Naive Bayes and EM Algorithm

By Zach Wood-Doughty;
adapted from a somewhat similar writeup by Michael Collins

February 26, 2024: v1.2

1 Introduction and notation

This note will recap the Naive Bayes method and derive the maximum-likelihood estimate in the fully-supervised setting. Then we will introduce the expectation-maximization (EM) algorithm for the case where (some) labels are missing from the training data.

Suppose we want to use a dataset of email messages for the supervised learning task of classifying whether an email is spam. We'll represent our dataset of N examples as $\{X_i, y_i\}_{i=1}^{N}$. X_i is a vector of shape (1, V), where $X_{i,j}$ is a binary indicator of whether the jth word appears in email i, and V is the number of words in our vocabulary. This is sometimes referred to as a "bag-of-words" representation. y_i is the label for the corresponding email, which we will assume is either 1 (if spam) or 0 (if not). We'll assume there exists some (noisy) target function $f : X \to y$ such that $y_i = f(X_i)$, and our goal is to learn a hypothesis h such that $h(X) \approx f(X)$. We'll denote the parameters of our model as $\Theta = \{\alpha, \beta\}$. Our hypothesis $h(X)$ is uniquely determined these parameters, and our goal is to learn these parameters from the data such that $h(X)$ is as close as possible to $f(X)$.

2 The Naive Bayes model

Naive Bayes is a *generative* probabilistic model, meaning that it can model the entire distribution $p(X, y)$. This means we can use it both to predict the label of a test set example and sample new examples that should 'look similar' to the existing examples. To make it easier to distinguish two kinds of parameters in our model, we will write $\Theta = \{\alpha, \beta\}$. The α parameters define the overall probability of the labels, with $p(y_i = y) = \alpha_y$. The β parameters define the probability of ever seeing the jth word in a document with label y. So for a document with label y_i, we say $p(X_{i,j} = 1 \mid Y = y_i) = \beta_{j,y_i}$. By rules of probability, $p(X_{i,j} = 0 \mid Y = y_i) = (1 - \beta_{j,y_i})$

In our example of binary spam classification, $p(y_i)$ is the baseline probability that an email is spam, *without* looking at the text of the email. So if on average only 1% of emails are spam, we would want to learn $\alpha = [0.99, 0.01]$ such that $p(y_i = 1) = \alpha_1 = 0.01$ for any i.[1]

$p(X_{i,j} = 1 \mid Y = y_i; \beta)$ is the probability of jth word being used at least once in a document with label y_i. Because in this assignment we'll assume that y is binary, β is a matrix of shape $[V, 2]$ (where V is again the size of the vocabulary).[2] For example, if "jackpot" is word 37 and appears once in every four spam emails but only once in every ten non-spam emails, then $\beta_{37} = [0.1, 0.25]$.

[1] Note that it's possible to represent α as a single parameter rather than a vector of two parameters that sum to one. The homework test cases will expect you to represent this as a vector of two parameters; this will also make easier to use numpy's matrix multiplication operations.

[2] If y were not binary and instead had K labels, then α would be an array of K values that sum to 1, and β would be a matrix of shape $[V, K]$.

Our goal is to find the values of α and β that *best explain* the data that we have. Remember from lecture that we wrote the derivation for finding the maximum-likelihood Θ^* as:

$$\begin{aligned}
\Theta^* &= \arg\max_{\Theta} p(\Theta \mid X, y) \\
&= \arg\max_{\Theta} \frac{p(X, y \mid \Theta)p(\Theta)}{p(X, y)} \\
&= \arg\max_{\Theta} p(X, y \mid \Theta)p(\Theta) \qquad (1) \\
&= \arg\max_{\Theta} p(X, y \mid \Theta) \qquad (2)
\end{aligned}$$

In (2), the $p(X, y \mid \Theta)$ term is referred the *likelihood* of the data, or the probability of seeing this particular dataset given our current parameters for the model.[3] While we will want to use our model to *predict* Y_i from X_i using the conditional distribution $p(Y_i \mid X_i)$, our model parameters α and β represent $p(Y_i)$ and $p(X_i \mid Y_i)$.

Because X_i is a vector with V values, writing out the entire conditional distribution would require $\mathcal{O}(2^V)$ parameters. For a reasonably large vocabulary of thousands of words, this is impossible. Instead, we will drastically simplify the problem by assuming that the occurrences of the jth word are *independent* of occurrences of all other words, conditional on the label y. This is an unrealistic (*naive*) assumption; for example, we might expect the word "win" to co-occur with the word "jackpot" in both spam and non-spam emails. The Naive Bayes assumption says no: $p(X_{i,j}, X_{i,j'} \mid Y = y_i) = p(X_{i,j} \mid Y = y_i)p(X_{i,j'} \mid Y = y_i)$ for all words j and j'. In plain language, we're assuming that whether "jackpot" appears in a document depends *only* on whether the email is spam or not; it does not depend on any of the other words in the email. This assumption allows us to write:

$$p(X_{i,1}, X_{i,2}, \ldots X_{i,V} \mid Y_i) = \prod_{j=1}^{V} p(X_{i,j} \mid Y_i) \qquad (3)$$

Note that our data matrix X is binary. That is, $X_{i,j}$ is one if document i contains word j and otherwise is zero. Even if the word j shows up a thousand times in that document, $X_{i,j}$ is still just 1. However, when we are calculating the probability of the document, words that do not appear in that document still contribute to its probability. If $X_{i,j} = 0$, we use $p(X_{i,j} = 0 \mid Y = y_i)$ to quantify how likely it is that we would see a document with label y_i that does not contain word j. We can use this to write:

$$\begin{aligned}
& p(X_{i,j} \mid Y = y_i, \beta) \\
&\qquad = p(X_{i,j} = 1 \mid Y = y_i, \beta)^{(X_{i,j})} p(X_{i,j} = 0 \mid Y = y_i, \beta)^{(1 - X_{i,j})} \qquad (4) \\
\Rightarrow\ & \log p(X_{i,j} \mid Y = y_i, \beta) \\
&\qquad = \log p(X_{i,j} = 1 \mid Y = y_i, \beta)^{(X_{i,j})} + \log p(X_{i,j} = 0 \mid Y = y_i, \beta)^{(1 - X_{i,j})} \\
&\qquad = (X_{i,j}) \log p(X_{i,j} = 1 \mid Y = y_i, \beta) + (1 - X_{i,j}) \log p(X_{i,j} = 0 \mid Y = y_i, \beta) \\
&\qquad = X_{i,j} \log \beta_{j,y_i} + (1 - X_{i,j}) \log(1 - \beta_{j,y_i}) \qquad (5)
\end{aligned}$$

[3] In this derivation, (1) is the maximum a posteriori (MAP) estimate for Θ^*. By dropping the prior (the $p(\Theta)$ term), we get the maximum likelihood estimate, which assumes that all Θ values are equally likely. For this coding assignment, we will only consider maximum likelihood, not MAP.

We can put this all together to connect $\Theta^* = \{\alpha^*, \beta^*\}$ for our dataset of N examples as follows:

$$\begin{aligned}
\{\alpha^*, \beta^*\} &= \arg\max_{\alpha,\beta} \prod_{i=1}^{N} p(X_i, y_i \mid \alpha, \beta) \\
&= \arg\max_{\alpha,\beta} \prod_{i=1}^{N} p(y_i \mid \alpha) p(X_i \mid y_i, \beta) \\
&= \arg\max_{\alpha,\beta} \log \prod_{i=1}^{N} p(y_i \mid \alpha) p(X_i \mid y_i, \beta) \qquad (6) \\
&= \arg\max_{\alpha,\beta} \sum_{i=1}^{N} (\log p(y_i \mid \alpha) + \log p(X_i \mid y_i, \beta)) \\
&= \arg\max_{\alpha,\beta} \sum_{i=1}^{N} \log p(y_i \mid \alpha) + \sum_{i=1}^{N} \log p(X_i \mid y_i, \beta) \\
&= \arg\max_{\alpha,\beta} \sum_{i=1}^{N} \log p(y_i \mid \alpha) + \sum_{i=1}^{N} \sum_{j=1}^{V} \log p(X_{i,j} \mid y_i, \beta) \\
&= \arg\max_{\alpha,\beta} \sum_{i=1}^{N} \log \alpha_{y_i} + \sum_{i=1}^{N} \sum_{j=1}^{V} (X_{i,j} \log \beta_{j,y_i} + (1 - X_{i,j}) \log(1 - \beta_{j,y_i})) \qquad (7) \\
&= \arg\max_{\alpha,\beta} \mathcal{L}(X, y, \alpha, \beta)
\end{aligned}$$

We'll use $\mathcal{L}(X, y, \alpha, \beta)$ to denote the log likelihood of the data and our specific parameters α and β. Fitting our model will be the process of finding $\arg\max_{\alpha,\beta} \mathcal{L}$. Note that (6) holds because log is monotonic (i.e., $\arg\max_x f(x) = \arg\max_x \log f(x)$). Equation (7) holds by plugging in (5).

Once we have learned α^* and β^*, we'll want to be able to use it to predict if an email is spam. This can be written as $p(y_i \mid X_i)$, or "what is the probability that the label y_i is 1 or 0 given that the email's text is X_i?" You might have noticed that we our parameters α and β aren't defined in terms of the probability distribution $p(y_i \mid X_i)$, but rather $p(y_i)$ and $p(X_{i,j} \mid y_i)$. This is because it's easier to track parameters in this way. We can write the probability that a document with text X_i has label k as:

$$p(Y = k \mid X_i) = \frac{p(Y = k|\alpha) p(X_i \mid Y = k, \beta)}{p(X_i)} = \frac{p(k|\alpha) p(X_i \mid k, \beta)}{\sum_{y'=1}^{2} p(y'|\alpha) p(X_i \mid y', \beta)} \qquad (8)$$

$$\begin{aligned}
\log p(Y = k \mid X_i) &= \log p(k|\alpha) + \log p(X_i \mid k, \beta) - \log \left(\sum_{y'=1}^{2} p(X_i \mid y', \beta) p(y'|\alpha) \right) \qquad (9) \\
&\propto \log p(k|\alpha) + \log p(X_i \mid k, \beta) \qquad (10) \\
&= \log p(Y = k|\alpha) + \sum_{j=1}^{V} \log p(X_{i,j} \mid k, \beta) \\
&= \log \alpha_k + \sum_{j=1}^{V} (X_{i,j} \log \beta_{j,k} + (1 - X_{i,j}) \log(1 - \beta_{j,k})) \qquad (11)
\end{aligned}$$

Before equation (10), the $\propto$ symbol means "is proportional to"; this is true because the summation inside the log term in (9) is the same for all X_i, regardless of the label being predicted. In your implementation, you can calculate $\log p(Y = k \mid X_i)$ using (11) for both possible values of y_i, and then use your `softmax` function to turn these unnormalized log probabilities into probabilities between 0 and 1.

Hint: if you want to make these calculations maximally efficient, you can calculate $\sum_{j=1}^{V} X_{i,j} \log \beta_{j,k}$ and $\sum_{j=1}^{V}(1-X_{i,j}) \log(1-\beta_{j,k})$ each as separate matrix multiplications. The `flip_bits_sparse_matrix` function we provide in the starter code will make it easy to compute $1 - X_{i,j}$.

3 Fully-supervised Naive Bayes updates

Remember that our dataset looks like $\{X_i, y_i\}_{i=1}^{N}$ and we've thus far assumed that every X_i has a corresponding y_i. In this setting, how do we learn good values for α and β? It turns out it is a reasonably simple process of counting:

$$\alpha_y = \frac{1}{N} \sum_{i=1}^{N} \mathbb{1}(y_i = y) \tag{12}$$

$$\beta_{j,y} = \frac{\sum_{i=1}^{N} X_{i,j} \mathbb{1}(y_i = y)}{\sum_{i=1}^{N} \mathbb{1}(y_i = y)} \tag{13}$$

Note that $\mathbb{1}(Z)$ is an *indicator* function which takes the value 1 if Z is true and 0 if Z is false. In plain language, we can read our estimate for α_y as, "among the dataset we have, what proportion of documents have label y?" We can similarly read $\beta_{j,y}$ as "among all documents we have with label y, what proportion contain at least one occurrence of the word j?"

3.1 Smoothing

Especially when we have a small number of documents or a large vocabulary size, there may be some combinations of y and j such that we never see word j in a document with label y. In that setting, we learn that $\beta_{j,y} = 0$ and thus $\log \beta_{j,y} = -\inf$. In plain language, we learn that it is *impossible* for word j to appear in a document with label y. If we ever see a document with word j, our model says it must have a 0% chance of having label y. This is bad; we don't want to draw such a strong conclusion from limited evidence. Equally bad is if every document with label y contains word j: our model will learn that it is impossible for a document with label y to *not contain* word j.

To avoid these issues, we introduce *smoothing*. Let λ be the amount of smoothing. We can replace (13) with the following:

$$\beta_{j,y} = \frac{\lambda + \sum_{i=1}^{N} X_{i,j} \mathbb{1}(y_i = y)}{2\lambda + \sum_{i=1}^{N} \mathbb{1}(y_i = y)} \tag{14}$$

Essentially, smoothing pretends that for each possible label (i.e., spam or not spam), we see 2λ new documents. In λ of those, our imagined documents have *every possible word.* In the other λ, our imagined documents have *no words.* Setting $\lambda > 0$ is sufficient to ensure that our model never believes that any event is impossible. As $\lambda \to \infty$, our β values will converge to 0.5 for all words.

4 Fully-unsupervised Naive Bayes

Suppose we have no labels for our documents, but we want to use these same definitions of α and β. Instead of y being the observed labels, we will treat them as unobserved (latent) variables that define clusters which separate the data into groups. How does this change our derivation from (7)? For each example in our dataset, we need to consider the probability it has label y', for all possible values of y'! We can write our *unsupervised* log likelihood as:

$$
\begin{aligned}
&\mathcal{L}(X,\alpha,\beta) \\
&\quad = \log \prod_{i=1}^{N} p(X_i \mid \alpha,\beta) \\
&\quad = \log \prod_{i=1}^{N} \sum_{y'=1}^{2} p(X_i, y_i = y' \mid \alpha,\beta) \\
&\quad = \log \prod_{i=1}^{N} \sum_{y'=1}^{2} p(y_i = y' \mid X_i,\alpha,\beta) p(y' \mid \alpha) \prod_{j=1}^{V} p(X_{i,j} \mid y',\beta) && (15) \\
&\quad = \sum_{i=1}^{N} \log \sum_{y'=1}^{2} p(y' \mid X_i,\alpha,\beta) p(y' \mid \alpha) \prod_{j=1}^{V} p(X_{i,j} \mid y',\beta) \\
&\quad = \sum_{i=1}^{N} \log \sum_{y'=1}^{2} \exp\log \left(p(y' \mid X_i,\alpha,\beta) p(y' \mid \alpha) \prod_{j=1}^{V} p(X_{i,j} \mid y',\beta) \right) && (16) \\
&\quad = \sum_{i=1}^{N} \log \sum_{y'=1}^{2} \exp \left(\log p(y' \mid X_i,\alpha,\beta) + \log p(y' \mid \alpha) + \sum_{j=1}^{V} \log p(X_{i,j} \mid y',\beta) \right) \\
&\quad = \sum_{i=1}^{N} \log \sum_{y'=1}^{2} \exp \left(\log p(y' \mid X_i,\alpha,\beta) + \log \alpha_{y'} + \sum_{j=1}^{V} [X_{i,j} \log \beta_{j,y'} + (1 - X_{i,j}) \log(1 - \beta_{j,y'})] \right) && (17)
\end{aligned}
$$

If you compare (17) against (7), you should see many similarities. However, there are important differences. First, we cannot distribute the log inside the $\sum_{y'=1}^{2}$. To maintain similarity to our previous derivation, we will add in an awkward $\exp\log$ in (16). This means we have an exp() function inside this equation, that we cannot get rid of. If V is large, the summation over V in (17) may be a large negative number, and exponentiating it may result in underflow. You will solve this problem in the `stable_log_sum` function you write in `src/utils.py`.

A second crucial difference is that our y_i variables are latent, and so we will estimate them using our $p(y_i \mid X_i, \alpha, \beta)$. You can connect this back to (7) by imagining that when we are given a fully-supervised dataset, that $p(y_i = y' \mid X_i, \alpha, \beta)$ is either 0 or 1 depending on whether y_i is in fact y' or not. When our y_i labels are latent, we need to predict a distribution over these y' labels. The challenge this introduces is that our likelihood now has three terms in it that are 'trapped' together inside the exp function: two depend on α, and two depend on β. This means we cannot compute derivatives directly, and will need to turn to the EM algorithm to optimize α and β.

We see in lecture that the general form of the EM algorithm involves alternating between two steps: first we will find the *expectation* of the unobserved values given our current model parameters, then we will pretend those are the true values and use them to find (*maximize* for) better model parameters. We saw Gaussian Mixture Models (GMMs) as an example of the EM algorithm, where the first ("E") step involved assigning each data point to a cluster based on the current location of the clusters, and the second ("M") step involved updating the parameters of each cluster based on the points assigned to it.

For fully-unsupervised Naive Bayes, we will take a similar approach. We start by initializing our model parameters somehow, and then in the E-step we use those parameters to predict $p(y_i \mid X_i, \alpha, \beta)$ for every example i. These predicted probabilities are our latent variables. In the M-step, we treat those predicted probabilities as fixed, and use them to update our α and β values. We keep iterating between E and M steps until the algorithm converges. The algorithm is described in Figure 1.

Inputs:

- A dataset $\{X_i\}_{i=1}^{N}$, where X has shape $[N, V]$.
- A value K of the number of labels to consider; in this assignment, $K = 2$.
- A maximum number of iterations T.

Initialization:

- Initialize $\alpha^0 = p(y_i = y') = \frac{1}{K}$ for all i and y'.
- Initialize $\beta^0 = p(X_{i,j} \mid y_i = y') = \frac{1}{2}$ for all i, j, and y'.

Algorithm:
For $t = 0 \ldots T$ or until convergence,

E-step: For all i and y', compute:

$$p(y_i = y' \mid X_i, \alpha^t, \beta^t) \propto p(X_i \mid y_i = y', \beta^t) p(y_i = y' \mid \alpha^t) \tag{18}$$

M-step: For all y' and j, compute:

$$\alpha_{y'}^{t+1} = \frac{1}{N} \sum_{i=1}^{N} p(y_i = y' \mid X_i, \alpha^t, \beta^t) \tag{19}$$

$$\beta_{j,y'}^{t+1} = \frac{\lambda + \sum_{i=1}^{N} X_{i,j}\, p(y_i = y' \mid X_i, \alpha^t, \beta^t)}{2\lambda + \sum_{i=1}^{N} p(y_i = y' \mid X_i, \alpha^t, \beta^t)} \tag{20}$$

Figure 1: The EM algorithm for fully-unsupervised Naive Bayes. Note that (18) is essentially identical to (8). Note the close similarity between (19) and (12) and between (20) and (13).

5 Semi-supervised Naive Bayes

Your homework considers a setting where we have *some* examples i for which we see y_i, which requires a slight modification. If we have a label y_i, use it; if not, predict it using $p(y_i = y' \mid X_i, \alpha^t, \beta^t)$. It is crucially important **not to overwrite** the true y_i with our E-step predictions. At each time t, $p(y_i = y' \mid X_i, \alpha^t, \beta^t)$ will *either* be a prediction based on our current α^t and β^t, or it will be the true y_i labels that were given to us at the beginning. Use the semi-supervised algorithm in Figure 2 to guide your implementation of `fit()` in `src/naive_bayes_em.py`. This is almost identical to the unsupervised model, except (21) and (22) replace (18). If we have a label y_i, we use it: saying $p(y_i = y' \mid X_i, \alpha, \beta)$ is 1 if y_i is equal to y', or else 0. For unlabeled examples, we infer a distribution over possible values y' and treat y_i as a latent variable.

Inputs:

- A dataset $\{X_i, y_i\}_{i=1}^N$, where X has shape $[N, V]$ and y has shape $[N, 1]$; each y_i can take a special value of "?" (i.e., `np.nan`) if the ith example has no label.
- A maximum number of iterations T.

Initialization:

- Initialize $\alpha^0 = p(y_i = y') = \frac{1}{K}$ for all i and y'.
- Initialize $\beta^0 = p(X_{i,j} \mid y_i = y') = \frac{1}{2}$ for all i, j, and y'.

Algorithm:
For $t = 0 \ldots T$ or until convergence,

E-step: For all i such that $y_i = ?$, for all y', compute:

$$p(y_i = y' \mid X_i, \alpha^t, \beta^t) \propto p(X_i \mid y_i = y', \beta^t) p(y_i = y' \mid \alpha^t) \tag{21}$$

Then, for all i such that $y_i \neq ?$, define

$$p(y_i = y' \mid X_i, \alpha^t, \beta^t) = \begin{cases} 1 & y' = y_i \\ 0 & \text{otherwise} \end{cases} \tag{22}$$

M-step: For all y' and j, compute:

$$\alpha_{y'}^{t+1} = \frac{1}{N} \sum_{i=1}^{N} p(y_i = y' \mid X_i, \alpha^t, \beta^t) \tag{23}$$

$$\beta_{j,y'}^{t+1} = \frac{\lambda + \sum_{i=1}^N X_{i,j}\, p(y_i = y' \mid X_i, \alpha^t, \beta^t)}{2\lambda + \sum_{i=1}^N p(y_i = y' \mid X_i, \alpha^t, \beta^t)} \tag{24}$$

Figure 2: The EM algorithm for semi-supervised Naive Bayes. Note that (19) is identical to (23) and (20) is identical to (24), except that $p(y_i \mid X_i, \alpha^t, \beta^t)$ is defined differently in the E-step.