplifies this to:
  $\mathrm{P}(\textit{the}, \textit{robot}, \textit{is}, \textit{talkative}) = \mathrm{P}(\textit{talkative} \mid \textit{is}) \cdot \mathrm{P}(\textit{is} \mid \textit{robot}) \cdot \mathrm{P}(\textit{robot} \mid \textit{the}) \cdot \mathrm{P}(\textit{the})$
- That is, we are not taking into account *long-distance dependencies* in language.
- Trade-off between accuracy of the model and trainability.

Naive Bayes

Naive Bayes

- A classifier is a ML algorithm which:
  - as input, takes *features*: computable aspects of the data, which we think are relevant for the task;
  - as output, returns a *class*: the answer to a question/task with multiple choices.
- A Naive Bayes classifier is a simple probabilistic classifier:
  - apply Bayes' theorem;
  - (naive) assumption that features input into the classifier are independent.
- Used mostly in document classification (e.g. spam filtering, classification into topics, authorship attribution, etc.)

Probabilistic classification

- We want to model the conditional probability of output labels **y** given input **x**.
- For instance, model the probability of a film review being positive (y) given the words in the review (x), e.g.:
  - $y = 1$ (review is positive) or $y = 0$ (review is negative)
  - $x = \{$ ... the, worst, action, film, ... $\}$
- We want to evaluate $P(y|x)$ and find $\text{argmax}_y\, P(y|x)$ (the class with the highest probability).

Bayes' Rule

- We can model $P(y|x)$ through Bayes' rule:

$$P(y|x) = \frac{P(x|y)P(y)}{P(x)} \tag{1}$$

- Finding the argmax means using the following equivalence:

$$\underset{y}{\text{argmax}}\, P(y|x) \propto \underset{y}{\text{argmax}}\, P(x|y)P(y) \tag{2}$$

(because the denominator will be the same for all classes.)

Naive Bayes Model

- Let $\Theta(x)$ be a set of features such that $\Theta(x) = \theta_1(x), \theta_2(x), ..., \theta_n(x)$.
- $P(x|y) = P(\theta_1(x), \theta_2(x), ..., \theta_n(x)|y)$.
- We use the naive bayes assumption of conditional independence: $P(\theta_1(x), \theta_2(x), ..., \theta_n(x)|y) = \prod_i P(\theta_i(x)|y)$
- $P(x|y)P(y) = P(y)\prod_i P(\theta_i(x)|y)$
- We want to find the maximum value of this expression, given all possible different y.

Relation to Maximum Likelihood Estimates (MLE)

- Let's define the likelihood function $\mathcal{L}(\Theta\,;y)$.
- MLE finds the values of Θ that maximize $\mathcal{L}(\Theta\,;y)$ (i.e. that make the data most probable).
- In our case, we simply estimate each $\theta_i(x)|y \in \Theta$ from the training data:
  $P(\theta_i(x)|y) = \frac{count(\theta_i(x),y)}{\sum_{\theta(x)\in\Theta} count(\theta(x),y)}$.

Naive Bayes Example

- Let's say your mailbox is organised as follows:
  - Work
    - Eva
    - Angeliki
    - Abhijeet
  - Friends
    - Tim
    - Jim
    - Kim
- You want to automatically file new emails according to their topic (work or friends).

Document classification

- Classify document into one of two classes: *work* or *friends*. $y = [0, 1]$, where 0 is for *work* and 1 is for *friends*.
- Use words as features (under the assumption that the meaning of the words will be indicative of the meaning of the documents, and thus its topic).
  $\theta_i(x) = w_i$
- We have one feature per word in our vocabulary V (the 'vocabulary' being the set of unique words in all texts encountered in training).

Some training emails

- E1: “Shall we go climbing at the weekend?”
  friends
- E2: “The composition function can be seen as one-shot learning.”
  work
- E3: “We have to finish the code at the weekend.”
  work

- $V = \{$ shall we go climbing at the weekend ? composition function can be seen as one-shot learning . have to finish code $\}$

Some training emails

- E1: “Shall we go climbing at the weekend?”
  friends
- E2: “The composition function can be seen as one-shot learning.”
  work
- E3: “We have to finish the code at the weekend.”
  work

- $\Theta(x) = \{$ shall we go climbing at the weekend ?
  composition function can be seen as one-shot learning .
  have to finish code $\}$

Some training emails

- E1: “Shall we go climbing at the weekend?”
  friends
- E2: “The composition function can be seen as one-shot learning.”
  work
- E3: “We have to finish the code at the weekend.”
  work

- Let’s now calculate the probability of each $\theta_i(x)$ given a class.
- $P(\theta_i(x)|y) = \frac{count(\theta_i(x),y)}{\sum_{\theta(x)\in\Theta} count(\theta(x),y)}$

Some training emails

- E1: “Shall we go climbing at the weekend?”
  friends
- E2: “The composition function can be seen as one-shot learning.”
  work
- E3: “We have to finish the code at the weekend.”
  work

- Let’s now calculate the probability of each $\theta_i(x)$ given a class.
- $P(we|y=0) = \frac{count(we,y=0)}{\sum_{w \in V} count(w,y=0)} = \frac{1}{20}$

Some training emails

- E1: “Shall we go climbing at the weekend?”
  friends
- E2: “The composition function can be seen as one-shot learning.”
  work
- E3: “We have to finish the code at the weekend.”
  work

- $P(\Theta(x)|y = 0) = \{$ (shall,0) (we,0.05) (go,0) (climbing,0) (at,0.05) (the,0.15) (weekend,0.05) (?,0) (composition,0.05) (function,0.05) (can,0.05) (be,0.05) (seen,0.05) (as,0.05) (one-shot,0.05) (learning,0.05) (.,0.05) (have,0.05) (to,0.05) (finish,0.05) (code,0.05) $\}$

Some training emails

- E1: “Shall we go climbing at the weekend?”
  friends
- E2: “The composition function can be seen as one-shot learning.”
  work
- E3: “We have to finish the code at the weekend.”
  work

- $P(\Theta(x)|y = 1) = \{$ (shall,0.125) (we,0.125) (go,0.125) (climbing,0.125) (at,0.125) (the,0.125) (weekend,0.125) (?,0.125) (composition,0) (function,0) (can,0) (be,0) (seen,0) (as,0) (one-shot,0) (learning,0) (.,0) (have,0) (to,0) (finish,0) (code,0) $\}$

Prior class probabilities

- $P(0) = \frac{f(doctopic{=}0)}{f(alldocs)} = \frac{2}{3} = 0.66$
- $P(1) = \frac{f(doctopic{=}1)}{f(alldocs)} = \frac{1}{3} = 0.33$

A new email

- E4: “When shall we finish the composition code?”

- We ignore unknown words: (*when*).
- $V = \{$ shall we finish the composition code ? $\}$
- We want to solve:

$$\underset{y}{\text{argmax}}\, P(y|\Theta(x)) \propto \underset{y}{\text{argmax}}\, P(\Theta(x)|y)P(y) \qquad (3)$$

Testing $y = 0$

$$
\begin{aligned}
& P(\Theta(x)|y) \\
= \quad & P(\textit{shall}|y=0) * P(\textit{we}|y=0) * P(\textit{finish}|y=0) * \\
& P(\textit{the}|y=0) * P(\textit{composition}|y=0) * \\
& P(\textit{code}|y=0) * P(?|y=0) \\
= \quad & 0 * 0.05 * 0.05 * 0.15 * 0.05 * 0.05 * 0 \\
= \quad & 0
\end{aligned}
$$

Oops.....

Smoothing

- When something has probability 0, we don't know whether that is because the probability is *really* 0, or whether the training data was simply 'incomplete'.
- Smoothing: we add some tiny probability to unseen events, just in case...
- Additive/Laplacian smoothing:

$$P(e) = \frac{f(e)}{\sum_{e'} f(e')} \rightarrow P(e) = \frac{f(e) + \alpha}{\sum_{e'} (f(e') + \alpha)} \tag{4}$$

Recalculating training probabilities...

- E1: “Shall we go climbing at the weekend?”
  friends
- E2: “The composition function can be seen as one-shot learning.”
  work
- E3: “We have to finish the code at the weekend.”
  work

- **Examples:**
  - $P(the|y=0) = \frac{3+0.01}{20*1.01} \approx 0.15$
  - $P(climbing|y=0) = \frac{0+0.01}{20*1.01} \approx 0.0005$

Testing $y = 0$ (*work*)

$$
\begin{aligned}
& P(\Theta(x)|y) \\
= \quad & P(shall|y=0) * P(we|y=0) * P(finish|y=0) * \\
& P(the|y=0) * P(composition|y=0) * \\
& P(code|y=0) * P(?|y=0) \\
= \quad & 0.0005 * 0.05 * 0.05 * 0.15 * 0.05 * 0.05 * 0.0005 \\
= \quad & 2.34 * 10^{-13}
\end{aligned}
$$

$$
\begin{aligned}
& P(\Theta(x)|y)P(y) \\
= \quad & 2.34 * 10^{-13} * 0.66 \\
= \quad & 1.55 * 10^{-13}
\end{aligned}
$$

Testing $y = 1$ (*friends*)

$$
\begin{aligned}
& P(\Theta(x)|y) \\
= \quad & P(shall|y=1) * P(we|y=1) * P(finish|y=1) * \\
& P(the|y=1) * P(composition|y=1) * \\
& P(code|y=1) * P(?|y=1) \\
= \quad & 0.13 * 0.13 * 0.0012 * 0.13 * 0.0012 * 0.0012 * 0.13 \\
= \quad & 4.94 * 10^{-13}
\end{aligned}
$$

$$
\begin{aligned}
& P(\Theta(x)|y)P(y) \\
= \quad & 4.94 * 10^{-13} * 0.33 \\
= \quad & 1.63 * 10^{-13}
\end{aligned}
$$

Using log in implementations

- In practice, it is useful to use the log of the probability function, converting our product into a sum.
- $\log_b(ij) = \log_b i + \log_b j$

$$
\begin{aligned}
& log(P(\Theta(x)|y)) \\
= \quad & log(P(shall|y=1) * P(we|y=1) * \\
& P(finish|y=1) * P(the|y=1) * \\
& P(composition|y=1) * \\
& P(code|y=1) * P(?|y=1)) \\
= \quad & log(0.13) + log(0.13) + log(0.0012) + log(0.13) \\
& + log(0.0012) + log(0.0012) + log(0.13) \\
= \quad & -12.31
\end{aligned}
$$

The issue of feature selection

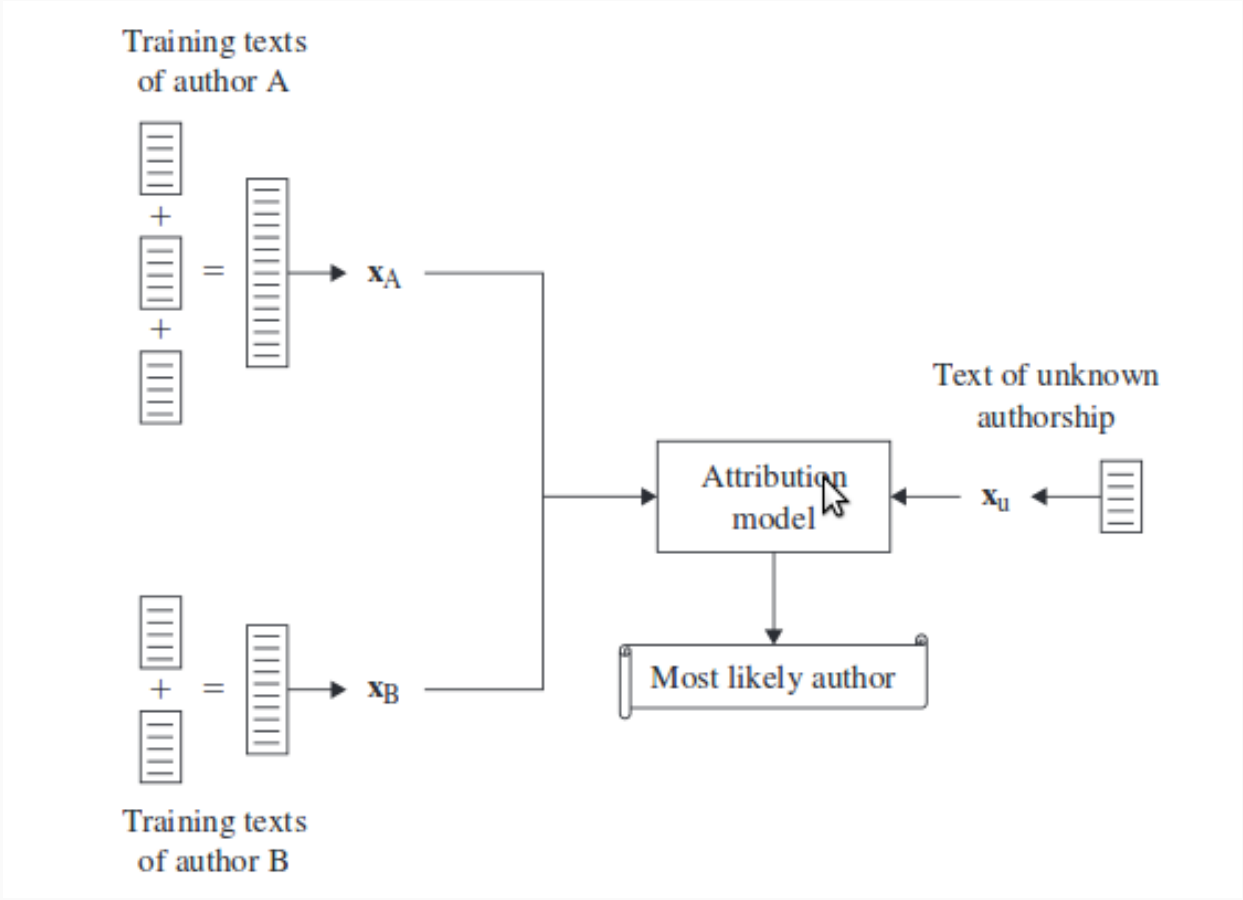
Authorship attribution

- Your mailbox is organised as follows:
  - Work
    - Eva
    - Angeliki
    - Abhijeet
  - Friends
    - Tim
    - Jim
    - Kim
- How different are the emails from Eva and Abhijeet? From Tim and Jim?

Authorship attribution

- The task of deciding *who* has written a particular text.
- Useful for historical, literature research. (Are those letters from Van Gogh?)
- Used in forensic linguistics.
- Interesting from the point of view of feature selection.

Basic architecture of authorship attribution



From Stamatatos (2009). *A Survey of Modern Authorship Attribution Methods.*

Choosing features

- Which features might be useful in authorship attribution?
  - Stylistic: does the person tend to use lots of adverbs? To hedge their statements with modals?
  - Lexical: *what* does the person talk about?
  - Syntactic: does the person prefer certain syntactic patterns to others?
  - Other: does the person write smileys with a nose or without? :-) :)

Stylistic features

- The oldest types of features for authorship attribution (Mendenhall, 1887).
- Word length, sentence length... (Are you pompous? Complicated?)
- Vocabulary richness (type/token ratio). But: dependent on text length. The size of vocabulary increases rapidly at the beginning of a text and then decreases.

Lexical features

- The most widely used feature in authorship attribution.
- A text is represented as a vector of word frequencies.
- This is then only a rough topical representation which disregard word order.
- N-grams combine the best of all words, encoding order and some lexical information.

Syntactic features

- Syntax is used largely unconsciously and is thus a good indicator of authorship.
- An author might keep using the same patterns (e.g. prefer passive forms to active ones).
- But producing good features relies on having a good parser...
- Partial solution: use shallow syntactic features, e.g. sequences of POS tags (DT JJ NN).

The case of emoticons

- Which ones are used? :-) :D :P ^_^
  - Indication of geographical provenance.
- How are they written? :-) or :)
  - Indication of age.
- Miscellaneous: how do you put a smiley at the end of a parenthesis?
  a) (cool! :)) b) (cool! :) c) (cool! :)) ...

Simple is best

- The best features for authorship attribution are often the simplest.
- Use of function words (prepositions, articles, punctuation) is usually more revealing than content words. They are mostly used unconsciously by authors.
- *Character N-grams* are a powerful and simple technique:
  - unigrams: n, -, g, r, a, m
  - bigrams: n-, -g, gr, ra, am, ms
  - trigrams: n-g, -gr, gra, ram, ams

N-grams

- N-grams which is both robust to noise and captures various types of information, including:
  - frequency of various prepositions (_*in*_, *for*_);
  - use of punctuation (*;_an*);
  - abbreviations (*e_&_*);
  - even lexical features (*type*, *text*, *ment*).

Ablation

- Which features are best for my task?
- A good way to find out is to perform an *ablation*.
- We train the system with *all* features and then remove each one individually and re-train. Does the performance of the system goes up or down?

Ablation: example

Features used	Precision
n-grams + syntax + word length + sentence length	0.70
- n-grams	0.55
- syntax	0.72
- word length	0.65
- sentence length	0.68

Tomorrow's practical

- Let's download texts from various authors and train a Naive Bayes system on those texts.
- Can we correctly identify the author's identity for an unknown text?
- Which features worked best? Can we think of other ones?

Evaluation

What is the task?

- Let's define our task precisely (not always easy):
  - Give each word in a text a part of speech.
  - Reproduce human similarity judgements for word pairs.
  - Translate a text from Hindi to German.
  - Chat with a human in an acceptable way.

Evaluation data

- Usually, we will have a *gold standard* for our task. It could be:
  - Raw data (see the language modelling task: we have some sentences and we want to predict the next word).
  - Some data annotated by experts (e.g. text annotated with parts of speech by linguists).
  - Some data annotated by volunteers (e.g. crowdsourced similarity judgments for word pairs).
  - Parallel corpora: translations of the same content in various languages.
- We may also evaluate by collecting human judgments on the output of the system (e.g. quality of chat, 'beauty' of an automatically generated poem, etc).

Splitting the evaluation data

- A typical ML pipeline involves a *training* phase (where the system learns) and a *testing* phase (where the system is tested).
- We need to split our gold standard to ensure that the system is tested on *unseen* data. Why?
  - We don't want the system to just memorise things.
  - We want it to be able to *generalise* what it has learnt to new cases.
- We split the data between training, (development), and test sets. A usual split might be 70%, 20%, 10% of the data.

Development set?

- A development set may or may not be used.
- We use it *during development*, where we need to test different configurations or feature representations for the system.
- For example:
  - We train a word-based authorship classification algorithm. It doesn't do so well on the dev test.
  - We decide to try another kind of features, which include syntactic information. We re-test on the dev test and get better results.
  - Finally, we check that indeed those features are the 'best' ones by testing the system on completely unseen data (the test set).

Precision and recall

	Predicted +	Predicted -
Actual +	TP	FN
Actual -	FP	TN

- Precision: $\frac{TP}{TP+FP}$
- Recall: $\frac{TP}{TP+FN}$

Precision and recall: example

- We have a collection of 50 novels by several authors, and we want to retrieve all 6 Jane Austen novels in that collection.
- We set two classes, A and B, where class A is the class of Austen novels and B is the class of books by other authors.
- Let's assume our system gives us the following results:

	Predicted A	Predicted B	Sum
Gold A	4	2	6
Gold B	10	34	44
Sum	14	36	50

- Precision: $\frac{4}{14} = 0.29$
- Recall: $\frac{4}{6} = 0.67$

F-score

- Often, we want to have a system that performs well both in terms of precision and recall:

  F_1 score: $2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$

- The F-score formula can be weighted to give more or less weight to either precision or recall:

  $F_\beta = (1 + \beta^2) \cdot \frac{\text{precision} \cdot \text{recall}}{\beta^2 \cdot \text{precision} + \text{recall}}$

F-score: example

- Let's try different weights for β on our book example:

	Predicted A	Predicted B	Sum
Gold A	4	2	6
Gold B	10	34	44
Sum	14	36	50

- $F_1 = (1 + 1^2) \cdot \frac{0.29 \cdot 0.67}{1^2 \cdot 0.29 + 0.67} = 0.40$
- $F_2 = (1 + 2^2) \cdot \frac{0.29 \cdot 0.67}{2^2 \cdot 0.29 + 0.67} = 0.53$
- $F_{0.5} = (1 + 0.5^2) \cdot \frac{0.29 \cdot 0.67}{0.5^2 \cdot 0.29 + 0.67} = 0.33$

Accuracy

- Accuracy is used when we care about true negatives. (How important is it to us that books that were *not* by Jane Austen were correctly classified?)
-

	Predicted +	Predicted -
Actual +	TP	FN
Actual -	FP	TN

- Accuracy: $\frac{TP+TN}{TP+FN+FP+TN}$

Accuracy: example

	Predicted A	Predicted B	Sum
Gold A	4	2	6
Gold B	10	34	44
Sum	14	36	50

- Accuracy: $\frac{38}{50} = 0.76$

Imbalanced data

- Note how our Jane Austen classifier get high accuracy whilst being, in fact, not so good.
- Accuracy is not such a good measure when the data is imbalanced.
- Only 6 out of 50 books are by Jane Austen. A (dumb) classifier that always predicts a book to be by another author would have $\frac{44}{50} = 0.88$ accuracy.

Baselines

- To know how well we are doing with the classification, it is important to have a point of comparison for our results.
- A *baseline* can be:
  - A simple system that tells us how hard our task is, *with respect to a particular measure*.
  - A previous system that we want to improve on.
- Note: a classifier that always predicts a book to be by another author than Jane Austen will have $\frac{44}{50} = 0.88$ accuracy and $\frac{0}{6} = 0$ precision. Which measure should we report?

Multiclass evaluation

- How to calculate precision/recall in the case of a multiclass problem (for instance, authorship attribution across 4 different authors).
- Calculate precision e.g. for class A by collapsing all other classes together.
-

	Predicted A	Predicted B	Predicted C
Actual A	TA	FB	FC
Actual B	FA	TB	FC
Actual C	FA	FB	TC

Multiclass evaluation

- How to calculate precision/recall in the case of a multiclass problem (for instance, authorship attribution across 4 different authors).
- Calculate precision e.g. for class A by collapsing all other classes together.
-

	Predicted A	Predicted $\overline{A}$
Actual A	TA = TA	F$\overline{A}$ = FB+FC
Actual $\overline{A}$	F$\overline{A}$ = FA_B+FA_C	T$\overline{A}$=TB+TC

Tomorrow:
bring your laptops!

For those without python on their machine:
sign up for an account on
`https://www.pythonanywhere.com/`

(Everybody got wifi?)