FNLP - Week 3: More Smoothing, Spelling Correction, Dynamic Programming & Naive Bayes

Antonio León Villares

Feburary 2022

Contents

For **edit distance**, I recommend checking my notes for IADS here.

1 Further Smoothing

1.1 Recap: Smoothing

- **How does sparsity affect n-gram models?**
  - n-grams assume a **fixed length history** to reduce the effect of sparsity
  - not enough: unseen items $\implies$ 0 counts $\implies$ n-gram assumes items can **never** exist
- **How does smoothing resolve the sparsity problem?**
  - transfer **probability mass** from seen elements to **unseen** ones
- **How do Laplace/Lindstone smoothing work?**
  - **halluccinate** $\alpha \leq 1$ observations for each word in vocabulary
  - transfers too much probability mass, reducing the effect of high frequency counts
- **How does Good-Turing improve the distribution of probability mass?**
  - the halluccinated counts are more adaptive:

    $$c^* = (c+1)\frac{N_{c+1}}{N_c}$$

    where N_c is the number of n-grams observed c times
  - the probability becomes:

    $$P_{GT} = \frac{c^*}{n}$$

    For a trigram "I spent three":
    - n: counts of "I spent"
    - c: count of "I spent three"
    - c^*: adjusted count of "I spent three"
- **Is Good-Turing infallible?**
  - still has problems:
    - unknown vocabulary size
    - "holes" in higher frequency counts
    - discounts high frequencies
- **What is a fundamental problem in all smoothings so far?**
  - unknown n-grams which are **smoothed** will **all** receive the **same probability**, regardless of the n-gram:

    $$P(\textit{drinkers} \mid \textit{Scottish beer}) = P(\textit{eaters} \mid \textit{Scottish beer})$$

  - the former should naturally have higher probability, but smoothing doesn't know this
  - to solve this, we can **incorporate** information from **lower order** n-grams (so use our knowledge of "beer drinkers" and "beer eaters")

1.2 Interpolation

- **How does interpolation use lower-order n-grams?**
  - use a **linear combination** of all lower-order n-grams:

$$\hat{P}(w_n \mid w_{n-2},\ w_{n-1}) = \lambda_1 P(w_n) + \lambda_2 P(w_n \mid w_{n-1}) + \lambda_3 P(w_n \mid w_{n-2},\ w_{n-1})$$

$$\hat{P}(three \mid I,\ spent) = \lambda_1 P(three) + \lambda_2 P(three \mid spent) + \lambda_3 P(three \mid I,\ spent)$$

  - **low-order** n-grams have **robust counts** (but **limited context**)
  - **high-order** n-grams have better **context**, but **sparse counts**
  - **interpolation** is a way to get the best of both worlds
- **How do we determine the λ_i?**
  - these are known as **interpolarion parameters** or **mixture weights** (since they defined a **mixture model** - a distribution composed of a combination of distributions)
  - we pick λ_i such that they **maximise likelihood** (or **minimise perplexity**) in a **held-out** corpus (done using **expectation maximisation**)
  - we must have that:

$$\sum_i \lambda_i = 1$$

$$\begin{aligned}
1 &= \sum_{w_3} P_{\text{INT}}(w_3|w_1, w_2) \\
&= \sum_{w_3} [\lambda_1\ P_1(w_3) + \lambda_2\ P_2(w_3|w_2) + \lambda_3\ P_3(w_3|w_1, w_2)] \\
&= \lambda_1 \sum_{w_3} P_1(w_3) + \lambda_2 \sum_{w_3} P_2(w_3|w_2) + \lambda_3 \sum_{w_3} P_3(w_3|w_1, w_2) \\
&= \lambda_1 + \lambda_2 + \lambda_3
\end{aligned}$$

- **How can we pick more sophisticated interpolation weights?**
  - define the **weights** by **conditioning** them on **context** (i.e high frequency context $\Longrightarrow$ larger weights)

1.3 Back-Off Smoothing

- **How does back-off use lower-order n-grams?**
  - defines the probability of an n-gram using the **largest** order n-gram with **non-zero evidence**
  - for example, if "I spent three years" never appears, consider "spent three years"; if this never appears, consider "three years"; if this never appears, consider "years"
- **Does this define a probability distribution?**
  - if we add different order n-gram probabilities, the result won't be 1
    - think of it: if a trigram has probability zero, changing if for a non-zero bigram will increase the overall probability mass

  - as with **Good-Turing**, need to **discount** probability mass from higher-order n-grams

- **What is Katz-Backoff?**
  - the way of converting the **back-off** into a true distribution:

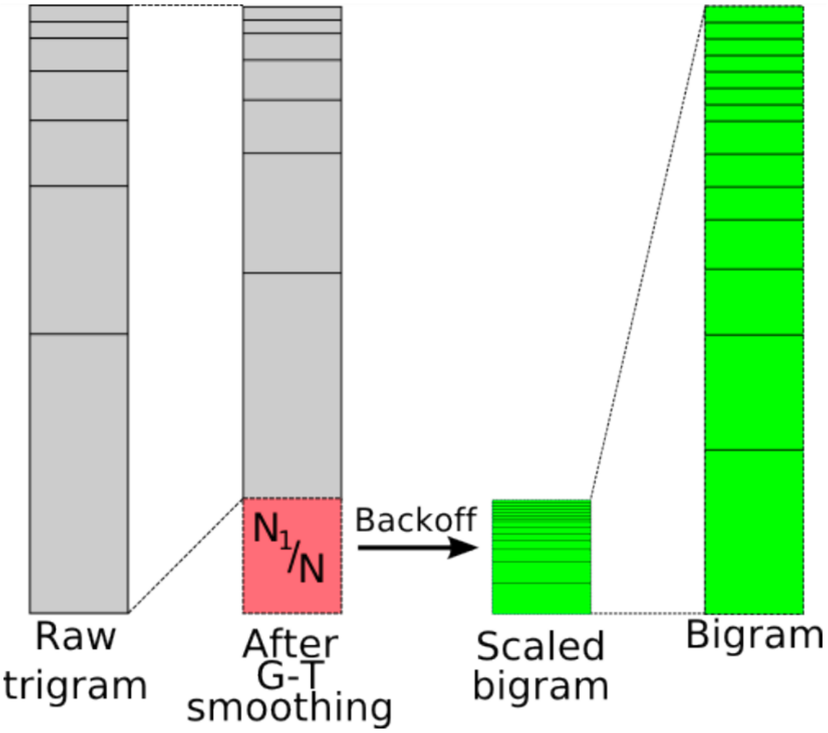


Figure 1: The first bar represents the raw trigram counts. When applying Good-Turing, the counts are proportionally squashed, to leave a probability mass of $\frac{N_1}{N}$ for the bigrams. What Katz back-off does is take the bigram counts (green column), and scale the probabilities to fit in the red square of probability mass. In this way, we ensure that the probability mass across trigrams is preserved.

  - mathematically:

$$P_{BO}(w_i|w_{i-N+1},...,w_{i-1}) =$$
$$= \begin{cases} P^*(w_i|w_{i-N+1},...,w_{i-1}) \\ \qquad\qquad \text{if count}(w_{i-N+1},...,w_i) > 0 \\ \alpha(w_{i-N+1},...,w_{i-1})\ P_{BO}(w_i|w_{i-N+2},...,w_{i-1}) \\ \qquad\qquad \text{else} \end{cases}$$

where:

  - P_{BO} is the Katz Back-Off probability for the n-gram
  - P^* is a **discounted probability** for a seen n-gram
  - α is a **back-off weight** used to distribute the probability mass of higher n-grams across lower n-grams

1.4 Kneser-Ney Smoothing

- **What does Kneser-Ney Smoothing focus on?**
  - how certain **frequent** words only appear in limited context (low **diversity of history**)
  - for example, “York” can have high frequency in a text, but it will typically occur after “New”
  - the probability of “York” appearing after any other word should be lower than the probability of “York” appearing after “New”
- **How does Kneser-Ney deal with diversity of history?**
  - we consider the **number of different histories of a word** (i.e the number of unique words which occur before a given word w_i):

$$N_{1+}(\circ\ w_i) = \|\{w_{i-1}\ | c(w_{i-1}, w_i) > 0\}\|$$

  - instead of using an MLE estimate, use **count histories** to get word probabilities:

$$P_{KN}(w_i) = \frac{N_{1+}(\circ\ w_i)}{\sum_w N_{1+}(\circ\ w)}$$

- **How is Kneser-Ney used in practice?**
  - we use **interpolated Kneser-Ney**:

$$P_{KN}(w_i \mid w_{i-1}) = \frac{\max(c(w_{i-1}, w_i) - d, 0)}{c(w_{i-1})} + \lambda(w_{i-1}) P_{KN}(w_i)$$

    where d is a discounting constant, and λ is a **normalising constant** use dto distribute probability mass:

$$\lambda(w_{i-1}) = \frac{d}{\sum_v c(w_{i-1v})} \times \|\{w \mid c(w_{i-1},\ w) > 0\}\|$$

  - in general, a recursive formula is used for this, for more than bigrams

1.5 Distributed Representations

- **What are distributed language models?**
  - use of **neural networks** to project words into a **continuous space**
  - in this way, words can be represented by **vectors**, such that similar words are mapped to vectors which are **close**
  - these vectors are known as **embeddings**
  - useful: if we know $P(salmon \mid caught\ two)$, we can attempt to derive $P(swordfish \mid caught\ two)$ (n-grams can’t do this!)

2 The Noisy Channel Model & Spelling Correction

2.1 Defining the Noisy Channel Model

- **What is the noisy channel model?**

- we consider a **system** in which an **input sequence** is polluted with **noise**
- we only get the **output** after the **noise** has been added
- a **noisy channel model** is a way of **modelling** the way in which noise is added to the input sequence

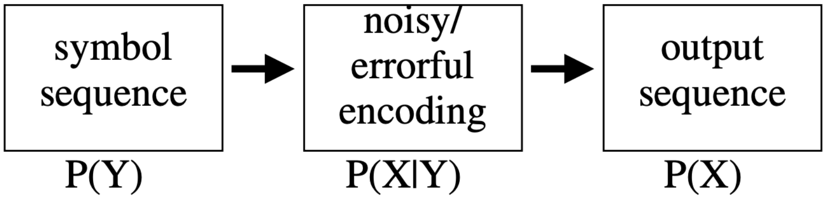


Figure 2: For example, in **speech recognition** the input Y can be a sequence of **spoken words**, and the output X will be an **acoustic signal**. We want to get the **noisy channel model** which tells us the probability $P(X|Y)$ of obtaining output X, given input Y.

- **How does the noisy channel model apply to spelling errors?**
  - we can think of it in terms of **spelling correction**:
    1. $P(Y)$ is a **language model**: the distribution of words intended by the user
    2. $P(X|Y)$ is the **noise model**: what user is **likely** to type, given their **intention**
    3. $P(X)$: the distribution of what the user sees
  - if we want to correct spelling, we can (potentially) pass every word through the model, and see which one comes closest to the misspelled word
- **How can we describe the noisy channel model mathematically?**
  - we want a model which maximises $P(x|y)$; in other words, given an output x, we want to pick the y with the largest probability:

$$\hat{y} = \underset{y \in V}{argmax} P(y|x)$$

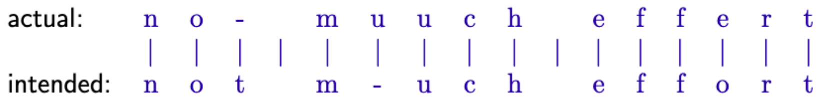
  - using **Bayes**:

$$\hat{y} = \underset{y \in V}{argmax} P(x|y)P(y)$$

- **Why bother using Bayes rule? Can't we train $P(y \mid x)$ directly?**
  - in practice yes
  - however, training $P(x \mid y)$ or $P(y \mid x)$ requires **similar effort**: in both cases we need a corpus of **input-output** pairs:
    - **speech recognition**: speech wave forms and transcribed text
    - **spelling correction**: misspelled words and correct spelling
  - such data is **rare**, so the model won't be too good
  - however, training $P(y)$ is **easy**: it's a **language model**, so plenty of data available
  - if we train both $P(x \mid y)$ and $P(y)$, the latter will be **very good**, so it adds reliability for the overall model
  - **How can we train the noisy model?**
    - naively, we can search through **all** possible inputs, until we find the **best** one
    - this is a **search problem**, which is rather **inefficient** (nearly infinite possible inputs)
    - instead, we make **assumptions**, and use **edit distance**

2.2 First Try: Spelling Correction

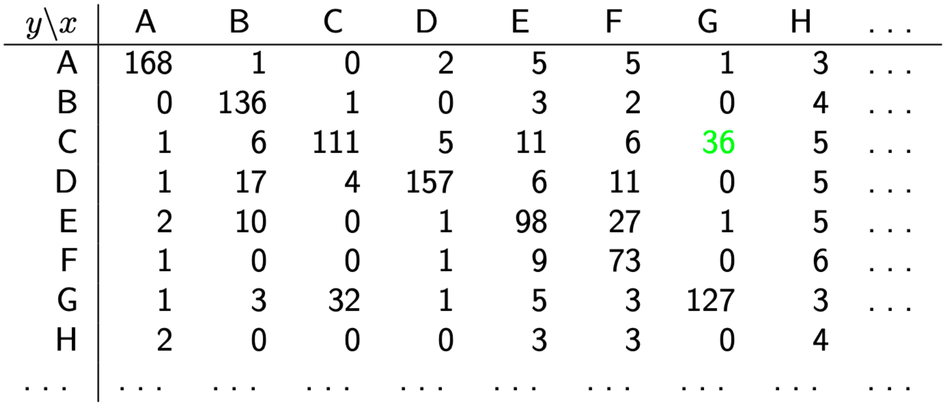
- **What simplified model can we use for spelling correction?**
  - we make 3 **extremely unrealistic** assumptions:
    1. existence of **large dictionary** with **real words**
    2. spelling mistakes not caused by word **splits** or **merges**
    3. spelling mistaks create **non-words** by **inserting, deleting** or **substituting** characetrs in **real words**
  - under these assumptions, to correct mistakes:
    1. Produce a list of all words $\underline{y}$ differing from the non-word $\underline{x}$ by 1 character
    2. Return the $\underline{y}$ with the highest:
$$P(\underline{x} \mid \underline{y})P(\underline{y})$$
- **Under the above assumptions, how can we compute the noise model** $P(\underline{x} \mid \underline{y})$
  - we assume that typing a character x_i depends **only** on the character that was meant to be typed y_i
  - since we ignore **context**, we get that:
$$P(\underline{x} \mid \underline{y}) = \prod_{i=1}^{n} P(x_i \mid y_i)$$
  - for example,
$$P(no \mid not) = P(n \mid n) \times P(o \mid o) \times P(- \mid t)$$
- **What is an alignment corpus?**
  - the **corpus** used to compute the probabilities:
$$P(x_i \mid y_i)$$
  - it contains **alignments** between our correct words $\underline{y}$, and the misspelled words $\underline{x}$



```
actual:    n  o  -     m  u  u  c  h     e  f  f  e  r  t
           |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
intended:  n  o  t     m  -  u  c  h     e  f  f  o  r  t
```

Figure 3: We have one **substitution** ($o \rightarrow e$), one **deletion** ($t \rightarrow -$) and one **insertion** **(**$- \rightarrow u$**)**. The actions are always from the produced output, to the intended input.

- **How do we use an alignment corpus to compute probabilities?**
  - construct a **confusion matrix**, with the rows indicating the **intended input character**, and the columns indicating the **output character**
  - each entry contain the number of times in which a character y_i was typed as x_i (including the empty character "-" to account for deletion/insertion)



$y\backslash x$	A	B	C	D	E	F	G	H	. . .
A	168	1	0	2	5	5	1	3	. . .
B	0	136	1	0	3	2	0	4	. . .
C	1	6	111	5	11	6	36	5	. . .
D	1	17	4	157	6	11	0	5	. . .
E	2	10	0	1	98	27	1	5	. . .
F	1	0	0	1	9	73	0	6	. . .
G	1	3	32	1	5	3	127	3	. . .
H	2	0	0	0	3	3	0	4	
. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .

Figure 4: We intended to write C, but wrote G, a total of **36** times.

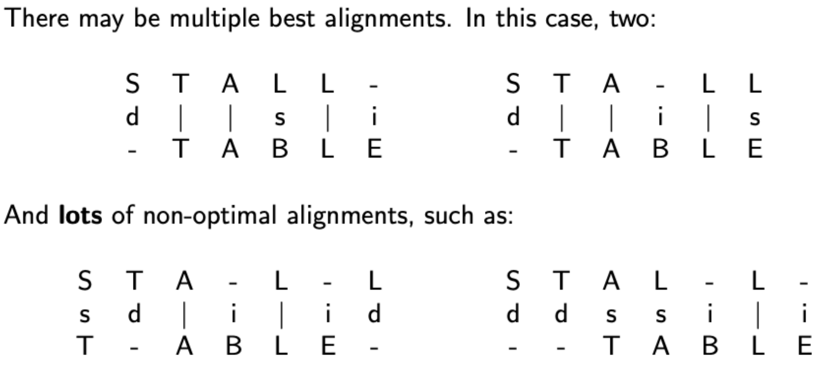
  - apply MLE/smoothing to estimate probabilities. For example:

$$P(H \mid H) = \frac{c(H,H)}{c(H)} = \frac{4}{2+3+3+4} = \frac{4}{12} = \frac{1}{3}$$

- **How is the spelling correction system constructed?**
  - with the alignment corpus, we construct the confusion matrix
  - we can then see how many words $\underline{y}$ are 1 edit away from x (for example, see this implementation by Peter Norvig)
  - finally, compute $P(\underline{x} \mid \underline{y}P(\underline{y}))$, and pick the $\underline{y}$ that produces the largest probability

2.3 Dynamic Programming: Edit Distance

- **What is minimum edit distance?**
  - the number of **character changes** required to convert a word into another word
    - $med(stall, table) = 3$: delete s, change first l for b, insert an e at the end
    - $med(intention, execution)$: 5 actions
  - a character change can have different values (for example, if a substitution costs 2, MED is known as the **Levenshtein** distance)
- **How many possible alignments are there?**
  - many possibilities
  - MED seeks **optimal** alignments, but many **non-optimal alignments** are possible



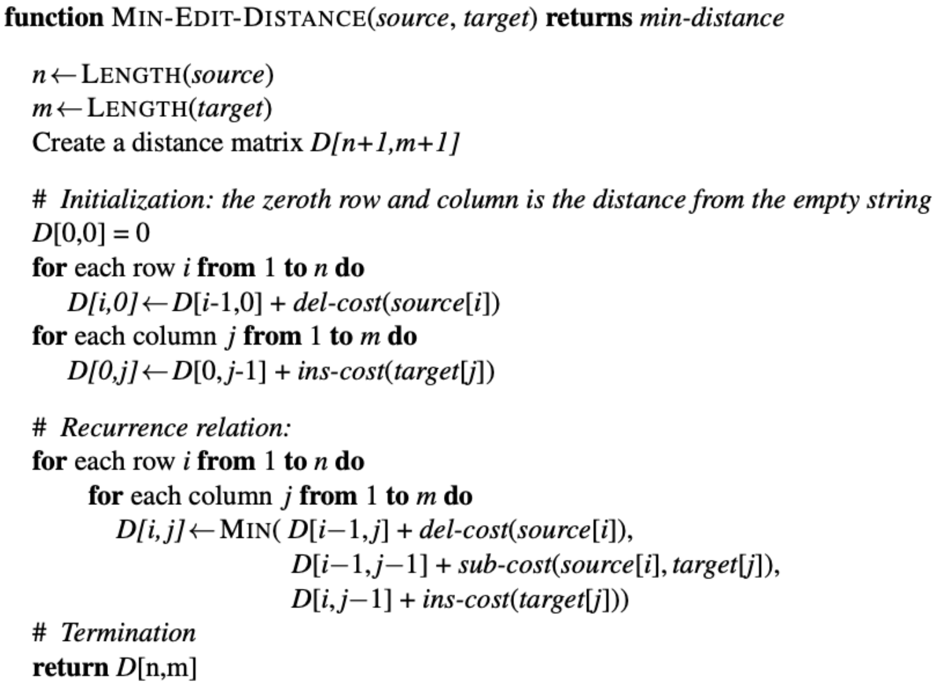
There may be multiple best alignments. In this case, two:

```
S T A L L -        S T A - L L
d | | s | i        d | | i | s
- T A B L E        - T A B L E
```

And **lots** of non-optimal alignments, such as:

```
S T A - L - L      S T A L - L -
s d | i | i d      d d s s i | i
T - A B L E -      - - T A B L E
```

- **Why is edit distance useful?**
  - it helps solve the 3 key problems in the naive approach above:
    - unrealistic independence assumption
    - errors typically occur in more than 1 character
    - can't always have an alignment corpus
  - using edit distance, we can easily develop our spelling corrector (just need a dictionary of words)
- **Can we compute edit distance with brute force?**
  - the number of possible alignments of 2 strings is **exponential**
  - checking each possibility is **unfeasible**
- **What is the edit distance algorithm?**
  - a **dynamic programming** algorithm, which guarantees $\mathcal{O}(mn)$ runtime (where m, n are the lengths of the strings)
  - the key idea is that the minimum distance between 2 string $D(s[1, n], t[1, m])$ is one of the following:
    - $D(s[1, n], t[1, m-1]) + cost(ins)$ (we insert a character into t)
    - $D(s[1, n-1], t[1, m]) + cost(del)$ (we delete a character from t)
    - $D[s[1, n-1], t[1, m-1]) + cost(sub)$ (we substitute a character from t)
  - we can employ this recursive idea, and store results in a table, in which entry $D[i, j]$ corresponds to the MED of $s[1, i], t[1, j]$



```
function MIN-EDIT-DISTANCE(source, target) returns min-distance

  $n \leftarrow$ LENGTH(source)
  $m \leftarrow$ LENGTH(target)
  Create a distance matrix $D[n+1,m+1]$

  # Initialization: the zeroth row and column is the distance from the empty string
  $D[0,0] = 0$
  for each row $i$ from 1 to $n$ do
     $D[i,0] \leftarrow D[i-1,0]$ + del-cost(source[$i$])
  for each column $j$ from 1 to $m$ do
     $D[0,j] \leftarrow D[0,j-1]$ + ins-cost(target[$j$])

  # Recurrence relation:
  for each row $i$ from 1 to $n$ do
       for each column $j$ from 1 to $m$ do
          $D[i,j] \leftarrow$ MIN( $D[i-1,j]$ + del-cost(source[$i$]),
                          $D[i-1,j-1]$ + sub-cost(source[$i$], target[$j$]),
                          $D[i,j-1]$ + ins-cost(target[$j$]))
  # Termination
  return $D$[n,m]
```

- **What costs should be used in MED?**
  - we can choose them depending on how we want to define the importance of each operation

  - for our noise model, we can compute the **most probable** way of changing one word into another by using:

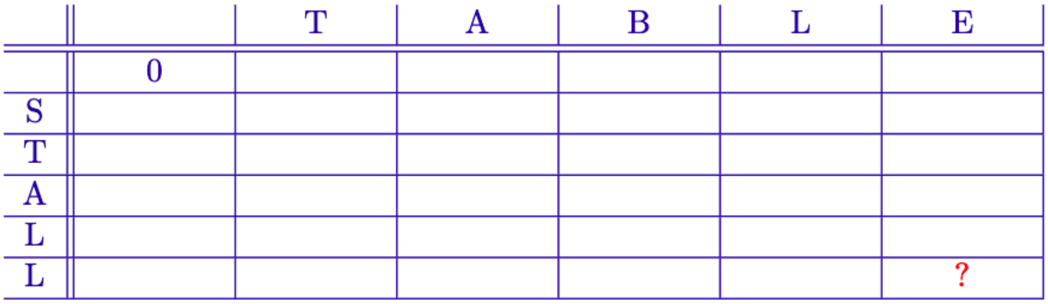
$$cost(sub(c, c')) = P(c' \mid c)$$

  - however, this leads to a Catch 22: to compute the cost of operations in edit distance, we need to use the noisy channel model, which relies on alignments; but to get the alignments, we need to use the costs for edit distance

- **How can we keep track of the alignment after applying the edit distance algorithm?**
  - instead of using an alignment corpus, we can use the edit distance table to generate the alignment
  - just use pointers in each cell to indicate how the substrings were aligned
  - in particular:
    * `move down = deletion`
    * `move right = insertion`
    * `move diagonal right down = substitution`

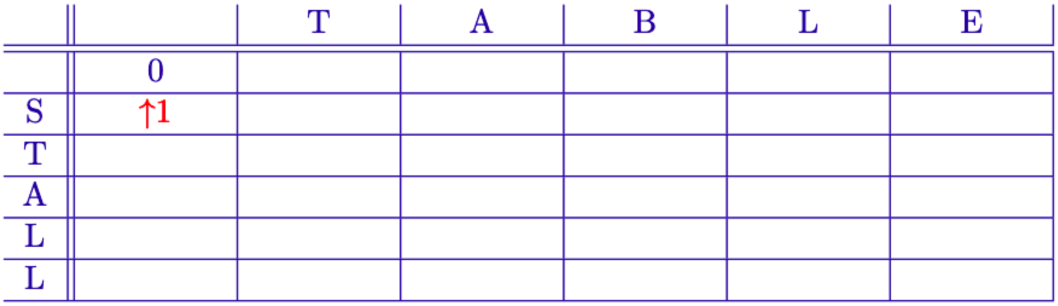
- **What is MED used for?**
  - spell correction
  - morphological analysis (how are words related?)
  - aligning DNA sequences

2.3.1 Worked Example: Edit Distance



		T	A	B	L	E
	0					
S						
T						
A						
L						
L						?

Figure 5: We want to find the MED and alignment of stall and table.



		T	A	B	L	E
	0					
S	↑1					
T						
A						
L						
L						

Figure 6: We begin with $med(S, -)$. Only possibility is to perform a **deletion**, so 1 action performed: $D[1, 0] = 1$. We add a pointer to show that S gets aligned with $-$.

		T	A	B	L	E
	0					
S	↑1					
T	↑2					
A						
L						
L						

Figure 7: We continue with $med(ST, -)$. Only possibility is to perform 2 **deletions**, so 2 action performed: $D[2, 0] = D[1, 0] + 1 = 2$. We add a pointer to show that ST gets aligned with $--$.

		T	A	B	L	E
	0					
S	↑1					
T	↑2					
A	↑3					
L	↑4					
L	↑5					

Figure 8: The gist is the same as we move down the column, since we are performing deletions, to align "stall" with the empty string.

		T	A	B	L	E
	0	←1				
S	↑1					
T	↑2					
A	↑3					
L	↑4					
L	↑5					

Figure 9: We now consider $med(-, T)$. Only possibility is to perform 1 **insertion**, so 1 action performed: $D[0, 1] = 1$. We add a pointer to show that $-$ gets aligned with T.

		T	A	B	L	E
	0	←1				
S	↑1	←↖↑2				
T	↑2					
A	↑3					
L	↑4					
L	↑5					

Figure 10: We now consider $med(S, T)$. We consider the 3 possibilities.
We can make a **substitution**: $(-, -) \to (S, T)$. This has a cost of 2. Add a diagonal backpointer, since we align S and T.
We can make a **deletion**: $(S, T) \to (-, T)$. Hence, $med(S, T) = med(-, T) + 1 = 2$. Add a vertical pointer, since we align $(-, T)$ and (S, T).
We can make an **insertion**: $(S, -) \to (S, T)$. Hence, $med(S, T) = med(S, -) + 1 = 2$. Add a horizontal pointer, since we align $(S, -)$ and (S, T)
Either way, we see that $D[1, 1] = 2$.

		T	A	B	L	E
	0	←1				
S	↑1	←↖↑2				
T	↑2	↖1				
A	↑3					
L	↑4					
L	↑5					

Figure 11: We now consider $med(ST, -T)$. We consider the 3 possibilities.
No need to substitute, since the last 2 characters align. Hence, $med(ST, -T) = med(S, -) = 1$.
Just look to the right, diagonal and top, and select the smallest plus the cost. In this case: $D[1, 1] + 0 = 1$, $D[2, 0] + 1 = 2 + 1 = 3$, $D[1, 1] + 1 = 2 + 1 = 3$. Hence, we go to $(1, 1)$.

		T	A	B	L	E
	0	←1	←2	←3	←4	←5
S	↑1	←↖↑2	←↖↑3	←4	←5	←6
T	↑2	↖1	←2	←3	←4	←5
A	↑3	↑2	↖1	←2	←3	←4
L	↑4	↑3	↑2	←↖↑3	↖2	←3
L	↑5	↑4	↑3	←↖↑4	↖↑3	←↖↑4

Figure 12: The complete table. The MED is 4. The total number of optimal alignments is obtained by considering all possible paths through the arrows.

2.4 Expectation Maximisation

- **What stops us from using edit distance?**
  - we have a catch-22:
    - to get the **character alignment costs**, we need the alignment probability (to ensure that edit distance gives the most likely alignment)
    - to get the **alignment probability**, we need edit distance to compute the character alignments
- **What is expectation maximisation?**
  - algorithm used to **estimate** model parameters, when these, alongside auxiliary variables, are **missing**
  - when **parameters** tell us about the variables, and variables tell us the **parameters**, but we have neither
- **How can we use expectation maximisation with edit distance?**
  1. Initialise character alignment costs **arbitrarily** (i.e all to 1)
  2. Run MED to get **optimal character alignment**
  3. Use the alignments as an **alignment corpus**, and compute $P(x_i \mid y_i)$
  4. Use the probabilities as new costs, repeating 2 and 3 until model parameters stop changing
- **Is EM optimal?**
  - the above is **hard EM**, which converges to *something* (not nice to define mathematically)
  - **true EM** (deals with probabilities, expect values) converges to **local optima**
  - θ are the model parameters, $P(data \mid \theta)$ is the **likelihood** of the model; **true EM** will **converge** to a global maximum of the likelihood function

3 Text Classification

3.1 The Classification Task

- **What is text classification?**
  - the process of taking an **observation**, extracting useful **features**, and assigning it a **discrete class**
  - typically use **supervised machine learning**: use training data , labelled with **classes**, to learn how to map an observation to a class
- **How can we formalise supervised classification?**
  - we have an **input** x_i
  - a set of **output** classes $c_1, c_2, \ldots, c_M$
  - the supervised task involves using $(input, class)$ pairs, to learn where a new input gets mapped to
- **What is a probabilistic classifier?**
  - assigns a class using a **probability**

- **What are some tasks handled by text classification?**
  - spam detection
  - sentiment analyser
  - topic classification
  - authorship attribution
  - native language identification
  - gender/dialect/political orientation
- **What are the 2 types of classifiers?**
  - a **generative** classifier models how a class **generates** data (i.e we can sample from the distribution to get instances of a class)
  - a **discriminative** classifier learns the **features** which can be used to **discriminate** between different classes
  - for example, when classifying pictures of cats and dogs, a **generative** classifier learns how a dog/cat looks; a **discriminative** classifier learns features used to discriminate (i.e whether there is a dog collar)
  - **generative** classifiers tend to be **probabilistic**; **discriminative** are not generally (logsitic regression is, but ANNs/DTs aren't)
- **Can n-grams be used as a classifier?**
  - possible, but not practical:
    - want to focus on words, not their organisation
    - features need not be words (i.e POS tags, metadata)

3.2 Naive Bayes: Supervised Classification

- **What is the multinomial naive Bayes classifier?**
  - a **generative, probabilistic, Bayesian** classifier
  - generate a distribution, such that if we have a **document** d, and a set of classes, we derive:

$$\hat{c} = \underset{c \in C}{argmax} P(c \mid d)$$

- **What features does Naive Bayes use?**
  - treats a document as a **bag of words**. For example:

    "This was very exciting. This was great."

    would be given by $d = \{$*(this, 2), (was, 2), (very,1), (exciting, 1), (great,1)*$\}$
  - each feature corresponds to the number of times a word appears
- **Why is Naive Bayes called thus?**
  - it is **Bayesian**, since to learn the distribution we use Bayes Rule:

$$\begin{aligned}\hat{c} &= \underset{c \in C}{argmax} P(c \mid d) \\ &= \underset{c \in C}{argmax} \frac{P(d \mid c)P(c)}{P(d)} \\ &= \underset{c \in C}{argmax} P(d \mid c)P(c)\end{aligned}$$

    - we can remove $P(d)$: it's just a constant across all classes
    - $P(c)$ is the **prior**: the probability of picking a class, without any **prior** knowledge of the document
    - $P(d \mid c)$ is the **likelihood**: the probability of d being generated by class c
  - it is Naive because estimating:

    $$P(d \mid c)P(c) = P(f_1, f_2, \dots, f_n \mid c)P(c)$$

    (where $f_1, \dots, f_n$ are the features of d) is hard (data sparsity), so we make a naive assumption:

    - **Naive Bayes** assumes that **features** are **conditionally independent** given the **class**, so:

      $$P(d \mid c)P(c) = \prod_{i=1}^{n} P(f_i \mid c)P(c)$$

    - this means that according to Naive Bayes, a word occurs just because of the class
    - it doesn't consider word order, or the existence/non-existence of other words

- **Why is Naive Bayes a linear classifier?**
  - instead of multiplying many small probabilities, we can add **costs** (negative log of probabilities)
  - hence:

    $$\hat{c} = \underset{c \in C}{argmin} \left(-\log(P(c)) - \sum_{i=1}^{n} \log(P(f_i \mid c)) \right)$$

  - this leads to a linear function in log space

- **Are word counts the only type of features possible?**
  - **binary features**: $f_i = 1$ if a word appears, and 0 otherwise
  - **binarisation**: don't double-count words which appear more than once in a document
    - for example, "and it was so fun, and so great", "and" and "so" would only add 1 to the count
  - ignore **stopwords**
    - might not always work: stopwords can be useful
    - depressive people use more 1st person; people with schizophrenia use more 2nd person
  - only use **task-relevant** words
    - for example, in **sentiment analysis**, consider only **sentiment lexicon** (i.e brave, acclaimed are positive; abysmal, cold are negative)
    - however, some other words might be useful
    - for instance, for computer reviews, "quite" and "memory" can be useful
    - if negations (i.e not, don't, won't) appear, we can change subsequent words via:

      $$\textit{“not great”} \rightarrow \textit{“not NOT\_great”}$$

  - more complex features: syntactic, n-grams (these are used in language identification - for example "nya" can be indicative of Eastern European languages), morphological

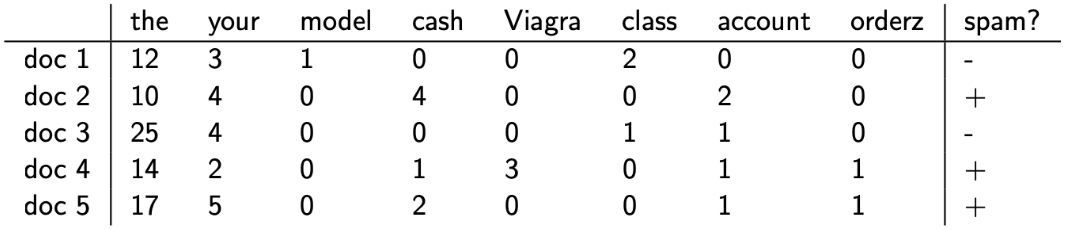
- **How does Naive Bayes learn the prior distribution?**

- just use a MLE estimation:

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

  where:

  - N_c is the number of documents in **training** with class c
  - N is the total number of documents in training

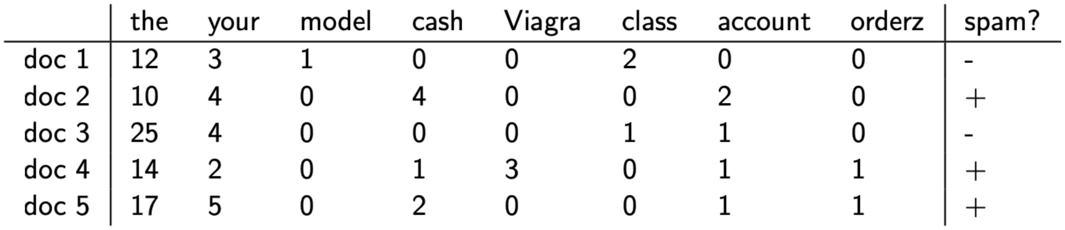


	the	your	model	cash	Viagra	class	account	orderz	spam?
doc 1	12	3	1	0	0	2	0	0	-
doc 2	10	4	0	4	0	0	2	0	+
doc 3	25	4	0	0	0	1	1	0	-
doc 4	14	2	0	1	3	0	1	1	+
doc 5	17	5	0	2	0	0	1	1	+

Figure 13: In binary classification, we have here that $P(spam) = 3/5 = 0.6$

- **How does Naive Bayes learn the likelihood distribution?**
  - use **Laplace Smoothing** MLE:

$$P(f_i \mid c) = \frac{count(f_i, c) + \alpha}{\left(\sum_{f \in F} count(f, c) + \alpha\right)} = \frac{count(f_i, c) + \alpha}{\left(\sum_{f \in F} count(f, c)\right) + \alpha F}$$

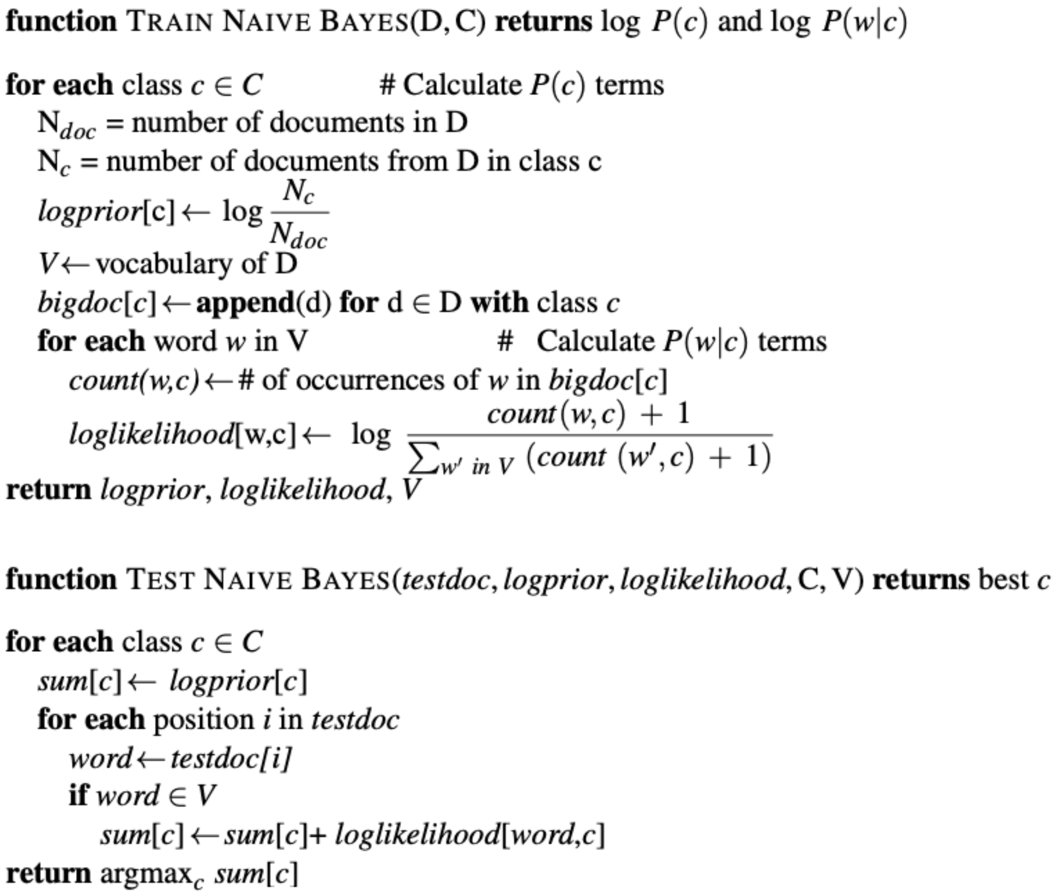


	the	your	model	cash	Viagra	class	account	orderz	spam?
doc 1	12	3	1	0	0	2	0	0	-
doc 2	10	4	0	4	0	0	2	0	+
doc 3	25	4	0	0	0	1	1	0	-
doc 4	14	2	0	1	3	0	1	1	+
doc 5	17	5	0	2	0	0	1	1	+

Figure 14: For example, the likelihood of "your" appearing in a spam document is:

$$P(your \mid spam) = \frac{11 + \alpha}{68 + \alpha F}$$

where "your" appears 11 times in spam documents, and there are 68 word counts in spam documents.



```
function TRAIN NAIVE BAYES(D, C) returns log $P(c)$ and log $P(w|c)$

for each class $c \in C$            # Calculate $P(c)$ terms
    N$_{doc}$ = number of documents in D
    N$_c$ = number of documents from D in class c
    logprior[c] ← log $\frac{N_c}{N_{doc}}$
    V ← vocabulary of D
    bigdoc[c] ← append(d) for d ∈ D with class c
    for each word w in V                # Calculate $P(w|c)$ terms
        count(w,c) ← # of occurrences of w in bigdoc[c]
        loglikelihood[w,c] ← log $\frac{count(w,c)\ +\ 1}{\sum_{w' \text{ in } V} (count\ (w',c)\ +\ 1)}$
return logprior, loglikelihood, V

function TEST NAIVE BAYES(testdoc, logprior, loglikelihood, C, V) returns best c

for each class $c \in C$
    sum[c] ← logprior[c]
    for each position i in testdoc
        word ← testdoc[i]
        if word ∈ V
            sum[c] ← sum[c] + loglikelihood[word,c]
return argmax$_c$ sum[c]
```

3.2.1 Naive Bayes: Worked Example

[In practice, if we are given an instance to classify, and it contains a word which does not appear in the vocabulary, such word is ignored.]

We again consider:

	the	your	model	cash	Viagra	class	account	orderz	spam?
doc 1	12	3	1	0	0	2	0	0	-
doc 2	10	4	0	4	0	0	2	0	+
doc 3	25	4	0	0	0	1	1	0	-
doc 4	14	2	0	1	3	0	1	1	+
doc 5	17	5	0	2	0	0	1	1	+

and we try to classify:

"get your cash and your orderz"

- $count(get, spam) = 0$
- $count(your, spam) = 4 + 2 + 5 = 11$
- $count(cash, spam) = 4 + 1 + 2 = 7$

- $count(and, spam) = 0$
- $count(your, spam) = 11$
- $count(orderz, spam) = 1 + 1 = 2$

Hence:

$$P(spam \mid d) = \frac{3}{5} \times \frac{\alpha}{68 + \alpha F} \times \frac{11 + \alpha}{68 + \alpha F} \times \frac{7 + \alpha}{68 + \alpha F} \times \frac{\alpha}{68 + \alpha F} \times \frac{11 + \alpha}{68 + \alpha F} \times \frac{2 + \alpha}{68 + \alpha F}$$

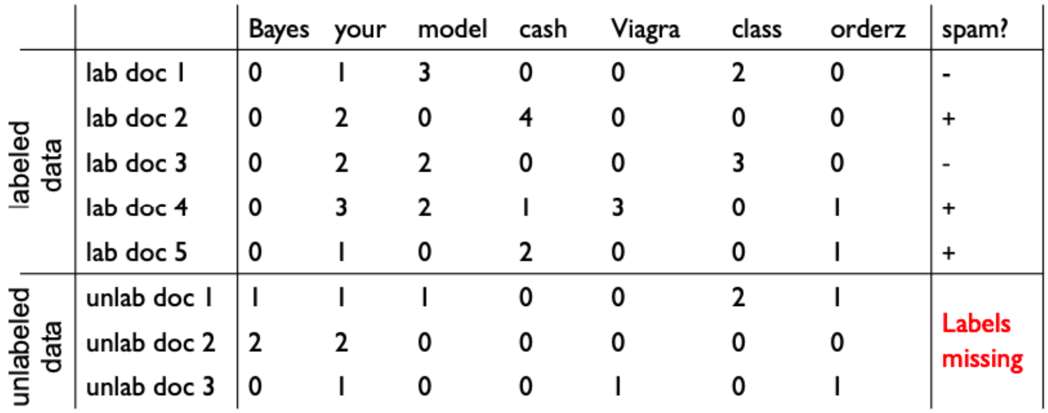
If we do the same thing for "not spam", we can then classify based on which of the 2 has the largest probability.

3.3 Naive Bayes: Self-Supervised Classification

- **What is semi-supervised learning?**
  - a way of improving a classifier, in a situation with:
    - limited **annotated** data
    - plentiful **unannotated** data
- **How can we apply semi-supervised learning to NB?**
  1. Train NB on the labelled data
  2. Predict labels of unlabelled data
  3. Now that all the data has an annotation, re-estimate NB
  4. Keep re-training NB, and re-labelling the unlabelled data

3.3.1 Naive Bayes (Semi-Supervised): Worked Example

Consider the following:



		Bayes	your	model	cash	Viagra	class	orderz	spam?
labeled data	lab doc 1	0	1	3	0	0	2	0	-
	lab doc 2	0	2	0	4	0	0	0	+
	lab doc 3	0	2	2	0	0	3	0	-
	lab doc 4	0	3	2	1	3	0	1	+
	lab doc 5	0	1	0	2	0	0	1	+
unlabeled data	unlab doc 1	1	1	1	0	0	2	1	Labels missing
	unlab doc 2	2	2	0	0	0	0	0	
	unlab doc 3	0	1	0	0	1	0	1	

We have $F = 5, \alpha = 0.1$. In the labelled docs, in spam we have 20 words; in non-spam we have 13 words. We can classify the unlabelled doc 2:

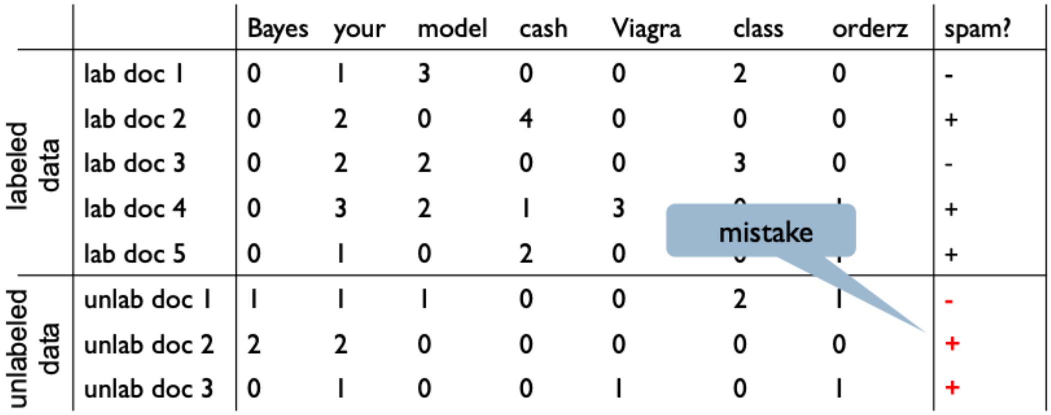
$$P(d2 \mid +) = \frac{\alpha^2}{(20 + 5\alpha)^2} \times \frac{(6 + \alpha)^2}{(20 + 5\alpha)^2} = \frac{\alpha^2 (6 + \alpha)^2}{(20 + 5\alpha)^4} \approx 2.1 \times 10^{-6}$$

$$P(d2 \mid -) = \frac{\alpha^2}{(13+5\alpha)^2} \times \frac{(3+\alpha)^2}{(13+5\alpha)^2} = \frac{\alpha^2(3+\alpha)^2}{(13+5\alpha)^4} \approx 2.9 \times 10^{-6}$$

Thus:

$$P(+ \mid d2) \propto \frac{3}{5} \times 2.1 \times 10^{-6} \qquad P(- \mid d2) \propto \frac{2}{5} \times 2.9 \times 10^{-6}$$

Hence, the second document is most likely spam. Doing this for the remaining docs:



		Bayes	your	model	cash	Viagra	class	orderz	spam?
labeled data	lab doc 1	0	1	3	0	0	2	0	-
	lab doc 2	0	2	0	4	0	0	0	+
	lab doc 3	0	2	2	0	0	3	0	-
	lab doc 4	0	3	2	1	3	0	1	+
	lab doc 5	0	1	0	2	0	0	1	+
unlabeled data	unlab doc 1	1	1	1	0	0	2	1	-
	unlab doc 2	2	2	0	0	0	0	0	+
	unlab doc 3	0	1	0	0	1	0	1	+

mistake

We can retrain NB. Again for doc 2 (now with $F = 7$, since we add "Bayes" and "Viagra"):

$$P(d2 \mid +) = \frac{(2+\alpha)^2}{(20+7+7\alpha)^2} \times \frac{(6+3+\alpha)^2}{(20+7+7\alpha)^2} = \frac{(2+\alpha)^2(9+\alpha)^2}{(27+7\alpha)^4} \approx 6.2 \times 10^{-4}$$

$$P(d2 \mid -) = \frac{(1+\alpha)^2}{(13+6+7\alpha)^2} \times \frac{(3+1+\alpha)^2}{(13+6+7\alpha)^2} = \frac{(1+\alpha)^2(4+\alpha)^2}{(19+7\alpha)^4} \approx 1.35 \times 10^{-4}$$

$$P(+) = \frac{3+2}{5+3} = \frac{5}{8}$$

Notice that since we are treating the unlabelled data as labelled, this changes the counts: for example, + now has 7 new instances, and "your" appears in spam 3 more times.

The classification becomes:

$$P(+ \mid d2) \propto \frac{5}{8} \times 6.2 \times 10^{-4} \qquad P(- \mid d2) \propto \frac{3}{8} \times 1.35 \times 10^{-4}$$

Even d2 is **not** spam, it is continuously classified as such (and with bigger probability). Why does this happen? Computing the original probability:

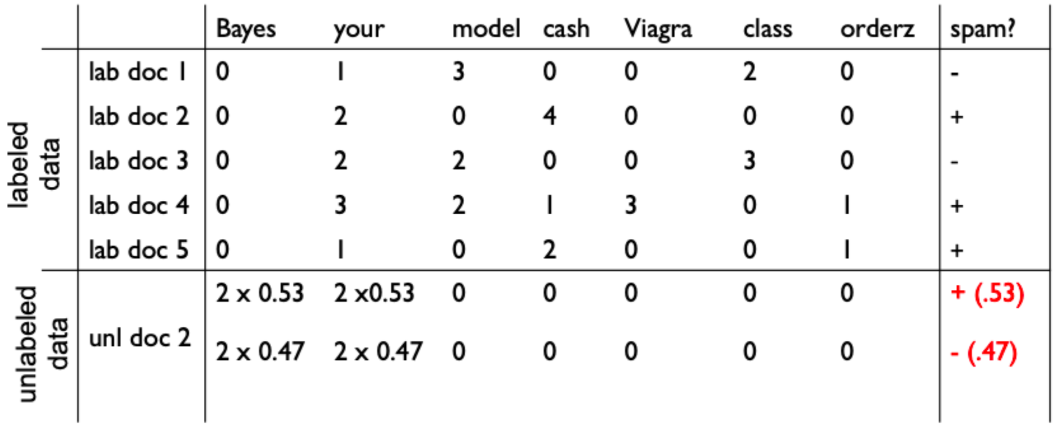
$$P(+ \mid d2) = \frac{\frac{3}{5} \times 2.1 \times 10^{-6}}{\frac{3}{5} \times 2.1 \times 10^{-6} + \frac{2}{5} \times 2.9 \times 10^{-6}} \approx 0.52$$

In other words, the **model** is not confident in its decision, and nonetheless we are treating the decision as a **gold label**.

- **What are the pros and cons of self-training?**
  - pros:
    - simple
    - works on other classifiers

    - ∗ NB is simple to update given new data
  - cons:
    - ∗ doesn't account for uncertainty
  - improvements:
    - ∗ discard low-confidence prediction
    - ∗ use a **curriculum** (begin training the model with unlabelled data similar to labelled data)

- **How can we improve self-supervised NB?**
  - instead of using **gold labels**, use **soft labels** (**expectation maximisation**)
  - that is, each document is spam and not spam with a probability. In the example above, doc 2 is 0.52+ **and** 0.48-
  - similarly, the observed counts also get adapted:



		Bayes	your	model	cash	Viagra	class	orderz	spam?
labeled data	lab doc 1	0	1	3	0	0	2	0	-
	lab doc 2	0	2	0	4	0	0	0	+
	lab doc 3	0	2	2	0	0	3	0	-
	lab doc 4	0	3	2	1	3	0	1	+
	lab doc 5	0	1	0	2	0	0	1	+
unlabeled data	unl doc 2	2 x 0.53	2 x0.53	0	0	0	0	0	+ (.53)
		2 x 0.47	2 x 0.47	0	0	0	0	0	- (.47)

Figure 15: I rounded to 2 significant figures, so my probabilities were 0.52.

  - self-learning is nothing but **hard EM**
  - improves likelihood of observed data

- **What is the EM algorithm for NB?**
  1. Train NB on labelled data
  2. Make soft predictions on unlabelled data
  3. Recompute NB model using soft counts

3.3.2 Naive Bayes (Semi-Supervised, EM: Worked Example

If we now want to compute the probabilities for doc2 being spam, we need to use:

$$P_{doc2}(spam) = \frac{3 + 0.52}{5 + 1}$$

The above is **only** applicable to doc2; for the others the probability changes.

$$P(your \mid +) = \frac{6 + 2 \times 0.52 + \alpha}{20 + 4 \times 0.52 + \alpha F}$$

$$P(your \mid -) = \frac{3 + 2 \times 0.48 + \alpha}{13 + 4 \times 0.48 + \alpha F}$$

$$P(Bayes \mid +) = \frac{2 \times 0.52 + \alpha}{20 + 4 \times 0.52 + \alpha F}$$

$$P(Bayes \mid -) = \frac{2 \times 0.48 + \alpha}{13 + 4 \times 0.48 + \alpha F}$$

3.3.3 Evaluating Naive Bayes

In general, NB works reasonably well, and should be a classification baseline.

- **What are the pros?**
  - easy to implement
  - fast to train/classify
  - good for small data sets
  - easy to update with new data (just update counts)
- **What are the cons?**
  - its ... *naive*. For example, if the classes are *travel* and *sport*, the features “beach”, “sun”, “snow”, “football”, “pitch” are **not** independent given the category. Given *travel*, if we see “beach”, it is more likely to see “sun” than “snow”
  - tends to be overconfident (5 features pointing to class 1 treated as 5 independent sources of evidence)