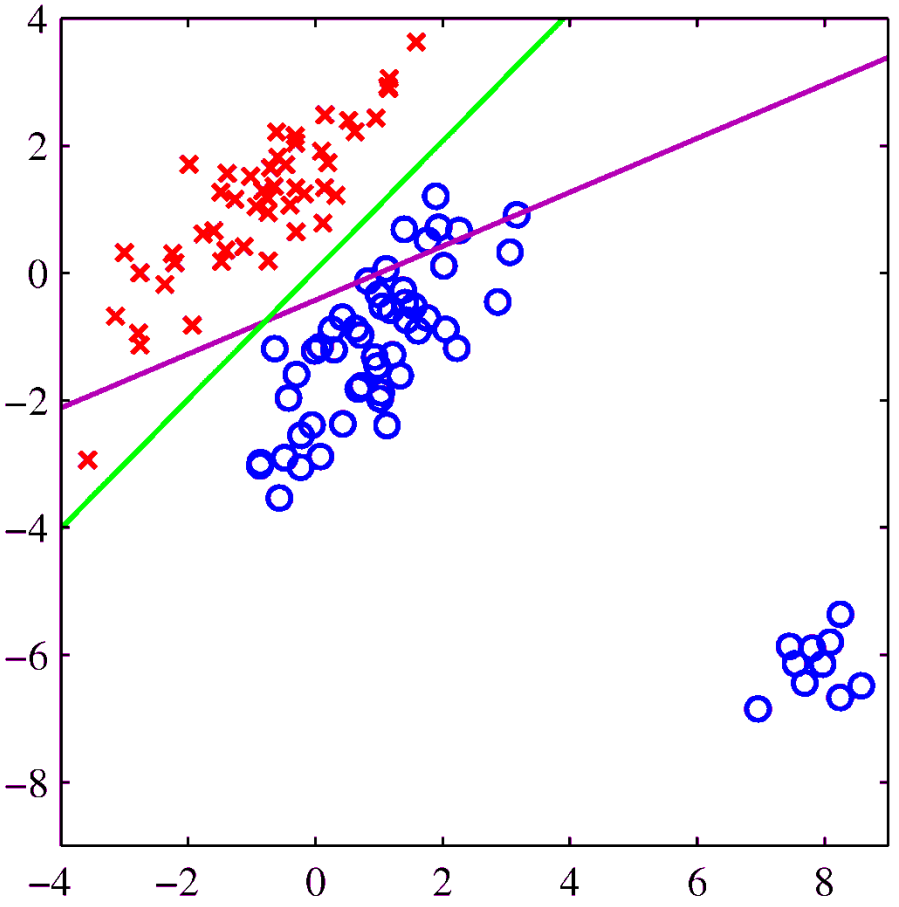
Linear Classification

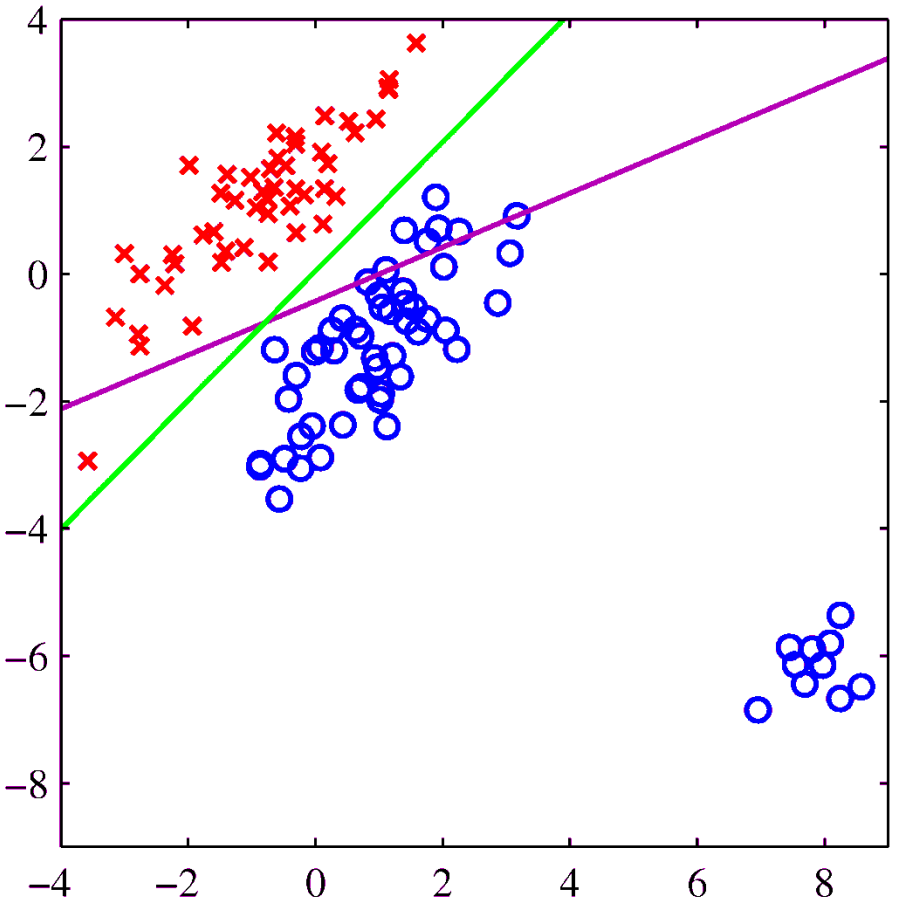
CSE 4309 – Machine Learning
Vassilis Athitsos
Computer Science and Engineering Department
University of Texas at Arlington

Example of Linear Classification



- Red points: patterns belonging to class C_1.
- Blue points: patterns belonging to class C_2.
- Goal: find a **linear decision boundary** separating C_1 from C_2.
- Points on one side of the line will be classified as belonging to C_1, points on the other side will be classified as C_2.
- The red line is one example of such a decision boundary.
  - It misclassified a few patterns.
- The green line is another example.

Linear Classification

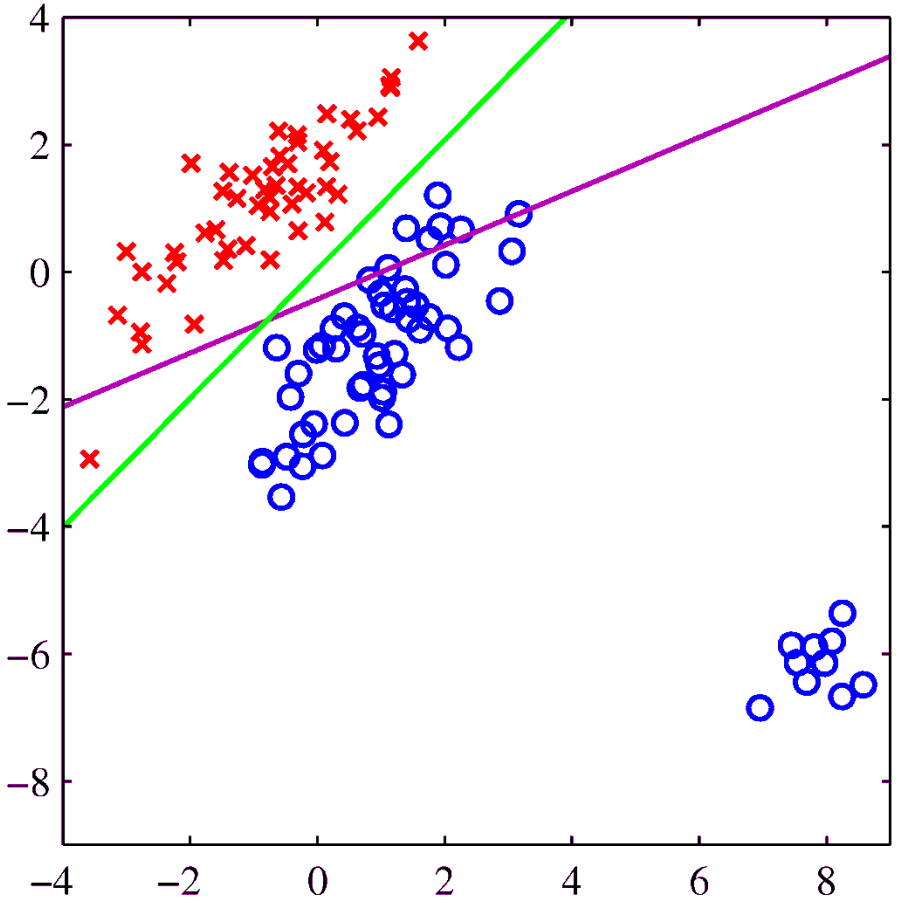


- Mathematically, assuming input patterns are D-dimensional vectors:
  - We are looking for a **decision boundary** in the form of a (D-1)-dimensional hyperplane separating the two classes.
  - Points on one side of the hyperplane will be classified as belonging to C_1, points on the other side will be classified as C_2.
- If inputs are 2-dimensional vectors, the decision boundary is a line.
- If inputs are 3-dimensional vectors, the decision boundary is a 2-dimensional surface.

Linear Classification

- Input space: $\mathbb{R}^D$
- The decision boundary is is a (D-1)-dimensional hyperplane defined using this equation:

$$\boldsymbol{w}^T\boldsymbol{x} + w_0 = \mathrm{c}$$



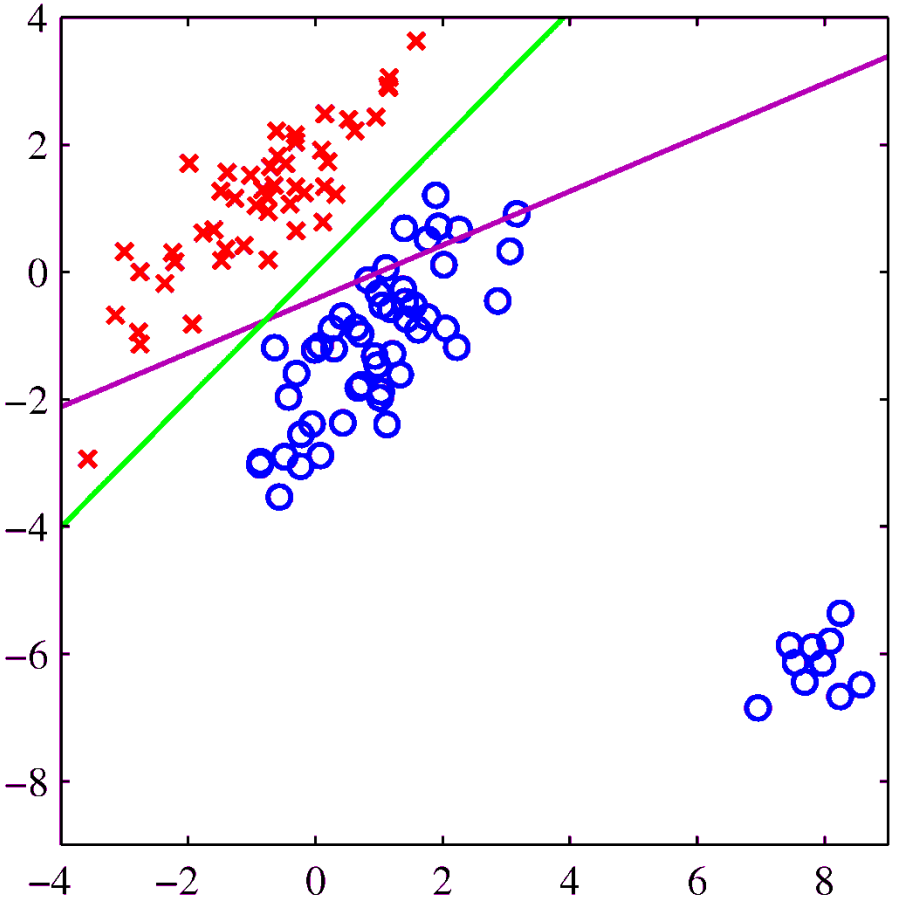
- In the equation above: w_0 and c are real numbers, **w** and **x** are column vectors:

$$\mathbf{w} = \begin{bmatrix} w_1 \\ \cdots \\ w_D \end{bmatrix} \qquad \mathbf{x} = \begin{bmatrix} x_1 \\ \cdots \\ x_D \end{bmatrix}$$

Linear Classification

- In order to be able to use vector and matrix notation, we extend vector $\boldsymbol{w}$ and $\boldsymbol{x}$, as follows:

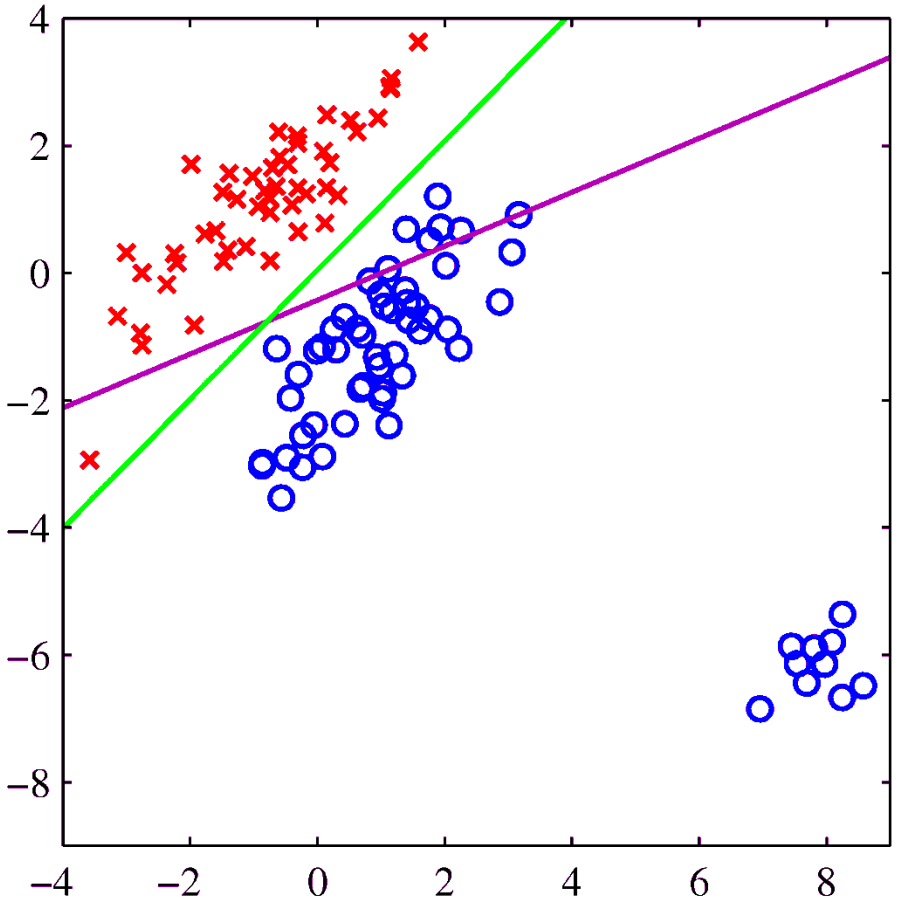
$$\boldsymbol{w} = \begin{bmatrix} w_0 \\ w_1 \\ \cdots \\ w_D \end{bmatrix} \qquad \boldsymbol{x} = \begin{bmatrix} 1 \\ x_1 \\ \cdots \\ x_D \end{bmatrix}$$



- Then, the decision boundary is defined by equation:

$$\boldsymbol{w}^T \boldsymbol{x} = \mathrm{c}$$

Linear Classification



- Furthermore, we can use basis functions $\varphi_i(x)$, which we pick manually.
- As before, $\varphi_0(x) = 1$.

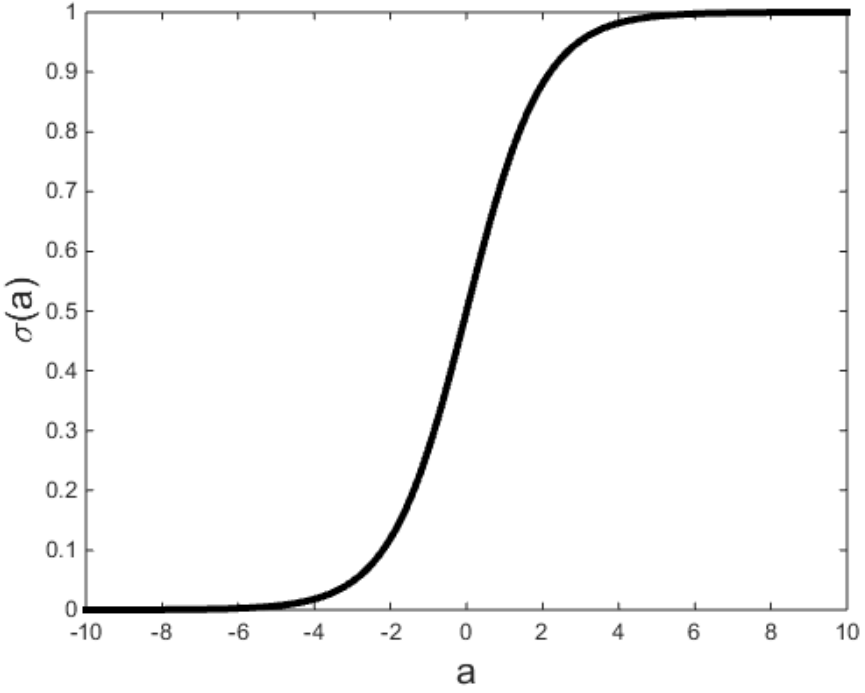
$$\boldsymbol{w} = \begin{bmatrix} w_0 \\ w_1 \\ \dots \\ w_D \end{bmatrix} \quad \varphi(x) = \begin{bmatrix} 1 \\ \varphi_1(x) \\ \dots \\ \varphi_D(x) \end{bmatrix}$$

- Then, the decision boundary is defined by equation:

$$\boldsymbol{w^T}\varphi(x) = \mathrm{c}$$

Logistic Regression

- Despite the word "regression" in its name, **logistic regression** is actually a linear classification method.
- In logistic regression, the goal is to compute a decision boundary $\boldsymbol{w}^{\boldsymbol{T}}\varphi(x) = 0$.
- The classification function itself is defined as:



$$\mathrm{y(x)} = \sigma\left(\boldsymbol{w}^{\mathrm{T}}\varphi(x)\right)$$

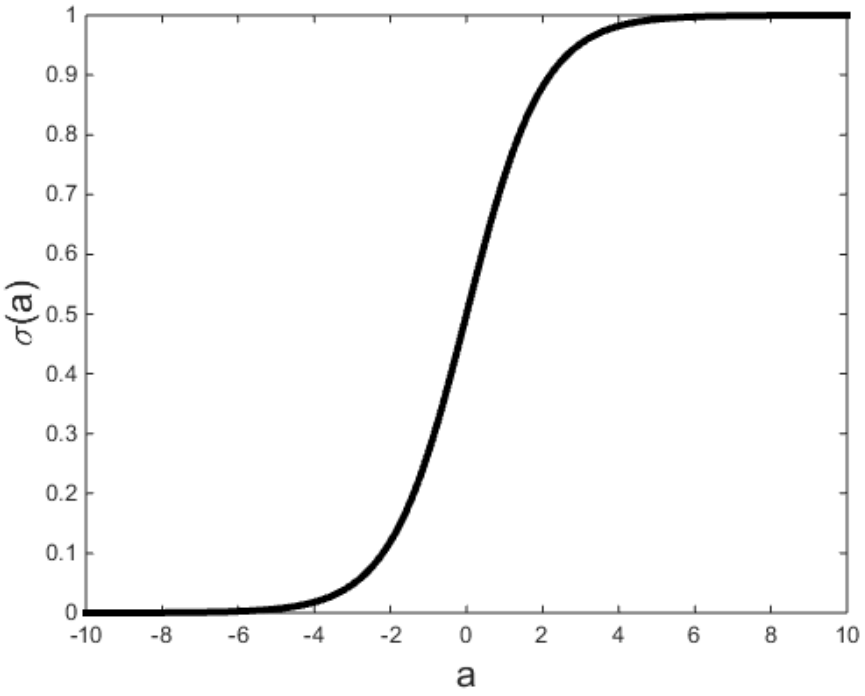
where σ is the sigmoidal function we have seen before:

$$\sigma(a) = \frac{1}{1 + e^{-a}}$$

Logistic Regression

- In logistic regression, the goal is to compute a decision boundary $\boldsymbol{w^T}\varphi(x) = 0$.
- The classification function itself is defined as:

$$\mathrm{y(x)} = \sigma\left(\boldsymbol{w}^{\mathrm{T}}\varphi(x)\right)$$



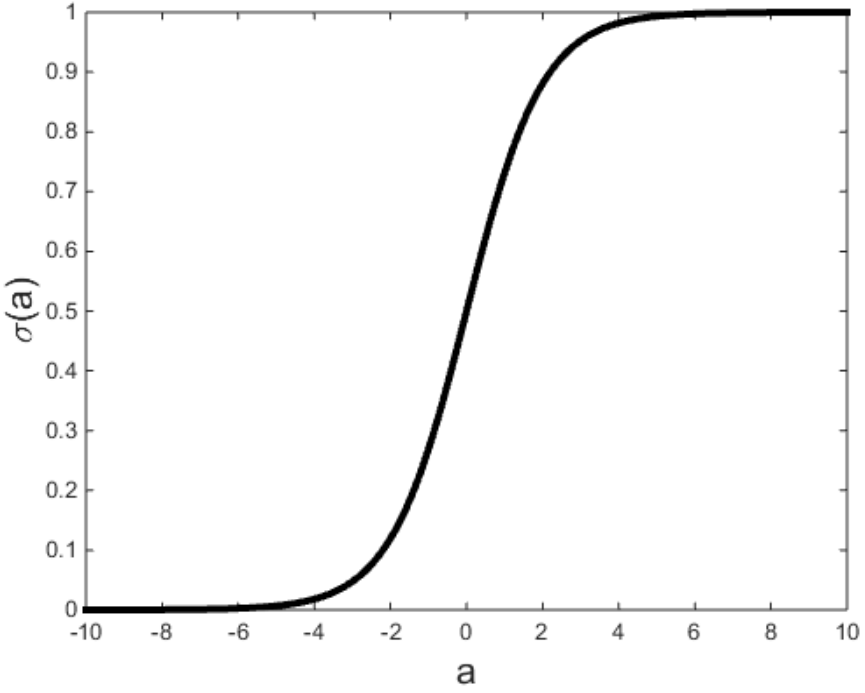
- We want to separate two classes, denoted as C_0 and C_1.
- We have a set of training inputs: $X = \{x_1, \ldots, x_N\}$
- We also have a set of corresponding outputs: $\boldsymbol{t} = \{t_1, \ldots, t_N\}$
- Each t_n is either 0 or 1:
  - 0 corresponds to class C_0.
  - 1 corresponds to class C_1.

Logistic Regression

- Classification function:

$$\mathrm{y}(\mathrm{x}) = \sigma\left(\boldsymbol{w}^{\mathrm{T}}\varphi(x)\right)$$

- We treat the output of $\sigma\left(\boldsymbol{w}^{\mathrm{T}}\varphi(x)\right)$ as an estimate of probability $\mathrm{p}(C_1|x)$.



- Then, $\mathrm{p}(C_0|x) = 1 - \mathrm{p}(C_1|x)$.

- When $\boldsymbol{w}^{\boldsymbol{T}}\varphi(x) > 0$, $\mathrm{p}(C_1|x) > 0.5$, so C_1 is more likely than C_0.

- When $\boldsymbol{w}^{\boldsymbol{T}}\varphi(x) < 0$, $\mathrm{p}(C_0|x) < 0.5$, so C_0 is more likely than C_1.

Finding the Most Likely Solution

- Our rationale is almost identical to the rationale we used for finding the most likely solution for linear regression.
- Suppose we have a set of training inputs: $X = \{x_1, \ldots, x_N\}$
- We also have a set of corresponding outputs: $\boldsymbol{t} = \{t_1, \ldots, t_N\}$
- We assume that training inputs are independent of each other.
- We assume that outputs are conditionally independent of each other, given their inputs and our estimate of $\boldsymbol{w}$.
- Then:

$$p(\boldsymbol{w}|X, \boldsymbol{t}) = \frac{p(\boldsymbol{t}|X, \boldsymbol{w}) * p(\boldsymbol{w}|X)}{p(\boldsymbol{t}|X)}$$

Finding the Most Likely Solution

$$p(w|X, \boldsymbol{t}) = \frac{p(\boldsymbol{t}|X, \boldsymbol{w}) * p(\boldsymbol{w}|X)}{p(\boldsymbol{t}|X)}$$

- We assume that, given X, all values of **w** are equally likely.
- Then, $\frac{p(\boldsymbol{w}|X)}{p(\boldsymbol{t}|X)}$ is a constant that does not depend on **w**.
- Therefore, finding the w that maximizes $p(\boldsymbol{w}|X, \boldsymbol{t})$ is the same as finding the w that maximizes $p(\boldsymbol{t}|X, \boldsymbol{w})$.
- $p(\boldsymbol{t}|X, \boldsymbol{w})$ is the **likelihood** of the training data.
- So, to find the most likely answer **w**, we must find the value of **w** that maximizes the likelihood of the training data.

Probability of t_n

- We want to find the **w** that maximizes $p(\boldsymbol{t}|\, X, \boldsymbol{w})$.

$$p(\boldsymbol{t}\,|X, \boldsymbol{w}) = \prod_{n=1}^{n} p(t_n\,|x_n, \boldsymbol{w})$$

- How can we compute $p(t_n\,|x_n, \boldsymbol{w})$?
- Remember: we assume that $\mathrm{y}(\mathrm{x}, \mathbf{w})$ is an estimate of $\mathrm{p}(C_1 \mid x)$.
- Under this assumption, for a given x_n:
  - The probability that t_n (the correct class for x_n) is 1 is $\mathrm{y}(\mathrm{x}, \mathbf{w})$.
  - The probability that t_n (the correct class for x_n) is 0 is $\{1 - \mathrm{y}(\mathrm{x}, \mathbf{w})\}$.

Probability of t_n

- To simplify notation, define $y_n = y(x_n, \mathbf{w})$.
- Under this assumption, for a given x_n:
  - $p(t_n = 1|x_n, w) = y_n$.
  - $p(t_n = 0|x_n, w) = (1 - y_n)$.
- So:
  - If $t_n = 1$, then $p(t_n|x_n, w) = y_n$.
  - If $t_n = 0$, then $p(t_n|x_n, w) = (1 - y_n)$.
- Regardless of whether t_n is 1 or 0, the following holds:

$$p(t_n|x_n, w) = (y_n)^{t_n}(1 - y_n)^{1-t_n}$$

Probability of t_n

- Regardless of whether t_n is 1 or 0, the following holds:

$$p(t_n|x_n, w) = (y_n)^{t_n}(1 - y_n)^{1-t_n}$$

- To make sense of this formula, we just need to verify that it is correct when $t_n = 1$ and when $t_n = 0$.
- If $t_n = 1$, then $p(t_n|x_n, w) = y_n$. The above formula becomes:

$$\begin{aligned} p(t_n|x_n, w) &= (y_n)^{t_n}(1 - y_n)^{1-t_n} \\ &= (y_n)^1(1 - y_n)^{1-1} \\ &= y_n(1 - y_n)^0 = y_n \end{aligned}$$

- So, the formula is verified for the case where $t_n = 1$.

Probability of t_n

- Regardless of whether t_n is 1 or 0, the following holds:

$$p(t_n|x_n, w) = (y_n)^{t_n}(1 - y_n)^{1-t_n}$$

- If $t_n = 0$, then $p(t_n|x_n, w) = (1 - y_n)$. The above formula becomes:

$$\begin{aligned} p(t_n|x_n, w) &= (y_n)^{t_n}(1 - y_n)^{1-t_n} \\ &= (y_n)^{0}(1 - y_n)^{1-0} \\ &= 1(1 - y_n)^{1} = (1 - y_n) \end{aligned}$$

- So, the formula is verified for the case where $t_n = 0$.

Probability of $\boldsymbol{t}$

- We have shown that:

$$p(t_n|x_n, w) = (y_n)^{t_n}(1 - y_n)^{1-t_n}$$

- Therefore:

$$p(\boldsymbol{t}\,|X, \boldsymbol{w}) = \prod_{n=1}^{n} p(t_n\,|x_n, \boldsymbol{w})$$

$$= \prod_{n=1}^{n} [(y_n)^{t_n}(1 - y_n)^{1-t_n}]$$

Log Likelihood of $\boldsymbol{t}$

$$p(\boldsymbol{t}\,|X,\boldsymbol{w}) = \prod_{n=1}^{n}[(y_n)^{t_n}(1-y_n)^{1-t_n}]$$

- Therefore, the log likelihood $\ln(p(\boldsymbol{t}\,|X,\boldsymbol{w}))$ is:

$$\ln(p(\boldsymbol{t}\,|X,\boldsymbol{w})) = \sum_{n=1}^{n}[\ln((y_n)^{t_n}(1-y_n)^{1-t_n})] \Rightarrow$$

$$\ln(p(\boldsymbol{t}\,|X,\boldsymbol{w})) = \sum_{n=1}^{n}[t_n \ln(y_n) + (1-t_n)\ln(1-y_n))]$$

Gradient of the Log Likelihood of $\boldsymbol{t}$

- We want to find the value of **w** that maximizes

$$\ln(p(\boldsymbol{t}\,|X,\boldsymbol{w})) = \sum_{n=1}^{n}[t_n \ln(y_n) + (1-t_n)\ln(1-y_n))]$$

- The textbook defines $E(\boldsymbol{w}) = -\ln(p(\boldsymbol{t}\,|X,\boldsymbol{w}))$.
- That negative sign is not important for our purposes. It just means that to maximize $\ln(p(\boldsymbol{t}\,|X,\boldsymbol{w}))$ we need to minimize $E(\boldsymbol{w})$.
- Thus, we need to find the **w** such that the gradient $\nabla E(\boldsymbol{w})$ becomes 0.

Gradient of the Log Likelihood of $\boldsymbol{t}$

- We will not derive this, but it turns out that:

$$\nabla E(\boldsymbol{w}) = \sum_{n=1}^{n} [(y_n - t_n)\varphi_n]$$

- There is no closed-form solution for computing a value of w so that $\nabla E(\boldsymbol{w}) = 0$.
- We have two alternatives for minimizing $E(\boldsymbol{w})$.
  - An online (sequential learning) method, where we update **w** every time we get a new training example (x_n, t_n).
  - A batch processing method called **iterative reweighted least squares.**

Sequential Learning

- We follow the same approach as we did in sequential learning for linear regression.
- We first, somehow, get an initial estimate $\boldsymbol{w}^{(0)}$.
  - For example, we can initialize $\boldsymbol{w}^{(0)}$ to random values between -1 and 1.
  - You can also initialize $\boldsymbol{w}^{(0)}$ to be the zero vector (all entries equal to zero). That also works, I have verified it in my code.
- Then, we observe training examples, one by one.
- Every time we observe a new training example, we update the estimate.

Sequential Learning

- The nth training example contributes to the overall gradient $\nabla E(\boldsymbol{w})$ a term $\nabla E_n(\boldsymbol{w})$ defined as:

$$\nabla E_n(\boldsymbol{w}) = (y_n - t_n)\varphi(x_n)$$

- When we observe the nth training example, we update the estimate from $w^{(\tau)}$ to $w^{(\tau+1)}$ as follows:

$$\boldsymbol{w}^{(\tau+1)} = \boldsymbol{w}^{(\tau)} - \eta\nabla E_n(\boldsymbol{w}) = \boldsymbol{w}^{(\tau)} - \eta(y_n - t_n)\varphi(x_n)$$

- η is called the **learning rate**. It is picked manually.
- This whole process is called **stochastic gradient descent.**

Sequential Learning - Intuition

- When we observe the nth training example, we update the estimate from $w^{(\tau)}$ to $w^{(\tau+1)}$ as follows:

$$\boldsymbol{w}^{(\tau+1)} = \boldsymbol{w}^{(\tau)} - \eta \nabla E_n(\boldsymbol{w}) = \boldsymbol{w}^{(\tau)} - (y_n - t_n)\varphi(x_n)$$

- What is the intuition behind this update?
- The gradient $\nabla E_n(\boldsymbol{w})$ is a vector that points in the direction where $E_n(\boldsymbol{w})$ increases.
- Therefore, subtracting a very small amount of $\nabla E_n(\boldsymbol{w})$ from $\boldsymbol{w}$ should make $E_n(\boldsymbol{w})$ a little bit smaller.

Sequential Learning - Intuition

- When we observe the nth training example, we update the estimate from $w^{(\tau)}$ to $w^{(\tau+1)}$ as follows:

$$\boldsymbol{w}^{(\tau+1)} = \boldsymbol{w}^{(\tau)} - \eta \nabla E_n(\boldsymbol{w}) = \boldsymbol{w}^{(\tau)} - (y_n - t_n)\varphi(x_n)$$

- Choosing a good value for η is important.
  - If η is too small, the minimization may happen too slowly and require too many training examples.
  - If η is large, the update may overfit the most recent training example, and overall **w** may fluctuate too much from one update to the next.
- Unfortunately, picking a good η is more of an art than a science, and involves trial-and-error.

Iterative Reweighted Least Squares

- Remember, we want the value of **w** that miminizes $E(\boldsymbol{w})$, which is defined as:

$$E(w) = -\sum_{n=1}^{n} [t_n \ln(y_n) + (1 - t_n) \ln(1 - y_n))]$$

- It can be proven that $E(\boldsymbol{w})$ has a unique minimum.
- There is no closed form solution for that minimum.
- However, there is an iterative method mimimizing $E(\boldsymbol{w})$.
- This method is called **iterative reweighted least squares**.

Iterative Reweighted Least Squares

- **Iterative reweighted least squares** is an application of a more general optimization method, called **Newton-Raphson**.
- In the Newton-Raphson method, to minimize $E(\boldsymbol{w})$ we do a sequence of updates on the value of $\boldsymbol{w}$, using this formula:

$$\boldsymbol{w}^{(new)} = \boldsymbol{w}^{(old)} - \boldsymbol{H}^{-1}\nabla E\left(\boldsymbol{w}^{(old)}\right)$$

- In the above formula, $\boldsymbol{H}$ is the **Hessian** matrix, whose elements are the second derivatives of $E(\boldsymbol{w})$ with respect to $\boldsymbol{w}$.

Iterative Reweighted Least Squares

- To minimize $E(\boldsymbol{w})$ we iteratively update $\boldsymbol{w}$, using this formula:

$$w^{(new)} = w^{(old)} - \boldsymbol{H}^{-1}\nabla E(\boldsymbol{w})$$

- To use the above formula, we must compute matrix $\boldsymbol{H}$, which is the **Hessian** matrix. Its elements are the second derivatives of $E(\boldsymbol{w})$ with respect to $\boldsymbol{w}$.
- We have already seen that $\nabla E(\boldsymbol{w}) = \sum_{n=1}^{n}[(y_n - t_n)\varphi_n]$.
- This can be rewritten as: $\nabla E(\boldsymbol{w}) = \boldsymbol{\Phi}^{\mathrm{T}}(\boldsymbol{y} - \boldsymbol{t})$, where:

$$\boldsymbol{\Phi} = \begin{bmatrix} \varphi_0(x_1), \ldots, \varphi_D(x_1) \\ \varphi_0(x_2), \ldots, \varphi_D(x_2) \\ \ldots \\ \varphi_0(x_N), \ldots, \varphi_D(x_N) \end{bmatrix} \qquad \mathbf{t} = \begin{bmatrix} t_1 \\ t_2 \\ \ldots \\ t_N \end{bmatrix} \qquad \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \ldots \\ y_N \end{bmatrix}$$

Iterative Reweighted Least Squares

- So, we have:

$$\nabla E(\boldsymbol{w}) = \boldsymbol{\Phi}^{\mathrm{T}}(\boldsymbol{y} - \boldsymbol{t})$$

$$H = \nabla\nabla E(\boldsymbol{w}) = \boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{R}\boldsymbol{\Phi}$$

- In the above formula, we have introduced a new symbol $\boldsymbol{R}$, defined as a diagonal matrix as follows:
  - If $i \neq j$, then $R_{ij} = 0$, where R_{ij} is the value of $\boldsymbol{R}$ at row i and column j.
  - Every diagonal value R_{nn} is defined as $R_{nn} = y_n(1 - y_n)$

Iterative Reweighted Least Squares

- Putting all those formulas together, we get:

$$\nabla E(\boldsymbol{w}) = \boldsymbol{\Phi}^{\mathrm{T}}(\boldsymbol{y} - \boldsymbol{t})$$

$$H = \nabla\nabla E(\boldsymbol{w}) = \boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{R}\boldsymbol{\Phi}$$

$$w^{(new)} = w^{(old)} - (\boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{R}\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^{\mathrm{T}}(\boldsymbol{y} - \boldsymbol{t})$$

- This is now a closed form solution for how to iteratively update $\boldsymbol{w}$.
  - You stop the iterative updating when the value of $\boldsymbol{w}$ converges, and changes little or not at all from one iteration to the next.

Logistic Regresssion: Recap

- Classification function: $\mathrm{y}(\mathrm{x}) = \sigma\left(\boldsymbol{w}^{\mathrm{T}}\varphi(x)\right)$
  - We treat $\mathrm{y}(\mathrm{x})$ as an estimate of probability $\mathrm{p}(C_1|x)$.
- Error measure: $E(\boldsymbol{w}) = -\ln(p(\boldsymbol{t}\,|X, \boldsymbol{w}))$
- There is no closed form formula for finding the best $\boldsymbol{w}$.
- However, a unique best $\boldsymbol{w}$ exists, and can be found using iterative reweighted least squares:

$$w^{(new)} = w^{(old)} - \left(\boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{R}\boldsymbol{\Phi}\right)^{-1}\boldsymbol{\Phi}^{\mathrm{T}}(\boldsymbol{y} - \boldsymbol{t})$$

- We can also minimize $\mathrm{E}(\boldsymbol{w})$ using sequential learning:

$$\boldsymbol{w}^{(\tau+1)} = \boldsymbol{w}^{(\tau)} - \eta\nabla E_n(\boldsymbol{w}) = \boldsymbol{w}^{(\tau)} - \eta(y_n - t_n)\varphi(x_n)$$

Fisher's Linear Discriminant

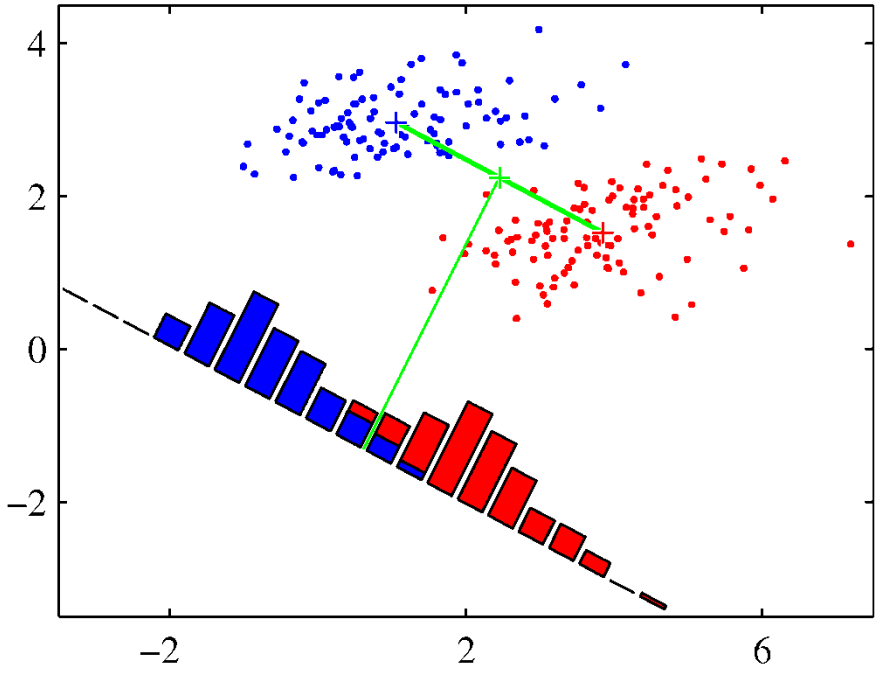
- Remember, in linear classification, the decision boundary is defined by equation:

$$\boldsymbol{w}^T x = 0$$

  - The formula above is the simple version, without using basis functions.
- Let's define function $y(\boldsymbol{x}) = \boldsymbol{w}^T \boldsymbol{x}$.
- Geometrically, what does function $y(\boldsymbol{x})$ do?
  - $y(\boldsymbol{x})$: $\mathbb{R}^{\boldsymbol{D}} \to \mathbb{R}$
  - $y(\boldsymbol{x})$ maps (projects) D-dimensional vectors $\boldsymbol{x}$ to points on a line.
- If $y(\boldsymbol{x}) > \boldsymbol{0}$ then $\boldsymbol{x}$ is classified as belonging to class C_1.
- If $y(\boldsymbol{x}) < \boldsymbol{0}$ then $\boldsymbol{x}$ is classified as belonging to class C_0.

Fisher's Linear Discriminant

- Let's define function $y(\boldsymbol{x}) = \boldsymbol{w}^T \boldsymbol{x}$.
- $y(\boldsymbol{x})$ maps (projects) D-dimensional vectors $\boldsymbol{x}$ to points on a line.
  - If $y(\boldsymbol{x}) > \mathbf{0}$ then $\boldsymbol{x}$ is classified as belonging to class C_1.
  - If $y(\boldsymbol{x}) < \mathbf{0}$ then $\boldsymbol{x}$ is classified as belonging to class C_0.
- The figure shows an example:
  - Input points $\boldsymbol{x}$ are projected on a line.
  - If they project onto one side of the line, they are classified as blue.
  - If they project on the other side, they are classified as red.
- In this particular example, some points are misclassified.

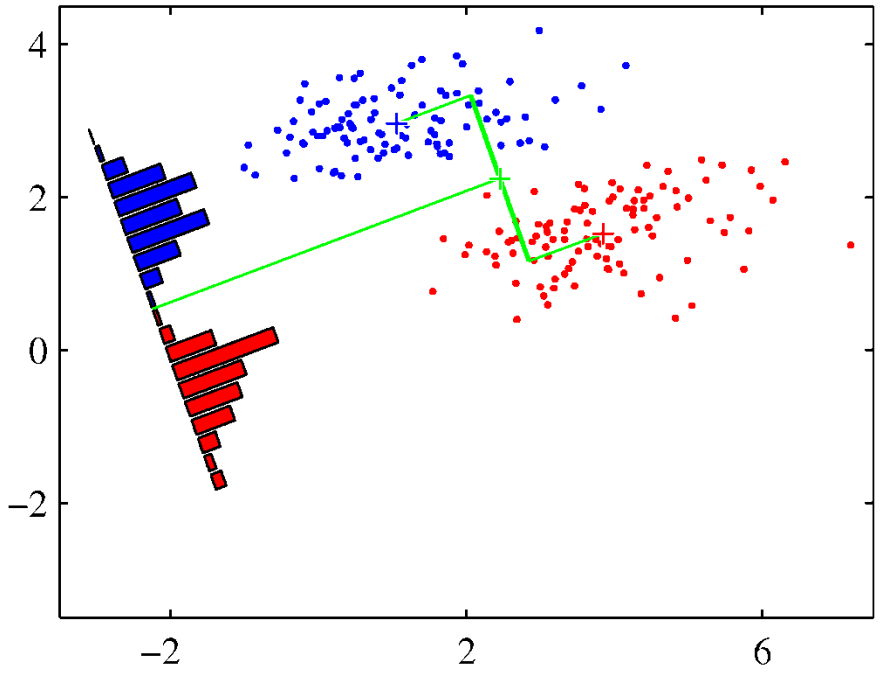


Fisher's Linear Discriminant

- Let's define function $y(\boldsymbol{x}) = \boldsymbol{w}^T \boldsymbol{x}$.
- $y(\boldsymbol{x})$ maps (projects) D-dimensional vectors $\boldsymbol{x}$ to points on a line.
  - If $y(\boldsymbol{x}) > \mathbf{0}$ then $\boldsymbol{x}$ is classified as belonging to class C_1.
  - If $y(\boldsymbol{x}) < \mathbf{0}$ then $\boldsymbol{x}$ is classified as belonging to class C_0.
- What $\boldsymbol{w}$ is the best solution?
  - Logistic regression provided one answer, by minimizing a specific error measure:

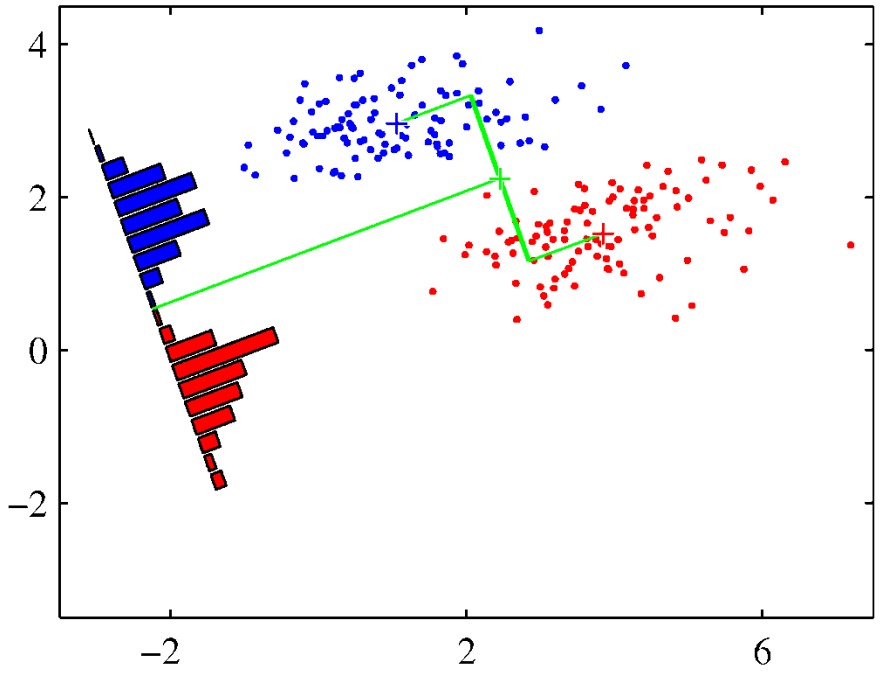
$$E(\boldsymbol{w}) = -\ln\big(p(\boldsymbol{t}\,|X, \boldsymbol{w})\big)$$

  - Fisher's linear discriminant is an alternative method, that computes a value for $\boldsymbol{w}$ using a different criterion.



Fisher's Linear Discriminant

- Goal in Fisher's Linear Discriminant: find the line projection that maximizes the separation of the classes.
- Key question: how do we measure separation of the classes?
- One simple (but not very useful) answer: maximize the separation of the means of the classes.

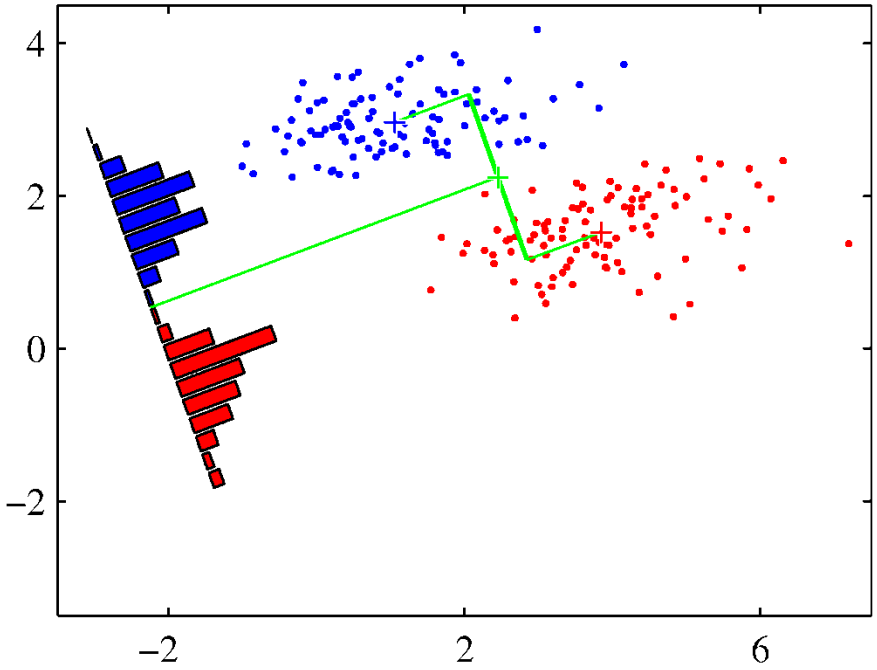


Maximizing Separation of Means

- One simple (but not very useful) answer: maximize the separation of the means of the classes.

$$\boldsymbol{m}_0 = \frac{1}{N_0}\sum_{n\in C_0} x_n$$

$$\boldsymbol{m}_1 = \frac{1}{N_1}\sum_{n\in C_1} x_n$$



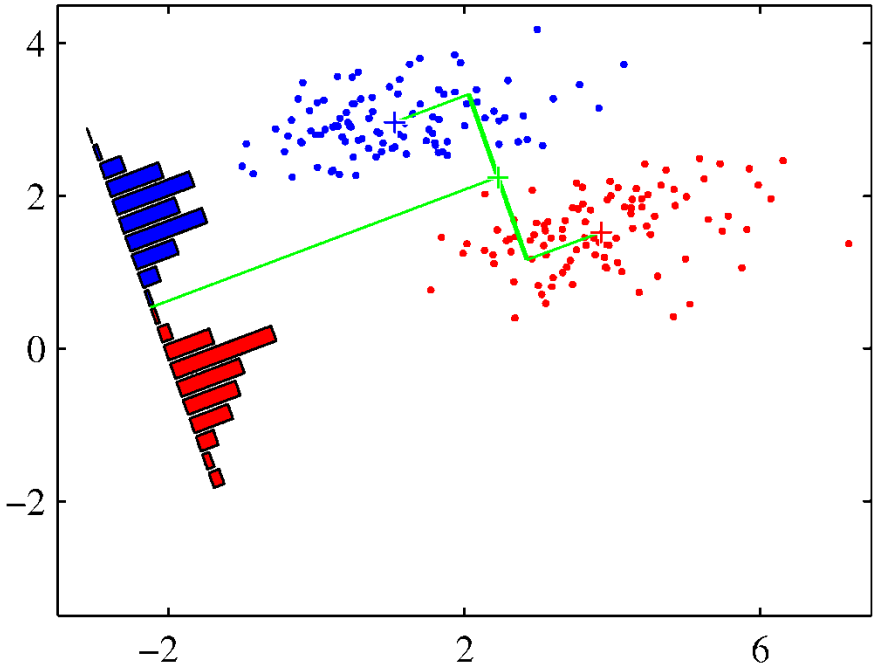
- Goal: maximize $\boldsymbol{w}^T\boldsymbol{m}_1 - \boldsymbol{w}^T\boldsymbol{m}_0$
- Same as maximizing $\boldsymbol{w}^T(\boldsymbol{m}_1 - \boldsymbol{m}_0)$
- Do you see any problem with this goal?

Maximizing Separation of Means

- One simple (but not very useful) answer: maximize the separation of the means of the classes.

$$\boldsymbol{m}_0 = \frac{1}{N_0}\sum_{n\in C_0} x_n$$

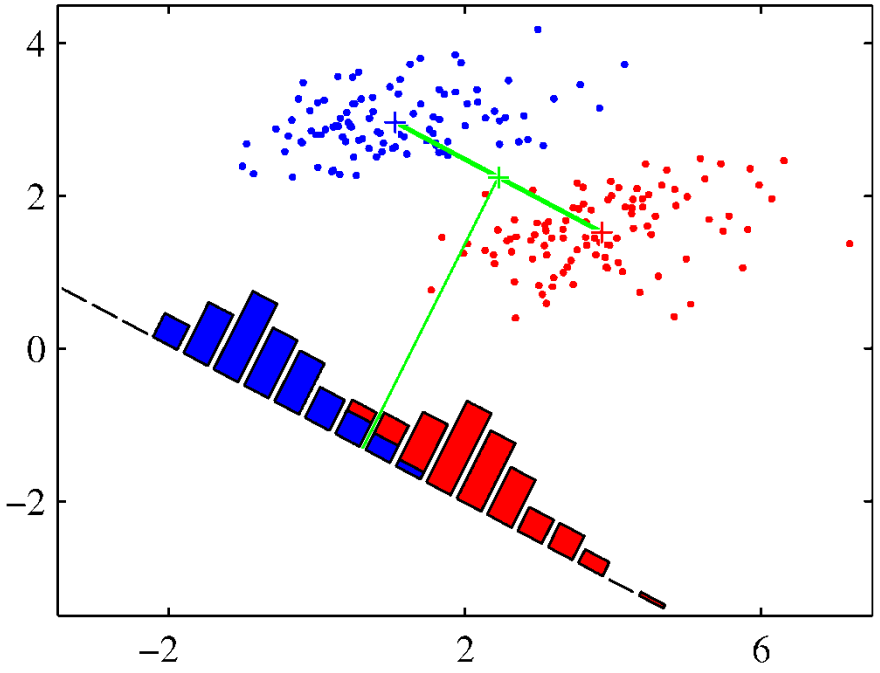
$$\boldsymbol{m}_1 = \frac{1}{N_1}\sum_{n\in C_1} x_n$$

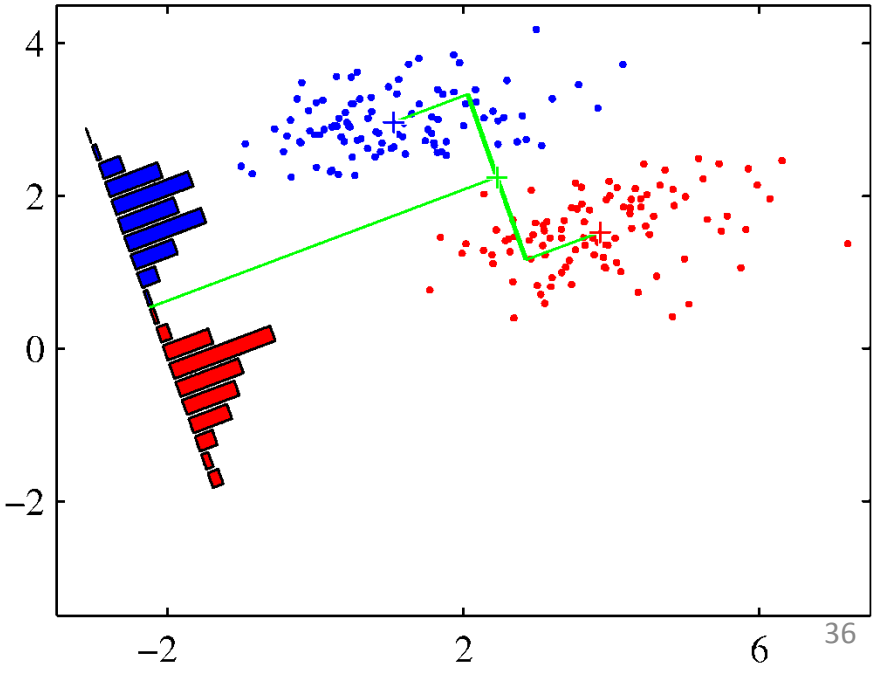


- Goal: maximize $\boldsymbol{w}^T\boldsymbol{m}_1 - \boldsymbol{w}^T\boldsymbol{m}_0$
- Same as maximizing $\boldsymbol{w}^T(\boldsymbol{m}_1 - \boldsymbol{m}_0)$
- For any $\boldsymbol{w}$, replacing $\boldsymbol{w}$ by $1000 * \boldsymbol{w}$, we increase the separation of the means by a factor of 1000, but classification accuracy stays the same.
- Maximizing $\boldsymbol{w}^T(\boldsymbol{m}_1 - \boldsymbol{m}_0)$ does not help improve accuracy.

Maximizing Separation of Means

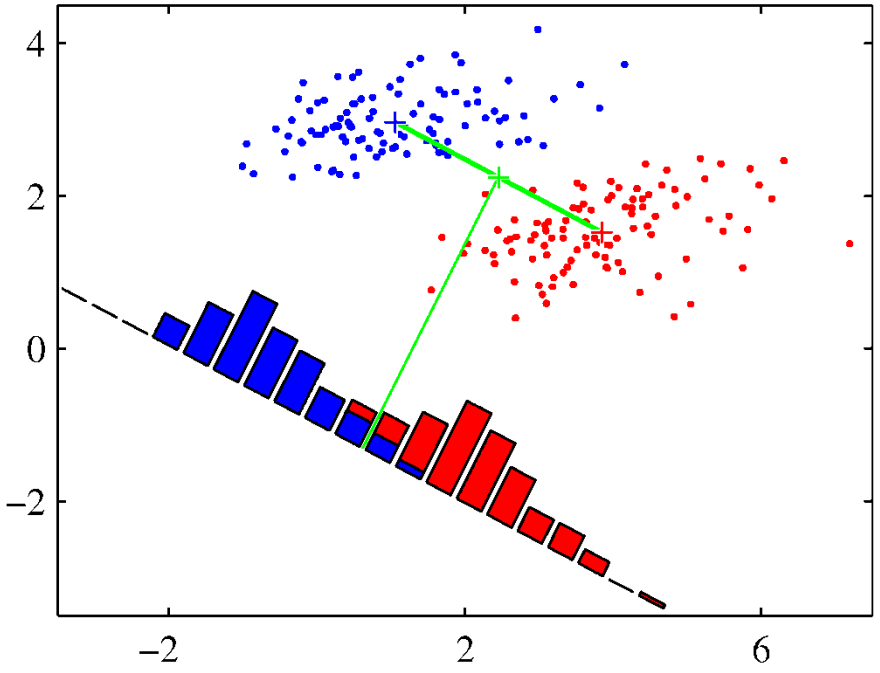
- A second not very useful answer: maximize the separation of the means of the classes, subject to $\|\boldsymbol{w}\| = 1$.
- Now we cannot replace $\boldsymbol{w}$ by $1000 * \boldsymbol{w}$, because that would violate the constraint that $\|\boldsymbol{w}\| = 1$.

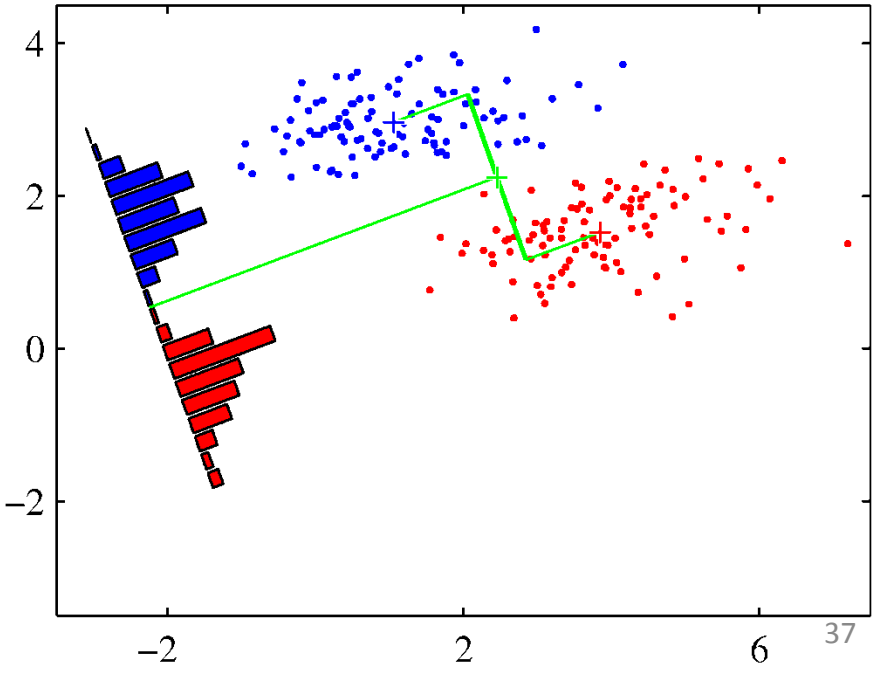




Maximizing Separation of Means

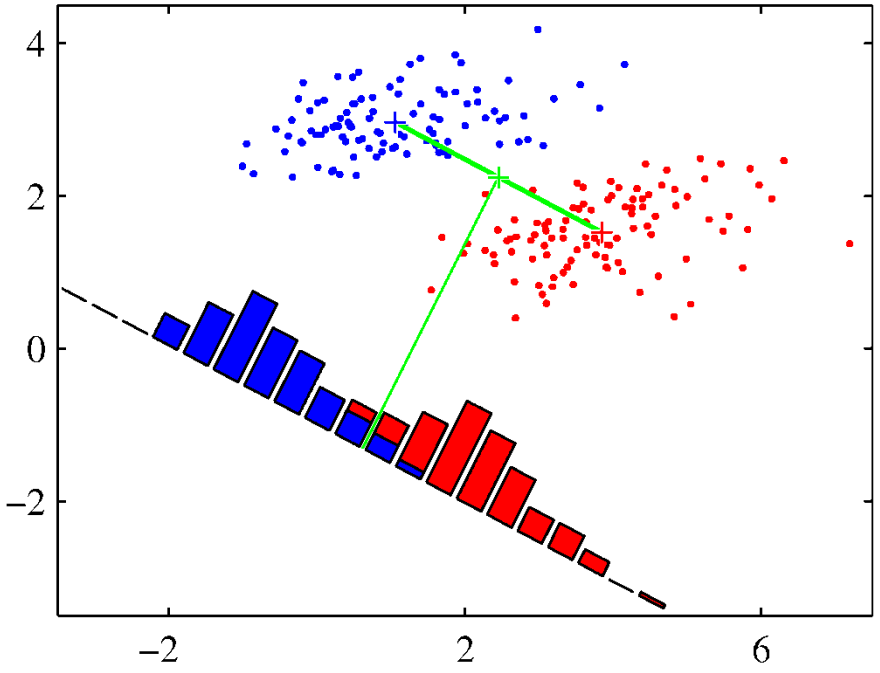
- A second not very useful answer: maximize the separation of the means of the classes, subject to $\|\boldsymbol{w}\| = 1$.
- The solution according to this criterion is shown on the left figure.
  - Do you see any problems?

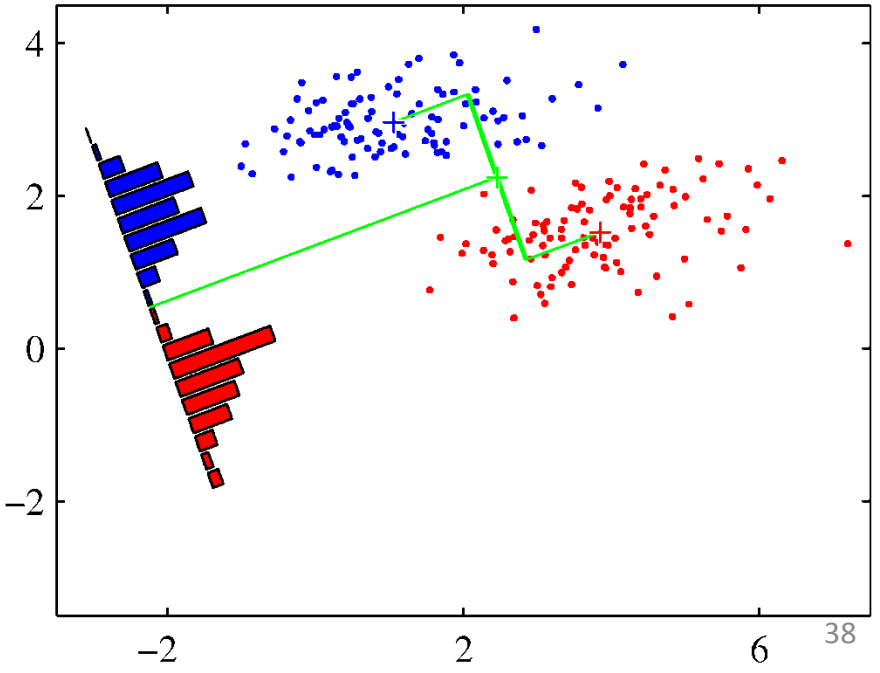




Maximizing Separation of Means

- A second not very useful answer: maximize the separation of the means of the classes, subject to $\|\boldsymbol{w}\| = 1$.
- The solution according to this criterion is shown on the left figure.
  - The projection on the right figure gives more accurate classification.
  - Our criterion fails to identify that far better solution.





Between Class Variance, Within-Class Variance,

- $\boldsymbol{w}^T(\boldsymbol{m}_1 - \boldsymbol{m}_0)$ is called the **between-class variance**.
  - It measures how far (on average) a point from class C_0 is from a point in class C_1.
- We can also define the **within-class variance** $(s_k)^2$:
  - How far (on average) a point from class C_k is from another point in the same class C_k.

$$(s_k)^2 = \sum_{n \in C_k} (\boldsymbol{w}^T \boldsymbol{x}_n - \boldsymbol{w}^T \boldsymbol{m}_k)^2$$

Fisher Criterion

- Fisher criterion: maximize the ratio $J(\boldsymbol{w})$ of between-class variance over within-class variance.

$$J(\boldsymbol{w}) = \frac{(\boldsymbol{w}^T \boldsymbol{m}_1 - \boldsymbol{w}^T \boldsymbol{m}_0)^2}{(s_0)^2 + (s_1)^2}$$

- Intuition: we want, at the same time, a projection where:
  - Between-class variance is large (inputs belonging to different classes should map to points that are far from each other).
  - Within-class variance is small (inputs belonging to the same class should map to points that are close to each other).

Maximizing $J(\boldsymbol{w})$

- We will use the following definitions:
  - $\boldsymbol{S}_B$ is the **between-class covariance matrix**.

$$\boldsymbol{S}_B = (\boldsymbol{m}_1 - \boldsymbol{m}_0)(\boldsymbol{m}_1 - \boldsymbol{m}_0)^T$$

  - $\boldsymbol{S}_W$ is the total **within-class covariance matrix**.

$$\boldsymbol{S}_W = \left(\sum_{n \in C_0} (\boldsymbol{x}_n - \boldsymbol{m}_0)(\boldsymbol{x}_n - \boldsymbol{m}_0)^T\right) + \left(\sum_{n \in C_1} (\boldsymbol{x}_n - \boldsymbol{m}_1)(\boldsymbol{x}_n - \boldsymbol{m}_1)^T\right)$$

  - Then: $J(\boldsymbol{w}) = \dfrac{\boldsymbol{w}^T \boldsymbol{S}_B \boldsymbol{w}}{\boldsymbol{w}^T \boldsymbol{S}_W \boldsymbol{w}}$

Maximizing $J(\boldsymbol{w})$

- We want to find the $\boldsymbol{w}$ that maximizes:

$$J(\boldsymbol{w}) = \frac{\boldsymbol{w}^T \boldsymbol{S}_B \boldsymbol{w}}{\boldsymbol{w}^T \boldsymbol{S}_W \boldsymbol{w}}$$

- Differentiating $J(\boldsymbol{w})$ with respect to $\boldsymbol{w}$, and solving equation $\nabla J(\boldsymbol{w}) = \boldsymbol{0}$, we get this solution:

$$\boldsymbol{w} \propto (\boldsymbol{S}_W)^{-1}(\boldsymbol{m}_1 - \boldsymbol{m}_0)$$

- The symbol $\propto$ means that $\boldsymbol{w}$ is proportional to $(\boldsymbol{S}_W)^{-1}(\boldsymbol{m}_1 - \boldsymbol{m}_0)$.
- Therefore, $\boldsymbol{w} = (\boldsymbol{S}_W)^{-1}(\boldsymbol{m}_1 - \boldsymbol{m}_0)$ is a solution.
- For any real number c, $\boldsymbol{w} = c(\boldsymbol{S}_W)^{-1}(\boldsymbol{m}_1 - \boldsymbol{m}_0)$ is also a solution.