Lecture 6

Discriminative learning of linear decision boundaries

- Classification and regression
- Perceptron learning
- Fisher linear discriminant
- Logistic regression

Recap

Bayesian Decision Theory has shown us how to design an optimal classifier if we know the **prior probabilities** $P(\omega_i)$ and the **class-conditional densities** $p(\mathbf{x} \mid \omega_i)$.

Steps required:

- Estimate $p(\mathbf{x} \mid \omega)$ and $P(\omega)$ from training data.
- Use Bayes' Rule to construct a decision rule
- Decision rule imposes a decision boundary

If have good estimates of $p(\mathbf{x} \mid \omega)$ and $P(\omega)$ then this approach minimizes $P(\text{error})$.

However what about directly learning the decision boundary??

Instead

Decide on a **form** of the **decision boundary**. This function is defined by a set of parameters and a decision is often made via

$$\text{Class}\,(\mathbf{x}) = \text{sgn}(f(\mathbf{x};\mathbf{w}))$$

where

$$\text{sgn}(f(\mathbf{x};\mathbf{w})) = \begin{cases} \omega_1 & \text{if } f(\mathbf{x};\mathbf{w}) > 0 \\ \omega_2 & \text{if } f(\mathbf{x};\mathbf{w}) < 0 \end{cases}$$

Have two separate problems given labelled training data

1. Decide on the form of f

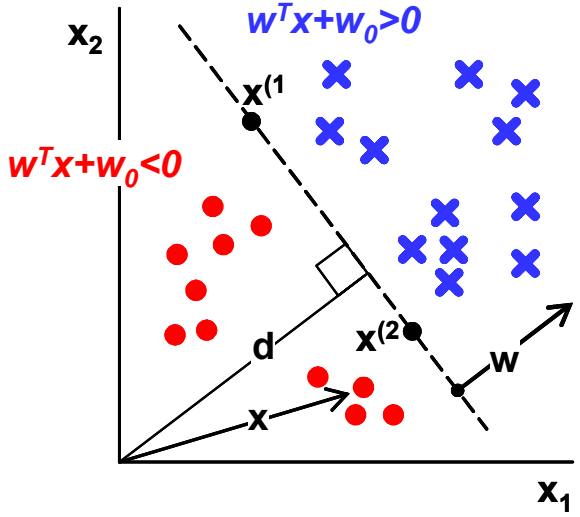
2. Estimate parameters, $\mathbf{w}$, defining $f(\cdot;\mathbf{w})$

Linear discriminant functions

Linear discriminant functions for the 2-class problem are of the form

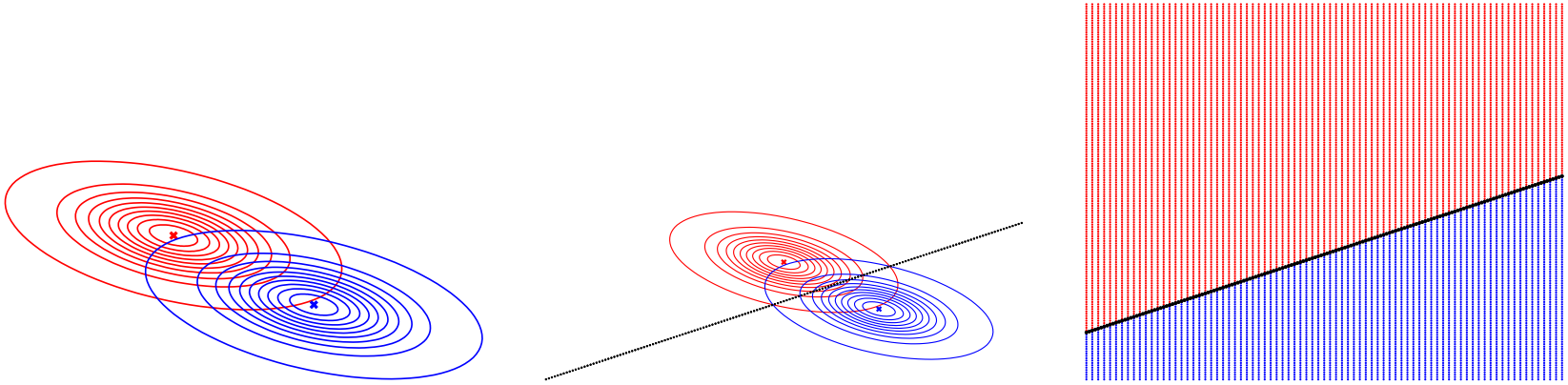
$$f(\mathbf{x};\mathbf{w}) = \mathbf{w}_1^T\mathbf{x} + w_0, \quad \text{Class}\,(\mathbf{x}) = \begin{cases} \omega_1 & \text{if } f(\mathbf{x};\mathbf{w}) > 0 \\ \omega_2 & \text{if } f(\mathbf{x};\mathbf{w}) < 0 \end{cases}$$

where $\mathbf{w}$ is the weight vector and w_0 is the bias.



Linear discriminant functions

Similar discriminant functions were derived where each $p(\mathbf{x} \mid \omega_i)$ is Normally distributed with equal covariance matrices.



class distributions **decision boundary** **partition of space**

In this lecture, **no assumptions**, made about the underlying densities.

The form of the discriminant function will be assumed to be linear.

Learning the discriminant function

If the form of $f(\mathbf{x}; \mathbf{w})$ has been decided upon, in our case a linear function, then

> Given training data $\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_n, y_n)\}$ where each $\mathbf{x}_i \in \mathcal{R}^d$ is a feature vector and the corresponding $y_i \in \{-1, 1\}$ its label/target value, we have to learn/estimate the f's parameters, $\mathbf{w}$, from the labeled training data.

Common approach is:

- define an error function $J(\mathbf{w})$ based on the parameters of interest
- find minimum of $J(\mathbf{w})$ w.r.t. these parameters.

Learning the discriminant function

1. Define an error function $J(\mathbf{w})$ based on the parameters of interest

$$J(\mathbf{w}) = \sum_{i=1}^{n} L(y_i, f(\mathbf{x}_i; \mathbf{w}))$$

   where $L : \{-1, +1\} \times \mathcal{R} \rightarrow \mathcal{R}^+$ is a loss function, which measures how well $f(\mathbf{x}_i; \mathbf{w})$ predicts the label of $\mathbf{x}_i$.

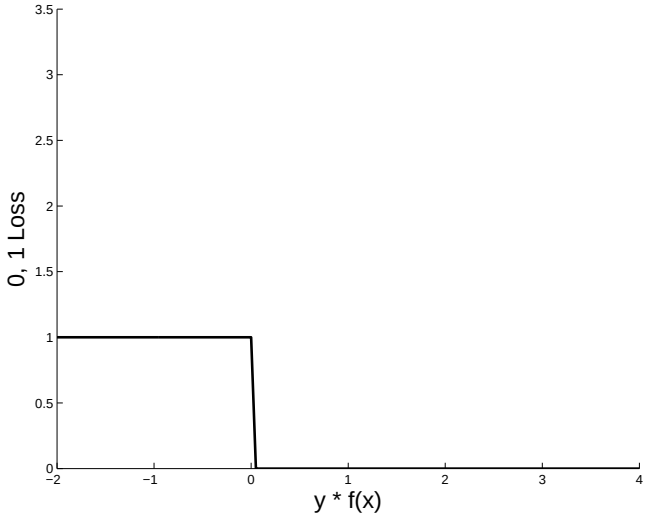
2. Find minimum of $J(\mathbf{w})$ w.r.t. these parameters.

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} J(\mathbf{w})$$

3. Then the classifier is defined by the sign of $f(\mathbf{x}; \mathbf{w}^*)$.

Common loss functions

0, 1 Loss function



$$L(y, f(\mathbf{x}; \mathbf{w})) = \begin{cases} 0 & \text{if } y = \text{sgn}(f(\mathbf{x}; \mathbf{w})) \\ 1 & \text{if } y \neq \text{sgn}(f(\mathbf{x}; \mathbf{w})) \end{cases}$$

$$= \begin{cases} 0 & \text{if } y \times \text{sgn}(f(\mathbf{x}; \mathbf{w})) > 0 \\ 1 & \text{if } y \times \text{sgn}(f(\mathbf{x}; \mathbf{w})) < 0 \end{cases}$$

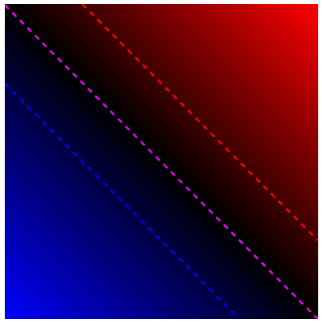
0, 1 Loss function

Consider the case when $f(\mathbf{x}; \mathbf{w})$ is a hyper-plane

$$f(\mathbf{x}; \mathbf{w}) = \mathbf{w}_1^T \mathbf{x} + w_0 \quad \text{with } \mathbf{w} = (\mathbf{w}_1, w_0)$$

Two dimensional example of a linear decision function

blue $\Longrightarrow$ negative values
red $\Longrightarrow$ positive values
black $\Longrightarrow$ values close to 0.

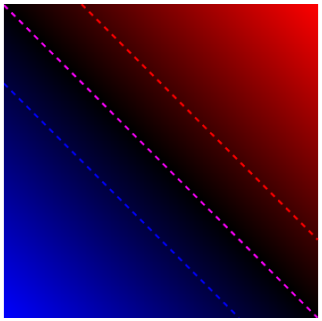


$f(\mathbf{x}; \mathbf{w}) = 0$
$f(\mathbf{x}; \mathbf{w}) = 1$
$f(\mathbf{x}; \mathbf{w}) = -1$

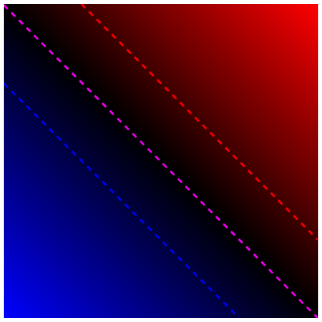
$f(\mathbf{x}; \mathbf{w}) = \mathbf{w}_1^T \mathbf{x} + w_0$

0, 1 Loss function

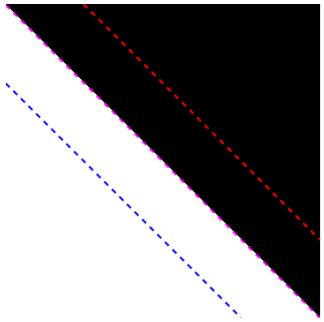
Our linear function used for classification



All points belong to class ω_1



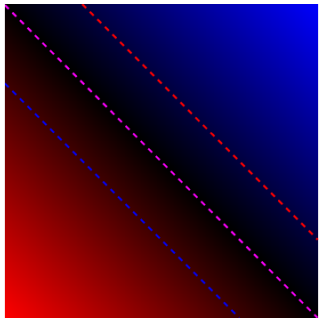
$1 \times f(\mathbf{x}; \mathbf{w})$



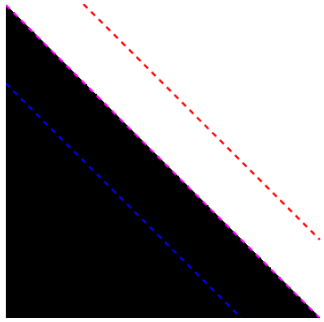
$L(1, f(\mathbf{x}; \mathbf{w}))$

Misclassified pts have loss 1, otherwise loss 0

All points belong to class ω_2



$-1 \times f(\mathbf{x}; \mathbf{w})$

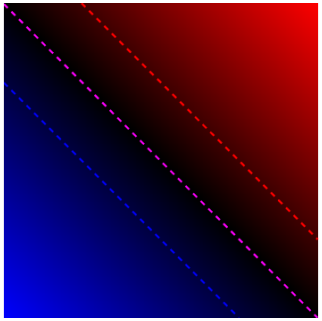


$L(-1, f(\mathbf{x}; \mathbf{w}))$

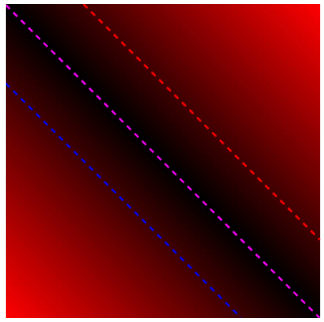
Misclassified pts have loss 1, otherwise loss 0

0, 1 Loss function

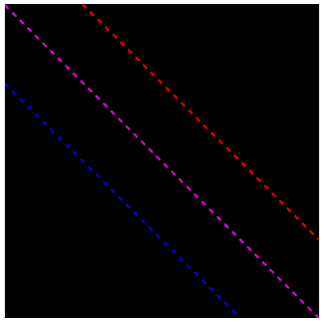
Our linear function used for classification



If all points in plane correctly classified



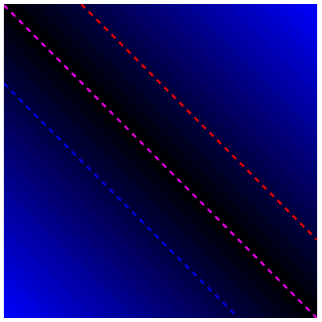
$y \times f(\mathbf{x}; \mathbf{w})$



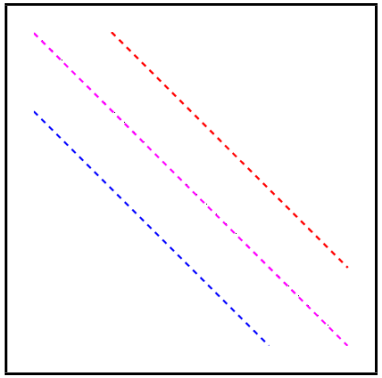
$L(y, f(\mathbf{x}; \mathbf{w}))$

Loss is zero for each pt

If all points in plane incorrectly classified



$y \times f(\mathbf{x}; \mathbf{w})$

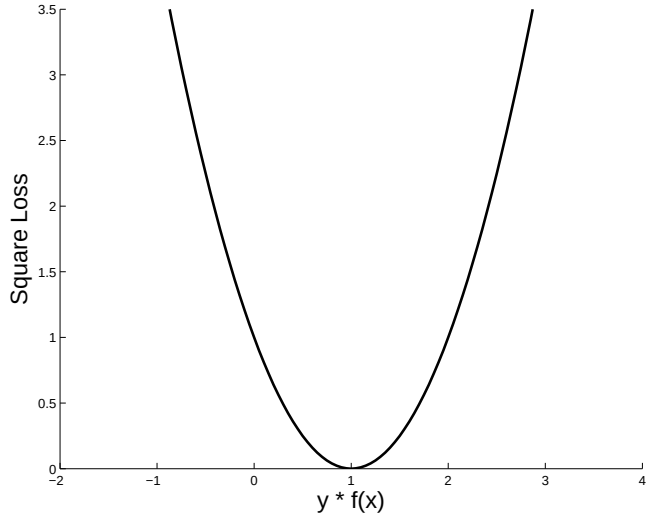


$L(y, f(\mathbf{x}; \mathbf{w}))$

Loss is one for each pt

Common loss functions

Squared Error loss

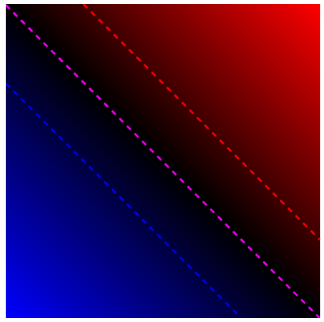


$$L(y, f(\mathbf{x}; \mathbf{w})) = (y - f(\mathbf{x}; \mathbf{w}))^2$$

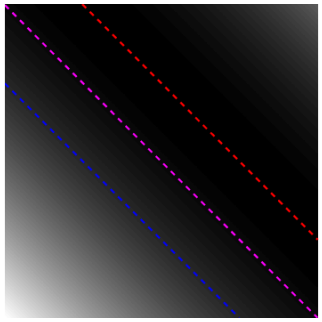
$$= (1 - yf(\mathbf{x}; \mathbf{w}))^2$$

Squared error loss

All points belong to class ω_1

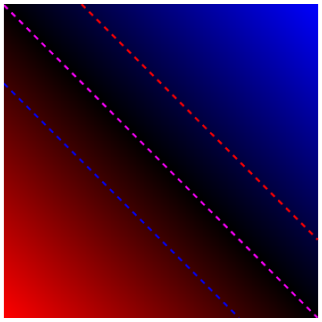


$1 \times f(\mathbf{x}; \mathbf{w})$

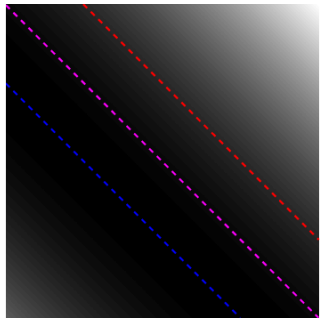


$L(1, f(\mathbf{x}; \mathbf{w}))$

All points belong to class ω_2



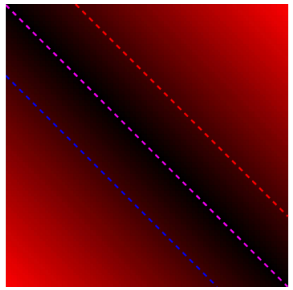
$-1 \times f(\mathbf{x}; \mathbf{w})$



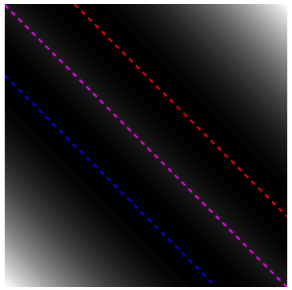
$L(-1, f(\mathbf{x}; \mathbf{w}))$

Squared error loss

All points in plane correctly classified

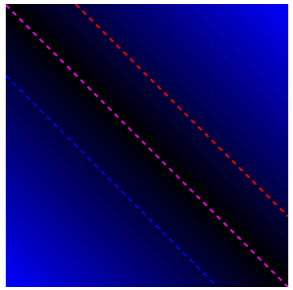


$y \times f(\mathbf{x}; \mathbf{w})$

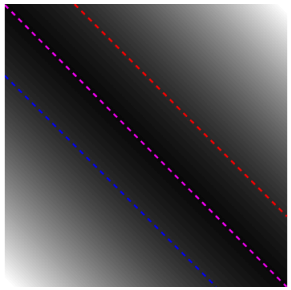


$L(y, f(\mathbf{x}; \mathbf{w}))$

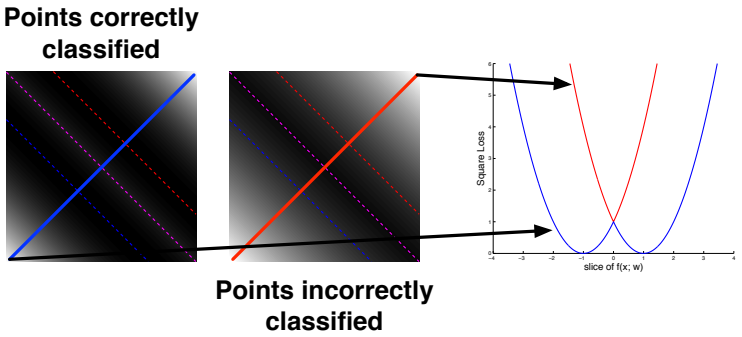
All points in plane incorrectly classified



$y \times f(\mathbf{x}; \mathbf{w})$



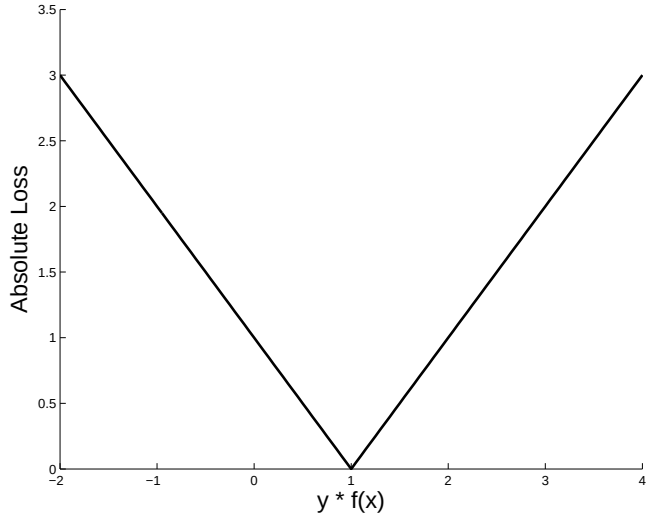
$L(y, f(\mathbf{x}; \mathbf{w}))$



1D cross section of the 2D loss function

Common loss functions

Absolute loss

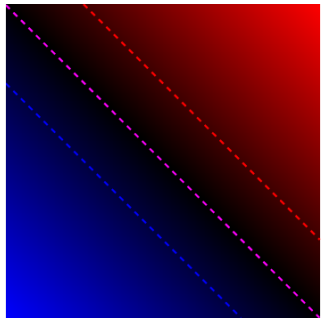


$$L(y, f(\mathbf{x}; \mathbf{w})) = |y - f(\mathbf{x}; \mathbf{w})|$$

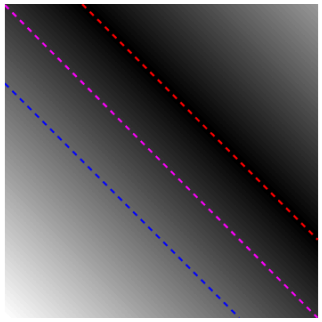
$$= |1 - yf(\mathbf{x}; \mathbf{w})|$$

Absolute loss

All points belong to class ω_1

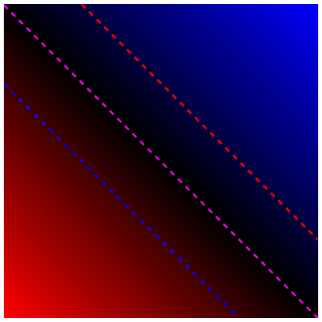


$1 \times f(\mathbf{x}; \mathbf{w})$

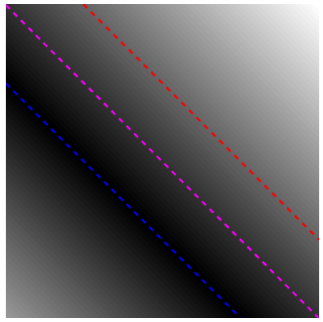


$L(1, f(\mathbf{x}; \mathbf{w}))$

All points belong to class ω_2



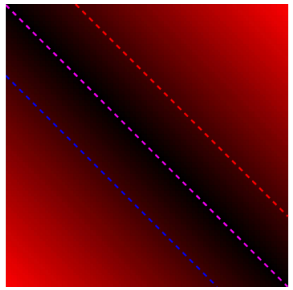
$-1 \times f(\mathbf{x}; \mathbf{w})$



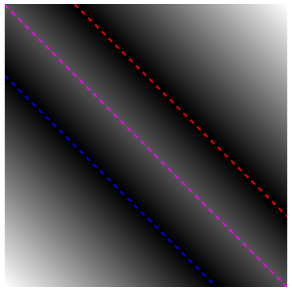
$L(-1, f(\mathbf{x}; \mathbf{w}))$

Absolute loss

All points in plane correctly classified

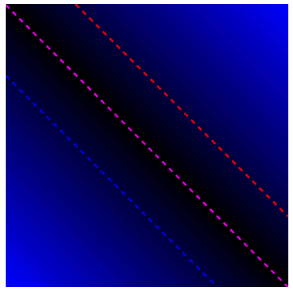


$y \times f(\mathbf{x}; \mathbf{w})$

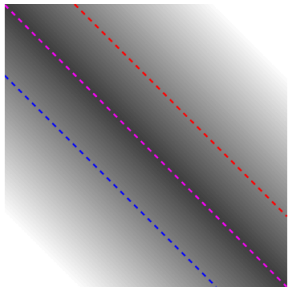


$L(y, f(\mathbf{x}; \mathbf{w}))$

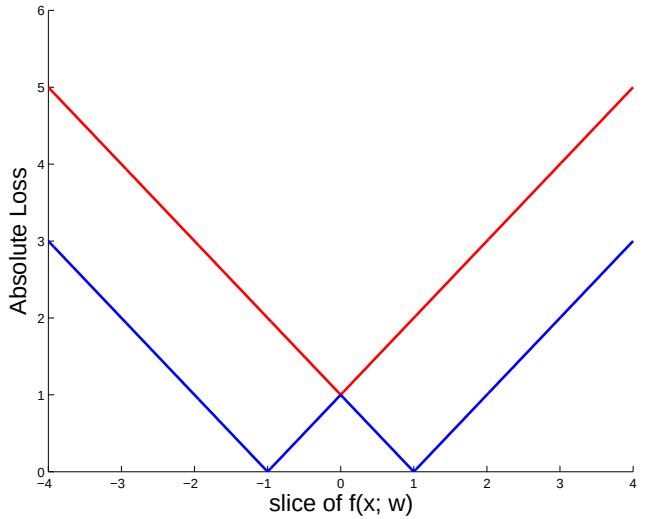
All points in plane incorrectly classified



$y \times f(\mathbf{x}; \mathbf{w})$



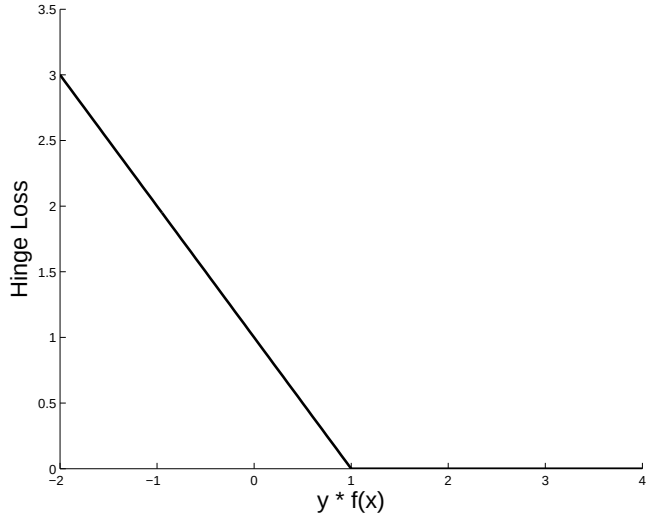
$L(y, f(\mathbf{x}; \mathbf{w}))$



1D cross section of the 2D loss function

Common loss functions

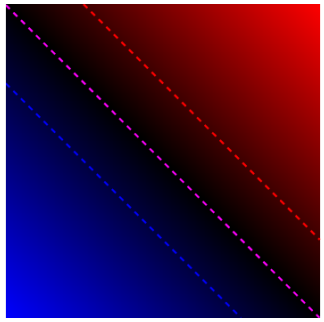
Hinge loss



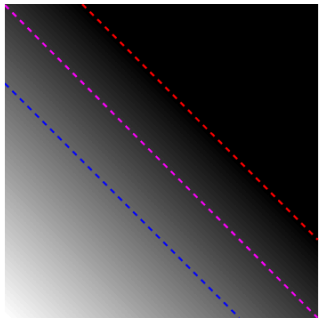
$$L(y, f(\mathbf{x}; \mathbf{w})) = \max(1 - yf(\mathbf{x}; \mathbf{w}), 0)$$

Hinge loss

All points belong to class ω_1

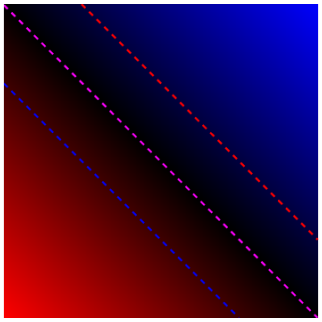


$1 * f(\mathbf{x}; \mathbf{w})$

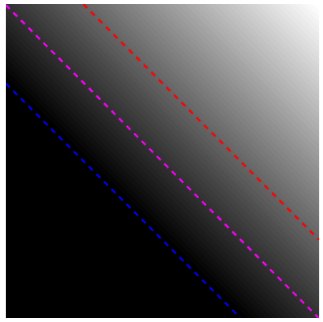


$L(1, f(\mathbf{x}; \mathbf{w}))$

All points belong to class ω_2



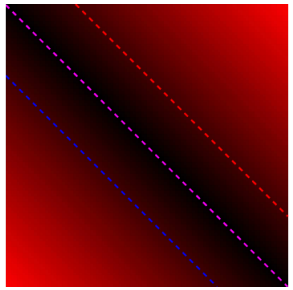
$-1 * f(\mathbf{x}; \mathbf{w})$



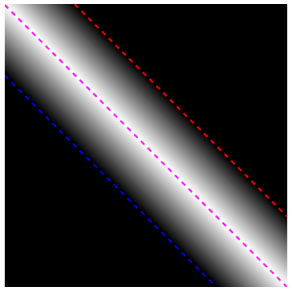
$L(-1, f(\mathbf{x}; \mathbf{w}))$

Hinge loss

All points in plane correctly classified

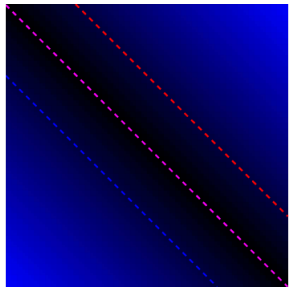


$y * f(\mathbf{x}; \mathbf{w})$

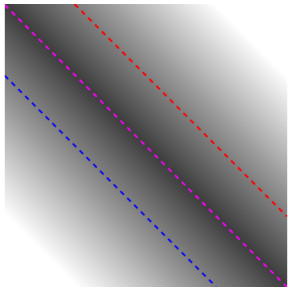


$L(y, f(\mathbf{x}; \mathbf{w}))$

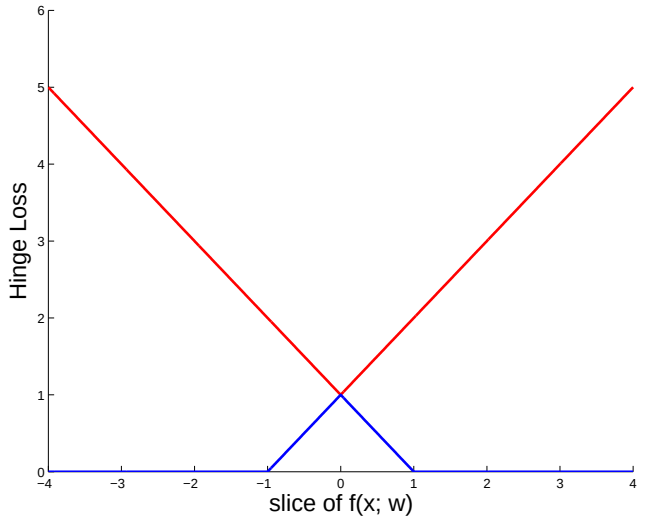
All points in plane incorrectly classified



$y * f(\mathbf{x}; \mathbf{w})$

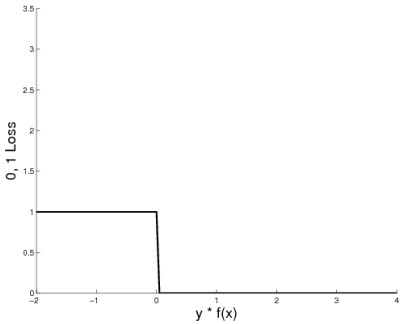


$L(y, f(\mathbf{x}; \mathbf{w}))$



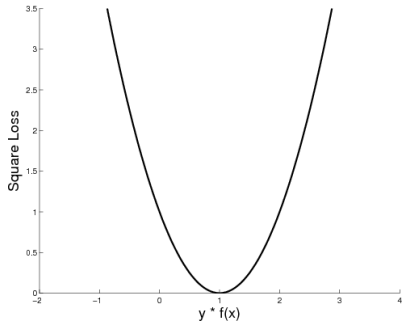
1D cross section of the 2D loss function

Summary of common loss functions



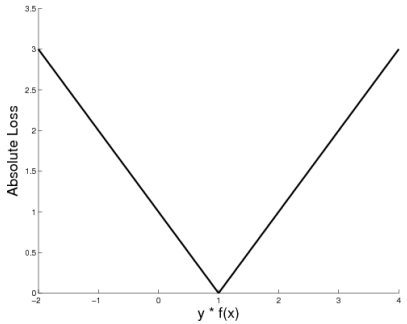
0, 1 Loss

$\mathsf{Ind}(y \neq \mathrm{sgn}(f(\mathbf{x};\mathbf{w})))$



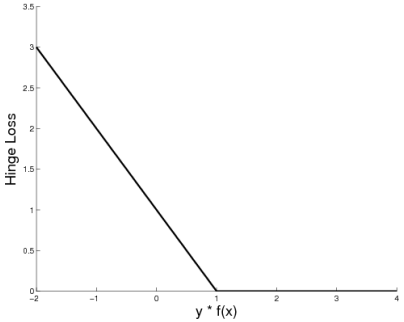
Square Loss

$(1 - yf(\mathbf{x};\mathbf{w}))^2$



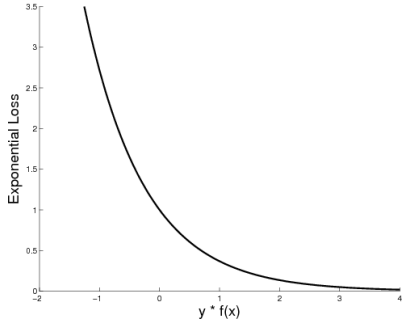
Absolute Loss

$|1 - yf(\mathbf{x};\mathbf{w})|$



Hinge Loss

$\max(1 - yf(\mathbf{x};\mathbf{w}), 0)$



Exponential Loss

$\exp\left(-yf(\mathbf{x};\mathbf{w})\right)$

Unconstrained optimization

Recall that the minimum of a function $J(\mathbf{x})$ is defined by the zeros of the gradient

$$\mathbf{x}^* = \arg\min_{\forall \mathbf{x}} J(\mathbf{x}) \implies \nabla_{\mathbf{x}} J(\mathbf{x}) = \mathbf{0}$$

However

- Only in very special cases does this minimization function have a closed form solution
- In some other cases, a closed form solution may exist, but is numerically ill-posed or impractical (e.g., memory requirements).

Technical interlude: Iterative Optimization

$$\mathbf{w}^* = \arg\min_{\forall \mathbf{w}} J(\mathbf{w})$$

Common approach to solving such unconstrained optimization problem is iterative non-linear optimization.

Start with an estimate $\mathbf{w}^{(0)}$ and try to improve it by finding successive new estimates $\mathbf{w}^{(1)}, \mathbf{w}^{(2)}, \mathbf{w}^{(3)}, \ldots$ such that $J(\mathbf{w}^{(1)}) \leq J(\mathbf{w}^{(2)}) \leq J(\mathbf{w}^{(3)}) \leq \cdots$, until convergence.

The search at each iteration to find a better estimate is performed locally around the current estimate.

Such iterative approaches will find a local minima.

Technical interlude: Iterative Optimization

These non-linear iterative methods alternate between these two steps:

Decide search direction Choose a search direction based on the local properties of the cost function.

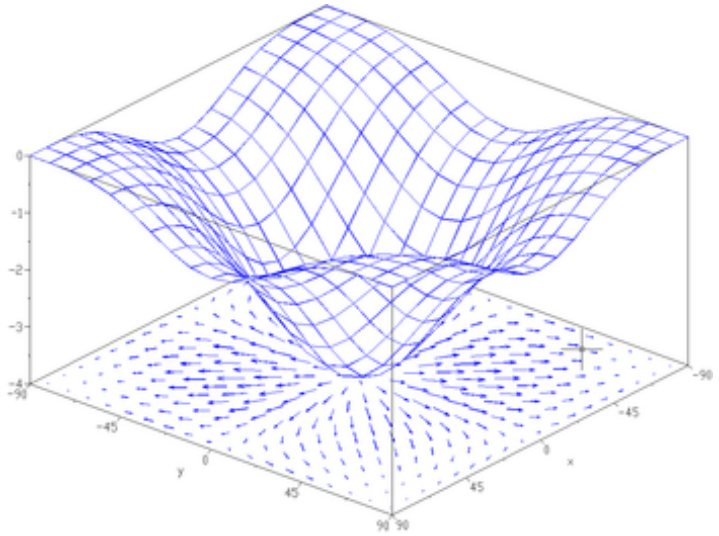
Line Search Perform an intensive search to find the minimum along the chosen direction.

Technical interlude: the gradient

Choosing a search direction (simplest approach)

The gradient is defined as:

$$\nabla_{\mathbf{x}} J(\mathbf{x}) \equiv \frac{\partial J(\mathbf{x})}{\partial \mathbf{x}} = \begin{pmatrix} \frac{\partial J(\mathbf{x})}{\partial x_1} \\ \frac{\partial J(\mathbf{x})}{\partial x_2} \\ \vdots \\ \frac{\partial J(\mathbf{x})}{\partial x_d} \end{pmatrix}$$

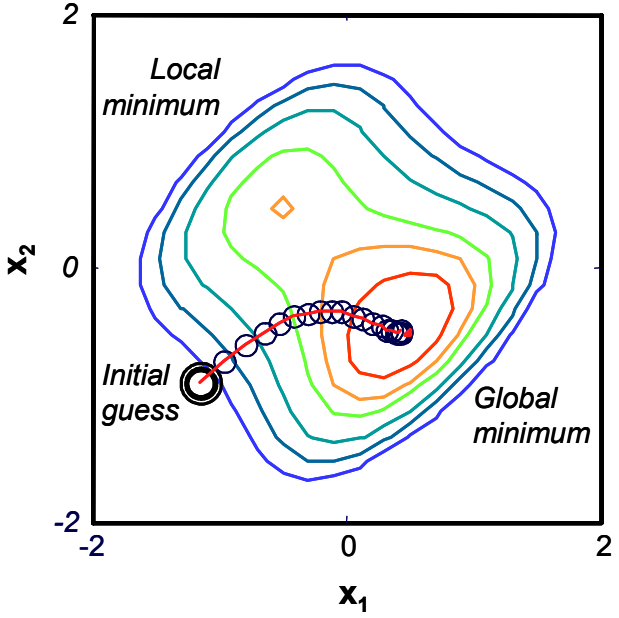


The gradient points in the direction of the greatest increase of $J(\mathbf{x})$.

Technical interlude: Gradient descent

Gradient descent is general method for function minimization

Gradient descent finds the minimum in an iterative fashion by moving in the direction of steepest descent.



Gradient Descent Minimization

1. Start with an arbitrary solution $\mathbf{x}^{(0)}$.
2. Compute the gradient $\nabla_{\mathbf{x}} J(\mathbf{x}^{(k)})$.
3. Move in the direction of steepest descent:

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - \eta^{(k)} \nabla_{\mathbf{x}} J(\mathbf{x}^{(k)}).$$

   where $\eta^{(k)}$ is the step size.
4. Go to 2 (until convergence).

Line Search: Armijo rule

Still need to figure out a way to set the step size to guarantee convergence.

$$\mathbf{x} \leftarrow \mathbf{x}^{(k)} + \eta^{(k)} \nabla_{\mathbf{x}} J(\mathbf{x}^{(k)})$$

Armijo's Rule

Set the largest possible step size η_0 and $\beta \in (0, 1)$.

Choose $\eta^{(k)}$ be the largest $\eta \in \{\eta_0, \beta\eta_0, \beta^2\eta_0, \beta^3\eta_0, \ldots\}$ such that

$$J(\mathbf{x}^{(k)}) - J\left(\mathbf{x}^{(k)} + \eta \nabla_{\mathbf{x}} J(\mathbf{x}^{(k)})\right) \geq \frac{1}{2} \eta \left\| \nabla_{\mathbf{x}} J(\mathbf{x}^{(k)}) \right\|^2$$

Armijo's rule is guaranteed to converge to a (local) maximum under certain technical assumptions.

Some derivatives of vectors

$$\frac{\partial\, \mathbf{x}^T \mathbf{a}}{\partial \mathbf{x}} = \frac{\partial\, \mathbf{a}^T \mathbf{x}}{\partial \mathbf{x}} = \mathbf{a}$$

$$\frac{\partial\, \mathbf{x}^T B \mathbf{x}}{\partial \mathbf{x}} = (B + B^T)\mathbf{x}$$

Thus if $J(\mathbf{x}) = \mathbf{x}^T A\, \mathbf{x} + \mathbf{a}^T \mathbf{x}$ then (you'll be doing this frequently)

$$\nabla_{\mathbf{x}} J(\mathbf{x}) = (A + A^T)\, \mathbf{x} + \mathbf{a}$$

Finding the decision boundary: Attempt 1
classification via regression

Have training data

point in $\mathcal{R}^d$	actual label	target label in learning
$\mathbf{x}_1$	ω_2	$y_1 = 1$
$\mathbf{x}_2$	ω_2	$y_2 = 1$
$\mathbf{x}_3$	ω_1	$y_3 = -1$
$\mathbf{x}_4$	ω_1	$y_4 = -1$
$\vdots$	$\vdots$	$\vdots$

Ignoring the fact that the target output y is binary rather than a continuous variable. Approximate the output with a linear regression

function

$$f(\mathbf{x}; \mathbf{w}) = w_0 + w_1 x_1 + w_2 x_2 + \cdots + w_d x_d = w_0 + \mathbf{w}_1^T \mathbf{x}$$

Learn $\mathbf{w}$ from the training data by finding the $\mathbf{w}$ that minimizes:

$$J(\mathbf{w}) = \frac{1}{2} \sum_{i=1}^{n} \underbrace{\left(f(\mathbf{x}_i; \mathbf{w}) - y_i\right)^2}_{\text{Squared error loss}}$$

such a J is known as the sum-of-squares error function.

The $\mathbf{w}^*$ that minimizes $J(\mathbf{w})$ is known as the **Minimum Squared Error** solution.

This minimum is found as follows:

Pseudo-Inverse solution

Substitute in the expression for $f(\mathbf{x}_i; \mathbf{w})$, then error function becomes

$$
\begin{aligned}
J(\mathbf{w}) &= \frac{1}{2}\sum_{i=1}^{n}\left(f(\mathbf{x}_i;\mathbf{w}) - y_i\right)^2 \\
&= \frac{1}{2}\sum_{i=1}^{n}\left(w_0 + \mathbf{w}_1^T\mathbf{x}_i - y_i\right)^2 \\
&= \frac{1}{2}\sum_{i=1}^{n}\left(\mathbf{w}^T\mathbf{x}_i' - y_i\right)^2, \quad \mathbf{x}_i' = (1, \mathbf{x}_i^T)^T
\end{aligned}
$$

Writing in matrix notation this becomes

$$J(\mathbf{w}) = \frac{1}{2}\|X\mathbf{w} - \mathbf{y}\|^2 = \frac{1}{2}(X\mathbf{w} - \mathbf{y})^T(X\mathbf{w} - \mathbf{y})$$
$$= \frac{1}{2}\left(\mathbf{w}^T X^T X\mathbf{w} - 2\mathbf{y}^T X\mathbf{w} + \mathbf{y}^T\mathbf{y}\right)$$

where

$$\mathbf{y} = (y_1, \ldots, y_n)^T$$
$$\mathbf{w} = (w_0, \ldots, w_d)^T$$
$$X = \begin{pmatrix} 1 & \mathbf{x}_1^T \\ 1 & \mathbf{x}_2^T \\ \vdots & \vdots \\ 1 & \mathbf{x}_n^T \end{pmatrix}$$

Pseudo-Inverse solution

The gradient of $J(\mathbf{w})$ wrt $\mathbf{w}$:

$$\nabla_{\mathbf{w}} J(\mathbf{w}) = X^T X \mathbf{w} - X^T \mathbf{y}$$

Setting this equal to zero yields $X^T X \mathbf{w} = X^T \mathbf{y}$ and

$$\mathbf{w} = X^\dagger \mathbf{y}$$

where

$$X^\dagger \equiv \left(X^T X\right)^{-1} X^T$$

$X^\dagger$ is called the **pseudo-inverse** of X. Note that $X^\dagger X = I$ but in

general $XX^\dagger \neq I$. If X^TX is singular, there is no unique solution to $X^TX\mathbf{w} = X^T\mathbf{y}$.

However, if instead one uses

$$X^{\text{ridge}} \equiv \left(X^TX + \epsilon I\right)^{-1} X^T$$

where $\epsilon > 0$ is small then a solution for $\mathbf{w}$ will be found even if X^TX is singular and will be close to the true solution when $(X^TX)^{-1}$ exists. This is know as **ridge regression**.

Gradient descent solution

The error function $J(\mathbf{w})$ could also be minimized wrt $\mathbf{w}$ by using a **gradient descent procedure**.

Why

- This avoids the numerical problems that arise when $X^T X$ is (nearly) singular.
- In addition, it also avoids the need for working with large matrices

How

1. Begin with an initial guess $\mathbf{w}^{(0)}$ for $\mathbf{w}$.

2. Update the weight vector by moving a small distance in $\mathbf{w}$-space in the direction $-\nabla_{\mathbf{w}} J$.

Solution

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{\tau} + \eta^{(\tau)} X^T (\mathbf{y} - X\mathbf{w}^{(\tau)})$$

It can be shown that if $\eta^{(\tau)} = \eta_0/\tau$, where η_0 is any positive constant, this rule generates a sequence of vectors that converge to a solution to $X^T(X\mathbf{w} - \mathbf{y}) = \mathbf{0}$ irrespective of whether $X^T X$ is singular or not.

Sequential gradient descent solution

The storage requirements of the previous algorithm can be reduced by considering each training sample sequentially

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} + \eta^{(\tau)}(y_i - \mathbf{x}_i^T \mathbf{w}^{(\tau)})\mathbf{x}_i$$

This is known as the **Widrow-Hoff**, **least-mean-squares** (LMS) or **delta rule** [Mitchell, 1997].

Classification via regression

The resulting regression function

$$f(\mathbf{x}; \mathbf{w}^*) = w_0^* + \mathbf{w}_1^{*T}\mathbf{x}$$

is then used to classify any new (test) example $\mathbf{x}$ according to

$$\text{Class}\,(\mathbf{x}) = \text{sgn}(f(\mathbf{x}; \mathbf{w}^*))$$

$f(\mathbf{x}; \mathbf{w}^*) = 0$ therefore defines a **linear decision boundary** that partitions the input space into two class specific regions.

Is this such a good thing to do?

Extension of model: Generalized linear discriminants

Can transform the input vector $\mathbf{x}$ using a set of m pre-defined non-linear functions ϕ_j. This permits a much larger range of possible decision boundaries.

$$f(\mathbf{x}; \mathbf{w}) = \sum_{j=1}^{m} w_j \phi_j(\mathbf{x}) + w_0$$

The bias can be absorbed by defining an extra basis function $\phi_0 = 1$, so that

$$f(\mathbf{x}; \mathbf{w}) = \sum_{j=0}^{m} w_j \phi_j(\mathbf{x}) = \mathbf{w}^T \boldsymbol{\phi}$$

$f(\mathbf{x}; \mathbf{w})$ represents a much larger class of function. This, of course, depends on the choice of ϕ_j's.

Generalized linear discriminants: An example

Say $\mathbf{x} \in \mathcal{R}^2$ and

$$\phi_1(\mathbf{x}) = x_1^2,\ \phi_2(\mathbf{x}) = x_1,\ \phi_3(\mathbf{x}) = x_2^2,\ \phi_4(\mathbf{x}) = x_2,\ \phi_5(\mathbf{x}) = x_1\, x_2$$

Then the decision boundary can take the form of bivariate quadratic curve (ie circle, parabola, ellipse, etc..)

Or indeed ϕ_j's could be the output of a set of linear filters to an image applied at particular locations.

The MSE solution for $\mathbf{w}$, in this more general case, is the solution to

$$\Phi^T \Phi \mathbf{w} = \Phi^T \mathbf{y} \quad \text{where} \quad \Phi = \begin{pmatrix} 1 & \boldsymbol{\phi}_1^T \\ 1 & \boldsymbol{\phi}_2^T \\ \vdots & \vdots \\ 1 & \boldsymbol{\phi}_n^T \end{pmatrix} \quad \text{and} \quad \boldsymbol{\phi}_i = \begin{pmatrix} \phi_1(\mathbf{x}_i) \\ \phi_2(\mathbf{x}_i) \\ \vdots \\ \phi_m(\mathbf{x}_i) \end{pmatrix}$$

Attempt 2: Perceptron learning

Model of a Neuron

The following has been used as a simple mathematical model for the behavior of a single neuron in a biological nervous system.

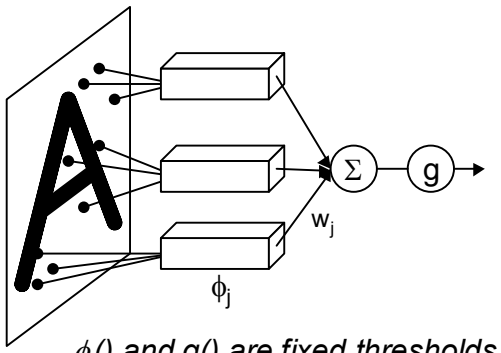
$$y = g(\mathbf{w}_1^T \mathbf{x} + w_0), \qquad \text{with} \quad g(a) = \begin{cases} -1 & \text{where } a < 0 \\ 1 & \text{where } a \geq 0 \end{cases}$$

- x_i - the level of activity of neurons in the system connected to this neuron.
- w_i - the strength of the interconnections (*synapses*).
- w_0 - the threshold for the neuron to *fire*.

Model introduced by McCulloch and Pitts (1943).
Networks of these thresholded units were studied by Rosenblatt (1962) under the name *perceptrons*.

The Perceptron

Single-layer networks, with threshold activation functions were studied by Rosenblatt (1962) who called them perceptrons. These networks were applied to classification problems, in which the inputs were binary images of characters and simple shapes. The ϕ_j's were fixed processing elements that were typically a threshold activation function based on pixel values from the binary image. Once again ϕ_0 is set to 1 to absorb the bias w_0.



ϕ_j() and g() are fixed thresholds
w_j are variable weights

$$f(\mathbf{x};\mathbf{w}) = g\left(\sum_{j=0}^{m} w_j \phi_j(\mathbf{x})\right) = g(\mathbf{w}^T \boldsymbol{\phi})$$

with $g(a) = \begin{cases} -1 & \text{if } a < 0 \\ 1 & \text{if } a \geq 0 \end{cases}$

The Perceptron criterion

Once again have to estimate $\mathbf{w}$. Each training example $\mathbf{x}_i$ has an associated target value y_i with $y_i \in \{-1, 1\}$ depending if $\mathbf{x}_i$ belongs to class ω_1 or ω_2. Each training example $\mathbf{x}_i$ generates a corresponding vector of activations $\boldsymbol{\phi}_i$. Want $\mathbf{w}^T\boldsymbol{\phi}_i > 0$ for vectors belonging to class ω_1 and $\mathbf{w}^T\boldsymbol{\phi}_i < 0$ otherwise. Thus want

$$y_i(\mathbf{w}^T\boldsymbol{\phi}_i) > 0 \text{ for all the training examples.}$$

Define the error function, known as the *Perceptron criterion* function

$$J_{\mathrm{P}}(\mathbf{w}) = -\sum_{\boldsymbol{\phi}_i \in \mathcal{M}} y_i \, \mathbf{w}^T\boldsymbol{\phi}_i$$

where $\mathcal{M} = \{\boldsymbol{\phi}_i \mid \boldsymbol{\phi}_i$ misclassified by the current value of $\mathbf{w}\}$.

Perceptron learning

To find the minimum, use gradient descent

- The gradient is defined by

$$\nabla_{\mathbf{w}} J_{\mathrm{P}}(\mathbf{w}) = -\sum_{\boldsymbol{\phi}_i \in \mathcal{M}} y_i \, \boldsymbol{\phi}_i$$

- And the gradient descent update rule, known in this case as the **Perceptron Rule**, becomes

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} + \eta \sum_{\boldsymbol{\phi}_i \in \mathcal{M}} \boldsymbol{\phi}_i y_i$$

  This is known as the **perceptron batch update rule**.

- The weight vector may also be updated in an *on-line* fashion, that is, after the presentation of each individual example.

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} + \eta \boldsymbol{\phi}_i y_i$$

where $\boldsymbol{\phi}_i$ is an example that has been misclassified by $\mathbf{w}^{(\tau)}$.

Perceptron learning: properties

If classes are linearly separable with respect to the ϕ_j's, the perceptron rule is guaranteed to converge to a valid solution.

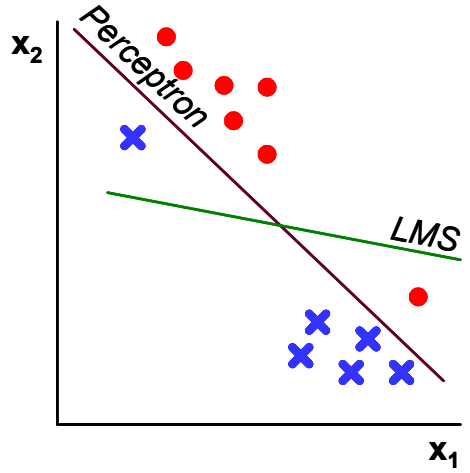
- Some version of the perceptron rule use a variable learning rate $\eta^{(\tau)}$ - In this case, convergence is guaranteed only under certain conditions

However, if the two classes are not linearly separable, the perceptron rule will not converge.

- Since no weight vector $\mathbf{w}$ can correctly classify every sample in a non-separable dataset, the corrections in the perceptron rule will never cease.
- One ad-hoc solution to this problem is to enforce convergence by using variable learning rates $\eta^{(\tau)} \to 0$ as $\tau \to \infty$.

Summary: Perceptron vs. MSE procedures

- **Perceptron rule** The perceptron rule always finds a solution if the classes are linearly separable, but does not converge if the classes are non-separable.



In this particular example $\phi_j(\mathbf{x}) = x_j$.

- **MSE Criterion** The MSE solution has guaranteed convergence, but it **may not find** a **separating hyperplane** if classes **are linearly separable**

MSE tries to minimize the sum of the squares of the distances of the projected training data on the separating hyperplane to the class labels, as opposed to finding a separating hyperplane.

Attempt 3: Projections and classifications

A linear function

$$f(\mathbf{x}; \mathbf{w}) = w_0 + \mathbf{w}_1^T \mathbf{x}$$

projects each point $\mathbf{x} = (x_1, x_2, \ldots, x_d)^T$ to a line parallel to $\mathbf{w}_1$.

point in $\mathcal{R}^d$	class label	projected point in $\mathcal{R}$
$\mathbf{x}_1$	y_1	$z_1 = w_0 + \mathbf{w}_1^T \mathbf{x}_1$
$\mathbf{x}_2$	y_2	$z_2 = w_0 + \mathbf{w}_1^T \mathbf{x}_2$
$\vdots$	$\vdots$	$\vdots$
$\mathbf{x}_n$	y_n	$z_n = w_0 + \mathbf{w}_1^T \mathbf{x}_n$

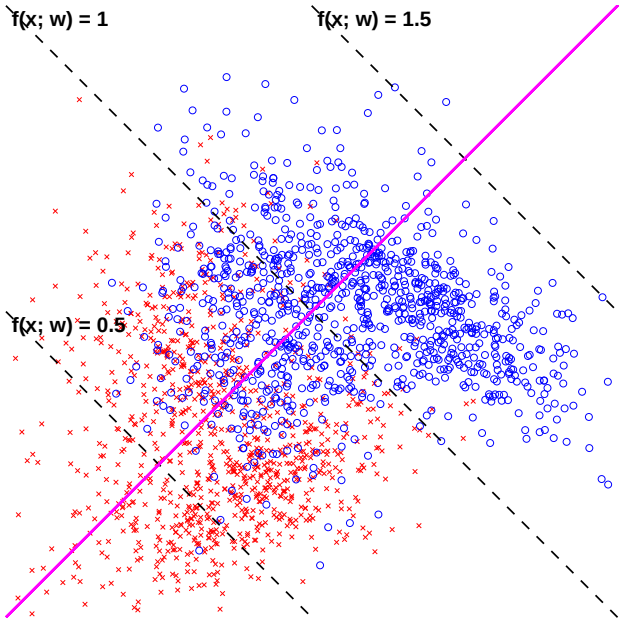
Can study how well the projected points $\{z_1, \ldots, z_n\}$, viewed as a function of $\mathbf{w}_1$, are separated across the classes.

Projections

A linear function

$$f(\mathbf{x}; \mathbf{w}) = w_0 + \mathbf{w}_1^T \mathbf{x}$$

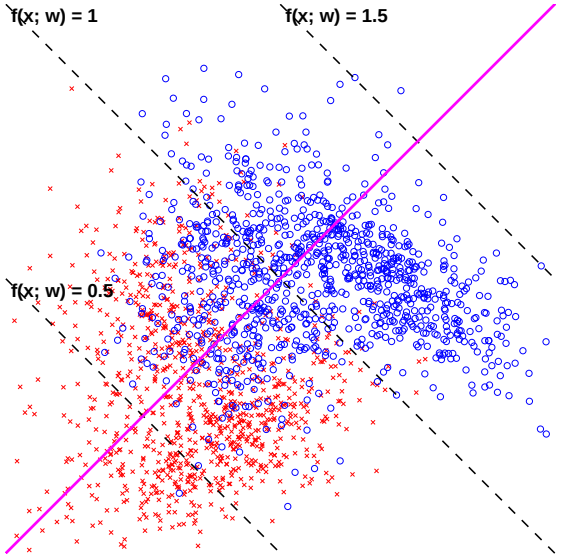
projects each point $\mathbf{x} = (x_1, x_2, \ldots, x_d)^T$ to a line parallel to $\mathbf{w}_1$.

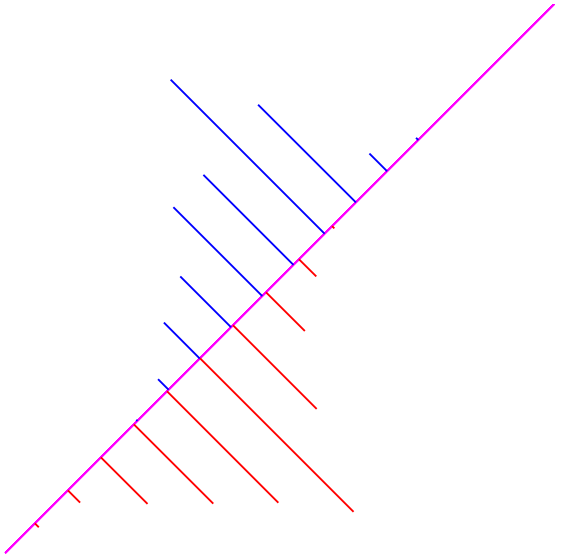


Can study how well the projected points $\{z_1, \ldots, z_n\}$, viewed as a function of $\mathbf{w}_1$, are separated across the classes.

Projections

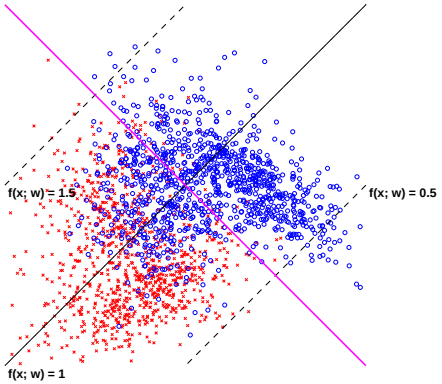
Data points and the projection direction

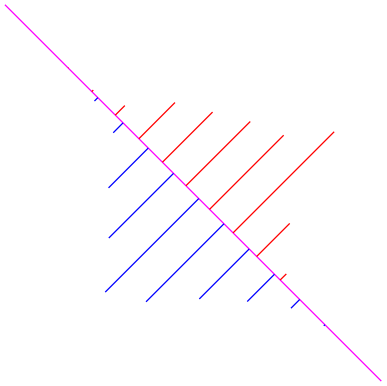


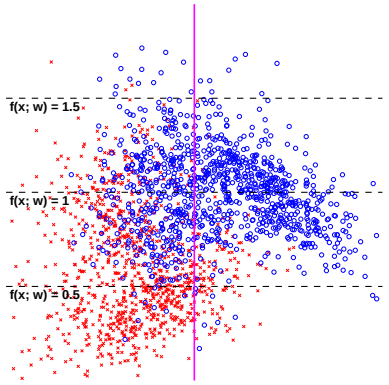


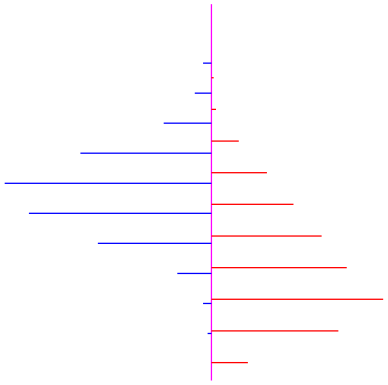
histograms of z_i's for the two classes

Projection and classification









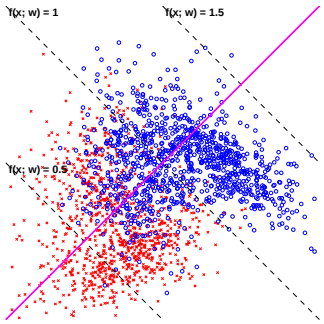
projection direction **histograms of z_i's**

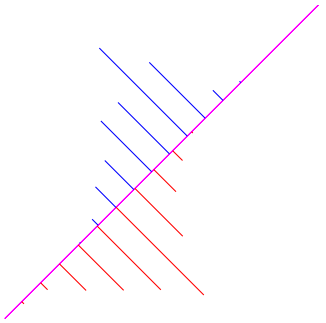
Varying $\mathbf{w}_1$ varies the separation between the projected points.

Optimizing the projection

Would like to find $\mathbf{w}_1$ that somehow maximizes the separation of the projections points across classes.

Data points and the projection direction





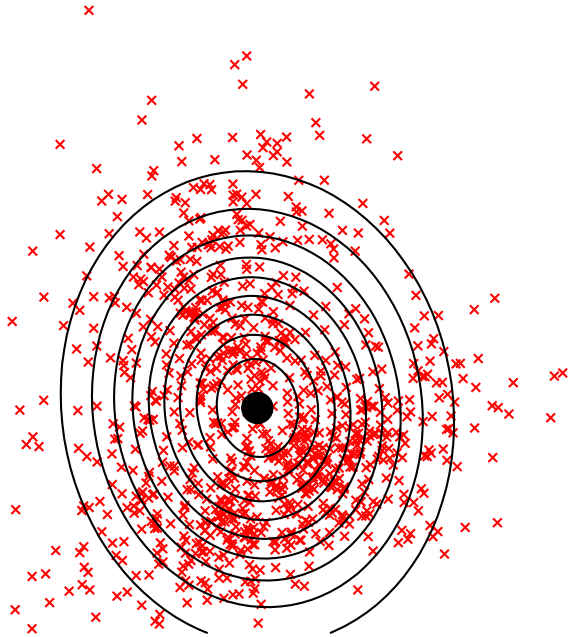
histograms of z_i's for the two classes

Can quantify the separation (overlap) in terms of means and variances of the resulting 1-dimensional class distributions.

Fisher linear discriminant: preliminaries

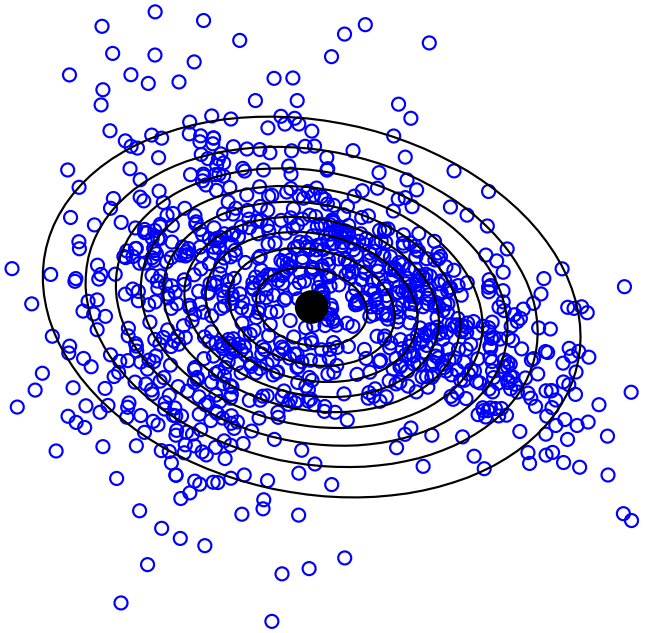
Class descriptions in $\mathcal{R}^d$:

Class ω_1 points



n_1 samples, mean $\boldsymbol{\mu}_1$, covariance Σ_1

Class ω_2 points

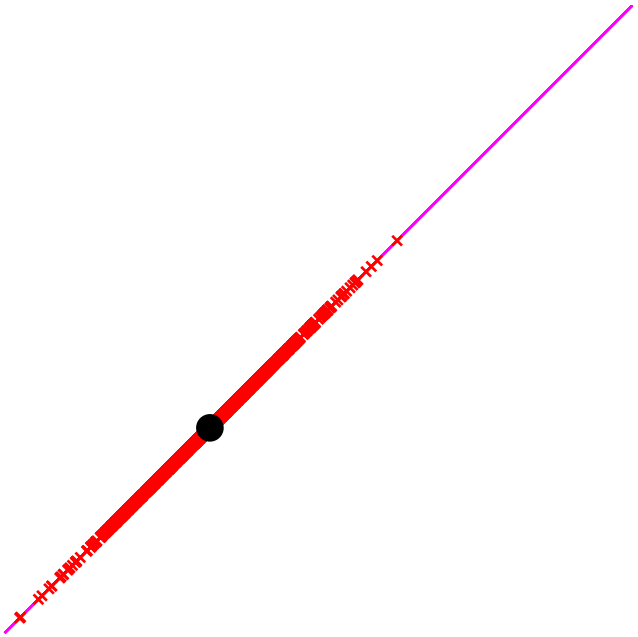


n_2 samples, mean $\boldsymbol{\mu}_2$, covariance Σ_2

Fisher linear discriminant: preliminaries

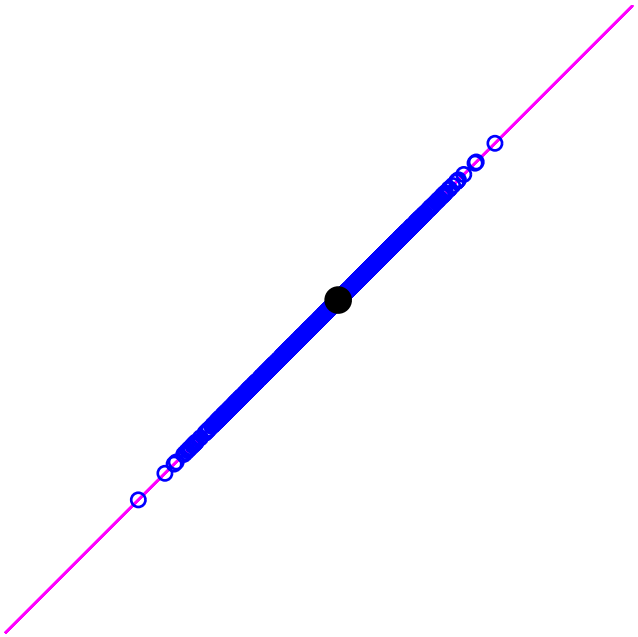
Projected class descriptions in $\mathcal{R}$:

Projected class ω_1 points



n_1 samples, mean $\mu_1 = \mathbf{w}_1^T \boldsymbol{\mu}_1$,
variance $\sigma_1^2 = \mathbf{w}_1^T \Sigma_1 \mathbf{w}_1$

Projected class ω_2 points



n_2 samples, mean $\mu_2 = \mathbf{w}_1^T \boldsymbol{\mu}_2$,
variance $\sigma_2^2 = \mathbf{w}_1^T \Sigma_2 \mathbf{w}_1$

The expressions for μ_i and σ_i in terms of $\boldsymbol{\mu}_i$ and Σ_i for $i = 1, 2$ can be easily derived. First define $\mathcal{X}_i = \{\mathbf{x}_j \mid y_j = \omega_i\}$ and $\mathcal{Z}_i = \{z_j = \mathbf{w}_1^T \mathbf{x}_j \mid y_j = \omega_i\}$

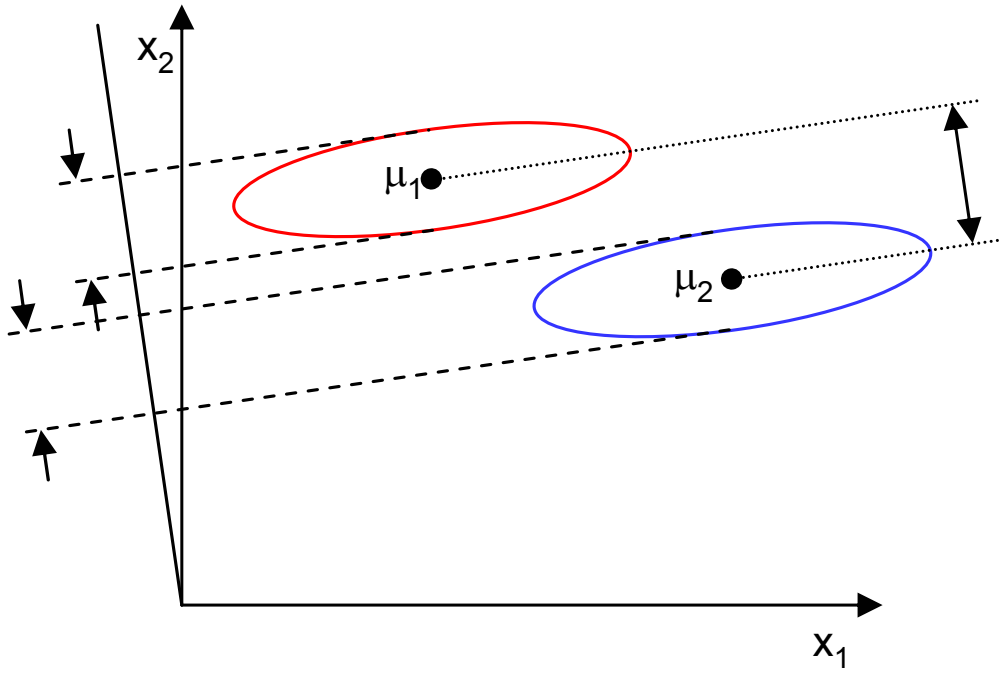
$$\mu_i = \frac{1}{n_i} \sum_{z \in \mathcal{Z}_i} z = \frac{1}{n_i} \sum_{\mathbf{x} \in \mathcal{X}_i} \mathbf{w}_1^T \mathbf{x} = \mathbf{w}_1^T \left(\frac{1}{n_i} \sum_{\mathbf{x} \in \mathcal{X}_i} \mathbf{x} \right) = \mathbf{w}_1^T \boldsymbol{\mu}_i$$

and

$$\begin{aligned}
\sigma_i^2 = \frac{1}{n_i} \sum_{z \in \mathcal{Z}_i} (z - \mu_i)^2 &= \frac{1}{n_i} \sum_{\mathbf{x} \in \mathcal{X}_i} (\mathbf{w}_1^T \mathbf{x} - \mathbf{w}_1^T \boldsymbol{\mu}_i)^2 \\
&= \frac{1}{n_i} \sum_{\mathbf{x} \in \mathcal{X}_i} (\mathbf{w}_1^T (\mathbf{x} - \boldsymbol{\mu}_i))^2 \\
&= \frac{1}{n_i} \sum_{\mathbf{x} \in \mathcal{X}_i} \mathbf{w}_1^T (\mathbf{x} - \boldsymbol{\mu}_i)(\mathbf{x} - \boldsymbol{\mu}_i)^T \mathbf{w}_1 \\
&= \mathbf{w}_1^T \left(\frac{1}{n_i} \sum_{\mathbf{x} \in \mathcal{X}_i} (\mathbf{x} - \boldsymbol{\mu}_i)(\mathbf{x} - \boldsymbol{\mu}_i)^T \right) \mathbf{w}_1 = \mathbf{w}_1^T \Sigma_i \mathbf{w}_1
\end{aligned}$$

Fisher linear discriminant

Intuitively, look for a projection where examples from the same class are projected very close to each other and, at the same time, the projected means are as far apart as possible.



The solution proposed by **Fisher** is to maximize a cost function that represents the difference between the means, normalized by a measure of the within-class scatter:

$$
\begin{aligned}
J_{\text{Fisher}}(\mathbf{w}_1) &= \frac{\big(\text{Separation of projected means}\big)^2}{\text{Sum of within class scatter}} \\
&= \frac{(\mathbf{w}_1^T \boldsymbol{\mu}_2 - \mathbf{w}_1^T \boldsymbol{\mu}_1)^2}{n_1 \mathbf{w}_1^T \Sigma_1 \mathbf{w}_1 + n_2 \mathbf{w}_1^T \Sigma_2 \mathbf{w}_1} \\
&= \frac{\mathbf{w}_1^T (\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1)(\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1)^T \mathbf{w}_1}{\mathbf{w}_1^T (n_1 \Sigma_1 + n_2 \Sigma_2) \mathbf{w}_1}
\end{aligned}
$$

Fisher linear discriminant

The solution to the optimization problem

$$\mathbf{w}_1^* = \arg\max_{\mathbf{w}_1} \; J_{\text{Fisher}}(\mathbf{w}_1)$$

is the **Fisher Linear Discriminant** (1936)

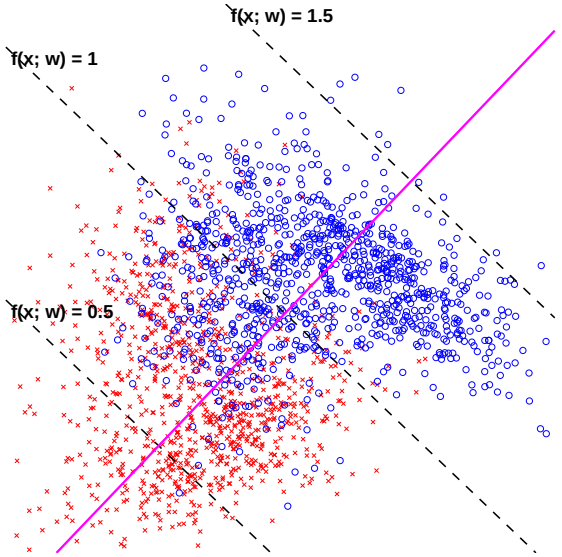
$$\mathbf{w}_1^* \propto (n_1\Sigma_1 + n_2\Sigma_2)^{-1}(\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1)$$

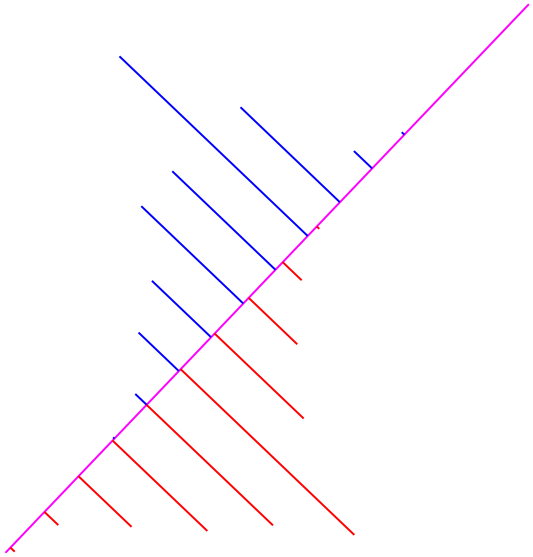
It should be noted it is not a discriminant but rather a specific choice of direction for the projection of the data down to one dimension.

Fisher linear discriminant

For our toy example $\mathbf{w}_1^* \propto (n_1\Sigma_1 + n_2\Sigma_2)^{-1}(\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1)$ is shown below

Data points and the projection direction





histograms of z_i's for the two classes

Details of the optimization

Let $S_B = (\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1)(\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1)^T$ and $S_W = n_1\Sigma_1 + n_2\Sigma_2$, then the cost function is written as

$$J_{\text{Fisher}} = \frac{\mathbf{w}_1^T S_B \mathbf{w}_1}{\mathbf{w}_1^T S_W \mathbf{w}_1}$$

The derivative wrt $\mathbf{w}_1$ is

$$\begin{aligned}
\frac{\partial J_{\text{Fisher}}(\mathbf{w}_1)}{\partial \mathbf{w}_1} &= \frac{1}{\mathbf{w}_1^T S_W \mathbf{w}_1}(S_B + S_B^T)\,\mathbf{w}_1 - \frac{\mathbf{w}_1^T S_B \mathbf{w}_1}{(\mathbf{w}_1^T S_W \mathbf{w}_1)^2}(S_W + S_W^T)\,\mathbf{w}_1 \\
&= \frac{2S_B\,\mathbf{w}_1}{\mathbf{w}_1^T S_W \mathbf{w}_1} - \frac{2(\mathbf{w}_1^T S_B \mathbf{w}_1) S_W\,\mathbf{w}_1}{(\mathbf{w}_1^T S_W \mathbf{w}_1)^2}, \quad \text{as } S_W = S_W^T,\ S_B = S_B^T \\
&= \frac{2(\mathbf{w}_1^T S_W \mathbf{w}_1) S_B\,\mathbf{w}_1 - 2(\mathbf{w}_1^T S_B \mathbf{w}_1) S_W\,\mathbf{w}_1}{(\mathbf{w}_1^T S_W \mathbf{w}_1)^2}
\end{aligned}$$

Set the derivative to 0 and get

$$(\mathbf{w}_1^T S_W \mathbf{w}_1) S_B \, \mathbf{w}_1 - (\mathbf{w}_1^T S_B \mathbf{w}_1) S_W \, \mathbf{w}_1 = 0$$

$$\implies (\mathbf{w}_1^T S_W \mathbf{w}_1) S_B \, \mathbf{w}_1 = (\mathbf{w}_1^T S_B \mathbf{w}_1) S_W \, \mathbf{w}_1$$

Can we solve this? Remember $S_B = (\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1)(\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1)^T$ so

$$S_B \, \mathbf{w}_1 = \Big((\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1)(\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1)^T \Big) \, \mathbf{w}_1 = \Big((\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1)^T \mathbf{w}_1 \Big) \, (\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1)$$

Thus

$$S_B \, \mathbf{w}_1 \propto (\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1)$$

We do not care about the magnitude of $\mathbf{w}_1$ only its direction. Then by dropping scale factors get

$$S_W \, \mathbf{w}_1 \propto (\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1)$$

$$\implies \mathbf{w}_1 \propto S_W^{-1} (\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1)$$

Fisher classifier

Use $\mathbf{w}_1$ to construct a classifier. Project the training data onto a line parallel to $\mathbf{w}_1$. Must find a threshold θ such that

$$\mathbf{w}_1^T\mathbf{x} \geq \theta \implies \mathbf{x} \text{ belongs to class } \omega_1$$
$$\mathbf{w}_1^T\mathbf{x} < \theta \implies \mathbf{x} \text{ belongs to class } \omega_2$$

How should we learn θ ?

Fisher classifier

Use $\mathbf{w}_1$ to construct a classifier. Project the training data onto the line $\mathbf{w}_1$. Must find a threshold θ such that

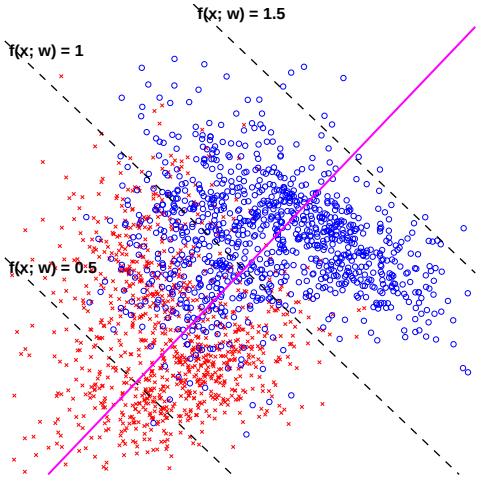
$$\mathbf{w}_1^T\mathbf{x} \geq \theta \implies \mathbf{x} \text{ belongs to class } \omega_1$$
$$\mathbf{w}_1^T\mathbf{x} < \theta \implies \mathbf{x} \text{ belongs to class } \omega_2$$

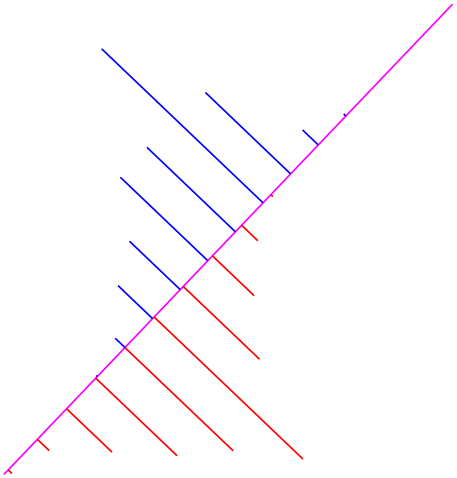
How should we learn θ ?

- Project the training data from both classes onto $\mathbf{w}$.
- Search for θ such that the number of mis-classifications is minimized.

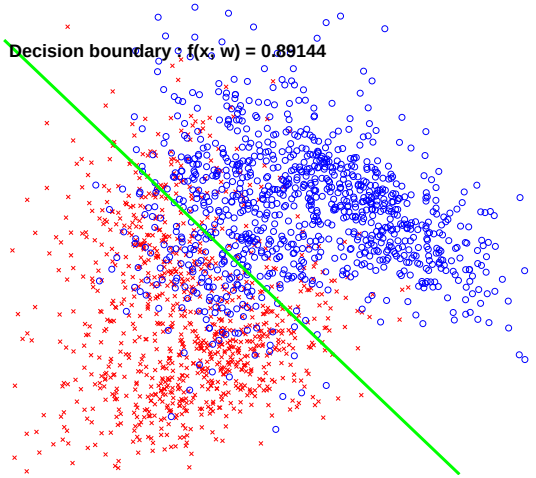
Classifier based on Fisher discriminant

projection direction





histograms of z_i's



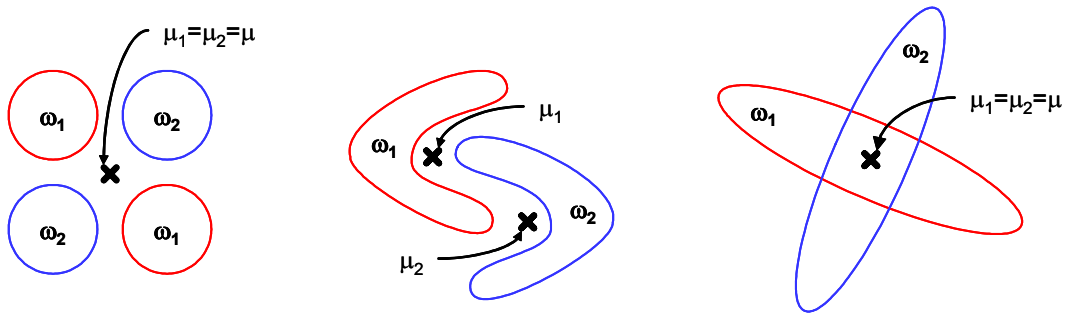
decision boundary

Is this our only option ?

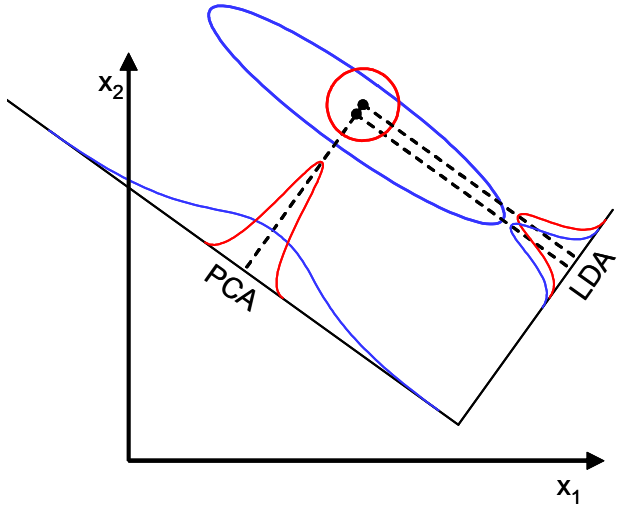
Limitations of Fisher discrimination

The Fisher linear discriminant (a.k.a linear discriminant analysis) assumes unimodal Gaussian likelihoods

If the distributions are significantly non-Gaussian, the LDA projections will not be able to preserve any complex structure of the data, which may be needed for classification.



LDA will fail when the discriminatory information is not in the mean but rather in the variance of the data.



(**Note**: Fisher's LDA generalizes very gracefully for c-class problems. Instead of one projection y, seek $(c-1)$ projections $(y_1, y_2, \ldots, y_{c-1})$ by means of $(c-1)$ projection vectors $\mathbf{w}_i$, which can be arranged by columns into a projection matrix $W = [\mathbf{w}_1 \mathbf{w}_2 \ldots \mathbf{w}_{c-1}]$.)

Attempt 4: Logistic regression

The optimal decisions are based on the posterior class probabilities $P(\omega|\mathbf{x})$. Can write these decisions as

$$\text{Class}\,(\mathbf{x}) = \begin{cases} 1, & \text{if} \quad \log \frac{P(\omega=1\,|\,\mathbf{x})}{P(\omega=0\,|\,\mathbf{x})} > 0 \\ 0, & \text{otherwise.} \end{cases}$$

Generally, don't know $P(\omega\,|\,\mathbf{x})$ but could parametrise the decision boundary as

$$\log \frac{P(\omega=1\,|\,\mathbf{x})}{P(\omega=0\,|\,\mathbf{x})} = f(\mathbf{x};\mathbf{w}) = w_0 + \mathbf{w}_1^T\mathbf{x}$$

Logistic regression cont'd

This log-odds model

$$\log \frac{P(\omega = 1 \,|\, \mathbf{x})}{P(\omega = 0 \,|\, \mathbf{x})} = w_0 + \mathbf{w}_1^T \mathbf{x}$$

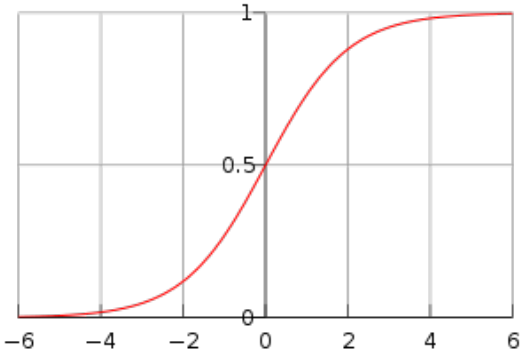
gives rise to a specific form of the conditional likelihood of the labels (the logistic model):

$$P(\omega = 1 \,|\, \mathbf{x}, \mathbf{w}) = g(w_0 + \mathbf{w}_1^T \mathbf{x})$$

where

$$g(z) = (1 + \exp(-z))^{-1} = \frac{\exp(z)}{1 + \exp(z)}$$

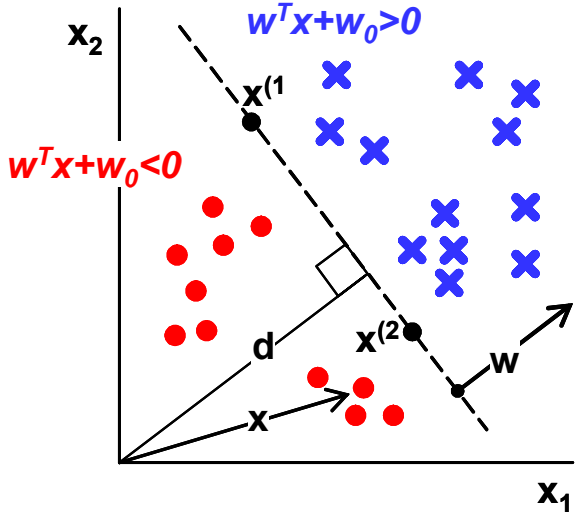
is a logistic *squashing function* that turns linear predictions into probabilities.



Logistic regression: decisions

A logistic regression model implies a linear decision boundary

$$\log \frac{P(\omega = 1 \,|\, \mathbf{x})}{P(\omega = 0 \,|\, \mathbf{x})} = w_0 + \mathbf{w}_1^T \mathbf{x} = 0$$



Fitting logistic regression models

Given training data $\{(\mathbf{x}_i, y_i)\}_{i=1}^n$, we can fit the logistic model using the maximum (conditional) log-likelihood criterion:

$$l(D; \mathbf{w}) = \sum_{i=1}^{n} \log P(\omega = y_i \,|\, \mathbf{x}_i)$$

where

$$P(\omega = 1 \,|\, \mathbf{x}) = g(w_0 + \mathbf{w}_1^T \mathbf{x})$$

The log-likelihood function $l(D; \mathbf{w})$ is a *jointly concave* function of the parameters $\mathbf{w}$; a number of optimization techniques are available for finding the maximizing parameters.

Re-writing

Can re-write the conditional distribution as

$$\log P(\omega = y_i \,|\, \mathbf{x}_i) = y_i \, \log P(\omega = 1 \,|\, \mathbf{x}_i) + (1 - y_i) \, \log(1 - P(\omega = 1 \,|\, \mathbf{x}_i))$$

When $y_i = 1$ then

$$\begin{aligned} \log P(\omega = 1 \,|\, \mathbf{x}_i) &= 1 \, \log P(\omega = 1 \,|\, \mathbf{x}_i) + (1 - 1) \, \log(1 - P(\omega = 1 \,|\, \mathbf{x}_i)) \\ &= \log P(\omega = 1 \,|\, \mathbf{x}_i) \end{aligned}$$

and when $y_i = 0$ then

$$\begin{aligned} \log P(\omega = 0 \,|\, \mathbf{x}_i) &= 0 \, \log P(\omega = 1 \,|\, \mathbf{x}_i) + (1 - 0) \, \log(1 - P(\omega = 1 \,|\, \mathbf{x}_i)) \\ &= \log(1 - P(\omega = 1 \,|\, \mathbf{x}_i)) \\ &= \log P(\omega = 0 \,|\, \mathbf{x}_i) \end{aligned}$$

Thus log-likelihood of the data becomes

$$l(D; \mathbf{w}) = \sum_{i=1}^{n} \log P(\omega = y_i \,|\, \mathbf{x}_i)$$
$$= \sum_{i=1}^{n} [y_i \, \log P(\omega = 1 \,|\, \mathbf{x}_i) + (1 - y_i) \, \log(1 - P(\omega = 1 \,|\, \mathbf{x}_i))]$$

Remember

$$P(\omega = 1 \,|\, \mathbf{x}) = \frac{\exp(w_o + \mathbf{w}_1^T \, \mathbf{x})}{1 + \exp(w_o + \mathbf{w}_1^T \, \mathbf{x})}$$
$$1 - P(\omega = 1 \,|\, \mathbf{x}) = \frac{1}{1 + \exp(w_o + \mathbf{w}_1^T \, \mathbf{x})}$$

Substituting these into the expression for the log-likelihood and after

a little algebra get

$$l(D; \mathbf{w}) = \sum_{i=1}^{n} \left[y_i \left(w_o + \mathbf{w}_1^T \mathbf{x}_i \right) - \log(1 + \exp(w_o + \mathbf{w}_1^T \mathbf{x}_i)) \right]$$

Let $\mathbf{w} = (w_0, \mathbf{w}_1^T)^T$ and $\mathbf{x}' = (1, \mathbf{x}^T)^T$ then

$$l(D; \mathbf{w}) = \sum_{i=1}^{n} \left[y_i \, \mathbf{w}^T \mathbf{x}_i' - \log(1 + \exp(\mathbf{w}^T \mathbf{x}_i')) \right]$$

Take the derivatives of the log-likelihood w.r.t. the parameters and

set to zero get

$$\begin{aligned}\frac{\partial}{\partial \mathbf{w}} l(D;\mathbf{w}) &= \sum_{i=1}^{n} \left[y_i \mathbf{x}'_i - \frac{\exp(\mathbf{w}^T \mathbf{x}'_i)}{1+\exp(\mathbf{w}^T \mathbf{x}'_i)} \mathbf{x}'_i \right] \\ &= \sum_{i=1}^{n} \mathbf{x}'_i \left[y_i - \frac{\exp(\mathbf{w}^T \mathbf{x}'_i)}{1+\exp(\mathbf{w}^T \mathbf{x}'_i)} \right] \\ &= \sum_{i=1}^{n} \mathbf{x}'_i \left[y_i - P(\omega = 1 \mid \mathbf{x}_i) \right] = \mathbf{0}\end{aligned}$$

Now have a set of $d+1$ non-linear equations $\frac{\partial}{\partial w_j} l(D;\mathbf{w}) = 0$ for $j = 0, 1, 2, \ldots, d$, no easy closed form solution.

Stochastic gradient ascent

Can maximize the log-likelihood in an *on-line* or incremental fashion.

Given each training input $\mathbf{x}_i$ and its binary label $y_i \in \{0, 1\}$, we can change the parameters $\mathbf{w}$ slightly to increase the corresponding log-probability:

$$\begin{aligned}\mathbf{w} &\leftarrow \mathbf{w} + \eta \frac{\partial}{\partial \mathbf{w}} \log P(y_i \,|\, \mathbf{x}_i, \mathbf{w}) \\ &= \mathbf{w} + \eta \underbrace{(y_i - P(w = 1 \,|\, \mathbf{x}_i, \mathbf{w}))}_{\text{prediction error}} \mathbf{x}_i'\end{aligned}$$

where η is the *learning rate*.

Examples that are already correctly classified do not lead to any significant updates.

Gradient ascent of the log-likelihood

Can also perform gradient ascent steps on the log-likelihood of all the training labels given examples at the same time. In other words,

$$\begin{aligned}\mathbf{w} &\leftarrow \mathbf{w} + \eta \frac{\partial}{\partial \mathbf{w}} l(D; \mathbf{w}) \\ &= \mathbf{w} + \eta \sum_{i=1}^{n} (\, y_i - P(w = 1 \,|\, \mathbf{x}_i, \mathbf{w}))\, \mathbf{x}'_i\end{aligned}$$

Still need to figure out a way to set the learning rate to guarantee convergence.

Additive models and classification

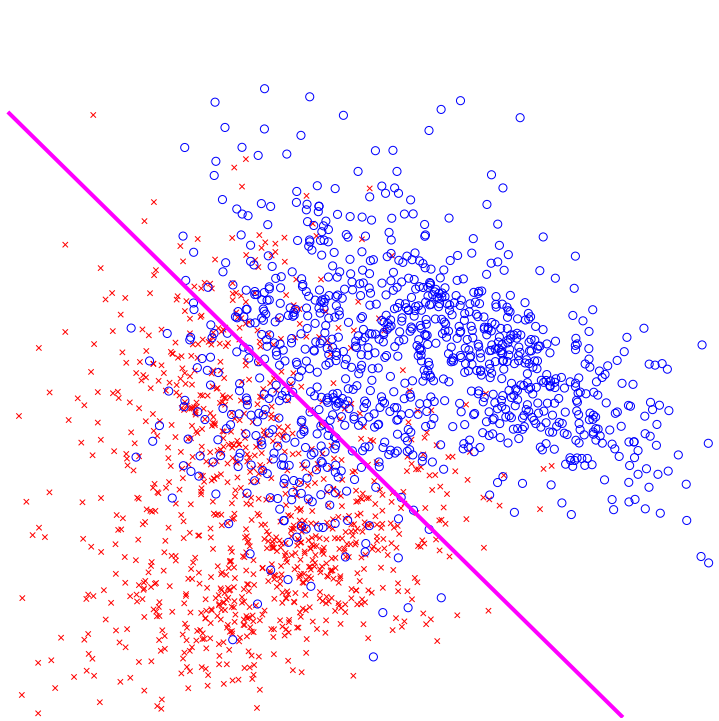
Can extend the logistic regression models to additive logistic models

$$P(\omega = 1 \,|\, \mathbf{x}, \mathbf{w}) = g(w_0 + w_1\, \phi_1(\mathbf{x}) + \cdots + w_m\, \phi_m(\mathbf{x}))$$

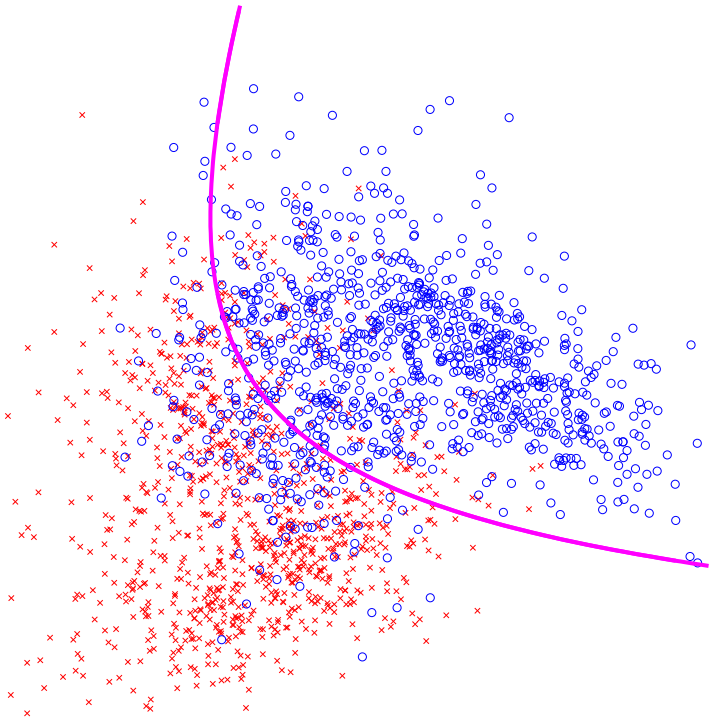
Free to choose the basis functions $\phi_i(\mathbf{x})$ to capture relevant properties of any specific classification problem.

Since one can also easily over-fit, use leave-one-out cross-validation (in terms of log-likelihood or classification error) to estimate the generalization performance.

Toy Example - Logistic regression classifier



linear decision boundary



quadratic decision boundary

Pen & Paper Assignment

- Details available on the course website.
- You will implement perceptron learning to find the separating hyperplane between images of digits.
- Mail me about any errors you spot in the Exercise notes.
- I will notify the class about errors spotted and corrections via the course website and mailing list.