CS181 Lecture 18 — Naive Bayes and Autoclass

So far, we have discussed both parametric and non-parametric methods for density estimation. Non-parametric methods have the advantage that they make no prior assumptions on the shape of the probability density, but suffer from the curse of dimensionality. Parametric methods do make assumptions on the form of the density, and perform badly when those assumptions are incorrect, but perform very well when the assumptions are correct, even in high-dimensional spaces. We have also looked at two clustering algorithms, hierarchical agglomerative clustering and k-means.

Today, we consider another density estimator: the naive Bayes model. It forms the basis for both a classifier and a clustering algorithm called Autoclass.

1 The Naive Bayes Model

We have seen that a Gaussian density estimator makes strong assumptions about the shape of the density function, while non-parametric methods such as histograms suffer from the curse of dimensionality. One can view the naive Bayes model as a way of combining some of the advantages of each method. As far as each individual dimension is concerned, Naive Bayes behaves like a histogram. However, it avoids blowing up in high dimensions by assuming that all the attributes are *independent* of each other.

Formally, suppose we have attributes $X_1, \ldots, X_n$. These attributes may be Boolean, discrete, or continuous. We can treat a continuous attribute as discrete by dividing it into a finite number of ranges, and assuming a uniform density within each range. Let $\mathrm{Dom}(X_i)$ be the domain of attribute X_i. A *state* specifies an element in $\mathrm{Dom}(X_i)$ for each variable X_i.

For each attribute, we specify the density $p(X)$ explicitly. I.e., for a Boolean or discrete attribute, we explicitly provide the probability of each value. For a continuous attribute, we provide the probability of each range, and assume a uniform density throughout each range. By the independence assumption, we have

$$P(\mathbf{X}) = P(X_1, \ldots, X_n) = \prod_{i=1}^{n} P(X_i)$$

What have we achieved here? First, how many parameters are required to specify a general probability distribution of the form $P(X_1, \ldots, X_n)$, that is uniform within each state? The number of states is $|\mathbf{X}| = \prod_{i=1}^{n} |\mathrm{Dom}(X_i)|$,

and P specifies a probability for each $\mathbf{x}$ in $\mathbf{X}$. We have the constraint that $\sum_{\mathbf{x}} \mathbf{x} \in \mathbf{X} = 1$. So the number of independent parameters is actually $|\mathbf{X}| - 1$. In the case where all attributes are Boolean, the number of parameters is $2^n - 1$.

Now look at the naive Bayes density function. How many parameters does it have? We need only to specify the probability density for each attribute independently. We then assume that the attributes are independent of each other to get the joint probability of a state. For each attribute X_i, there are $\text{Dom}(X_i) - 1$ independent parameters, for a total of $\sum_{i=1}^{n} |\text{Dom}(X_i)| - n$. In the case where all attributes are Boolean, the number of parameters is just n.

Quite a contrast! Because the naive Bayes model has so few parameters, it scales well to high dimensions. Of course, this comes at the expense of considering a much more limited hypothesis space, which may not be able to accurately model the true distribution.

The naive Bayes model leads to a simple classifier, naturally called the *naive Bayes* classifier. For each class, we assume that the class conditional density function is a naive Bayes model. In other words, we assume that for each pair of features X_i and X_j, X_i is conditionally independent of X_j given the classification C. Mathematically, this means that

$$P(X_i, X_j \mid C) = P(X_i \mid C)P(X_j \mid C).$$

The joint distribution over the attributes and the class can be decomposed as follows:

$$\begin{aligned} P(X_1, \ldots, X_n, C) &= P(C)P(X_1, \ldots, X_n \mid C) \\ &= P(C) \prod_i P(X_i \mid C). \end{aligned}$$

Classification with a naive Bayes model is very simple. As we said earlier, we want to choose the class c that maximizes $P(c \mid x_1, \ldots, x_n)$. By Bayes rule,

$$P(c \mid x_1, \ldots, x_n) = \frac{P(c)P(x_1, \ldots, x_n|c)}{P(x_1, \ldots, x_n)} = \frac{P(c) \prod_{i=1}^{n} P(x_i|c)}{P(x_1, \ldots, x_n)}.$$

The denominator, which is equal to $\sum_c P(c)P(x_i, \ldots, x_n \mid c)$, is the same for all c. We call it a *normalizing factor* — it does not need to be computed in order to determine which class is the most likely. The most likely class is the one that maximizes $P(c) \prod_{i=1}^{n} P(x_i|c)$. This expression has two components. The first, $P(c)$, indicates the prior likelihood of the class c, without considering any of the feature values. The second component indicates the likelihood of the features given c.

In order to gain more understanding of the naive Bayes classifier, lets look at its hypothesis space. Let's just consider the case of n Boolean attributes and a Boolean class. We're going to show that a naive Bayes classifier is always a linear separator. We can write the parameters of the model as follows:

$$P(C = T) = \theta_C$$

$$\begin{aligned} P(C = F) &= 1 - \theta_C \\ P(X_i = T \mid C = T) &= \theta_i^T \\ P(X_i = F \mid C = T) &= 1 - \theta_i^T \\ P(X_i = T \mid C = F) &= \theta_i^F \\ P(X_i = F \mid C = F) &= 1 - \theta_i^F \end{aligned}$$

Now we can use a trick. If we encode T as 1 and F as 0, so x_i is either 0 or 1, we can write the following:

$$P(X_i = x_i \mid C = c) = (\theta_i^c)^{x_i}(1 - \theta_i^c)^{1-x_i}$$

Now, a naive Bayes model classifies x as true if

$$P(C = T)\prod_i P(X_i = x_i \mid C = T) \geq P(C = F)\prod_i P(X_i = x_i \mid C = F)$$

I.e.

$$\theta_C \prod_i (\theta_i^T)^{x_i}(1 - \theta_i^T)^{(1-x_i)} \geq (1 - \theta_C)\prod_i (\theta_i^F)^{x_i}(1 - \theta_i^F)^{(1-x_i)}$$

Now take logs:

$$\begin{array}{ll} \ln\theta_C & +\sum_i (x_i \ln\theta_i^T + (1 - x_i)\ln(1 - \theta_i^T)) \quad \geq \\ \ln(1 - \theta_C) & +\sum_i (x_i \ln\theta_i^F + (1 - x_i)\ln(1 - \theta_i^F)) \end{array}$$

Rearranging terms,

$$\begin{array}{l} \ln\theta_C + \sum_i \ln(1 - \theta_i^T) + \sum_i (\ln\theta_i^T - \ln(1 - \theta_i^T))x_i \geq \\ \ln(1 - \theta_C) + \sum_i \ln(1 - \theta_i^F) + \sum_i (\ln\theta_i^F - \ln(1 - \theta_i^F))x_i \end{array}$$

We get

$$w_0 + \sum_{i=1}^{n} w_i x_i \quad \geq \quad 0 \tag{1}$$

where

$$w_0 = \ln\theta_C + \sum_i \ln(1 - \theta_i^T) - [\ln(1 - \theta_C) + \sum_i \ln(1 - \theta_i^F)]$$

and

$$w_i = \ln\theta_i^T - \ln(1 - \theta_i^T) - [\ln\theta_i^F - \ln(1 - \theta_i^F)]$$

The important thing here is not the specific formulas for the weights, but the basic form of the model. Equation **??** looks just like the equation for a perceptron, with a rather fancy expression for the weights in terms of the model parameters. This shows that a naive Bayes classifier is just a linear separator! When we consider the independence assumption, that different attributes are conditionally independent of each other given the classification, this is not surprising. The assumption tells us that different attributes can only influence the classification individually — they cannot be combined in funky ways, as in XOR.

2 Learning the Naive Bayesian Model

Now let's consider learning a naive Bayes classifier. While the naive Bayes model is very simple, it is very easy to implement, very quick to train, and performs surprisingly well in many domains. Therefore, it is often used as a baseline against which to compare other learning algorithms.

We will use the maximum likelihood approach to estimate the parameters of the naive Bayes model. Recall that in the maximum likelihood method, we are given a data set $\mathbf{D}$ consisting of samples $\mathbf{x}^1, \ldots, \mathbf{x}^N$, and we wish to find values of the parameters θ that maximize the likelihood $P(\mathbf{D} \mid \theta)$. To make things simpler, we take logarithms, so we want to maximize the log likelihood

$$\mathcal{L}(\theta) = \ln P(\mathbf{D} \mid \theta) = \sum_{i=1}^{N} \ln P(\mathbf{x}^i \mid \theta).$$

To keep things simple, we will consider only the case a naive Bayes classifier with Boolean attributes. We saw above that the model can be characterized by the parameters

$$\begin{aligned} \theta_C &= P(C = T) \\ \theta_i^T &= P(X_i = T \mid C = T) \\ \theta_i^F &= P(X_i = T \mid C = F) \end{aligned}$$

We also saw that if an instance $\mathbf{x}$ has the classification $c = T$, the probability of $\mathbf{x}$ is

$$\theta_C \prod_i (\theta_i^T)^{x_i} (1 - \theta_i^T)^{1 - x_i}.$$

(Recall that we use 1 and 0 for true and false.) Similarly, if the classification is false, the probability of $\mathbf{x}$ is

$$(1 - \theta_C) \prod_i (\theta_i^F)^{x_i} (1 - \theta_i^F)^{1 - x_i}.$$

Splitting up D into the cases with positive and negative classifications, we get the following expression for the likelihood of the data given the parameters θ.

$$\begin{array}{rcccl} \mathcal{L}(\theta) & = & \sum_{j:c^j=T} & \ln \theta_C & + \sum_i (x_i^j \ln \theta_i^T + (1 - x_i^j) \ln(1 - \theta_i^T)) \quad + \\ & & \sum_{j:c^j=F} & \ln(1 - \theta_C) & + \sum_i (x_i^j \ln \theta_i^F + (1 - x_i^j) \ln(1 - \theta_i^F)) \end{array}$$

To find values of the parameters that maximize this likelihood, we differentiate with respect to each of the parameters. Let's start with θ_i^T. Only the first half of the above expression, for the examples with positive classification, involves this parameter. Also, only a single term in the summation over i applies. We get

$$\frac{\partial \mathcal{L}}{\partial \theta_i^T} = \sum_{j:c^j=T} \frac{x_i^j}{\theta_i^T} - \frac{1 - x_i^j}{1 - \theta_i^T}.$$

To find extreme values, we equate this to zero, to get

$$\sum_{j:c^j=T} (x_i^j(1-\theta_i^T) - (1-x_i^j)\theta_i^T = 0)$$

Let's define

$$\begin{aligned}
N &= \text{total number of instances} \\
N_T &= \text{number of instances } j \text{ such that } c^j \text{ is } T \\
N_F &= \text{number of instances } j \text{ such that } c^j \text{ is } F \\
N_{i,T}^T &= \text{number of instances } j \text{ such that } c^j \text{ is } T \text{ and } x_i^j = 1 \\
N_{i,F}^T &= \text{number of instances } j \text{ such that } c^j \text{ is } T \text{ and } x_i^j = 0 \\
N_{i,T}^F &= \text{number of instances } j \text{ such that } c^j \text{ is } F \text{ and } x_i^j = 1 \\
N_{i,F}^F &= \text{number of instances } j \text{ such that } c^j \text{ is } F \text{ and } x_i^j = 0
\end{aligned}$$

(Note that $N_T = N_{i,T}^T + N_{i,F}^T$ and $N_F = N_{i,T}^T + N_{i,F}^T$.) Then the above equation can be replaced by

$$N_{i,T}^T(1-\theta_i^T) - N_{i,F}^T\theta_i^T = 0$$

Solving, we get

$$\theta_i^T = \frac{N_{i,T}^T}{N_{i,T}^T + N_{i,F}^T} = \frac{N_{i,T}^T}{N_T}.$$

It is easy to check by inspection that this indeed maximizes the likelihood. Similarly, we find that

$$\theta_i^F = \frac{N_{i,T}^F}{N_F}.$$

We can also take derivatives with respect to θ_C, to get that

$$\theta_C = \frac{N_T}{N}.$$

These parameters are exactly what you would expect them to be. The value of $P(C = T)$ that maximizes the likelihood is exactly the proportion of instances with positive classification in the data set. Similarly, the value of $P(X_i = T \mid C = T)$ is exactly the proportion, amongst those instances that have positive classification, of the instances that have $X_i = T$.

The family of N-values that we defined above are called the *sufficient statistics* of the data. They are all we need to know about the data in order to determine the parameter values. The sufficient statistics summarize the data in a very compact manner. All we need to know is how many times each class appears, and how many times each feature value appears for each class. We can ignore information about which particular feature values appeared together in particular data cases. This fits exactly with the independence assumptions made by the naive Bayes model. Since the different features are independent of each other given the class, we don't care about the co-occurence of feature values in particular data cases, just about how many times each feature value appears with a classification.

3 Autoclass

We can use the naive Bayes model for unsupervised learning. Again, we'll assume two clusters and Boolean attributes for convenience. Our data $\mathbf{D}$ consists of points $\mathbf{x}^1, \ldots, \mathbf{x}^N$, but no classifications, and we wish to find a classification rule that will cluster the observed data and also classify unseen data into one of the clusters. We will make the modeling assumption that the data is generated by a naive Bayes model, i.e., that in the conditional distribution over the attributes given the class, the attributes are independent of each other. As we have seen, this model can be characterized by the parameters θ_C, θ_i^T and θ_i^F.

We will again use the maximum likelihood approach to learn the parameter values. As before, we seek values of θ that maximize $\mathcal{L}(\theta) = \sum_{i=1}^{N} \ln P(\mathbf{x}^i \mid \theta)$. The only difference from before is that the data no longer contains the classifications. This makes a closed form solution for the ML parameter values difficult.

Instead, we use a more sophisticated algorithm called Autoclass, based on the *expectation-maximization (EM)* algorithm. EM is a very powerful, general algorithm for learning probabilistic models from data that is not fully observed, and it has an enormous number of applications.

The basic idea of EM is quite simple. It consists of the following observations:

1. If we have the actual sufficient statistics of the data, we can compute the parameter values that maximize the likelihood of the data. This is just the problem of learning a probabilistic model from complete data, which we have already discussed.

2. Suppose we actually succeed in learning the model parameters. We could then compute a probability distribution over the values of the missing attributes. For the naive Bayes example, we could compute

$$P(C^j \mid x_1^j, \ldots, x_n^j) \quad = \quad \frac{P(C^j) \prod_{i=1}^{n} P(x_i^j \mid C^j)}{P(x_1^j, \ldots, x_n^j)} \tag{2}$$

   As usual, the denominator is a normalizing constant which we don't have to compute directly. Instead, we can compute the numerator for each possible classification, and scale the results so that they sum to 1.

   Given that we can do this, even though we don't have actual sufficient statistics of the data, we can compute *expected values* for the sufficient statistics. In our naive Bayes example, $E[N_T]$ is the expected number of instances with positive classification, and is equal to $\sum_{j=1}^{N} P(C^j = T \mid x_1^j, \ldots, x_n^j)$. Similarly, $E[N_{i,T}^T]$ is the expected number of instances with positive classification where X_i is true, and is equal to $\sum_{j:x_i^j=T} P(C^j = T \mid x_1^j, \ldots, x_n^j)$.

   So to compute the expected sufficient statistics, we simply need to compute Equation(**??**) for each instance, and then perform the appropriate summation for each of the sufficient statistics.

The EM algorithm puts these two observations together into an *iterative algorithm*, as follows:

```
Set θ to some initial values.
Repeat until convergence
  Compute the expected sufficient statistics E[N], using θ
  Choose θ so as to maximize the likelihood of E[N]
```

The first step of each iteration is called the *expectation* or E step, while the second step is called the *maximization* or M step. Hence the name EM.

For the naive Bayes example, this is elaborated to:

Set θ_C, θ_i^T and θ_i^F to initial values
Repeat until convergence
  /* Expectation Step */
  $E[N_T] = 0$
  $E[N_{i,T}^T] = 0$ for each i
  $E[N_{i,T}^F] = 0$ for each i
  For each instance D^j
    $p_T = \theta_C \prod_i (\theta_i^T)^{x_i^j} (1 - \theta_i^T)^{(1-x_i^j)}$
    $p_F = (1 - \theta_C) \prod_i (\theta_i^F)^{x_i^j} (1 - \theta_i^F)^{(1-x_i^j)}$
    $P(C^j = T | x_1^j, ..., x_n^j) = \frac{p_T}{p_T + p_F}$
    $E[N_T]$ += $P(C^j = T | x_1^j, ..., x_n^j)$
    For each attribute i
      If $x_i^j = T$
        $E[N_{i,T}^T]$ += $P(C^j = T | x_1^j, ..., x_n^j)$
        $E[N_{i,T}^F]$ += $1 - P(C^j = T | x_1^j, ..., x_n^j)$

  /* Maximization Step */
  $\theta_C = \frac{E[N_T]}{N}$
  For each attribute i
    $\theta_i^T = \frac{E[N_{i,T}^T]}{E[N_T]}$
    $\theta_i^F = \frac{E[N_{i,T}^F]}{N - E[N_T]}$

It is important to understand how this algorithm works, because it is paradigmatic of a general algorithm that can be applied to many different probabilistic models. If you can reconstruct this algorithm, you should be able to construct an EM algorithm for many different examples. The homework problem asks you to do this for a slightly more complex model than naive Bayes.

Why does EM work? It is a basic fact (which we're not going to prove here), that on each iteration, the likelihood of the data given the new parameter values is at least as great as the likelihood given the old ones. So EM behaves a lot like gradient descent — at each step, it adjusts the parameter values so as to improve the likelihood of the data. It follows that EM must converge to a set of

parameter values that locally maximize the likelihood. (It may also converge to a saddle point.) As always with iterative improvement algorithms, local optima may be a problem. We can use standard techniques such as random restarts to deal with this.

One point about the initial parameter values: it turns out that starting with uniform parameter values, where each is equal to $1/2$, is a saddle point in the naive Bayes model, because of symmetry. Consider that in clustering, the names of the clusters never appear in the training set, so they are essentially meaningless. Thus if we take a model, and then permute the cluster labels, we will get an essentially equivalent and symmetric model. Therefore, the likelihood function is symmetric, with the center of symmetry being the uniform parameter values. Therefore it is better to start with random, asymmetric parameter values.