Will Monroe
CS 109

Lecture Notes #21
August 11, 2017

Naïve Bayes

Based on a chapter by Chris Piech

At this point in the class we are ready to dive into the field of machine learning. Machine learning is the subfield of computer science that gives computers the ability to perform tasks without being explicitly programmed. There are several different tasks that fall under the domain of machine learning and several different algorithms for "learning". In this lecture we are going to focus on classification, starting with a classic classification algorithm, Naïve Bayes.

Classification

In classification tasks, your job is to build a function $\hat{Y} = g(\mathbf{X})$ that takes in a vector of **features** $\mathbf{X}$ (also called "inputs") and predicts a **label** Y (also called the "class" or "output"). Features are things you know, and the label is what your algorithm is trying to figure out; for example, the label might be a binary variable indicating whether an animal is a cat or a dog, and the features might be the length of the animal's whiskers, the animal's weight in pounds, and a binary variable indicating whether the animal's ears stick up or are droopy. Your algorithm needs to tell dogs and cats apart (Y) using only this information about weight, whiskers, and ears ($\mathbf{X}$).

A natural way to define this function is to predict the label with the highest conditional probability: choose $g(\mathbf{X}) = \arg\max_y P(Y = y|\mathbf{X})$. To help with computing the probabilities $P(Y = y|\mathbf{X})$, you are able to use training data consisting of examples of feature–label pairs $(\mathbf{X}, Y)$. You are given N of these pairs: $(\mathbf{x}^{(1)}, y^{(1)}), (\mathbf{x}^{(2)}, y^{(2)}), \ldots, (\mathbf{x}^{(N)}, y^{(N)})$, where $\mathbf{x}^{(i)}$ is a vector of m discrete features for the ith training example and $y^{(i)}$ is the discrete label for the ith training example.

In classification, Y takes on discrete values. (The alternative is **regression**, in which your algorithm is trying to predict a continuous value.) For this class we are going to assume that all values in our training dataset are binary. While this is not a necessary assumption (both Naïve Bayes and next lecture's logistic regression algorithm can work for non-binary data), it makes it much easier to learn the core concepts. Specifically, we assume that all labels are binary ($\forall i.\ y^{(i)} \in \{0, 1\}$) and all features are binary ($\forall i, j.\ x_j^{(i)} \in \{0, 1\}$).

Naïve Bayes algorithm

We'll first present the Naïve Bayes algorithm, and then we will show the theory behind it.

Training

The objective in training is to estimate the probabilities $P(Y)$ and $P(X_j|Y)$ for all $2 \geq j \leq m$ features.

Using an MLE estimate:

$$\hat{p}(X_j = x_j|Y = y) = \frac{(\text{\# training examples where } X_j = x_j \text{ and } Y = y)}{(\text{training examples where } Y = y)}$$

Using a Laplace MAP estimate:

$$\hat{p}(X_j = x_j|Y = y) = \frac{(\text{\# training examples where } X_j = x_j \text{ and } Y = y) + 1}{(\text{training examples where } Y = y) + 2}$$

Prediction

For an example with $\mathbf{x} = [x_1, x_2, \ldots, x_m]$, estimate the value of y as:

$$\hat{y} = g(\mathbf{X}) = \arg\max_y \hat{P}(Y)\hat{P}(\mathbf{X}|Y) \qquad \text{this is equal to } \arg\max_y \hat{P}(Y = y|\mathbf{X})$$

$$= \arg\max_y \hat{p}(Y = y) \prod_{j=1}^{m} \hat{p}(X_j = x_j|Y = y) \qquad \text{Naïve Bayes assumption}$$

$$= \arg\max_y \left(\log \hat{p}(Y = y) + \sum_{j=1}^{m} \log \hat{p}(X_j = x_j|Y = y) \right) \qquad \text{log version for numerical stability}$$

Note that for small enough datasets you may not need to use the log version of the argmax.

Theory

We can solve the classification task using a brute force solution. To do so we will learn the full joint distribution $\hat{P}(Y, \mathbf{X})$. Again, in the world of classification, when we make a prediction, we want to chose the highest-probability value of Y given $\mathbf{X}$: $g(\mathbf{X}) = \underset{y}{\text{argmax}}\ \hat{P}(Y = y|\mathbf{X})$.

$$\hat{y} = g(\mathbf{x}) = \arg\max_y \hat{P}(Y|\mathbf{X}) = \arg\max_y \frac{\hat{P}(\mathbf{X}, Y)}{\hat{P}(\mathbf{X})} \qquad \text{By definition of conditional probability}$$

$$= \arg\max_y \hat{P}(\mathbf{X}, Y) \qquad \text{Since } \hat{P}(\mathbf{X}) \text{ is constant with respect to Y}$$

Using our training data, we could interpret the joint distribution of $\mathbf{X}$ and Y as one giant multinomial with a different parameter for every combination of $\mathbf{X} = \mathbf{x}$ and $Y = y$. If, for example, the input vectors are only length one—in other words, $\mathbf{X}$ = some scalar X—and the number of values that X and Y can take on are small (binary, perhaps), then this is a totally reasonable approach. We could estimate the multinomial using MLE or MAP estimators and then calculate the arg max with a few lookups to our table.

The bad times hit when the number of features becomes large. Recall that our multinomial needs to estimate a parameter for every unique combination of assignments to the vector $\mathbf{X}$ and the value Y. If there are $|\mathbf{X}| = n$ binary features, then this strategy is going to take $O(2^n)$ space and there will likely be many parameters that are estimated without any training data that matches the corresponding assignment.

Naïve Bayes Assumption

The Naïve Bayes Assumption is that each feature of $\mathbf{X}$ is independent of the others given Y. That assumption is wrong, but useful. It allows us to make predictions using space and data which is linear with respect to the size of the features: $O(n)$ if $|\mathbf{X}| = n$. That allows us to train and make

predictions for huge feature spaces, such as a set of features consisting of an indicator for every word on the internet. Using this assumption simplifies the prediction algorithm.

$$\begin{aligned}
\hat{y} = g(\mathbf{x}) &= \arg\max_y \hat{P}(\mathbf{X}, Y) && \text{as we last left off} \\
&= \arg\max_y \hat{P}(Y)\hat{P}(\mathbf{X}|Y) && \text{by chain rule} \\
&= \arg\max_y \hat{P}(Y)\prod_{j=1}^{m}\hat{p}(X_j|Y) && \text{using the Naïve Bayes assumption} \\
&= \arg\max_y \left(\log \hat{p}(Y=y) + \sum_{j=1}^{m}\log \hat{p}(X_j = x_j|Y=y)\right) && \text{log version for numerical stability}
\end{aligned}$$

This algorithm is both fast and stable both when training and making predictions. If we think of each X_i, Y pair as a multinomial, we can find MLE and MAP estimations for the values. See the "algorithm" section for the optimal values of each p in the multinomial.

Naïve Bayes is a simple form of a field of machine learning called Probabilistic Graphical Models. In that field you make a graph of how your variables are related to one another and you come up with conditional independence assumptions that make it computationally tractable to estimate the joint distribution. Take CS 228 if you are interested in learning more!

Example

Say we have thirty examples of people's preferences (like or not) for the movies Star Wars, Harry Potter and Lord of the Rings. Each training example has x_1, x_2 and y where x_1 is whether or not the user liked Star Wars, x_2 is whether or not the user liked Harry Potter and y is whether or not the user liked Lord of the Rings.

For the 30 training examples the MLE estimates are as follows:

	counts		MLE $\hat{P}(X_1 \mid Y)$	
Y \ X_1	0	1	0	1
0	3	10	$\frac{3}{13}$	$\frac{10}{13}$
1	4	13	$\frac{4}{17}$	$\frac{13}{17}$

	counts		MLE $\hat{P}(X_2 \mid Y)$	
Y \ X_2	0	1	0	1
0	5	8	$\frac{5}{13}$	$\frac{8}{13}$
1	7	10	$\frac{7}{17}$	$\frac{10}{17}$

Y	counts	MLE $\hat{P}(Y)$
0	13	$\frac{13}{30}$
1	17	$\frac{17}{30}$

For a new user who likes Star Wars ($x_1 = 1$) but not Harry Potter ($x_2 = 0$) do you predict that they will like Lord of the Rings? We can compute this from the Naïve Bayes formula:

$$\hat{y} = \arg\max_y \left(\log \hat{p}(Y=y) + \sum_{j=1}^{m}\log \hat{p}(X_j = x_j|Y=y)\right)$$

This means plugging in each of the two different values for y:

$$\begin{aligned}
\text{For } y = 0\text{:}\quad & \log \hat{p}(Y = 0) + \sum_{j=1}^{m} \log \hat{p}(X_j = x_j | Y = 0) \\
&= \log \hat{p}(Y = 0) + \log \hat{p}(X_1 = 1 | Y = 0) + \log \hat{p}(X_2 = 0 | Y = 0) \\
&= \log 13/30 + \log 10/13 + \log 5/13 \approx -2.05 \\
\text{For } y = 1\text{:}\quad & \log \hat{p}(Y = 1) + \sum_{j=1}^{m} \log \hat{p}(X_j = x_j | Y = 1) \\
&= \log \hat{p}(Y = 1) + \log \hat{p}(X_1 = 1 | Y = 1) + \log \hat{p}(X_2 = 0 | Y = 1) \\
&= \log 17/30 + \log 13/17 + \log 7/17 \approx -1.72
\end{aligned}$$

-1.72 is larger (less negative) than -2.05, so our prediction is $\hat{y} = 1$: our new user is likely to like Lord of the Rings.

The above was done with maximum likelihood estimation; the process is the same for Laplace smoothing, but with different fractions for the conditional probabilities. The decision to smooth only the conditional probabilities $\hat{p}(X_j = x_j | Y = y)$ and not the prior $\hat{p}(Y = t)$ is due to the fact that Laplace smoothing makes a difference primarily when counts are small, while counts for the prior tend to be large enough that Laplace smoothing doesn't change the resulting probabilities much.

	counts		Laplace $\hat{P}(X_1 \mid Y)$	
Y \ X_1	0	1	0	1
0	3	10	$\frac{4}{15}$	$\frac{11}{15}$
1	4	13	$\frac{5}{19}$	$\frac{14}{19}$

	counts		Laplace $\hat{P}(X_2 \mid Y)$	
Y \ X_2	0	1	0	1
0	5	8	$\frac{6}{15}$	$\frac{9}{15}$
1	7	10	$\frac{8}{19}$	$\frac{11}{19}$

$$\begin{aligned}
\text{For } y = 0\text{:}\quad & \log \hat{p}(Y = 0) + \sum_{j=1}^{m} \log \hat{p}(X_j = x_j | Y = 0) \\
&= \log \hat{p}(Y = 0) + \log \hat{p}(X_1 = 1 | Y = 0) + \log \hat{p}(X_2 = 0 | Y = 0) \\
&= \log 13/30 + \log 11/15 + \log 6/15 \approx -2.06 \\
\text{For } y = 1\text{:}\quad & \log \hat{p}(Y = 1) + \sum_{j=1}^{m} \log \hat{p}(X_j = x_j | Y = 1) \\
&= \log \hat{p}(Y = 1) + \log \hat{p}(X_1 = 1 | Y = 1) + \log \hat{p}(X_2 = 0 | Y = 1) \\
&= \log 17/30 + \log 14/19 + \log 8/19 \approx -1.74
\end{aligned}$$

The results are barely different from the ones for maximum likelihood, and they bring us to the same conclusion: the user will probably like Lord of the Rings.