8 Classification

In classification, we are trying to learn a map from an input space to some finite output space. In the simplest case we simply detect whether or not the input has some property or not. For example, we might want to determine whether or not an email is spam, or whether an image contains a face. A task in the health care field is to determine, given a set of observed symptoms, whether or not a person has a disease. These detection tasks are *binary classification* problems.

In *multi-class classification* problems we are interested in determining to which of multiple categories the input belongs. For example, given a recorded voice signal we might wish to recognize the identity of a speaker (perhaps from a set of people whose voice properties are given in advance). Another well studied example is optical character recognition, the recognition of letters or numbers from images of handwritten or printed characters.

The input $\mathbf{x}$ might be a vector of real numbers, or a discrete feature vector. In the case of binary classification problems the output y might be an element of the set $\{-1, 1\}$, while for a multi-dimensional classification problem with N categories the output might be an integer in $\{1, \ldots, N\}$.

The general goal of classification is to learn a *decision boundary*, often specified as the level set of a function, e.g., $a(\mathbf{x}) = 0$. The purpose of the decision boundary is to identity the regions of the input space that correspond to each class. For binary classification the decision boundary is the surface in the feature space that separates the test inputs into two classes; points $\mathbf{x}$ for which $a(\mathbf{x}) < 0$ are deemed to be in one class, while points for which $a(\mathbf{x}) > 0$ are in the other. The points on the decision boundary, $a(\mathbf{x}) = 0$, are those inputs for which the two classes are equally probable.

In this chapter we introduce several basis methods for classification. We focus mainly on on binary classification problems for which the methods are conceptually straightforward, easy to implement, and often quite effective. In subsequent chapters we discuss some of the more sophisticated methods that might be needed for more challenging problems.

8.1 Class Conditionals

One approach is to describe a "generative" model for each class. Suppose we have two mutually-exclusive classes C_1 and C_2. The prior probability of a data vector coming from class C_1 is $P(C_1)$, and $P(C_2) = 1 - P(C_1)$. Each class has a distribution for its data: $p(\mathbf{x}|C_1)$, and $p(\mathbf{x}|C_2)$. In other words, to sample from this model, we would first randomly choose a class according to $P(C_1)$, and then sample a data vector $\mathbf{x}$ from that class.

Given labeled training data $\{(\mathbf{x}_i, y_i)\}$, we can estimate the distribution for each class by maximum likelihood, and estimate $P(C_1)$ by computing the ratio of the number of elements of class 1 to the total number of elements.

Once we have trained the parameters of our *generative* model, we perform classification by comparing the posterior class probabillities:

$$P(C_1|\mathbf{x}) > P(C_2|\mathbf{x}) \; ? \tag{1}$$

That is, if the posterior probability of C_1 is larger than the probability of C_2, then we might classify the input as belonging to class 1. Equivalently, we can compare their ratio to 1:

$$\frac{P(C_1|\mathbf{x})}{P(C_2|\mathbf{x})} > 1 \; ? \tag{2}$$

If this ratio is greater than 1 (i.e. $P(C_1|\mathbf{x}) > P(C_2|\mathbf{x})$) then we classify $\mathbf{x}$ as belonging to class 1, and class 2 otherwise.

The quantities $P(C_i|\mathbf{x})$ can by computed using Bayes' Rule as:

$$P(C_i|\mathbf{x}) = \frac{p(\mathbf{x}|C_i)\, P(C_i)}{p(\mathbf{x})} \tag{3}$$

so that the ratio is:

$$\frac{p(\mathbf{x}|C_1)\, P(C_1)}{p(\mathbf{x}|C_2)\, P(C_2)} \tag{4}$$

Note that the $p(\mathbf{x})$ terms cancel and so do not need to be computed. Also, note that these computations are typically done in the logarithmic domain as this is often faster and more numerically stable.

Gaussian Class Conditionals. As a concrete example, consider a generative model in which the inputs associated with the i^{th} class (for $i = 1, 2$) are modeled with a Gaussian distribution, i.e.,

$$p(\mathbf{x}|C_i) = G(\mathbf{x}; \mu_i, \mathbf{\Sigma}_i) \; . \tag{5}$$

Also, let's assume that the prior class probabilities are equal:

$$P(C_i) = \frac{1}{2} \; . \tag{6}$$

The values of μ_i and $\mathbf{\Sigma}_i$ can be estimated by maximum likelihood on the individual classes in the training data.

Given this models, you can show that the log of the posterior ratio (4) is given by

$$a(\mathbf{x}) = -\frac{1}{2}(\mathbf{x} - \mu_1)^T \mathbf{\Sigma}_1^{-1}(\mathbf{x} - \mu_1) - \frac{1}{2}\ln|\mathbf{\Sigma}_1| + \frac{1}{2}(\mathbf{x} - \mu_2)^T \mathbf{\Sigma}_2^{-1}(\mathbf{x} - \mu_2) + \frac{1}{2}\ln|\mathbf{\Sigma}_2| \tag{7}$$

The sign of this function determines the class of $\mathbf{x}$, since the ratio of posterior class probabilities is greater than 1 when this log is greater than zero. Since $a(\mathbf{x})$ is quadratic in $\mathbf{x}$, the decision boundary (i.e., the set of points satisfying $a(\mathbf{x}) = 0$) is a conic section (e.g., a parabola, an ellipse, a line, etc.). Furthermore, in the special case where $\mathbf{\Sigma}_1 = \mathbf{\Sigma}_2$, the decision boundary is linear (why?).

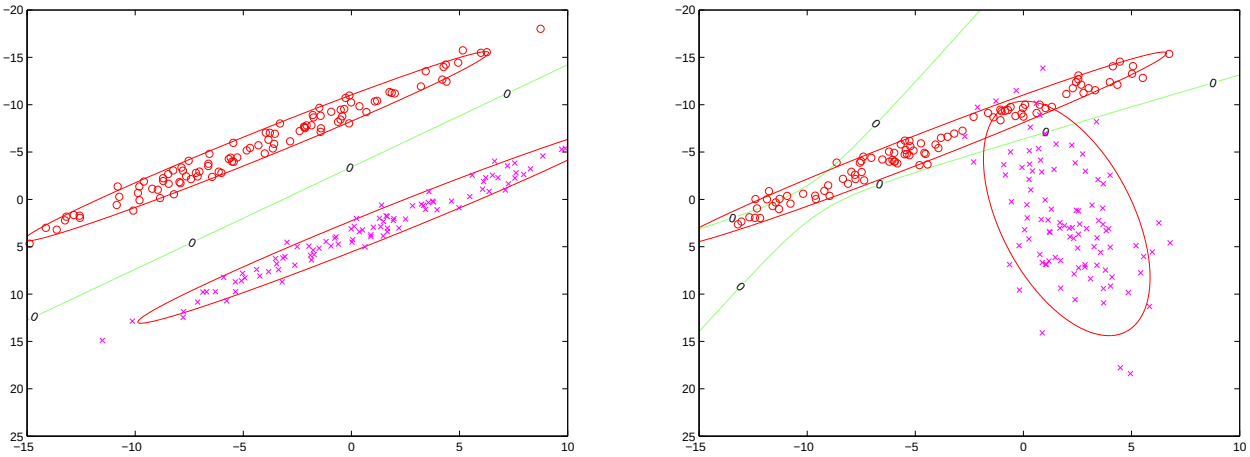

Figure 1: GCC classification boundaries for two cases. Note that the decision boundary is linear when both classes have the same covariance.

8.2 Logistic Regression

Noting that $p(\mathbf{x})$ can be written as (why?)

$$p(\mathbf{x}) \;=\; p(\mathbf{x}, C_1) + p(\mathbf{x}, C_2) \;=\; p(\mathbf{x}|C_1)P(C_1) + p(\mathbf{x}|C_2)P(C_2)\;, \tag{8}$$

we can express the posterior class probability as

$$P(C_1|\mathbf{x}) = \frac{p(\mathbf{x}|C_1)P(C_1)}{p(\mathbf{x}|C_1)P(C_1) + p(\mathbf{x}|C_2)P(C_2)}\;. \tag{9}$$

Dividing both the numerator and denominator by $p(\mathbf{x}|C_1)P(C_1)$ we obtain:

$$P(C_1|\mathbf{x}) \;=\; \frac{1}{1 + e^{-a(\mathbf{x})}} \tag{10}$$

$$\;=\; g(a(\mathbf{x})) \tag{11}$$

where $a(\mathbf{x}) = \ln \frac{p(\mathbf{x}|C_1)P(C_1)}{p(\mathbf{x}|C_2)P(C_2)}$ and $g(a)$ is the sigmoid function. Note that $g(a)$ is monotonic, so that the probability of class C_1 grows as a grows and is precisely $\frac{1}{2}$ when $a = 0$. Since $P(C_1|\mathbf{x}) = \frac{1}{2}$ represents equal probability for both classes, this is the boundary along which we wish to make decisions about class membership.

For the case of Gaussian class conditionals where both Gaussians have the same covariance, a is a linear function of $\mathbf{x}$. In this case the classification probability can be written as

$$P(C_1|\mathbf{x}) = \frac{1}{1 + e^{-\mathbf{w}^T\mathbf{x} - b}} = g(\mathbf{w}^T\mathbf{x} + b)\;, \tag{12}$$

or, if we augment the data vector with a 1 and the weight vector with b,

$$P(C_1|\mathbf{x}) = \frac{1}{1 + e^{-\mathbf{w}^T\mathbf{x}}}\;. \tag{13}$$

At this point, we can forget about the generative model (e.g., the Gaussian distributions) that we started with, and **use this as our entire model.** In other words, rather than learning a distribution over each class, we learn only **the conditional probability of y given $\mathbf{x}$**. As a result, we have fewer parameters to learn since the number of parameters in logistic regression is linear in the dimension of the input vector, while learning a Gaussian covariance requires a quadratic number of parameters. With fewer parameters we can learn models more effectively with less data. On the other hand, we cannot perform other tasks that we could with the generative model (e.g., sampling from the model; classify data with noisy or missing measurements).

We can learn logistic regression with maximum likelihood. In particular, given data $\{\mathbf{x}_i, y_i\}$, we minimize the negative log of:

$$\begin{aligned} p(\{\mathbf{x}_i, y_i\}|\mathbf{w}, b) &\propto p(\{y_i\} \,|\, \{\mathbf{x}_i\}, \mathbf{w}, b) \\ &= \prod_i p(y_i|\mathbf{x}_i, \mathbf{w}, b) \\ &= \prod_{i:y_i=C_1} P(C_1|\mathbf{x}_i) \prod_{i:y_i=C_2} (1 - P(C_1|\mathbf{x}_i)) \end{aligned} \tag{14}$$

In the first step above we have assumed that the input features are independent of the weights in the logistic regressor, i.e., $p(\{\mathbf{x}_i\}) = p(\{\mathbf{x}_i\}|\mathbf{w}, b)$. So this term can be ignored in the likelihood since it is constant with respect to the unknowns. In the second step we have assumed that the input-output pairs are independent, so the joint likelihood is the product of the likelihoods for each input-output pair.

The decision boundary for logistic regression is linear; in 2D, it is a line. To see this, recall that the decision boundary is the set of points $P(C_1|\mathbf{x}) = 1/2$. Solving for $\mathbf{x}$ gives the points $\mathbf{w}^T\mathbf{x} + b = 0$, which is a line in 2D, or a hyperplane in higher dimensions.

Although this objective function cannot be optimized in closed-form, it is convex, which means that it has a single minimum. Therefore, we can optimize it with gradient descent (or any other gradient-based search technique), which will be guaranteed to find the global minimum.

If the classes are linearly separable, this approach will lead to very large values of the weights $\mathbf{w}$, since as the magnitude of $\mathbf{w}$ tends to infinity, the function $g(a(x))$ behaves more and more like a step function and thus assigns higher likelihood to the data. This can be prevented by placing a weight-decay prior on $\mathbf{w}$: $p(\mathbf{w}) = G(\mathbf{w};\, 0, \sigma^2)$.

Multiclass classification. Logistic regression can also be applied to multiclass classification, i.e., where we wish to classify a data point as belonging to one of K classes. In this case, the probability of data vector $\mathbf{x}$ being in class i is:

$$P(C_i|\mathbf{x}) = \frac{e^{-\mathbf{w}_i^T\mathbf{x}}}{\sum_{k=1}^{K} e^{-\mathbf{w}_k^T\mathbf{x}}} \tag{15}$$

You should be able to see that this is equivalent to the method described above in the two-class case. Furthermore, it is straightforward to show that this is a sensible choice of probability: $0 \leq P(C_i|\mathbf{x})$, and $\sum_k P(C_k|\mathbf{x}) = 1$ (verify these for yourself).

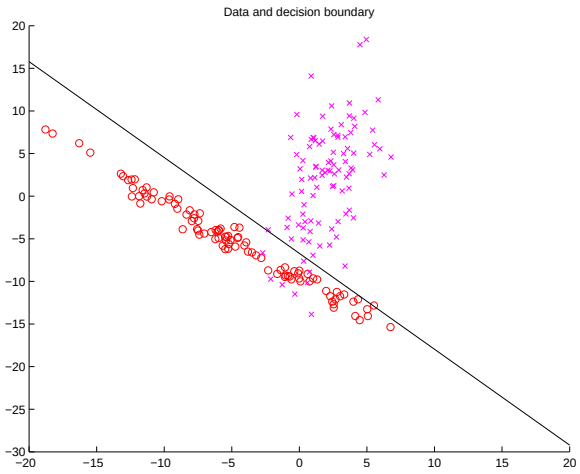

Figure 2: Classification boundary for logistic regression.

8.3 Artificial Neural Networks

Logistic regression works for linearly separable datasets, but may not be sufficient for more complex cases. We can generalize logistic regression by replacing the linear function $\mathbf{w}^T\mathbf{x} + b$ with any other function. If we replace it with a neural network, we get:

$$P(C_1|\mathbf{x}) = g\left(\sum_j w_j^{(1)} g\left(\sum_k w_{k,j}^{(2)} x_k + b_j^{(2)}\right) + b^{(1)}\right) \tag{16}$$

This representation is no longer connected to any particular choice of class-conditional model; it is purely a model of the class probability given the measurement.

8.4 K-Nearest Neighbors Classification

We can apply the KNN idea to classification as well. For class labels $\{-1, 1\}$, the classifier is:

$$y_{new} = \text{sign}\left(\sum_{i \in N_K(\mathbf{x})} y_i\right) \tag{17}$$

where

$$\text{sign}(z) = \begin{cases} -1 & z \leq 0 \\ 1 & z > 0 \end{cases} \tag{18}$$

Alternatively, we might take a weighted average of the K-nearest neighbors:

$$y = \text{sign}\left(\sum_{i \in N_K(\mathbf{x})} w(\mathbf{x}_i) y_i\right), \quad w(\mathbf{x}_i) = e^{-||\mathbf{x}_i - \mathbf{x}||^2/2\sigma^2} \tag{19}$$

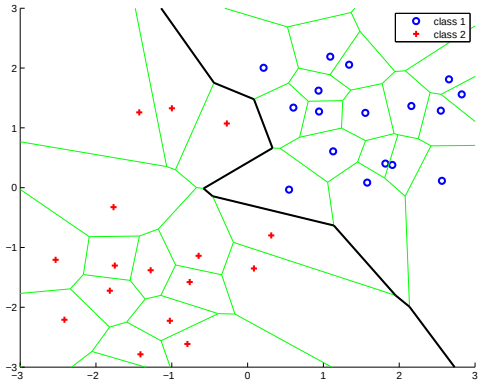

Figure 3: For two classes and planar inputs, the decision boundary for a 1NN classifiifier (the bold black curve) is a subset of the perpendicular bisecting line segments (green) between pairs of neighbouring points (obtained with a Voronoi tesselation).

where σ^2 is an additional parameter to the algorithm.

For KNN the decision boundary will be a collections of hyperplane patches that are perpendicular bisectors of pairs of points drawn from the two classes. As illustrated in Figure 3, this is a set of bisecting line segments for 2D inputs. Figure 3, shows a simple case but it is not hard to imagine that the decision surfaces can get very complex, e.g., if a point from class 1 lies somewhere in the middle of the points from class 2. By increasing the number of nearest neighbours (i.e., K) we are effectively smoothing the decision boundary, hopefully thereby improving generalization.

8.5 Generative vs. Discriminative models

The classifiers described here illustrate a distinction between two general types of models in machine learning:

1. **Generative models**, such as the GCC, describe the complete probability of the data $p(\mathbf{x}, y)$.

2. **Discriminative models**, such as LR, ANNs, and KNN, describe the conditional probability of the output given the input: $p(y|\mathbf{x})$

The same distinction occurs in regression and classification, e.g., KNN is a discriminative method that can be used for either classification or regression.

The distinction is clearest when comparing LR with GCC with equal covariances, since they are both linear classifiers, but the training algorithms are different. This is because they have different goals; LR is optimized for classification performance, where as the GCC is a "complete" model of the probability of the data that is then pressed into service for classification. As a consequence, GCC may perform poorly with non-Gaussian data. Conversely, LR is not premised on any particular form of distribution for the two class distributions. On the other hand, LR can **only** be

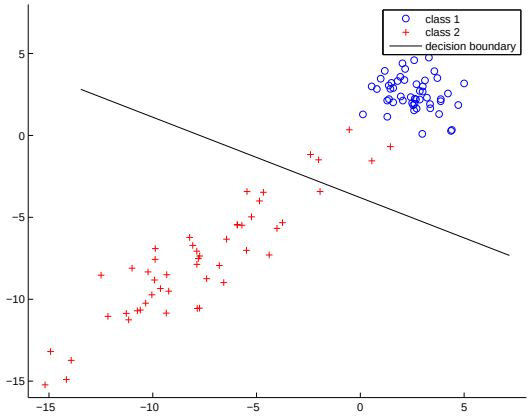

Figure 4: In this example there are two classes, one with a small isotropic covariance, and one with an anistropic covariance. One can clearly see that the data are linearly separable (i.e., a line exists that correctly separates the input training samples). Despite this, LS regression does not separate the training data well. Rather, the LS regression decision boundary produces 5 incorrectly classified training points.

used for classification, whereas the GCC can be used for other tasks, e.g., to sample new $\mathbf{x}$ data, to classify noisy inputs or inputs with outliers, and so on.

The distinctions between generative and discriminative models become more significant in more complex problems. Generative models allow us to put more prior knowledge into how we build the model, but classification may often involve difficult optimization of $p(y|\mathbf{x})$; discriminative methods are typically more efficient and generic, but are harder to specialize to particular problems.

8.6 Classification by LS Regression

One tempting way to perform classification is with least-squares rgression. That is, we could treat the class labels $y \in \{-1, 1\}$ as real numbers, and estimate the weights by minimizing

$$E(\mathbf{w}) = \sum_i (y_i - \mathbf{x}_i^T \mathbf{w})^2 \,, \tag{20}$$

for labeled training data $\{\mathbf{x}_i, y_i\}$. Given the optimal regression weights, one could then perform regression on subsequent test inputs and use the sign of the output to determine the output class.

In simple cases this can perform well, but in general it will perform poorly. This is because the objective function in linear regression measures the distance from the modeled class labels (which can be any real number) to the true class labels, which may not provide an accurate measure of how well the model has classified the data. For example, a linear regression model will tend to produce predicted labels that lie outside the range of the class labels for "extreme" members of a given class (e.g. 5 when the class label is 1), causing the error to be measured as high even when the classification (given, say, by the sign of the predicted label) is correct. In such a case the decision

boundary may be shifted towards such an extreme case, potentially reducing the number of correct classifications made by the model. Figure 4 demonstrates this with a simple example.

The problem arises from the fact that the constraint that $y \in (-1, 1)$ is not built-in to the model (the regression algorithm knows nothing about it), and so wastes considerable representational power trying to reproduce this effect. It is much better to build this constraint into the model.

8.7 Naïve Bayes

One problem with class conditional models, as described above, concerns the large number of parameters required to learn the likelihood model, i.e., the distribution over the inputs conditioned on the class. In Gaussian Class Conditional models, with d-dimensional input vectors, we need to estimate the class mean and class covariance matrix for each class. The mean will be a d-dimensional vector, but the number of unknowns in the covariance matrix grows quadratically with d. That is, the covariance is a $d \times d$ matrix (although because it is symmetric we do not need to estimate all d^2 elements).

Naïve Bayes aims to simplify the estimation problem by assuming that the different input features (e.g., the different elements of the input vector), are conditionally independent. That is, they are assumed to be independent when conditioned on the class. Mathematically, for inputs $\mathbf{x} \in \mathbb{R}^d$, we express this as

$$p(\mathbf{x}|C) \;=\; \prod_{i=1}^{d} p(x_i|C) \;. \tag{21}$$

With this assumption, rather than estimating one d-dimensional density, we instead estimate d 1-dimensional densities. This is important because each 1D Gaussian only has two parameters, its mean and variance, both of which are scalars. So the model has $2d$ unknowns. In the Gaussian case, the Naïve Bayes model effectively replaces the general $d \times d$ covariance matrix by a diagonal matrix. There are d entries along the diagonal of the covariance matrix; the i^{th} entry is the variance of $x_i|C$. This model is not as expressive but it is much easier to estimate.

8.7.1 Discrete Input Features

Up to now, we have looked at algorithms for real-valued inputs. We now consider the Naïve Bayes classification algorithm for discrete inputs. In discrete Naïve Bayes, the inputs are a discrete set of "features", and each input either has or doesn't have each feature. For example, in document classification (including spam filtering), a feature might be the presence or absence of a particular word, and the feature vector for a document would be a list of which words the document does or doesn't have.

Each data vector is described by a list of discrete features $F_{1:D} = [F_1, ..., F_D]$. Each feature F_i has a set of possible values that it can take; to keep things simple, we'll assume that each feature is binary: $F_i \in \{0, 1\}$. In the case of document classification, each feature might correspond to the presence of a particular word in the email (e.g., if $F_3 = 1$, then the email contains the word

"business"), or another attribute (e.g., $F_4 = 1$ might mean that the mail headers appear forged). Similarly, a classifier to distinguish news stories between sports and financial news might be based on particular words and phrases such as "team," "baseball," and "mutual funds."

To understand the complexity of discrete class conditional models in general (i.e., without using the Naïve Bayes model), consider the distribution over 3 inputs, for class $C = 1$, i.e., $P(F_{1:3} \,|\, C = 1)$. (There will be another model for $C = 0$, but for our little thought experiment here we'll just consider the model for $C = 1$.) Using basic rules of probability, we find that

$$\begin{aligned} P(F_{1:3} \,|\, C = 1) &= P(F_1 \,|\, C = 1, F_2, F_3)\, P(F_2, F_3 \,|\, C = 1) \\ &= P(F_1 \,|\, C = 1, F_2, F_3)\, P(F_2 \,|\, C = 1, F_3)\, P(F_3 \,|\, C = 1) \end{aligned} \tag{22}$$

Now, given $C = 1$ we know that F_3 is either 0 or 1 (ie. it is a coin toss), and to model it we simply want to know the probability $P(F_3 = 1 \,|\, C = 1)$. Of course the probability that $F_3 = 0$ is simply $1 - P(F_3 = 1 \,|\, C = 1)$. In other words, with one parameter we can model the third factor above, $P(F_3 \,|\, C = 1)$.

Now consider the second factor $P(F_2 \,|\, C = 1, F_3)$. In this case, because F_2 depends on F_3, and there are two possible states of F_3, there are two distributions we need to model, namely $P(F_2 \,|\, C = 1, F_3 = 0)$ and $P(F_2 \,|\, C = 1, F_3 = 1)$. Acordingly, we will need two parameters, one for $P(F_2 = 1 \,|\, C = 1, F_3 = 0)$ and one for $P(F_2 = 1 \,|\, C = 1, F_3 = 1)$. Using the same logic, to model $P(F_1 \,|\, C = 1, F_2, F_3)$ will require one model parameter for each possible setting of (F_2, F_3), and of course there are 2^2 such settings. For D-dimensional binary inputs, there are $O(2^{D-1})$ parameters that one needs to learn. The number of parameters required grows prohibitively large as D increases.

The Naïve Bayes model, by comparison, only have D parameters to be learned. The assumption of Naïve Bayes is that the feature vectors are all conditionally independent given the class. The independence assumption is often very naïve, but yet the algorithm often works well nonetheless. This means that the likelihood of a feature vector for a particular class j is given by

$$P(F_{1:D}|C = j) = \prod_i P(F_i|C = j) \tag{23}$$

where C denotes a class $C \in \{1, 2, ...K\}$. The probabilities $P(F_i|C)$ are parameters of the model:

$$P(F_i = 1|C = j) = a_{i,j} \tag{24}$$

We must also define class priors $P(C = j) = b_j$.

To classify a new feature vector using this model, we choose the class with maximum probability given the features. By Bayes' Rule this is:

$$P(C = j|F_{1:D}) = \frac{P(F_{1:D}|C = j)P(C = j)}{P(F_{1:D})} \tag{25}$$

$$= \frac{\left(\prod_i P(F_i|C = j)\right) P(C = j)}{\sum_{\ell=1}^K P(F_{1:D}, C = \ell)} \tag{26}$$

$$= \frac{\left(\prod_{i:F_i=1} a_{i,j} \prod_{i:F_i=0} (1 - a_{i,j})\right) b_j}{\sum_{\ell=1}^K \left(\prod_{i:F_i=1} a_{i,\ell} \prod_{i:F_i=0} (1 - a_{i,\ell})\right) b_\ell} \tag{27}$$

If we wish to find the class with maximum posterior probability, we need only compute the numerator. The denominator in (27) is of course the same for all classes j. To compute the denominator one simply divides the numerators for each class by their sum.

The above computation involves the product of many numbers, some of which might be quite small. This can lead to underflow. For example, if you take the product $a_1 a_2 ... a_N$, and all $a_i << 1$, then the computation may evaluate to zero in floating point, even though the final computation after normalization should not be zero. If this happens for all classes, then the denominator will be zero, and you get a divide-by-zero error, even though, mathematically, the denominator cannot be zero. To avoid these problems, it is safer to perform the computations in the log-domain:

$$\alpha_j = \left(\sum_{i:F_i=1} \ln a_{i,j} + \sum_{i:F_i=0} \ln(1 - a_{i,j}) \right) + \ln b_j \tag{28}$$

$$\gamma = \max_j \alpha_j \tag{29}$$

$$P(C = j | F_{1:D}) = \frac{\exp(\alpha_j - \gamma)}{\sum_\ell \exp(\alpha_\ell - \gamma)} \tag{30}$$

which, as you can see by inspection, is mathematically equivalent to the original form, but will not evaluate to zero for at least one class.

8.7.2 Learning

For a collection of N training vectors F_k, each with an associated class label C_k, we can learn the parameters by maximizing the data likelihood (i.e., the probability of the data given the model). This is equivalent to estimating multinomial distributions (in the case of binary features, binomial distributions), and reduces to simple counting of features.

Suppose there are N_k examples of each class, and N examples total. Then the prior estimate is simply:

$$b_k = \frac{N_k}{N} \tag{31}$$

Similarly, if class k has $N_{i,k}$ examples where $F_i = 1$, then

$$a_{i,k} = \frac{N_{i,k}}{N_k} \tag{32}$$

With large numbers of features and small datasets, it is likely that some features will never be seen for a class, giving a class probability of zero for that feature. We might wish to regularize, to prevent this extreme model from occurring. We can modify the learning rule as follows:

$$a_{i,k} = \frac{N_{i,k} + \alpha}{N_k + 2\alpha} \tag{33}$$

for some small value α. In the extreme case where there are no examples for which feature i is seen for class k, the probability $a_{i,k}$ will be set to $1/2$, corresponding to no knowledge. As the number of examples N_k becomes large, the role of α will become smaller and smaller.

In general, given in a multinomial distribution with a large number of classes and a small training set, we might end up with estimates of prior probability b_k being zero for some classes. This might be undesirable for various reasons, or be inconsistent with our prior beliefs. Again, to avoid this situation, we can regularize the maximum likelihood estimator with our prior believe that all classes should have a nonzero probability. In doing so we can estimate the class prior probabilities as

$$b_k = \frac{N_k + \beta}{N + K\beta} \tag{34}$$

for some small value of β. When there are no observations whatsoever, all classes are given probability $1/K$. When there are observations the estimated probabilities will lie between N_k/N and $1/K$ (converging to N_k/N as $N \to \infty$).

Derivation. Here we derive just the per-class probability assuming two classes, ignoring the feature vectors; this case reduces to estimating a binomial distribution. The full estimation can easily be derived in the same way.

Suppose we observe N examples of class 0, and M examples of class 1; what is b_0, the probability of observing class 0? Using maximum likelihood estimation, we maximize:

$$\prod_i P(C_i = k) = \left(\prod_{i:C_i=0} P(C_i = 0)\right)\left(\prod_{i:C_i=1} P(C_i = 1)\right) \tag{35}$$

$$= b_0^N b_1^M \tag{36}$$

Furthermore, in order for the class probabilities to be a valid distribution, it is required that $b_0 + b_1 = 1$, and that $b_k \geq 0$. In order to enforce the first constraint, we set $b_1 = 1 - b_0$:

$$\prod_i P(C_i = k) = b_0^N (1 - b_0)^M \tag{37}$$

The log of this is:

$$L(b_0) = N \ln b_0 + M \ln(1 - b_0) \tag{38}$$

To maximize, we compute the derivative and set it to zero:

$$\frac{dL}{db_0} = \frac{N}{b_0} - \frac{M}{1 - b_0} = 0 \tag{39}$$

Multiplying both sides by $b_0(1 - b_0)$ and solving gives:

$$b_0^* = \frac{N}{N + M} \tag{40}$$

which, fortunately, is guaranteed to satisfy the constraint $b_0 \geq 0$.

