

Naïve Bayes classifiers

Terence Parr
MSDS program
University of San Francisco

Common problem: detect spam tweets

- Q. Is the following tweet spam or ham (not spam)?



- Without knowledge of content, what's the best we can do?



SPAM at big package store

Common problem: detect spam tweets

- Q. Is the following tweet spam or ham (not spam)?



- Without knowledge of content, what's the best we can do?
- Use prior knowledge about relative likelihoods of spam/ham
- If *a priori*, we know 75% of tweets are spam, always guess spam
- (Note: this is solving same problem as, say, article topic classification)

Our base model

- If 75% of tweets are spam, always guessing spam gives us a baseline of 75% accuracy (which we hope to surpass)
- *Accuracy* has formal definition: % correctly-identified tweets
- A superior model must do better than 75% accuracy
- What if *a priori* spam rate was 99%? Model has 99% accuracy
- That hints that accuracy can be very misleading by itself and for imbalanced datasets
- How can we do better than just the *a priori* probabilities?

Better model using knowledge of content

- If we can see tweet words, we have more to go on; e.g.,

Viagra sale

Buy catfood

- Given “Viagra sale”, how do you know it’s spam? Because we know:

$$P(\text{spam} \mid \text{Viagra} \cap \text{sale}) > P(\text{ham} \mid \text{Viagra} \cap \text{sale})$$

- We know that (partially) because “Viagra sale” is much more likely to appear in spam emails than in ham emails:

$$\begin{array}{l} P(\text{Viagra} \cap \text{sale} \mid \text{spam}) \text{ is high} \\ P(\text{Viagra} \cap \text{sale} \mid \text{ham}) \text{ is low} \end{array}$$

- Likewise, we know “Buy catfood” is **unlikely** to occur in spam email

Model based upon tweet likelihoods

- Predict **spam** if:

$$P(\text{Viagra} \cap \text{sale} \mid \text{spam}) > P(\text{Viagra} \cap \text{sale} \mid \text{ham})$$

- Predict **ham** if:

$$P(\text{Viagra} \cap \text{sale} \mid \text{spam}) < P(\text{Viagra} \cap \text{sale} \mid \text{ham})$$

- This model works great but makes an assumption by not taking into consideration what knowledge? The *a priori* probabilities; assumes equal priors

Model combining priors and content info

- Predict **spam** if:

$$P(\text{spam})P(\text{Viagra} \cap \text{sale} \mid \text{spam}) > P(\text{ham})P(\text{Viagra} \cap \text{sale} \mid \text{ham})$$

- Predict **ham** if:

$$P(\text{spam})P(\text{Viagra} \cap \text{sale} \mid \text{spam}) < P(\text{ham})P(\text{Viagra} \cap \text{sale} \mid \text{ham})$$

- We are weighting the content likelihoods by prior overall spam rate
- If spam-to-ham priors are .5-to-.5 the prior terms cancel out

Are these computations probabilities?

- Do these terms sum to 1.0 after weighting (covering all likelihood)?
 - $P(\text{spam})P(\text{Viagra} \cap \text{sale} \mid \text{spam}) + P(\text{ham})P(\text{Viagra} \cap \text{sale} \mid \text{ham}) \neq 1$
- Nope. Must normalize term by dividing by (unconditional) probability of ever seeing that specific word sequence:

$$P(\text{Viagra} \cap \text{sale})$$

Imagine
 $P(\text{spam}) =$
 $P(\text{ham}) = 0.5$
and .7 for
conditional
probabilities,
sum is $0.7 \neq 1$

- Dividing by the *marginal* probability makes the terms fractions of the possibilities:

$$\frac{P(\text{Viagra} \cap \text{sale} \mid \text{spam})}{P(\text{Viagra} \cap \text{sale})}$$

- Answers “How much of unconditional does conditional cover?”

Yay! You've just reinvented *Bayes Theorem*

- Normalized likelihood decision rule:

$$\frac{P(\text{spam})P(\text{Viagra} \cap \text{sale} \mid \text{spam})}{P(\text{Viagra} \cap \text{sale})} \gtrless \frac{P(\text{ham})P(\text{Viagra} \cap \text{sale} \mid \text{ham})}{P(\text{Viagra} \cap \text{sale})}$$

- Says how to adjust *a priori* knowledge of spam rate with tweet content evidence

Maximum *a posteriori* classifier

- Choose class/category for document d with max likelihood:

$$c^* = \underset{c}{\operatorname{argmax}} P(c|d)$$

- Substitute Bayes' theorem:

$$c^* = \underset{c}{\operatorname{argmax}} \frac{P(c)P(d|c)}{P(d)}$$

Bayes' theorem

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

- You will often see the classification decision rule called the *Bayes test for minimum error*:

$$P(c_1 | d) \geq P(c_2 | d)$$

Simplifying the classifier

- $P(d)$ is constant on both sides so we can drop it for classification:

$$c^* = \underset{c}{\operatorname{argmax}} P(c)P(d|c)$$

- If $P(c)$ is same for all c OR we don't know $P(c)$, we drop that too:

$$c^* = \underset{c}{\operatorname{argmax}} P(d|c)$$

Training the classifier

- We need to estimate $P(c)$ and $P(d \mid c)$ for all c and d
- Estimating $P(c)$? The number of documents in class c divided by the total number of documents; e.g., frequency of spam docs
- Estimating $P(d \mid c)$? E.g., we need $P(\text{Viagra} \cap \text{sale} \mid \text{spam})$
- That means considering all 2-word combinations (*bigrams*); n -grams grow exponentially with length n !
- For 10 word tweet we need to estimate probability of a 10-gram; considering all 10-grams is intractable

The naïve assumption

- Naïve assumption: conditional independence; Estimate $P(\text{Viagra} \cap \text{sale} \mid \text{ham})$ as $P(\text{Viagra} \mid \text{ham}) \times P(\text{sale} \mid \text{ham})$

$$P(d|c) = \prod_{w \in d} P(w|c)$$

- So, our classifier becomes

$$c^* = \underset{c}{\operatorname{argmax}} P(c) \prod_{w \in d} P(w|c)$$

- where w is each word in d with repeats, not V (vocabulary words)

Fixed-length word-count vectors

- Rather than arbitrary-length word vectors for each document d , it's much easier to use fixed-length vectors of size $|V|$ with word counts:

$$c^* = \underset{c}{\operatorname{argmax}} P(c) \prod_{w \in d} P(w|c)$$

becomes:

$$c^* = \underset{c}{\operatorname{argmax}} P(c) \prod_{w \in V} P(w|c)^{n_w(d)}$$

(If w not present in document, exponent goes to 0, which drops out $P(w|c)$ for that w)

Estimating $P(w | c)$

- Use the number of times w appears in all documents from class c divided by the total number of words (including repeats) in all documents from class c :

$$P(w|c) = \frac{\text{wordcount}(w, c)}{\text{wordcount}(c)}$$

- Or, use num docs with w divided by number of docs (which could be better with really short docs like tweets)

What if w never used in docs of class c ?

Laplace smoothing

- If $P(w | c) = 0$ then the entire product goes to zero. Ooops!

$$c^* = \underset{c}{\operatorname{argmax}} P(c) \prod_{w \in V} P(w|c)^{n_w(d)}$$

- To avoid, add 1 to each word count in numerator and compensate by adding $|V|$ to denominator (to keep a probability)

$$P(w|c) = \frac{\text{wordcount}(w, c) + 1}{\text{wordcount}(c) + |V|}$$

- (We have added +1 to **each** word count in V and there are $|V|$ words in each word-count vector)

Dealing with “mispeled” or unknown words

- Laplace smoothing deals with w that is in the vocabulary V but not in class c : *i.e.*, when $P(w | c) = 0$ such as $P(\text{viagra}|\text{ham})=0$
- What should $\text{wordcount}(w,c)$ be for a word not in V when classifying new doc? Zero doesn't seem right; OTOH, if $\text{wordcount}(w,c)=0$ for all classes, classifier is not biased
- Instead: map all unknown w to a wildcard word in V so then $\text{wordcount}(\text{unknown},c)=0$ is ok but $|V|$ is 1 word longer

- Likely not a huge factor...

$$P(w|c) = \frac{\text{wordcount}(w, c) + 1}{\text{wordcount}(c) + |V| + 1}$$

- Store count of unknown words in word vector at index 0;
the $\text{wordcount}(\text{unknown},c)$ is same for all c so not biased
- Likelihood of any unknown word is small: $1 / (\text{wordcount}(c) + |V| + 1)$

Avoiding floating point underflow

- In practice, multiplying lots of probabilities in $[0,1]$ range tends to get too small to represent with finite floating-point numbers
- Take log (a monotonic function) and product becomes summation

$$c^* = \underset{c}{\operatorname{argmax}} P(c) \prod_{w \in V} P(w|c)^{n_w(d)}$$

$$c^* = \underset{c}{\operatorname{argmax}} \left\{ \log(P(c)) + \sum_{w \in V} n_w(d) \times \log(P(w|c)) \right\}$$

An example

Documents as word-count vectors

- One column per vocab word, one row per document

d1 = "sale viagra sale"

d2 = "free viagra free viagra free viagra free"

d3 = "buy catfood, buy eggs"

d4 = "buy eggs"

Priors:

$P(\text{spam}) = 2/4$

$P(\text{ham}) = 2/4$

column for
unknown
words

		X matrix							
Document		unknown	buy	catfood	eggs	free	sale	viagra	spam
	0	0	0	0	0	0	2	1	1
	1	0	0	0	0	4	0	3	1
	2	0	2	1	1	0	0	0	0
	3	0	1	0	1	0	0	0	0

spam or ham

Note:

$|V|=6$

+1 for unknown

Estimating probabilities: $P(w \mid \text{spam})$

- 1st, get total word count in spam category: sum across rows or cols then sum that result
- `wordcount(spam) = spam.sum(axis=1).sum()`

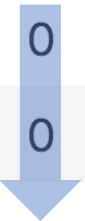

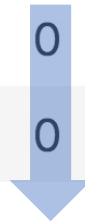
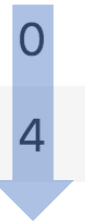
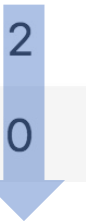
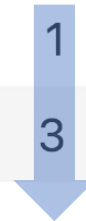
		<i>spam</i>							
		unknown	buy	catfood	eggs	free	sale	viagra	
0	0	0	0	0	0	0	2	1	3
1	0	0	0	0	0	4	0	3	7

wordcount(spam) = 10

$$P(w|c) = \frac{\text{wordcount}(w, c) + 1}{\boxed{\text{wordcount}(c)} + |V| + 1}$$

Estimating probabilities: $P(w \mid \text{spam})$

- 2nd, get total count for each word in spam docs, $\text{wordcount}(w, \text{spam})$

	<i>spam</i>							
	unknown	buy	catfood	eggs	free	sale	viagra	
0	0	0	0	0	0	2	1	
1	0	0	0	0	4	0	3	
								
		0	0	0	4	2	4	$\frac{\text{wordcount}(w c)}{\text{wordcount}(c)}$
		10	10	10	10	10	10	

$$P(w|c) = \frac{\boxed{\text{wordcount}(w, c)} + 1}{\text{wordcount}(c) + |V| + 1}$$

Estimating probabilities: $P(w \mid \text{spam})$

- 3rd, compute $P(w|\text{spam})$ w/smoothing & unknown word adjustment
 - $\text{wordcount}(w, \text{spam}) + 1 =$

unknown	buy	catfood	eggs	free	sale	viagra
1	1	1	1	5	3	5
 - $\text{wordcount}(\text{spam}) + |V| + 1 = 10 + 6 + 1 = 17$

$P(w \mid \text{spam}) \rightarrow$	unknown	buy	catfood	eggs	free	sale	viagra
	0.058824	0.058824	0.058824	0.058824	0.294118	0.176471	0.294118

$$P(w|c) = \frac{\text{wordcount}(w, c) + 1}{\text{wordcount}(c) + |V| + 1}$$

Estimate $P(c|w)$ w/o $P(d)$ normalization

- Dot product of X matrix with log of $P(w|c)$ vector, add $\log(P(c))$

$\log(0.5) + 1 \cdot \log(0.294118) + 2 \cdot \log(0.176471)$

	P(spam d)	P(ham d)
d1 = "sale viagra sale"	-5.386125	-8.387995
d2 = "free viagra free viagra free viagra free"	-9.259575	-18.647793
d3 = "buy catfood and buy eggs"	-12.026001	-6.388596
d4 = "buy eggs"	-6.359574	-3.338139

$$c^* = \underset{c}{\operatorname{argmax}} \left\{ \log(P(c)) + \sum_{w \in V} n_w(d) \times \log(P(w|c)) \right\}$$

Key takeaways

- Naïve bayes is classifier applied to text classification; e.g., spam/ham, topic labeling, etc...
- Less often used these days with rise of deep learning
- Fixed-length "bag of words" vectors are the feature vector per doc
- Bayes theorem gives formula for $P(c|d)$
- Naïve assumption is conditional independence $\longrightarrow P(d|c) = \prod_{w \in d} P(w|c)$
- Training estimates $P(c)$, $P(w|c)$ for each w and c
- $P(c)$ is ratio of docs in c to overall number of docs
- $P(w|c)$ is ratio of word count of w in c to total word count in c
- Classifier: $c^* = \underset{c}{\operatorname{argmax}} P(c) \prod_{w \in V} P(w|c)^{n_w(d)}$

Implementation takeaways

- Avoid vanishing floating-point values from product; take log:

$$c^* = \underset{c}{\operatorname{argmax}} \left\{ \log(P(c)) + \sum_{w \in V} n_w(d) \times \log(P(w|c)) \right\}$$

- Avoid $P(w|c)=0$ via Laplace smoothing
 - add 1 to all word counts
 - adjust $P(w|c)$ denominator with $|V|$ since every doc now has every word
 - this is for missing words where w not in d but in V
- Treat test doc words w not in V , unknown words, as likelihood:
 $1 / (\text{wordcount}(c) + |V| + 1)$

Lab time

- Exploring Naïve Bayes

<https://github.com/parr/msds621/blob/master/labs/bayes/naive-bayes.ipynb>