# Note Set 6: Predictive Modeling: Classification

Padhraic Smyth,
Department of Computer Science
University of California, Irvine

March 2023

## 1 Classification Notation and Terminology

As with regression, in the standard classification problem we have a dataset $D = \{(\underline{x}_i, y_i)\}, i = 1, \ldots, N$ consisting of IID samples from some underlying unknown distribution $p(\underline{x}, y) = p(y|\underline{x})p(\underline{x})$. We are interested in learning a model for $p(y|\underline{x})$ that can predict a value or distribution for $y$ given a vector of input values $\underline{x}$. However, unlike regression, the $Y$ variable is now categorical (discrete), in general taking one of $K$ possible values or class labels, i.e., $y \in \{1, \ldots, K\}$. These $K$ values will typically be mapped to semantic categories that have a particular meaning, e.g., for an image classification problem $y = 1$ might correspond to the class "dog", $y = 2$ might be the class "cat", and so on: we will index the class labels as $k = 1, \ldots, K$ to maintain generality. The $\underline{x}$ input vector to our classification model will in general be $d$-dimensional with real or integer-valued components, where one of the components typically is the constant value 1 so that our model has an intercept term (as in regression modeling).

For classification applications we are typically interested in models that can produce estimates of class probabilities, since having an estimate of the conditional probability of a particular class $k$ given an input $\underline{x}$ is very useful in many practical applications. More specifically, let $p(y = k|\underline{x})$ be the unknown true conditional probability of label $k$ given input $\underline{x}$: this is what we want to model or learn. and let $\underline{f}(\underline{x}; \underline{\theta}) = (f_1(\underline{x}; \underline{\theta}), \ldots, f_K(\underline{x}; \underline{\theta}))$ be our model's estimate of the $K$-ary vector of true conditional class probabilities $p(y = k|\underline{x}), k = 1, \ldots, K$. This is similar to the multinomial model (discussed in earlier notes) but where now the $K$ conditional probabilities for $y$ are allowed to change as a function of $\underline{x}$. Because we are interpreting the $K$ outputs of our model as conditional probabilities, then we will need to impose constraints on the $f_k$ values, i.e., in particular we will need $\sum_k f_k(\underline{x}; \underline{\theta}) = 1$ and that $0 \leq f_k(\underline{x}; \underline{\theta}) \leq 1, k = 1, \ldots, K$ (we will describe a general way to enforce this constraint below).

Technically, we will only ever need to estimate $K - 1$ class probabilities, and so our prediction model could just have $K - 1$ outputs instead of $K$, i.e., $\underline{f}(\underline{x}; \underline{\theta}) = (f_1(\underline{x}; \underline{\theta}), \ldots, f_{K-1}(\underline{x}; \underline{\theta}))$ and $f_K(\underline{x}; \underline{\theta})$ can be found by computing $\sum_{k=1}^{K-1} f_k(\underline{x}; \underline{\theta})$. In practice, however, its common in machine learning, for $K > 2$, to ignore this constraint and build/train models with $K$ outputs rather than $K - 1$ (and this generally doesn't cause any problems). The exception in machine learning is the special case of $K = 2$, i.e., binary

classification. For $K = 2$ it is common in machine learning to just have a single output for the classification model, namely $f(\underline{x};\underline{\theta})$, which approximates the true $p(y = 1|\underline{x})$, with $p(y = 0|\underline{x})$ being approximated by $1 - f(\underline{x};\underline{\theta})$. (The $K = 2$ case is also different to $K > 2$ in terms of notation in that it is traditional to index the labels as $\{0, 1\}$ rather than $\{1, 2\}$ but this is not an important detail).

## 2 Logistic and Neural Classifiers

There are a broad range of classification models whose outputs can be written in the general form:

$$f_k(\underline{x};\underline{\theta}) \;=\; \frac{\exp(z_k(\underline{x};\underline{\theta}))}{\sum_{m=1}^{K}\exp(z_m(\underline{x};\underline{\theta}))} \quad k = 1,\ldots,K$$

where the $z_k(\underline{x};\underline{\theta})$'s are scalar-valued functions of the input that can lie anywhere on the real line, i.e., $z_k(\underline{x};\underline{\theta}) \in \mathcal{R}$. This formulation allows us to use any unconstrained modeling method we want in order to generate the $z_k(\underline{x};\underline{\theta})$'s, e.g., $z_k(\underline{x};\underline{\theta})$ could be a linear weighted sum that could take positive or negative value. The equation above is then just a convenient way to transform these $z$'s to be positive (numerator) and to ensure that they sum to 1 (denominator), i.e., to satisfy our constraint that the outputs of our model, the $f_k$'s are valid estimates of a conditional probability distribution. The functional form above is sometimes referred to as the "softmax" function in machine learning, and shows up in statistics as the functional form for multinomial logistic regression models.

Two of the most well-known models in machine learning of this general form are discussed below. Note that these 2 classes of models parallel the use of linear and neural models for regression (see NoteSet 5), but where now we have the additional aspect of generating $K$ outputs (conditional probability estimates) via the softmax equation above.

**1. Logistic (linear) Classifiers:** (also known as logistic regression). For a logistic classifier, the functions $z_k(\underline{x};\underline{\theta})$ are linear, i.e., $z_k(\underline{x};\underline{\theta}) = \underline{\beta}_k^T\underline{x}$, where $\underline{\beta}_k$ is a $d$-dimensional vector of weights, one per class, and the total set of parameters is $\underline{\theta} = (\underline{\beta}_1,\ldots,\underline{\beta}_k)$. This classifier is very simple: given an input $\underline{x}$, compute a weighted sum of the inputs, one weighted sum per class (using weights $\underline{\beta}_k$ for class $k$) and then convert the outputs to probabilities using the softmax operation above. Even though the output stage of the model (the softmax transformation) is non-linear, the logistic classifier is usually considered to be a linear classifier since it produces in linear decision boundaries between the classes (which we discuss later) and (equivalently) it is computed from linear functions of the inputs. However, its important to note that its functional form (the $f_k$'s) are not linear in the parameters (due to the softmax operation).

Note also that a direct extension of the logistic model is to augment the input $\underline{x}$ with additional predefined variables that are functions of $\underline{x}$ using $\underline{x}'$, e.g., $\underline{x}' = (1, x_1, x_1^2, x_2, x_2^2, \ldots)$, e.g., for low-dimensional problems where we want additional flexibility, i.e., non-linearity in the original $\underline{x}$ space (but still linear in the augmented $\underline{x}'$ space).

**2. Neural (nonlinear) Classifiers:** This is a very broad class of models, particularly in the area of deep learning, but for our purposes we will view them as being defined as follows: $z_k(\underline{x};\underline{\theta}) = \underline{\beta}_k^T \underline{g}(\underline{x};\phi)$ where the $z_k$'s are then transformed to be probabilities using the softmax operation above.

Here $\underline{g}(\underline{x};\phi)$ is an $h$-dimensional vector representing some hidden representation that the neural network has learned, that tranforms the $d$-dimensional input $\underline{x}$, using parameters $\phi$, into a set of $h$ real-valued numbers represented by the vector-valued $\underline{g}(\underline{x};\phi)$. For example if $\underline{x}$ corresponds to a set of pixels in an image (with a large value of $d$) then $\underline{g}(\underline{x};\phi)$ could be some much lower-dimensional representation of $\underline{x}$ that is (perhaps) translation and scale-invariant in terms of the classification problem[1].

The parameters of the overall neural network model can be defined as $\underline{\theta} = \{\underline{\beta}_1, \ldots, \underline{\beta}_K, \phi\}$, where the $\underline{\beta}$ vectors play the same role as the class-specific weight vectors in a logistic model, and the $\phi$ parameters are the feature extraction part of the model (transforming the original inputs $\underline{x}$ into a representation $\underline{g}$ that is useful for prediction).

## 3 Conditional Likelihood and Log-Loss/Cross-Entropy Loss

What loss function should we use to optimize the parameters $\underline{\theta}$ for a classifier model such as a logistic or neural network model? One way to approach this is to define a conditional likelihood for our problem. Our data consists of $D = (D_x, D_y) = \{(\underline{x}_i, y_i\}, 1 \leq i \leq N$, and we want to work with the conditional likelihood $p(D_y|D_x, \underline{\theta})$. Assuming IID data samples from $p(\underline{x}, y)$ we can define a multinomial log-likelihood of the form

$$L(\underline{\theta}) \;=\; \prod_{i=1}^{N} p(y_i|\underline{x}_i;\underline{\theta})$$

where $y_i \in \{1, \ldots, K\}$. The terms $p(y_i|\underline{x}_i, \underline{\theta})$ represent our model-based probabilities, namely the $f_k$ functions, parametrized by $\underline{\theta}$, that are estimates of the true (unknown) probabilities $p(y_i|\underline{x})$. In particular, if $y_i = k$, then we want to use $f_k(\underline{x}_i;\underline{\theta})$ as the corresponding probability from the model, i.e., if $y_i = k$, then according to our model $p(y_i|\underline{x}_i, \underline{\theta}) = f_k(\underline{x}_i;\underline{\theta})$. We can represent this notationally by the use of an indicator function $I(y_i, k)$ which takes value 1 if $y_i = k$ and value 0 otherwise. Thus, our likelihood becomes:

$$L(\underline{\theta}) \;=\; \prod_{i=1}^{N} p(y_i|\underline{x}_i;\underline{\theta}) \;=\; \prod_{i=1}^{N}\prod_{k=1}^{K} f_k(\underline{x}_i;\underline{\theta})^{I(y_i,k)}.$$

In effect the indicator function is selecting the appropriate output of the model to use in the likelihood for each example $i$: for each datapoint $i$, the output $k$ such that $y_i = k$ will have value $f_k(\underline{x}_i;\underline{\theta})^1$ and all the

[1]For readers familiar with neural networks, $\underline{g}(\underline{x};\phi)$ would typically be the last hidden layer in the neural model before the output. By characterizing the neural via $\underline{g}(\underline{x};\phi)$, we are hiding a huge amount of detail in terms of how these neural models are built, particularly in deep learning; for example in image classification and in language modeling (where the "class" $y$ is the identity of the next word in a sequence), $\phi$ could represent billions of parameters (weights) and the function $\underline{g}(\underline{x};\phi)$ could be very complex (e.g., containing transformer components, etc). But we are ignoring (on purpose) this level of detail here.

other $K-1$ terms will have value $f_k(\underline{x}_i;\underline{\theta})^0 = 1$ and be factored out. Another way to write this would be as

$$L(\underline{\theta}) \;=\; \prod_{k=1}^{K} \prod_{i:y_i=k}^{K} f_k(\underline{x}_i;\underline{\theta})$$

which creates $K$ separate products in the likelihood, one per class $k$. We can see that maximizing this likelihood $L(\underline{\theta})$ corresponds to having the model give as high a probability as possible (e.g., close to 1) for each true class label $y_i$ in the training data, where the prediction is generated as a function of the input $\underline{x}_i$.

We can rewrite in the form of a log-likelihood as

$$l(\underline{\theta}) \;=\; \sum_{i=1}^{N}\sum_{k=1}^{K} I(y_i,k)\log f_k(\underline{x}_i;\underline{\theta})$$

and if we take the negative of this and divide by $N$ we get

$$-\frac{1}{N}l(\underline{\theta}) \;=\; \frac{1}{N}\sum_{i=1}^{N}\sum_{k=1}^{K} I(y_i,k)\log\frac{1}{f_k(\underline{x}_i;\underline{\theta})}$$

which is the empirical risk function used widely for classification problems in machine learning. Thus, minimizing this function (as a function of $\underline{\theta}$) is equivalent to maximizing the conditional log-likelihood, i.e., we have $R(\underline{\theta}) = \frac{1}{N}\sum_{i=1}^{N}\Delta(y_i, f_k(\underline{x}_i;\underline{\theta}))$ where $\Delta(y_i, f_k(\underline{x}_i;\underline{\theta})) = \sum_k I(y_i,k)\log\frac{1}{f_k(\underline{x}_i;\underline{\theta})}$. This loss is often referred to as the log-loss or cross-entropy loss in machine learning. For convenience we will define

$$CE(\underline{\theta}) \;=\; \frac{1}{N}\sum_{i=1}^{N}\sum_{k=1}^{K} I(y_i,k)\log\frac{1}{f_k(\underline{x}_i;\underline{\theta})}$$

i.e., the empirical risk using the cross-entropy (CE) loss.

For example, for $K = 2$ (the special case of binary classification, with $y \in \{0,1\}$, we can write the empirical risk as

$$CE(\underline{\theta}) \;=\; \frac{1}{N}\sum_{i=1}^{N}\left(y_i\log\frac{1}{f(\underline{x}_i;\underline{\theta})} + (1-y_i)\log\frac{1}{1-f(\underline{x}_i;\underline{\theta})}\right)$$

where $f$ is the single output of the model and the binary $y_i$'s select the appropriate loss term to use within the sum for each datapoint.

So, what we have shown above is that finding the parameters that minimize the well-known log-loss/cross-entropy loss function (used to train many machine learning classifiers, e.g., in image classification, in language modeling, etc), corresponds directly to finding the parameters that maximize the log-likelihood of a multinomial likelihood model with an IID assumption. And long as our model $\underline{f}(\underline{x}_i;\underline{\theta})$ is differentiable as a function of $\underline{\theta}$, we can define gradients with respect to the parameters (the components of $\underline{\theta}$) and use any of a wide variety of gradient-based methods to maximize $l(\underline{\theta})$ (or equivalently minimize $CE(\underline{\theta})$).

Without any regularization in $CE(\underline{\theta})$ (or any priors in our likelihood-based setup) this optimization is a maximum likelihood estimation procedure. With priors, as with regression, the log of the prior will show up

as an additional regularization term (e.g., proportional to $\sum_{j=1} \theta_j^2$ for Gaussian priors or L2 regularization) in addition to $CE(\underline{\theta})$ and minimization of $CE(\underline{\theta})+\lambda r(\underline{\theta})$, where $\lambda$ is the relative weight of the regularization term and $r(\underline{\theta})$ is the regularization function itself (e.g., $r(\underline{\theta}) = \sum_{j=1} \theta_j^2$) is equivalent to maximizing the product of the likelihood times the prior, i.e., performing MAP estimation.

And as with regression, we can also be fully Bayesian, averaging over parameter uncertainty, to generate a predictive distribution $p(y|\underline{x}, D_y, D_x)$ to make predictions for any future $\underline{x}$. Unfortunately, as with regression, doing this computation exactly is impossible for most models of interest (even simple models such as logistic classifiers), and approximate methods such as Monte Carlo sampling methods or deterministic approximations (Laplace, variational) must be relied on if one wants to be fully Bayesian.

# 4 Discriminant Functions and Decision Regions

**Note on Notation:** The notation in this section below is a little different from the notation in the earlier sections, e.g., discriminants $g_k(\underline{x})$ can in principle be any function of $\underline{x}$: they could be linear, could be non-linear, could be transformed (or not) to sum to 1 and lie between 0 and 1, and so on. So $g_k(\underline{x})$ is intended to be very general below.

In this section we take a look at classifiers in terms of how they implement mappings from inputs $\underline{x}$ to "hard decisions" $\hat{y} \in \{1, \ldots, K\}$, rather than necessarily mapping to $K$ conditional probabilities. Classifiers like the logistic model or neural networks can make hard decisions by selecting a specific label given an input $\underline{x}$, e.g., $\hat{y}_{\underline{x}} = \arg\max f_k(\underline{x}; \underline{\theta})$, i.e., the most likely class. Its easy to see that the softmax operation doesn't change the identity of the most likely class, i.e., that $\hat{y}_{\underline{x}} = \arg\max_k f_k(\underline{x}; \underline{\theta}) = \arg\max_k z_k(\underline{x}; \underline{\theta})$ , where $f_k()$ is the softmax transformation of $z_k()$.

We can define a general form for classifiers by using the notion of **discriminant functions**, defined as follows:

- For each class $k \in \{1, \ldots, K\}$ we have a discriminant function $g_k(\underline{x})$ that produces a real-valued scalar discriminant value for each class $k$, conditioned on an input $\underline{x}$. Each discriminant function can be parametrized by parameters $\underline{\theta}_k$ (the dependence on $\underline{\theta}_k$ is suppressed below for simplicity).
- The classifier makes a decision on any input $\underline{x}$ by computing the $K$ discriminant values and assigning $\underline{x}$ to the class with the largest value, i.e., $\hat{y}_{\underline{x}} = \arg\max_k g_k(\underline{x})$

Discriminant functions provide a very general way to think about classifiers, including both probabilistic and non-probabilistic approaches (e.g., tree-based classifiers).

Some examples of discriminant functions are:

- **Linear discriminants** where $g_k(\underline{x}; \underline{\theta}) = \underline{\theta}_k^T \underline{x}$ where $\underline{\theta}_k$ is a $d$-dimensional vector of parameters;
- **Polynomial discriminants** where $g_k(\underline{x}; \underline{\theta}_k)$ is an $m$th order polynomial function of $\underline{x}$ with polynomial coefficients defined by $\underline{\theta}_k$ (e.g., $m = 2$ for a quadratic).

- **Non-linear discriminants** such as $g_k(\underline{x}) = h(\underline{\theta}_k^T \underline{x})$ where $h$ is a complex non-linear function such as defined by a neural network.

## Decision Regions and Decision Boundaries

We define the decision region $\mathcal{R}_k$ for class $k$, $1 \leq k \leq K$, as the region of the ($d$-dimensional, real-valued) input space $\underline{x}$ where the discriminant function for class $k$ is larger than any of the other discriminant functions, i.e.,

$$\underline{x} \in \mathcal{R}_k \Leftrightarrow \hat{y}_x = k \Leftrightarrow k = \arg\max_j g_j(\underline{x}), \quad j = 1, \ldots, K$$

So, decision region $\mathcal{R}_k$ is the region in the input space where $\underline{x}$ is classified as being in class $k$ by the classifier, or equivalently, $g_k(\underline{x}) > g_j(\underline{x}), \ \forall j \neq k$. Note that a decision region need not be a single contiguous region in $\underline{x}$ but could be the union of multiple disjoint subregions (depending on how the discriminant functions are defined).

Decision boundaries between decision regions are defined by equality of the respective discriminant functions. Consider the two-class case ($K = 2$) for simplicity, with $g_1(\underline{x})$ and $g_2(\underline{x})$. Points in the input space for which $g_1(\underline{x}) = g_2(\underline{x})$ are by definition on the decision boundary between the two classes. This is equivalent to saying that the equation for the decision boundary is defined by $g_1(\underline{x}) - g_2(\underline{x}) = 0$. In fact for the two-class case it is clear that we don't need two discriminant functions, i.e., we can just define a single discriminant function $g(\underline{x}) = g_1(\underline{x}) - g_2(\underline{x})$ and predict class 1 if $g(\underline{x}) > 0$ and class 2 if $g(\underline{x}) < 0$. And if $g(\underline{x}) = 0$ we would randomly select class 1 or 2.

As mentioned earlier, *linear discriminants* are defined as $g_k(\underline{x}) = \underline{\theta}_k^T \underline{x}$, i.e., an inner product of a weight vector and the input vector $\underline{x}$, where $\underline{\theta}_k$ is a $d$-dimensional weight vector for class $k$. In particular, for the two-class case, the decision boundary equation is defined as $g_1(\underline{x}) - g_2(\underline{x}) = \underline{\theta}_1^T \underline{x} - \underline{\theta}_2^T \underline{x} = \underline{\theta}^T \underline{x} = 0$, i.e., we only need a single weight vector $\underline{\theta}$. The equation $\underline{\theta}^T \underline{x} = 0$ will in general define the equation of a $(d-1)$-dimensional hyperplane in the $d$-dimensional $\underline{x}$ space, partitioning the input space into two contiguous decision regions separated by the linear hyperplane. More generally, for $K > 2$, linear discriminant functions will lead to piecewise linear decision regions in the input space $\underline{x}$.

We also note that if our discriminant is defined as $g_k(\underline{x}) = h(\underline{\theta}_k^T \underline{x})$, where $h()$ is some monotonic function, then we have in effect in a linear discriminant, i.e., we have linear decision boundaries in the input space $\underline{x}$, since the maximization operation $\hat{y}_{\underline{x}} = \arg\max_k g_k(\underline{x})$ to select the predicted class is unchanged whether we use $h(\underline{\theta}_k^T \underline{x})$ or $\underline{\theta}_k^T \underline{x}$ as our definition for $g_k(\underline{x})$. An example is the logistic classifier where the non-linear function $h()$ is the logistic function and where the decision boundaries in the input space $\underline{x}$ are linear for $K = 2$ and piecewise linear for $K > 2$.

More generally, if the $g_k(\underline{x})$ are polynomial functions of order $r$ in $\underline{x}$, the decision boundaries in the general case will also be polynomials of order $r$. An example of a discriminant function that produces quadratic boundaries ($r = 2$) in the general case is the multivariate Gaussian classifier (which we discuss later). And if the $g_k(\underline{x})$ are non-linear functions of $\underline{x}$ then in general we will get non-linear decision boundaries (e.g., for neural networks with one or more hidden layers).

# 5 Optimal Discriminant Functions

*Optimal discriminant functions* are discriminant functions that minimize the classification error rate of a classifier on average with respect to some underlying $p(\underline{x}, y)$. Below we will assume that all misclassification errors incur equal cost[2], known as the 0-1 cost function or 0-1 loss. It is not hard to see, that for 0-1 loss that the optimal discriminant is

$$g_k(\underline{x}) \; = \; p(y = k|\underline{x}), \quad 1 \leq k \leq K$$

which is equivalent to maximizing the posterior probability of the discrete-valued class variable $Y$ given $\underline{x}$, i.e., picking the most likely class for each $\underline{x}$.

From the definition of the optimal discriminant it is easy to see that another version of the optimal discriminant (in the sense that it will lead to the same decision for any $\underline{x}$) is defined by $g_k(\underline{x}) \; = \; p(\underline{x}|y = k)p(y \; = \; k)$ (because of Bayes rule). Again, this is theoretically optimal if we know it, but in practice we usually do not. And similarly, any monotonic function of these discriminants, such as $\log p(\underline{x}|y \; = \; k) + \log p(y = k)$ are also optimal discriminants.

Note that these are optimal predictions *in theory*, i.e., if we know the true $p(y = k|\underline{x})$ (or some monotonic function of it) exactly. In practice will usually need to approximate this conditional distribution by assuming some functional form for it (e.g., the logistic form) and learning the parameters for this functional form from data. The assumption of a specific functional form may lead to *bias* (or approximation error) in our estimate of the optimal discriminant function and learning parameters from a data set will lead to *variance* (or estimation error).

# 6 The Bayes Error Rate

As mentioned above, the optimal discriminant for any classification problem is defined by $g_k(\underline{x}) \; = \; p(y = k|\underline{x})$ (or some monotonic function of this quantity). Even though in practice we won't know the precise functional form or the parameters of $p(y \; = \; k|\underline{x})$ it is nonetheless useful to look at the error rate that we would get with the optimal classifier. The error rate of the optimal classifier is known as the **Bayes error rate** and provide a lower bound on the performance of any actual classifier (analogous to the unexplainable variance $\sigma_y^2$ in regression problems). As we will see below, the Bayes error rate depends on how much overlap there is between the density functions for each class, $p(\underline{x}|y = k)$, in the input space $\underline{x}$: if there is a lot of overlap we will get a high Bayes error rate, and with little or no overlap we get a low (near zero) Bayes error rate. For example, for high Bayes error, think of a 2-dimensional $\underline{x}$ space, with $K$ two-dimensional Gaussians in this space that are heavily overlapped; and if we could "pull" these Gaussiansn further and further apart to reduce overlap, then the Bayes error rate will decrease.

Consider the error rate of the optimal classifier at some particular point $\underline{x}$ in the input space. The probability of error is

$$e_{\underline{x}} = 1 - p(y = k^*|\underline{x}) = 1 - \max_k \{p(y = k|\underline{x}).$$

[2]More generally we can minimize expected cost where different errors may have different costs, but we will not pursue that here

Since our optimal classifier will always select $k^*$ given $\underline{x}$, the classifier will be correct $p(y = k^*|\underline{x})$ fraction of the time and incorrect $1-p(y = k^*|\underline{x})$ fraction of the time, at $\underline{x}$. (We are conveniently ignoring any points $\underline{x}$ here that might fall exactly on a decision boundary, and aslo assuming that our classifiers are deterministic as long as an input $\underline{x}$ is not on a decision boundary).

To compute the overall error rate (i.e., the probability that the optimal classifier will make an error on a random $\underline{x}$ drawn from $p(\underline{x})$) we need to compute the expected value of the error rate with respect to $p(\underline{x})$:

$$\begin{aligned} P_e^* &= E_{p(\underline{x})}[e_{\underline{x}}] \\ &= \int_{\underline{x}} e_{\underline{x}}\, p(\underline{x}) d\underline{x} \\ &= \int_{\underline{x}} \Big(1 - \max_k \{p(y = k|\underline{x})\}\Big) p(\underline{x}) d\underline{x} \end{aligned}$$

This term $P_e^*$ is known as the **Bayes error rate**. It is the optimal (lowest possible) error rate for any classifier with for some fixed feature space $\underline{x}$ with respect to $p(\underline{x}, y)$.

Rearranging, we can rewrite $P_e^*$ in terms of decision regions as

$$P_e^* = \sum_{k=1}^{K} \Big( \int_{\mathcal{R}_k} \big(1 - p(y = k|\underline{x})\big) p(\underline{x}) d\underline{x} \Big)$$

i.e., as the sum of $K$ different error terms defined over the $K$ decision regions. These error terms, one per class $k$, are proportional to how much overlap class $k$ has with each of the other class densities.

In the two-class case we can further simplify this to

$$\begin{aligned} P_e^* &= \int_{\mathcal{R}_1} \Big(1 - p(y = 1|\underline{x})\Big) p(\underline{x}) d\underline{x} + \int_{\mathcal{R}_2} \Big(1 - p(y = 2|\underline{x})\Big) p(\underline{x}) d\underline{x} \\ &= \int_{\mathcal{R}_1} p(y = 2|\underline{x}) p(\underline{x}) d\underline{x} + \int_{\mathcal{R}_2} p(y = 1|\underline{x}) p(\underline{x}) d\underline{x} \end{aligned}$$

Although we can only ever compute this for toy problems where we assume full knowledge of $p(y|\underline{x})$ and $p(\underline{x})$, the concept of the Bayes error rate is nonetheless useful and it can be informative to see how it depends on density overlap.

Note that any achievable classifier can never do better than the Bayes error rate in terms of its accuracy: the only way to improve on the Bayes error rate is to change the input feature vector $\underline{x}$, e.g., to add one or more features to the input space that can potentially separate the class densities more. So in principle it would seem as if it should always be a good idea to include as many features as we can in a classification problem, since the Bayes error rate of a higher dimensional space is always at least as low (if not lower) than any subset of dimensions of that space. This is true in theory, in terms of the optimal error rate in the higher-dimensional space: but in practice, when learning from a finite amount of data $N$, adding more

features (more dimensions) means we need to learn more parameters, so our actual classifier in the higher-dimensional space could in fact be less accurate (due to estimation noise) than a lower-dimensional classifier, even though the higher-dimensional space might have a lower Bayes error rate.

In general, the actual error rate of a classifier can be thought of as having components similar to those for regression. In regression, for a real-valued $y$ and the squared error loss function, the error rate of a prediction model can be decomposed into a sum of the inherent variability of $y$ given $\underline{x}$, plus a bias term, plus a variance term. For classification problems the decomposition does not have this simple additive form, but it is nonetheless useful to think of the actual error rate of a classifier as having components that come from (1) the Bayes error rate, (2) the bias (approximation error) of the classifier, and (3) variance (estimation error due to fitting the model to a finite amount of data).

# 7 Generative (Joint) Classification Models

The definitions above of optimal discriminants suggest two different strategies for learning classifiers from data. In the first we try to learn $p(y|\underline{x})$ directly: this is referred to as the *conditional or discriminative approach* and examples include logistic regression and neural networks that we discussed earlier.

The second approach, which we discuss in this section, is where we learn a model the input data $p(\underline{x}|y = k)$ for each class $k$ , and then use Bayes rule to make predictions via $p(y = k|\underline{x})$. This is sometimes referred to as the *generative or joint approach* since we are in effect modeling the joint distribution $p(\underline{x}, y)$, rather than just the conditional $p(y|\underline{x})$, allowing us in principle to generate or simulate data from the model (well-known examples are Gaussian classifiers and naive Bayes classifiers).

Generative models have the drawback that modeling $p(\underline{x}|y)$ can be difficult to do accurately, particularly as the dimensionality $d$ increases, whereas modeling the conditional distribution $p(y|\underline{x})$ can be much easier. Thus, joint or generative classifiers are less widely-used in practice than their conditional counterparts.

The generative approach can have some advantages, however, compared to the conditional approach, particularly if we know something about the distribution of the data in the input space $\underline{x}$. In particular, generative models can be useful for dealing with missing data in the input space, for semi-supervised learning, or for detecting outliers or distribution shifts or data from novel classes in the input space.

**Gaussian Classifiers** A classical approach to generative classifiers for multivariate $d$-dimensional data $\underline{x}$ is to assume that the conditional densities for each class have a multivariate Gaussian distribution, i.e.,

$$p(\underline{x}|y = k) \;=\; N(\underline{\mu}_k, \Sigma_k) \quad 1 \leq k \leq K$$

where $\underline{\mu}_k$ is a $d$-dimensional mean and $\Sigma_k$ is a $d \times d$ covariance matrix, and $\underline{\mu}$ and $\Sigma_k$ can be different for each class. In the general case we have discriminant functions that can be written in the form

$$\begin{aligned} g_k(\underline{x}) &= \log p(\underline{x}|y = k) + \log p(y = k) \\ &\propto (\underline{\mu}_k - \underline{x})^T \Sigma_k^{-1} (\underline{\mu}_k - \underline{x}) + C_k \end{aligned} \tag{1}$$

where $C_k$ involves terms that depend on $k$ (such as $\det(\Sigma_k)$ and $p(y = k)$) but that don't depend on $\underline{x}$. The first term is a quadratic in $\underline{x}$. Thus, the discriminant functions for Gaussian generative classifiers are (in general) **quadratic functions**[3], and consequently the decision boundaries are also quadratic in form. For example, for $K = 2$ we can solve for the $\underline{x}$ values that satisfy $g_1(\underline{x}) = g_2(\underline{x})$ to find the quadratic function in $\underline{x}$ that defines the decision boundaries.

The main weaknesses of Gaussian classifiers are that (i) they require us to make a strong assumption about the parametric form of the distributions of $\underline{x}$ in the input space, and (ii) they require $O(d^2)$ parameters per class, which can be problematic for problems where $d$ is large (e.g., if $d$ is the number of pixels in an image). To address the $O(d^2)$ problem one approach is to approximate the full covariance matrix for each class with a diagonal matrix (all covariances except the diagonals set to 0): this is obviously a big approximation but may be good enough in some cases for discriminating between classes.

**Markov Sequence Classifiers** Generative classifiers can often be easy to extend to non-vector data. For example, consider data consisting of multiple sequences $\underline{x}_i, 1 \leq i \leq N$, where each $\underline{x}_i$ is a categorical sequence and the sequences can have different lengths. Examples might be protein sequences in bioinformatics or sequences of visits to Web pages by different visitors to a Website. Consider also that the sequences are classified with labels $y_i$, e.g., different types of proteins or visitors who make purchases on a Website versus those that don't.

We can in principle build a generative model for each class, i.e., $p(\underline{x}|y = k)$, such as a Markov model, with discriminant functions $\log p(\underline{x}|y = k) + \log p(y = k), 1 \leq k \leq K$. Given a sequence $\underline{x}$ of any length, its probability $p(\underline{x}|y = k)$ can be computed for each class $k$, via the factorized representation implicit in the Markov chain. This general approach was the basis of speech recognition systems for many years, where each class $k$ corresponded to a word in the vocabulary (and $K$ would be quite large, e.g., $K = 50,000$), and the underlying generative models per class (or word) were hidden Markov models.

**Parameter Estimation for Generative Models** Estimating the parameters of a generative model is usually quite straightforward. Since the data are assumed to be all labeled, we can separate the likelihood (or log-likelihood) into separate products (or sums), one for each class and for the parameters of that class, i.e., $\log L(\underline{\theta}) = \sum_{k=1}^{K} \left( \sum_{y_i:y_i=k} \log p(\underline{x}_i|y_i = k, \underline{\theta}_k) p(y_i = k) \right)$. This is for the case where the parameters for each class are independent of the parameters in other classes: if the parameters are tied or linked in some way then we follow a different path, one which is also usually quite straightforward.

We can then partition our training data into $K$ subsets according to the $K$ class labels, and then optimize (separately) the log-likelihood term for each class, i.e., $\sum_{y_i:y_i=k} \log p(\underline{x}_i|y_i = k, \underline{\theta}_k) p(y_i = k)$ to find the $\underline{\theta}_k$'s. We can do this parameter estimation per class using any of our favorite methods from density estimation, e.g., maximum likelihood, maximum a posteriori, etc. We can even be fully Bayesian if we wish to, by inferring posterior distributions for $\underline{\theta}_k$ for each class and then averaging over these posterior distributions when making predictions about the class labels $y$ for a new data point $\underline{x}$.

[3]There are some exceptions, such as when each class has a common covariance, $\Sigma_k = \Sigma$, and the quadratic terms drop out resulting in linear discriminants.