Chapter 5

Naive Bayes

In the last chapter, we introduced the idea of maximum likelihood. However, we only considered very simple (i.e., univariate or one-dimensional) estimation problems.

In this chapter, we will extend the maximum likelihood idea to the binary classification setting. Our goal in this setting is to model the distribution

$$P(y = 1 \mid \mathbf{x}), \tag{5.1}$$

i.e., the conditional probability that the output is true ($y = 1$) given some input feature $\mathbf{x}$.

5.1 Motivating Example and Setup

As a motivating example we will continue to build upon the spam classification task discussed in the previous chapter. However, rather than just naively counting how often previous emails were spam (as in the last chapter), in this chapter, we will assume that our goal is to classify spam by conditioning on the content of the email. In particular, we will assume that $y = 1$ corresponds to labeling an email as spam, while $y = 0$ corresponds to a non-spam email. To solve this task, we are given a dataset $\mathcal{D} = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), ...(\mathbf{x}_n, y_n)\}$, which contains a set of labelled emails for which we know whether they are spam (whether $y_i = 1$ or $y_i = 0$) and for which we have extracted an m-dimensional feature vector $\mathbf{x}_i$. For most of this chapter, we will assume that the features are *binary*, meaning that $\mathbf{x}_i \in \{0, 1\}^m$. Thus, $\mathbf{x}_i[j] = x_{i,j} = 1$ indicates that ith example in the dataset $\mathcal{D}$ contains feature j. In the spam email classification example this might indicate that the ith email contains some particular trait, e.g., that the email contains content in all-caps font.

We will also assume that we have split our dataset $\mathcal{D}$ into two partitions $\mathcal{D}_{\text{train}}$ and $\mathcal{D}_{test}$, where $\mathcal{D} = \mathcal{D}_{\text{train}} \cup \mathcal{D}_{\text{test}}$ and $\mathcal{D}_{\text{train}} \cap \mathcal{D}_{\text{test}} = \emptyset$. As usual, we use the training set $\mathcal{D}_{\text{train}}$ to train our model and the second set $\mathcal{D}_{test}$ to

evaluate how good our model is. (Recall that we split the data in this way so that the model cannot cheat by simply memorizing the training data).

We will use $\hat{y}_i$ to denote the predicted value output by our model for a specific example i, and similarly we will often use $\hat{P}(y \mid \mathbf{x}_i)$ to denote the estimated conditional distribution learned by our model. In other words, we will use $\hat{}$ as a superscript to distinguish between the estimates made by our model and the ground truth. Generally, we will assume that

$$\hat{y}_i = \begin{cases} 1 & \text{if } \hat{P}(y \mid \mathbf{x}_i) > 0.5. \\ 0 & \text{otherwise} \end{cases} \tag{5.2}$$

In the spam example, this means that our model will predict $\hat{y}_i = 1$ (i.e., predict that an email is spam) if and only if our model thinks that the email has more than a 50% chance of being spam.[1]

5.2 Features, Class Priors, and Bayes Rule

In order to model the conditional distribution $P(y = 1|\mathbf{x})$, we must consider two important sources of information.

Feature likelihood

The first source of information is from the *features*. Intuitively, some features $\mathbf{x}$ are more likely to occur when $y = 1$ while other features are more likely to occur when $y = 0$. In our example of spam classification, we might expect that spelling errors are more common in spam emails, and supposing that our jth feature is a binary value indicating whether a spelling error occurs in the email, this would mean that

$$P(\mathbf{x}[j] = 1 \mid y = 1) > P(\mathbf{x}[j] = 1 \mid y = 0). \tag{5.3}$$

That is, the probability of seeing a spelling error in a spam email $P(\mathbf{x}[j] = 1 \mid y = 1)$ is larger than the probability of seeing a spelling error in a non-spam email $P(\mathbf{x}[j] = 1 \mid y = 0)$.

In general, for each of the $j \in [m]$ different binary features[2], we could estimate

$$\hat{P}(\mathbf{x}[j] \mid y = 1), \tag{5.4}$$

which would tell us how likely we think these features are to occur for examples from the positive class. Note that we also need to separately estimate $\hat{P}(\mathbf{x}[j]|y = 0)$, i.e., how likely the features are from the negative class, since

$$P(\mathbf{x}[j] \mid y = 0) \neq 1 - P(\mathbf{x}[j] \mid y = 1). \tag{5.5}$$

[1] In general, we can choose different thresholds instead of 50%, depending on the application and our goals. For example, if we want to minimize the number of false positives—i.e., cases where $\hat{y}_i = 1$ but $y = 0$—then we might want to set a higher threshold. This issue is sometimes known as calibration.

[2] Here, I use the shorthand notation $[m]$ to indicate the set of integers $\{0, 2, ..., m - 1\}$.

Once we have estimates for each of the different features, we can then combine this information together to estimate how likely an entire feature vector is to occur for a given class, i.e., $\hat{P}(\mathbf{x} \mid y = 1)$. Note that now we are estimating the probability of the full feature vector $\mathbf{x}$ rather than just a single entry in this vector $\mathbf{x}[j]$. A simple assumption we can make is that all the features are *conditionally independent* given the class. Intuitively, this means that the occurrence of one feature does not influence the occurrence of another, and formally, this means that we can estimate $\hat{P}(\mathbf{x}|y)$ as

$$\hat{P}(\mathbf{x} \mid y) = \prod_{j \in [m]} \hat{P}(\mathbf{x}[j] \mid y). \tag{5.6}$$

Based on this idea, given some input features for a training or testing example $\mathbf{x}_i$, we can figure out whether the features are more likely to come from the positive or negative class, and we call

$$P(\mathbf{x} \mid y) \tag{5.7}$$

the *feature likelihood*.

Naive Bayes Assumption In order to compute the feature likelihood in the above example, we made the assumption that all the features $\mathbf{x}[j]$ are *conditionally independent* given the class variable y. This is commonly known as the *Naive Bayes assumption*, and it is very useful for facilitating efficient learning. The key idea here is the assumption that the joint distribution over the full set of m features $P(\mathbf{x}|y)$ *factorizes* into a product of m independent distributions:

$$P(\mathbf{x}|y) = \prod_{j \in [m]} P(\mathbf{x}[j] \mid y). \tag{5.8}$$

So, what does this assumption mean? The key idea is that we are assuming all the features are independent from each other, given knowledge of the underlying class. For example, suppose we have an email that we know is spam (i.e., it is a labelled example in the training set), under the Naive Bayes assumption the occurrence of spelling errors and all-caps in this email should be uncorrelated. In other words, if I have an email, and I know this email is spam (or not), then seeing a spelling error should have no influence on the expected likelihood of seeing other features.

Note, however, that the *conditional* aspect of this independence is critical. The features are only assumed to be independent if we already know the underlying class. For instance, if I have an *unlabeled* example email and see a spelling error, then this would probably increase my expected likelihood of seeing the usage of all-caps, since both these features are correlated with the underlying label (i.e., whether the email is spam). The essential idea in Naive Bayes is that we assume the correlation between features is completely explained by the underlying class variable.

Moreover, it is also important to note that the Naive Bayes assumption is just an assumption. It is entirely possible that our true data distribution violates this assumption. Nonetheless, it is a useful assumption to make. Indeed, most—if not all—the assumptions we will make about our data are wrong, but many assumptions are useful and making assumptions is critical to make machine learning work.

So, what makes the Naive Bayes assumption particularly useful? The Naive Bayes assumption is so useful because it drastically simplifies our learning problem. The full joint distribution $P(\mathbf{x} \mid y)$ is potentially very complicated. In the worst case, learning about this distribution could require storing information about all 2^m distinct possible combinations of features (i.e., all possible combinations of m binary features). However, under the Naive Bayes assumption, we only need to store information about m independent binary outcomes.

Class priors

Estimating how prevalent different features are for examples from different classes gives us one source of information. For example, we might observe many features that indicate an email looks like spam (e.g., an email might contain spelling mistakes, suspicious links, and requests for personal information).

However, we also need to keep in mind how frequent spam emails are in general in our dataset. For instance, it might be the case that for a particular company's corporate email server, the average inbox contains only 1% spam. In contrast, for a personal email on a public domain server the proportion of spam might be as high as 50%. In the former case, we might be reluctant to classify something as spam, even if it has many features that indicate spam, but this is not true in the latter setting.

This information about the overall likelihood of spam in a particular dataset is captured via the notion of *class priors*. In particular, for a particular dataset we can estimate

$$\hat{P}(y = 1), \tag{5.9}$$

which gives us an estimate of how likely each class is *independent of observing any features*. We call this a class *prior* because it can be interpreted as our prior expectation of observing a particular class before seeing any evidence from the features.

5.2.1 Combining Evidence via Bayes Rules

So, suppose we have estimated how likely the features are for each class—i.e., $\hat{P}(\mathbf{x}|y = 1)$—as well as the prior for each class—i.e., $\hat{P}(y = 1)$. How do we combine this information to make a classification? Here, we rely on the fact that

$$P(y = 1|\mathbf{x}) = \frac{P(\mathbf{x}|y = 1)P(y = 1)}{P(\mathbf{x})}, \tag{5.10}$$

which follows from Bayes rule. And, note that the numerator of Equation (5.10) is simply the product of the class prior and the feature likelihood! The denominator $P(\mathbf{x})$, on the other hand, measures the *marginal likelihood* of the feature vector $\mathbf{x}$—i.e., how likely these features are in general, independent of the different classes.

Luckily, we do not need to worry about the denominator, since it is unrelated to the class variable y. More formally, if all we want to know is whether $P(y = 1|\mathbf{x}) > P(y = 0|\mathbf{x})$, then we can ignore the denominator in Equation (5.10), since the relative likelihood of these two possibilities does not depend on the denominator:

$$\frac{P(y=1|\mathbf{x})}{P(y=0|\mathbf{x})} = \frac{\frac{P(\mathbf{x}|y=1)P(y=1)}{P(\mathbf{x})}}{\frac{P(\mathbf{x}|y=0)P(y=0)}{P(\mathbf{x})}} \tag{5.11}$$

$$= \frac{P(\mathbf{x}|y=1)P(y=1)}{P(\mathbf{x}|y=0)P(y=0)}. \tag{5.12}$$

The term $\frac{P(y=1|\mathbf{x})}{P(y=0|\mathbf{x})}$ is often known as the *odds ratio*, since it tells us which class is more likely given the features.[3]

5.3 Naive Bayes Algorithm

The previous subsection introduced the key concepts behind the Naive Bayes approach. In particular, we saw how we can use Bayes rule to combine information about the feature likelihoods $P(\mathbf{x} \mid y)$ and the class priors $P(y)$ to classify an example using the odds ratio, i.e.,

$$\frac{P(y=1|\mathbf{x})}{P(y=0|\mathbf{x})} = \frac{P(\mathbf{x}|y=1)P(y=1)}{P(\mathbf{x}|y=0)P(y=0)}. \tag{5.13}$$

In this section, we will describe how to estimate the feature likelihoods and class priors using maximum likelihood.

5.3.1 Learning via Maximum Likelihood

Our goal is to determine the feature likelihoods and class priors for both classes. Under the Naive Bayes assumption and since we are assuming only binary features, this corresponds to estimating the parameters of $2m+1$ distinct Bernoulli[4] distributions:

- One Bernoulli distribution $P(\mathbf{x}[j] \mid y = k)$ for each feature $j \in [m]$ conditioned on each class $y \in \{0, 1\}$. Since we need to estimate two distributions for each of the m features—i.e., $P(\mathbf{x}[j] = 1|y = 1)$ and $P(\mathbf{x}[j] = 1|y = 0)$—this amounts to $2m$ distributions in total.

[3] Note that if we want to estimate exact probabilities, rather than relative likelihoods, then we do need to estimate the denominator.

[4] Recall that Bernoulli distribution is the formal name for the distribution of a binary random variable.

- One distribution for the class prior, i.e., $P(y = 1)$. Note we only need to estimate one distribution even though there are two classes, since $P(y = 0) = 1 - P(y = 1)$.

In terms of a formal likelihood model, the likelihood of Naive Bayes decomposes as a product of the independent Bernoulli likelihoods:

$$\mathcal{L}(\Theta, \mathcal{D}_{\text{train}}) = \prod_{(\mathbf{x}_i, y_i) \in \mathcal{D}_{\text{train}}} P_\Theta(y_i \mid \mathbf{x}_i) \tag{5.14}$$

$$= \prod_{(\mathbf{x}_i, y_i) \in \mathcal{D}_{\text{train}}} \frac{P_\Theta(y_i) \prod_{j \in [m]} P_\Theta(\mathbf{x}_i[j] \mid y_i)}{P(\mathbf{x}_i[j])} \tag{5.15}$$

$$\propto \prod_{(\mathbf{x}_i, y_i) \in \mathcal{D}_{\text{train}}} P_\Theta(y_i) \prod_{j \in [m]} P_\Theta(\mathbf{x}_i[j] \mid y_i), \tag{5.16}$$

where in the last expression we remove the $P(\mathbf{x})$ term from the denominator, since it does not depend on the model parameters. The log-likelihood can then be written as

$$\log \mathcal{L}(\Theta, \mathcal{D}_{\text{train}}) \propto \sum_{(\mathbf{x}_i, y_i) \in \mathcal{D}_{\text{train}}} \log P_\Theta(y_i) + \sum_{j \in [m]} \log P_\Theta(\mathbf{x}_i[j] \mid y_i), \tag{5.17}$$

where again each term in the sum $\log P_\Theta(y_i) + \sum_{j \in [m]} \log P_\Theta(\mathbf{x}_i[j] \mid y_i)$ comes from an independent Bernoulli distribution.

As discussed in the previous chapter, Bernoulli distributions are one of the simplest distributions and are determined by a single parameter θ, which (without loss of generality) determines the probability of seeing a 1, i.e, $P(x = 1) = \theta$. So, in our case, the likelihood is defined by $2m + 1$ different θ parameters. To keep things clear, we will use subscripts to differentiate between the different parameters: $\theta_{j,k}$ will denote the parameter for the feature distribution $P(\mathbf{x}[j] \mid y = k) = \theta_{j,k}$ and θ_1 will denote the parameter for the class prior $P(y = 1) = \theta_1$.

To estimate these parameters, we rely on the maximum likelihood estimate of the Bernoulli that we derived from the previous chapter. Note that the fact that all the Bernoulli's are *independent* is critical to make this analysis possible. For example, consider the case of the $\theta_{j,k}$ parameters. Our goal is to find the $\theta_{j,k}$ parameter so that the log-likelihood is maximized, i.e., where

$$\frac{\partial \log \mathcal{L}(\Theta, \mathcal{D}_{\text{train}})}{\partial \theta_{j,k}} = 0. \tag{5.18}$$

Now, without loss of generality and to simplify notation, assume we are solving the above equation for a specific $\theta_{j,k}$ parameter where $j = 1$ and $k = 1$, i.e., $\theta_{1,1}$, which corresponds to the parameter that estimates the likelihood of seeing

the first feature given that we have an example from class 1. We have that

$$\frac{\partial \log \mathcal{L}(\Theta, \mathcal{D}_{\text{train}})}{\partial \theta_{1,1}}$$
$$= \frac{\partial}{\partial \theta_{1,1}} \left(\sum_{(\mathbf{x}_i, y_i)} \log(P_\Theta(y_i)) - \log(P(\mathbf{x})) + \sum_{j \in [m]} \log(P_\Theta(\mathbf{x}_i[j] \mid y_i)) \right).$$

However, because of the independence between the distributions only $P_\theta(\mathbf{x}_i[1]|y_i)$ depends on the parameter $\theta_{1,1}$! Thus, we have that

$$\begin{aligned} \frac{\partial \log \mathcal{L}(\Theta, \mathcal{D}_{\text{train}})}{\partial \theta_{1,1}} &= \sum_{(\mathbf{x}_i, y_i)} \frac{\partial}{\partial \theta_{1,1}} \log(P_\Theta(\mathbf{x}_i[j] \mid y_i)) \\ &= \sum_{(\mathbf{x}_i, y_i)} \frac{\partial}{\partial \theta_{1,1}} \log \left(\theta_{1,1}^{\mathbf{x}_*[j]} (1 - \theta_{1,1})^{(1 - \mathbf{x}_*[j])} \right) \end{aligned}$$

And, as we saw from the previous chapter, we differentiate and solve for this expression to be equal to 0 and obtain

$$\theta_{1,1} = \frac{|\{\mathbf{x}_i, y_i) \in \mathcal{D}_{\text{train}} : \mathbf{x}_i[1] = 1, y_i = 1\}|}{|\{\mathbf{x}_i, y_i) \in \mathcal{D}_{\text{train}} : y_i = 1\}|}, \tag{5.19}$$

i.e., the maximum likelihood solution for $\theta_{1,1}$ is simply equal to the proportion of points from class 1 that have feature 1 present.

Generalizing this same idea to all the parameters, we have that

$$\hat{\theta}_{j,k} = \frac{|\{\mathbf{x}_i, y_i) \in \mathcal{D}_{\text{train}} : \mathbf{x}_i[j] = 1, y_i = k\}|}{|\{\mathbf{x}_i, y_i) \in \mathcal{D}_{\text{train}} : y_i = k\}|} \tag{5.20}$$

and

$$\hat{\theta}_1 = \frac{|\{\mathbf{x}_i, y_i) \in \mathcal{D}_{\text{train}} : y_i = 1\}|}{|\mathcal{D}_{\text{train}}|}. \tag{5.21}$$

In other words, we can simply estimate the feature likelihood $\hat{\theta}_{j,k} = \hat{P}(\mathbf{x}[j] = 1 \mid y = k)$ by counting how often that feature occurred in training examples with label $y = k$, and we can estimate the class prior $\hat{\theta}_1 = P(y = 1)$ by simply counting what proportion of our training data has label $y = 1$.

Smoothing One subtle point about Naive Bayes parameter estimation, is that a single feature can have a large impact on our predictions. For example, suppose we estimate that $\theta_{j,k} = P_\Theta(\mathbf{x}[j] = 1|y = k) = 0$, i.e., there are no training examples from class k that contain feature j. In this case, we will end up with a model that *always* predicts $P(y = k|\mathbf{x}) = 0$ for points that have $\mathbf{x}[j] = 0$. In other words, if we estimate that a feature has zero likelihood for a certain class, then we will always predict a zero probability for that class whenever we see that feature. This behavior is somewhat extreme and can lead to undesirable behaviors.

To combat this issue—and improve the stability of Naive Bayes overall—it is common for researchers to *smooth* the parameter estimates. Intuitively, smoothing involves adding fake counts to our data to stabilize the model. Formally, smoothing involves replacing the maximum likelihood parameter estimates with the following:

$$\hat{\theta}_{j,k} = \frac{|\{\mathbf{x}_i, y_i) \in \mathcal{D}_{\text{train}} : \mathbf{x}_i[j] = 1, y_i = k\}| + \alpha}{|\{\mathbf{x}_i, y_i) \in \mathcal{D}_{\text{train}} : y_i = k\}| + m\alpha}, \tag{5.22}$$

where $\alpha \in \mathbb{R}^+$ is a smoothing hyperparameter and m is the number of features in our dataset. The intuition is that we are (implicitly) adding α "fake" training examples containing each feature for each class. The most popular form of smoothing is known as *Laplace* or *add-one* smoothing, and in this approach, we set $\alpha = 1$. By using a positive smoothing term, we ensure that our model never predicts zero probability for the likelihood of any feature.

5.3.2 Making a Prediction

After we have computed the Naive Bayes parameters from the training data, we can then make a prediction on a new datapoint by plugging our estimated parameters into the odds ratio (Equation 5.11). In particular, suppose we are given some feature input $\mathbf{x}_*$ for an unlabeled point, we can compute the odds ratio as follows:

$$\frac{\hat{P}(y=1|\mathbf{x}_*)}{\hat{P}(y=0 \mid \mathbf{x}_*)} = \frac{\hat{P}(\mathbf{x}_*|y=1)\hat{P}(y=1)}{\hat{P}(\mathbf{x}_*|y=0)\hat{P}(y=0)} \tag{5.23}$$

$$= \frac{\prod_{j\in[m]} \hat{P}(\mathbf{x}_*[j] \mid y=1)\hat{P}(y=1)}{\prod_{j\in[m]} \hat{P}(\mathbf{x}_*[j] \mid y=0)\hat{P}(y=0)} \tag{5.24}$$

$$= \frac{\prod_{j\in[m]} \hat{\theta}_{j,1}^{\mathbf{x}_*[j]}(1-\hat{\theta}_{j,1})^{(1-\mathbf{x}_*[j])}\hat{\theta}_1}{\prod_{j\in[m]} \hat{\theta}_{j,0}^{\mathbf{x}_*[j]}(1-\hat{\theta}_{j,0})^{(1-\mathbf{x}_*[j])}(1-\hat{\theta}_1)}. \tag{5.25}$$

The final expression in Equation (5.25) gives our *estimated odds* that an example point with features $\mathbf{x}_*$ comes from class 1 over class 0. In terms of our predictive model, we can then classify new points using this estimated odds-ratio.

$$f_\Theta(\mathbf{x}_*) = \begin{cases} 1 & \text{if } \frac{\hat{P}(y=1|\mathbf{x}_*)}{\hat{P}(y=0 \mid \mathbf{x}_*)} > \gamma \\ 0 & \text{if } \frac{\hat{P}(y=1|\mathbf{x}_*)}{\hat{P}(y=0 \mid \mathbf{x}_*)} < \gamma \end{cases}, \tag{5.26}$$

where the $\gamma \in \mathbb{R}^+$ is a threshold hyperparameter. Usually, we simply set $\gamma = 1$, which specifies that we will simply predict whichever class has a higher odds. Sometimes, we might choose $\gamma \neq 1$ in order to calibrate our model for some application. For instance, in the spam classification example, we might find

that some users prefer a less aggressive filter, in which case we could increase the threshold γ required to make a positive prediction.

Odds and log-odds

One issue when computing the estimated odds ratio in Equation (5.25) is numerical stability. Computing both the numerator and the denominator involves multiplying a series of probabilities or likelihoods, which are all numbers between 0 and 1. If we have many features, these product of many small numbers can end up being very small and even cause underflow issues! For this reason—as well as other factors of mathematical convenience—we tend to work with the *log-odds ratio*:

$$\log\left(\frac{\hat{P}(y=1|\mathbf{x}_*)}{\hat{P}(y=0\mid\mathbf{x}_*)}\right) = \log(\hat{\theta}_1) + \sum_{j\in[m]} \mathbf{x}_*[j]\log(\hat{\theta}_{j,1}) + (1-\mathbf{x}_*[j])\log(1-\hat{\theta}_{j,1}) - \log(1-\hat{\theta}_1) - \sum_{j\in[m]} \mathbf{x}_*[j]\log(\hat{\theta}_{j,0}) + (1-\mathbf{x}_*[j])\log(1-\hat{\theta}_{j,0}). \tag{5.27}$$

As with likelihoods, the key benefit for this equation is that we can compute the odds ratio by summing log-probabilities, which is generally more numerically convenient than multiplying probability values.

Also, note that the natural decision boundary for the log-odds ratio is 0: the log-odds ratio will be zero if and only if the two classes have equal probability; it will be positive if the positive class has higher probability; and, it will be negative if the negative class has higher probability.

Log-odds as a linear decision boundary

Note that the log-odds ratio of Naive Bayes actually defines a linear decision boundary. If the log-odds is greater than some threshold (usually zero), then we classify as a positive point; otherwise, we classify the point as negative. And, one can verify that the log-odds is a *linear* function of the input features, with feature coefficients equal to

$$w_j = \log(\hat{\theta_{j,1}}) - \log(1-\hat{\theta_{j,1}}) - \log(\hat{\theta_{j,0}}) + \log(1-\hat{\theta_{j,0}}) \tag{5.28}$$

and a bias term equal to

$$b = \log(\hat{\theta}_1) - \log(1-\hat{\theta}_1) + \sum_{j\in[m]} \log(1-\hat{\theta}_{j,1}) - \sum_{j\in[m]} \log(1-\hat{\theta}_{j,0}). \tag{5.29}$$

That is, the log-odds in Equation 5.27 can be rearranged so that

$$\log\left(\frac{\hat{P}(y=1|\mathbf{x}_*)}{\hat{P}(y=0\mid\mathbf{x}_*)}\right) = b + \sum_{j\in[m]} w_j\mathbf{x}[j], \tag{5.30}$$

with w_j and b defined as in Equations 5.28 and 5.29 above. In terms of a predictive model, we can thus express re-express the decision boundary from Equation 5.26 as follows:

$$f_\Theta(\mathbf{x}_*) = \begin{cases} 1 & \text{if } b + \sum_{j \in [m]} w_j \mathbf{x}[j] > \log(\gamma) \\ 0 & \text{if } b + \sum_{j \in [m]} w_j \mathbf{x}[j] < \log(\gamma) \end{cases}. \tag{5.31}$$

Note that we take the logarithm of the threshold parameter from Equation 5.26, which means that the default decision boundary of $\gamma = 1$ corresponds to $\log(\gamma) = \log(1) = 0$ in this formulation.

Thus, even though we derived our model based upon maximum likelihood, we can still interpret the learned model as a linear decision boundary with a natural threshold at 0, similar to the perceptron from Chapter 3. However, unlike the perceptron, the Naive Bayes model specifies a *unique* linear decision boundary, derived from the maximum likelihood principle.

5.4 Advanced Naive Bayes

So far, we have focused on a Naive Bayes model for binary features. This model is commonly known as the Bernoulli Naive Bayes model. However, what if we have non-binary (e.g., continuous) features? In such cases, we can generalize the Naive Bayes model to leverage other distributions, such as the Gaussian Naive Bayes model for continuous features.

Gaussian Naive Bayes

In the Gaussian Naive Bayes model, we assume that the features are generated by independent Gaussian distributions, i.e.,

$$p(\mathbf{x}[j] | y = k) = \frac{1}{\sqrt{2\pi}\sigma_{j,k}} e^{-\frac{(x - \mu_{j,k})^2}{2\sigma_{j,k}^2}}. \tag{5.32}$$

Note that as with the Bernoulli case, we have unique parameters for each feature conditioned on each class. However, in this case, we have two parameters—i.e., both the mean and the variance—so we end up with $4m + 1$ parameters overall, assuming that we have m continuous features. This means that the full parameter set is equal to $\Theta = \{\theta_k, \mu_{j,k}, \sigma_{j,k}, \forall j = 1...m, k = 0, 1\}$, and the full likelihood of the model is given by

$$\log \mathcal{L}(\Theta, \mathcal{D}_{\text{train}}) \propto \sum_{(\mathbf{x}_i, y_i) \in \mathcal{D}_{\text{train}}} \log P_\Theta(y_i) + \sum_{j \in [m]} \log P_\Theta(\mathbf{x}_i[j] \mid y_i) \tag{5.33}$$

$$= \sum_{(\mathbf{x}_i, y_i) \in \mathcal{D}_{\text{train}}} \log \left(\theta^{y_i} (1 - \theta)^{(1 - y_i)} \right) + \sum_{j \in [m]} \frac{1}{\sqrt{2\pi}\sigma_{j,y_i}} e^{-\frac{(\mathbf{x}_i[j] - \mu_{j,y_i})^2}{2\sigma_{j,y_i}^2}}. \tag{5.34}$$

As with the Bernoulli Naive Bayes, we can solve for the maximum likelihood estimate for each conditional feature likelihood independently. This maximum likelihood estimate follows directly from the derivations in the last chapter, and is given by

$$\hat{\mu}_{j,k} = \sum_{(\mathbf{x}_i, y_i) \in \mathcal{D}_{\text{train}} : y_i = k} \frac{\mathbf{x}_i[j]}{|(\mathbf{x}_i, y_i) \in \mathcal{D}_{\text{train}} : y_i = k|}. \tag{5.35}$$

In other words, the parameter $\mu_{j,k}$ is simply the average value of feature j for points in class k. The estimate for the (uncorrected) sample variance is analogous:

$$\hat{\sigma}_{j,k} = \sqrt{\sum_{(\mathbf{x}_i, y_i) \in \mathcal{D}_{\text{train}} : y_i = k} \frac{(\mathbf{x}_i[j] - \hat{\mu}_{j,k})^2}{|(\mathbf{x}_i, y_i) \in \mathcal{D}_{\text{train}} : y_i = k|}}. \tag{5.36}$$

Mixing and Generalizing the Feature Likelihoods

The Gaussian Naive Bayes allows one to model continuous features based on an assumption of independent Gaussians. In principle, we can define Naive Bayes models based upon a wide range of distributional assumptions. For example, we might assume that count-based features (e.g., word counts) have a multinomial distribution. The Naive Bayes approach can be adapted to work with a wide range of distributions for the feature likelihood, and—in fact—one could even mix different distributions within the same model (e.g., have some continuous Gaussian features and some binary Bernoulli features).

5.5 Beyond Binary Classification

The basic Naive Bayes model introduced above is only defined for binary classification. However, we can easily generalize this model the multiclass setting, where we have more than two classes. In particular, we can build a feature likelihood model $P(\mathbf{x}|y = k)$ independently for each class k, and we can generalize our class prior $P(y)$ to be a discrete distribution over k classes, rather than just a Bernoulli distribution. Thus, in the end, for k classes, we must estimate a model with $km + k - 1$ parameters: m parameters for each class-specific feature likelihood $P(\mathbf{x}|y = k)$ and $k - 1$ parameters for the class prior $P(y)$.[5]

Concretely, we will estimate $\theta_{j,k}$ parameters for each class k using the same formula as Equation 5.22. The only thing we need to change is that we must estimate k different class priors:

$$\hat{\theta}_k = \frac{|\{\mathbf{x}_i, y_i) \in \mathcal{D}_{\text{train}} : y_i = k\}|}{|\mathcal{D}_{\text{train}}|}. \tag{5.37}$$

[5]Note that we technically only need $k - 1$ parameters to define a distribution over k values because the probabilities must always sum to one.

We can then predict the log-likelihood of a point belonging to a particular class as

$$\log\left(P(y=k|\mathbf{x})\right) = \log(\hat{\theta}_k) + \sum_{j\in[m]} \mathbf{x}[j]\log(\hat{\theta}_{j,k}) + (1-\mathbf{x}[j])\log(1-\hat{\theta}_{j,k)}). \quad (5.38)$$

And, to make a prediction on a point $\mathbf{x}_*$, we can compute:

$$\arg\max_k \log\left(P(y=k|\mathbf{x}_*)\right). \quad (5.39)$$

This generalization of Naive Bayes to multiple classes recovers the binary setting as a special case, and it allows us to apply the Naive Bayes model to a much richer set of real-world problems.