CS 4650/7650, Lecture 2
Supervised Classification

Jacob Eisenstein

August 22, 2013

1 Linear classification and features

Suppose you want to build a spam detector. Spam vs. Ham. How would you do it, (using only the text in the email)?

One solution is to represent document i as a column vector of word counts: $\boldsymbol{x}_i = [0\ 1\ 1\ 0\ 0\ 2\ 0\ 1\ 13\ 0 \ldots]^\mathsf{T}$, where $x_{i,j}$ is the count of word j in document i. Suppose the size of the vocabulary is M.

We've thrown out grammar, sentence boundaries, paragraphs — everything but the words! But this could still work. If you see the word *free*, is it spam or ham? How about *c1al1s*? How about *Bayesian*?

Now, suppose we want to build a multi-way classifier to distinguish stories about sports, celebrities, music, and business? Each label is an element y in a set of K possible labels $\mathcal{Y}$. Then for any pair $\langle \boldsymbol{x}_i, y_i \rangle$, we can define a *feature vector* $\boldsymbol{f}(x_i, y_i)$, such that:

$$\boldsymbol{f}(x, 0) = [\boldsymbol{x}_i\ \mathbf{0}_{M(K-1)}]^\mathsf{T} \tag{1}$$
$$\boldsymbol{f}(x, 1) = [\mathbf{0}_M\ \boldsymbol{x}_i\ \mathbf{0}_{M(K-2)}]^\mathsf{T} \tag{2}$$
$$\boldsymbol{f}(x, 2) = [\mathbf{0}_{2M}\ \boldsymbol{x}_i\ \mathbf{0}_{M(K-3)}]^\mathsf{T} \tag{3}$$
$$\ldots \tag{4}$$
$$\boldsymbol{f}(x, K) = [\mathbf{0}_{M(K-1)}\ \boldsymbol{x}_i]^\mathsf{T}, \tag{5}$$

where $\mathbf{0}_{MK}$ is a vector of MK zeros. Often we'll add an *offset* feature at the end of $\boldsymbol{x}$, which is always 1. This makes the length of the entire feature vector $(M+1)K$.

Now, given a vector of weights, $\boldsymbol{w} \in \mathcal{R}^{(M+1)K}$, we can compute the inner product $\boldsymbol{w}^\mathsf{T}\boldsymbol{f}(x, y)$. Then for any document $\boldsymbol{x}_i$, we can predict a label $\hat{y}$ as

$$\hat{y} = \arg\max_y \boldsymbol{w}^\mathsf{T}\boldsymbol{f}(\boldsymbol{x}_i, y) \tag{6}$$

We could just set the weights by hand. If we wanted to distinguish, say, English from Spanish, we could just use English and Spanish dictionaries, and

set each weight to 1. For example,

$$\begin{aligned} w_{\text{english,bicycle}} &= 1 & w_{\text{spanish,bicycle}} &= 0 \\ w_{\text{english,bicycleta}} &= 0 & w_{\text{spanish,bicycle}} &= 1 \\ w_{\text{english,con}} &= 1 & w_{\text{spanish,con}} &= 1 \\ w_{\text{english,ordinateur}} &= 0 & w_{\text{spanish,ordinateur}} &= 0 \end{aligned}$$

Similarly, if we want to distinguish positive and negative sentiment, we could use positive and negative *sentiment lexicons*. You'll do this in Project 1.

But it's usually not easy to set the weights by hand. Instead, we will learn them from data. An important tool for this is probability.

2 Review of basic probability

- Conditional probability: $P(x|y) = P(x,y)/P(y)$
- Chain rule: $P(x,y) = P(x|y)P(y)$
- We can apply the chain rule multiple times:

$$\begin{aligned} P(x,y,z) &= P(x,y|z)P(z) \\ &= P(x|y,z)P(y,z) \\ &= P(x|y,z)P(y|z)P(z) \\ P(x,y|z) &= P(x|y,z)P(y|z) \end{aligned}$$

- Bayes' rule follows from the Chain rule: $P(y|x) = P(x,y)/P(x) = P(x|y)P(y)/P(x)$

Often we want the maximum a posteriori (MAP) estimate

$$\begin{aligned} \hat{y} &= \arg\max_y P(y|x) \\ &= \arg\max_y P(x|y)P(y)/P(x) \\ &\propto \arg\max_y P(x|y)P(y) \end{aligned}$$

We don't need to normalize the probability because $P(x)$ is the same for all values of y.

Sometimes we want the **expectation** of a function, such as $E[g(x)|y] = \sum_{x \in \mathcal{X}} g(x)P(x|y)$. For this, we need $P(x|y)$ to be a normalized probability.

Expectations are easiest to think about in terms of probability distributions over discrete events:

- If it is sunny, Marcia will eat three ice creams.
- If it is rainy, she will eat only one ice cream.
- There's a 80% chance it will be sunny.

- The expected number of ice creams she will eat is $0.8 \times 3 + 0.2 \times 1 = 2.6$.

If the random variable X is continuous, the sum becomes an integral. For example, a fast food restaurant in Quebec gives a 1% discount on french fries for every degree below zero. Assuming they used a thermometer with infinite precision, the expected price would be an integral over all possible temperatures.

3 Naïve Bayes

A Naïve Bayes classifier chooses the weights $\boldsymbol{w}$ to maximize the *joint* probability $P(x, y)$. We first need to define this probability. We'll do that through a "generative model," which describes a hypothesized stochastic process that has generated the observed data.[1]

- For each document i,
  - draw the label $y_i \sim \text{Categorical}(\theta)$
  - draw the vector of counts $\boldsymbol{x}_i \sim \text{Multinomial}(\phi_{y_i})$

The first thing this generative model tells us is that we can treat each document independently: the probability of the whole dataset is equal to the product of the probabilities of each individual document.

$$P(x, y; \theta, \phi) = \prod_i P(x_i, y_i; \theta, \phi) \tag{7}$$

In other words, the documents are *conditionally independent* given the parameters θ and ϕ.

When we write $y_i \sim \text{Categorical}(\theta)$, that means y_i is a stochastic draw from a categorical distribution with parameter θ. A categorical distribution is just like a weighted die. A multinomial distribution is similar, but you draw a vector of counts.[2] By specifying the multinomial distribution, we are working with *multinomial naïve Bayes*.[3]

3.1 Distributions

A categorical distribution is very simple: $P_{\text{categorical}}(Y = y; \theta) = \theta_y$, where θ_y is the probability of the outcome $Y = y$.

[1] We'll see a lot of different generative stories in this course. They are a helpful tool because they clearly and explicitly define the assumptions that underly the model.

[2] Technically, a multinomial distribution requires a second parameter, the total number of counts (the number of words in the document). Even more technically, that number should be treated as a random variable, and drawn from some other distribution. But none of that matters for classification.

[3] You can plug in any probability distribution to the generative story and it will still be naïve Bayes, as long as you are making the "naïve" assumption that your features are generated independently. We'll talk about this later.

A multinomial distribution is only slightly more complex:

$$P_{\text{multinomial}}(x;\phi) = \frac{\left(\sum_j x_j\right)!}{\prod_j x_j!} \prod_j \phi_j^{x_j} \tag{8}$$

We require that $\sum_j \phi_j = 1$ and $\phi_j \geq 0$ for all j. The first part of the equation doesn't depend on ϕ, and can usually be ignored. Can you see why we need the first part at all?

Note that we write $P(x|y;\phi)$ to indicate the conditional probability of random variable x given y, with parameter ϕ. Parameters are not random variables, so we cannot write $P(\phi)$.

3.2 Prediction

The Naive Bayes prediction rule is to choose the label y which maximizes $P(x, y; \phi, \theta)$:

$$\begin{aligned}
\hat{y} =& \arg\max_y P(\boldsymbol{x}, y; \theta, \phi) \\
=& \arg\max_y P(\boldsymbol{x}|y;\phi) P(y;\theta) \\
=& \arg\max_y \log P(\boldsymbol{x}|y;\phi) + \log P(y;\theta)
\end{aligned}$$

Converting to logarithms makes the notation easier. It doesn't change the prediction rule because the log function is monotonically increasing.

Now we can plug in the probability distributions from the generative story.

$$\begin{aligned}
\log P(\boldsymbol{x}, y; \theta, \phi) =& \arg\max_y \log P(\boldsymbol{x}|y;\phi) + \log P(y;\theta) \\
=& \log \left[\frac{\left(\sum_j x_j\right)!}{\prod_j x_j!} \prod_j \phi_{y,j}^{x_j} \right] + \log \theta_y \\
=& \log \frac{\left(\sum_j x_j\right)!}{\prod_j x_j!} + \sum_j x_j \log \phi_{y,j} + \log \theta_y \\
\propto& \sum_j x_j \log \phi_{y,j} + \log \theta_y \\
=& \boldsymbol{w}^\mathsf{T} \boldsymbol{f}(\boldsymbol{x}, y),
\end{aligned}$$

where

$$\begin{aligned}
\boldsymbol{w} &= [\boldsymbol{w^{(1)}}\ \boldsymbol{w^{(2)}}\ \ldots\ \boldsymbol{w^{(K)}}] \\
\boldsymbol{w}^{(y)} &= [\log \phi_{y,1}\ \log \phi_{y,2}\ \ldots\ \log \phi_{y,M} \log \theta_y]
\end{aligned}$$

and $\boldsymbol{f}(\boldsymbol{x}, y)$ is a vector of word counts and an offset, padded by zeros for the labels not equal to y (see equations 1-5).

3.3 Estimation

The parameters of a multinomial distribution have a simple interpretation: they're the expected proportions for each word. Based on this interpretation, it's tempting to set the parameters empirically, as

$$\phi_{y,j} = \frac{\sum_{i:Y_i=y} x_{i,j}}{\sum_{j'} \sum_{i:Y_i=y} x_{i,j'}} = \frac{\text{count}(y,j)}{\sum_{j'} \text{count}(y,j')} \tag{9}$$

In NLP this is called a *relative frequency estimator*. It can be justified more rigorously as a *maximum likelihood estimate*.

As in prediction, our desiderata is to maximize the joint likelihood of the data,

$$L = \sum_i \log P_{\text{multinomial}}(\boldsymbol{x}_i; \boldsymbol{\phi}_{y_i}) + P_{\text{categorical}}(y_i; \theta) \tag{10}$$

Since $P(y)$ is unrelated to ϕ, we can forget about it. But before we can just optimize L, we have to deal with a constraint:

$$\sum_j \phi_{y,j} = 1 \tag{11}$$

We'll do this by adding a Lagrange multiplier. Here's the objective of the unconstrained optimization problem:

$$\ell[\boldsymbol{\phi}_y] = \sum_{i:Y_i=y} \sum_j x_{ij} \log \phi_{y,j} + \lambda(1 - \sum_j \phi_{y,j}) \tag{12}$$

We solve by setting $\frac{\partial \ell}{\boldsymbol{\phi}_j} = 0$.

$$\begin{aligned}
0 &= \sum_{i:Y_i=y} x_{i,j}/\phi_{y,j} - \lambda \\
\lambda\phi_{y,j} &= \sum_{i:Y_i=y} x_{i,j} \\
\phi_{y,j} &\propto \sum_{i:Y_i=y} x_{i,j} \\
&= \frac{\sum_{i:Y_i=y} x_{i,j}}{\sum_{j'} \sum_{i:Y_i=y} x_{i,j'}}
\end{aligned}$$

Similarly, $\theta_y \propto \sum_i \delta(Y_i = y)$, where $\delta(Y_i = y) = 1$ if $Y_i = y$ and 0 otherwise.

3.4 Smoothing and MAP estimation

If data is sparse, you can end up with values of $\phi = 0$, allowing a single feature to completely veto a label. This is undesirable, because it imposes high **variance**: depending on what data happens to be in the training set, we could get vastly different classification rules.

One solution is Laplace smoothing: adding "pseudo-counts" of α to each estimate, and then normalize.

$$\phi_{y,j} = \frac{\alpha + \sum_{i:Y_i=y} x_{i,j}}{\sum_{j'} \alpha + \sum_{i:Y_i=y} x_{i,j'}} = \frac{\alpha + \text{count}(i,j)}{V\alpha + \sum_{j'} \text{count}(i,j')} \tag{13}$$

Laplace smoothing has a nice Bayesian justification, in which we extend the generative story to include ϕ as a random variable (rather than as a parameter). The resulting estimate is called *maximum a posteriori*, or MAP.

Smoothing reduces **variance**, but it takes us away from the maximum-likelihood estimate: it imposes a **bias** (towards uniform probabilities). Machine learning theory shows that errors on held out data result from the sum of bias and variance. Techniques for reducing variance typically increase the bias, so there is a **bias-variance tradeoff**.

- Unbiased classifiers **overfit** the training data, yielding poor performance on unseen data.
- But if we set a very large smoothing value, we can **underfit** instead. In the limit of $\alpha \to \infty$, we have zero variance: it is the same classifier no matter what data we see! But the bias of such a classifier will be high.
- Navigating this tradeoff is hard. But in general, as you have more data, you can afford more variance.

3.5 Training, testing, and tuning (development) sets

We'll soon talk about more learning algorithms, but whichever one we apply, we will want to report its accuracy. Really, this is an educated guess about how well the algorithm will do on new data in the future.

To do this, we need to hold out a separate "test set" from the data that we use for estimation (i.e., training, learning). Otherwise, if we measure accuracy on the same data that is used for estimation, we will badly overestimate the accuracy we're likely to get on new data. See `http://xkcd.com/1122/` for a cartoon related to this idea.

Many learning algorithms also have "tuning" parameters:

- the smoothing pseudo-counts α in Naive Bayes
- the regularization λ in logistic regression
- the slack weight C in the Passive-Aggressive algorithm

All of these tuning parameters really do the same thing: they navigate the bias-variance tradeoff. Where is the best position on this tradeoff curve? It's hard to tell in advance. Sometimes it is tempting to see which tuning parameter gives the best performance on the test set, and then report that performance. Resist this temptation! It will also lead to overestimating accuracy on truly unseen future data. For that reason, this is a surefire way to get your research

paper rejected. Instead, you should split off a piece of your training data, called a "development set" (or "tuning set").

Sometimes, people average across multiple test sets and/or multiple development sets. One way to do this is to divide your data into "folds," and allow each fold to be the development set one time. This is called **K-fold cross-validation**. In the extreme, each fold is a single data point. This is called **leave-one-out**.

3.6 Pros and cons of Naive Bayes

Naive Bayes is very simple to work with. Estimation and prediction can done in closed form, and the nice probabilistic interpretation makes it relatively easy to extend the model in various ways.

But Naive Bayes makes assumptions which seriously limit its accuracy, especially in NLP.

- The multinomial distribution assumes that each word is generated indepently of all the others (conditioned on the parameter ϕ_y).
- But this is clearly wrong, because words travel together: if a document contains the word *naïve*, it is far more likely than average to also contain the word *Bayes*!
- Put another way, $P(\textit{naïve}, \textit{Bayes}) \neq P(\textit{naïve})P(\textit{Bayes})$.

Traffic lights Dan Klein makes this point with an example about traffic lights. In his hometown of Pittsburgh, there is a 1/7 chance that the lights will be broken, and both lights will be red. There is a 3/7 chance that the lights will work, and the north-south lights will be green; there is a 3/7 chance that the lights work and the east-west lights are green.

The *prior* probability that the lights are broken is 1/7. If they are broken, the conditional likelihood of each light being red is 1. The prior for them not being broken is 6/7. If they are not broken, the conditional likelihood of each being light being red is 1/2.

Now, suppose you see that both lights are red. According to Naive Bayes, the probability that the lights are broken is $1/7 \times 1 \times 1 = 1/7 = 4/28$. The probability that the lights are not broken is $6/7 \times 1/2 \times 1/2 = 6/28$. So according to naive Bayes, there is a 60% chance that the lights are not broken!

What went wrong? We have made an independence assumption to factor the probability $P(R, R|\text{not-broken}) = P_{\text{north-south}}(R|\text{not-broken})P_{\text{east-west}}(R|\text{not-broken})$. But this independence assumption is clearly incorrect, because $P(R, R|\text{not-broken}) = 0$.

Less Naive Bayes? Of course we could decide not to make the naive Bayes assumption, and model $P(R, R)$ explicitly. But this idea does not scale when the feature space is large (as it often is in NLP). The number of possible feature

configurations grows exponentially, so our ability to estimate accurate parameters will suffer from high variance. With an infinite amount of data, we'd be fine; but we never have that. Naive Bayes accepts some bias (because of the incorrect modeling assumption) in exchange for lower variance.