Chris Piech
CS109

Handout #39
May 18th, 2016

Naïve Bayes

Classification

In classification tasks, your job is to use training data with feature label pairs ($\mathbf{x}$, y) in order to estimate a function $\hat{y} = g(\mathbf{x})$. This function can then be used to make a prediction. In classification the value of y takes on one of a discrete number of values. As such we often chose $g(\mathbf{x}) = \underset{y}{\text{argmax}}\ \hat{P}(Y = y|\mathbf{X})$.

Naïve Bayes algorithm

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

Data

Given N training pairs: $(\mathbf{x}^{(1)}, y^{(1)}), (\mathbf{x}^{(2)}, y^{(2)}), \dots, (\mathbf{x}^{(N)}, y^{(N)})$ 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.

Training

The objective in training is to estimate the probabilities $P(Y)$ and $P(X_i|Y)$ for all $0 < i \le m$ 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:

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

Prediction

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

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

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

$$\begin{aligned}
\hat{y} = g(\mathbf{x}) &= \operatorname*{argmax}_y \hat{P}(Y|\mathbf{X}) = \operatorname*{argmax}_y \frac{\hat{P}(\mathbf{X},Y)}{\hat{P}(\mathbf{X})} && \text{By definition of conditional probability} \\
&= \operatorname*{argmax}_y \hat{P}(\mathbf{X},Y) && \text{Since } \hat{P}(\mathbf{X}) \text{ is constant with respect to Y}
\end{aligned}$$

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. In other words $|\mathbf{x}| = 1$ 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 $|\mathbf{x}| = n$ binary features then this strategy is going to take order $\mathcal{O}(2^n)$ 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 one another given y. That assumption is 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: $\mathcal{O}(n)$ if $|\mathbf{x}| = n$. 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.

$$\begin{aligned}
\hat{y} = g(\mathbf{x}) &= \operatorname*{argmax}_y \hat{P}(\mathbf{X},Y) && \text{As we last left off} \\
&= \operatorname*{argmax}_y \hat{P}(\mathbf{X}|Y)\hat{P}(Y) && \text{By chain rule} \\
&= \operatorname*{argmax}_y \prod_{i=1}^{n} \hat{p}(X_i|Y)\hat{P}(Y) && \text{Using the naïve bayes assumption} \\
&= \operatorname*{argmax}_y \sum_{i=1}^{m} \log \hat{p}(X_i = x_i|Y = y) + \log \hat{p}(Y = y) && \text{Log version for numerical stability}
\end{aligned}$$

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.

Example

Say we have thirty examples of people's preferences (like or not) for 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 MAP and MLE estimates are as follows: For a new user who likes Star Wars

Y \ X₁	0	1	MLE estimates	
0	3	10	0.10	0.33
1	4	13	0.13	0.43

Y \ X₂	0	1	MLE estimates	
0	5	8	0.17	0.27
1	7	10	0.23	0.33

Y	#	MLE est.
0	13	0.43
1	17	0.57

($x_1 = 1$) but not Harry Potter ($x_2 = 0$) do you predict that they will like Lord of the Rings? See the lecture slides for the answer.