

Machine Learning for NLP

Statistical NLP

Aurélie Herbelot

2021

Centre for Mind/Brain Sciences
University of Trento

Probabilities and language

Probability of an event

- Most introductions to probabilities start with coin and dice examples:
 - The probability $P(H)$ of a fair coin falling heads is 0.5.
 - The probability $P(2)$ of rolling a 2 with a fair six-sided die is $\frac{1}{6}$.
- $P(A)$: the frequency of event A , relative to all other possible events, given an experiment repeated an *infinite* number of times.

Probability of a word

The probability $P(\text{the})$ of a speaker uttering *the* is...?



<https://www.pinterest.com/pin/353180795749152830/>

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).
- So probabilities in language are always approximations, reflecting the fact that a speaker's language is a) dynamic; b) incomplete.

On incompleteness

Let's assume two omniscient speakers, Vishnu and Artemis, who have a 'perfect language' (a complete dice): they know what there is in the world, and how to name it.

They share the same model M , including interpretation function $\|\cdot\|$, and the same grammar G .

$$\|oak\|_{Vishnu} = \|oak\|_{Artemis} = \|oak\|$$

(the true denotation of *oak*).

They are perfectly aligned. They can understand each other perfectly.

On incompleteness



Vishnu: All oaks are trees

Vishnu: Zeus had 37 strawberries for breakfast.

Vishnu (lying): You know something I don't.

Vishnu: Whatever...



Artemis: True.

Artemis: True.

Artemis: False.

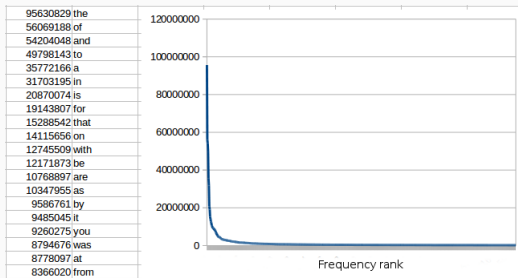
If we were dice with a fixed number of faces encoding our linguistic knowledge, we wouldn't need to speak to each other.

Using corpora

- 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.
- A corpus is finite and incomplete.

Zipf Law

- From corpora, we can get some general idea of the likelihood of a word by observing its frequency in a large corpus. Across languages, we find that words have a Zipfian distribution.



But what for?

At a theoretical level, questions about lexical predictability factor into a broader trend in cognitive science that casts prediction as a core explanatory principle of information processing (Clark, 2013). Under such a predictive processing framework, the brain uses relevant contextual knowledge to preactivate features of an upcoming stimulus or event before it is perceived, which leads to processing facilitation when the perception matches the prediction, or error-driven learning when the two do not match. Indeed, this view has

Lowder et al (2018)

Language Modelling

- A language model (LM) is a model that computes the probability of a sequence of words.
- It can be used for:
 - language interpretation / understanding;
 - language generation.
- Today, we will look at the simplest form of probabilistic language modelling. Later in the course, we will see neural variants of the ideas covered in this lecture.

A unigram language model

- 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 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 a o o o o.

What might it say next? How confident are you in your answer?

A unigram language model

- Remember, computing a probability assumes an experiment repeated an *infinite* number of times.
- The estimated probabilities are approximations:
 - $o a a$:
 $P(a) = \frac{2}{3}$ with low confidence.
 - $o a a o o o o o o o o o o o o o o a o o o o$:
 $P(a) = \frac{3}{22}$ 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*
- **Are those in any sense language-like?**

Conditional probability and bigram language models

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

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

Chain rule: given all the times I have B, how many times do I have A too?



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.

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

Chain rule: given all the times I have B, how many times do I have A too?



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.

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

Chain rule: given all the times I have B, how many times do I have A too?



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

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

Conditional probability and bigram language models

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

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

Chain rule: given all the times I have B, how many times do I have A too?



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

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

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*

What about context?

- Why are those sentences so weird?
- We are estimating the probability of a word without taking into account the broader context of the sentence.
- Let's assume the following sentence:
The robot is talkative.
- We are computing $P(\textit{talkative}|\textit{is})$ but ideally, we would want $P(\textit{talkative}|\textit{the, robot, is})$.

In the following, I will write $P(\textit{talkative}|\textit{is, robot, the})$ to emphasise that we are predicting from the end of the sentence, going back in time.

The Markov assumption

- We can use the chain rule for calculating the probability of a sentence:

$$P(A_n, \dots, A_1) = P(A_n | A_{n-1}, \dots, A_1) \cdot P(A_{n-1}, \dots, A_1)$$

- $P(A_{n-1}, \dots, A_1)$ is $P(A_{n-1} | A_{n-2}, \dots, A_1) \cdot P(A_{n-2}, \dots, A_1)$
- $P(A_{n-2}, \dots, A_1)$ is $P(A_{n-2} | A_{n-3}, \dots, A_1) \cdot P(A_{n-3}, \dots, A_1)$
- etc...

- So for our example:

$$P(\textit{talkative}, \textit{is}, \textit{robot}, \textit{the}) = P(\textit{talkative} | \textit{is}, \textit{robot}, \textit{the}) \cdot P(\textit{is} | \textit{robot}, \textit{the}) \cdot P(\textit{robot} | \textit{the}) \cdot P(\textit{the})$$

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:

$P(\textit{talkative} \mid \textit{is, robot, the, and, today, sunny, is, it})$

- So we make a simplifying **Markov assumption**:

$P(\textit{talkative} \mid \textit{is, robot, the}) \approx P(\textit{talkative} \mid \textit{is})$ (bigram)

or

$P(\textit{talkative} \mid \textit{is, robot, the}) \approx P(\textit{talkative} \mid \textit{is, robot})$ (trigram)

The Markov assumption

- Coming back to our example:

$$P(\textit{the, robot, is, talkative}) = P(\textit{talkative} \mid \textit{is, robot, the}) \cdot P(\textit{is} \mid \textit{robot, the}) \cdot P(\textit{robot} \mid \textit{the}) \cdot P(\textit{the})$$

- A bigram model simplifies this to:

$$P(\textit{the, robot, is, talkative}) = P(\textit{talkative} \mid \textit{is}) \cdot P(\textit{is} \mid \textit{robot}) \cdot P(\textit{robot} \mid \textit{the}) \cdot P(\textit{the})$$

- 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 \mathbf{y} given input \mathbf{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)
 - $x = \{ \dots \text{the, worst, action, film, } \dots \}$
- We want to evaluate $P(y|x)$ for each y and find $\operatorname{argmax}_y P(y|x)$ (the class with the highest probability).

Bayes' Rule

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

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

- Finding the argmax means using the following equivalence (\propto = proportional to):

$$\operatorname{argmax}_y P(y|x) \propto \operatorname{argmax}_y P(x|y)P(y)$$

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

Naive Bayes Model

- Let $\Theta(x)$ be a set of features such that

$$\Theta(x) = \theta_1(x), \theta_2(x), \dots, \theta_n(x)$$

($\theta_1(x)$ = feature 1 of input data x .)

- $P(x|y) = P(\theta_1(x), \theta_2(x), \dots, \theta_n(x)|y)$.

We are *expressing* x in terms of the thetas.

- We use the naive bayes assumption of conditional independence:

$$P(\theta_1(x), \theta_2(x), \dots, \theta_n(x)|y) = \prod_i P(\theta_i(x)|y)$$

(Let's do as if $\theta_1(x)$ didn't have anything to do with $\theta_2(x)$.)

- $P(x|y)P(y) = (\prod_i P(\theta_i(x)|y))P(y)$
- 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 given a class).
- In our case, we simply estimate each $\theta_i(x)|y$ from the training data:

$$P(\theta_i(x)|y) = \frac{\text{count}(\theta_i(x),y)}{\sum_{\theta(x) \in \Theta} \text{count}(\theta(x),y)}$$

- (Lots of squiggles to say that we're counting the number of times a particular feature occurs in a particular class.)

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

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.
- $$P(\theta_i(x)|y) = \frac{\text{count}(\theta_i(x),y)}{\sum_{\theta(x) \in \Theta} \text{count}(\theta(x),y)}$$

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.
- $$P(we|y = 0) = \frac{\text{count}(we, y=0)}{\sum_{w \in V} \text{count}(w, y=0)} = \frac{1}{20}$$

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

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

Prior class probabilities

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

A new email

- E4: “When shall we finish the composition code?”
- We ignore unknown words: (*when*).
- $V = \{ \text{shall we finish the composition code ?} \}$
- We want to solve:

$$\operatorname{argmax}_y P(y|\Theta(x)) \propto \operatorname{argmax}_y P(\Theta(x)|y)P(y)$$

Testing $y = 0$

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

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:

$$P(e) = \frac{c(e)}{\sum_{e'} c(e')} \rightarrow P(e) = \frac{c(e) + \alpha}{\sum_{e'} (c(e') + \alpha)}$$

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

Testing $y = 0$ (work)

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

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

Testing $y = 1$ (*friends*)

$$P(\Theta(x)|y)$$

$$\begin{aligned} &= P(\text{shall}|y = 1) * P(\text{we}|y = 1) * P(\text{finish}|y = 1) * \\ &P(\text{the}|y = 1) * P(\text{composition}|y = 1) * \\ &P(\text{code}|y = 1) * P(?|y = 1) \end{aligned}$$

$$= 0.13 * 0.13 * 0.0012 * 0.13 * 0.0012 * 0.0012 * 0.13$$

$$= 4.94 * 10^{-13}$$

$$P(\Theta(x)|y)P(y)$$

$$= 4.94 * 10^{-13} * 0.33$$

$$= \mathbf{1.63 * 10^{-13}}$$

:(

Using log in implementations

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

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

Avoid *underflow* problems: rounding very small numbers to 0. Also addition faster than multiplication in many architectures.

Evaluation

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 set.
 - We decide to try another kind of features, which include syntactic information. We re-test on the dev set 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).

Evaluating our Language Model: perplexity

- A better LM is one that gives higher probability to ‘the word that actually comes next’ in the test data.
- Examples:
 - For my birthday, I got a purple | parrot | bicycle | theory... .
 - Did you go crazy | elephant | fluffy | to...
 - I saw a shopping | cat | building | red...
- More uncertainty = more *perplexity*. So low perplexity is good!

Evaluating our Language Model: perplexity

- Given a sentence $S = w_1 w_2 \dots w_N$, perplexity is defined as:

$$PP(S) = P(S)^{-\frac{1}{N}} = P(w_1 w_2 \dots w_N)^{-\frac{1}{N}}$$

- For a unigram model:

$$PP(S) = [P(w_1) \times P(w_2) \dots \times P(w_N)]^{-\frac{1}{N}}$$

- Example:
 - Three words $w_1 \dots w_3$ with probabilities 0.8, 0.19, 0.01.
 - $PP(w_3 w_2 w_1) = [0.01 \times 0.19 \times 0.8]^{-\frac{1}{3}} \approx 8.7$

Evaluating our Language Model: perplexity

- Given a sentence $S = w_1 w_2 \dots w_N$, perplexity is defined as:

$$PP(S) = P(S)^{-\frac{1}{N}} = P(w_1 w_2 \dots w_N)^{-\frac{1}{N}}$$

- For a unigram model:

$$PP(S) = [P(w_1) \times P(w_2) \dots \times P(w_N)]^{-\frac{1}{N}}$$

- Example:
 - Three words $w_{1\dots 3}$ with probabilities 0.8, 0.19, 0.01.
 - $PP(w_1 w_2 w_3) = [0.8 \times 0.19 \times 0.19]^{-\frac{1}{3}} \approx 3.3$

Classification: Precision and recall

- Let's assume the table below shows the true positives (TP), true negatives (TN), false positives (FP) and false negatives (FN) for a spam detection algorithm.
- We want to calculate the precision and recall of the spam detection.

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

- **Precision** (how many emails tagged as spam *are* spam): $\frac{TP}{TP+FP}$
- **Recall** (how many of the spam emails did we catch?): $\frac{TP}{TP+FN}$

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: $\frac{4}{6} = 0.67$

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

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

- 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

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

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.

- 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.
- 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. Those are *baseline* figures that we will want to improve on.

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 \bar{A}
Actual A	$T_A = T_A$	$F_{\bar{A}} = F_B + F_C$
Actual \bar{A}	$F_A = F_{A_B} + F_{A_C}$	$T_{\bar{A}} = T_B + T_C$