Machine Learning

Linear Models for Classification I

Manfred Jaeger

Aalborg University

Course Logistics

Teachers

Manfred Jaeger `jaeger@cs.aau.dk`
Thomas D. Nielsen `tdn@cs.aau.dk`
Peter Dolog `dolog@cs.aau.dk`

Times

Lectures and excercises: Mondays 8:15-12:00
Self study: Wednesdays 12:30-16:15

Places

Lectures: Seminar room
Exercises and self study: Group rooms; work spaces

Course consists of 12 units:

1-3 (Manfred)

- Linear models for classification
- Support vector machines

4-7 (Thomas)

- Probabilistic graphical models
- Deep neural networks

8-11 (Peter)

- Learning with graph data
- Link prediction

12 (Manfred)

- Graph Kernels

Self studies

- Extended, applied exercises (implementation, experimentation)
- Best done in small groups (2-3 students)
- No hand-ins or deliverables
- Limited support available approximately Wed. 13:30-15:30

Oral or written exam

- Oral/written: to be determined soon
- Either way: questions at the exam about
  - methods/theory/examples discussed in the lectures
  - applications/data investigated in the self studies (not specifics of Python code)

"It should be stressed that AAU expects each student to spend 30 hours of study per ECTS credit, amounting to 150 hours"

Activity	hours
Lectures and exercises	12×4
Self studies	12×4
Reading	12×2
Exam preparation	25
Other	5
Total	150

Literature

For the first 3 lectures we will use selected chapters from:

C.M.Bishop: Pattern Recognition and Machine Learning. Springer, 2006

Reading material for following lectures announced as we go along!

Exercises

Exercises after the lectures: "theoretical" exercises that can be solved with pencil and paper (or use tools of your choice).

Self studies

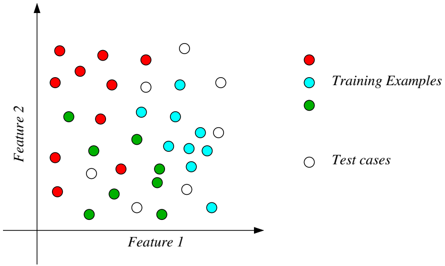
Self studies are extended practical exercises using Python and Jupyter notebooks. You should have installed:

- Python with
  - NumPy `http://www.numpy.org/`
  - Scikit-learn (`sklearn`) `http://scikit-learn.org`
  - Pandas `http://pandas.pydata.org/`
  - Matplotlib `https://matplotlib.org/`
- Jupyter (`http://jupyter.org/`)

Classification & Regression

Learning to predict:

- Based on the meterological data, is the sun shining tomorrow?
- Based on the players last moves, what is he going to do next?
- Based on a customers recent purchases, which product would he also be interested in?
- Based on the observed symptoms, what is the most likely diagnosis for this patient?

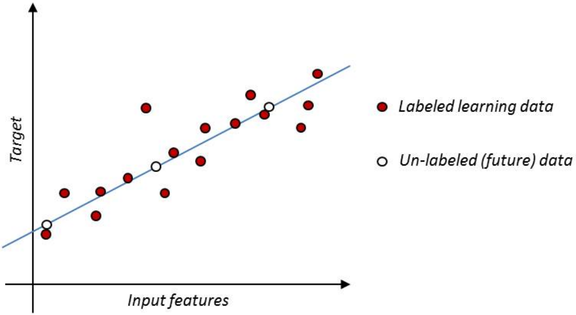


All instances (data points) have:

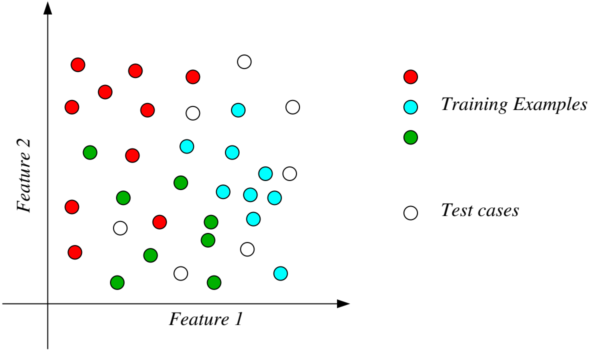
- known values for a set of **features** (a.k.a. predictors, attributes, . . .)
- a known or unknown **class label**

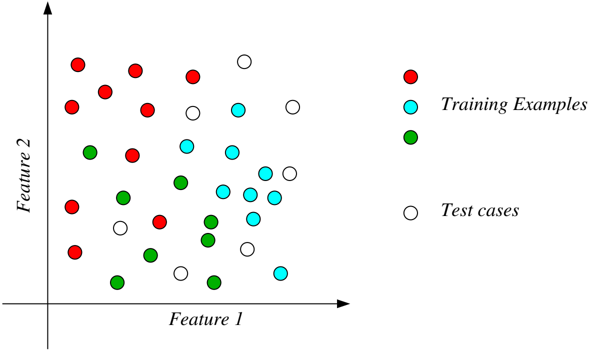
Learning to predict (continuous target):

- Based on demographic data, previous click patterns and searches; what is the CTR (click-through rate) for an online ad?
- Based on stock prices in the past; what will the price be tomorrow?
- Based on observed symptoms; how many years will a patient survive?



K Nearest Neighbor Classifier


Feature 2
Feature 1
Training Examples
Test cases



Principle: near neighbors tend to have the same label.

Data: Set of labeled training instances $(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_N, y_N)$ consisting of D-dimensional input feature (attribute) vectors $\boldsymbol{x}_i = (x_{i,1}, \ldots, x_{i,D})$, and a class label y_i.

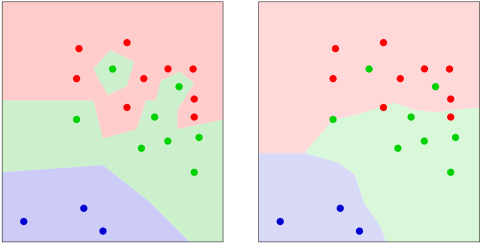
Required: distance function $d(\boldsymbol{x}, \boldsymbol{x}')$ to measure distances between attribute vectors (e.g. Euclidean distance if all attributes are real numbers, i.e., $\boldsymbol{x} \in \mathbb{R}^D$).

Classify new instance $(\boldsymbol{x}, ?)$ as follows:

- Let $(\boldsymbol{x}_{j_1}, y_{j_1}), \ldots, (\boldsymbol{x}_{j_K}, y_{j_K})$ be the K training instances whose attribute vectors are closest to $\boldsymbol{x}$.
- Predict for $(\boldsymbol{x}, ?)$ the class label that occurs most frequently among $y_{j_1}, \ldots, y_{j_K}$.

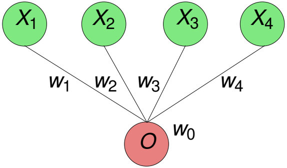
Instance space (input space): space of all possible values for the input features $\boldsymbol{x}$.

A classifier partitions the instance space into **decision regions**: each class label y defines the region of points $\boldsymbol{x}$ for which the predicted label will be y.



Decision regions (approximately) for 1-nearest neighbor (left) and 5-nearest neighbor (right).

Perceptron and Naive Bayes



Neural Network with:

- Input layer
- No hidden layers
- One output neuron
- *Sign activation function* at output neuron

Function computed:

$$O(x_1, \ldots, x_n) = \begin{cases} 1 & \text{if } w_0 + w_1 x_1 + \ldots w_n x_n > 0 \\ -1 & \text{otherwise} \end{cases}$$

The perceptron can be used to classify a binary class variable based on (numeric) predictor attributes $X_1, \ldots, X_n$.

Visualization for class label Y with values $+, -$ and $n = 2$.

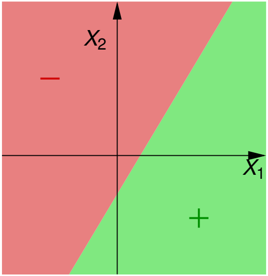
Shown are decision regions

$$\{(x_1, x_2) \mid \boldsymbol{w} \cdot \boldsymbol{x} > 0\}$$

and

$$\{(x_1, x_2) \mid \boldsymbol{w} \cdot \boldsymbol{x} \leq 0\},$$

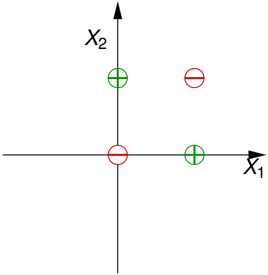
and the predicted class labels for instances in these regions.



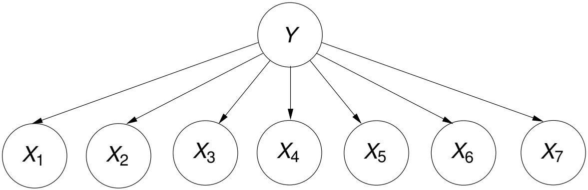
The decision regions defined by a perceptron are always separated by a **linear hyperplane**.

The perceptron can not produce the following classification (XOR of *A* and *B*):

X_1	X_2	Class
1	1	$\ominus$
1	0	$\oplus$
0	1	$\oplus$
0	0	$\ominus$



Naive Bayes Model: features are independent, given the class label:



Prediction: Binary Class/Atrributes

If Y and all X_i are binary: classify instance as $\oplus$ if

$$
\begin{aligned}
& P(\oplus \mid X_1, \ldots, X_n) \geq P(\ominus \mid X_1, \ldots, X_n) \\
\Leftrightarrow \quad & \log P(\oplus \mid X_1, \ldots, X_n) \geq \log P(\ominus \mid X_1, \ldots, X_n) \\
\Leftrightarrow \quad & \ldots \text{ [Bayes rule and then some rewriting] } \ldots \\
\Leftrightarrow \quad & \sum_{i=1}^{n} \log \frac{P(X_i{=}1 \mid \oplus) P(X_i{=}0 \mid \ominus)}{P(X_i{=}0 \mid \oplus) P(X_i{=}1 \mid \ominus)} X_i + \sum_{i=1}^{n} \frac{P(X_i{=}0 \mid \oplus)}{P(X_i{=}0 \mid \ominus)} + \log \frac{P(\oplus)}{P(\ominus)} \geq 0
\end{aligned}
$$

- Linear function in the X_i, with coefficients defined by network parameters $P(X_i = 1 \mid \oplus), \ldots$!

- Perceptron and Naive Bayes are limited
- Can not learn to classify XOR function
- More powerful types of classifiers:
  - Support vector machines
  - Neural networks with hidden layers
  - Tree-augmented Naive Bayes
  - General Bayesian Network models
  - ...

$\leadsto$ why bother with restricted classes?

- Perceptron and Naive Bayes are limited
- Can not learn to classify XOR function
- More powerful types of classifiers:
  - Support vector machines
  - Neural networks with hidden layers
  - Tree-augmented Naive Bayes
  - General Bayesian Network models
  - ...

$\rightsquigarrow$ why bother with restricted classes?

➡Still very useful baseline models:

- robust
- work with limited data

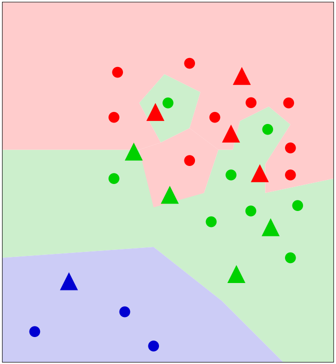
➡Integral component of more sophisticated model types:

- Basic support vector machine
- Final component (layer) of deep neural networks

Overfitting

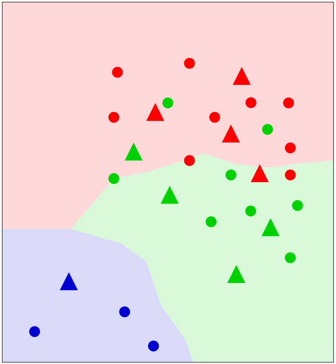
Classification of new cases (triangles; color indicates true class label; predicted label according to color of decision region):

1-Nearest Neighbor

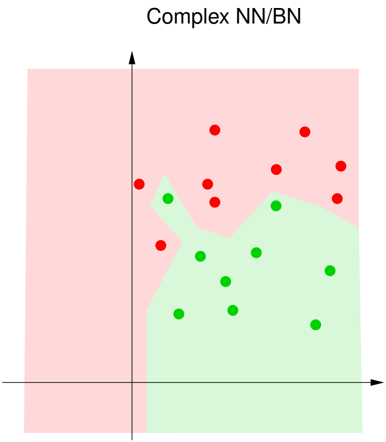


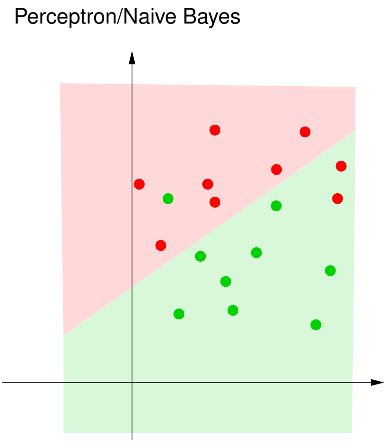
Accuracy train: 100%
Accuracy test: 6/9 $\sim$ 66%

5-Nearest Neighbor

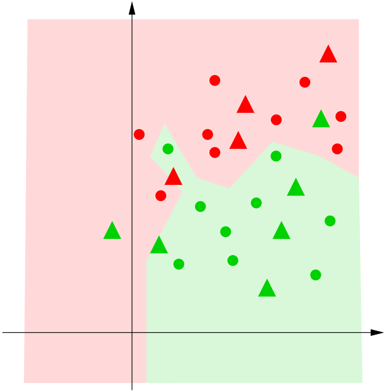


Accuracy train:15/20 $\sim$ 75%
Accuracy test: 7/9 $\sim$ 77%%

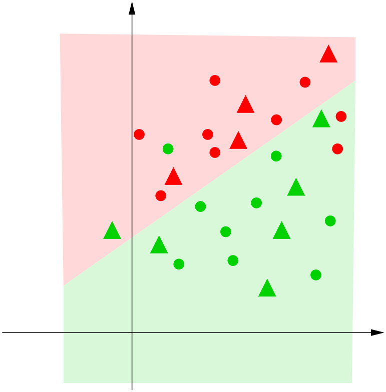

Complex NN/BN


Perceptron/Naive Bayes

Complex NN/BN



Perceptron/Naive Bayes

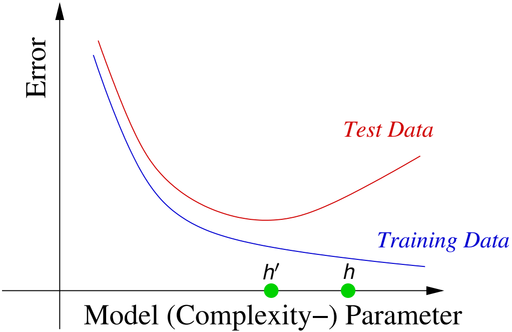


Given a hypothesis space H, a hypothesis $h \in H$ is said to **overfit** the training data if there exists some alternative hypothesis $h' \in H$, such that h has smaller error than h' over the training examples, but h' has a smaller error than h over the entire distribution of instances [Mitchell, p. 67].

[T. Mitchell: Machine Learning. McGraw-Hill, 1997]

Typically:

- the hypothesis space (or model space) can be structured by some model parameter
  - Nearest Neighbor classifier: k value
  - Neural Network: Number of hidden units/layers
  - Bayesian Network: Complexity of BN structure
- different parameter values lead to more or less complex decision regions
  - NN classifier: small $k \rightsquigarrow$ complex decision regions
- a hypothesis h overfits, if its decision regions are too closely fitted to the training data.



Benefits

- Unlikely to overfit
  - But still possible: even for linear models we use techniques to prevent overfitting
- Easy to learn from data
- Well understood
- Central building block also for more complex non-linear models

Diversity

Different types of linear classification models

- have the same **capacity**: they can represent exactly the same classification rules,

but they

- are based on different **objective functions** that are optimized in learning
  - Perceptron: minimize an **error function**
  - Naive Bayes: maximize **likelihood function**
- provide different **learning methods/algorithms**,
- return different results!

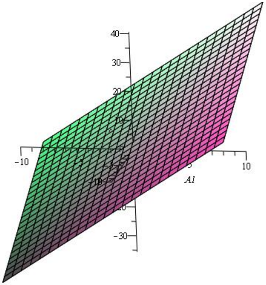
Linear Functions

Linear function of numeric attributes $X_1, \ldots, X_D$ with **coefficients** $w_0, w_1, \ldots, w_D \in \mathbb{R}$:

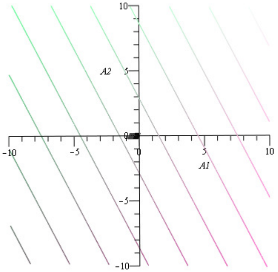
$$y(x_1, \ldots, x_D) = w_0 + w_1 x_1 + \ldots + w_D x_D$$

Plots for $D = 2$, $w_0 = 2.5$, $w_1 = 1.3$, $w_2 = 1.3$:

Graph



Top-down view: contour lines



Note: Strictly speaking, these are *affine functions*, and linear is the special case $w_0 = 0$.

Vector Notation

$$y(x_1, \ldots, x_D) = w_0 + w_1 x_1 + \ldots + w_D x_D = w_0 + \boldsymbol{w} \cdot \boldsymbol{x}$$

where $\boldsymbol{w} = (w_1, \ldots, w_D)$, $\boldsymbol{x} = (x_1, \ldots, x_D)$, and $\boldsymbol{w} \cdot \boldsymbol{x}$ is the **dot product** of vectors in $\mathbb{R}^D$.

Decision Regions

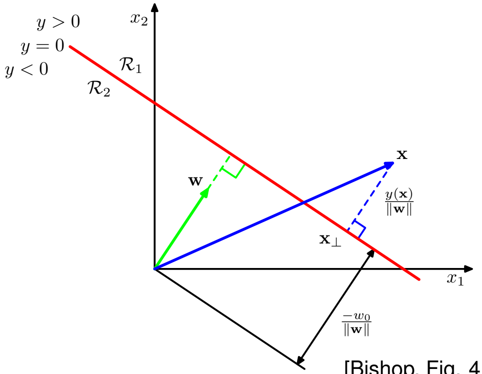
A linear function partitions the input space into the decision regions

$$\mathcal{R}_1 = \{\boldsymbol{x} \mid y(\boldsymbol{x}) \geq 0\}$$
$$\mathcal{R}_2 = \{\boldsymbol{x} \mid y(\boldsymbol{x}) < 0\}$$

$$y(\boldsymbol{x}) = w_0 + \boldsymbol{w} \cdot \boldsymbol{x}$$

$\boldsymbol{w}$: orientation of decision boundary;

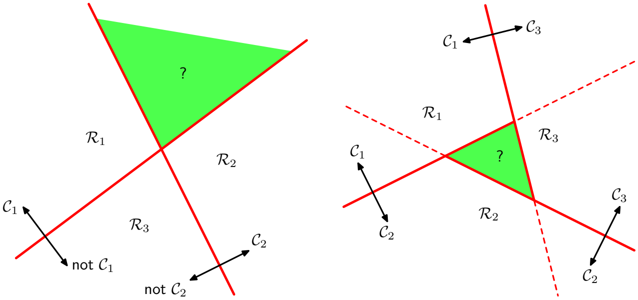
$|w_0| / \| \boldsymbol{w} \|$: distance of decision boundary from origin, where $\| \boldsymbol{w} \| = \sqrt{\boldsymbol{w} \cdot \boldsymbol{w}}$ is the length of the vector $\boldsymbol{w}$.



[Bishop, Fig. 4.1; here w_0 is negative!]

Several approaches to use linear functions for classification with more than two different class labels:

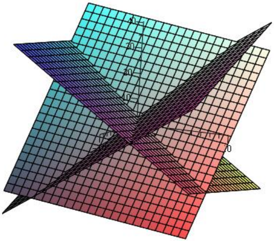
- Multiple binary “one against all” classifications
- Multiple binary “one against one” classifications



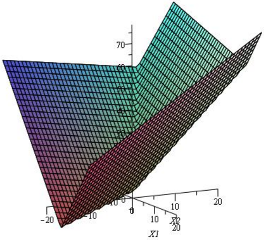
[Bishop, Fig. 4.2]

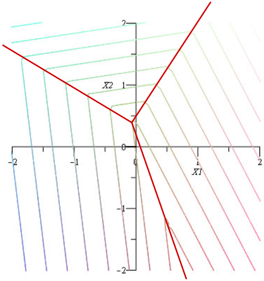
- construct one linear **Discriminant Function** y_k for each class label k
- classify $\boldsymbol{x}$ to belong to class k for which $y_k(\boldsymbol{x})$ is maximal

$y_1(\boldsymbol{x}), y_2(\boldsymbol{x}), y_3(\boldsymbol{x})$



Two views of $\max(y_1(\boldsymbol{x}), y_2(\boldsymbol{x}), y_3(\boldsymbol{x}))$





K discriminant functions for K class labels:

$$
\begin{array}{lllll}
y_1(\boldsymbol{x}) = & w_{1,0} + w_{1,1}x_1 + & \ldots & +w_{1,D}x_D & = w_{1,0} + \boldsymbol{w}_1 \cdot \boldsymbol{x} \\
y_2(\boldsymbol{x}) = & w_{2,0} + w_{2,1}x_1 + & \ldots & +w_{2,D}x_D & = w_{2,0} + \boldsymbol{w}_2 \cdot \boldsymbol{x} \\
\ldots & \ldots & \ldots & \ldots & \ldots \\
y_K(\boldsymbol{x}) = & w_{K,0} + w_{K,1}x_1 + & \ldots & +w_{K,D}x_D & = w_{K,0} + \boldsymbol{w}_K \cdot \boldsymbol{x}
\end{array}
$$

K discriminant functions for K class labels:

$$
\begin{array}{llllll}
y_1(\boldsymbol{x}) = & w_{1,0} + w_{1,1}x_1 + & \dots & +w_{1,D}x_D & = w_{1,0} + \boldsymbol{w}_1 \cdot \boldsymbol{x} \\
y_2(\boldsymbol{x}) = & w_{2,0} + w_{2,1}x_1 + & \dots & +w_{2,D}x_D & = w_{2,0} + \boldsymbol{w}_2 \cdot \boldsymbol{x} \\
\dots & \dots & \dots & \dots & \dots \\
y_K(\boldsymbol{x}) = & w_{K,0} + w_{K,1}x_1 + & \dots & +w_{K,D}x_D & = w_{K,0} + \boldsymbol{w}_K \cdot \boldsymbol{x}
\end{array}
$$

In vector-matrix notation:

$$\boldsymbol{y}(\boldsymbol{x}) = \boldsymbol{w}_0 + \boldsymbol{W}^T \boldsymbol{x}$$

where

$$\boldsymbol{w}_0 = \begin{pmatrix} w_{1,0} \\ \dots \\ w_{K,0} \end{pmatrix} \quad \boldsymbol{W} = \begin{pmatrix} w_{1,1} & w_{2,1} & \dots & w_{K,1} \\ \dots & \dots & \dots & \dots \\ w_{1,D} & w_{2,D} & \dots & w_{K,D} \end{pmatrix} \quad \boldsymbol{x} = \begin{pmatrix} x_1 \\ \dots \\ x_D \end{pmatrix}$$

K discriminant functions for K class labels:

$$
\begin{array}{llllll}
y_1(\boldsymbol{x}) = & w_{1,0} + w_{1,1}x_1 + & \dots & +w_{1,D}x_D & = w_{1,0} + \boldsymbol{w}_1 \cdot \boldsymbol{x} \\
y_2(\boldsymbol{x}) = & w_{2,0} + w_{2,1}x_1 + & \dots & +w_{2,D}x_D & = w_{2,0} + \boldsymbol{w}_2 \cdot \boldsymbol{x} \\
\dots & \dots & \dots & \dots & \dots \\
y_K(\boldsymbol{x}) = & w_{K,0} + w_{K,1}x_1 + & \dots & +w_{K,D}x_D & = w_{K,0} + \boldsymbol{w}_K \cdot \boldsymbol{x}
\end{array}
$$

In vector-matrix notation:

$$\boldsymbol{y}(\boldsymbol{x}) = \boldsymbol{w}_0 + \boldsymbol{W}^T\boldsymbol{x}$$

where

$$
\boldsymbol{w}_0 = \begin{pmatrix} w_{1,0} \\ \dots \\ w_{K,0} \end{pmatrix} \quad \boldsymbol{W} = \begin{pmatrix} w_{1,1} & w_{2,1} & \dots & w_{K,1} \\ \dots & \dots & \dots & \dots \\ w_{1,D} & w_{2,D} & \dots & w_{K,D} \end{pmatrix} \quad \boldsymbol{x} = \begin{pmatrix} x_1 \\ \dots \\ x_D \end{pmatrix}
$$

Even shorter:

$$\boldsymbol{y}(\boldsymbol{x}) = \tilde{\boldsymbol{W}}^T\tilde{\boldsymbol{x}}$$

where

$$
\tilde{\boldsymbol{W}} = \begin{pmatrix} w_{1,0} & w_{2,0} & \dots & w_{K,0} \\ w_{1,1} & w_{2,1} & \dots & w_{K,1} \\ \dots & \dots & \dots & \dots \\ w_{1,D} & w_{2,D} & \dots & w_{K,D} \end{pmatrix} \quad \tilde{\boldsymbol{x}} = \begin{pmatrix} 1 \\ x_1 \\ \dots \\ x_D \end{pmatrix}
$$

Classification as regression

For data case $\boldsymbol{x}_n$ with class label $y_n \in 1, \ldots, K$ define K-dimensional **target vector**

$$\boldsymbol{t}_n = (0, \ldots, 0, 1, 0, \ldots, 0)^T$$

with "1" in the y_n'th position (also called the **one-hot encoding** of y_n).

Goal: find weight matrix $\tilde{\boldsymbol{W}}$, so that

$$\boldsymbol{y}(\boldsymbol{x}_n) \sim \boldsymbol{t}_n$$

Sum-of-squares error

Try to minimize:

$$E_D(\tilde{\boldsymbol{W}}) = \frac{1}{2}\sum_{n=1}^{N} \parallel \boldsymbol{y}(\boldsymbol{x}_n) - \boldsymbol{t}_n \parallel^2 = \frac{1}{2}\sum_{n=1}^{N} \parallel \tilde{\boldsymbol{W}}^T \tilde{\boldsymbol{x}}_n - \boldsymbol{t}_n \parallel^2$$

In Matrix notation (Bishop, Equation (4.15)):

$$E_D(\tilde{\boldsymbol{W}}) = \frac{1}{2}\mathrm{Tr}\{(\tilde{\boldsymbol{X}}\tilde{\boldsymbol{W}} - \boldsymbol{T})^T(\tilde{\boldsymbol{X}}\tilde{\boldsymbol{W}} - \boldsymbol{T})\}$$

In "pedestrian" notation:

$$E_D(\tilde{\boldsymbol{W}}) = \frac{1}{2}\sum_n \sum_k (\sum_d w_{kd}\tilde{x}_{nd} - t_{nk})^2$$

Derivative w.r.t w_{kd}:

$$\sum_n (\sum_{d'} w_{kd'}\tilde{x}_{nd'} - t_{nk})\tilde{x}_{nd}$$

Setting this to zero:

$$\sum_n \tilde{x}_{nd} \sum_{d'} \tilde{x}_{nd'} w_{kd'} = \sum_n \tilde{x}_{nd} t_{nk}$$

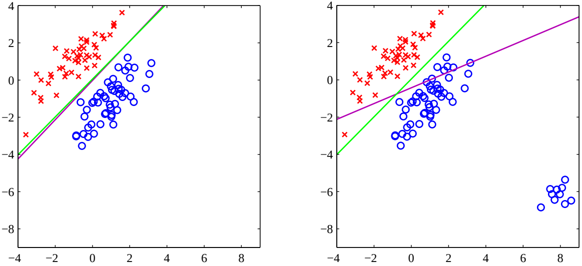
Collecting this for all d, k in a matrix equation:

$$\tilde{\boldsymbol{X}}^T \tilde{\boldsymbol{X}} \tilde{\boldsymbol{W}} = \tilde{\boldsymbol{X}}^T \boldsymbol{T}$$

Solving for $\tilde{\boldsymbol{W}}$:

$$\tilde{\boldsymbol{W}} = (\tilde{\boldsymbol{X}}^T \tilde{\boldsymbol{X}})^{-1} \tilde{\boldsymbol{X}}^T \boldsymbol{T}$$

Even for linearly separable datasets, the learned least-squares model may not separate the classes (magenta: decision boundary of least-squares):



[Bishop, Fig. 4.4]

Sum-of-squares error minimization does not directly minimize classification error.

Three data points ($N = 3$) in two dimensions ($D = 2$) with two possible labels ($K = 2$):

$$\boldsymbol{X} = (\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3) = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 5 & 1 \end{pmatrix} \qquad \boldsymbol{T} = (\boldsymbol{t}_1, \boldsymbol{t}_2, \boldsymbol{t}_3) = \begin{pmatrix} 1 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Two weight matrices and their predictions:

$$\boldsymbol{W}_1 = \begin{pmatrix} 10 & -1 \\ 0 & 1 \end{pmatrix} \quad \boldsymbol{y}_1(\boldsymbol{X}) = \boldsymbol{W}_1^T \boldsymbol{X} = \begin{pmatrix} 10 & 10 & 0 \\ 0 & 4 & 1 \end{pmatrix}$$

$$\boldsymbol{W}_2 = \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \quad \boldsymbol{y}_2(\boldsymbol{X}) = \boldsymbol{W}_2^T \boldsymbol{X} = \begin{pmatrix} 2 & 6 & 1 \\ 0 & -4 & -1 \end{pmatrix}$$

➥ $\boldsymbol{W}_1$ has higher accuracy (100%) than $\boldsymbol{W}_2$ (misclassifies $\boldsymbol{x}_3$), but $\boldsymbol{W}_2$ is preferred on sum-of-squares error criterion.

($\boldsymbol{W}_2$ is still far from being the solution with minimum sum-of-squares error; the actual optimal solution, in this case, will also have 100% accuracy, but this is not guaranteed in general)