Tutorial Linear Classification & Kernels

Suggested reading for linear classification:

Murphy: 4.2 (examples for using gaussians as class conditional densities) (optional), 8.1, 8.2, 8.3, 8.6 until 8.6.1 (incl.)
and Barber 17.4 (part on the perceptron).
as always very elegant and concise, contains a lot of links and further reading (e.g. on bayesian logistic regression etc.)

OR

Barber: 17.4, 17.6
very concise, the whole chapter 17 covers lin. regression, lin. Classification, SVMs, Kernels and optimization techniques in a very economical easy to understand and short way.
probably the reading of choice for all of these topics if time is scarce and/or passing the course is the main interest.

OR

Bishop: 4.1.1, 4.1.2, 4.1.7, 4.2, 4.3.2, 4.3.4
well rounded presentation. as always didactically nice and complete.

Suggested Reading on Kernels:

Murphy ch 14

OR

Bishop ch 6

OR

Barber, chapter 17

Part I: Problems w.r.t. Kernels

1 The Gaussian kernel

Problem 1. One of the nice things about kernels is that new kernels can be constructed out of already given ones. Use the five kernel construction rules from the lecture to prove that the function

$$\exp\left(-\frac{|\boldsymbol{x}-\boldsymbol{y}|^2}{2\sigma^2}\right)$$

is a kernel.

(Hint: Use the Taylor expansion of the exponential function to prove that $\exp(K_1(\boldsymbol{x},\boldsymbol{y}))$ is a kernel if $K_1(\boldsymbol{x},\boldsymbol{y})$ is a kernel.)

2nd hint: It might help to apply the rule $K_3(\boldsymbol{\phi}(\boldsymbol{x}),\boldsymbol{\phi}(\boldsymbol{y}))$ with the linear kernel $K_3(\boldsymbol{x},\boldsymbol{y}) = \boldsymbol{x}^T\boldsymbol{y}$ and consider a feature map $\boldsymbol{\phi}(\boldsymbol{z})$ with only one feature.

Let K_1 and K_2 be kernels on $\mathcal{X} \subseteq \mathbb{R}^n$, then the following functions are kernels:

1. $K(\boldsymbol{x}, \boldsymbol{y}) = K_1(\boldsymbol{x}, \boldsymbol{y}) + K_2(\boldsymbol{x}, \boldsymbol{y})$,
2. $K(\boldsymbol{x}, \boldsymbol{y}) = a\, K_1(\boldsymbol{x}, \boldsymbol{y})$ for $a > 0$,
3. $K(\boldsymbol{x}, \boldsymbol{y}) = K_1(\boldsymbol{x}, \boldsymbol{y}) K_2(\boldsymbol{x}, \boldsymbol{y})$,
4. $K(\boldsymbol{x}, \boldsymbol{y}) = K_3\big(\boldsymbol{\phi}(\boldsymbol{x}), \boldsymbol{\phi}(\boldsymbol{y})\big)$ for K_3 kernel on $\mathbb{R}^m$ and $\boldsymbol{\phi}: \mathcal{X} \to \mathbb{R}^m$,
5. $K(\boldsymbol{x}, \boldsymbol{y}) = \boldsymbol{x}^T \boldsymbol{B} \boldsymbol{y}$ for $\boldsymbol{B}$ symmetric and positive semi-definite $n \times n$ matrix.

Let K_1 and K_2 be kernels on $\mathcal{X} \subseteq \mathbb{R}^n$, then the following functions are kernels:

1. $K(\boldsymbol{x}, \boldsymbol{y}) = K_1(\boldsymbol{x}, \boldsymbol{y}) + K_2(\boldsymbol{x}, \boldsymbol{y})$,
2. $K(\boldsymbol{x}, \boldsymbol{y}) = a\, K_1(\boldsymbol{x}, \boldsymbol{y})$ for $a > 0$,
3. $K(\boldsymbol{x}, \boldsymbol{y}) = K_1(\boldsymbol{x}, \boldsymbol{y}) K_2(\boldsymbol{x}, \boldsymbol{y})$,
4. $K(\boldsymbol{x}, \boldsymbol{y}) = K_3\big(\boldsymbol{\phi}(\boldsymbol{x}), \boldsymbol{\phi}(\boldsymbol{y})\big)$ for K_3 kernel on $\mathbb{R}^m$ and $\boldsymbol{\phi}: \mathcal{X} \to \mathbb{R}^m$,
5. $K(\boldsymbol{x}, \boldsymbol{y}) = \boldsymbol{x}^T \boldsymbol{B} \boldsymbol{y}$ for $\boldsymbol{B}$ symmetric and positive semi-definite $n \times n$ matrix.

$$\exp(K_1(\boldsymbol{x}, \boldsymbol{y})) = 1 + \sum_{n=1}^{\infty} \frac{1}{n!} K_1(\boldsymbol{x}, \boldsymbol{y})^n$$

$K_1(\boldsymbol{x}, \boldsymbol{y})^n$ is a kernel — R3

$(1/n!) K_1(\boldsymbol{x}, \boldsymbol{y})^n$ is a kernel — R2

$\sum_{n=1}^{\infty} 1/(n!) K_1(\boldsymbol{x}, \boldsymbol{y})^n$ is a kernel — R1

1 is a kernel : R5: $x^T y$ is kernel (B=Id.) (with $x, y \in \mathbb{R}^m$)
R4: $\Phi(x) = \Phi(y) = (1/\sqrt{m}, \ldots, 1/\sqrt{m})$

or:
R5: $x^T y$ is kernel (B=Id.) (with $x, y \in \mathbb{R}^{m=1}$)
R4: $\Phi(x) = \Phi(y) = 1$

Let K_1 and K_2 be kernels on $\mathcal{X} \subseteq \mathbb{R}^n$, then the following functions are kernels:

1. $K(\boldsymbol{x}, \boldsymbol{y}) = K_1(\boldsymbol{x}, \boldsymbol{y}) + K_2(\boldsymbol{x}, \boldsymbol{y})$,
2. $K(\boldsymbol{x}, \boldsymbol{y}) = a\, K_1(\boldsymbol{x}, \boldsymbol{y})$ for $a > 0$,
3. $K(\boldsymbol{x}, \boldsymbol{y}) = K_1(\boldsymbol{x}, \boldsymbol{y}) K_2(\boldsymbol{x}, \boldsymbol{y})$,
4. $K(\boldsymbol{x}, \boldsymbol{y}) = K_3\big(\boldsymbol{\phi}(\boldsymbol{x}), \boldsymbol{\phi}(\boldsymbol{y})\big)$ for K_3 kernel on $\mathbb{R}^m$ and $\boldsymbol{\phi}: \mathcal{X} \to \mathbb{R}^m$,
5. $K(\boldsymbol{x}, \boldsymbol{y}) = \boldsymbol{x}^T \boldsymbol{B} \boldsymbol{y}$ for $\boldsymbol{B}$ symmetric and positive semi-definite $n \times n$ matrix.

$$\exp\left(-\frac{|\boldsymbol{x}-\boldsymbol{y}|^2}{2\sigma^2}\right) = \exp\left(-\frac{(\boldsymbol{x}-\boldsymbol{y})^T(\boldsymbol{x}-\boldsymbol{y})}{2\sigma^2}\right) = \exp\left(-\frac{\boldsymbol{x}^T\boldsymbol{x} - 2\boldsymbol{x}^T\boldsymbol{y} + \boldsymbol{y}^T\boldsymbol{y}}{2\sigma^2}\right)$$
$$= \exp\left(-\frac{\boldsymbol{x}^T\boldsymbol{x}}{2\sigma^2}\right) \exp\left(-\frac{\boldsymbol{y}^T\boldsymbol{y}}{2\sigma^2}\right) \exp\left(\frac{\boldsymbol{x}^T\boldsymbol{y}}{\sigma^2}\right) .$$

R5: $x^T y$ is kernel (B=Id.), with $x, y \in \mathbb{R}^{m=1}$

R4: $\Phi(x) = \exp\left(\frac{-x^T x}{2\sigma}\right)$, $\Phi(y) = \exp\left(\frac{-y^T y}{2\sigma}\right)$

slightly alternate approach to problem 1:

Problem 1a: Verify that $k(X_i, X_j) = f(X_i)k_1(X_i, X_j)f(X_j)$, where k_1 is a valid kernel, is also a kernel.

Problem 1a: Verify that $k(X_i, X_j) = f(X_i)k_1(X_i, X_j)f(X_j)$, where k_1 is a valid kernel, is also a kernel.

A kernel is defined as $k(X_i, X_j) = \phi(X_i)^T\phi(X_j)$. I.e. it is the scalar product between the feature vectors for two input vectors. If k_1 is a valid kernel then it follows that

$$ck_1(X_i, X_j) = u(X_i)^T u(X_j)$$

where

$$u(X_n) = c^{1/2}\phi(X_n)$$

and so $ck_1(X_i, X_j)$ can be expressed as the scalar product of feature vectors. This is how we show the validity of rule 2. We can follow a similar path for showing the above function is also a kernel.

We can write

$$f(X_i)k_1(X_i, X_j)f(X_j) = v(X_i)^T v(X_j)$$

where we have defined $v(X_n) = f(X_n)\phi(X_n)$.

Again, we see that $f(X_i)k_1(X_i, X_j)f(X_j)$ can be expressed as the scalar product of feature vectors, and hence is a valid kernel.

Problem 1b: Verify that $k(X_i, X_j) = q(k_1(X_i, X_j))$ (where q is a polynomial and k_1 is a valid kernel) and $k(X_i, X_j) = \exp(k_1(X_i, X_j))$ are valid rules for constructing a valid kernel.

Problem 1b: Verify that $k(X_i, X_j) = q(k_1(X_i, X_j))$ (where q is a polynomial and k_1 is a valid kernel) and $k(X_i, X_j) = \exp(k_1(X_i, X_j))$ are valid rules for constructing a valid kernel.

We can show that $k(X_i, X_j) = q(k_1(X_i, X_j))$ follows from $k(x, y) = c(k_1(X_i, X_j))$ where $c > 0$ is a constant (Rule 2), $k(X_i, X_j) = (k_1(X_i, X_j)k_2(X_i, X_j))$ where k_2 is also a valid kernel (Rule 3), and $k(X_i, X_j) = k_1(X_i, X_j) + k_2(X_i, X_j)$ (Rule 1).

For $k(X_i, X_j) = \exp(k_1(X_i, X_j))$ we re-write the exponential as a power series yielding:

$$k(X_i, X_j) = \exp(k_1(X_i, X_j))$$

$$= \sum_{m=0}^{\infty} \frac{(k_1(X_i, X_j))^m}{m!}$$

Since this is a polynomial in $k_1(X_i, X_j)$ we can use our proof from above to show that this is also a valid kernel.

Problem 1c: One commonly used kernel is the Gaussian kernel, i.e.

$$k(X_i, X_j) = \exp\left(\frac{-\|X_i - X_j\|^2}{2\sigma^2}\right),$$

but remember that in this context it is not interpreted as a probability density, that's why there isn't a normalization coefficient. We can see that this is a valid kernel by expanding the square

$$\|X_i - X_j\|^2 = X_i^T X_i + X_j^T X_j + 2X_i^T X_j$$

which gives us

$$k(X_i, X_j) = \exp\left(\frac{-X_i^T X_i}{2\sigma^2}\right) \exp\left(\frac{X_i^T X_j}{\sigma^2}\right) \exp\left(\frac{-X_j^T X_j}{2\sigma^2}\right)$$

Which we can derice since we know $k(X_i, X_j) = f(X_i)k_1(X_i, X_j)f(X_j)$ is a valid kernel, and we know that $k(X_i, X_j) = \exp(k_1(X_i, X_j))$ is a valid kernel.

So, by using the expanded version of the kernel from above, and expanding the middle factor as a power series, show that the Gaussian kernel equation we showed at the very top of this exercise can be expressed as the inner product of an infinite-dimensional feature vector.

Since we know that the exponential kernel, $k(X_i, X_j) = \exp(k_1(X_i, X_j))$, can be written as the infinite sum of terms, each of which can itself be written as an inner product of feature vectors (since we know that $k(X_i, X_j) = q(k_1(X_i, X_j))$ where q is a polynomial, is a valid kernel). Thus, by concatenating the feature vectors of the individual terms in that sum, we can write this as an inner product of infinite dimension vectors. More formally,

$$\exp(X_i^T X_j/\sigma^2) = \sum_{m=0}^{\infty} \phi_m(X_i)^T \phi_m(X_j))$$

$$= \psi(X_i)^T \psi(X_j)$$

where $\psi(X_n)^T = [\phi_0(X_n)^T, \phi_1(X_n)^T, \dots]$. Hence, we can write the original form for the Gaussian kernel as

$$k(X_i, X_j) = \rho(X_i)^T \rho(X_j)$$

where

$$\rho(X_n) = \exp\left(\frac{X_n^T X_n}{\sigma^2}\right) \psi(X_n).$$

end of slightly alternate approach to problem 1.

Problem 2: Find an infinite-dimensional feature space $\phi(\boldsymbol{x})$ corresponding to the Gaussian kernel, i.e. determine $\phi(\boldsymbol{x})$ so that

$$\phi(\boldsymbol{x})^T \phi(\boldsymbol{y}) = \exp\left(-\frac{|\boldsymbol{x}-\boldsymbol{y}|^2}{2\sigma^2}\right) .$$

(Hint: The multinomial formula turns a power of a sum into a weighted sum of products,

$$\left(\sum_{t=1}^{m} x_t\right)^n = \sum_{k_1+k_2+\cdots+k_m=n} \binom{n}{k_1, k_2, \ldots, k_m} \prod_{t=1}^{m} x_t^{k_t} ,$$

with $\binom{n}{k_1,k_2,\ldots,k_m} = \frac{n!}{k_1!k_2!\cdots k_m!}$.)

Let $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^m$ and without loss of generality let $\sigma = 1$. We expand the Gaussian kernel into an infinite sum using the Taylor expansion of the exponential function and use the multinomial formula to separate the scalar product,

$$\begin{aligned}
\exp\left(-\frac{|\boldsymbol{x}-\boldsymbol{y}|^2}{2}\right) &= \exp\left(-\frac{\boldsymbol{x}^2}{2}\right)\exp\left(-\frac{\boldsymbol{y}^2}{2}\right)\exp\left(-\boldsymbol{x}^T\boldsymbol{y}\right) \\
&= \exp\left(-\frac{\boldsymbol{x}^2}{2}\right)\exp\left(-\frac{\boldsymbol{y}^2}{2}\right)\sum_{n=0}^{\infty}\frac{(\boldsymbol{x}^T\boldsymbol{y})^n}{n!} \\
&= \exp\left(-\frac{\boldsymbol{x}^2}{2}\right)\exp\left(-\frac{\boldsymbol{y}^2}{2}\right)\sum_{n=0}^{\infty}\frac{1}{n!}\left(\sum_{t=1}^{m}x_t y_t\right)^n \\
&= \exp\left(-\frac{\boldsymbol{x}^2}{2}\right)\exp\left(-\frac{\boldsymbol{y}^2}{2}\right)\sum_{n=0}^{\infty}\frac{1}{n!}\sum_{k_1+k_2+\cdots+k_m=n}\binom{n}{k_1,k_2,\ldots,k_m}\prod_{t=1}^{m}(x_t y_t)^{k_t} \\
&= \sum_{n=0}^{\infty}\sum_{k_1+k_2+\cdots+k_m=n}\frac{\exp\left(-\frac{\boldsymbol{x}^2}{2}\right)}{\sqrt{n!}}\sqrt{\binom{n}{k_1,k_2,\ldots,k_m}}\left(\prod_{t=1}^{m}x_t^{k_t}\right)\cdot \\
&\quad \frac{\exp\left(-\frac{\boldsymbol{y}^2}{2}\right)}{\sqrt{n!}}\sqrt{\binom{n}{k_1,k_2,\ldots,k_m}}\left(\prod_{t=1}^{m}y_t^{k_t}\right).
\end{aligned}$$

Thus the feature space of the Gaussian kernel is

$$\phi(\boldsymbol{x}) = \left[\frac{\exp\left(-\frac{\boldsymbol{x}^2}{2}\right)}{\sqrt{k_1!k_2!\cdots k_m!}}\prod_{t=1}^{m}x_t^{k_t}\right]_{n=0,\ldots,\infty;\, k_1+k_2+\cdots+k_m=n}.$$

Since $k_1!k_2!\cdots k_m! \to \infty$ for $n \to \infty$ the feature vector can be truncated at sufficiently large n.

Problem 3: Informally explain the difference between doing linear regression with a Gaussian RBF basis and doing linear regression with a Gaussian kernel.

When doing regression with a Gaussian RBF basis the centers of the RBFs can be placed anywhere while the Gaussian kernel implicitly uses the training samples as centers. Also, formally the centers of the Gaussian RBFs must be fixed in advance (before seeing the training samples), because the basis functions officially do not depend on the training samples. Linear regression using the Gaussian kernel can only be done in the dual representation, since the associated feature space has infinite dimensionality.

2 Kernel Perceptron

Problem 4: In this exercise, we develop a dual formulation of the perceptron algorithm. Using the perceptron learning rule you learned in the lecture, show that the learned weight vector can be written as a linear combination of the vectors $t^{(n)}\boldsymbol{\phi}(\boldsymbol{x}^{(n)})$ where $t^{(n)} \in \{-1, +1\}$. Denote the coefficients of this linear combination by α_n and derive a formulation of the perceptron learning algorithm, and the predictive function for the perceptron in terms of the α_n. Show that the feature vector $\boldsymbol{\phi}(\boldsymbol{x})$ enters only in the form of the kernel function $K(\boldsymbol{x}, \boldsymbol{y}) = \boldsymbol{\phi}(\boldsymbol{x})^T\boldsymbol{\phi}(\boldsymbol{y})$.

Perceptron:

$$\text{class } t^{(n)} = sgn\left(w^T \phi^{(n)}\right) \in \{-1,1\}$$

correctly classfied $\phi^{(n)}$ ($a^{(n)} = 0$)

$$\phi\left(x^{(n)}\right) = \phi^{(n)} \text{ in class } t^{(n)} = 1 \rightarrow w^T \phi^{(n)} > 0$$

$$\phi\left(x^{(n)}\right) = \phi^{(n)} \text{ in class } t^{(n)} = -1 \rightarrow w^T \phi^{(n)} < 0$$

$$\rightarrow \quad w^T \phi^{(n)} t^{(n)} > 0$$

incorrectly classfied $\phi^{(n)}$ ($a^{(n)} = 1$)

$$\rightarrow \quad w^T \phi^{(n)} t^{(n)} < 0$$

→ minimize Error: $E = -\sum_n a^{(n)} w^T \phi^{(n)} t^{(n)}$

update rule: $w^{new} = w^{old} - \eta \nabla_w E$

$$w^{new} = \begin{cases} w^{old} - \eta\, a^{(n)} \phi^{(n)} t^{(n)} & \text{for online learning} \\ w^{old} - \eta \sum_n a^{(n)} \phi^{(n)} t^{(n)} & \text{for batch learning} \end{cases}$$

$\eta = 1, w^{old} = 0$ → $w = \sum_n a^{(n)} \phi^{(n)} t^{(n)}$

class $t^{(t)} = sgn\left(w^T \phi^{(t)}\right) \in \{-1,1\}$

$$= sgn\left(\sum_n \mathrm{a}^{(n)} t^{(n)} \phi^{(n)T} \phi^{(t)}\right)$$

$$= sgn\left(\sum_n \mathrm{a}^{(n)} t^{(n)} \mathrm{K}(\mathrm{x}^{(n)}, \mathrm{x}^{(t)})\right) = \mathrm{sgn}(\mathrm{g}(x^{(t)}))$$

→ learning rule for Kernel Perceptron:

Initialize $a_j = 0$ for all $j = 1, \ldots, N$
do
  for $t = 1, \ldots, n$
    $y' \leftarrow sgn(g(\boldsymbol{x}^{(t)}))$
    if $y' \neq t^{(t)}$ then $a_t \leftarrow a_t + 1$
  end for
repeat until no classification mistakes

Thus the update rule $a_t \leftarrow a_t + 1$ for a misclassified training sample j corresponds to the update rule $\boldsymbol{w} \leftarrow \boldsymbol{w} + \boldsymbol{\phi}(\boldsymbol{x}^{(j)})$ of the conventional Perceptron training algorithm.

Part II: Problems w.r.t. Linear Classification

Problem 1: Given a set of data points $\{\boldsymbol{x}_n\}$, we can define the *convex hull* to be the set of all points $\boldsymbol{x}$ given by

$$\boldsymbol{x} = \sum_n \alpha_n \boldsymbol{x}_n$$

where $\alpha_n \geq 0$ and $\sum_n \alpha_n = 1$. Consider a second set of points $\{\boldsymbol{y}_n\}$ together with their corresponding convex hull. By definition, the two sets of points will be linearly separable if there exists a vector $\boldsymbol{w}$ and a scalar w_0 such that $\boldsymbol{w}^T \boldsymbol{x}_n + w_0 > 0$ for all $\boldsymbol{x}_n$, and $\boldsymbol{w}^T \boldsymbol{y}_n + w_0 < 0$ for all $\boldsymbol{y}_n$. Show that if their convex hulls intersect, the two sets of points cannot be linearly separable, and conversely that if they are linearly separable, their convex hulls do not intersect.

Problem 1: Given a set of data points $\{\boldsymbol{x}_n\}$, we can define the *convex hull* to be the set of all points $\boldsymbol{x}$ given by

$$\boldsymbol{x} = \sum_n \alpha_n \boldsymbol{x}_n$$

where $\alpha_n \geq 0$ and $\sum_n \alpha_n = 1$. Consider a second set of points $\{\boldsymbol{y}_n\}$ together with their corresponding convex hull. By definition, the two sets of points will be linearly separable if there exists a vector $\boldsymbol{w}$ and a scalar w_0 such that $\boldsymbol{w}^T\boldsymbol{x}_n + w_0 > 0$ for all $\boldsymbol{x}_n$, and $\boldsymbol{w}^T\boldsymbol{y}_n + w_0 < 0$ for all $\boldsymbol{y}_n$. Show that if their convex hulls intersect, the two sets of points cannot be linearly separable, and conversely that if they are linearly separable, their convex hulls do not intersect.

We prove *Convex hull intersect* $\Rightarrow$ *not separable.* We have two sets of points, $\{\boldsymbol{x}_n\}$ and $\{\boldsymbol{y}_n\}$. Assume there is a $\boldsymbol{z}$ that lies in both convex hulls, i.e. there are $\{\alpha_n\}$ and $\{\beta_n\}$ such that

$$\boldsymbol{z} = \sum_n \alpha_n \boldsymbol{x}_n = \sum_n \beta_n \boldsymbol{y}_n$$

Now, assume the two set of points are linearly separable, i.e. there is a $\boldsymbol{w}$ for all $\{\boldsymbol{x}_n\}$ and $\{\boldsymbol{y}_n\}$ with $\boldsymbol{w}^T\boldsymbol{x}_n + w_0 > 0$ and $\boldsymbol{w}^T\boldsymbol{y}_n + w_0 < 0$. Now for $\boldsymbol{w}^T\boldsymbol{z}$ we get

$$\boldsymbol{w}^T\boldsymbol{z} = \sum_n \alpha_n \boldsymbol{w}^T\boldsymbol{x}_n > \sum_n \alpha_n(-w_0) = -w_0$$

but also

$$\boldsymbol{w}^T\boldsymbol{z} = \sum_n \beta_n \boldsymbol{w}^T\boldsymbol{y}_n < \sum_n \beta_n(-w_0) = -w_0$$

Logistic Regression (a **discriminative** classifier)

Problem 3 *Given a dataset* $\mathcal{D} = \{(\mathbf{x}^n, c^n), n = 1, \ldots, N\}$, *where* $c^n \in \{0, 1\}$, *logistic regression uses the model* $p(c = 1|\mathbf{x}) = \sigma(\mathbf{w}^T\mathbf{x} + b)$. *Assuming the data is drawn independently and identically, show that the derivative of the log likelihood* L *with respect to* $\mathbf{w}$ *is*

$$\nabla_{\mathbf{w}} L = \sum_{n=1}^{N} \left(c^n - \sigma\left(\mathbf{w}^T\mathbf{x}^n + b \right) \right) \mathbf{x}^n$$

source:Barber

Logistic Regression (a **discriminative** classifier)

Problem 3 *Given a dataset* $\mathcal{D} = \{(\mathbf{x}^n, c^n), n = 1, \ldots, N\}$, *where* $c^n \in \{0, 1\}$, *logistic regression uses the model* $p(c = 1|\mathbf{x}) = \sigma(\mathbf{w}^T\mathbf{x} + b)$. *Assuming the data is drawn independently and identically, show that the derivative of the log likelihood* L *with respect to* $\mathbf{w}$ *is*

$$\nabla_{\mathbf{w}} L = \sum_{n=1}^{N} \left(c^n - \sigma\left(\mathbf{w}^T\mathbf{x}^n + b \right) \right) \mathbf{x}^n$$

$$L = \sum_n c^n \log \sigma(\mathbf{w}^\mathsf{T}\mathbf{x}^n + b) + (1 - c^n) \log(1 - \sigma(\mathbf{w}^\mathsf{T}\mathbf{x}^n + b))$$

Using the derivative result $\sigma' = \sigma(1 - \sigma)$, and $\sigma_n \equiv \sigma(\mathbf{w}^\mathsf{T}\mathbf{x}^n + b)$ we have

$$\nabla_{\mathbf{w}} L = \sum_{n=1} c^n(1 - \sigma_n)\mathbf{x}^n - (1 - c^n)\sigma_n\mathbf{x}^n = \sum_{n=1} (c^n - \sigma_n)\mathbf{x}^n$$

source:Barber

Multi-Class Logistic Regression (a **discriminative** classifier)

Problem 4 *Consider the softmax function for classifying an input vector* $\mathbf{x}$ *into one of* $c = 1, \ldots, C$ *classes using*

$$p(c|\mathbf{x}) = \frac{e^{\mathbf{w}_c^T \mathbf{x}}}{\sum_{c'=1}^{C} e^{\mathbf{w}_{c'}^T \mathbf{x}}}$$

A set of input-class examples is given by $\mathcal{D} = \{(\mathbf{x}^n, c^n), n = 1, \ldots, N\}$.

1. *Relate this model to logistic regression when* $C = 2$.
2. *Write down the log-likelihood* L *of the classes conditional on the inputs, assuming that the data is i.i.d.*

source:Barber

Multi-Class Logistic Regression (a **discriminative** classifier)

Problem 4 *Consider the softmax function for classifying an input vector* $\mathbf{x}$ *into one of* $c = 1, \ldots, C$ *classes using*

$$p(c|\mathbf{x}) = \frac{e^{\mathbf{w}_c^T \mathbf{x}}}{\sum_{c'=1}^{C} e^{\mathbf{w}_{c'}^T \mathbf{x}}}$$

A set of input-class examples is given by $\mathcal{D} = \{(\mathbf{x}^n, c^n), n = 1, \ldots, N\}$.

1. *Relate this model to logistic regression when* $C = 2$.
2. *Write down the log-likelihood* L *of the classes conditional on the inputs, assuming that the data is i.i.d.*

1. When $C = 2$ we have

$$p(c = 1|\mathbf{x}) = \frac{e^{\mathbf{w}_1^\mathsf{T}\mathbf{x}}}{e^{\mathbf{w}_1^\mathsf{T}\mathbf{x}} + e^{\mathbf{w}_2^\mathsf{T}\mathbf{x}}} = \frac{1}{1 + e^{(\mathbf{w}_2 - \mathbf{w}_1)^\mathsf{T}\mathbf{x}}} = \sigma\left(\mathbf{w}_2 - \mathbf{w}_1{}^\mathsf{T}\mathbf{x}\right)$$

For $C = 2$, this is therefore logistic regression parameterised by the difference $\mathbf{w} \equiv \mathbf{w}_2 - \mathbf{w}_1$. Note that there is therefore redundancy in the softmax parameterisation since only this difference plays a role in computing the probability. The same hold for $C > 2$ – by multiplying and dividing by $\exp(-\mathbf{w}_d^\mathsf{T}\mathbf{x})$, for some chosen $d \in \{1, \ldots, C\}$ only the differences $\mathbf{w}'_c \equiv \mathbf{w}_c - \mathbf{w}_d$ are required to specify the probability. This is potentially important to realise for optimisation purposes since there will therefore be equivalent solutions in the redundant parameterisation.

2.

$$L(\mathcal{D}) = \sum_n \left[\mathbf{w}_{c^n}^\mathsf{T}\mathbf{x}^n - \log \sum_{d=1}^{C} e^{\mathbf{w}_d^\mathsf{T}\mathbf{x}^n}\right]$$

source:Barber

Regularized Logistic Regression (MAP Solution to w)

(a **discriminative** classifier)

(Source: Jaaakkola.)

Problem 5

a. Consider the data in Figure AAA, where we fit the model $p(y = 1|\mathbf{x}, \mathbf{w}) = \sigma(w_0 + w_1 x_1 + w_2 x_2)$. Suppose we fit the model by maximum likelihood, i.e., we minimize

$$J(\mathbf{w}) = -\ell(\mathbf{w}, \mathcal{D}_{\text{train}}) \tag{8.133}$$

where $\ell(\mathbf{w}, \mathcal{D}_{\text{train}})$ is the log likelihood on the training set. Sketch a possible decision boundary corresponding to $\hat{\mathbf{w}}$. (Copy the figure first (a rough sketch is enough), and then superimpose your answer on your copy, since you will need multiple versions of this figure). Is your answer (decision boundary) unique? How many classification errors does your method make on the training set?

b. Now suppose we regularize only the w_0 parameter, i.e., we minimize

$$J_0(\mathbf{w}) = -\ell(\mathbf{w}, \mathcal{D}_{\text{train}}) + \lambda w_0^2 \tag{8.134}$$

Suppose λ is a very large number, so we regularize w_0 all the way to 0, but all other parameters are unregularized. Sketch a possible decision boundary. How many classification errors does your method make on the training set? Hint: consider the behavior of simple linear regression, $w_0 + w_1 x_1 + w_2 x_2$ when $x_1 = x_2 = 0$.

c. Now suppose we heavily regularize only the w_1 parameter, i.e., we minimize

$$J_1(\mathbf{w}) = -\ell(\mathbf{w}, \mathcal{D}_{\text{train}}) + \lambda w_1^2 \tag{8.135}$$

Sketch a possible decision boundary. How many classification errors does your method make on the training set?

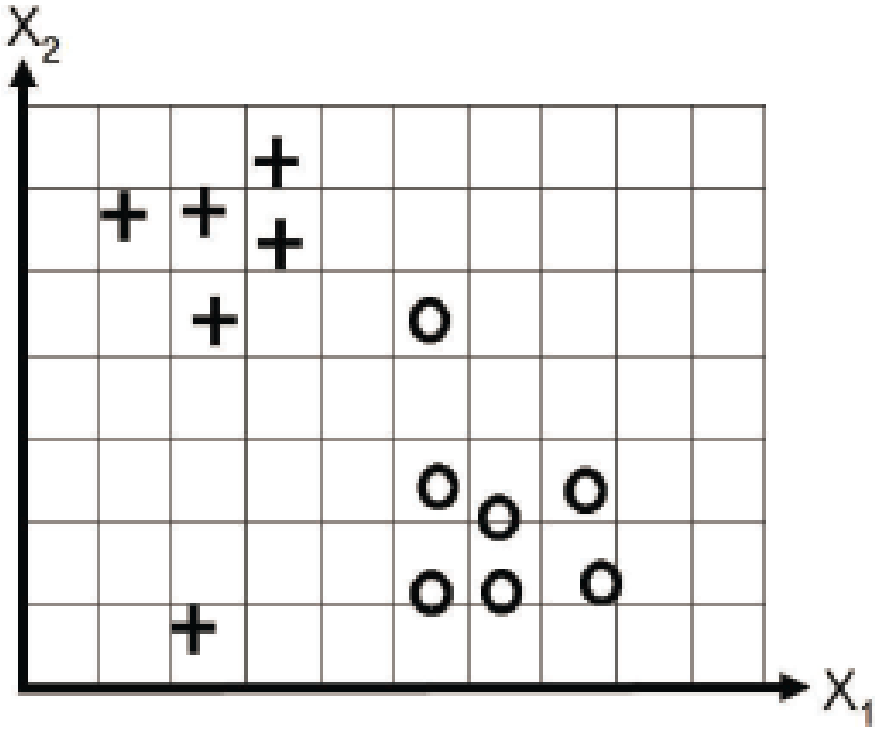


Figure AAA

1. Unregularized solution: see Figure 14.1(a). There is a unique ML decision boundary which makes 0 errors, but there are several possible decision boundaries which all makes **0 errors**.

2. Heavily regularizing w_0 sets $w_0 = 0$, so the line must go through the origin. There are several possible lines with different slopes, but all will make **1 error**. see Figure 14.1(b).

3. Regularizing w_1 makes the line horizontal since x_1 is ignored. This incurs **2 errors**. see Figure 14.1(c).

4. Regularizing w_2 makes the line vertical, since x_2 is ignored. This incurs **0 errors**. see Figure 14.1(d).

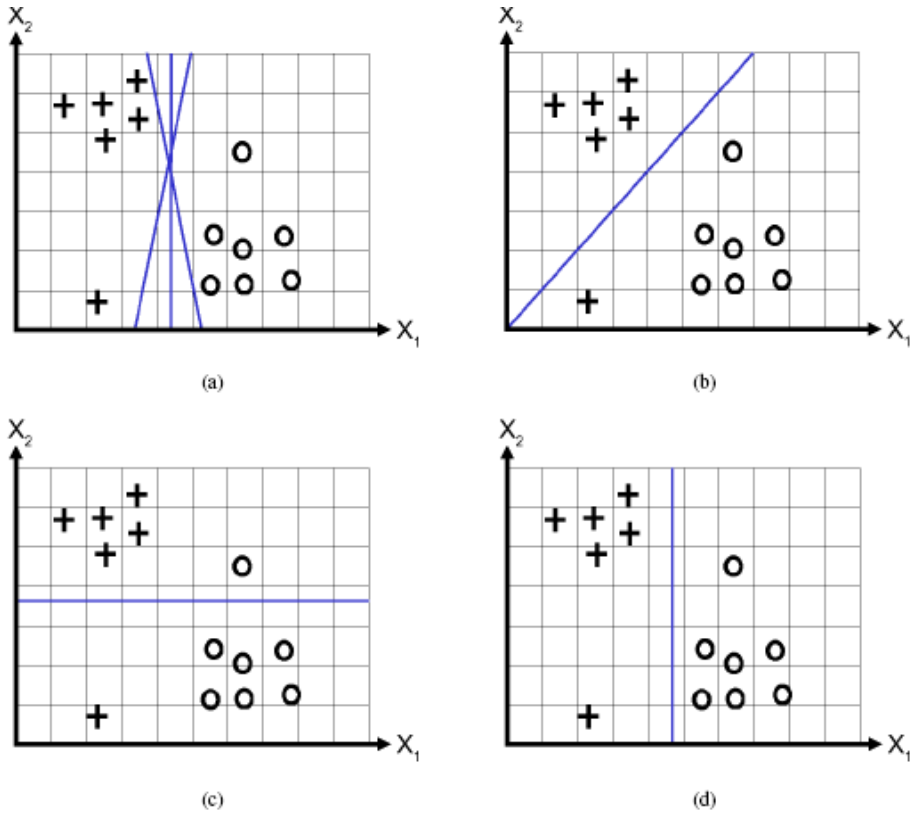


Figure 14.1: (a) Unregularized. (b) Regularizing w_0. (c) Regularizing w_1. (d) Regularizing w_2.

Multi-Class Classification (**Generative** Multi-Class Classifiers)

Problem 6: Consider a generative classification model for K classes defined by prior class probabilities $p(y = k) = \pi_k$ and general class-conditional densities $p(\phi(x)|y = k, \theta_k)$ where $\phi(x)$ is the input feature vector and $\theta = \{\theta_k\}_{k=1}^{K}$ are further model parameters. Suppose we are given a training set $\mathcal{D} = \{(\phi(x^{(n)}), t^{(n)})\}_{n=1}^{N} = \{(\phi^{(n)}, t^{(n)})\}_{n=1}^{N}$ where $t^{(n)}$ is a binary target vector of length K that uses the 1-of-K(hot one) coding scheme, so that it has components $t_j^{(n)} = \delta_{jk}$ if pattern n is from class $y = k$. Assuming that the data points are iid, show that the maximum-likelihood solution for the prior probabilities is given by

$$\pi_k = \frac{N_k}{N}$$

where N_k is the number of data points assigned to class $y = k$.

The likelihood function of the parameters $\{\pi_k, \theta_k\}_{k=1}^K$ is given by

$$p(\mathcal{D}|\{\pi_k, \theta_k\}_{k=1}^K) = \prod_{n=1}^N \prod_{k=1}^K (p(\phi^{(n)}|\theta_k)\pi_k)^{t_k^{(n)}}$$

so the log-likelihood is given by

$$\log p(\mathcal{D}|\{\pi_k, \theta_k\}_{k=1}^K) = \sum_{n=1}^N \sum_{k=1}^K t_k^{(n)} \log \pi_k + \text{const in } \pi_k$$

In order to maximize the log likelihood with respect to π_k we need to preserve the constraint $\sum_k \pi_k = 1$. This can be done by introducing a Lagrange multiplier *lambda* and maximizing

$$\sum_{n=1}^N \sum_{k=1}^K t_k^{(n)} \log \pi_k + \lambda \left(\sum_{k=1}^K \pi_k - 1 \right).$$

Setting the derivative with respect to π_k equal to zero, we obtain

$$\lambda = \frac{1}{\pi_k} \sum_{n=1}^N t_j^{(n)} = \frac{1}{\pi_k} N_k.$$

Setting the derivative with respect to λ equal to zero, we obtain our constraint

$$\sum_{k=1}^K \pi_k = 1.$$

where we can now insert the previous result $\pi_k = \frac{N_k}{\lambda}$ and obtain

$$\lambda = \sum_k N_k = N.$$

So overall we obtain

$$\pi_k = \frac{N_k}{N}.$$

Linear Discriminant Analysis (LDA) (a *generative* classifier)

Problem 7

Topic:

Gaussian Discriminant Analysis (GDA)

(Variant (a): Quadratic Discriminant Analysis (QDA) : Σ_c class specific

Variant (b): Linear Discriminant Analysis (LDA): Σ shared among classes

)

vs. Logistic Regression

(compare lecture-slides on linear generative models)

Problem 7: Using the same classification model as in the previous question, now suppose that the class-conditional densities are given by Gaussian distributions with a shared covariance matrix, so that

$$p(\phi(x)|y = k, \theta_k) = p(\phi(x)|\theta_k) = \mathcal{N}(\phi \mid \mu_k, \Sigma).$$

Show that the maximum likelihood solution for the mean of the Gaussian distribution for class C_k is given by

$$\mu_k = \frac{1}{N_k} \sum_{\{n|\phi^{(n)} \epsilon C_k\}} \phi^{(n)}$$

which represents the mean of those feature vectors assigned to class C_k.

Similarly, show that the maximum likelihood solution for the shared covariance matrix is given by

$$\Sigma = \sum_{k=1}^{K} \frac{N_k}{N} \mathbf{S}_k$$

where

$$\mathbf{S}_k = \frac{1}{N_k} \sum_{\{n|\phi^{(n)} \epsilon C_k\}} (\phi^{(n)} - \mu_k)(\phi^{(n)} - \mu_k)^T.$$

Thus Σ is given by a weighted average of the covariances of the data associated with each class, in which the weighting coefficients N_k/N are the prior probabilities of the classes.

If we substitute $p(\phi|\theta_k) = \mathcal{N}(\phi \mid \mu_k, \Sigma)$ into $\log p(\mathcal{D}|\{\pi_k, \theta_k\}_{k=1}^K)$ and then use the definition of the multivariate Gaussian, we obtain

$$\log p(\mathcal{D}|\{\pi_k, \theta_k\}_{k=1}^K) = \frac{-1}{2} \sum_{n=1}^{N} \sum_{k=1}^{K} t_k^{(n)} (\log|\Sigma| + (\phi^{(n)} - \mu_k)^T \Sigma^{-1} (\phi^{(n)} - \mu_k) + \log \pi_k)$$

Dropping terms independent of μ_k and Σ we get

$$\log p(\mathcal{D}|\{\theta_k\}_{k=1}^K) = \frac{-1}{2} \sum_{n=1}^{N} \sum_{k=1}^{K} t_k^{(n)} (\log|\Sigma| + (\phi^{(n)} - \mu_k)^T \Sigma^{-1} (\phi^{(n)} - \mu_k))$$

Setting the derivative of the above equation w.r.t μ_k, (obtained using $\frac{\partial}{\partial x}(x^T a) = \frac{\partial}{\partial x}(a^T x) = a$), to zero we get

$$\sum_{n=1}^{N} t_k^{(n)} \Sigma^{-1} (\phi^{(n)} - \mu_k) = 0.$$

Making use of what we learned in the last problem, i.e.

$$\sum_{n=1}^{N} t_k^{(n)} = N_k,$$

we can re-arrange this to obtain

$$\mu_k = \frac{1}{N_k} \sum_{n=1}^{N} t_k^{(n)} \phi^{(n)}.$$

Using the trace trick ($a = \mathrm{Tr}(a)$ for $a \in \mathbb{R}$ and $\mathrm{Tr}(ABC) = \mathrm{Tr}(BCA)$) we can rewrite our original expression $\log p(\mathcal{D}|\{\theta_k\}_{k=1}^K) = \frac{-1}{2}\sum_{n=1}^N \sum_{k=1}^K t_k^{(n)}(\log|\Sigma| + (\phi^{(n)} - \mu_k)^T\Sigma^{-1}(\phi^{(n)} - \mu_k))$ as

$$\log p(\mathcal{D}|\{\theta_k\}_{k=1}^K) = \frac{-1}{2}\sum_{n=1}^N \sum_{k=1}^K t_k^{(n)}(\log|\Sigma| + \mathrm{Tr}(\Sigma^{-1}(\phi^{(n)} - \mu_k)(\phi^{(n)} - \mu_k)^T).$$

We can now use $\frac{\partial}{\partial A}Tr(AB) = B^T$ and $\frac{\partial}{\partial A}\ln|A| = (A^{-1})^T$ and $|A^{-1}| = -|A|$ to calculate the derivative w.r.t. Σ^{-1}. Setting this to zero we obtain

$$\frac{1}{2}\sum_{n=1}^N \sum_{k=1}^K t_k^{(n)}(\Sigma - (\phi^{(n)} - \mu_k)(\phi^{(n)} - \mu_k)^T) = 0.$$

Making use of $\sum_{n=1}^N t_k^{(n)} = N_k$, we can re-arrange this to obtain

$$\Sigma = \sum_{k=1}^K \frac{N_k}{N}\mathbf{S}_k$$

,

where

$$\mathbf{S}_k = \frac{1}{N_k}\sum_{n=1}^N t_k^{(n)}(\phi^{(n)} - \mu_k)(\phi^{(n)} - \mu_k)^T.$$

Note that we do not enforce that Σ should be symmetric, but the solution shows that it is automatically symmetric.

Questions to think about:

(a) What exactly is the difference between a generative classifier (e.g. GDA) and a discriminative classifier (e.g. Logistic Regression)?

(a) What are pros and cons of each approach?

(for „solution“ see Murphy 4.2 and 8.6)

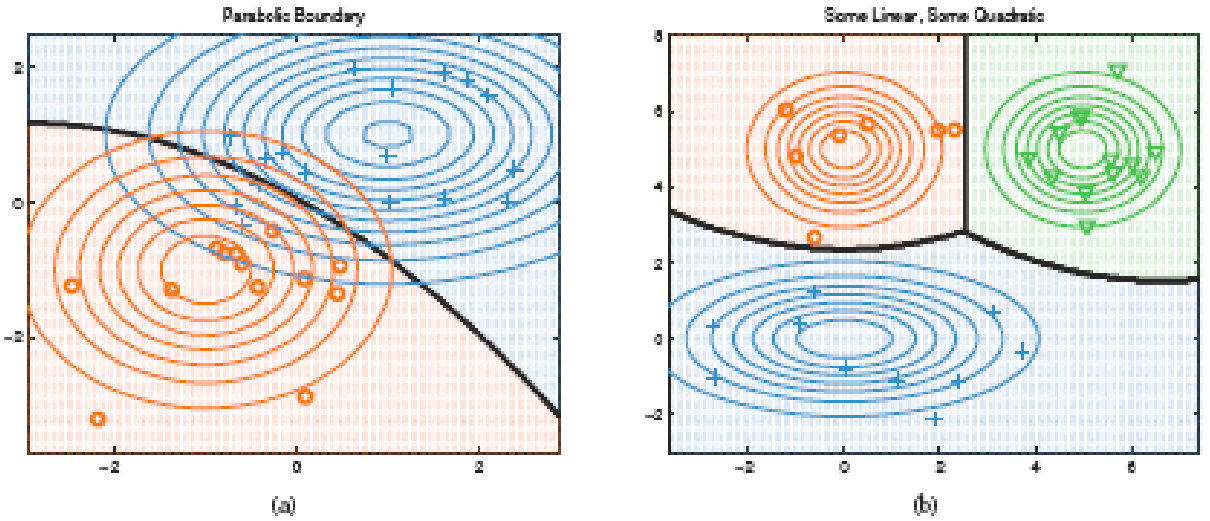


Figure 4.3 Quadratic decision boundaries in 2D for the 2 and 3 class case. Figure generated by `discrimAnalysisDboundariesDemo`.

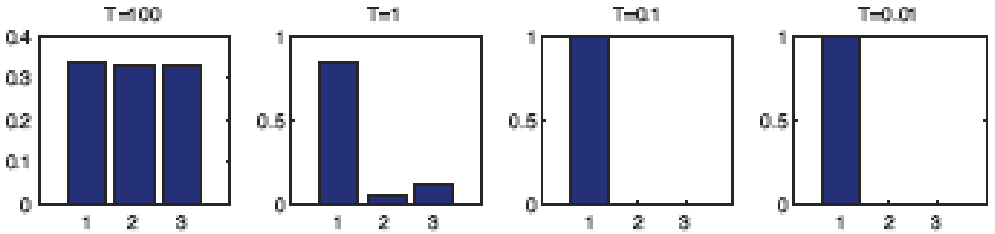


Figure 4.4 Softmax distribution $\mathcal{S}(\eta/T)$, where $\eta = (3, 0, 1)$, at different temperatures T. When the temperature is high (left), the distribution is uniform, whereas when the temperature is low (right), the distribution is "spiky", with all its mass on the largest element. Figure generated by `softmaxDemo2`.

$$p(y = c|\mathbf{x}, \theta) = \frac{\pi_c |2\pi\Sigma_c|^{-\frac{1}{2}} \exp\left[-\frac{1}{2}(\mathbf{x} - \mu_c)^T \Sigma_c^{-1}(\mathbf{x} - \mu_c)\right]}{\sum_{c'} \pi_{c'} |2\pi\Sigma_{c'}|^{-\frac{1}{2}} \exp\left[-\frac{1}{2}(\mathbf{x} - \mu_{c'})^T \Sigma_{c'}^{-1}(\mathbf{x} - \mu_{c'})\right]}$$

source: Murphy

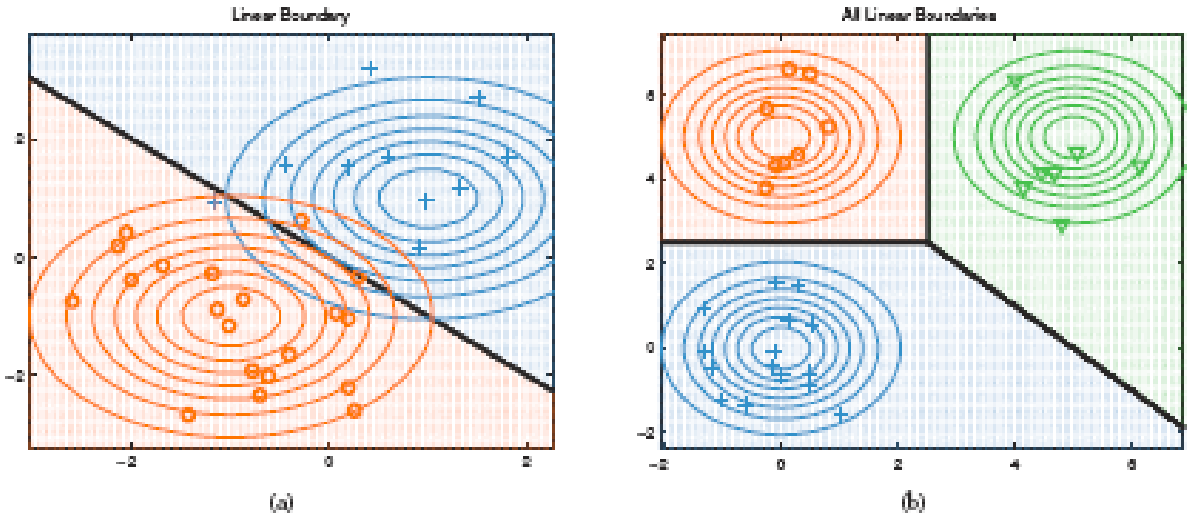


Figure 4.5 Linear decision boundaries in 2D for the 2 and 3 class case. Figure generated by `discrimAnalysisDboundariesDemo`.

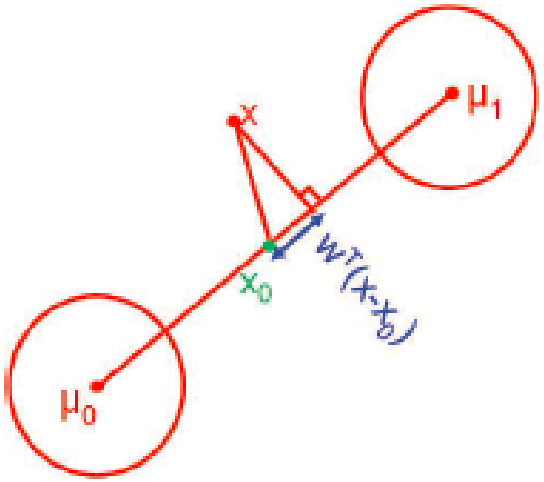


Figure 4.6 Geometry of LDA in the 2 class case where $\Sigma_1 = \Sigma_2 = \mathbf{I}$.

source: Murphy

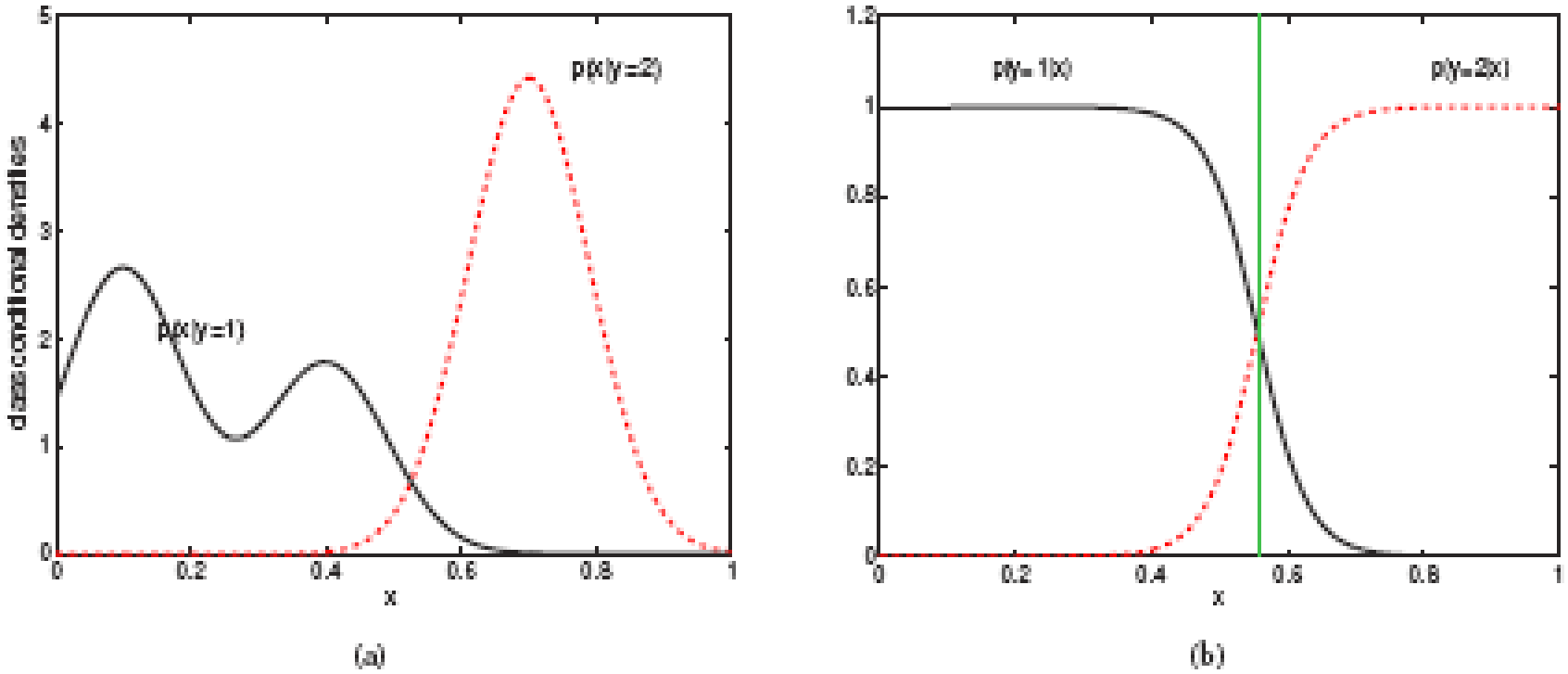


Figure 8.10 The class-conditional densities $p(x|y = c)$ (left) may be more complex than the class posteriors $p(y = c|x)$ (right). Based on Figure 1.27 of (Bishop 2006a). Figure generated by `generativeVsDiscrim`.

source: Murphy

Pros and cons of each approach

- **Easy to fit?** As we have seen, it is usually very easy to fit generative classifiers. For example, in Sections 3.5.1.1 and 4.2.4, we show that we can fit a naive Bayes model and an LDA model by simple counting and averaging. By contrast, logistic regression requires solving a convex optimization problem (see Section 8.3.4 for the details), which is much slower. g +1
- **Fit classes separately?** In a generative classifier, we estimate the parameters of each class conditional density independently, so we do not have to retrain the model when we add more classes. In contrast, in discriminative models, all the parameters interact, so the whole model must be retrained if we add a new class. (This is also the case if we train a generative model to maximize a discriminative objective Salojarvi et al. (2005).) g +1
- **Handle missing features easily?** Sometimes some of the inputs (components of $\mathbf{x}$) are not observed. In a generative classifier, there is a simple method for dealing with this, as we discuss in Section 8.6.2. However, in a discriminative classifier, there is no principled solution to this problem, since the model assumes that $\mathbf{x}$ is always available to be conditioned on (although see (Marlin 2008) for some heuristic approaches). g +1
- **Can handle unlabeled training data?** There is much interest in **semi-supervised learning**, which uses unlabeled data to help solve a supervised task. This is fairly easy to do using generative models (see e.g., (Lasserre et al. 2006; Liang et al. 2007)), but is much harder to do with discriminative models. g +1
- **Symmetric in inputs and outputs?** We can run a generative model "backwards", and infer probable inputs given the output by computing $p(\mathbf{x}|y)$. This is not possible with a discriminative model. The reason is that a generative model defines a joint distribution on $\mathbf{x}$ and y, and hence treats both inputs and outputs symmetrically. g +1
- **Can handle feature preprocessing?** A big advantage of discriminative methods is that they allow us to preprocess the input in arbitrary ways, e.g., we can replace $\mathbf{x}$ with $\phi(\mathbf{x})$, which could be some basis function expansion, as illustrated in Figure 8.9. It is often hard to define a generative model on such pre-processed data, since the new features are correlated in complex ways. d +1
- **Well-calibrated probabilities?** Some generative models, such as naive Bayes, make strong independence assumptions which are often not valid. This can result in very extreme posterior class probabilities (very near 0 or 1). Discriminative models, such as logistic regression, are usually better calibrated in terms of their probability estimates. d +1

source: Murphy

Exercise 4.20 Logistic regression vs LDA/QDA

(Source: Jaakkola.) Suppose we train the following binary classifiers via maximum likelihood.

a. GaussI: A generative classifier, where the class conditional densities are Gaussian, with both covariance matrices set to $\mathbf{I}$ (identity matrix), i.e., $p(\mathbf{x}|y=c) = \mathcal{N}(\mathbf{x}|\mu_c, \mathbf{I})$. We assume $p(y)$ is uniform.

b. GaussX: as for GaussI, but the covariance matrices are unconstrained, i.e., $p(\mathbf{x}|y=c) = \mathcal{N}(\mathbf{x}|\mu_c, \mathbf{\Sigma}_c)$.

c. LinLog: A logistic regression model with linear features.

d. QuadLog: A logistic regression model, using linear and quadratic features (i.e., polynomial basis function expansion of degree 2).

After training we compute the performance of each model M on the training set as follows:

$$L(M) = \frac{1}{n}\sum_{i=1}^{n} \log p(y_i|\mathbf{x}_i, \hat{\theta}, M) \tag{4.285}$$

(Note that this is the *conditional* log-likelihood $p(y|\mathbf{x}, \hat{\theta})$ and not the joint log-likelihood $p(y, \mathbf{x}|\hat{\theta})$.) We now want to compare the performance of each model. We will write $L(M) \leq L(M')$ if model M *must* have lower (or equal) log likelihood (on the training set) than M', for any training set (in other words, M is worse than M', at least as far as training set logprob is concerned). For each of the following model pairs, state whether $L(M) \leq L(M')$, $L(M) \geq L(M')$, or whether no such statement can be made (i.e., M might sometimes be better than M' and sometimes worse); also, for each question, briefly (1-2 sentences) explain why.

a. GaussI, LinLog.

b. GaussX, QuadLog.

c. LinLog, QuadLog.

d. GaussI, QuadLog.

e. Now suppose we measure performance in terms of the average misclassification rate on the training set:

$$R(M) = \frac{1}{n}\sum_{i=1}^{n} I(y_i \neq \hat{y}(\mathbf{x}_i)) \tag{4.286}$$

Is it true in general that $L(M) > L(M')$ implies that $R(M) < R(M')$? Explain why or why not.

source: Murphy

1. GaussI $\leq$ LinLog. Both have logistic (sigmoid) posteriors $p(y|\mathrm{x}, \mathrm{w}) = \sigma(y\mathrm{w}^T\mathrm{x})$, but LinLog is the logistic model which is trained to maximize $p(y|\mathrm{x}, \mathrm{w})$. (GaussI may have high joint $p(y, \mathrm{x})$, but this does not necessarily mean $p(y|\mathrm{x})$ is high; LinLog can achieve the maximum of $p(y|\mathrm{x})$, so will necessarily do at least as well as GaussI.)

2. GaussX $\leq$ QuadLog. Both have logistic posteriors with quadratic features, but QuadLog is the model of this class maximizing the average log probabilities.

3. LinLog $\leq$ QuadLog. Logistic regression models with linear features are a subclass of logistic regression models with quadratic functions. The maximum from the superclass is at least as high as the maximum from the subclass.

4. GaussI $\leq$ QuadLog. Follows from above inequalities.

5. Although one might expect that higher log likelihood results in better classification performance, in general, having higher average log $p(y|\mathrm{x})$ does not necessarily translate to higher or lower classification error. For example, consider linearly separable data. We have L(linLog) $>$ L(GaussI), since maximum likelihood logistic regression will set the weights to infinity, to maximize the probability of the correct labels (hence $p(y_i|\mathrm{x}_i, \hat{\mathrm{w}}) = 1$ for all i). However, we have R(linLog) = R(gaussI), since the data is linearly separable. (The GaussI model may or may not set σ very small, resulting in possibly very large class conditional pdfs; however, the posterior over y is a discrete pmf, and can never exceed 1.)

   As another example, suppose the true label is always 1 (as opposed to 0), but model M always predicts $p(y = 1|\mathrm{x}, M) = 0.49$. It will always misclassify, but it is at least close to the decision boundary. By contrast, there might be another model M' that predicts $p(y = 1|\mathrm{x}, M') = 1$ on even-numbered inputs, and $p(y = 1|\mathrm{x}, M') = 0$ on odd-numbered inputs. Clearly $R(M') = 0.5 < R(M) = 1$, but $L(M') = -\infty < L(M) = \log(0.49)$.

Hinge loss

The hinge loss is given as

$$\mathcal{L}(\mathbf{x}_i) = \max(0, 1 - y_i \tilde{z}_i),$$

where $y_i = \mathbf{x}_i^T \mathbf{w}$ is the model output and z_i the target variable ($w_0 = b$ and $x_{i,0} = 1$).

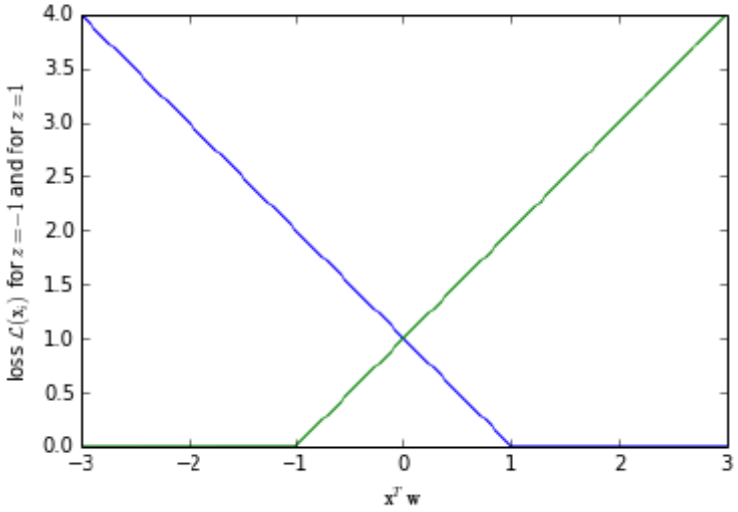
Note that in this case the computation uses class labels $\tilde{z} = 2z - 1 \in \{-1, 1\}$ instead of $z \in \{0, 1\}$.

For multiple samples $\mathbf{X}$ and respective outputs $\mathbf{y}$ and $\tilde{\mathbf{z}}$ the loss is $\mathcal{L}(\mathbf{X}) = \sum_i \mathcal{L}(\mathbf{x}_i)$.

Problem 8 Try to understand what the hinge loss does and explain it in a few of words

Problem 9 Derive a sub-gradient of the hinge loss $\frac{\mathrm{d}\mathcal{L}(\mathbf{x}_i)}{\mathrm{dw}}$. (You may assume that $w_0 = b$. Take 0 for the sub-gradient at non-differentiable points. Otherwise deriving a sub-gradient is equivalent to deriving the gradient.)

The decision boundary of the hinge loss is where $y = \boldsymbol{x}^T \boldsymbol{w} = 0$. The loss function for $z = 1$ and $z = -1$ looks like:



The hinge loss attempts to create a small margin between the decision boundary and the closest training samples. It slightly penalizes even samples that are on the correct side of the decision boundary but are within the margin ($|\boldsymbol{x}^T \boldsymbol{w}| < 1$) and penalizes incorrectly classified samples linearly with their distance from the (inner) margin.

The hinge loss is not differentiable at $y_i \tilde{z}_i = 1$. However, since the hinge loss is convex, we can derive a subgradient:

$$\begin{aligned}
\frac{\mathrm{d}\mathcal{L}(\boldsymbol{x}_i)}{\mathrm{d}\boldsymbol{w}} &= \begin{cases} \frac{\mathrm{d}(1-y_i\tilde{z}_i)}{\mathrm{d}\boldsymbol{w}} & \text{if } y_i\tilde{z}_i < 1 \\ 0 & \text{else} \end{cases} \\
&= \begin{cases} \frac{\mathrm{d}(1-\tilde{z}_i\boldsymbol{x}_i^T\boldsymbol{w})}{\mathrm{d}\boldsymbol{w}} & \text{if } y_i\tilde{z}_i < 1 \\ 0 & \text{else} \end{cases} \\
&= \begin{cases} -\tilde{z}_i\boldsymbol{x}_i & \text{if } y_i\tilde{z}_i < 1 \\ 0 & \text{else} \end{cases}
\end{aligned}$$

For multiple outputs $\boldsymbol{y} = \boldsymbol{X}\boldsymbol{w}$ and targets $\tilde{z}$:

$$\frac{\mathrm{d}\mathcal{L}(\boldsymbol{X})}{\mathrm{d}\boldsymbol{w}} = \sum_i -z_i\boldsymbol{x}_i, \qquad i \in \{1, \ldots, N\} : y_i z_i < 1$$

Soft Zero-one loss

The soft zero-one loss is given as

$$\mathcal{L}(\mathbf{x}_i) = (\sigma(\beta y_i) - z_i)^2,$$

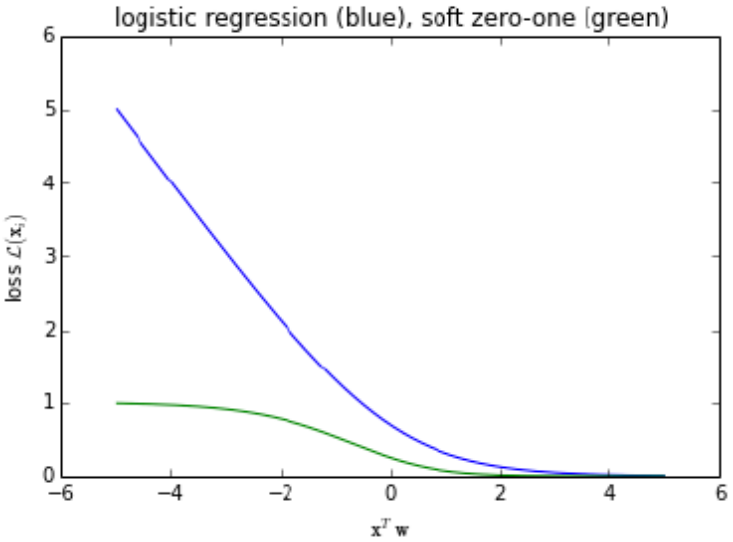
with $y_i = \mathbf{x}_i^T \mathbf{w}$ the model output and z_i the target variable.

Problem 10 Explain the soft zero-one loss in a few words.

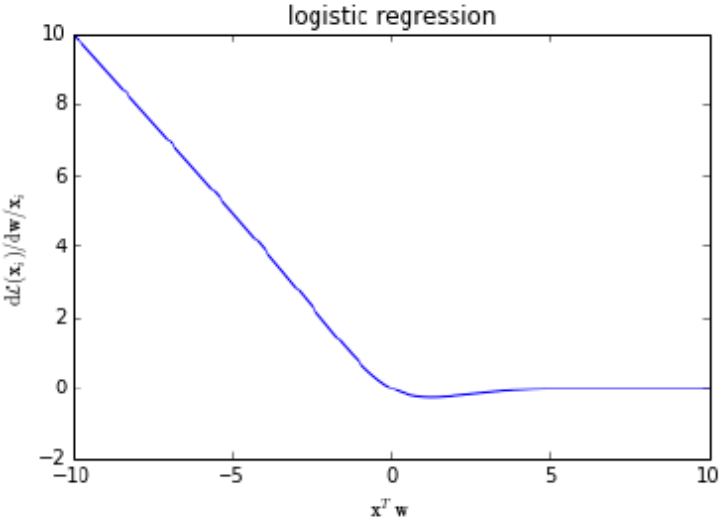
Problem 11 Derive the gradient $\frac{\mathrm{d}\mathcal{L}(\mathbf{x}_i)}{\mathrm{d}\mathbf{w}}$.

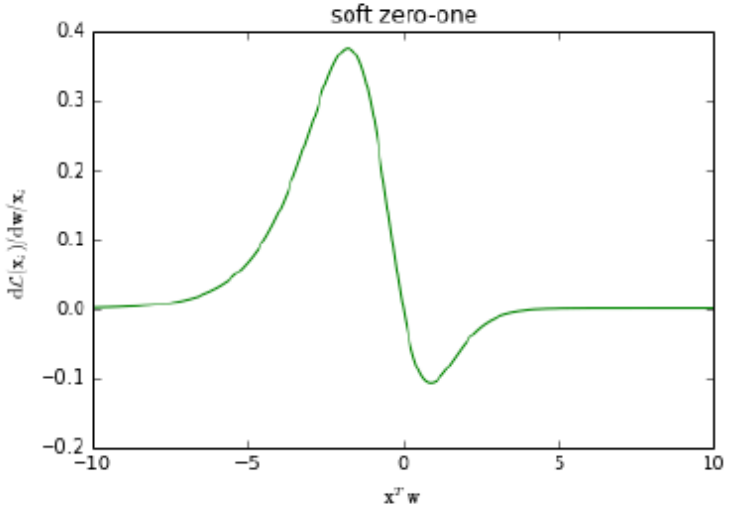
It is difficult to see the difference between the loss of logistic regression and the soft zero-one loss. One difference is the use of the β parameter which controls the steepness of the sigmoid function. However, a steeper or flatter sigmoid can also be attained by increasing or decreasing the magnitude of the weight vector.

The main difference, however, is in the loss function. The logistic regression loss (the negative log likelihood) is unbounded. A sample that is incorrectly labeled and far from the decision boundary has a loss that increases linearly with the distance from the boundary. In contrast, the soft zero-one loss is bounded. This is the reason why it is more robust against outliers because even incorrectly labeled samples that are far from the decision boundary ($|\boldsymbol{x}^T\boldsymbol{w}| \gg 0$) will change the loss by no more than 1.



Looking at the gradient can give us additional insight. Interestingly, the gradient of the soft zero-one loss approaches zero for incorrectly classified samples that are far from the decision boundary. Therefore, they almost do not contribute to the gradient, whereas in logistic regression their impact is linear with the distance from the decision boundary:





$$\frac{\mathrm{d}\mathcal{L}(\mathbf{x}_i)}{\mathrm{d}\boldsymbol{w}} = \frac{\mathrm{d}(\sigma(\beta y_i) - z_i)^2}{\mathrm{d}\boldsymbol{w}}$$
$$= 2\left(\sigma(\beta y_i) - z_i\right)\sigma(\beta y_i)\left(1 - \sigma(\beta y_i)\right)\beta x_i$$