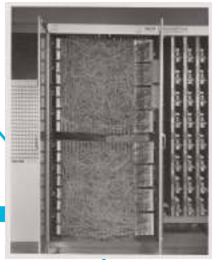
Week 5 Linear Classifiers

Pattern Recognition (Technische Universiteit Delft)

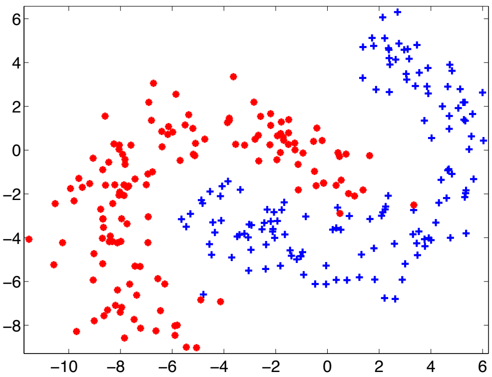
Linear Classifiers

David M.J. Tax

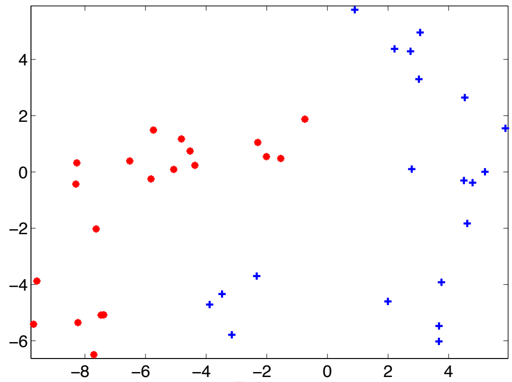


Peer review

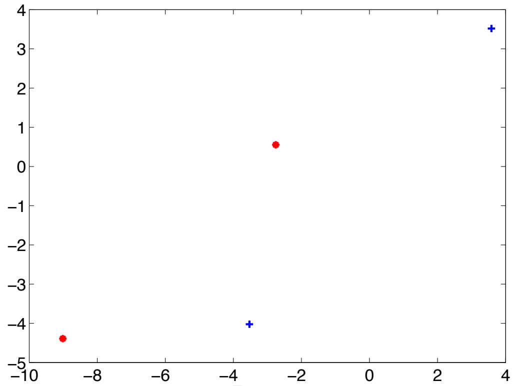
- Deadline is **tonight**, 23:59
- Please give:
  - Textual feedback
  - 'Slider' feedback



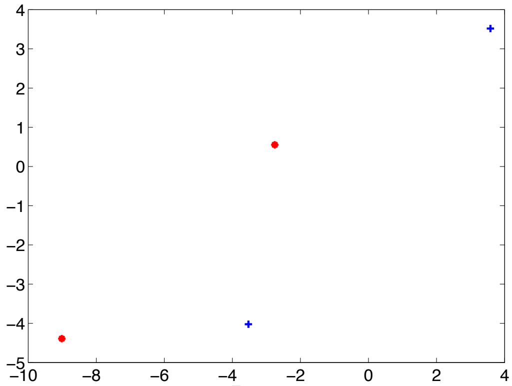
- Density estimation!! $p(\mathbf{x}|\omega)$



- Less samples: density estimation?



- What to do here?



- What would you use: `ldc, qdc, nmc, parzenc, knnc, naivebc, ...?`

Linear classifiers

- Linear discriminants, classifiers that do not do density estimation:
  - Perceptron
  - Fisher classifier
  - Logistic classifier
  - Least squares
  - (Next week: support vector classifier)
- Bias-variance dilemma

Linear discriminant

- Let us assume that the decision boundary can be described by:

$$g(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + w_0 = 0$$

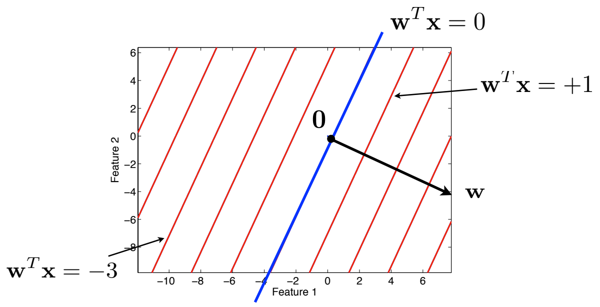
- Weight vector $\mathbf{w}$ and bias term (offset) w_0
- Classify:

$$\text{classify } \mathbf{x} \text{ to} \begin{cases} \omega_1 & \text{if } \mathbf{w}^T\mathbf{x} + w_0 \geq 0 \\ \omega_2 & \text{if } \mathbf{w}^T\mathbf{x} + w_0 < 0 \end{cases}$$

- In the most general sense, this is called linear discriminant analysis

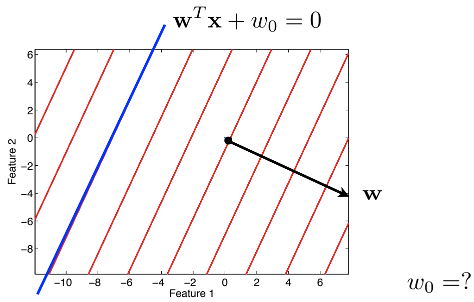
Linear function?

- What does $\mathbf{w}^T\mathbf{x}$ mean?
  Assume I have a 2-dimensional $\mathbf{w}$

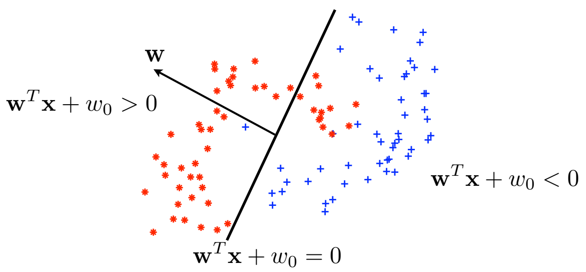


Linear function, the bias

- What does $\mathbf{w}^T\mathbf{x} + w_0$ mean?
  Assume I have



Linear discriminant



- Classifier is a linear function of the features
- The classification depends if the weighted sum of the features is above or below 0

Incorporate the bias term

- Quite often you see

$$g(\mathbf{x}) = \mathbf{w}^T\mathbf{x} > 0$$

instead of

$$g(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + w_0 > 0$$

- No problem, if you (re-)define the feature vector as: (homogeneous coordinates)

$$\tilde{\mathbf{x}} = \begin{bmatrix} \mathbf{x} \\ 1 \end{bmatrix}$$

- Then:

$$g(\mathbf{x}) = [\mathbf{w}^T \ w_0] \begin{bmatrix} \mathbf{x} \\ 1 \end{bmatrix} = \tilde{\mathbf{w}}^T\tilde{\mathbf{x}}$$

The nearest mean

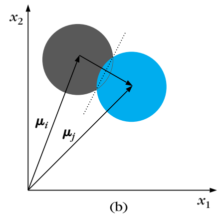
nmc

- How to find $\mathbf{w}, w_0$?
- In week 3 we assume a Gaussian distribution per class. When we assume $\Sigma = \sigma^2 I$, we find:

$$\mathbf{w} = \boldsymbol{\mu}_2 - \boldsymbol{\mu}_1$$

$$w_0 = \text{bla bla}$$

- Picture:



The perceptron algorithm

- Another way to find $\mathbf{w}, w_0$?
- First **assume** that the (two) classes are linearly separable. So there is an optimal $\mathbf{w}^*$:

$$\mathbf{w}^{*T}\mathbf{x} > 0 \quad \forall \mathbf{x} \in \omega_1 \quad (y = +1)$$
$$\mathbf{w}^{*T}\mathbf{x} < 0 \quad \forall \mathbf{x} \in \omega_2 \quad (y = -1)$$

- Define the perceptron error/loss:

$$J(\mathbf{w}) = \sum_{\text{misclassified } \mathbf{x}_i} -y_i \mathbf{w}^T \mathbf{x}_i$$

Cost function optimization

- Assume I have a cost function $J(\boldsymbol{\theta})$ how to minimize this function?

$$J(\mathbf{w}) = \sum_{\text{misclassified } \mathbf{x}_i} -y_i \mathbf{w}^T \mathbf{x}_i$$

Cost function optimization

- Assume I have a cost function $J(\boldsymbol{\theta})$ how to minimize this function?
(1) Set the derivative to 0, and solve:

$$\frac{\partial J(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} = 0$$

(typically hard/impossible to do)

(2) Follow the gradient until you hit a (local) minimum:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \rho \frac{\partial J(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}}$$

(ρ is called the learning rate)

Gradient descent

- Find the weights that minimize the loss:

$$J(\mathbf{w}) = \sum_{\text{misclassified } \mathbf{x}_i} -y_i \mathbf{w}^T \mathbf{x}_i$$

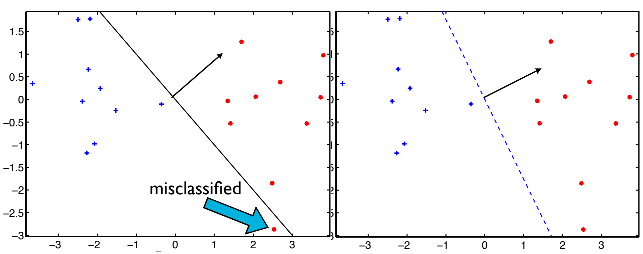
- Update the weight by:

$$\mathbf{w}(t+1) = \mathbf{w}(t) - \rho_t \left.\frac{\partial J(\mathbf{w})}{\partial \mathbf{w}}\right|_{\mathbf{w}=\mathbf{w}(t)}$$

- gives:

$$\mathbf{w}(t+1) = \mathbf{w}(t) + \rho_t \sum_{\text{misclassified } \mathbf{x}_i} y_i \mathbf{x}_i$$

Perceptron optimization



$$\mathbf{w}_0 = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \qquad \mathbf{w}_1 = \begin{bmatrix} 1.25 \\ 0.72 \end{bmatrix}$$

- One erroneous object at (2.5, -2.8)
- Learning rate $\rho = 0.1$

The perceptron

perlc

- Just a 'simple' linear classifier
- ✓ Is trained incrementally, or in batches (applicable to very large datasets...)
- ✓ When the data is separable, it will find the solution (proof: see book!)
- When the data is not separable, it will update for ever, and ever, and ever...

- Perceptron is the basis for the neural networks
- Different variants available, inspired by the 'real' perceptron

General idea

- Invent a general model/function for the classifier:

$$g(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + w_0$$

- Invent a loss function:

$$J(\mathbf{w}) = \sum_{\text{misclassified } \mathbf{x}_i} -y_i \mathbf{w}^T \mathbf{x}_i$$

- Optimise the parameters **w** to optimise J.

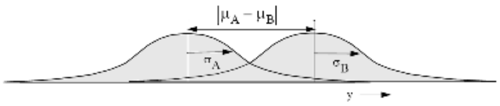
Fisher Linear Discriminant

- Again, two-class problem and a linear classifier

$$g(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + w_0 = 0$$

- Find the weights such that separability is maximised: **Fisher's criterion**:

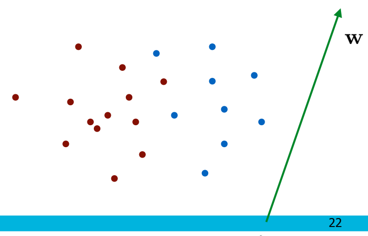
$$J_F = \frac{\sigma^2_{\text{between}}}{\sigma^2_{\text{within}}} = \frac{|\mu_A - \mu_B|^2}{\sigma_A^2 + \sigma_B^2}$$



Fisher Linear Discriminant

- Fisher criterion along the direction w:

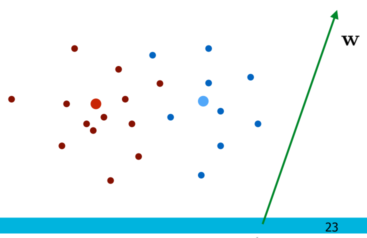
$$J_F = \frac{|\mathbf{w}^T\mu_A - \mathbf{w}^T\mu_B|^2}{\mathbf{w}^T\Sigma_A\mathbf{w} + \mathbf{w}^T\Sigma_B\mathbf{w}}$$



Fisher Linear Discriminant

- Fisher criterion along the direction w:

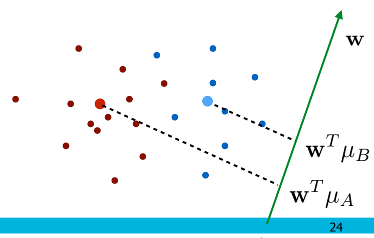
$$J_F = \frac{|\mathbf{w}^T\mu_A - \mathbf{w}^T\mu_B|^2}{\mathbf{w}^T\Sigma_A\mathbf{w} + \mathbf{w}^T\Sigma_B\mathbf{w}}$$



Fisher Linear Discriminant

- Fisher criterion along the direction w:

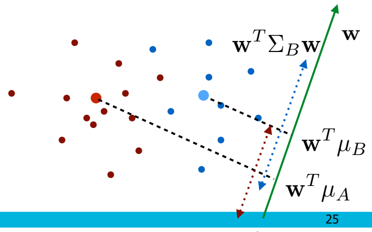
$$J_F = \frac{|\mathbf{w}^T\mu_A - \mathbf{w}^T\mu_B|^2}{\mathbf{w}^T\Sigma_A\mathbf{w} + \mathbf{w}^T\Sigma_B\mathbf{w}}$$



Fisher Linear Discriminant

- Fisher criterion along the direction w:

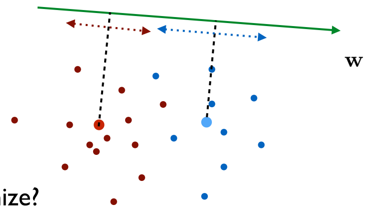
$$J_F = \frac{|\mathbf{w}^T\mu_A - \mathbf{w}^T\mu_B|^2}{\mathbf{w}^T\Sigma_A\mathbf{w} + \mathbf{w}^T\Sigma_B\mathbf{w}}$$



Fisher Linear Discriminant

- Fisher criterion along the direction w:

$$J_F = \frac{|\mathbf{w}^T\mu_A - \mathbf{w}^T\mu_B|^2}{\mathbf{w}^T\Sigma_A\mathbf{w} + \mathbf{w}^T\Sigma_B\mathbf{w}}$$



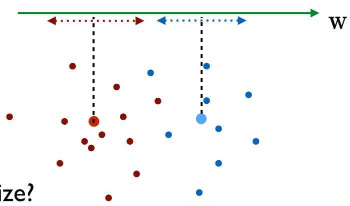
- How to optimize?

Fisher Linear Discriminant

- Fisher criterion along the direction w:

$$J_F = \frac{|\mathbf{w}^T\mu_A - \mathbf{w}^T\mu_B|^2}{\mathbf{w}^T\Sigma_A\mathbf{w} + \mathbf{w}^T\Sigma_B\mathbf{w}} = \frac{\mathbf{w}^T(\mu_A-\mu_B)(\mu_A-\mu_B)^T\mathbf{w}}{\mathbf{w}^T(\Sigma_A+\Sigma_B)\mathbf{w}}$$

$$= \frac{\mathbf{w}^T\Sigma_{\text{between}}\mathbf{w}}{\mathbf{w}^T\Sigma_{\text{within}}\mathbf{w}}$$



- How to optimize?

Optimal weight?

fisherc

- To optimise $J_F = \dfrac{\mathbf{w}^T\Sigma_{\text{between}}\mathbf{w}}{\mathbf{w}^T\Sigma_{\text{within}}\mathbf{w}}$

- Take the derivative with respect to the weight,
- set derivative to zero
- solve!

- The solution: $\mathbf{w} = \Sigma_W^{-1}(\mu_A - \mu_B)$

- So, the classifier becomes

$$g(\mathbf{x}) = \mathbf{x}^T\mathbf{w} + w_0 = \mathbf{x}^T\Sigma_W^{-1}(\mu_A-\mu_B) + w_0$$

Fisher and LDA

- Wait, this classifier: (Fisher classifier)

$$g(\mathbf{x}) = \mathbf{x}^T\Sigma_W^{-1}(\mu_A-\mu_B) + w_0$$

looks a lot like: (Linear discriminant analysis) (cf. pg. 27 book)

$$f(\mathbf{x}) = (\mu_2-\mu_1)^T\Sigma^{-1}\mathbf{x} + w_0$$

- One optimises the Fisher criterion, the other assumes a Gaussian distribution per class, and equal covariance matrices

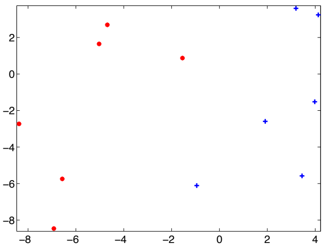
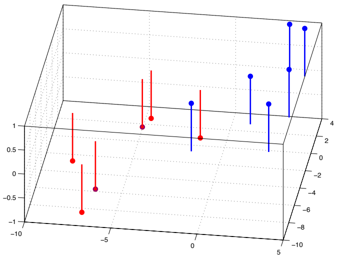
- But the solution is the same…

Comparison Fisher and Linear discriminant

- The normal-based linear classifier assumes a density per class,
  the Fisher classifier just tries to optimize the Fisher criterion
- For the Fisher classifier the bias term is (in principle) still free to optimize.

- Both classifiers rely on the inverse of S_W, so it can therefore become undefined when insufficient data is available.

Least squares

- Consider classification as a regression problem
- Fit a function to predict the label:

$$y = +1/-1$$

Least squares

- Define the cost function:

$$J(\mathbf{w}) = E[|y - \mathbf{w}^T \mathbf{x}|^2]$$

- The expectation E[.] is over the joint pdf of $(\mathbf{x}, y)$
$p(\mathbf{x}, y)$
- This means: how good does $\mathbf{w}^T \mathbf{x}$ predict the label?
- Use the definition of E[.] to derive:

$$\hat{\mathbf{w}} = R_x^{-1} E[\mathbf{x}y]$$

Auto- and cross-correlation

- The correlation R is the *correlation matrix*:

$$R_x = E[\mathbf{x}\mathbf{x}^T] = \begin{bmatrix} E[x_1 x_1] & \dots & E[x_1 x_d] \\ E[x_2 x_1] & \dots & E[x_2 x_d] \\ \vdots & \vdots & \vdots \\ E[x_d x_1] & \dots & E[x_d x_d] \end{bmatrix}$$

for an d-dimensional dataset…

- The cross-correlation is

$$E[\mathbf{x}y] = E\left[\begin{bmatrix} x_1 y \\ x_2 y \\ \vdots \\ x_d y \end{bmatrix}\right]$$

In real life…

- For the correlation, and cross-correlation, we need the true distribution $p(\mathbf{x}, y)$.
- When we just have samples:

$$\left(\sum_{i=1}^{N} \mathbf{x}_i \mathbf{x}_i^T\right) \hat{\mathbf{w}} = \sum_{i=1}^{N} (\mathbf{x}_i y_i)$$

- So we solve

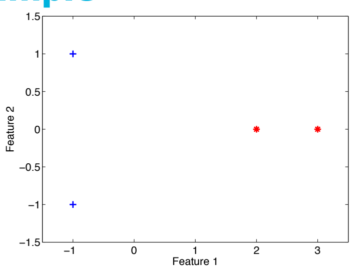
$$\hat{\mathbf{w}} = \left(\sum_{i=1}^{N} \mathbf{x}_i \mathbf{x}_i^T\right)^{-1} \sum_{i=1}^{N} (\mathbf{x}_i y_i)$$

$$\hat{\mathbf{w}} = (X^T X)^{-1} X^T \mathbf{y}$$

where X is a $N \times d$ dataset matrix

Example

$$X = \begin{bmatrix} -1 & -1 \\ -1 & +1 \\ 2 & 0 \\ 3 & 0 \end{bmatrix} \qquad \mathbf{y} = \begin{bmatrix} -1 \\ -1 \\ +1 \\ +1 \end{bmatrix}$$

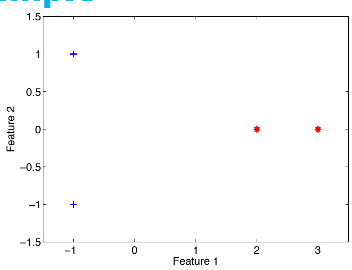


- 2D dataset, with 2 classes and 4 objects, find $\hat{\mathbf{w}}$

$$X^T X = \left(\sum_{i=1}^{N} \mathbf{x}_i \mathbf{x}_i^T\right) = ?$$

Example

$$X = \begin{bmatrix} -1 & -1 \\ -1 & +1 \\ 2 & 0 \\ 3 & 0 \end{bmatrix} \qquad \mathbf{y} = \begin{bmatrix} -1 \\ -1 \\ +1 \\ +1 \end{bmatrix}$$



- 2D dataset, with 2 classes and 4 objects, find $\hat{\mathbf{w}}$

$$X^T X = \left(\sum_{i=1}^{N} \mathbf{x}_i \mathbf{x}_i^T\right) = \begin{bmatrix} 15 & 0 \\ 0 & 2 \end{bmatrix}$$

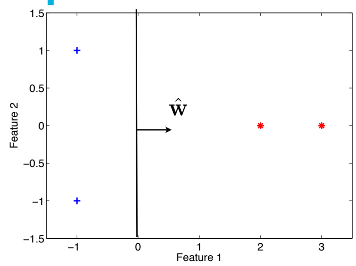
Example

- We want: $\hat{\mathbf{w}} = (X^T X)^{-1} X^T \mathbf{y}$
- $(X^T X)^{-1} = \begin{bmatrix} 15 & 0 \\ 0 & 2 \end{bmatrix}^{-1} = \begin{bmatrix} 1/15 & 0 \\ 0 & 1/2 \end{bmatrix}$
- $X^T \mathbf{y} = \begin{bmatrix} 7 \\ 0 \end{bmatrix}$
- so in total:

$$\hat{\mathbf{w}} = \begin{bmatrix} 1/15 & 0 \\ 0 & 1/2 \end{bmatrix} \begin{bmatrix} 7 \\ 0 \end{bmatrix} = \begin{bmatrix} 7/15 \\ 0 \end{bmatrix}$$

Example

$$\hat{\mathbf{w}} = \begin{bmatrix} 7/15 \\ 0 \end{bmatrix}$$



- Note that I did **not** take care for the offset w_0
- Incorporate the offset by defining new feature vectors $\tilde{\mathbf{x}} = \begin{bmatrix} \mathbf{x} \\ 1 \end{bmatrix}$ try it yourself!

Logistic classifier

- We can rewrite:

$$\ln(p(\omega_1|\mathbf{x})) - \ln(p(\omega_2|\mathbf{x})) = \ln\left(\frac{p(\omega_1|\mathbf{x})}{p(\omega_2|\mathbf{x})}\right)$$

- Assume we can approximate (our model):

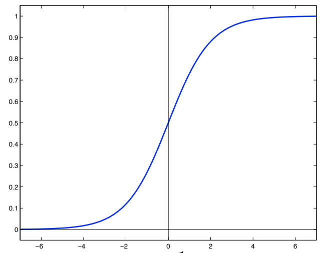
$$\ln\left(\frac{p(\omega_1|\mathbf{x})}{p(\omega_2|\mathbf{x})}\right) = \beta_0 + \beta^T \mathbf{x}$$

- The classifier becomes (lab course):

$$p(\omega_2|\mathbf{x}) = \frac{1}{1+\exp(\beta_0 + \beta^T \mathbf{x})}$$

Logistic function

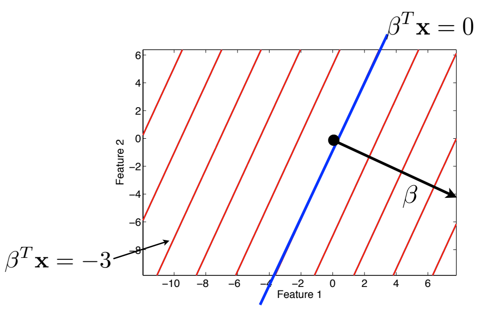
- The function looks like (for scalar x):



$$f(x) = \frac{1}{1+\exp(-x)}$$

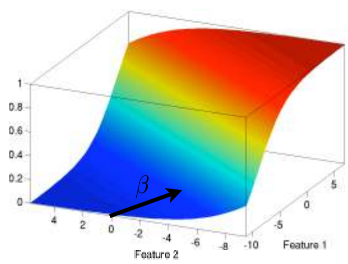
Higher dimensions

- What does $\beta^T \mathbf{x}$ mean?
Assume I have a 2-dimensional β



Logistic function

- For a 2D logistic:



Optimizing the logistic

- To optimize the parameters on a training set, maximize the likelihood

$$L = \prod_{i=1}^{n_1} p(\mathbf{x}_i^{(1)}|\omega_1) \prod_{i=1}^{n_2} p(\mathbf{x}_i^{(2)}|\omega_2)$$

where $\mathbf{x}_i^{(1)}$ is the i-th object from class ω_1.

- Maximization using gradient descent/ascent
- It appears to be easier to maximize $\ln(L)$
- The weights are iteratively updated as:

$$\beta_{new} = \beta_{old} + \eta \frac{\partial \ln(L)}{\partial \beta}$$

TUDelft

Optimizing the logistic

- Function to maximize (now using log):

$$\ln(L) = \sum_{i=1}^{n_1} \ln p(\mathbf{x}_i^{(1)}|\omega_1) + \sum_{i=1}^{n_2} \ln p(\mathbf{x}_i^{(2)}|\omega_2)$$

- Using Bayes theorem

$$\ln p(\mathbf{x}_i^{(1)}|\omega_1) = \ln p(\omega_1|\mathbf{x}_i^{(1)}) + \ln p(\mathbf{x}_i^{(1)}) - \ln p(\omega_1)$$

fixed for the given data

- Therefore:

$$\ln(L) = \sum_{i=1}^{n_1} \ln p(\omega_1|\mathbf{x}_i^{(1)}) + \sum_{i=1}^{n_2} \ln p(\omega_2|\mathbf{x}_i^{(2)}) + K$$

TUDelft

Optimizing the logistic

- Now use that

$$p(\omega_2|\mathbf{x}) = \frac{1}{1 + \exp(\beta_0 + \beta^T \mathbf{x})}$$

and fill this in:

$$\ln(L') = \sum_{r=1}^{n_1} (\beta_0 + \beta^T \mathbf{x}_r^{(1)}) - \sum_{i=1}^{n_1+n_2} \ln(1 + \exp(\beta_0 + \beta^T \mathbf{x}_i))$$

(also lab course)

TUDelft

Derivative of the ln(L)

loglc

- Take the derivative of ln(L) w.r.t. β_0, β

$$\frac{\partial \ln(L)}{\partial \beta_0} = n_1 - \sum_{i=1}^{n_1+n_2} p(\omega_1|\mathbf{x}_i)$$

$$\frac{\partial \ln(L)}{\partial \beta_j} = \sum_{i=1}^{n_1} (\mathbf{x}_i^{(1)})_j - \sum_{i=1}^{n_1+n_2} p(\omega_1|\mathbf{x}_i)(\mathbf{x}_i)_j$$

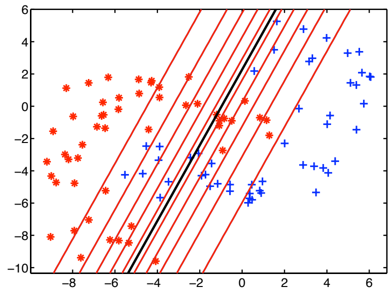
- Take initial values $\beta_0 = 0, \ \beta = \mathbf{0}$
- Keep iterating

$$\beta_{new} = \beta_{old} + \eta \frac{\partial \ln(L)}{\partial \beta}$$

till convergence

TUDelft

Result Logistic Classifier



- Logistic classifier with equal-posterior-probability lines.

TUDelft

Squared Error

- When we have the error

$$J(\mathbf{w}) = E[|\mathbf{w}^T \mathbf{x} - y|^2]$$

we can actually derive something more general...

- In general?!
- Unfortunately, theory in Pattern Recognition is tough…
how to deal with all possible datasets?

TUDelft

Bias-variance dilemma

- When we are given some data, we may get lucky, or unlucky:
  sometimes we get very a-typical data:-(
- To say something general, we need to **average** over different (training-) sets

$$\mathcal{D} = \{(y_i, \mathbf{x}_i); i = 1, .., N\}$$

- The classifier is now also a function of the training set: $g(\mathbf{x}; \mathcal{D})$

Bias-variance dilemma

- Consider the squared error:

$$E_{\mathcal{D}}\left[(g(\mathbf{x};\mathcal{D}) - E[y|\mathbf{x}])^2\right]$$

- $E[y|\mathbf{x}]$ is the optimal mean-squared regressor (not proven now; see book)
- Now we do a trick:

$$= E_{\mathcal{D}}\left[(g(\mathbf{x};\mathcal{D}) - E_{\mathcal{D}}[g(\mathbf{x};\mathcal{D})] + E_{\mathcal{D}}[g(\mathbf{x};\mathcal{D})] - E[y|\mathbf{x}])^2\right]$$

Bias-variance dilemma

- Now working out the square:

$$\begin{aligned} &= E_{\mathcal{D}}\big[(g(\mathbf{x};\mathcal{D}) - E_{\mathcal{D}}[g(\mathbf{x};\mathcal{D})])^2 \\ &\quad + 2(g(\mathbf{x};\mathcal{D}) - E_{\mathcal{D}}[g(\mathbf{x};\mathcal{D})])(E_{\mathcal{D}}[g(\mathbf{x};\mathcal{D})] - E[y|\mathbf{x}]) \\ &\quad + (E_{\mathcal{D}}[g(\mathbf{x};\mathcal{D})] - E[y|\mathbf{x}])^2\big] \end{aligned}$$

$$\begin{aligned} &= E_{\mathcal{D}}\left[(g(\mathbf{x};\mathcal{D}) - E_{\mathcal{D}}[g(\mathbf{x};\mathcal{D})])^2\right] \quad \text{(variance)} \\ &\quad + \underbrace{E_{\mathcal{D}}\left[2(g(\mathbf{x};\mathcal{D}) - E_{\mathcal{D}}[g(\mathbf{x};\mathcal{D})])\right]}_{=0}(E_{\mathcal{D}}[g(\mathbf{x};\mathcal{D})] - E[y|\mathbf{x}])] \\ &\quad + E_{\mathcal{D}}\left[(E_{\mathcal{D}}[g(\mathbf{x};\mathcal{D})] - E[y|\mathbf{x}])^2\right] \quad \text{(bias}^2\text{)} \end{aligned}$$

Bias-variance dilemma

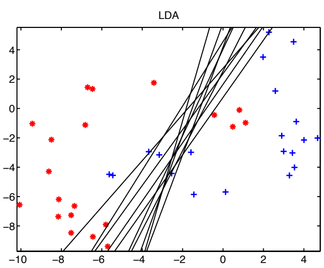
$$\begin{aligned} MSE &= E_{\mathcal{D}}\left[(g(\mathbf{x};\mathcal{D}) - E_{\mathcal{D}}[g(\mathbf{x};\mathcal{D})])^2\right] \quad \text{(variance)} \\ &\quad + E_{\mathcal{D}}\left[(E_{\mathcal{D}}[g(\mathbf{x};\mathcal{D})] - E[y|\mathbf{x}])^2\right] \quad \text{(bias}^2\text{)} \end{aligned}$$

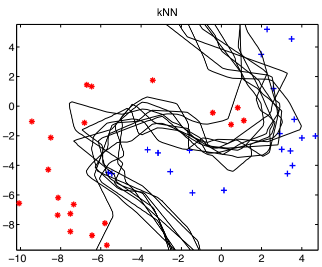
- variance: how much does classifier g vary over different training sets
- bias: how much does the average classifier g differ from the true output

Bias-variance dilemma

- Compare LDA with kNN:

low variance, high bias



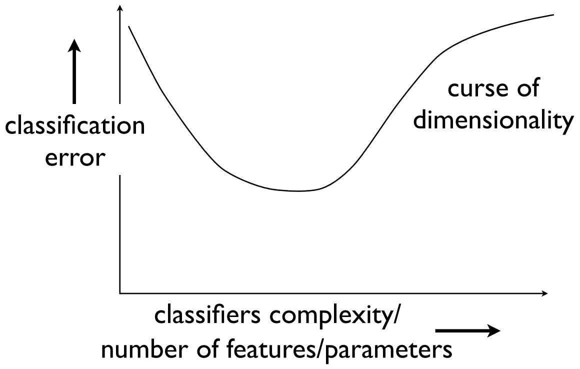


high variance, low bias

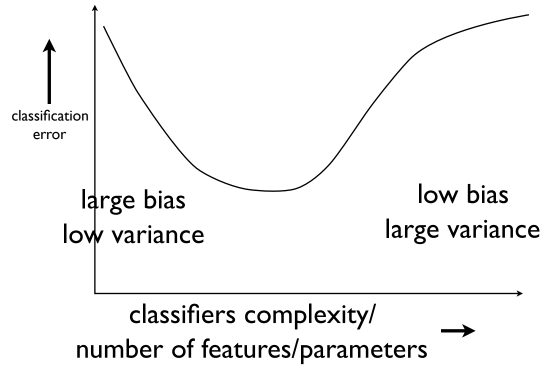
Bias-variance tradeoff

- Originally derived for neural networks
- It is a very general phenomenon: we encounter it more often in pattern recognition
- More simple classifier is more stable (and need less data to train)
- More complex classifier only works when you have sufficient number of training data

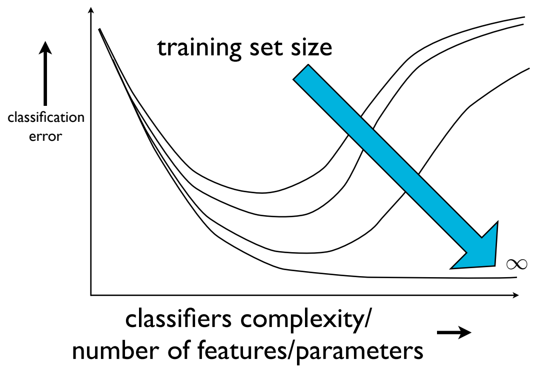
Feature curve



Feature curve



Feature curve



Conclusions

- We can make classifiers that do not depend on class densities
- There are alternative principles to find a good classifier:
  - minimising the classification error
  - minimising the mean squared error
  - maximising the likelihood
  - ... (see next week)
- There is a fundamental tradeoff between the bias and variance of a classifier (depending on how flexible/complex a classifier is)

Things to think about...

- Are the least-squares (Fisher) classifier and the perceptron (in)dependent on the class densities?
- The least-squares classifier and the perceptron are both linear classifiers: do they have the same complexity?
- How can we make a multi-class perceptron?
- How can we make a multi-class Fisher classifier?