Lisa Yan
CS109

Lecture Notes #24
November 15, 2019

Naïve Bayes

Based on a chapter by Chris Piech

Naïve Bayes is a type of machine learning algorithm called a classifier. It is used to predict the probability of a discrete *label* random variable Y based on the state of *feature* random variables $\mathbf{X}$. We are going to learn all necessary parameters for the probabilistic relationship between $\mathbf{X}$ and Y from data. Naïve Bayes is a s*supervised classification* Machine Learning algorithm.

1 Machine Learning: Classification

In *supervised* machine learning, your job is to use training data with feature/label pairs $(\mathbf{I}, Y)$ in order to estimate a label-predicting function $\hat{Y} = g(\mathbf{X})$. This function can then be used to make future predictions. A *classification* task is one where Y takes on one of a **discrete** number of values. Often in classification, $g(\mathbf{X}) = \underset{y}{\text{argmax}} \hat{P}(Y = y|\mathbf{X} = \mathbf{x})$.

To learn all parameters required to to calculate $g(\mathbf{X})$, you are given n different training pairs known as **training data**: $(\mathbf{x}^{(1)}, y^{(1)}), (\mathbf{x}^{(2)}, y^{(2)}), \dots, (\mathbf{x}^{(n)}, y^{(n)})$. $\mathbf{X}^{(j)}$ is a vector of m discrete features for the ith training example, where $\mathbf{X}^{(j)} = (x_1^{(j)}, x_2^{(j)}, \dots, x_m^{(j)})$. $y^{(j)}$ is the discrete label for the ith training example. This symbolic description of classification hides the fact that prediction is applied to interesting real life problems:

1. Predicting heart disease Y, based on a set of m observations from a heart scan, $\mathbf{X}$.
2. Predicting ancestry Y, based on m DNA states $\mathbf{X}$.
3. Predicting if a user will like a movie Y given whether or not they like a set of m movies $\mathbf{X}$.

In this section we are going to assume that all random variables are binary. While this is not a necessary assumption (Naïve Bayes can work for non-binary data), it makes it much easier to learn the core concepts. Specifically, we assume that all labels are binary $y \in \{0, 1\}$, and all features are binary $x_j \in \{0, 1\}, \forall j = 1, \dots, m$.

2 Naïve Bayes algorithm

Here is the Naïve Bayes algorithm. After presenting the algorithm I am going to show the theory behind it.

Prediction

For an example with $\mathbf{X} = [x_1, x_2, \ldots, x_m]$, we can make a corresponding prediction for Y. We use hats (e.g., $\hat{P}$ or $\hat{Y}$) to symbolize values which are estimated.

$$\begin{aligned}
\hat{Y} = g(\mathbf{x}) &= \underset{y\in\{0,1\}}{\text{argmax}}\ \hat{P}(Y)\hat{P}(\mathbf{X}|Y) && \text{This is equal to argmax } \hat{P}(Y = y|\mathbf{X}) \\
&= \underset{y\in\{0,1\}}{\text{argmax}}\ \hat{P}(Y = y)\prod_{i=1}^{m}\hat{P}(X_i = x_i|Y = y) && \text{Naïve Bayes assumption} \\
&= \underset{y\in\{0,1\}}{\text{argmax}}\ \log\hat{P}(Y = y) + \sum_{i=1}^{m}\log\hat{P}(X_i = x_i|Y = y) && \text{Log version for numerical stability}
\end{aligned}$$

In order to calculate this expression, we are going to need to learn the estimates $\hat{P}(Y = y)$ and $\hat{P}(X_i = x_i|Y = y)$ from data using a process called **training**.

Training

The objective in training is to estimate the probabilities $P(Y)$ and $P(X_i|Y)$ for all $0 < i \leq m$ features. Using an MLE estimate:

$$\hat{P}(X_i = x_i|Y = y) = \frac{(\text{\# training examples where } X_i = x_i \text{ and } Y = y)}{(\text{training examples where } Y = y)}$$

Using a Laplace MAP estimate:

$$\hat{P}(X_i = x_i|Y = y) = \frac{(\text{\# training examples where } X_i = x_i \text{ and } Y = y) + 1}{(\text{training examples where } Y = y) + 2}$$

Estimating $P(Y = y)$ is also straightforward. Using MLE estimation:

$$\hat{P}(Y = y) = \frac{(\text{\# training examples where } Y = y)}{(\text{training examples})}$$

3 Theory

Now that you have the algorithm spelled out, let's go over the theory of how we got there. To do so, we will first explore an algorithm which doesn't work, called "Brute Force Bayes." Then, we introduce the Naïve Bayes Assumption, which will make our calculations possible.

Brute Force Bayes

We can solve the classification task using a brute force solution. To do so we will learn the full joint distribution $\hat{P}(Y, \mathbf{X})$.

In the world of classification, when we make a prediction, we want to chose the value of y that maximizes $P(Y = y|X = x)$. If we can only estimate $\hat{P}(Y = y|X = x)$, then we want to find a function $g(\mathbf{X}) = \underset{y}{\text{argmax}}\ \hat{P}(Y|\mathbf{X})$.

$$
\begin{aligned}
\hat{y} = g(\mathbf{x}) &= \underset{y \in \{0,1\}}{\text{argmax}}\ \hat{P}(Y|\mathbf{X}) && \text{Our objective} \\
&= \underset{y \in \{0,1\}}{\text{argmax}}\ \frac{\hat{P}(\mathbf{X}|Y)\hat{P}(Y)}{\hat{P}(\mathbf{X})} && \text{By Bayes' Theorem} \\
&= \underset{y \in \{0,1\}}{\text{argmax}}\ \hat{P}(\mathbf{X}|Y)\hat{P}(Y) && \text{Since } \hat{P}(\mathbf{X}) \text{ is constant with respect to Y}
\end{aligned}
$$

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 (i.e., $|\mathbf{X}| = 1$) and the number of values that x and y can take on are small—say, binary—this is a totally reasonable approach. We could estimate the multinomial using MLE or MAP estimators and then calculate argmax over a few lookups in 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 $|\mathbf{X}| = m$ binary features then this strategy is going to take order $\mathcal{O}(2^m)$ 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 conditionally independent of one another given Y. That assumption is naïve (and often wrong), but useful. This assumption allows us to make predictions using space and data which is linear with respect to the size of the features: $\mathcal{O}(m)$ if $|\mathbf{x}| = m$. That allows us to train and make predictions for huge feature spaces, such as one which has an indicator for every word on the internet. Using this assumption the prediction algorithm can be simplified:

$$
\begin{aligned}
\hat{y} = g(\mathbf{x}) &= \underset{y \in \{0,1\}}{\text{argmax}}\ \hat{P}(\mathbf{X}, Y) && \text{As we last left off} \\
&= \underset{y \in \{0,1\}}{\text{argmax}}\ \hat{P}(Y)\hat{P}(\mathbf{X}|Y) && \text{By chain rule} \\
&= \underset{y \in \{0,1\}}{\text{argmax}}\ \hat{P}(Y)\prod_{i=1}^{m}\hat{P}(X_i|Y) && \text{Using the Naïve Bayes assumption} \\
&= \underset{y \in \{0,1\}}{\text{argmax}}\ \log \hat{P}(Y) + \sum_{i=1}^{m}\log \hat{P}(X_i|Y) && \text{Log version for numerical stability}
\end{aligned}
$$

This algorithm is fast and stable both when training and making predictions.

Let us consider a particular feature, the i-th feature X_i. How should we represent $\hat{P}(X_i = x_i | Y = y)$? For a particular event $Y = y$ that we condition on, X_i can take on one of k discrete values . Thus for each particular y, we can model the likelihood of X_i taking on values as a Multinomial random variable with k parameters. We can then find MLE and MAP estimators for the parameters of that Multinomial. Recall that the MLE to estimate parameter p_i for a Multinomial is just counting, whereas the MAP estimator (with Laplace prior) to estimate parameter p_i imagines one extra example of each outcome:

$$\hat{p}_{i,MLE} = \frac{n_i}{n} \qquad \text{and} \qquad \hat{p}_{i,MAP} = \frac{n_i + 1}{n + k},$$

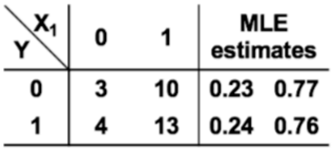
where n is the number of observations, n_i is the number of observations with outcome i, and k is the total possible number of outcomes k.

Note that in the version of classification we are using in CS109, X_i is binary (technically, a Multinomial with 2 parameters) and therefore $k = 2$. I used the Multinomial derivation to help you understand how one would handle a feature X_i that takes on multiple discrete values.

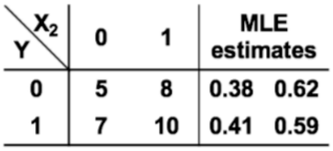
Naïve Bayes is a simple form of a Bayesian Network where the label Y is the only variable which directly influences the likelihood of each feature variable X_i. It is a simple model from a field of machine learning called Probabilistic Graphical Models. In that field you make a graph of how your variables are related to one another and you come up with conditional independence assumptions that make it computationally tractable to estimate the joint distribution.

Example

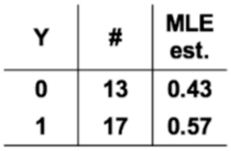
Say we have thirty examples of people's preferences (like or not) for Star Wars, Harry Potter and Pokemon. Each training example has X_1, X_2 and Y where X_1 is whether or not the user liked Star Wars, X_2 is whether or not the user liked Harry Potter and Y is whether or not the user liked Pokemon. For the 30 training examples, the MAP and MLE estimates are as follows:



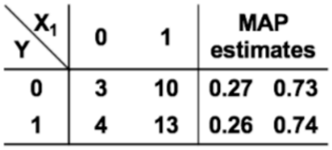
Y \ X1	0	1	MLE estimates	
0	3	10	0.23	0.77
1	4	13	0.24	0.76



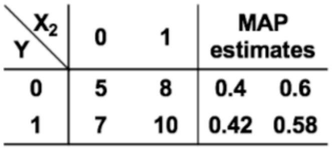
Y \ X2	0	1	MLE estimates	
0	5	8	0.38	0.62
1	7	10	0.41	0.59



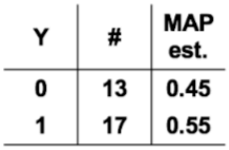
Y	#	MLE est.
0	13	0.43
1	17	0.57



Y \ X1	0	1	MAP estimates	
0	3	10	0.27	0.73
1	4	13	0.26	0.74



Y \ X2	0	1	MAP estimates	
0	5	8	0.4	0.6
1	7	10	0.42	0.58



Y	#	MAP est.
0	13	0.45
1	17	0.55

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 Pokemon? Yes! $Y = 1$ leads to a larger value in the argmax term:

$$\text{if } Y = 0: \hat{P}(X_1 = 1|Y = 0)\hat{P}(X_2 = 0|Y = 0)\hat{P}(Y = 0) = (0.77)(0.38)(0.43) \approx 0.126$$
$$\text{if } Y = 1: \hat{P}(X_1 = 1|Y = 1)\hat{P}(X_2 = 0|Y = 1)\hat{P}(Y = 1) = (0.76)(0.41)(0.57) \approx 0.178$$