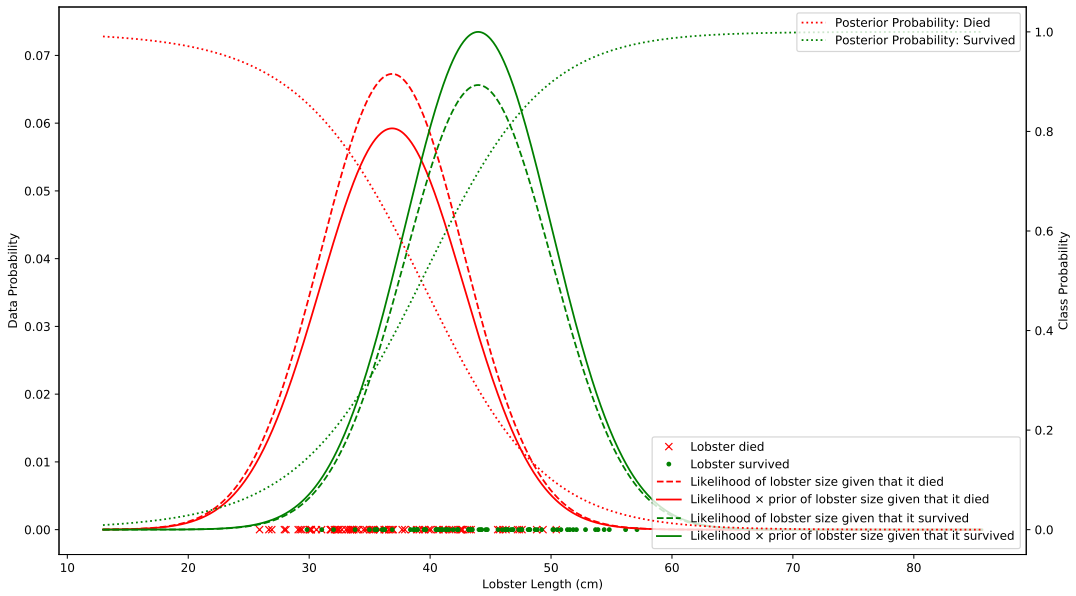

Figure 3.1: Bayes Classifier applied to the lobster problem.

CSCI447/547: Machine Learning **Spring 2018**

Lecture 3: Feb. 5, 2018

Lecturer: Doug Brinkerhoff

3.1 Naive Bayes' Classifier, Round 2

Last week we talked about the Bayes' classifier, in particular with the naive Bayes' assumption for the problem of classifying irises based on their features. Today, let's talk about this classifier in its most simplified form, choosing between two classes. In particular, let's consider a dataset that I got from UCI, in which lobsters were staked to the sea floor, and their size versus whethere they survived from predators were recorded (Fig. 3.1). The data has only one feature (lobster sizes), and the labels are binary (C_1 implies that the lobster survived, C_0 implies that it died). Let's assume that there were $N_0 = 20$ lobsters that died and $N_1 = 30$ lobsters that survived. Since there's only two classes, the decision criterion is

quite simple:

$$C = \begin{cases} C_1, \ P(C_1|\mathbf{x}, D) > \frac{1}{2} \\ C_0, \ \text{Otherwise} \end{cases} \tag{3.1}$$

To compute $P(C_1|\mathbf{x}, D)$, we use Bayes' theorem

$$P(C_1|\mathbf{x}, D) = \frac{P(\mathbf{x}|C_1, D)P(C_1|D)}{P(\mathbf{x}|C_1, D)P(C_1|D) + P(\mathbf{x}|C_0, D)P(C_0, D)}. \tag{3.2}$$

In the previous lecture, we modelled the class-conditional likelihoods $P(\mathbf{x}|C_1, D)$ as normally distributed, and we'll do the same here.

$$\mathbf{x}|C_1, D \sim \mathcal{N}(\boldsymbol{\mu}_1, \Sigma_1), \tag{3.3}$$

where $\boldsymbol{\mu}_1$ is the mean and Σ_1 the variance. The most straightforward way of finding these values is with maximum likelihood estimation over the training data or

$$\boldsymbol{\mu}_1 = \arg\max_{\boldsymbol{\mu}} P(D|\boldsymbol{\mu}, \Sigma, C_1), \tag{3.4}$$

and

$$\Sigma_1 = \arg\max_{\Sigma} P(D|\boldsymbol{\mu}, \Sigma, C_1). \tag{3.5}$$

We assume that the model for the training data D is the same as that for the feature vector $\mathbf{x}$. It is well known that the maximum likelihood estimator for a normally distribution is just the sample mean and sample covariance, or

$$\boldsymbol{\mu}_1 = \frac{\sum_{i=1}^{m} D_i[C_i = C_k]}{\sum_{i=1}^{m}[C_i = C_k]}, \tag{3.6}$$

$$\Sigma_1 = \frac{\sum_{i=1}^{m}(D_i - \boldsymbol{\mu}_1)(D_i - \boldsymbol{\mu}_1)^T[C_i = C_1]}{\sum_{i=1}^{m}[C_i = C_1]}, \tag{3.7}$$

or the sample mean and variance for those members of the training set labelled as C_1. The formulae for C_0 are similar. Note that we have not invoked the naive assumption here.

Here's what we're doing: for all the live lobsters, we're saying that the lobster size is normally distributed, with mean and variance given by the sample mean and sample variance over the data labelled as a live lobster. Conversely, for the lobsters that died, we're saying that the lobster size is also normally distributed, with similar sample mean and sample variance.

Then when we get a new lobster measurement, we see how likely that measurement is for each class, and we predict that the lobster belongs to the class for which the data is more probable.

3.2 A refactorization into the logistic function

Now let's look at the posterior class probability in a different way. Note that we can rewrite the class posterior probability as

$$P(C_1|\mathbf{x}, D) = \frac{1}{1 + \frac{P(\mathbf{x}|C_0,D)P(C_0|D)}{P(\mathbf{x}|C_1,D)P(C_1|D)}}. \tag{3.8}$$

The fraction in the denominator is a different but equivalent quantification of probability: the odds. The odds are asymmetrical and it's useful to work in symmetrical quantities when it comes to probability distributions, so we introduce still a different (but still equivalent) way of writing the class posterior probability

$$P(C_1|\mathbf{x}, D) = \frac{1}{1 + \mathrm{e}^{-a_1}}, \tag{3.9}$$

where

$$a_1 = \ln \frac{P(\mathbf{x}|C_1, D)P(C_1|D)}{P(\mathbf{x}|C_0, D)P(C_0|D)}, \tag{3.10}$$

where now we have a function of the so-called *log odds*. The log odds are sort of interesting. For $P(x|C_1, D) \to 1$, the log odds go to ∞ and as $P(x|C_1, D) \to 0$, the log odds go to negative infinity. Once again, log odds are a perfectly equivalent way of thinking about and writing about probability when there are only two mutually exclusive possibilities.

But, why is this a useful way of looking at this problem? The function that we've written down is one that is particularly useful, so much so that it has its own name: *the logistic function*. It's a member of a family of functions called *sigmoids* (for s-shaped). The logistic function in particular squashes the entire real line down to the interval $[0, 1]$, and it's often interpreted as a probablity, as we'll see when we talk about neural networks. This is why interpreting the output of a sigmoid as a probability is justified.

We'll introduce the shorthand for the sigmoid function as

$$\sigma(a_1) = \frac{1}{1 + \mathrm{e}^{-a_1}}. \tag{3.11}$$

3.3 Log odds as a quadratic function

Interestingly, it turns out that the log odds are, generally speaking, a quadratic function of the features, which means that the log-odds for C_1 can be written as

$$a_1 = \mathbf{x}^T C \mathbf{x} + \mathbf{w}^T \mathbf{x} + w_0, \tag{3.12}$$

where C is a $n \times n$ matrix of parameters, $\mathbf{w}$ is an $n \times 1$ parameter vector, and w_0 is a scalar parameter. Each of these parameters depend upon the maximum likelihood parameters μ_1, μ_0, Σ_1, and Σ_2.

Expanding the log odds we have

$$\begin{aligned}
a_1 &= \ln \frac{P(\mathbf{x}|C_1, D)P(C_1|D)}{P(\mathbf{x}|C_0, D)P(C_0|D)} && (3.13)\\
&= \ln P(\mathbf{x}|C_1, D)P(C_1|D) - \ln P(\mathbf{x}|C_0, D)P(C_0|D) && (3.14)\\
&= -\frac{1}{2}\left[(\mathbf{x}-\boldsymbol{\mu}_1)^T\Sigma_1(\mathbf{x}-\boldsymbol{\mu}_1) - (\mathbf{x}-\boldsymbol{\mu}_0)^T\Sigma_0(\mathbf{x}-\boldsymbol{\mu}_0)\right] + \ln\frac{P(C_1)}{P(C_0)} && (3.15)\\
&= \frac{1}{2}\mathbf{x}^T(\Sigma_0-\Sigma_1)\mathbf{x} + (\boldsymbol{\mu}_1^T\Sigma_1 - \boldsymbol{\mu}_0^T\Sigma_0)\mathbf{x} - \frac{1}{2}\boldsymbol{\mu}_1^T\Sigma_1\boldsymbol{\mu}_1 + \frac{1}{2}\boldsymbol{\mu}_0^T\Sigma_0\boldsymbol{\mu}_0 + \ln\frac{P(C_1)}{P(C_0)}, && (3.16)
\end{aligned}$$

which can be written in the form of Eq. 3.12, if we make the following identifications

$$C = \Sigma_0 - \Sigma_1$$

$$\mathbf{w}^T = \boldsymbol{\mu}_1^T\Sigma_1 - \boldsymbol{\mu}_0^T\Sigma_0$$

$$w_0 = -\frac{1}{2}\boldsymbol{\mu}_1^T\Sigma_1\boldsymbol{\mu}_1 + \frac{1}{2}\boldsymbol{\mu}_0^T\Sigma_0\boldsymbol{\mu}_0 + \ln\frac{P(C_1)}{P(C_0)}.$$

Thus the log-odds are a quadratic function of the feature vector $\mathbf{x}$, and the parameter values can be directly computed from the maximum likelihood estimators for the training data.

3.4 Generalized linear model

Now let's make a simplifying assumption: note that if the the covariance matrices Σ_1 and Σ_0 are close to the same, then this term is going to be close to zero, so let's just ignore it. This is like saying that the size range of lobsters that die is about the same size range as the lobsters that survive. Is this a good assumption? It depends on the dataset of course, but let's imagine that it holds for us. If we drop the quadratic term (equivalent to assuming that both classes have the same covariance), we can write the model as

$$P(C_1|\mathbf{x}, D) = \sigma(\mathbf{w}^T\mathbf{x} + w_0). \tag{3.17}$$

This type of model is called a *generalized linear model*. The *generalized* just means that we take a linear model like $\mathbf{w}^T\mathbf{x} + w_0$ and run it through some non-linear function, in this case the logistic function. This also provides an interesting possibility.

In the above derivation, we were able to write down closed solutions for the values of $\mathbf{w}$ and w_0, because we made the explicit assumption that the class conditional likelihoods were normally distributed, with maximum likelihood estimators that were easy to compute. However, what if rather than make this assumption *a priori* we could just solve for some optimal values of $\mathbf{w}$ and w_0 that produce predictions that closely match the observations?

We already have most of the tools to do this: define a cost function $\mathcal{J}(X, \mathbf{y}, \mathbf{w})$, where X is the design matrix (for example the Vandermond matrix from linear regression), $\mathbf{y} == \{y_1, \ldots, y_m\}$ is a vector of class labels ($y_i \in \{0, 1\}$ in this case), and in a slight abuse of notation, $\mathbf{w}$ is a vector of parameters that will now also include the bias parameter w_0. This function should measure the misfit between our predicted class given by $\sigma(\mathbf{w}^T X_i)$ and the observed class y_i. The optimal values of $\mathbf{w}$ will be the ones that minimize $\mathcal{J}(X, \mathbf{y}, \mathbf{w})$.

What's a good cost function? The cost function that we've seen already is *least squares*, which is the negative logarithm of the normal distribution. It doesn't make sense to use least squares, because that predicts real-valued data, but we are dealing with binary class labels. However, the idea of using the negative log likelihood of a probability density function as a cost function is sound. What probability density function do we use to model binary

data (like coin flips)? The Bernoulli distribution, which is given by

$$P(\mathbf{y}|\boldsymbol{\theta}) \propto \prod_{i=1}^{m} \theta_i^{y_i}(1-\theta_i)^{1-y_i}, \tag{3.18}$$

where $\boldsymbol{\theta}$ can be viewed as the means for each data point, or the probability that a given trial will yield a success. Just as in the case of the normal distribution where we assumed that the data were normally distributed around a mean that was given by our linear model, here we'll assume that the data labels are Bernoulli distributed around a mean that is given by our generalized linear model, so

$$P(\mathbf{y}|\mathbf{w}) = \prod_{i=1}^{m} \sigma(\mathbf{w}^T X_i)^{y_i}(1-\sigma(\mathbf{w}^T X_i))^{(1-y_i)}. \tag{3.19}$$

Maximizing this function is equivalent to finding the minimum of the negative logarithm:

$$\arg\max_{\mathbf{w}} P(\mathbf{y}|\mathbf{w}) = \arg\min_{\mathbf{w}} -\ln P(\mathbf{y}|\mathbf{w}). \tag{3.20}$$

$$-\ln P(\mathbf{y}|\mathbf{w}, X) = \mathcal{J}(X, \mathbf{y}, \mathbf{w}) = -\sum_{i=1}^{m} y_i \ln \sigma(\mathbf{w}^T X_i) + (1-y_i)\ln(1-\sigma(\mathbf{w}^T X_i)). \tag{3.21}$$

The derivative of this cost function with respect to the weights is

$$\frac{\partial \mathcal{J}}{\partial w_j} = \sum_{i=1}^{m} (\sigma(\mathbf{w}^T X_i) - y_i) X_{ij}, \tag{3.22}$$

where w_j is the $j-$th component of the parameter vector $\mathbf{w} = \{w_0, w_1, \ldots, w_n\}$, which has length $n+1$, where n is the number of dataset features.

3.5 Gradient descent

Unfortunately, this is a non-linear function of $\mathbf{w}$, and so it doesn't have an obvious closed-form solution like least squares did. What should we do about this? Well, we have a measurement of error that we can easily compute, and we have its gradient. How about we just follow the gradient downhill until our error is as low as it can get? This implies an algorithm:

```
while J(y,w,x)>tol:
    w_j = w_j - eta*dJ_dw_j
```

where η is the so-called 'learning rate'. This simple algorithm is not the fanciest on the block, but it works well enough and we'll use it extensively when talking about neural networks. Indeed, have a look at this week's ipython notebook to see how this can be implemented.

3.6 Generative versus discriminative modelling

So we saw, that we have two options when it comes to performing the two-class classification: first, we can make some assumptions about how the class-conditional probabilities are distributed. If we then use maximum likelihood estimation, then training this model is a snap (there are closed form solutions for the model parameters that we directly compute from the data). However, this assumption about the errors could be wrong. On the other hand, we can simply take the logistic regression approach, where we model the log odds as a linear combination of the input features. This lets us make a minimal number of assumptions about how the data is distributed, but leads to a non-linear training process, where we have to use an iterative technique like gradient descent to find the optimal parameters. These two choices correspond to two paradigms in machine learning. The case where we explicitly specify a model for class-conditional likelihoods is called *generative modelling*. The case where we assumed that log odds are a general linear function of the features, with the parameters to be determined, is called *discriminative modelling*. There is an easy test to determine which type of modelling you're dealing with: just ask yourself the question 'could I generate new data with this model?'. In the case where we know the class-conditional likelihoods, it would be easy to simulate data: choose a class, then draw the features from the normal distribution associated with that class. However, this isn't possible when we simply find the parameters that fit the data best, because we don't know what underlying likelihood we end up modelling.

3.7 Logistic regression for multiple classes, aka softmax regression, aka multinomial regression

Logistic regression is useful for cases where there only two classes. However, Bayes' rule for multiple classes, which is given by

$$P(C_k|\mathbf{x}, D) = \frac{P(\mathbf{x}|C_k, D)P(C_k|D)}{\sum_j P(\mathbf{x}|C_j, D)P(C_j)}, \tag{3.23}$$

can also be factorized by making the substitution

$$a_k = \ln P(\mathbf{x}|C_k, D)P(C_k|D), \tag{3.24}$$

so that we have

$$P(C_k|\mathbf{x}, D) = \frac{\exp(a_k)}{\sum_j \exp(a_j)}. \tag{3.25}$$

Note that for the logistic equation, we parameterized in terms of the log-odds. Since we're dealing with multiple classes, we can't sensibly talk about odds between them anymore. Instead, the arguments to Eq. 3.25 (a_k), are log probablities. Eq. 3.25 is called the *softmax* function, because it's a smoothed version of the arg max function: if a_k is much larger than any a_j, then $P(C_k|\mathbf{x}, D) \approx 1$, and $P(C_j|\mathbf{x}, D) \approx 0$. As before, we have two choices: we can take the generative approach, in which we specify a likelihood function explicitly, which allows us to compute maximum likelihood estimators for the class conditional densities, and use this information to compute the probability of each class; or we can take the discriminative approach, in which we assume that

$$a_k = \mathbf{w}_k^T \mathbf{x}, \tag{3.26}$$

or

$$P(C_k|\mathbf{x}, \mathbf{w}) = \text{Softmax}(\mathbf{w}, \mathbf{x}) = \frac{\exp(\mathbf{w}_k^T \mathbf{x})}{\sum_j \exp(\mathbf{w}_j^T \mathbf{x})}, \tag{3.27}$$

where $\mathbf{w}_k$ is the length $n+1$ vector of parameters corresponding to the k-th class. Because we don't have the ability to say that $P(C_1) = 1 - P(C_0)$, we have somewhat more parameters to solve for now. In particular, we'll have to find the values of $(n+1) \times N$ parameters, where n is the number of features in the dataset, and N is the number of classes.

Unsurprisingly, the procedure for training these parameters is similar to the 2-class case. The difference is that our likelihood function is now *multinomial*, which is to say that our parameters are drawn from the multinomial distribution. Following Bishop, it's easiest to model this as a one-hot encoding scheme for the classes which just implies that we write $\mathbf{y} \to \mathcal{T}$, where $\mathcal{T}$ is a $m \times N$ matrix with rows corresponding to observations and columns corresponding to classes, and $\mathbf{y}$ is again a vector of length m that contains class labels. The column of $\mathcal{T}$ corresponding to the class C_k is one if $y_i = C_k$. Thus, each row of $\mathcal{T}$ has exactly one entry that is one, with the rest being zeros. The likelihood function is then

$$P(\mathbf{T}|\mathbf{w}_1, \ldots, \mathbf{w}_N) = \prod_{i=1}^{m} \prod_{k=1}^{N} \text{Softmax}(\mathbf{w}_k, X_i)^{\mathcal{T}_{ik}}. \tag{3.28}$$

Taking the negative log-likelihood provides an objective function

$$\mathcal{J}(\mathbf{w}_{1,\ldots,N}, X, \mathbf{y}) = -\sum_{i=1}^{m} \sum_{k=1}^{N} \mathcal{T}_{ik} \ln \text{Softmax}(\mathbf{w}_k, X_i), \tag{3.29}$$

with the gradient

$$\frac{\partial \mathcal{J}}{\partial w_{jk}} = \sum_{i=1}^{m} (\text{Softmax}(X_i, \mathbf{w}_k) - \mathcal{T}_{ik}) X_{ij}. \tag{3.30}$$

As above, we can use the objective function and gradient in conjunction with gradient descent (or another algorithm for minimizing a function) to find the optimal weights.