# Introduction to Natural Language Processing and Text Classification

**Goal:** discuss how to build a spam filter.

documents come in and we decide $\begin{cases} \text{spam ?} \\ \text{ham ?} \end{cases}$

We will see how to train a classifier using prelabeled documents. (in this case, emails w/ 'spam' or 'ham' labels).

To do this requires knowing:

1. How to preprocess text
2. How to turn text data into features

   (the $X$ in the equation $Y = f(X) + \epsilon$)
3. Probability theory to build a classifier.
   - Bayes theorem
   - Laplace smoothing, log-probabilities (implementation details)

---

## Processing text

We've done this in the past (homeworks), but it is a key step to NLP (Natural Language Processing).

### Terminology:

Token - words, punctuation, numbers, etc.
Sentence - ordered sequence of tokens

Tokenization - process of segmenting a sentence into tokens. Whitespace makes tokenizing English, say, easy. But other languages, such as Chinese, are hard to tokenize.

Corpus - A body of text, usually w/ a large number of Sentences.

**Documents** - Many corpora are broken down into smaller bodies of text called documents. For example, each individual email is a document in an email corpus.

**Parts-of-Speech Tag** (POS) - Words may be nouns, verbs, adjectives, etc. POS tags are symbols representing those categories:

NN - Noun
VB - Verb
JJ - adjective
AT - article
etc....

**Stop words** - common, low meaning words "the", "that", etc. which are often, but not always removed from a text.

**POS tagging** - labelling the tokens in a sentence

$\underbrace{\text{The}}_{AT}\ \underbrace{\text{ball}}_{NN}\ \underbrace{\text{is}}_{VB}\ \underbrace{\text{red}}_{JJ}.$

**Stemming** - breaking words down to root morphemes by, among other things removing suffixes.

cats, catlike, catty ⇒ cat.

argue, argues, arguing, argued, ⇒ argu (not a word)

**Lemmatization** - grouping together the different inflected forms of a word.

- walk, walks, walked, walking ⇒ walk (lemma same as stem)
- better ⇒ good (lemma not matched by stemming)
- "meeting" may be a noun (lemma: meeting) or a verb (lemma: meet). Context required.

Stemming and Lemmatization are common text preprocessing steps. ⇒ Normalize the words.

## Turning text into features:

The simplest feature transform for a tokenized text is the Bag-of-Words model.

→ throw out any structure/order to the text and assume all the data is with the counts: the number of times each unique word occurred.

→ each document becomes a count vector

$v_d(\text{"foo"}) = 7$ → "foo" appeared in document d a total of 7 times.

→ dimensionality of v's → # unique words in the corpus.

(notice the effects stemming/lemmatization may have.)

→ unigrams ⇒ n-grams.

---

## Text classification → spam filtering

Given a document d what is the probability it is or is not spam?

We want $\Pr(\text{spam} \mid d)$. We can generalize to more than two categories/classes.

$$\Pr(c \mid d) \quad \text{where, for now, } c \in \{\text{spam}, \text{ham}\}.$$

If we can compute this probability for each c, we can decide which class the document belongs to.

$$c_{MAP} = \underset{c}{\arg\max}\ P(c \mid d) \quad \left(MAP = \text{maximum a posteriori, most likely}\right)$$

But, how to compute $\Pr(c \mid d)$?

First ingredient we need: <u>Bayes Theorem</u>

$$Pr(A \text{ and } B) = Pr(B \text{ and } A), \qquad Pr(A \text{ and } B) = P(A|B) \cdot P(B)$$

plug in

$$Pr(A|B) Pr(B) = Pr(B|A) Pr(A)$$

$$Pr(A|B) = \frac{Pr(B|A) Pr(A)}{Pr(B)}$$ that's it!

Bayes thm let's us "flip around" conditional probabilities.

In the text classifier it is easier to <u>measure</u> $Pr(d|c)$ than $Pr(c|d)$.

$$c_{MAP} = \underset{c}{\text{argmax}}\ Pr(c|d)$$

$$= \underset{c}{\text{argmax}}\ \frac{Pr(d|c) Pr(c)}{Pr(d)}$$

Now, to find a way to calculate $c_{MAP}$ we need to make some simplifications.

I. The document we want to classify is constant, this means that $Pr(d)$ is the same regardless of $c \Rightarrow$ it won't change what $c_{MAP}$ is:

$$c_{MAP} = \underset{c}{\text{argmax}}\ Pr(d|c) Pr(c)$$

Next, the document $d$ is a collection of words:

$$Pr(d|c) = Pr(w_1, w_2, \ldots, w_n | c)$$

this is <u>not</u> the bag of words (BOW) model.

Assume Bag-of-Words $\Rightarrow$ position/order of words doesn't matter.
Assume conditional independence $\Rightarrow$ probabilities of different words appearing together are independent given the document class.

$$Pr(d|c) = Pr(w_1, \ldots, w_n | c) = Pr(w_1|c) \cdot Pr(w_2|c) \cdots Pr(w_n|c)$$

Note these last two assumptions are <u>certainly not true</u> for a real text!

Put these together and you have constructed a (text) classifier called <u>Naive Bayes</u>

$$C_{MAP} = \underset{c}{\arg\max}\ \Pr(d|c)\Pr(c)$$

$$\Downarrow$$

$$C_{NB} = \underset{c}{\arg\max}\ \Pr(c)\prod_{i=1}^{n}\Pr(w_i|c)$$

## Learning Naive Bayes

How to compute these probabilities ...

Training corpus $N_{doc}$ documents, each labeled w/ c = spam or c = ham.

Estimate probabilities: $\hat{P}(c) = \dfrac{\#\text{ docs labeled } c}{N_{docs}}$

$$\hat{P}(w_i|c) = \frac{\text{count}(w_i, c)}{\sum_j \text{count}(w_j, c)}$$

fraction of the words among all c documents which are word i.

<u>Problem!</u> What if we use this $\hat{P}$ estimator and then, when we attempt to classify a new document we see a new word we have never seen before?

→ word will have a count of zero ⇒ $\hat{P}(w|c) = 0$
plug into $C_{NB}$ and it becomes zero:

$$C_{NB} = \underset{c}{\arg\max}\ \hat{P}(c)\prod_{i=1}^{n}\hat{P}(w_i|c)$$

one of those is now zero!

The fix is <u>madness</u> → Laplace (or additive) smoothing!

assume: $$\hat{P}(w_i|c) = \frac{\text{count}(w_i, c) + 1}{\sum_{j=1}\left(\text{count}(w_j, c) + 1\right)} = \frac{\text{count}(w_i, c) + 1}{\sum_{j=1} \text{count}(w_j, c) + |V|}$$

" n