1 Discriminant Functions and Decision Regions

Consider a classification problem with a $d$-dimensional input vector $\mathbf{x}$ and a class variable $C$ taking values $\{1, \ldots, K\}$. For simplicity you can consider the components of $\mathbf{x}$ to all be real-valued. We will also assume for convenience that $\mathbf{x}$ is defined in a way such that one of its input is always set to 1 (or some constant), to ensure that we have an intercept term in our models.

We can define a general form for a classifier as a collection of discriminant functions as follows:

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

Some examples of discriminant functions are:

- **Linear discriminants** where $g_k(\mathbf{x}; \theta) = \theta_k^T \mathbf{x}$ where $\theta_k$ is a $d$-dimensional vector of parameters;
- **Polynomial discriminants** where $g_k(\mathbf{x}; \theta_k)$ is an $m$th order polynomial of $\mathbf{x}$ with polynomial coefficients defined by $\theta_k$ (e.g., $m = 2$ for a quadratic).
- **Non-linear discriminants** such as $g_k(\mathbf{x}) = h(\theta_k^T \mathbf{x})$ where $h$ is a non-linear function (e.g., the logistic function); or more complex non-linear forms for $g_k(\mathbf{x})$ such as defined by neural networks.
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 input space $\mathbf{x}$ where the discriminant function for class $k$ is larger than any of the other discriminant functions, i.e.,

$$\mathbf{x} \in \mathcal{R}_k \iff \hat{y}_x = k \iff k = \arg \max g_j(\mathbf{x}), \; j = 1, \ldots, K$$

So, decision region $\mathcal{R}_k$ is the region in the input space where $\mathbf{x}$ is classified as being in class $k$ by the classifier, or equivalently, $g_k(\mathbf{x}) > g_j(\mathbf{x})$, $\forall j \neq k$. Note that a decision region need not be a single contiguous region in $\mathbf{x}$ 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 $g_1(\mathbf{x})$ and $g_2(\mathbf{x})$. Points in the input space for which $g_1(\mathbf{x}) = g_2(\mathbf{x})$ 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 $g_1(\mathbf{x}) - g_2(\mathbf{x}) = 0$. 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 $g(\mathbf{x}) = g_1(\mathbf{x}) - g_2(\mathbf{x})$ and predict class 1 if $g(\mathbf{x}) > 0$ and class 2 if $g(\mathbf{x}) < 0$. And if $g(\mathbf{x}) = 0$ we would randomly select class 1 or 2.

As mentioned earlier, linear discriminants are defined as $g_k(\mathbf{x}) = \theta_k^T \mathbf{x}$, i.e., an inner product of a weight vector and the input vector $\mathbf{x}$, where $\theta_k$ is a $d$-dimensional weight vector for class $k$. In particular, for the two-class case, the decision boundary equation is defined as $g_1(\mathbf{x}) - g_2(\mathbf{x}) = \theta_1^T \mathbf{x} - \theta_2^T \mathbf{x} = \theta^T \mathbf{x} = 0$, i.e., we only need a single weight vector $\theta$. The equation $\theta^T \mathbf{x} = 0$ will in general define the equation of a $(d-1)$-dimensional hyperplane in the $d$-dimensional $\mathbf{x}$ 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 $\mathbf{x}$.

We also note that if our discriminant is defined as $g_k(\mathbf{x}) = h(\theta_k^T \mathbf{x})$, where $h()$ can be any monotonic function, then we have in effect in a linear discriminant, i.e., we have linear decision boundaries in the input space $\mathbf{x}$, since the maximization operation $\hat{y}_x = \arg \max_k g_k(\mathbf{x})$ to select the predicted class is unchanged whether we use $h(\theta_k^T \mathbf{x})$ or $\theta_k^T \mathbf{x}$ as our definition for $g_k(\mathbf{x})$. 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 $\mathbf{x}$ are linear for $K = 2$ and piecewise linear for $K > 2$.

More generally, if the $g_k(\mathbf{x})$ are polynomial functions of order $r$ in $\mathbf{x}$, 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. And if the $g_k(\mathbf{x})$ are non-linear functions of $\mathbf{x}$ then in general we will get non-linear decision boundaries (e.g., for neural networks with one or more hidden layers).
Optimal Discriminants

Optimal discriminants are discriminant functions that minimize the error rate of a classifier on average with respect to some underlying $p(x, y)$. Here we will assume that all misclassification errors incur equal cost, 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

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

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

Note that these are optimal predictions in theory, i.e., if we know the true $p(y = k|x)$ exactly. In practice we 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) and learning parameters from a data set will lead to variance (or estimation error).

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 $x$) is defined by

$$g_k(x) = p(x|y = k)p(y = k)$$

(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 $\log p(x|y = k) + \log p(y = k)$ are also optimal discriminants.

The Bayes Error Rate

As mentioned above, the optimal discriminant for any classification problem is defined by $g_k(x) = p(y = k|x)$ (or some function of this quantity). Even though in practice we won’t know the precise functional form or the parameters of $p(y = k|x)$ 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. As we will see below, the Bayes error rate depends on how much overlap there is between the density functions for each class, $p(x|y = k)$, in the input space $x$: 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.

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

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

1 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 $x$, the classifier will be correct $p(y = k^* | x)$ fraction of the time and incorrect $1 - p(y = k^* | x)$ fraction of the time, at $x$.

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

$$P_e^* = E_{p(x)}[e_x] = \int_x e_x p(x) dx = \int_x \left(1 - \max_k \{ p(y = k | x) \} \right) p(x) dx$$

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 $x$ with respect to $p(x, y)$.

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

$$P_e^* = \sum_{k=1}^K \left( \int_{R_k} (1 - p(y = k | x)) p(x) dx \right)$$

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

$$P_e^* = \int_{R_1} \left(1 - p(y = 1 | x) \right) p(x) dx + \int_{R_2} \left(1 - p(y = 2 | x) \right) p(x) dx$$

$$= \int_{R_1} p(y = 2 | x) p(x) dx + \int_{R_2} p(y = 1 | x) p(x) dx$$

Although we can only ever compute this for toy problems where we assume full knowledge of $p(y|x)$ and $p(x)$, 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 $x$, 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 $x$, 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).

## 2 Discriminative versus Generative Models

The definitions above of optimal discriminants suggest two different strategies for learning classifiers from data. In the first we try to learn $p(y|x)$ directly: this is referred to as the conditional or discriminative approach and examples include logistic regression and neural networks. The second approach, where we learn $p(x|y)$ and $p(y)$, is often referred to as the generative approach since we are in effect modeling the joint distribution $p(x,y)$, rather than just the conditional $p(y|x)$, allowing us in principle to generate or simulate data (well-known examples are Gaussian and naive Bayes classifiers).

The conditional/discriminative approach tends to be much more widely used in practice than the generative approach. The primary reason for this is that modeling $p(x|y)$ for high-dimensional $x$ is usually very difficult to do accurately, whereas modeling the conditional distribution $p(y|x)$ can be much easier.

The generative model has some advantages, however, compared to the conditional/discriminative model, particularly in problems where it’s useful to know something about the distribution of the data in the input space $x$, such as dealing with missing data, semi-supervised learning, or detecting outliers and distributional shifts in the input space.

### 2.1 Generative Models

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

$$p(x|y = k) = N(\mu_k, \Sigma_k) \quad 1 \leq k \leq K$$
where $\mu_k$ is a $d$-dimensional mean and $\Sigma_k$ is a $d \times d$ covariance matrix, and $\mu$ and $\Sigma_k$ can be different for each class. In the general case we have discriminant functions that can be written in the form

$$g_k(x) = \log p(x|y = k) + \log p(y = k) \propto (\mu_k - x)^T \Sigma_k^{-1} (\mu_k - x) + C_k$$

where $C_k$ involves terms that depend on $k$ (such as $\det(\Sigma_k)$ and $p(y = k)$ but that don’t depend on $x$. The first term is a quadratic in $x$. Thus, the discriminant functions for Gaussian generative classifiers are (in general) quadratic functions and consequently the decision boundaries are also quadratic in form. For example, for $K = 2$ we solve for the $x$ values that satisfy $g_1(x) = g_2(x)$ to find the quadratic in $x$ 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 $x$ 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** One feature of generative classifiers is that they can easily handle input data $x$ that is not fixed-dimensional (and discriminative models can’t do this). For example, consider data consisting of multiple sequences $x_i$, $1 \leq i \leq N$, where each $x_i$ 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., $p(x|y = k)$, such as a Markov model, with discriminant functions $\log p(x|y = k) + \log p(y = k)$, $1 \leq k \leq K$. Given a sequence $x$ of any length, its probability $p(x|y = k)$ 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.,

---

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.
log $L(\theta) = \sum_{k=1}^{K} \left( \sum_{y_i : y_i = k} \log p(x_i | y_i = k, \theta_k) p(y_i = k | \theta_k) \right)$. 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., $\log p(x_i | y_i = k, \theta_k) p(y_i = k | \theta_k)$ to find the $\theta_k$’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 $\theta_k$ for each class and then averaging over these posterior distributions when making predictions about the class labels $y$ for a new data point $x$.

### 2.2 Conditional/Discriminative Models

**Logistic and Neural Models** In a conditional (or discriminative) model we dispense entirely with the idea of modeling the data in the input space $x$ and instead directly model the conditional distribution $p(y|x)$ directly. Well-known examples of this approach are logistic models and neural networks. We can think of both as having two components:

- First compute scalar-valued discriminant functions $g_k(x; \theta_k) = \beta_k^T z(x; \phi_k), k = 1, \ldots, K$, where (as in regression) $z(x; \phi_k)$ is a vector valued function of the inputs $x$ parametrized by $\phi_k$, and $\beta$ is a vector of parameters of the same dimensionality as $z$, and $\theta_k = \{\beta_k, \phi_k\}$. For a logistic classifier the function $z$ is just an identity function, i.e., $z(x) = x$ and the set of $\beta_k$’s, $1 \leq k \leq K$, consist of $K d$-dimensional weight vectors. For a neural network $z$ can in general be a very complicated non-linear function of the input $x$ and $\phi$ are the weights of the neural network, with an additional weight matrix of dimension $K \times d$ of $\beta_k$ weights per class, that converts from the dimensionality of $z$ to $K$, yielding $K$ scalar valued discriminants, the $g_k(x; \theta_k)$. In neural network parlance these discriminants $g_k()$ are often referred to as “logits.”

- In the second step the discriminants $g_k(x; \theta_k)$ are transformed into numbers that lie between 0 and 1 so that they correspond to estimates of probabilities. The usual manner in which this is done is via the logistic function (also known as “softmax” function in neural networks):

$$p(y = k | x) = \frac{e^{g_k(x; \theta_k)}}{\sum_{j=1}^{K} e^{g_j(x; \theta_j)}}$$

**Conditional Likelihood and the Log-Loss Function for Classification** What loss function should we use to optimize the parameters $\theta$ for a conditional model such as a logistic or neural network model? One

---

3 In practice we only actually need $K - 1$ vectors of weights given the constraint that the probabilities for $y$ must sum to 1.
way to approach this is to defined a conditional likelihood for our problem. Our data consists of $D = (D_x, D_y) = \{(x_i, y_i)\}_{1 \leq i \leq N}$, and we want to work with the conditional likelihood $p(D_y|D_x, \theta)$. Assuming IID data samples from $p(x, y)$ we can define a multinomial log-likelihood of the form

$$L(\theta) = \prod_{i=1}^{N} p(y_i|x_i, \theta)$$

where $y_i \in \{1, \ldots, K\}$. We can rewrite this as a log-likelihood in the form

$$l(\theta) = \sum_{i=1}^{N} \log p(y_i|x_i, \theta) = -N CE(\theta)$$

where $CE(\theta) = -\frac{1}{N} \sum_{i=1}^{N} \log p(y_i|x_i, \theta)$ is the cross-entropy loss (or log-loss) that is widely used in training conditional models, i.e., these loss functions are in fact log-likelihoods for a simple IID model. As long as our model for $p(y_i|x_i, \theta)$ is differentiable we can define gradients with respect to the parameters (the components of $\theta$) and use any of a wide variety of gradient-based methods to optimize $L(\theta)$ (or $CE(\theta)$).

Without any regularization in $CE(\theta)$ (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 $\sum_{j=1}^{J} \theta_j^2$ for Gaussian priors or L2 regularization) in addition to $CE(\theta)$ and minimization of $CE(\theta) + \text{regularization}$ 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 $p(y|x, D_y, D_x)$ to make predictions for any future $x$. 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.