Artificial Intelligence
EDAP01
Lecture 9: Machine Learning (II)

Pierre Nugues

Pierre.Nugues@cs.lth.se
http://cs.lth.se/pierre_nugues/

February 15, 2023

Linear Classifiers

Similar to decision trees, linear classifiers provide another set of techniques to learn classifiers from datasets.
This time, the objects will be represented by an **x** vector of numerical parameters, often called the **features**.
Linear classification methods operate in an n-dimensional space with a vector space equal to the number of features.
Linear classifiers include:

1. Perceptron
2. Logistic regression
3. Support vector machines
4. Neural networks

Problem Formulation

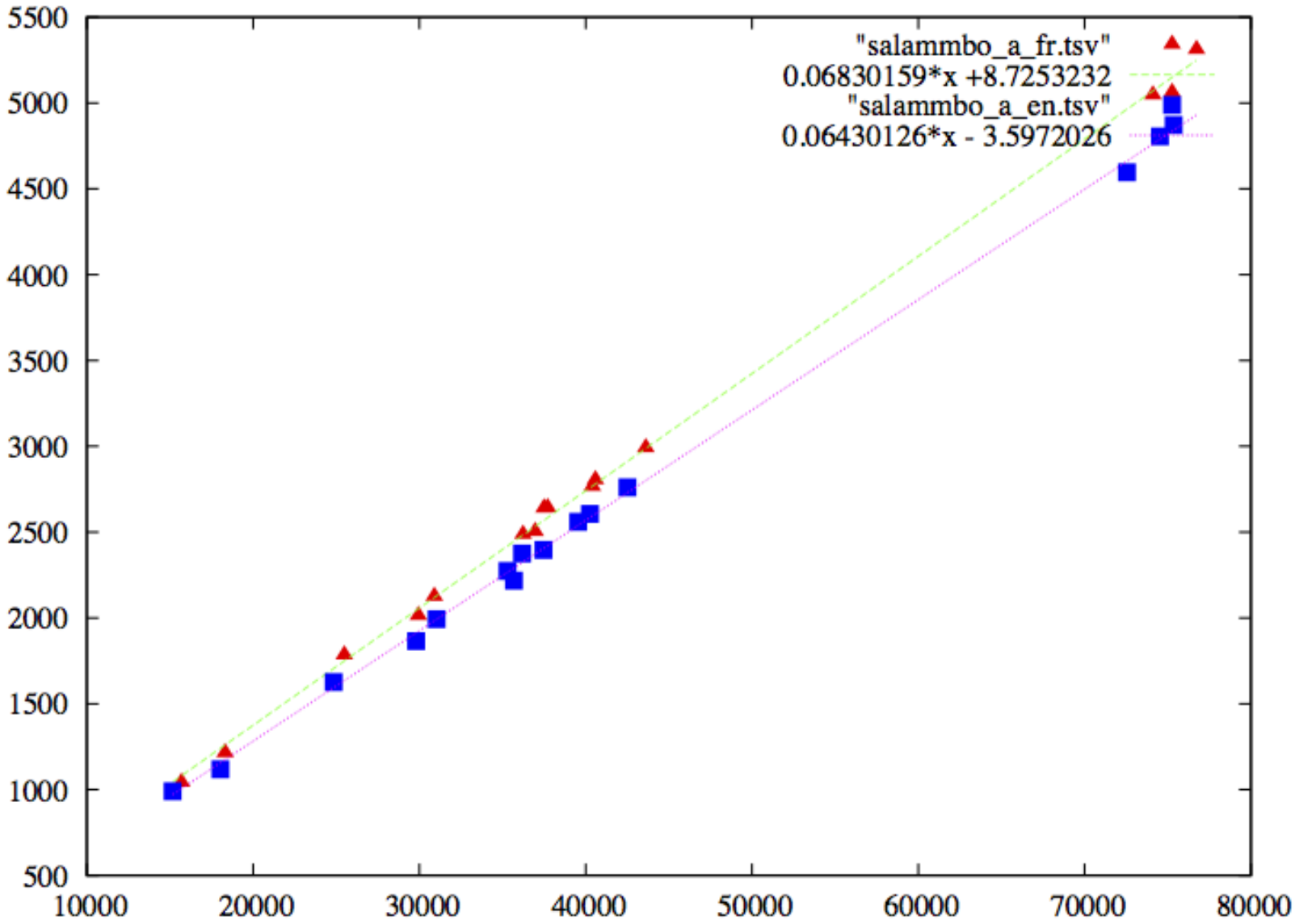
We can formulate the problem as finding lines automatically to:

1. Fit a dataset (regression). The output is a real continuous variable.
2. Separate classes inside a dataset (discriminant analysis, classification). The output is a discrete set of symbols, the classes, for instance {0, 1} or {negative, positive}.
   Here, we will consider the binary case only with two classes.
   Multiclass classification is a generalization of it.

Linear Regression: Letter Frequencies in *Salammbô*

Chapter	French		English	
	# characters	# *A*	# characters	# *A*
Chapter 1	36,961	2,503	35,680	2,217
Chapter 2	43,621	2,992	42,514	2,761
Chapter 3	15,694	1,042	15,162	990
Chapter 4	36,231	2,487	35,298	2,274
Chapter 5	29,945	2,014	29,800	1,865
Chapter 6	40,588	2,805	40,255	2,606
Chapter 7	75,255	5,062	74,532	4,805
Chapter 8	37,709	2,643	37,464	2,396
Chapter 9	30,899	2,126	31,030	1,993
Chapter 10	25,486	1,784	24,843	1,627
Chapter 11	37,497	2,641	36,172	2,375
Chapter 12	40,398	2,766	39,552	2,560
Chapter 13	74,105	5,047	72,545	4,597
Chapter 14	76,725	5,312	75,352	4,871
Chapter 15	18,317	1,215	18,031	1,119
Total	619,431	42,439	608,230	39,056

Fitting the Data



Fitting Equations

In a two-dimensional space, the straight line fitting the points is given by:

$$\hat{y} = mx + b,$$

Minimize the loss between:

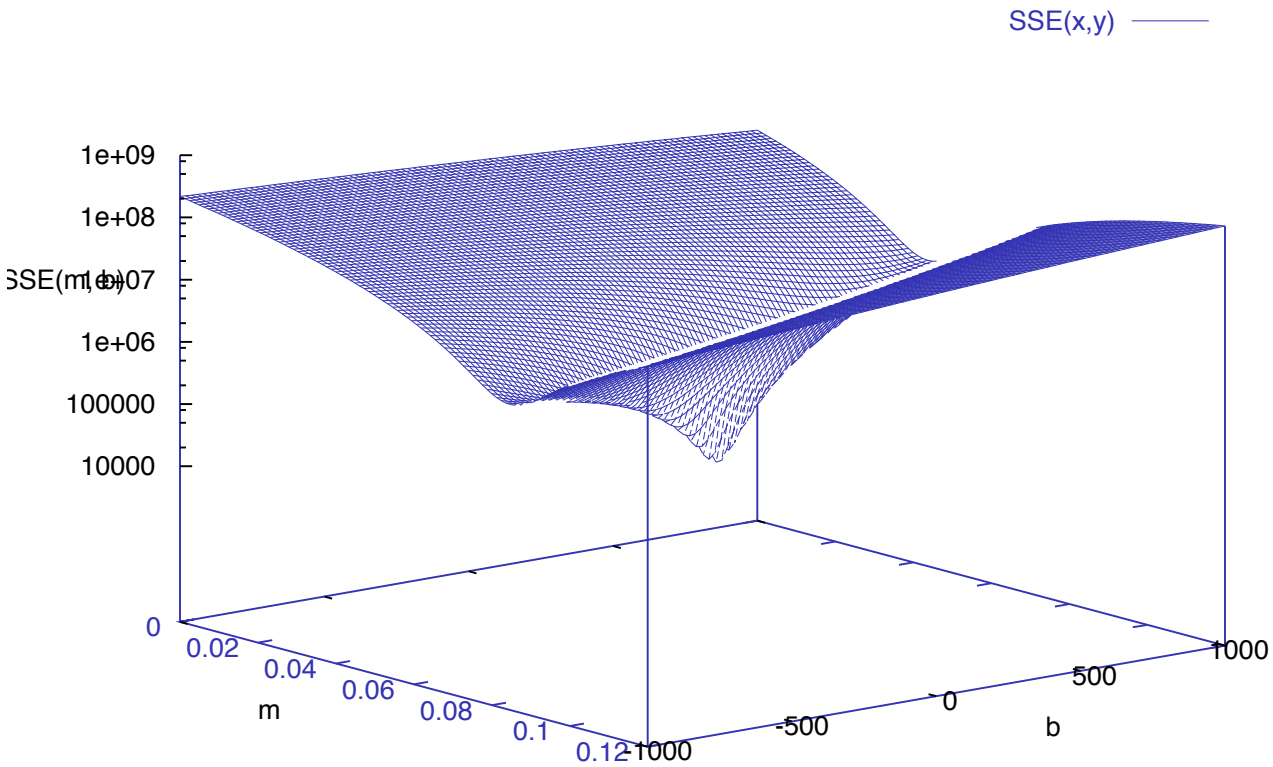
- The set of q observations: $\{(x_i, y_i)\}_{i=1}^{q}$ and
- A perfect linear alignment: $\{(x_i, f(x_i))\}_{i=1}^{q}$, where $f(x_i) = mx_i + b$.

The loss is traditionally modeled by the quadratic error (Legendre 1805):
$L_2 = \sum_{\text{Set of points}} (y - \hat{y})^2$.
For q points (x_i, y_i):

$$Loss(m, b) = \sum_{i=1}^{q} (y_i - (mx_i + b))^2$$

The objective of regression is to find the weights that minimize the loss.

Visualizing the Loss



Programs available here: `https://github.com/pnugues/ilppp/tree/master/programs/ch04/python`

Minimizing the Loss

The loss function is convex and has a unique minimum.
The loss reaches a minimum when the partial derivatives are zero:

$$\frac{\partial Loss}{\partial m} = \sum_{i=1}^{q} \frac{\partial}{\partial m}(y_i - (mx_i + b))^2 = -2\sum_{i=1}^{q} x_i(y_i - (mx_i + b)) = 0$$

$$\frac{\partial Loss}{\partial b} = \sum_{i=1}^{q} \frac{\partial}{\partial b}(y_i - (mx_i + b))^2 = -2\sum_{i=1}^{q} (y_i - (mx_i + b)) = 0$$

Fitting Equations (II)

This yields:

$$\begin{aligned} b &= \bar{y} - m\bar{x} \\ \sum_{i=1}^{N} x_i y_i - m\sum_{i=1}^{q} x_i^2 - qb\bar{x} &= \sum_{i=1}^{q} x_i y_i - m\sum_{i=1}^{q} x_i^2 - q\bar{x}(\bar{y} - m\bar{x}) = 0 \end{aligned}$$

$$m = \frac{\sum_{i=1}^{q} x_i y_i - q\bar{x}\bar{y}}{\sum_{i=1}^{q} x_i^2 - q\bar{x}^2} \quad \text{and} \quad b = \bar{y} - m\bar{x},$$

with

$$\bar{x} = \frac{1}{q}\sum_{i=1}^{q} x_i \quad \text{and} \quad \bar{y} = \frac{1}{q}\sum_{i=1}^{q} y_i.$$

$$\begin{aligned} \text{French:} \quad y &= 0.0683x + 8.7253 \\ \text{English:} \quad y &= 0.0643x - 3.5972 \end{aligned}$$

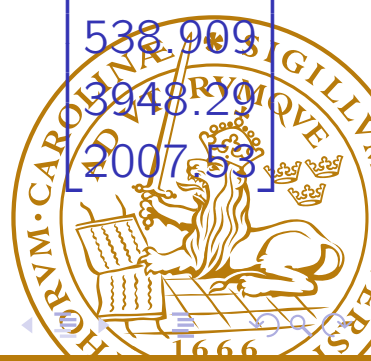
Regression: Notation in a Two-dimensional Space

1. The feature vector (or predictors): $\mathbf{x} = (1, x_1, x_2, ..., x_{n-1})$ representing the input. In our example, this corresponds to the letter frequencies: (1, 36961) in Chapter 1. The first parameter is set to 1 to model the intercept;
2. The output (or response or target): y. In our example, the frequencies of *a*: 2503 in Chapter 1;
3. The model: a weight vector $\mathbf{w}$, for French, $\mathbf{w} = (8.7253, 0.0683)$;
4. The predicted output:
   $\hat{y} = \mathbf{w} \cdot \mathbf{x} = 1 \times 8.7253 + 0.0683 \times 36961 = 2533.22$
5. The squared error for one observation:
   $(\hat{y} - y)^2 = (2503 - 2533.22)^2 = 30.22^2 = 913.26$
6. To denote the whole dataset, we use a matrix, X, where each row corresponds to an observation, a $\mathbf{y}$ vector for the outputs, and a $\hat{\mathbf{y}}$ vector for the predicted outputs.

Complete Matrix

For the French dataset, the complete matrix and vectors are:

$$X = \begin{bmatrix} 1 & 36961 \\ 1 & 43621 \\ 1 & 15694 \\ 1 & 36231 \\ 1 & 29945 \\ 1 & 40588 \\ 1 & 75255 \\ 1 & 37709 \\ 1 & 30899 \\ 1 & 25486 \\ 1 & 37497 \\ 1 & 40398 \\ 1 & 74105 \\ 1 & 76725 \\ 1 & 18317 \end{bmatrix}; \mathbf{w} = \begin{bmatrix} 8.7253 \\ 0.0683 \end{bmatrix}; \hat{\mathbf{y}} = X\mathbf{w} = \begin{bmatrix} 2533.22 \\ 2988.11 \\ 1080.65 \\ 2483.36 \\ 2054.02 \\ 2780.95 \\ 5148.76 \\ 2584.31 \\ 2119.18 \\ 1749.46 \\ 2569.83 \\ 2767.97 \\ 5070.21 \\ 5249.16 \\ 1259.81 \end{bmatrix}; \mathbf{y} = \begin{bmatrix} 2503 \\ 2992 \\ 1042 \\ 2487 \\ 2014 \\ 2805 \\ 5062 \\ 2643 \\ 2126 \\ 1784 \\ 2641 \\ 2766 \\ 5047 \\ 5312 \\ 1215 \end{bmatrix}; \mathbf{se} = \begin{bmatrix} 913.26 \\ 15.14 \\ 1493.86 \\ 13.25 \\ 1601.31 \\ 578.40 \\ 7527.51 \\ 3444.53 \\ 46.57 \\ 1193.04 \\ 5065.18 \\ 38920 \\ 538.969 \\ 3948.29 \\ 2007.53 \end{bmatrix}.$$



Regression: Notation in an N-dimensional Space

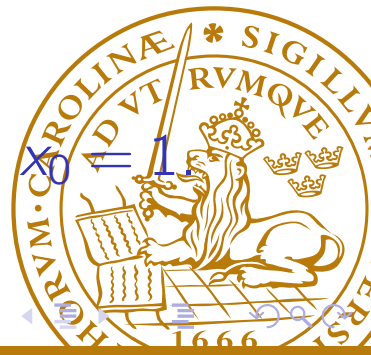
The least-squares method minimizes the sum of the squared errors for all the observations through optimal values of m and b. In an n-dimensional space, we have:

1. The weight vector: $(w_0, w_1, w_2, \ldots, w_{n-1})$, $(8.7253, 0.0683)$ for French and $(-3.5972, 0.0643)$ for English;
2. The intercept: w_0.

The hyperplane equation corresponds to the dot product of the weights by the features defined as:

$$y = \mathbf{w} \cdot \mathbf{x} = \sum_{i=0}^{n-1} w_i x_i,$$

where $\mathbf{w} = (w_0, w_1, w_2, \ldots, w_{n-1})$, $\mathbf{x} = (x_0, x_1, x_2, \ldots, x_{n-1})$, and $x_0 = 1$.



Another Technique: The Gradient Descent

The gradient descent is a numerical method to find the minimum of $f(w_0, w_1, w_2, ..., w_n) = y$, when there is no analytical solution.
Let us denote $\mathbf{w} = (w_0, w_1, w_2, ..., w_n)$
We derive successive approximations to find the minimum of f:

$$f(\mathbf{w}_1) > f(\mathbf{w}_2) > ... > f(\mathbf{w}_k) > f(\mathbf{w}_{k+1}) > .. > min$$

Points in the neighborhood of $\mathbf{w}$ are defined by $\mathbf{w} + \mathbf{v}$ with $||\mathbf{v}||$ small
Given $\mathbf{w}$, find $\mathbf{v}$ subject to $f(\mathbf{w}) > f(\mathbf{w} + \mathbf{v})$

The Gradient Descent I (Cauchy, 1847)

Using a Taylor expansion: $f(\mathbf{w}+\mathbf{v}) = f(\mathbf{w}) + \mathbf{v}\cdot\nabla f(\mathbf{w}) + \ldots$
The gradient is a direction vector corresponding to the steepest slope:

$$\nabla f(w_0, w_1, w_2, \ldots, w_n) = (\frac{\partial f}{\partial w_0}, \frac{\partial f}{\partial w_1}, \frac{\partial f}{\partial w_2}, \ldots, \frac{\partial f}{\partial w_n}).$$

$\mathbf{v}\cdot\nabla f(\mathbf{w})$ is maximal when $\mathbf{v}$ and $\nabla f(\mathbf{w})$ are colinear
$f(\mathbf{w}+\mathbf{v})$ reaches a minimum or a maximum then:

- Steepest ascent: $\mathbf{v} = \alpha\nabla f(\mathbf{w})$,
- Steepest descent: $\mathbf{v} = -\alpha\nabla f(\mathbf{w})$,

where $\alpha > 0$.
For the steepest descent, we have

$$f(\mathbf{w} - \alpha\nabla f(\mathbf{w})) \approx f(\mathbf{w}) - \alpha||\nabla f(\mathbf{w})||^2.$$

The Gradient Descent II (Cauchy, 1847)

We rewrite the inequality at step k:

$$\begin{aligned} f(\mathbf{w}_k) &> f(\mathbf{w}_{k+1}), \\ &> f(\mathbf{w}_k + \mathbf{v}) \end{aligned}$$

with the steepest descent as:

$$f(\mathbf{w}_k) > f(\mathbf{w}_k - \alpha \nabla f(\mathbf{w}_k)),$$

where

$$f(\mathbf{w}_k - \alpha \nabla f(\mathbf{w}_k)) \approx f(\mathbf{w}_k) - \alpha ||\nabla f(\mathbf{w}_k)||^2.$$

The sequence of inequalities enables us to define an iteration:

$$\mathbf{w}_{k+1} = \mathbf{w}_k - \alpha_k \nabla f(\mathbf{w}_k),$$

to reach the minimum.
In this iteration, the gradient gives us the direction of the descent and α_k, the step size (or learning rate).

Algorithm

For a dataset DS, we find the minimum through a slide (iteration) down the surface.

1: **function** Regression(DS)
2: $\quad \mathbf{w} \leftarrow \mathbf{w^0}$ ▷ $\mathbf{w^0}$: Any point in the space
3: $\quad$ **while** $||\nabla Loss(\mathbf{w})|| > \epsilon$ **do**
4: $\quad\quad \mathbf{w} \leftarrow \mathbf{w} - \alpha \nabla Loss(\mathbf{w})$
5: $\quad$ **return w**

α is a positive number either constant or variable.

In a Two-dimensional Space

To make the generalization easier, let us rename the straight line coefficients (b, m) as (w_0, w_1)
Given a dataset DS of q examples: $DS = \{(1, x_{i,1}, y_i) | i : 1..q\}$, where the error is defined as:

$$\begin{aligned} SSE(w_0, w_1) &= \sum_{i=1}^{q} (y_i - \hat{y}_i)^2 \\ &= \sum_{i=1}^{q} (y_i - (w_0 + w_1 x_{i,1}))^2. \end{aligned}$$

The gradient with q examples:

$$\begin{aligned} \frac{\partial SSE}{\partial w_0} &= -2 \sum_{i=1}^{q} (y_i - (w_0 + w_1 x_{i,1})) \\ \frac{\partial SSE}{\partial w_1} &= -2 \sum_{i=1}^{q} x_{i,1} \times (y_i - (w_0 + w_1 x_{i,1})) \end{aligned}$$

Updates

Batch gradient descent with q examples (all the dataset):

$$\begin{aligned} w_0 &\leftarrow w_0 + \frac{\alpha}{q}\sum_{i=1}^{q}(y_i - (w_0 + w_1 x_{i,1})) \\ w_1 &\leftarrow w_1 + \frac{\alpha}{q}\sum_{i=1}^{q} x_{i,1} \times (y_i - (w_0 + w_1 x_{i,1})) \end{aligned}$$

Stochastic gradient descent. The updates are carried out one observation at a time:

$$\begin{aligned} w_0 &\leftarrow w_0 + \alpha \cdot (y_i - (w_0 + w_1 x_{i,1})) \\ w_1 &\leftarrow w_1 + \alpha \cdot x_{i,1} \cdot (y_i - (w_0 + w_1 x_{i,1})) \end{aligned}$$

The stochastic variant is also called online learning and is usually faster. In practice, most systems use an intermediate technique with **minibatches** of 32, 64, 128 samples.

The Gradient Descent and Linear Regression

In the general case, we want to find the regression hyperplane:

$$\hat{y} = w_0 + w_1 x_1 + w_2 x_2 + ... + w_n x_n$$

given a dataset of q examples: $DS = \{(1, x_{i,1}, x_{i,2}, ..., x_{i,n}, y_i) | i : 1..q\}$, where the loss is defined as:

$$\begin{aligned} Loss(w_0, w_1, ..., w_n) &= \sum_{i=1}^{q} (y_i - \hat{y}_i)^2 \\ &= \sum_{i=1}^{q} (y_i - (w_0 + w_1 x_{i,1} + w_2 x_{i,2} + ... + w_n x_{i,n}))^2 \end{aligned}$$

We introduce $x_{i,0} = 1$ and we compute the gradient to determine the slope:

$$\frac{\partial Loss}{\partial w_j} = -2 \sum_{i=1}^{q} x_{i,j} \times (y_i - (w_0 x_{i,0} + w_1 x_{i,1} + w_2 x_{i,2} + ... + w_n x_{i,n}))$$

Updates

In the **batch gradient descent**, the iteration step considers all the examples, q examples here:

$$w_j \leftarrow w_j + \frac{\alpha}{q} \cdot \sum_{i=1}^{q} x_{i,j} \cdot (y_i - (w_0 x_{i,0} + w_1 x_{i,1} + w_2 x_{i,2} + \ldots + w_n x_{i,n}))$$

Using a matrix notation:

$$\mathbf{w} \leftarrow \mathbf{w} + \frac{\alpha}{q} X^\intercal (\mathbf{y} - X\mathbf{w}).$$

In the **stochastic gradient descent**, we carry out the updates using one example at a time, for $(\mathbf{x}_i, y_i)$, for instance:

$$w_j \leftarrow w_j + \alpha \cdot x_{i,j} \cdot (y_i - (w_0 x_{i,0} + w_1 x_{i,1} + w_2 x_{i,2} + \ldots + w_n x_{i,n}))$$

Using a vector notation with $\mathbf{x}_i$ and y_i, we have:

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha \cdot (y_i - \mathbf{x}_i \cdot \mathbf{w}) \cdot \mathbf{x}_i.$$

Definitions

An epoch is the period corresponding to one iteration over the complete dataset: the q examples.

α is called the learning rate or the step size. It is a positive number that is either fixed or varies (decreases) over the epochs.

Setting α can be difficult. In the assignments, you will manually try a set of values and check the convergence.
In *Probabilistic Machine Learning*, Murphy (2022) describes other methods (for logistic regression). See Sect. 8.4.2 and 8.4.3. They are outside the scope of this course.

The Analytical Solution Again

In a two-dimensional space, linear regression has an analytical solution:

$$m = \frac{\sum_{i=1}^{q} x_i y_i - q\bar{x}\bar{y}}{\sum_{i=1}^{q} x_i^2 - q\bar{x}^2} \qquad \text{and} \qquad b = \bar{y} - m\bar{x}.$$

This can be generalized for any dimension.
Let $X = (\mathbf{x}^1; \mathbf{x}^2; ...; \mathbf{x}^q)$ be our observations and $\mathbf{y}$, the output. We have:

$$X\mathbf{w} = \hat{\mathbf{y}}$$

We minimize the sum of squared errors with:

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

$(X^\intercal X)^{-1} X^\intercal$ is called the pseudo-inverse.

Inverting $X^\intercal X$

But: $X^\intercal X$ may be singular.
We have the property: $X^\intercal X$ is invertible if X has linearly independent columns.
In real datasets, highly correlated features (or duplicate ones) is a frequent phenomenon.
This means that singular matrices will also occur with frequently
A way to solve it is to add it a small scaled identity matrix (Hoerl, 1962):

$$\mathbf{w} = (X^\intercal X + \lambda I)^{-1} X^\intercal \mathbf{y}$$

Equivalent to adding the term $\lambda||\mathbf{w}||^2$ to the sum of squared errors (SSE).
This process is called a regularization. It is also used in classification.
Programs available here: `https://github.com/pnugues/ilppp/tree/master/programs/ch04/python`

Regularization

Correlated features result in large values of $\mathbf{w}$ that regularization tries to penalize.
We replace *Loss* with a *Cost*:

$$Cost(\mathbf{w}) = Loss(\mathbf{w}) + \lambda L_q(\mathbf{w}),$$

where

$$L_q = \sum_{i=1}^{n} |w_i|^q.$$

The most frequent regularizations are (ridge regression):

$$L_2 = \sum_{i=1}^{n} w_i^2,$$

and (LASSO regression)

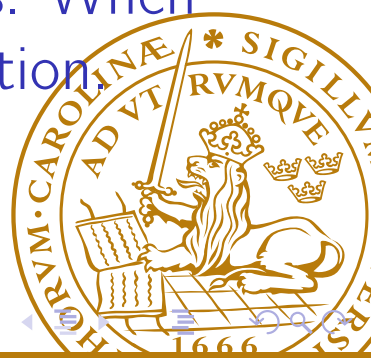
$$L_1 = \sum_{i=1}^{n} |w_i|.$$

w_0 may be part of the regularization.

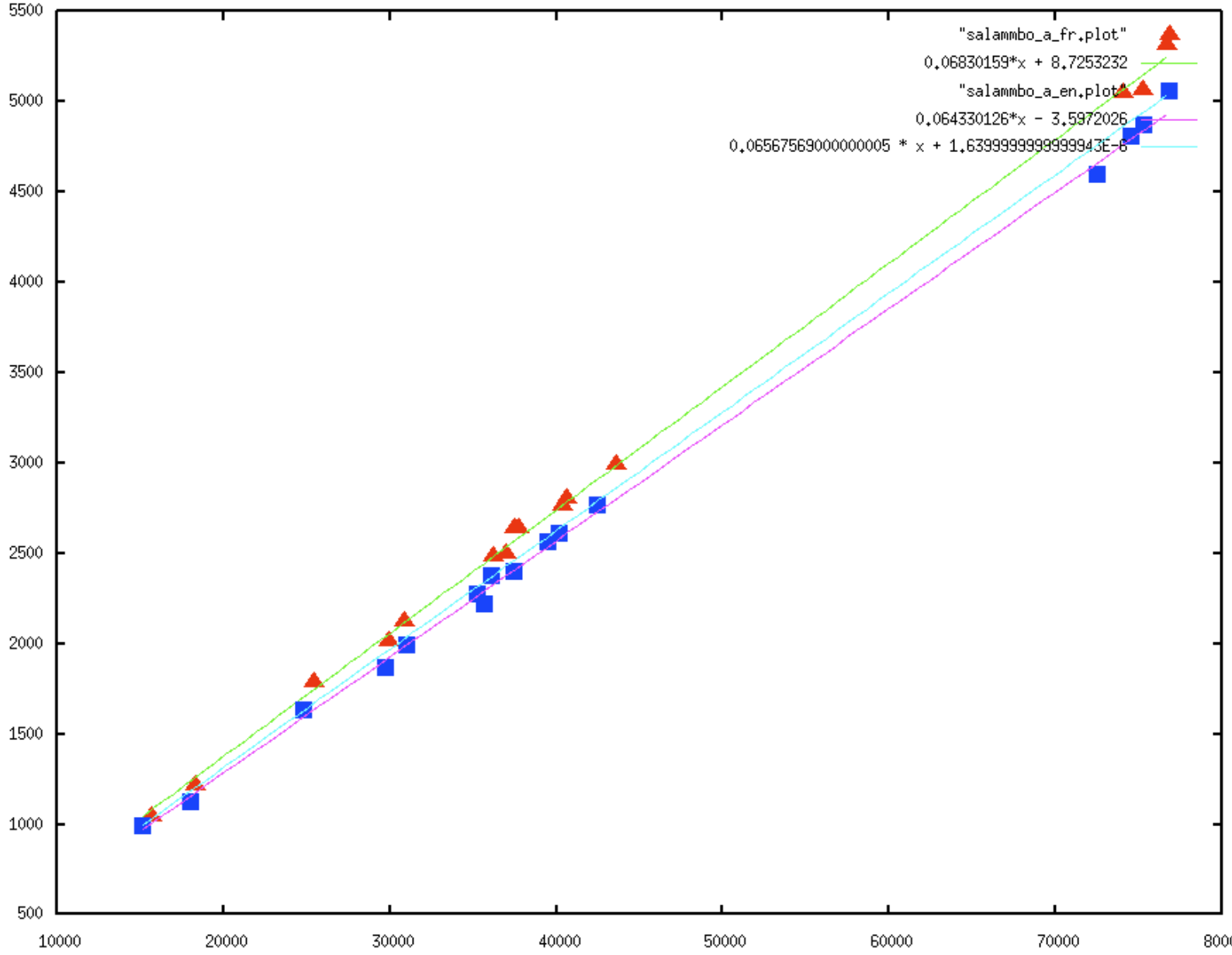
Classification vs. Regression

Regression: Given an input, compute a continuous numerical output.
For instance compute the number of *A* occurring in a text in French from the total number of characters.
Having 75,255 characters in Chapter 7, the regression line will predict 5,149 occurrences of *A* (5,062 in reality).

Classification: Given the number of characters and the number of *A* in a text, predict the language: French or English.
For instance, having the pair (75255, 5062), the classifier will predict French.
The classification output is a finite set of values. When there are two values, we have a binary classification.



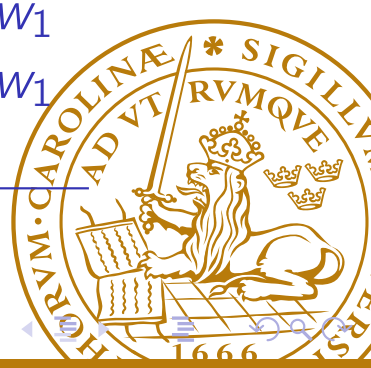
Separating Classes



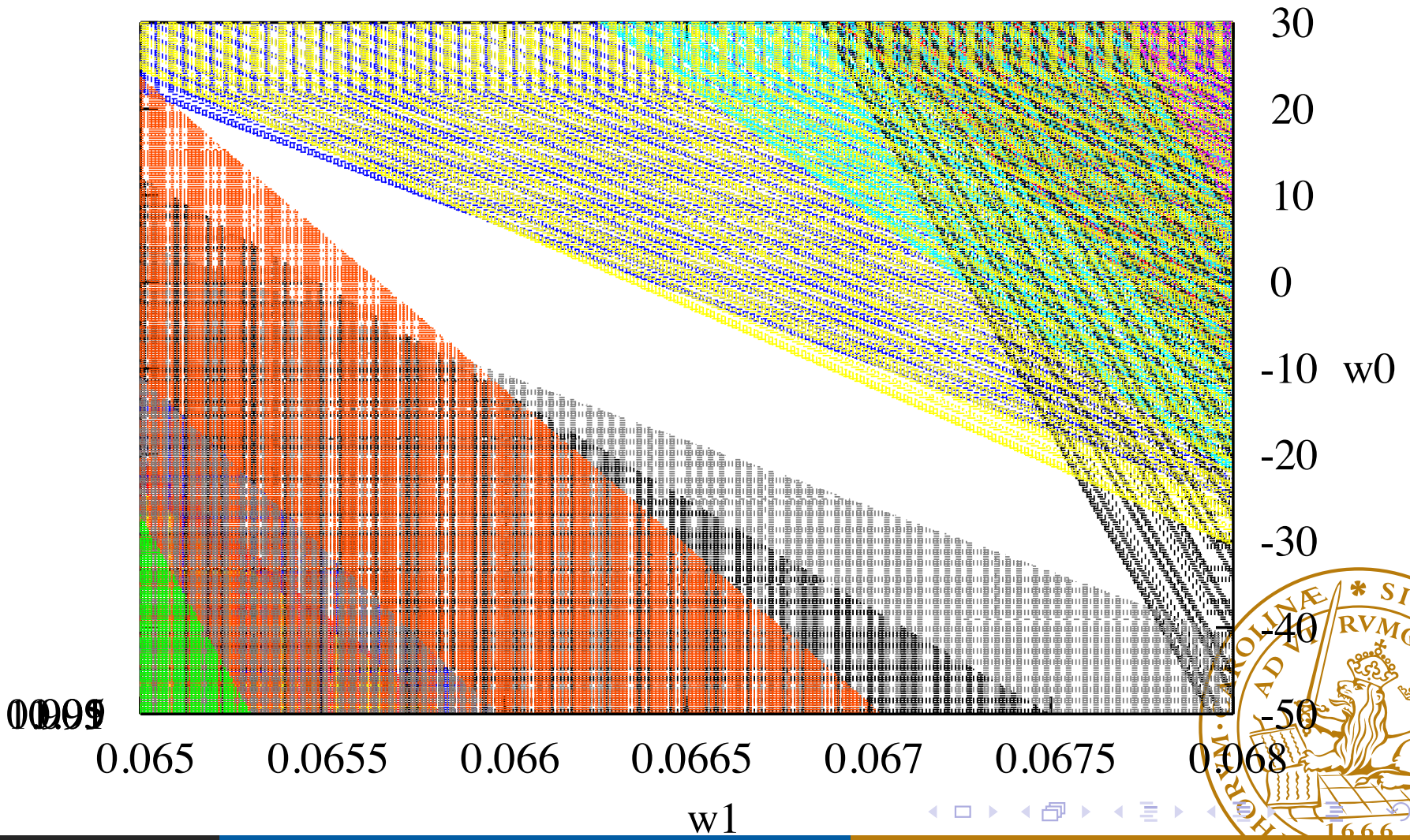
Classification: An Example

$y_i > w_0 + w_1 x_i$ for the set of points: $\{(x_i, y_i) | (x_i, y_i) \in \text{French}\}$ and
$y_i < w_0 + w_1 x_i$ for the set of points: $\{(x_i, y_i) | (x_i, y_i) \in \text{English}\}$,

Chapter	French	English
1	$2503 > w_0 + 36961 w_1$	$2217 < w_0 + 35680 w_1$
2	$2992 > w_0 + 43621 w_1$	$2761 < w_0 + 42514 w_1$
3	$1042 > w_0 + 15694 w_1$	$990 < w_0 + 15162 w_1$
4	$2487 > w_0 + 36231 w_1$	$2274 < w_0 + 35298 w_1$
5	$2014 > w_0 + 29945 w_1$	$1865 < w_0 + 29800 w_1$
6	$2805 > w_0 + 40588 w_1$	$2606 < w_0 + 40255 w_1$
7	$5062 > w_0 + 75255 w_1$	$4805 < w_0 + 74532 w_1$
...		

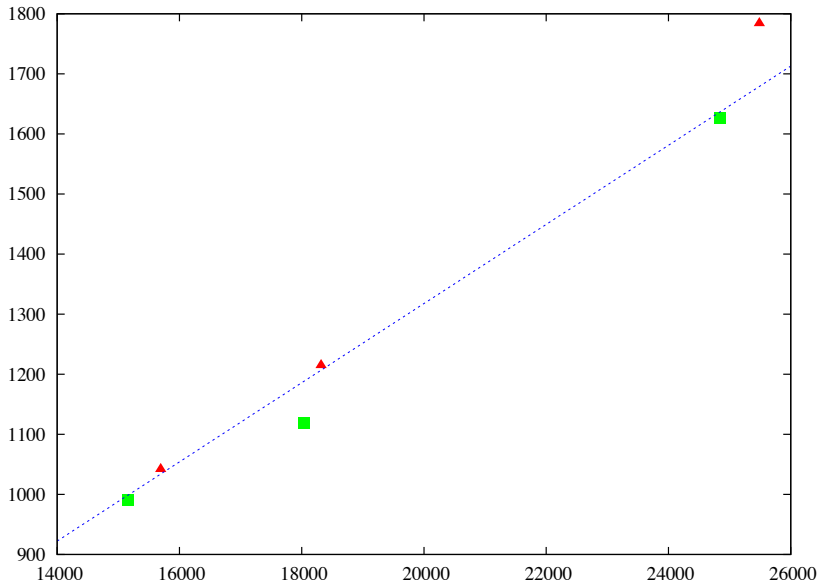


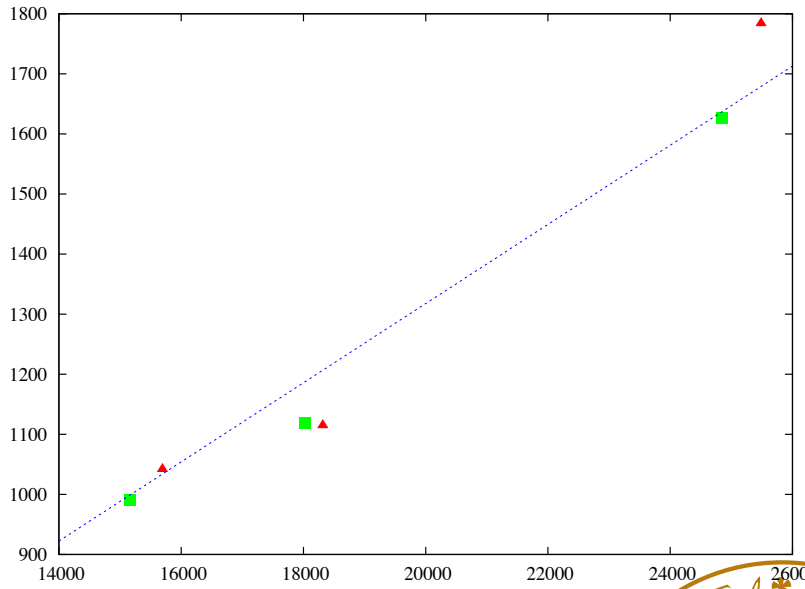
The Inequality System

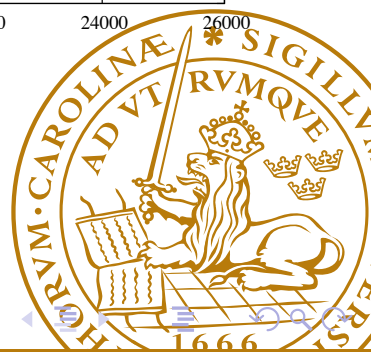


Separability

Linear classification is based on the linear separability of the classes.
This is not always the case







Dimension of the Feature Space

Regression predicts the value of one of the features given the value of $N-1$ features.
Classification predicts the class given the values of N features.
Compared to regression in our example, the dimension of the vector space used for the classification is $N+1$: The N features and the class.

# characters	# *A*	Language
36,961	2,503	French
35,680	2,217	English
43,621	2,992	French
42,514	2,761	English
15,694	1,042	French
15,162	990	English
36,231	2,487	French
35,298	2,274	English
etc.		

Classification

We represent classification using a threshold function (a variant of the *signum* function):

$$H(\mathbf{w}\cdot\mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{w}\cdot\mathbf{x} \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

The classification function associates P with 1 and N with 0.
We want to find the separating hyperplane:

$$\begin{aligned} \hat{y}(\mathbf{x}) &= H(\mathbf{w}\cdot\mathbf{x}) \\ &= H(w_0 + w_1x_1 + w_2x_2 + \ldots + w_nx_n), \end{aligned}$$

given a dataset of q examples: $DS = \{(1, x_{i,1}, x_{i,2}, \ldots, x_{i,n}, y_i) | i : 1..q\}$.
We use $x_{i,0} = 1$ to simplify the equations.
For a binary classifier, y has then two possible values {0, 1} corresponding in our example to {French, English}.

Loss Function

To assess the classification, we use the 0/1 loss defined as

$$L_{0/1}(y, \hat{y}) = \begin{cases} 0 & \text{if } y = \hat{y} \\ 1 & \text{otherwise} \end{cases}$$

The update rule of linear regression is:

$$\begin{aligned} w_j &\leftarrow w_j + \alpha \cdot \sum_{i=1}^{q} x_{i,j} \cdot (y_i - \hat{y}_i) \\ w_j &\leftarrow w_j + \alpha \cdot \sum_{i=1}^{q} x_{i,j} \cdot (y_i - (w_0 x_{i,0} + w_1 x_{i,1} + w_2 x_{i,2} + \ldots + w_n x_{i,n})) \end{aligned}$$

Using the same idea, we compute the weight updates of a classification as:

$$w_j \leftarrow w_j + \alpha \sum_{i=1}^{q} x_{i,j} \cdot (y_i - \hat{y}_i)$$

The Perceptron Learning Rule

If all the points are correctly classified, there is no update: $y_i - \hat{y}_i = 0$, either $0-0$ or $1-1$

The loss involves an update only for the misclassified points, either $1-0$ or $0-1$

The update is then: $\alpha \cdot x_{i,j}$ or $-\alpha \cdot x_{i,j}$

α is generally set to 1 as a division of the weight vector by a constant does not affect the update rule

The perceptron is usually trained one example at a time: stochastic learning

Examples should be selected randomly

In a Two-dimensional Space

Classification with two classes:

$$w_0 + w_1 x_1 + w_2 x_2 > 0$$

$$w_0 + w_1 x_1 + w_2 x_2 < 0,$$

Vectors:

$$\mathbf{x}_i = (1, x_{i,1}, x_{i,2}) \quad \text{and} \quad \mathbf{w} = (w_0, w_1, w_2)$$

Stochastic gradient descent. The updates are carried out one example at a time:

$$\begin{aligned} w_0 &\leftarrow w_0 + (y_i - \hat{y}_i) \\ w_1 &\leftarrow w_1 + x_{i,1} \cdot (y_i - \hat{y}_i) \\ w_2 &\leftarrow w_2 + x_{i,2} \cdot (y_i - \hat{y}_i) \end{aligned}$$

where $y_i - \hat{y}_i$ is either, 0, -1, or 1.

Stop Conditions

To find a hyperplane, the examples must be separable. This is rarely the case in practice
Workaround:

- Stop learning when the number of misclassified examples is low
- Use $\alpha(t) = \frac{1000}{1000 + t}$ instead of a fixed value. (According to the book)

Matrix Notation with Symbolic Attributes

- A feature vector (predictors): $\mathbf{x}$, and feature matrix: X;
- The class: y and the class vector (response): $\mathbf{y}$;
- The predicted class (predicted response): $\hat{y}$, and predicted class vector: $\hat{\mathbf{y}}$

$$X = \begin{bmatrix} \textit{Sunny} & \textit{Hot} & \textit{High} & \textit{False} \\ \textit{Sunny} & \textit{Hot} & \textit{High} & \textit{True} \\ \textit{Overcast} & \textit{Hot} & \textit{High} & \textit{False} \\ \textit{Rain} & \textit{Mild} & \textit{High} & \textit{False} \\ \textit{Rain} & \textit{Cool} & \textit{Normal} & \textit{False} \\ \textit{Rain} & \textit{Cool} & \textit{Normal} & \textit{True} \\ \textit{Overcast} & \textit{Cool} & \textit{Normal} & \textit{True} \\ \textit{Sunny} & \textit{Mild} & \textit{High} & \textit{False} \\ \textit{Sunny} & \textit{Cool} & \textit{Normal} & \textit{False} \\ \textit{Rain} & \textit{Mild} & \textit{Normal} & \textit{False} \\ \textit{Sunny} & \textit{Mild} & \textit{Normal} & \textit{True} \\ \textit{Overcast} & \textit{Mild} & \textit{High} & \textit{True} \\ \textit{Overcast} & \textit{Hot} & \textit{Normal} & \textit{False} \\ \textit{Rain} & \textit{Mild} & \textit{High} & \textit{True} \end{bmatrix}; \mathbf{y} = \begin{bmatrix} \text{N} \\ \text{N} \\ \text{P} \\ \text{P} \\ \text{P} \\ \text{N} \\ \text{P} \\ \text{N} \\ \text{P} \\ \text{P} \\ \text{P} \\ \text{P} \\ \text{P} \\ \text{N} \end{bmatrix}$$

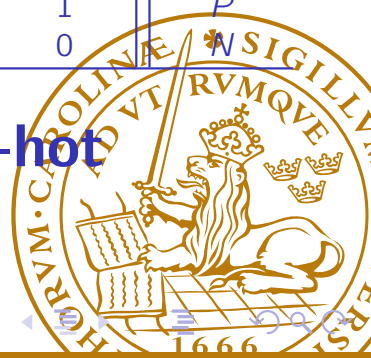
Linear classifiers are numerical systems.

Converting Symbolic Attributes into Numerical Vectors

Symbolic – nominal – attributes are mapped onto vectors of binary values: unit vectors.
Vectorization (conversion) of the weather dataset.

Object	Attributes										Class
	Outlook			Temperature			Humidity		Windy		
	Sunny	Overcast	Rain	Hot	Mild	Cool	High	Normal	True	False	
1	1	0	0	1	0	0	1	0	0	1	*N*
2	1	0	0	1	0	0	1	0	1	0	*N*
3	0	1	0	1	0	0	1	0	0	1	*P*
4	0	0	1	0	1	0	1	0	0	1	*P*
5	0	0	1	0	0	1	0	1	0	1	*P*
6	0	0	1	0	0	1	0	1	1	0	*N*
7	0	1	0	0	0	1	0	1	1	0	*P*
8	1	0	0	0	1	0	1	0	0	1	*N*
9	1	0	0	0	0	1	0	1	0	1	*P*
10	0	0	1	0	1	0	0	1	0	1	*P*
11	1	0	0	0	1	0	0	1	1	0	*P*
12	0	1	0	0	1	0	1	0	1	0	*P*
13	0	1	0	1	0	0	0	1	0	1	*P*
14	0	0	1	0	1	0	1	0	1	0	*N*

This kind of transformation is called **contrast coding** or **one-hot encoding** (OHE).

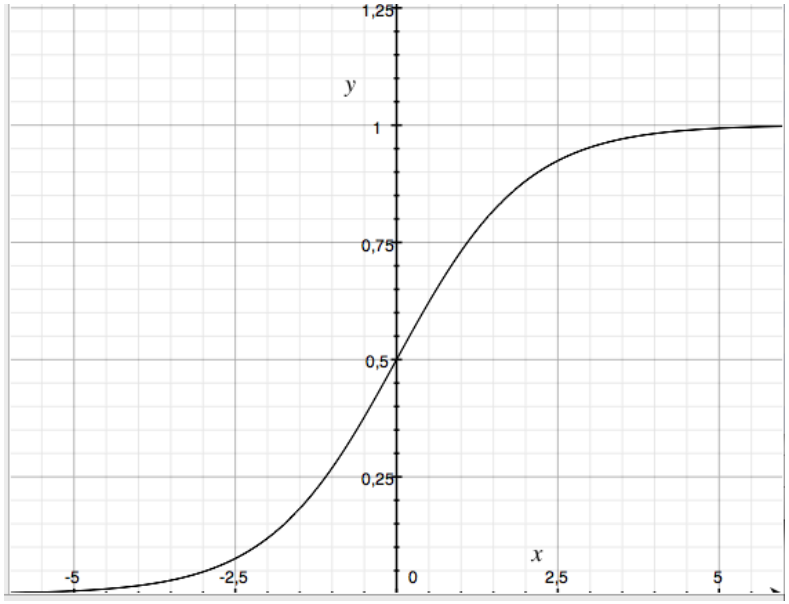


Matrix Notation

$$X = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \end{bmatrix}; \mathbf{y} = \begin{bmatrix} N \\ N \\ P \\ P \\ P \\ N \\ P \\ N \\ P \\ P \\ P \\ P \\ P \\ P \\ N \end{bmatrix}$$

Logistic Regression

The step function is not differentiable; that is why it is often replaced with the logistic curve (Verhulst, 1845, 1848):



$$\begin{aligned}\hat{y}(\mathbf{x}) &= Logistic(\mathbf{w}\cdot\mathbf{x}) \\ &= \frac{1}{1+e^{-\mathbf{w}\cdot\mathbf{x}}}\end{aligned}$$

Logistic Regression: The Idea

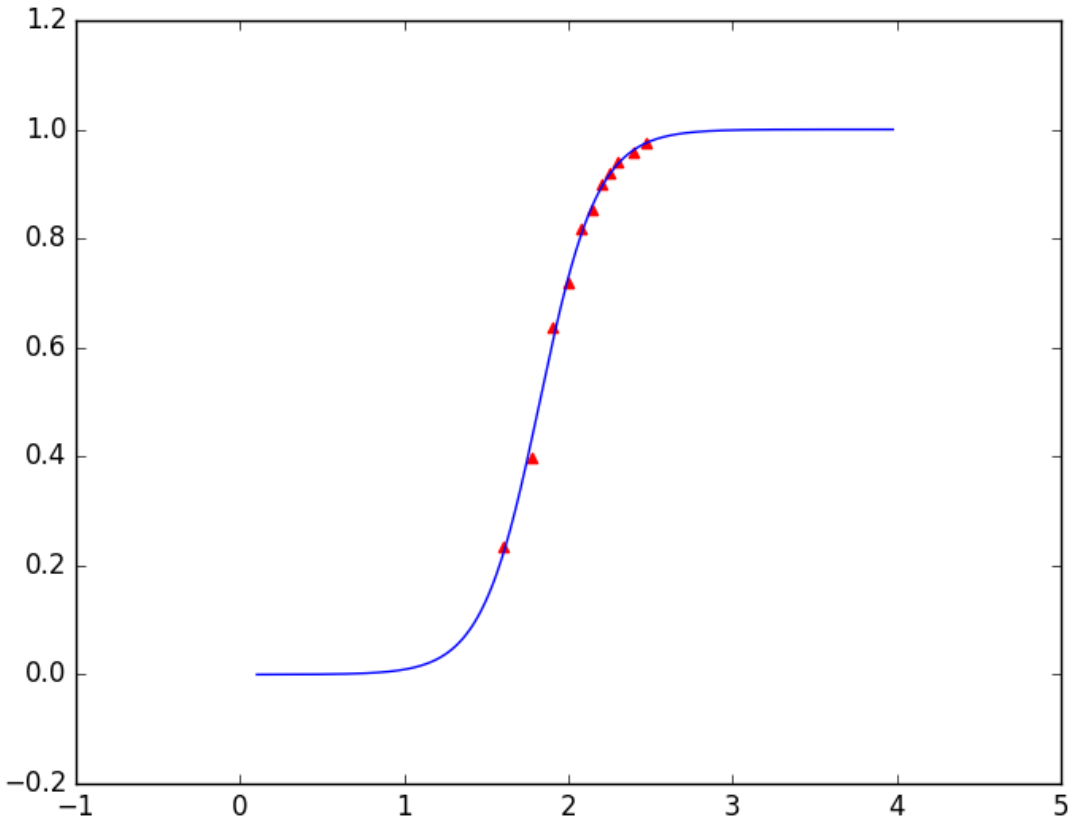
Drug concentration	Number exposed	Survive Class 0	Die Class 1	Mortality rate	Expected mortality
40	462	353	109	.2359	.2206
60	500	301	199	.3980	.4339
80	467	169	298	.6380	.6085
100	515	145	370	.7184	.7291
120	561	102	459	.8182	.8081
140	469	69	400	.8529	.8601
160	550	55	495	.9000	.8952
180	542	43	499	.9207	.9195
200	479	29	450	.9395	.9366
250	497	21	476	.9577	.9624
300	453	11	442	.9757	.9756

Table: A dataset. Adapted and simplified from the original article that described how to apply logistic regression to classification by Joseph Berkson, Application of the Logistic Function to Bio-Assay. *Journal of the American Statistical Association* (1944).

Fitting the Logistic Curve

Berkson fitted the curve to the data using the logarithm of the dose. He obtained:

$$\frac{1}{1+e^{-5.659746x+10.329884}}$$



Logistic Regression

Logistic regression attempts to model the probability of an observation $\mathbf{x}$ to belong to a class:

$$P(y=1|\mathbf{x}) = \frac{1}{1+e^{-\mathbf{w}\cdot\mathbf{x}}}$$

and

$$P(y=0|\mathbf{x}) = \frac{e^{-\mathbf{w}\cdot\mathbf{x}}}{1+e^{-\mathbf{w}\cdot\mathbf{x}}}$$

These probabilities are extremely useful in practice.
The logit assumption (Berkson, 1944) models the odd ratio as a hyperplane:

$$\ln \frac{P(y=1|\mathbf{x})}{P(y=0|\mathbf{x})} = \ln \frac{P(y=1|\mathbf{x})}{1-P(y=1|\mathbf{x})} = \mathbf{w}\cdot\mathbf{x}$$

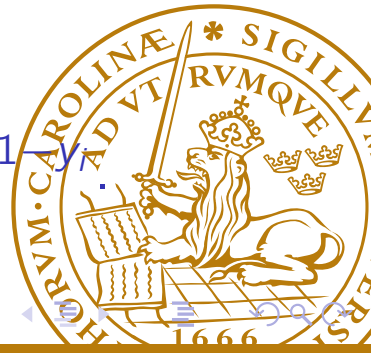
Likelihood of a Classification

Given a dataset, DS, containing a partition in two classes, P $(y = 1)$ and N $(y = 0)$, and a model $\mathbf{w}$, the likelihood to have the classification observed in this dataset is:

$$\begin{aligned} L(w) &= \prod_{\mathbf{x}_i \in P} P(y_i = 1|\mathbf{x}_i) \times \prod_{\mathbf{x}_i \in N} P(y_i = 0|\mathbf{x}_i), \\ &= \prod_{\mathbf{x}_i \in P} P(y_i = 1|\mathbf{x}_i) \times \prod_{\mathbf{x}_i \in N} (1 - P(y_i = 1|\mathbf{x}_i)). \end{aligned}$$

We can rewrite the product using y^j as powers of the probabilities as $y_i = 0$, when $\mathbf{x}_i \in P$ and $y_i = 1$, when $\mathbf{x}_i \in N$:

$$\begin{aligned} L(w) &= \prod_{\mathbf{x}_i \in P} P(y_i = 1|\mathbf{x}_i)^{y_i} \times \prod_{\mathbf{x}_i \in N} (1 - P(y_i = 1|\mathbf{x}_i))^{1-y_i}, \\ &= \prod_{(\mathbf{x}_i, y_i) \in DS} P(y_i = 1|\mathbf{x}_i)^{y_i} \times (1 - P(y_i = 1|\mathbf{x}_i))^{1-y_i}. \end{aligned}$$



Maximizing the Likelihood

We train a model by maximizing the likelihood of the observed classification:

$$\hat{w} = \arg\max_{w} \prod_{\mathbf{x}_i \in DS} P(y_i = 1|\mathbf{x}_i)^{y_i} \times (1 - P(y_i = 1|\mathbf{x}_i))^{1-y_i}$$

To maximize this term, it is more convenient to work with sums rather than with products and we take the logarithm of it (log-likelihood):

$$\hat{w} = \arg\max_{w} \sum_{(\mathbf{x}_i, y_i) \in DS} y_i \ln P(y_i = 1|\mathbf{x}_i) + (1 - y_i) \ln(1 - P(y_i = 1|\mathbf{x}_i))$$

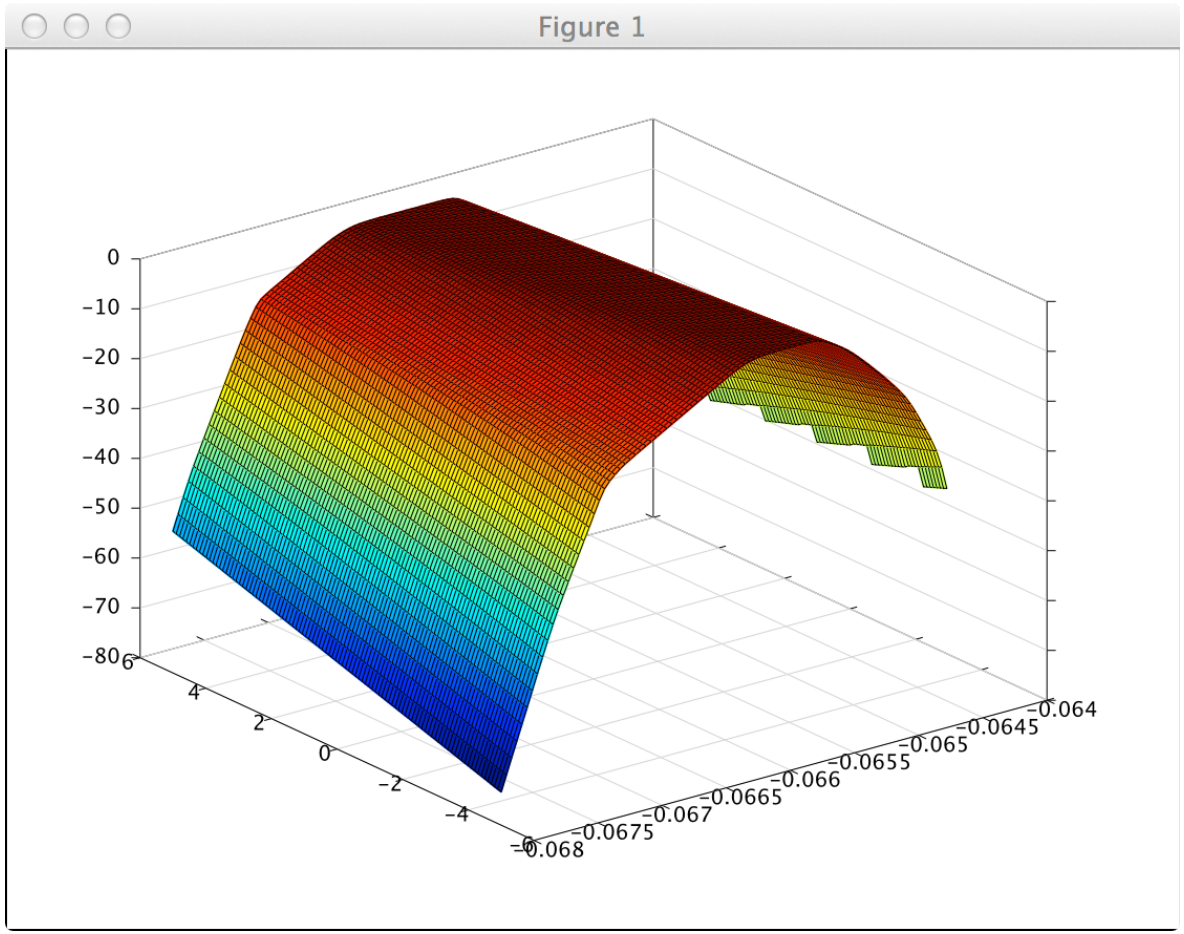
Using the logistic curves to express the probabilities, we have:

$$\hat{w} = \arg\max_{w} \sum_{(\mathbf{x}_i, y_i) \in DS} y_i \ln \frac{1}{1 + e^{-\mathbf{w}\cdot\mathbf{x}_i}} + (1 - y_i) \ln \frac{e^{-\mathbf{w}\cdot\mathbf{x}_i}}{1 + e^{-\mathbf{w}\cdot\mathbf{x}_i}}.$$

In contrast to linear regression that uses least mean squares, here we fit a logistic curve so that it maximizes the likelihood of the classification – partition – observed in the training set.

The Maximum

Graph of the probabilities from the *Salammbô* dataset.



The opposite is called the cross-entropy loss.

The Gradient Ascent

We can use the gradient ascent to compute the maximum.
Using a Taylor expansion: $\ell(\mathbf{w}+\mathbf{v}) = \ell(\mathbf{w}) + \mathbf{v}\cdot\nabla f(\mathbf{w}) + \ldots$
We have then: $\ell(\mathbf{w}+\alpha\nabla\ell(\mathbf{w})) \approx \ell(\mathbf{w}) + \alpha||\nabla\ell(\mathbf{w})||^2$.
The inequality:

$$\ell(\mathbf{w}) < \ell(\mathbf{w}+\alpha\nabla\ell(\mathbf{w}))$$

enables us to move one step up to the maximum.
We then use the iteration:

$$\mathbf{w}_{k+1} = \mathbf{w}_k + \alpha_k\nabla\ell(\mathbf{w}_k).$$

Computing the Model

We have:

$$\ell(\hat{y}, y) = y \ln \hat{y} + (1 - y) \ln(1 - \hat{y}),$$

where

$$\hat{y} = \frac{1}{1 + e^{-\mathbf{w} \cdot \mathbf{x}}}.$$

We can change this into a loss using the opposite:

$$BCELoss(\hat{\mathbf{y}}, \mathbf{y}) = -\frac{1}{q} \sum_{i=1}^{q} (y_i \ln \hat{y}_i + (1 - y_i) \ln(1 - \hat{y}_i)),$$

This is the usual notation.

Computing the Model (II)

The gradient of $\nabla_{\mathbf{w}}\ell(\hat{y}, y)$ is defined by the partial derivatives: $\dfrac{\partial \ell(\hat{y}_i, y_i)}{\partial w_j}$.

Using the chain rule, we have:

$$\begin{aligned}\frac{\partial \ell(\hat{y}_i, y_i)}{\partial w_j} &= \frac{\mathrm{d}\ell(\hat{y}_i, y_i)}{\mathrm{d}\hat{y}_i}\frac{\partial \hat{y}_i}{\partial w_j},\\ &= \frac{\mathrm{d}}{\mathrm{d}\hat{y}_i}(y_i \ln \hat{y}_i + (1-y_i)\ln(1-\hat{y}_i))\frac{\partial \hat{y}_i}{\partial w_j}.\end{aligned}$$

1. The derivative with respect to $\hat{y}_i$ is:

$$\begin{aligned}\frac{\mathrm{d}\ell(\hat{y}_i, y_i)}{\mathrm{d}\hat{y}_i} &= \frac{y_i}{\hat{y}_i} - \frac{1-y_i}{1-\hat{y}_i},\\ &= -\frac{\hat{y}_i - y_i}{\hat{y}_i(1-\hat{y}_i)}.\end{aligned}$$

2. For the second term, using the chain rule again, we have:

$$\frac{\partial \hat{y}_i}{\partial w_j} = \frac{\mathrm{d}\hat{y}_i}{\mathrm{d}\mathbf{w}\cdot\mathbf{x}_i}\frac{\partial \mathbf{w}\cdot\mathbf{x}_i}{\partial w_j}.$$

Computing the Model (III)

The first term of this product is the derivative of the logistic function:

$$\begin{aligned}\left(\frac{1}{1+e^{-x}}\right)' &= \frac{e^{-x}}{(1+e^{-x})^2},\\ &= \frac{1}{1+e^{-x}}\cdot\left(1-\frac{1}{1+e^{-x}}\right),\end{aligned}$$

while the second one is simply $x_{i,j}$. We have then:

$$\begin{aligned}\frac{\partial \hat{y}_i}{\partial w_j} &= \frac{\mathrm{d}\hat{y}_i}{\mathrm{d}\mathbf{w}\cdot\mathbf{x}_i}\frac{\partial \mathbf{w}\cdot\mathbf{x}_i}{\partial w_j},\\ &= \hat{y}_i\cdot(1-\hat{y}_i)\cdot x_{i,j}.\end{aligned}$$

Finally, we can rewrite the gradient parameters as:

$$\begin{aligned}\frac{\partial L(\hat{y}_i, y_i)}{\partial w_j} &= \frac{\mathrm{d}\ell(\hat{y}_i, y_i)}{\mathrm{d}\hat{y}_i}\frac{\partial \hat{y}_i}{\partial w_j},\\ &= -(\hat{y}_i - y_i)\cdot x_{i,j}.\end{aligned}$$

Weight Updates

For $DS = \{(1, x_{i,1}, x_{i,2}, ..., x_{i,n}, y_i) | i : 1..q\}$, the weight updates of logistic regression are then:

- Stochastic gradient ascent:

$$w_j \leftarrow w_j + \alpha \cdot x_{i,j} \cdot (y_i - \frac{1}{1 + e^{-\mathbf{w} \cdot \mathbf{x}_i}})$$

- Batch gradient ascent:

$$w_j \leftarrow w_j + \frac{\alpha}{q} \cdot \sum_{i=1}^{q} x_{i,j} \cdot (y_i - \frac{1}{1 + e^{-\mathbf{w} \cdot \mathbf{x}_i}})$$

As with gradient descent, we stop the iterations when:

$$||\nabla \ell(\mathbf{w})|| < \epsilon.$$

Loss: Binary Crossentropy

Alternatively, we can compute a loss and minimize it.
We just take the opposite value of the likelihood.
We then call the corresponding function the **log-loss** or **cross entropy** of $\hat{y}$ on y:
Machine learning practitioners and toolkits use the natural logarithm (i.e. $\ln$ and not $\log_2$) and minimize the mean:

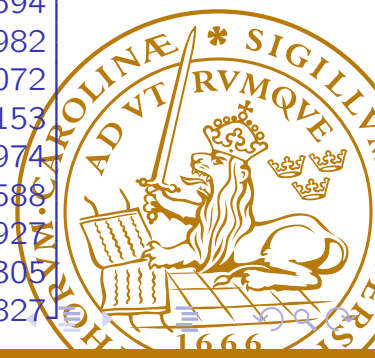
$$-\frac{1}{q}\sum_{j=1}^{q}(y^j \ln \hat{y}^j + (1-y^j)\ln(1-\hat{y}^j)).$$

Both approaches, maximum likelihood and binary crossentropy, are equivalent.
I feel the maximum likelihood one more intuitive, at least easier to visualize.

Crossentropy in Practice

Estimates obtained with Keras for the French/English dataset:

$$P = \begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 1 \\ 0 & 1 \\ 0 & 1 \\ 0 & 1 \\ 0 & 1 \\ 0 & 1 \\ 0 & 1 \\ 0 & 1 \\ 0 & 1 \\ 0 & 1 \\ 0 & 1 \\ 0 & 1 \\ 0 & 1 \\ 0 & 1 \end{bmatrix}; M = \begin{bmatrix} 0.9968438 & 0.00315617 \\ 0.8653501 & 0.13464986 \\ 0.79700726 & 0.20299272 \\ 0.93007773 & 0.06992232 \\ 0.99416685 & 0.00583313 \\ 0.89554983 & 0.10445023 \\ 0.9257818 & 0.07421817 \\ 0.96231836 & 0.03768164 \\ 0.9458056 & 0.05419444 \\ 0.74871475 & 0.2512853 \\ 0.7019538 & 0.2980462 \\ 0.897158 & 0.10284194 \\ 0.98296577 & 0.01703422 \\ 0.9072209 & 0.09277911 \\ 0.9971553 & 0.00284468 \\ 0.11270701 & 0.887293 \\ 0.03527627 & 0.96472377 \\ 0.4546863 & 0.54531366 \\ 0.03282152 & 0.9671785 \\ 0.19724798 & 0.80275196 \\ 0.01707165 & 0.98292834 \\ 0.19545405 & 0.80454594 \\ 0.00420181 & 0.9957982 \\ 0.02619275 & 0.9738072 \\ 0.00478472 & 0.9952153 \\ 0.00256019 & 0.99743974 \\ 0.04174117 & 0.9582588 \\ 0.06820729 & 0.9317927 \\ 0.01429696 & 0.98570305 \\ 0.47683164 & 0.52316827 \end{bmatrix}; CE = -\frac{1}{30}\sum \begin{bmatrix} 1 \cdot \ln 0.9968438 + 0 \cdot \ln 0.00315617 \\ 1 \cdot \ln 0.8653501 + 0 \cdot \ln 0.13464986 \\ 1 \cdot \ln 0.79700726 + 0 \cdot \ln 0.20299272 \\ 1 \cdot \ln 0.93007773 + 0 \cdot \ln 0.06992232 \\ 1 \cdot \ln 0.99416685 + 0 \cdot \ln 0.00583313 \\ 1 \cdot \ln 0.89554983 + 0 \cdot \ln 0.10445023 \\ 1 \cdot \ln 0.9257818 + 0 \cdot \ln 0.07421817 \\ 1 \cdot \ln 0.96231836 + 0 \cdot \ln 0.03768164 \\ 1 \cdot \ln 0.9458056 + 0 \cdot \ln 0.05419444 \\ 1 \cdot \ln 0.74871475 + 0 \cdot \ln 0.2512853 \\ 1 \cdot \ln 0.7019538 + 0 \cdot \ln 0.2980462 \\ 1 \cdot \ln 0.897158 + 0 \cdot \ln 0.10284194 \\ 1 \cdot \ln 0.98296577 + 0 \cdot \ln 0.01703422 \\ 1 \cdot \ln 0.9072209 + 0 \cdot \ln 0.09277911 \\ 1 \cdot \ln 0.9971553 + 0 \cdot \ln 0.00284468 \\ 0 \cdot \ln 0.11270701 + 1 \cdot \ln 0.887293 \\ 0 \cdot \ln 0.03527627 + 1 \cdot \ln 0.96472377 \\ 0 \cdot \ln 0.4546863 + 1 \cdot \ln 0.54531366 \\ 0 \cdot \ln 0.03282152 + 1 \cdot \ln 0.9671785 \\ 0 \cdot \ln 0.19724798 + 1 \cdot \ln 0.80275196 \\ 0 \cdot \ln 0.01707165 + 1 \cdot \ln 0.98292834 \\ 0 \cdot \ln 0.19545405 + 1 \cdot \ln 0.80454594 \\ 0 \cdot \ln 0.00420181 + 1 \cdot \ln 0.9957982 \\ 0 \cdot \ln 0.02619275 + 1 \cdot \ln 0.9738072 \\ 0 \cdot \ln 0.00478472 + 1 \cdot \ln 0.9952153 \\ 0 \cdot \ln 0.00256019 + 1 \cdot \ln 0.99743974 \\ 0 \cdot \ln 0.04174117 + 1 \cdot \ln 0.9582588 \\ 0 \cdot \ln 0.06820729 + 1 \cdot \ln 0.9317927 \\ 0 \cdot \ln 0.01429696 + 1 \cdot \ln 0.98570305 \\ 0 \cdot \ln 0.47683164 + 1 \cdot \ln 0.52316827 \end{bmatrix} = 0.1222.$$



Multinomial (Multiclass) Logistic Regression

Generalizing logistic regression from

$$\ln \frac{P(y=1|\mathbf{x})}{P(y=0|\mathbf{x})} = \mathbf{w} \cdot \mathbf{x}$$

to a multiclass setting is easy.
Using $y = 0$ as a pivot, we have for C classes:

$$\begin{aligned}
\ln \frac{P(y=1|\mathbf{x})}{P(y=0|\mathbf{x})} &= \mathbf{w}_1 \cdot \mathbf{x} \\
\ln \frac{P(y=2|\mathbf{x})}{P(y=0|\mathbf{x})} &= \mathbf{w}_2 \cdot \mathbf{x} \\
\ldots & \\
\ln \frac{P(y=C-1|\mathbf{x})}{P(y=0|\mathbf{x})} &= \mathbf{w}_{C-1} \cdot \mathbf{x}
\end{aligned}$$

and

$$\sum_{k=0}^{C-1} P(y=k|\mathbf{x}) = 1.$$

Multinomial (Multiclass) Logistic Regression (II)

We obtain:

$$P(y=0|\mathbf{x}) = \frac{1}{1+\sum_{i=1}^{C-1} e^{\mathbf{w}_i \cdot \mathbf{x}}}$$

$$P(y=k|\mathbf{x}) = \frac{e^{\mathbf{w}_k \cdot \mathbf{x}}}{1+\sum_{i=1}^{C-1} e^{\mathbf{w}_i \cdot \mathbf{x}}}, k = 1, 2, ..., C-1.$$

Equivalent to:

$$P(y=k|\mathbf{x}) = \frac{e^{\mathbf{w}_k \cdot \mathbf{x}}}{\sum_{i=0}^{C-1} e^{\mathbf{w}_i \cdot \mathbf{x}}}, k = 0, 1, 2, ..., C-1.$$

This function is called the *softmax* function.
Note that this is simply a renaming of Boltzmann's distribution (`https://en.wikipedia.org/wiki/Boltzmann_distribution`).

Loss: Categorical Crossentropy

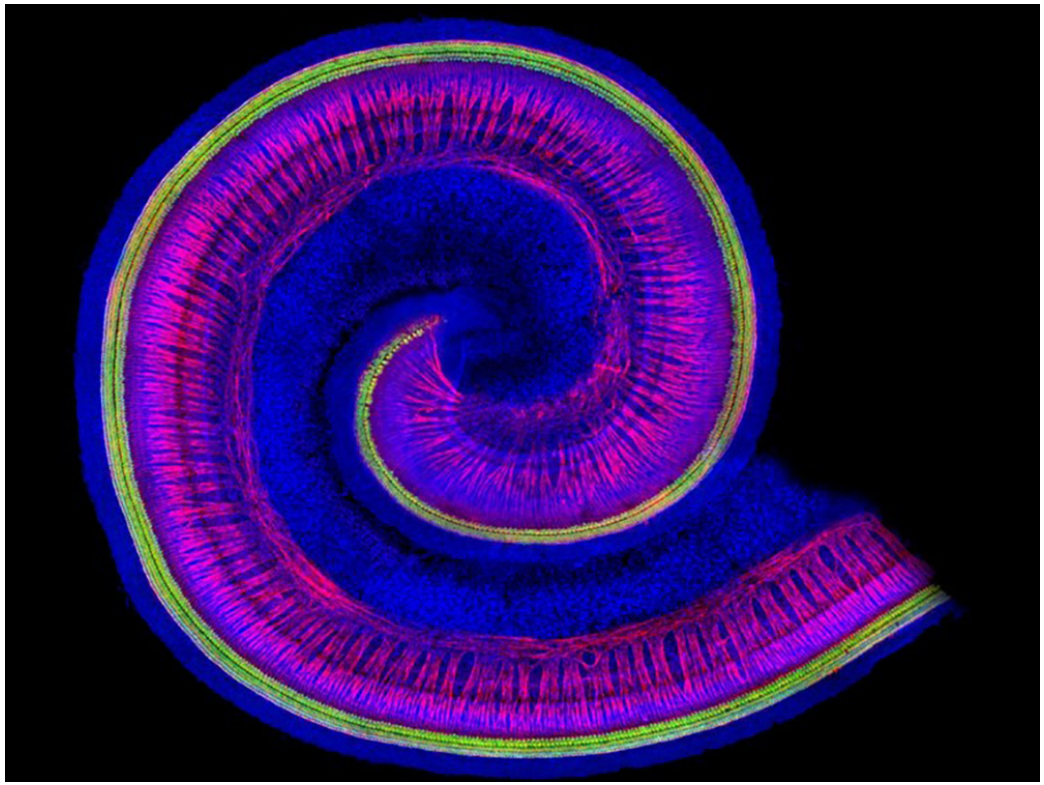
Binary logistic regression minimizes the crossentropy loss.
Categorical crossentropy covers multiple classes. It is defined as:

$$-\frac{1}{q}\sum_{j=1}^{q} y_j \cdot \ln \hat{y}_j$$

where y_j corresponds to the true class of the jth observation, represented as a one-hot vector, $\hat{y}_j$ is the probability distribution of the predicted classes.
The predicted probability for $\mathbf{x}$ to belong to class i is given by the softmax function:

$$P(y = i|\mathbf{x}) = \frac{e^{\mathbf{w}_i \cdot \mathbf{x}}}{\sum_{j=1}^{C} e^{\mathbf{w}_j \cdot \mathbf{x}}}.$$

Neural Networks



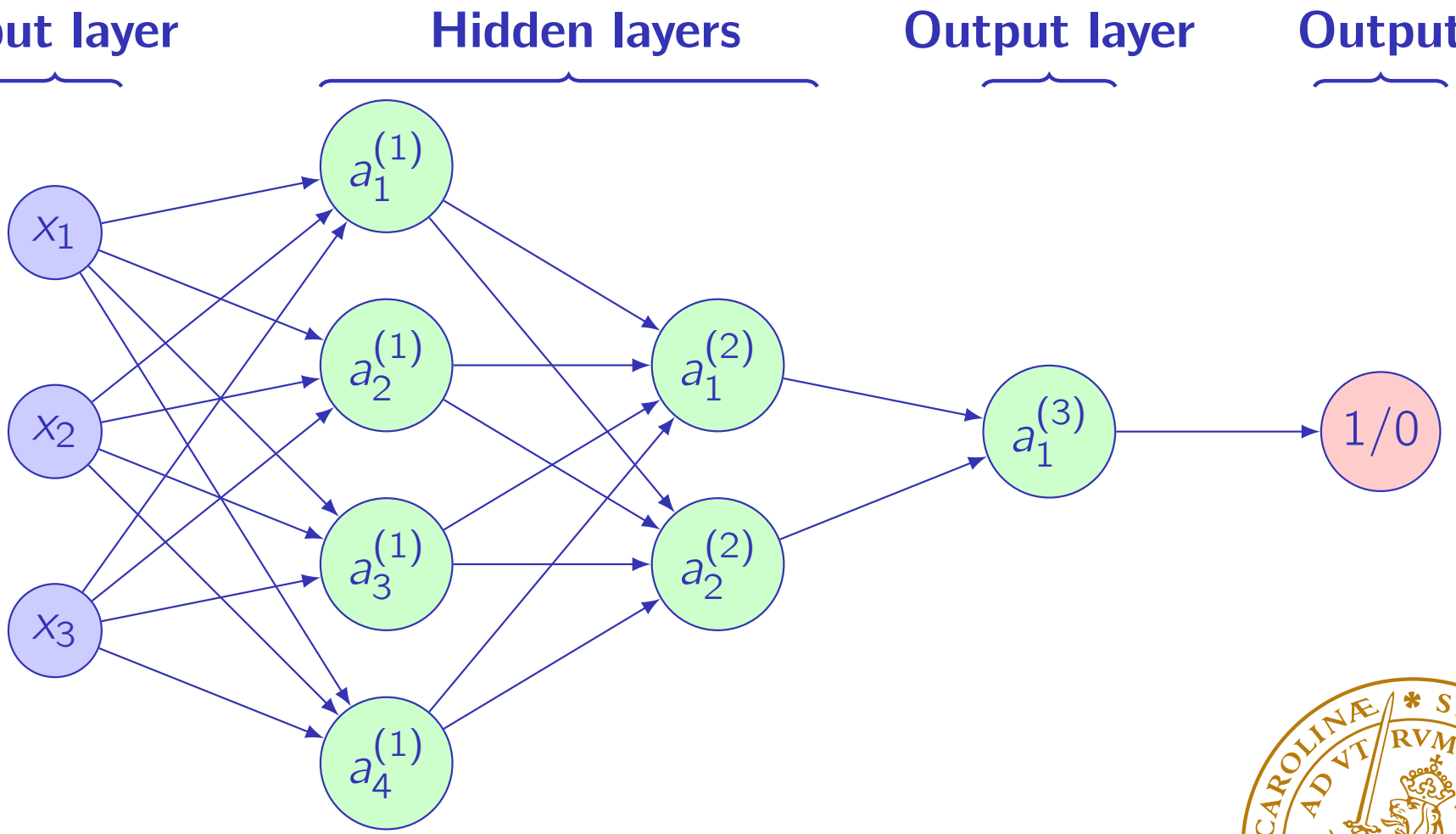
A photomicrograph showing the classic view of the snail-shaped cochlea with hair cells stained green and neurons showing reddish-purple. [Decibel Therapeutics (https://www.decibeltx.com)]. Source: https://www.genengnews.com/insights/targeting-the-inner-ear/

Neural Networks

Learning devices loosely inspired by brain neuron
Artificial neural networks abundantly use physiological metaphors, such as synapses, cortex, etc.
In our context, can be reframed as a multistage logistic regression to handle nonlinearities
Training is complex and demanding
Fell out of favor in the 1990-2000s. Experienced a revival with deep learning around 2010
Available programming environments include, Keras (on TensorFlow), TensorFlow, PyTorch, and PaddlePaddle:

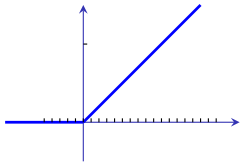
- `https://keras.io/`
- `https://www.tensorflow.org/`
- `http://pytorch.org/`
- `https://www.paddlepaddle.org.cn/`

Feed-Forward Neural Networks

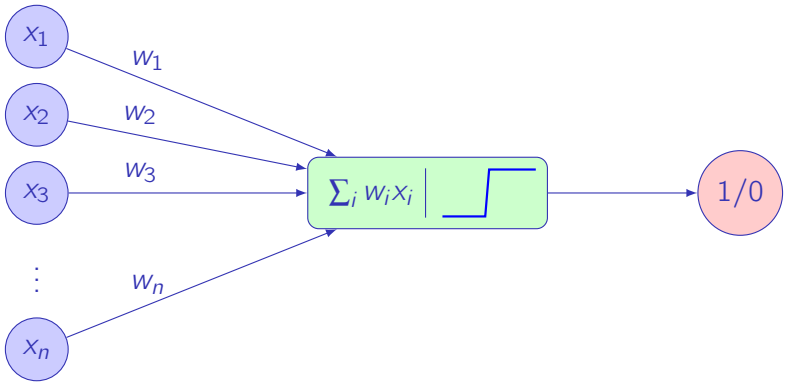


Activation Functions

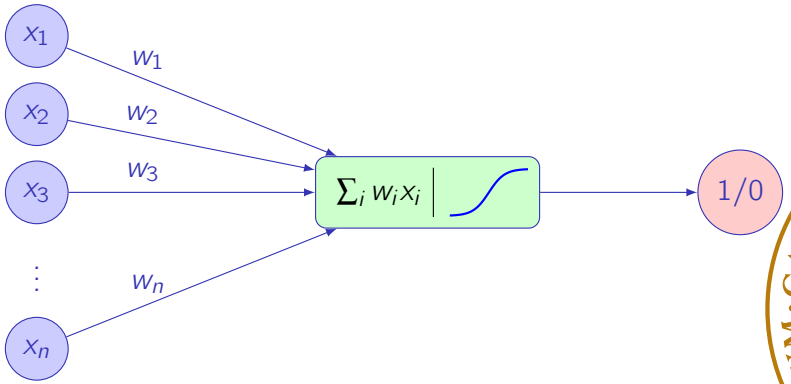
Rectified linear unit (ReLU)



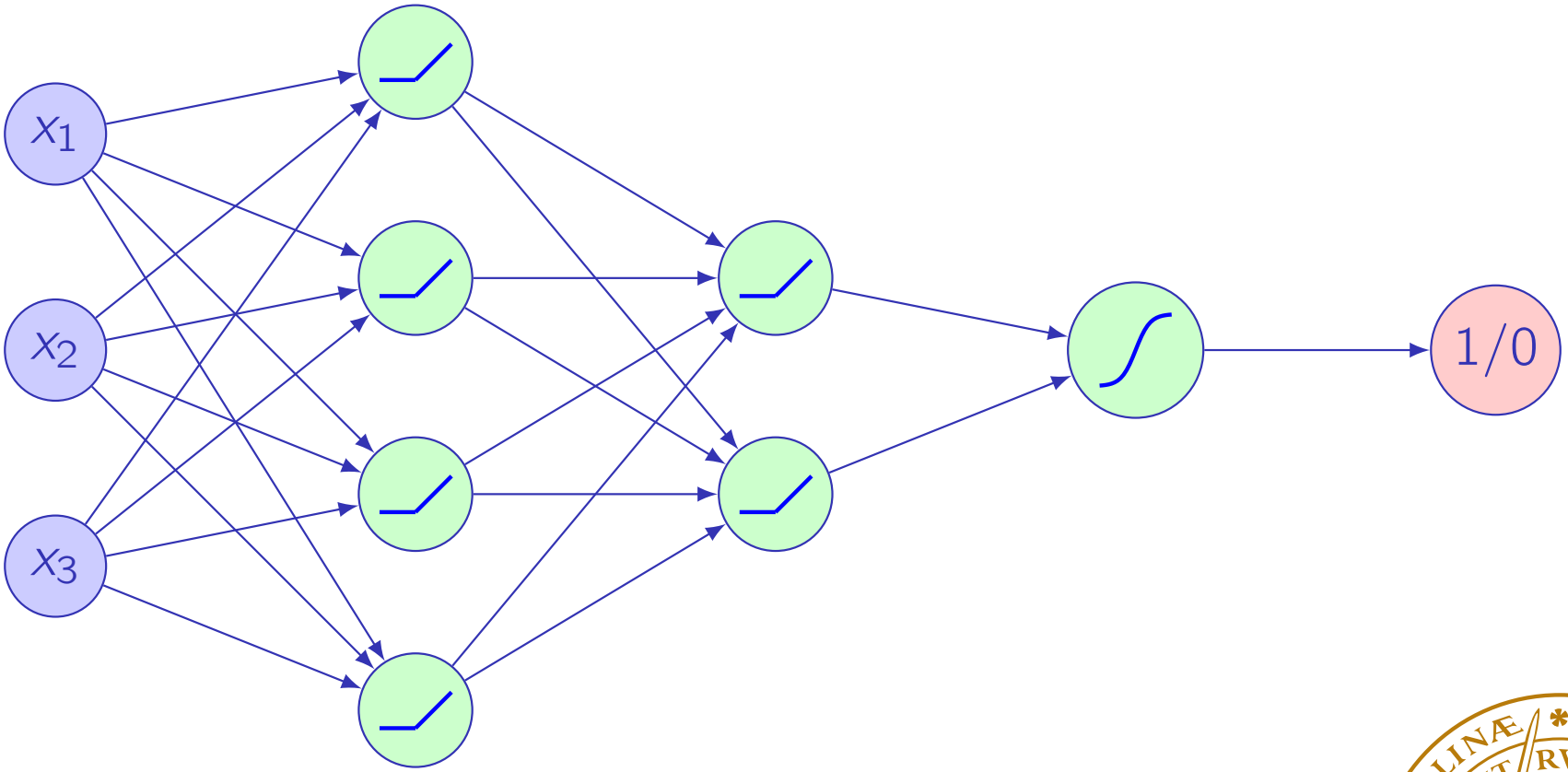
The perceptron



Logistic regression



Neural Networks with Hidden Layers



Ensemble Learning

Idea: Use many classifiers instead of one

Boosting is an example of ensemble learning

Uses a set of K weak classifiers and gives a weight to the examples: $w(i)$

This means that example i counts as $w(i)$ examples

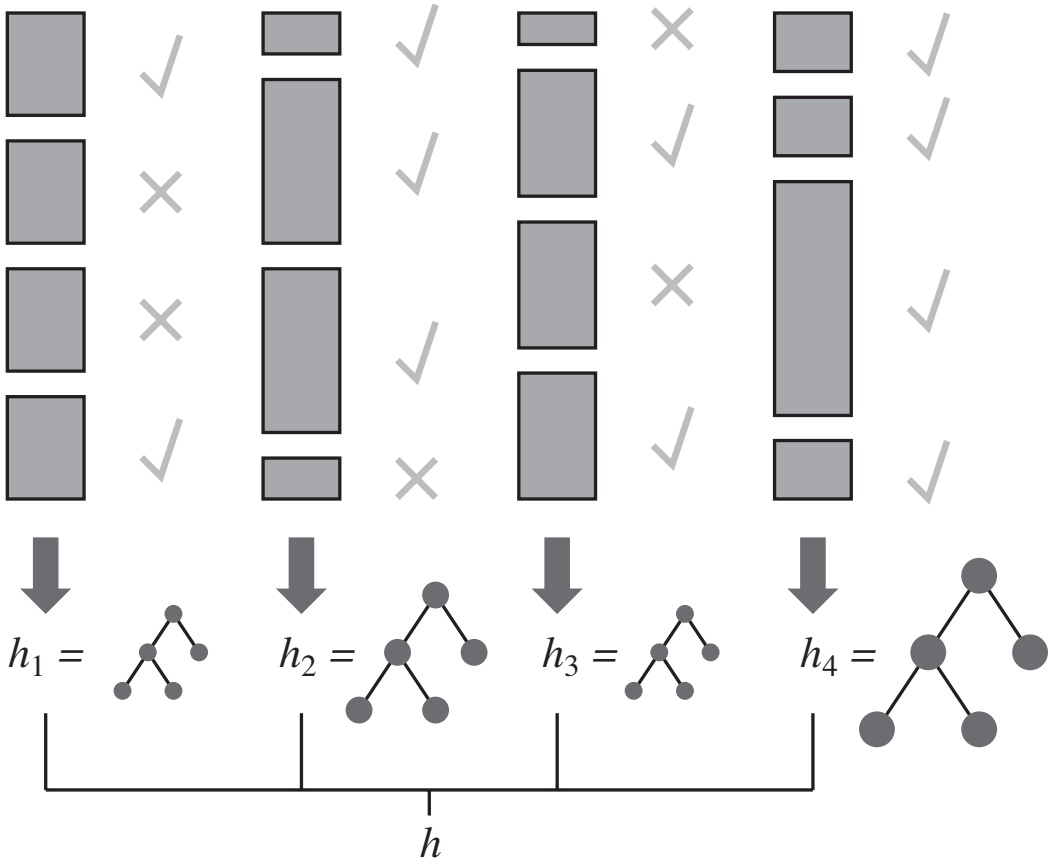
Decision stumps (decision trees with one level) are popular weak classifiers

The idea of the AdaBoost algorithm is:

1. Learn a first stump on the training set. Each example has a weight of 1
2. Set a higher weight to misclassified examples (think of just duplicating them in the dataset)
3. Learn a second decision stump.
4. and so on until we have K decision stumps

The final classifier is a weighted combination of the K classifiers, where the weights are computed from the performance of each individual classifier.

Ensemble Learning



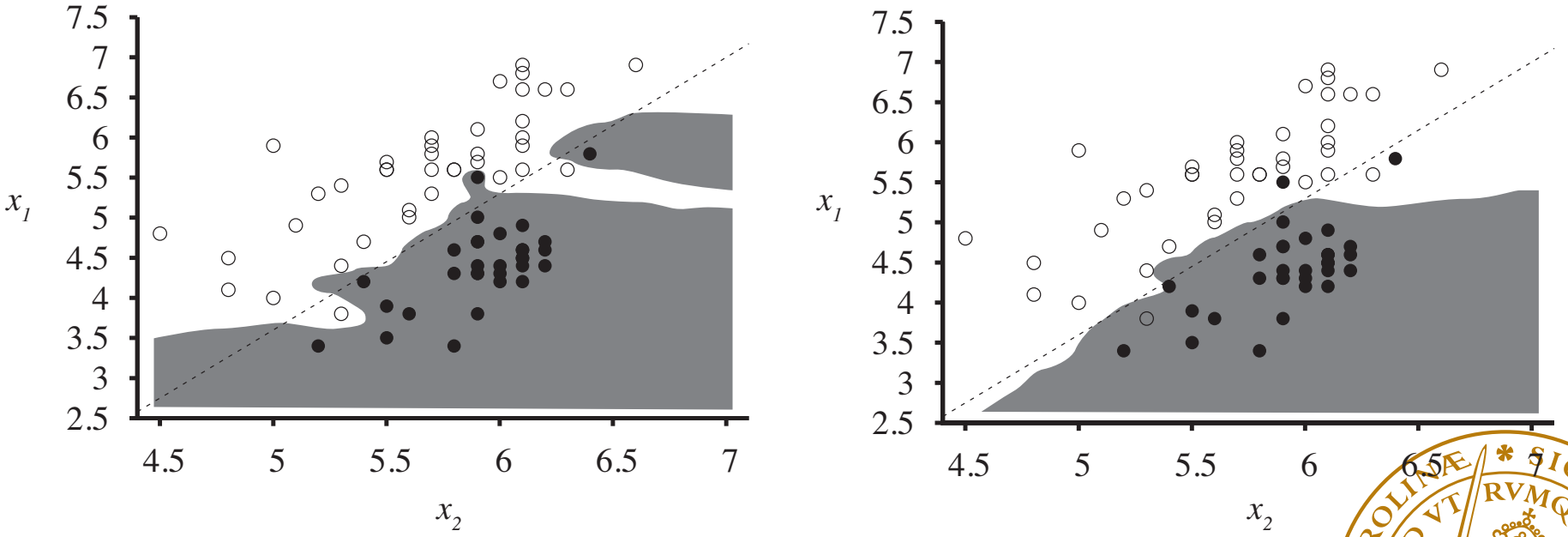
From the textbook: Stuart Russell and Peter Norvig, *Artificial Intelligence*, 4th ed., 2022, page 719.

Nonparametric Models

When there is no parametric model, such as a hyperplane, we can try to search the k nearest neighbors of $\mathbf{x}$: $NN(k, \mathbf{x})$

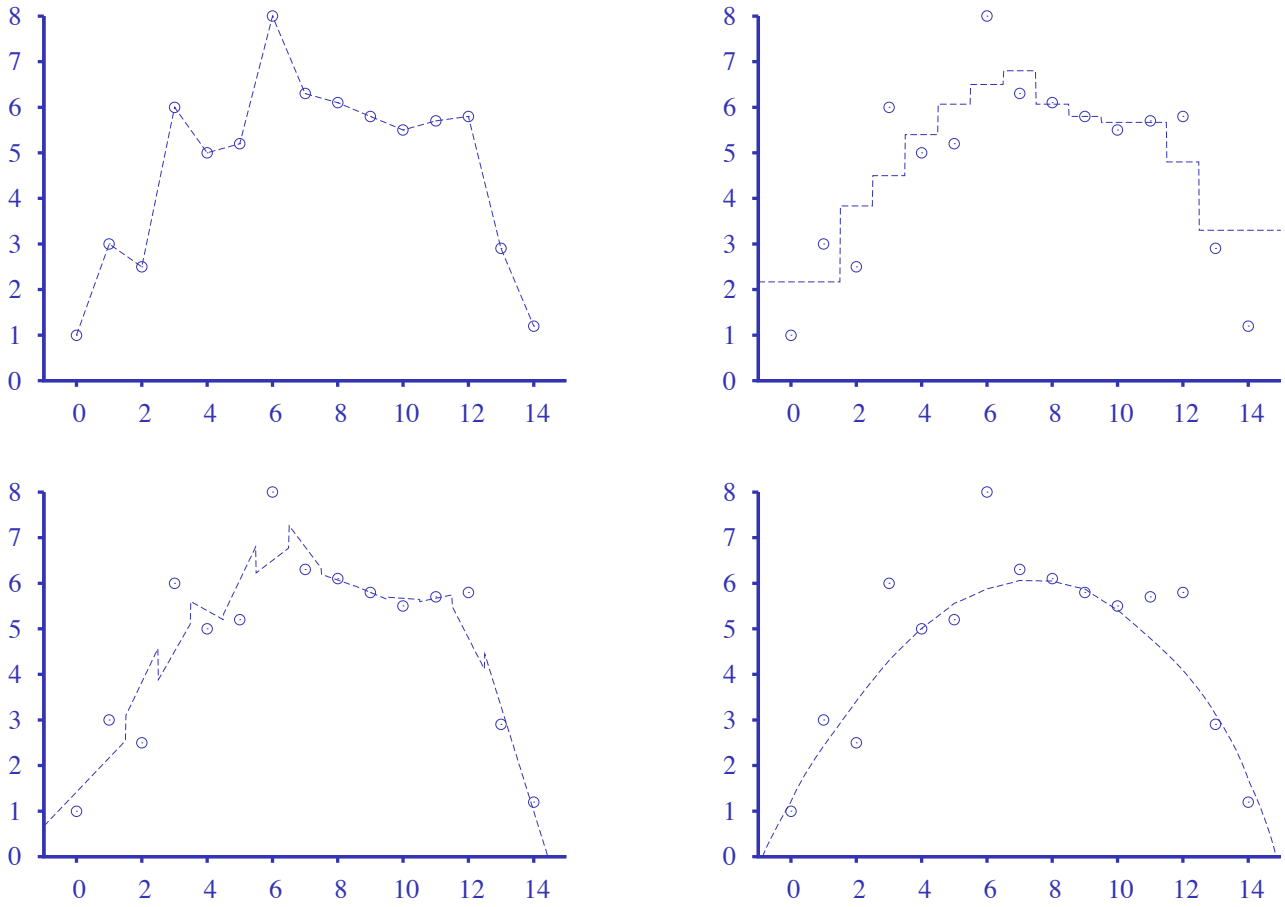
Given a position in the space, we decide on a class by looking at the neighbor's class

The distance is given by: $L^p(\mathbf{x}, \mathbf{y}) = \left(\sum_i |x_i - y_i|^p\right)^{\frac{1}{p}}$



k nearest neighbors with $k = 1$ and $k = 5$. From the textbook: Stuart Russell and Peter Norvig, *Artificial Intelligence*, 4th ed., 2022, page 705.

Nonparametric Regression



From the textbook: Stuart Russell and Peter Norvig, *Artificial Intelligence*, 4th ed., 2022, page 708.

scikit-learn

scikit-learn is a popular and comprehensive machine–learning library in Python built on top of numpy.
The classifiers use two main functions: `fit()` to train a model and `predict()` to predict a class. In the scikit-learn documentation, the functions adopt a notation, where:

- **x** denotes a feature vector (the predictors) describing one observation and X, a feature matrix representing the dataset;
- y denotes the class (or response or target) of one observation and **y**, the class vector for the whole dataset.

scikit-learn: Data

```
import numpy as np

y_train = np.array(
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
     1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1])

X_train = np.array(
    [[35680, 2217], [42514, 2761], [15162, 990], [35298, 2274],
     [29800, 1865], [40255, 2606], [74532, 4805], [37464, 2396],
     [31030, 1993], [24843, 1627], [36172, 2375], [39552, 2560],
     [72545, 4597], [75352, 4871], [18031, 1119], [36961, 2503],
     [43621, 2992], [15694, 1042], [36231, 2487], [29945, 2014],
     [40588, 2805], [75255, 5062], [37709, 2643], [30899, 2126],
     [25486, 1784], [37497, 2641], [40398, 2766], [74105, 5047],
     [76725, 5312], [18317, 1215]
     ])
```

scikit-learn: Training a Model

We then select and fit a model, here logistic regression with default parameters, with the lines:

```
from sklearn import linear_model


classifier = linear_model.LogisticRegression()
model = classifier.fit(X_train, y_train)
```

Once the model is trained, we can apply it to new observations. The next instruction just reapplies it to the training set:

```
y_test_predicted = classifier.predict(X_train)
```

which predicts exactly the classes we had in this set:

```
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1]
```

scikit-learn: Crossvalidation

scikit-learn has built-in cross validation functions. The code below shows an example of it with a 5-fold cross validation and a score corresponding to the accuracy:

```
from sklearn.model_selection import cross_val_score

scores = cross_val_score(classifier, X_train, y_train, cv=5,
                         scoring='accuracy')
print('Score', scores.mean())
```

scikit-learn: Nominal Data

$$X = \begin{bmatrix} \text{Sunny} & \text{Hot} & \text{High} & \text{False} \\ \text{Sunny} & \text{Hot} & \text{High} & \text{True} \\ \text{Overcast} & \text{Hot} & \text{High} & \text{False} \\ \text{Rain} & \text{Mild} & \text{High} & \text{False} \\ \text{Rain} & \text{Cool} & \text{Normal} & \text{False} \\ \text{Rain} & \text{Cool} & \text{Normal} & \text{True} \\ \text{Overcast} & \text{Cool} & \text{Normal} & \text{True} \\ \text{Sunny} & \text{Mild} & \text{High} & \text{False} \\ \text{Sunny} & \text{Cool} & \text{Normal} & \text{False} \\ \text{Rain} & \text{Mild} & \text{Normal} & \text{False} \\ \text{Sunny} & \text{Mild} & \text{Normal} & \text{True} \\ \text{Overcast} & \text{Mild} & \text{High} & \text{True} \\ \text{Overcast} & \text{Hot} & \text{Normal} & \text{False} \\ \text{Rain} & \text{Mild} & \text{High} & \text{True} \end{bmatrix} ; \mathbf{y} = \begin{bmatrix} \text{N} \\ \text{N} \\ \text{P} \\ \text{P} \\ \text{P} \\ \text{N} \\ \text{P} \\ \text{N} \\ \text{P} \\ \text{P} \\ \text{P} \\ \text{P} \\ \text{P} \\ \text{N} \end{bmatrix}$$

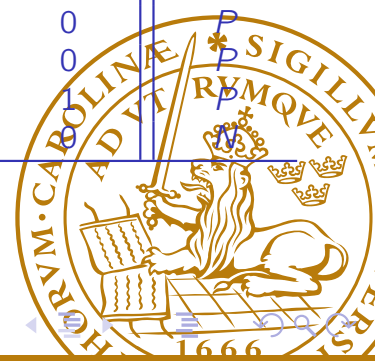
Converting Symbolic Attributes into Numerical Vectors

Linear classifiers are numerical systems.

Symbolic – nominal – attributes are mapped onto vectors of binary values.

A conversion of the weather dataset.

Object	Attributes										Class
	Outlook			Temperature			Humidity		Windy		
	Sunny	Overcast	Rain	Hot	Mild	Cool	High	Normal	True	False	
1	1	0	0	1	0	0	1	0	0	1	*N*
2	1	0	0	1	0	0	1	0	1	0	*N*
3	0	1	0	1	0	0	1	0	0	1	*P*
4	0	0	1	0	1	0	1	0	0	1	*P*
5	0	0	1	0	0	1	0	1	0	1	*P*
6	0	0	1	0	0	1	0	1	1	0	*N*
7	0	1	0	0	0	1	0	1	1	0	*P*
8	1	0	0	0	1	0	1	0	0	1	*N*
9	1	0	0	0	0	1	0	1	0	1	*P*
10	0	0	1	0	1	0	0	1	0	1	*P*
11	1	0	0	0	1	0	0	1	1	0	*P*
12	0	1	0	0	1	0	1	0	1	0	*P*
13	0	1	0	1	0	0	0	1	0	1	*P*
14	0	0	1	0	1	0	1	0	1	0	*N*



scikit-learn: Vectorizing the Data

To convert the data to numbers, store X in a list of dictionaries and use `DictVectorizer`:

```
from sklearn.feature_extraction import DictVectorizer

vec = DictVectorizer(sparse=False) # Should be true
X = vec.fit_transform(X_dict)
```

scikit-learn: Data Vectorization

```
array([[ 1.,  0.,  0.,  0.,  1.,  0.,  1.,  0.,  1.,  0.],
       [ 1.,  0.,  0.,  0.,  1.,  0.,  1.,  0.,  0.,  1.],
       [ 1.,  0.,  1.,  0.,  0.,  0.,  1.,  0.,  1.,  0.],
       [ 1.,  0.,  0.,  1.,  0.,  0.,  0.,  1.,  1.,  0.],
       [ 0.,  1.,  0.,  1.,  0.,  1.,  0.,  0.,  1.,  0.],
       [ 0.,  1.,  0.,  1.,  0.,  1.,  0.,  0.,  0.,  1.],
       [ 0.,  1.,  1.,  0.,  0.,  1.,  0.,  0.,  0.,  1.],
       [ 1.,  0.,  0.,  0.,  1.,  0.,  0.,  1.,  1.,  0.],
       [ 0.,  1.,  0.,  0.,  1.,  1.,  0.,  0.,  1.,  0.],
       [ 0.,  1.,  0.,  1.,  0.,  0.,  0.,  1.,  1.,  0.],
       [ 0.,  1.,  0.,  0.,  1.,  0.,  0.,  1.,  0.,  1.],
       [ 1.,  0.,  1.,  0.,  0.,  0.,  0.,  1.,  0.,  1.],
       [ 0.,  1.,  1.,  0.,  0.,  0.,  1.,  0.,  1.,  0.],
       [ 1.,  0.,  0.,  1.,  0.,  0.,  0.,  1.,  0.,  1.]])
```

Further Resources

A fine videolecture: Peter Norvig, *Statistical Learning as the Ultimate Agile Development Tool*
https://videolectures.net/cikm08_norvig_slatuad/

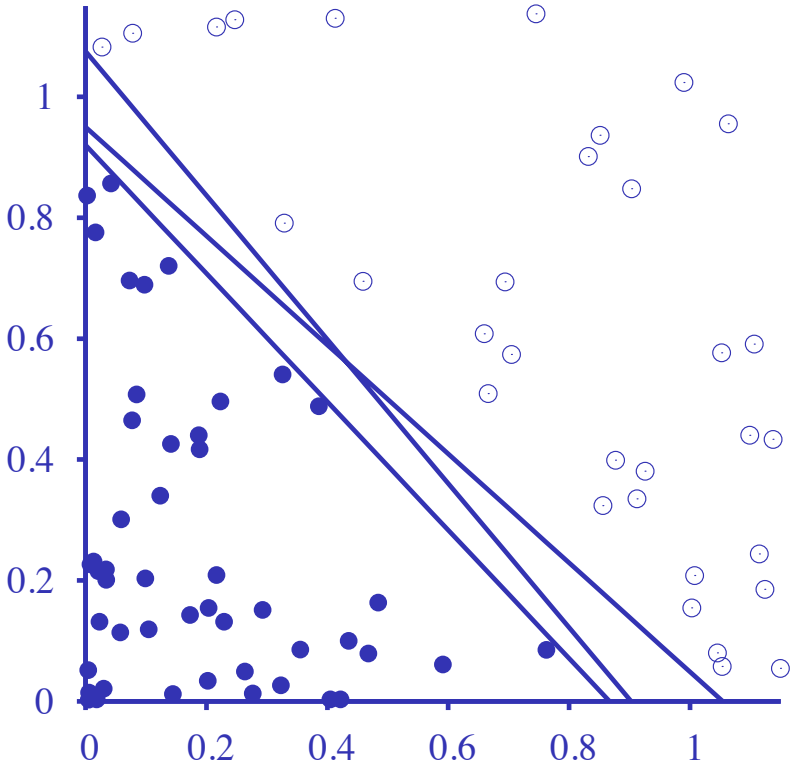
Online courses: openclassroom.stanford.edu, udacity.com, coursera.org, edx.org, etc.

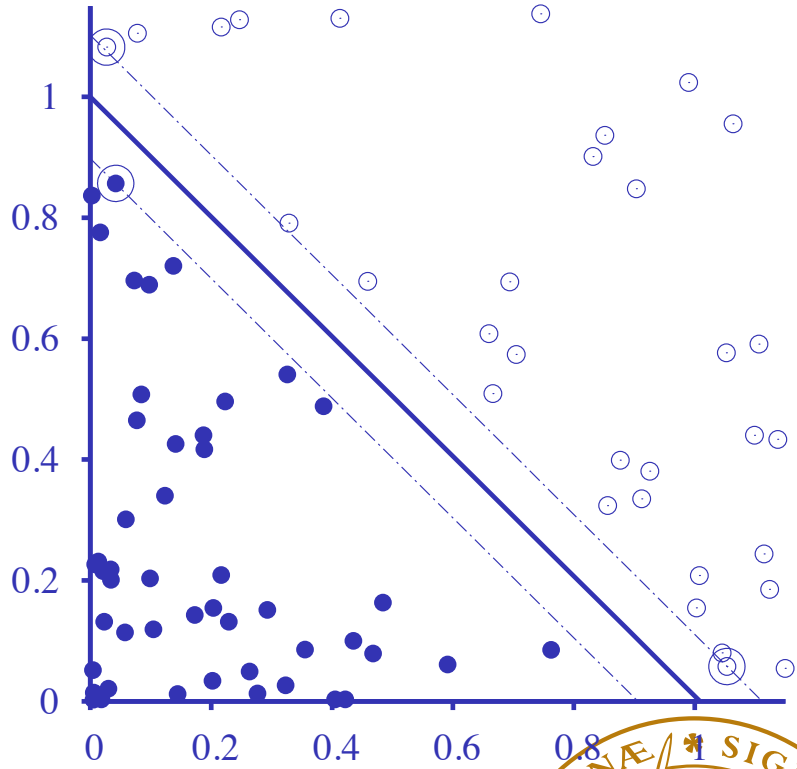
Machine learning: Two excellent books to go further: *An Introduction to Statistical Learning* (2nd ed., start with this one) and *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*
https://hastie.su.domains/pub.htm, see also *Probabilistic Machine Learning: An Introduction*,
https://probml.github.io/pml-book/book1.html

Deep learning: tutorials from Keras (https://keras.io/) and PyTorch (https://pytorch.org). See also *Deep learning*, by Goodfellow, Bengio, and Courville, https://www.deeplearningbook.org/

Tips from the expert: An interview with #1 on Kaggle:
https://blog.kaggle.com/2018/05/07/profiling-top-kagglers-bestfitting-currently-1-in-the-world/

Support Vector Machines (Optional)





From the textbook: Stuart Russell and Peter Norvig, *Artificial Intelligence*, 3rd ed., 2010, page 745.

Support Vector Machines (II) (Optional)

Support vector machines (SVM) maximize the margin between the separating hyperplane and the examples
The hyperplane is set at equal distance between the closest points of each class
The closest points are called the support vectors
Support vector machines can handle nonseparable examples using a soft margin or kernels.

Problem Description (Optional)

With two classes, support vector machines use the notation: $y = +1$ or $y = -1$.
We know from geometry and vector analysis that the distance of a point A to a hyperplane Hyp defined by the equation:

$$w_0 + w_1 x_1 + w_2 x_2 + \ldots + w_n x_n = 0$$

is given by the formula:

$$d(A, Hyp) = \frac{|w_0 + w_1 x_1 + w_2 x_2 + \ldots + w_n x_n|}{\sqrt{w_1^2 + w_2^2 + \ldots + w_n^2}}.$$

We want to find the hyperplane maximizing b so that:

$$y^j \cdot \frac{w_0 + \mathbf{w} \cdot \mathbf{x^j}}{||\mathbf{w}||} \geqslant b$$

for all the points in the dataset and with $\mathbf{w} = (w_1, \ldots, w_n)$.

Maximizing the Margin (Optional)

We normalize the equation with $b = \frac{1}{||\mathbf{w}||}$.
To find the maximal margin b, we minimize $||\mathbf{w}||^2$ with the constraint:

$$y^j \cdot (w_0 + \mathbf{w} \cdot \mathbf{x^j}) \geqslant 1,$$

where y^j values are either +1 or -1 depending on the class of the example.
We can solve this using gradient descent and hinge loss defined as:

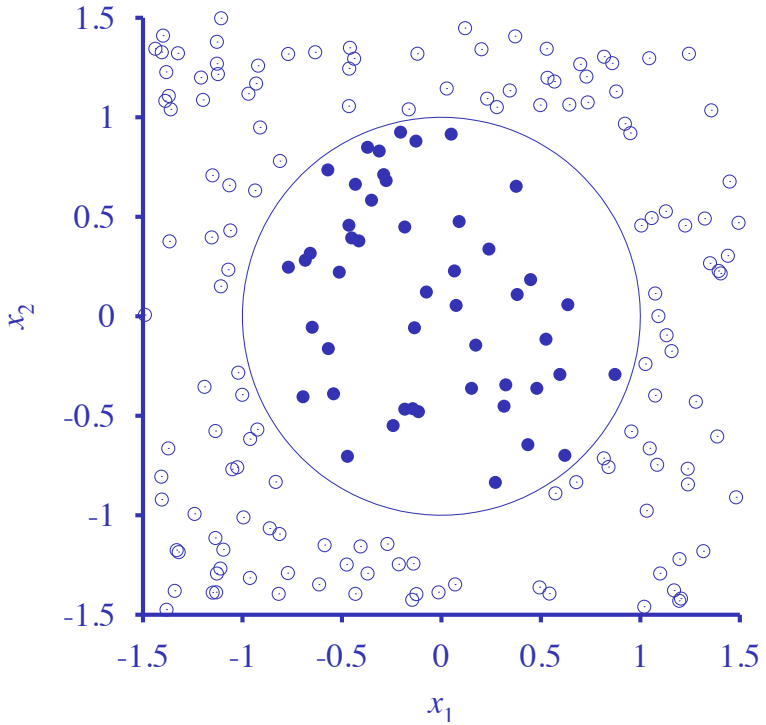
$$\max\{0, 1 - y\mathbf{w}^T\mathbf{x}\}, \quad y \in \{-1, +1\},$$

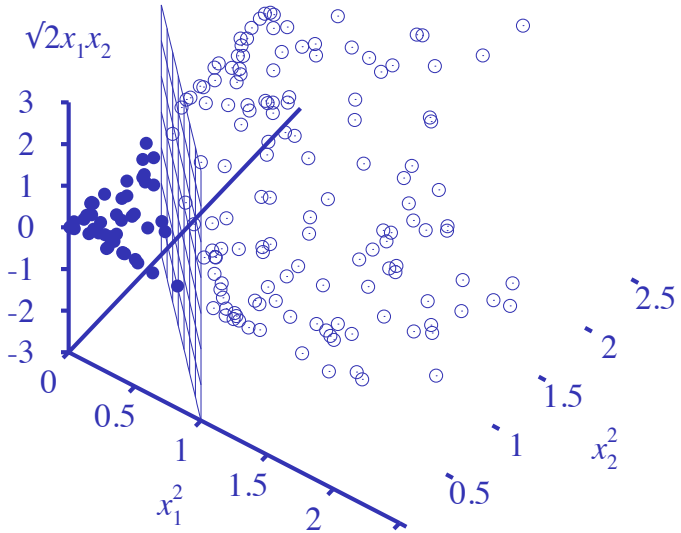
where the gradient is defined as:

$$\nabla L_{hinge} = \begin{cases} -y \cdot \mathbf{x} & \text{if } y\mathbf{w}^T\mathbf{x} < 1, \\ 0 & \text{otherwise.} \end{cases}$$

Kernels (Optional)

Many datasets are not linearly separable as in left-hand side figure.





From the textbook: Stuart Russell and Peter Norvig, *Artificial Intelligence*, 3rd ed., 2010, page 747.

It is always possible to find a space of higher dimension, where the points will be separable, here 3.

The Kernel Trick (Optional)

If we map the input data (x_1, x_2) to $(x_1^2, x_2^2, \sqrt{2}x_1x_2)$, we produce linearly separable classes:

$$F(x_1, x_2) = (x_1^2, x_2^2, \sqrt{2}x_1x_2)$$

We replace $\mathbf{x_j}\mathbf{x_t}$ with $F(\mathbf{x_j})F(\mathbf{x_t})$ in the Lagrangian:

$$L(,) = \sum_{j=1}^{q} \alpha_j - \frac{1}{2}\sum_{j,t}^{q} \alpha_j \alpha_t y^j y^t F(\mathbf{x_j})F(\mathbf{x_t}).$$

In the case of F, we have

$$\begin{aligned} F(\mathbf{x}) \cdot F(\mathbf{z}) &= x_1^2 z_1^2 + x_2^2 z_2^2 + 2x_1x_2z_1z_2 \\ &= (\mathbf{x} \cdot \mathbf{z})^2 \end{aligned}$$

In many cases, we can replace $\mathbf{x_j} \cdot \mathbf{x_t}$ with a kernel function $K(\mathbf{x_j}, \mathbf{x_t})$ like $K(\mathbf{x}, \mathbf{z}) = (\mathbf{x} \cdot \mathbf{z})^2$