Naïve Bayes classifiers

Terence Parr
MSDS program
University of San Francisco

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)?

Viagra sale

- 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 not spam detection)

Our base model

- If 75% of tweets are spam, always guessing spam gives us a lower bound of 75% accuracy
- *Accuracy* has formal definition: % correctly-identified tweets
- A superior model must do better than 75% accuracy
- What if a priori spam rate was 99%? Our model has 99% accuracy
- That hints that accuracy can be very misleading in isolation (more later)

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 among spam emails you've received, those words are highly likely to appear
- Words like "Buy catfood" are unlikely to occur in spam email
- Vice versa: "Viagra sale" low likelihood in non-spam
- $P(\text{Viagra} \cap \text{sale} \mid \text{spam})$ is high, $P(\text{Viagra} \cap \text{sale} \mid \text{ham})$ is low

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?

- Are these probabilities after weighting?
  - $P(spam)P(Viagra \cap sale \mid spam) + P(ham)P(Viagra \cap sale \mid ham) \neq 0$
- Nope. Must normalize term by probability of ever seeing that specific word sequence (the marginal probability):

$$P(Viagra \cap sale)$$

- Dividing by the marginal probability makes the terms probabilities
- Called likelihood ratio:

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

- Answers “How much of the marginal does the conditional cover?”

Yay! You've just reinvented *Bayes Theorem*

- Normalized likelihood:

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

- 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:

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

- Substitute Bayes' theorem:

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

Bayes' theorem

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

Simplifying the classifier

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

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

- If P(*c*) is same for all *c* OR we don‘t know P(*c*), we drop that too
- You‘ll often see the classification decision rule written as the *Bayes test for minimum error*:

$$P(c_1 \mid d) \gtrless P(c_2 \mid d)$$

Training the classifier

- We need to estimate P(*c*) and P(*d* | *c*) for all *c* and *d*
- Estimating P(*c*)? The number of documents in class c divided by the total number of documents
- Estimating P(*d* | *c*)? E.g., we need P(Viagra ∩ sale | spam)
- That means considering all 2-word combinations (*bigrams*), which grows in size with the square!

The naïve assumption

- Naïve assumption: conditional independence; Estimate P(Viagra ∩ sale | ham) as P(Viagra | spam) x P(sale | spam)

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

- So, our classifier becomes

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

- 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*| w/word counts:

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

becomes:

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

Estimating P(*w* | *c*)

- We have P(*c*) but we still need to estimate P(*w* | *c*)?
- It's 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:

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

Laplace smoothing

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

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

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

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

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

Dealing with “mispeled” or unknown words

- Laplace smoothing deals with *w* that are in the vocabulary *V* but not in class *c*, which gives P(*w* | *c*) = 0
- What should count(*w,c*) be for a word not in *V*? Zero doesn’t seem right as we have no data; OTOH, if wordcount(*w,c*)=0 for all classes, classifier is not biased
- wordcount(*w,c*)=0 actually won’t occur due to Laplace but it’s like mapping all unknown *w* to a wildcard word in *V*, which means |V| is one bigger
- Likely not a huge factor…

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

Avoiding floating point underflow

- In practice, multipying 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; also, since P(w|c) can't be 0 due to smoothing, we can use $w \in V$ rather than iterating through unique doc words

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

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

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 and buy eggs"
d4 = "buy eggs"
```

Priors:
P(spam) = 2/4
P(ham) = 2/4

column for unknown words

spam or ham

	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

Note:
|V|=6
+1 for unknown

Estimating probabilities: P(*w* | spam)

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

spam

	unknown	buy	catfood	eggs	free	sale	viagra	
0	0	0	0	0	0	2	1	3
1	0	0	0	0	4	0	3	7

wordcount(spam) = 10

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

Estimating probabilities: P(*w* | spam)

- 2nd, get total count for each word in spam docs, wordcount(w,spam)

spam

	unknown	buy	catfood	eggs	free	sale	viagra
0	0	0	0	0	0	2	1
1	0	0	0	0	4	0	3
	$\frac{0}{10}$	$\frac{0}{10}$	$\frac{0}{10}$	$\frac{0}{10}$	$\frac{4}{10}$	$\frac{2}{10}$	$\frac{4}{10}$

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

Estimating probabilities: P(*w* | spam)

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

unknown	buy	catfood	eggs	free	sale	viagra
1	1	1	1	5	3	5

  - wordcount(spam)+|V|+1 = 10+6+1=17

P(*w* | spam) →

unknown	buy	catfood	eggs	free	sale	viagra
0.058824	0.058824	0.058824	0.058824	0.294118	0.176471	0.294118

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

UNIVERSITY OF SAN FRANCISCO

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

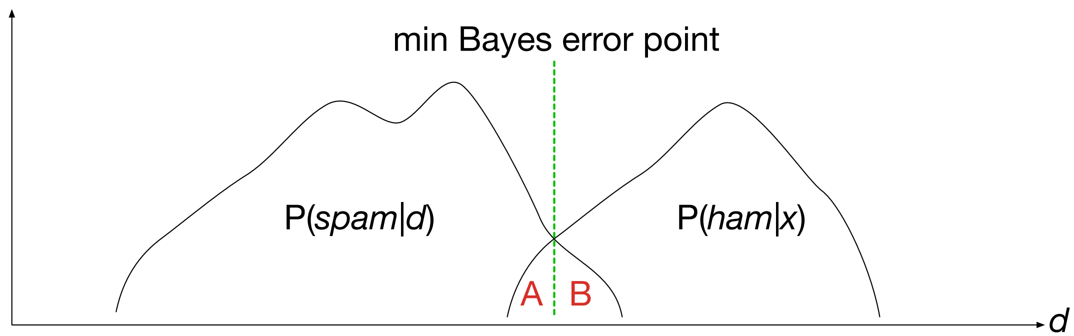
- Dot product of X matrix with log of P(w|c) vector, add log(P(c))

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

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

Quick idea on Bayes error

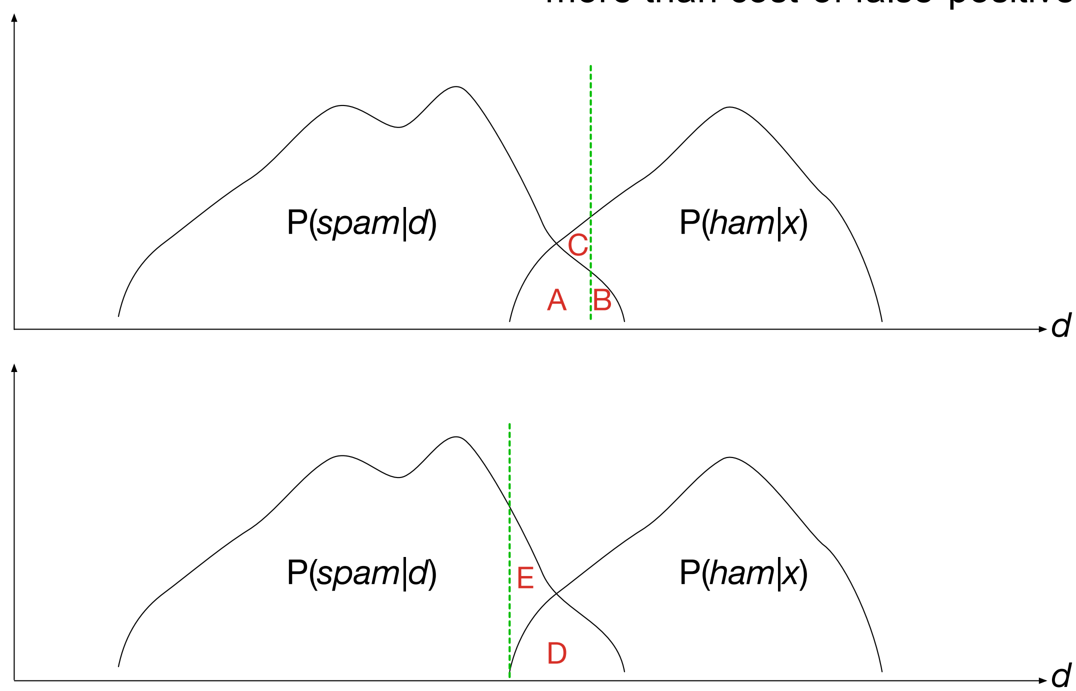
- Bayes error is the lowest possible total error from all classes
- Imagine squashing doc d down to single dimension
- Bayes error is area under P(*spam*|*d*) to right and P(*ham*|*d*) to left of decision point (A and B here)



Preview AUC ROC / PR curves

E.g., in disease testing,
might value catching disease
more than cost of false positive

- If our "utility function" values spam or ham differently, can shift boundary to left/right
- Later we'll examine ROC/PR curves that summarize effect of shifting boundary all over
- Tradeoff: to get 0 ham error, must accept many spam, area under P(*spam*|*d*) to right of threshold: D+E error



Key takeaways

- Naïve bayes is classifier often applied to text classification; e.g., spam/ham, topic labeling, etc…
- Less often used these days with rise of deep learning
- Fixed-length word-count 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: $$c^* = \underset{c}{argmax}\ P(c) \prod_{w \in V} P(w|c)^{n_w(d)}$$

Implementation takeaways

- Avoid vanishing floating-point values from product

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

- Avoid P(w|c)=0 via Laplace smoothing
  - add 1 to all word counts
  - adjust P(w|c) denominator with extra |V| since every doc now has every word
  - this is for missing words where *w* not in *d* but in *V*
- Treat unknown words as 1 / (count(c) + |V| + 1)