EDAN96
Applied Machine Learning
Lecture 7: Linear Classification with Logistic Regression

Pierre Nugues

Pierre.Nugues@cs.lth.se

November 24, 2025

Content

Overview and practice of the major neural network architectures:

- Datasets
- Regression
- Gradient descent
- Logistic regression

We will use:

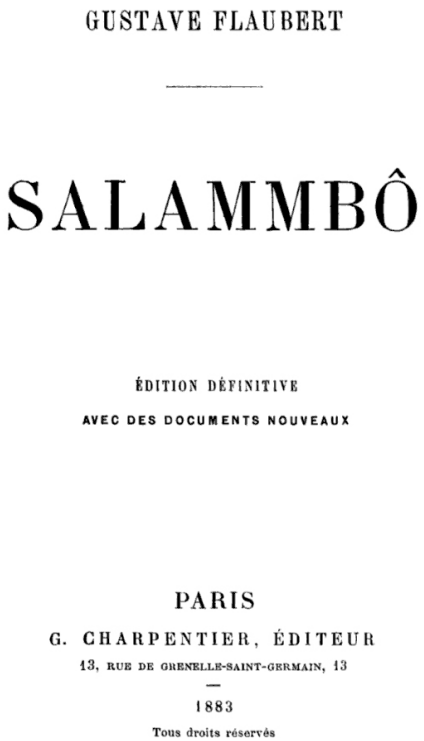
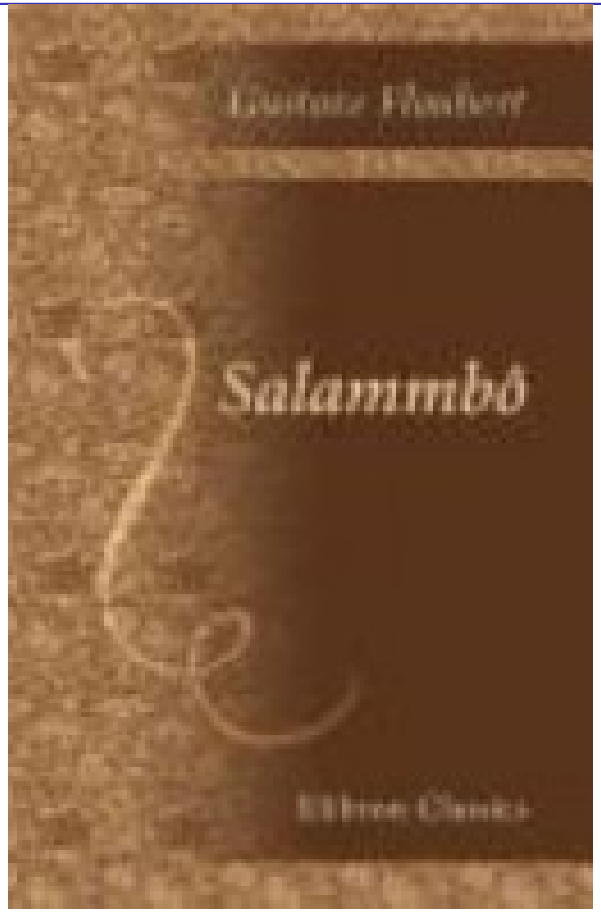
- PyTorch, `https://pytorch.org`, a powerful API to design and train network, and
- scikit-learn, `https://scikit-learn.org/stable/`, a general purpose machine-learning toolkit.

Some Definitions

1. Machine learning always starts with **datasets**: a collection of objects or observations.
2. Machine-learning algorithms can be classified along two main lines: **supervised** and **unsupervised** classification.
3. Supervised algorithms need a **training set**, where the objects are described in terms of attributes and belong to a known class or have a known output.
4. The performance of the resulting classifier is measured against a **test set**.
5. We can also use N-fold cross validation, where the test set is selected randomly from the training set N times, usually 10.
6. Unsupervised algorithms consider objects, where no class is provided.
7. Unsupervised algorithms learn regularities in datasets.

A Text Dataset: *Salammbô*

A corpus is a collection – a body – of texts.

French original	English translation
	

Supervised Learning

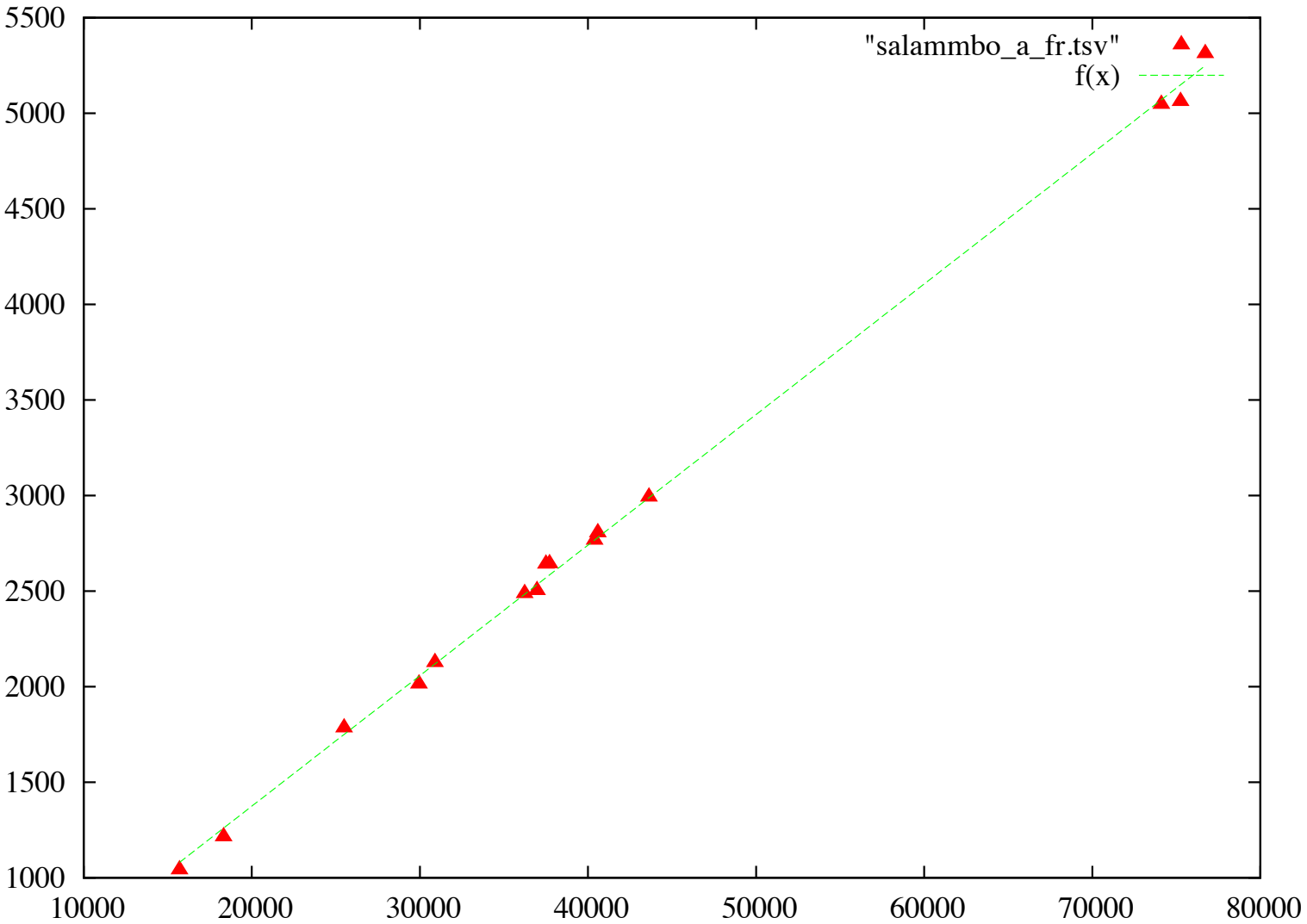
Letter counts from *Salammbô*

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

Datasets: `https://github.com/pnugues/pnlp/tree/main/datasets`

Supervised Learning: Regression

Letter count from *Salammbô* in French



Models

We will assume that data sets are governed by functions or models. For instance given the set:

$$\{(\mathbf{x}_i, y_i) | 0 < i \leqslant N\},$$

there exists a function such that:

$$f(\mathbf{x}_i) = y_i.$$

Supervised machine learning algorithms will produce hypothesized functions or models fitting the data.

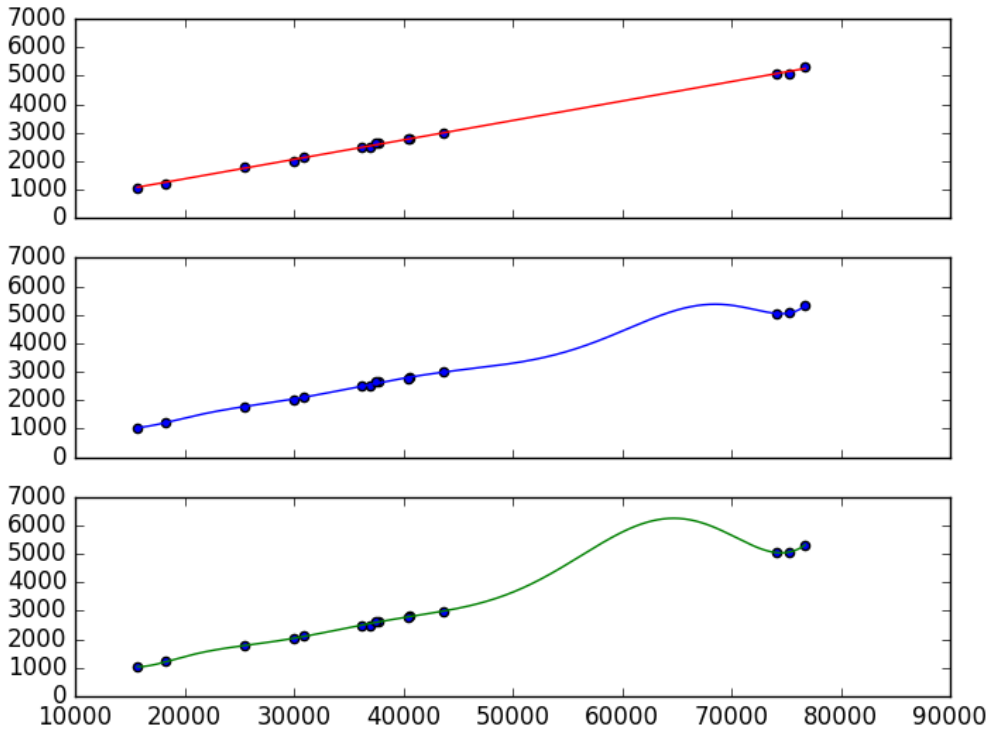
Notations

We will follow these notations:

- $\mathbf{x}$, the vector representing an observation (also sample, input, example, or predictor);
  in *Salammbô*, an observation is the number of letters in a chapter. We have 15 observations;
- y, the observed response (or target, or output); in programs, the variable names are `y` or `y_true`;
  in *Salammbô*, the number of *A*s in a chapter. We have 15 responses;
- $\hat{y}$, the value predicted by the model; in programs, the variable names are `y_pred` or `y_hat`;
- $\mathbf{w}$, the weights or parameters of the model, so that $\mathbf{w} \cdot \mathbf{x} = \hat{y}$; other possible notations are $\boldsymbol{\beta}$ or $\boldsymbol{\theta}$. They are mostly used in statistics.
- X, the matrix of all the observations. It is also called the batch;
- The batch order is how we stack the observations, normally vertically;
- $\mathbf{y}$, the vector of all the responses and $\hat{\mathbf{y}}$, for all the predictions

Selecting a Model

Often, multiple models can fit a data set:
Three polynomials of degree: 1, a straight line, 8, and 9 to fit the *Salammbô* dataset.



A general rule in machine learning is to prefer the simplest hypotheses, here the lower polynomial degrees.
Otherwise, the model can **overfit** the data.
In our case, the optimal model $\mathbf{w}$ has two parameters: (w_0, w_1).

Loss or Objective Function

What are the optimal values of $\mathbf{w}$?
The model should minimize the difference between:

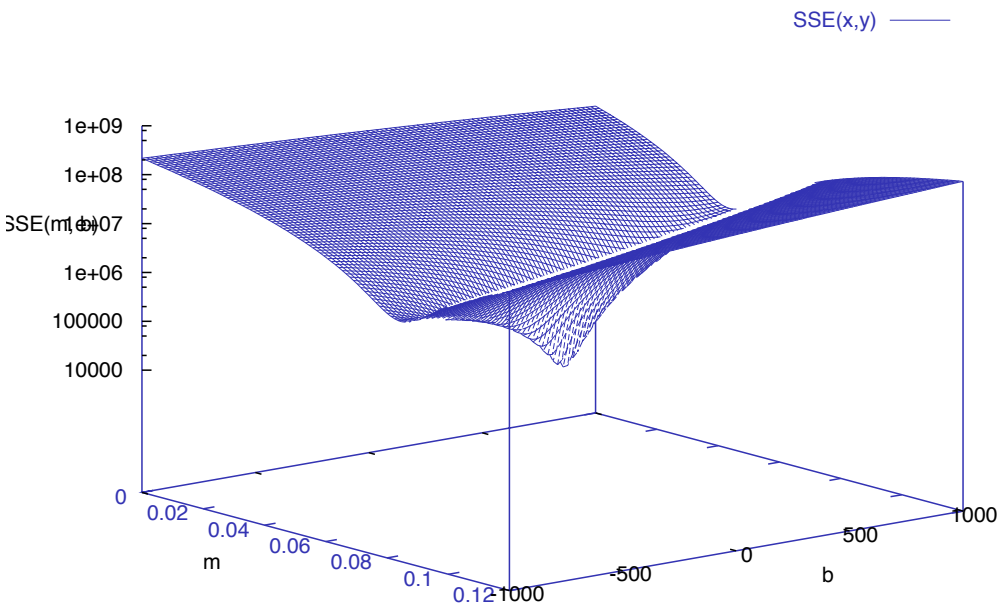
- the predicted values $\hat{\mathbf{y}}$ and
- the observed values $\mathbf{y}$.

This is called the **loss**. Sometimes the objective or the criterion.
For *Salammbô*, the loss is the *mean of the squared errors* (MSE):

$$\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

Visualizing the Loss

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



We will use the notation

$$\hat{y} = w_1 x + w_0 \cdot 1$$

to generalize to any dimension

The Matrices

$$X = \begin{bmatrix} 1 & 36961 \\ 1 & 43621 \\ 1 & 15694 \\ 1 & 36231 \\ 1 & 29945 \\ 1 & 40588 \\ 1 & 75255 \\ 1 & 37709 \\ 1 & 30899 \\ 1 & 25486 \\ 1 & 37497 \\ 1 & 40398 \\ 1 & 74105 \\ 1 & 76725 \\ 1 & 18317 \end{bmatrix}; \mathbf{w} = \begin{bmatrix} 8.7253 \\ 0.0683 \end{bmatrix}; \hat{\mathbf{y}} = \begin{bmatrix} 2533.22 \\ 2988.11 \\ 1080.65 \\ 2483.36 \\ 2054.02 \\ 2780.95 \\ 5148.76 \\ 2584.31 \\ 2119.18 \\ 1749.46 \\ 2569.83 \\ 2767.97 \\ 5070.21 \\ 5249.16 \\ 1259.81 \end{bmatrix}; \mathbf{y} = \begin{bmatrix} 2503 \\ 2992 \\ 1042 \\ 2487 \\ 2014 \\ 2805 \\ 5062 \\ 2643 \\ 2126 \\ 1784 \\ 2641 \\ 2766 \\ 5047 \\ 5312 \\ 1215 \end{bmatrix}; \mathbf{se} = \begin{bmatrix} 913.26 \\ 15.14 \\ 1493.86 \\ 13.25 \\ 1601.31 \\ 578.40 \\ 7527.51 \\ 3444.53 \\ 46.57 \\ 1193.04 \\ 5065.18 \\ 3.8920 \\ 538.909 \\ 3948.29 \\ 2007.53 \end{bmatrix}.$$

$$X\mathbf{w} = \hat{\mathbf{y}}; \text{se}_i = (\hat{y}_i - y_i)^2$$

Code Example: Visualizing the loss

https://github.com/pnugues/pnlp/blob/main/notebooks/07_01_visualize_sse.ipynb

Minimizing the Loss

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

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

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

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:

$$f(\mathbf{w}_k) > f(\mathbf{w}_k + \mathbf{v})$$

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 enable us to define an iteration to reach the minimum:

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

where the gradient gives us the direction of the descent and α_k, the step size (or learning rate).

PyTorch Loop

```
model.train()
for epoch in range(250):
    y_pred = model(X) # We compute Xw = y_hat
    loss = loss_fn(y_pred, y) # (h_hat - y)^2
    optimizer.zero_grad()
    loss.backward() # we compute the gradients
    optimizer.step() # we update the weights
```

Code Example

To find where a loss is minimal, a solver applies a *gradient descent* (GD), or a variant of it, that finds a sequence of model parameters that will reduce the loss.

PyTorch provides a set of optimizers: sgd, rmsprop, adam, nadam, etc.

Description `https://www.ruder.io/optimizing-gradient-descent/`

Experiment: First part of Jupyter Notebook: `https://github.com/pnugues/edan96/blob/main/programs/2-dataset%20and%20regression.ipynb` (up to Stochastic descent)

Updates

To carry out an update, the optimizer uses:

- The whole dataset: batch gradient descent
- One observation: stochastic gradient descent
- A few observations: mini-batch gradient descent

Code Example

Experiment: Second part of Jupyter Notebook:
https://github.com/pnugues/edan96/blob/main/programs/2-dataset%20and%20regression.ipynb
(up to PyTorch's Gradients)

Code Example

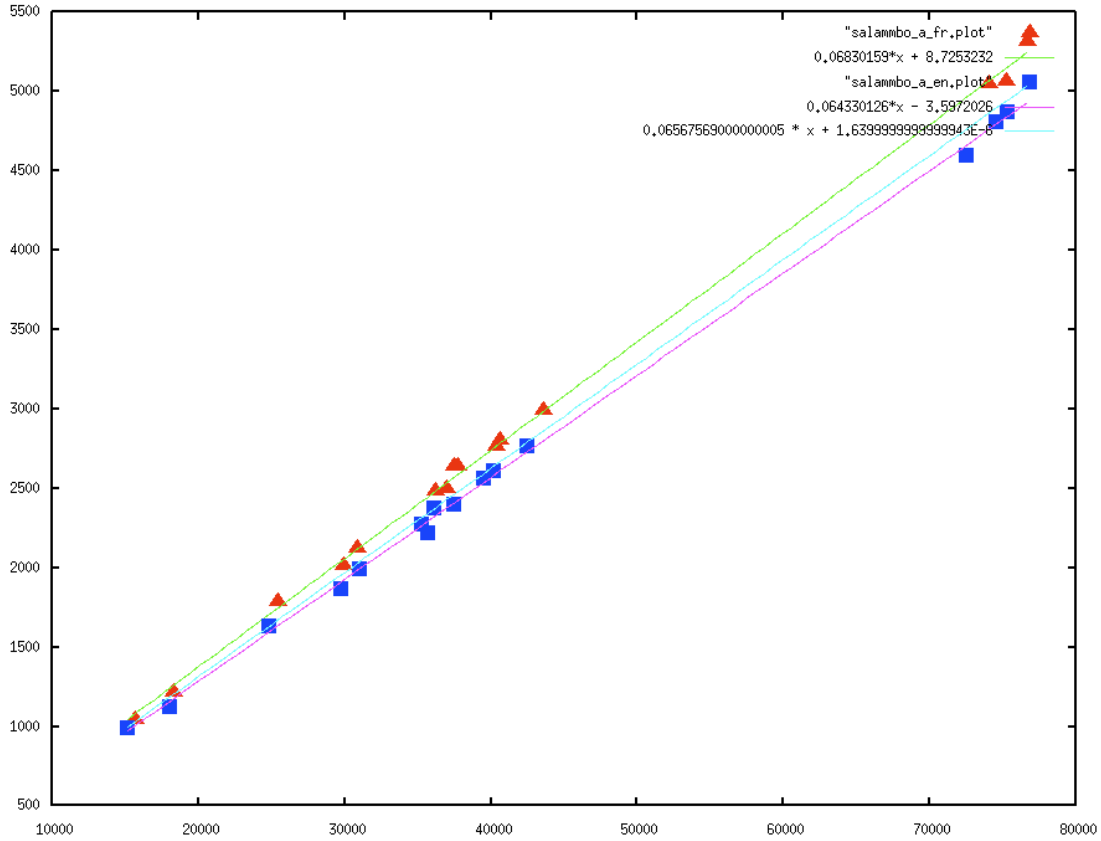
Experiment: Final part of Jupyter Notebook:
https://github.com/pnugues/edan96/blob/main/programs/2-dataset%20and%20regression.ipynb
(from PyTorch's Gradients)

Classification Dataset

Dataset for binary classification: *Salammbô* in French (1) and English (0)

	# char.	# *A*	class (y)	# char.	# *A*	class (y)
Chapter 1	36,961	2,503	1	35,680	2,217	0
Chapter 2	43,621	2,992	1	42,514	2,761	0
Chapter 3	15,694	1,042	1	15,162	990	0
Chapter 4	36,231	2,487	1	35,298	2,274	0
Chapter 5	29,945	2,014	1	29,800	1,865	0
Chapter 6	40,588	2,805	1	40,255	2,606	0
Chapter 7	75,255	5,062	1	74,532	4,805	0
Chapter 8	37,709	2,643	1	37,464	2,396	0
Chapter 9	30,899	2,126	1	31,030	1,993	0
Chapter 10	25,486	1,784	1	24,843	1,627	0
Chapter 11	37,497	2,641	1	36,172	2,375	0
Chapter 12	40,398	2,766	1	39,552	2,560	0
Chapter 13	74,105	5,047	1	72,545	4,597	0
Chapter 14	76,725	5,312	1	75,352	4,871	0
Chapter 15	18,317	1,215	1	18,031	1,119	0

Supervised Learning: Regression and Classification



Given the data set, $\{(\mathbf{x}_i, y_i)|0 < i \leqslant N\}$ and a model f:

- Classification: $f(\mathbf{x}) = y$ is discrete,
- Regression: $f(\mathbf{x}) = y$ is continuous.

Linear 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:

$$\begin{aligned} \hat{y}(\mathbf{x}) &= H(\mathbf{w}\cdot\mathbf{x}) \\ &= H(w_0 + w_1 x_1 + w_2 x_2 + \ldots + w_n x_n), \end{aligned}$$

given a data set of q examples: $DS = \{(1, x_1^j, x_2^j, \ldots, x_n^j, y^j) | j : 1..q\}$.
We use $x_0 = 1$ to simplify the equations.
For a binary classifier, y has then two possible values $\{0, 1\}$ corresponding in our example to {French, English}.

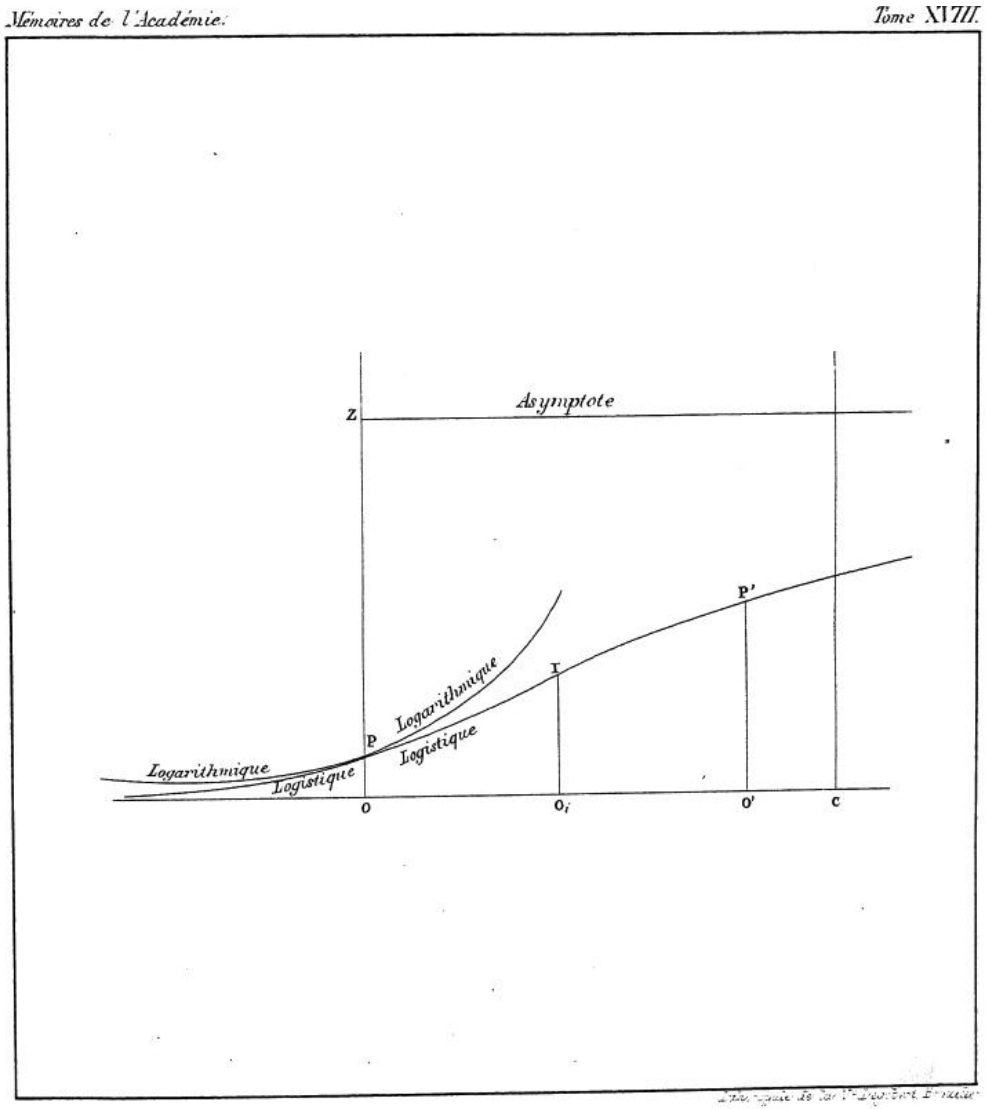
Berkson's Dataset (1944)

Binary classification with probabilities

Drug concentration	Number exposed	Survive Class 0	Die Class 1	Mortality rate	Expected mortality
40	462	352	110	.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 data set. 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).

Classification with Probabilities: The Logistic Curve (Verhulst)

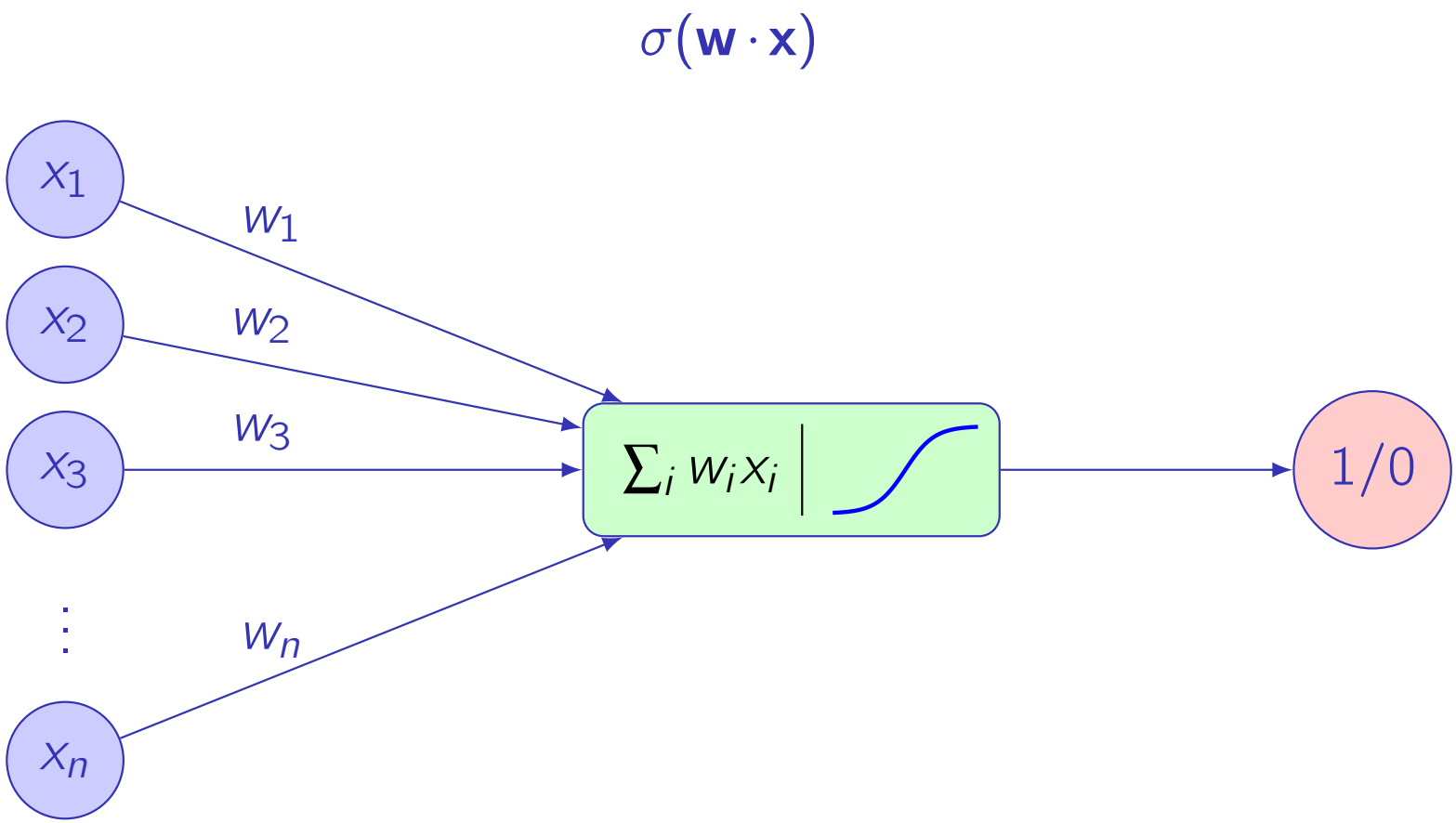


$$Logistic(x) = \frac{1}{1+e^{-x}}$$

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

The logistic curve is also called a sigmoid

Logistic Regression as a Neural Network



Code Example

Experiment: Jupyter Notebook: https://github.com/pnugues/edan96/blob/main/programs/3-Salammbo_torch.ipynb

Loss

The loss function is defined as $L(y, \hat{y})$ with $\hat{y} = h(\mathbf{x})$, where $\mathbf{x}$ is the vector of attributes, h the classifier, and y, the correct value.

Absolute value error	$L_1(y, \hat{y})$	$=$	$\lvert y - \hat{y} \rvert$
Squared error	$L_2(y, \hat{y})$	$=$	$(y - \hat{y})^2$
0/1 loss	$L_{0/1}(y, \hat{y})$	$=$	0 if $y = \hat{y}$ else 1
Binary crossentropy			
Crossentropy			

For PyTorch, see here:
`https://pytorch.org/docs/stable/nn.html#loss-functions` for the available losses

Empirical Loss

We compute the empirical loss of a classifier h on a set of examples E using the formula:

$$Loss(L, E, h) = \frac{1}{N} \sum_{E} L(y, h(x)).$$

For continuous functions:

$$Loss(L, E, h) = \frac{1}{N} \sum_{E} (y - h(x))^2.$$

Absolute value error	L1 or MAE
Squared error	L2 or MSE
0/1 loss	Perceptron
Binary crossentropy	BCE
Crossentropy	CrossEntropy

Cross Entropy for Binary Cases

In practice, we use the mean and the natural logarithm:

$$H(P, M) = -\frac{1}{|X|} \sum_{x \in X} P(x) \log M(x),$$

where P is the truth, and M is the prediction of the model, a probability in the case of logistic regression.
In binary classification:

- $P(x) = 1$
- $M(x)$ is the predicted probability of being class 1.
- If the observation belongs to class 0, its predicted probability is $1 - M(x)$.

Example of Cross Entropy

Computing the cross-entropy of six observations:

Observations	1	2	3	4	5	6
Dose	140	300	140	160	140	250
Observed class (Truth)	0	1	1	1	1	1
Model prediction of being class 1	0.3487	0.9964	0.8557	0.9056	0.8557	0.9882
Model prediction of being class 0	0.6513					
$-P(x)\log M(x)$:	0.4287	0.0036	0.1559	0.0992	0.1559	0.0119

Mean = 0.14252826

Code Example

Experiment: Jupyter Notebook: https://github.com/pnugues/edan96/blob/main/programs/3-Salammbo_torch.ipynb

Multiple Classes: Types of Iris

Iris virginica

Iris setosa

Iris versicolor

Courtesy Wikipedia

Fisher's Iris Dataset (1936)

Table I

Iris setosa				*Iris versicolor*				*Iris virginica*			
Sepal length	Sepal width	Petal length	Petal width	Sepal length	Sepal width	Petal length	Petal width	Sepal length	Sepal width	Petal length	Petal width
5·1	3·5	1·4	0·2	7·0	3·2	4·7	1·4	6·3	3·3	6·0	2·5
4·9	3·0	1·4	0·2	6·4	3·2	4·5	1·5	5·8	2·7	5·1	1·9
4·7	3·2	1·3	0·2	6·9	3·1	4·9	1·5	7·1	3·0	5·9	2·1
4·6	3·1	1·5	0·2	5·5	2·3	4·0	1·3	6·3	2·9	5·6	1·8
5·0	3·6	1·4	0·2	6·5	2·8	4·6	1·5	6·5	3·0	5·8	2·2
5·4	3·9	1·7	0·4	5·7	2·8	4·5	1·3	7·6	3·0	6·6	2·1
4·6	3·4	1·4	0·3	6·3	3·3	4·7	1·6	4·9	2·5	4·5	1·7
5·0	3·4	1·5	0·2	4·9	2·4	3·3	1·0	7·3	2·9	6·3	1·8
4·4	2·9	1·4	0·2	6·6	2·9	4·6	1·3	6·7	2·5	5·8	1·8
4·9	3·1	1·5	0·1	5·2	2·7	3·9	1·4	7·2	3·6	6·1	2·5
5·4	3·7	1·5	0·2	5·0	2·0	3·5	1·0	6·5	3·2	5·1	2·0
4·8	3·4	1·6	0·2	5·9	3·0	4·2	1·5	6·4	2·7	5·3	1·9
4·8	3·0	1·4	0·1	6·0	2·2	4·0	1·0	6·8	3·0	5·5	2·1
4·3	3·0	1·1	0·1	6·1	2·9	4·7	1·4	5·7	2·5	5·0	2·0
5·8	4·0	1·2	0·2	5·6	2·9	3·6	1·3	5·8	2·8	5·1	2·4
5·7	4·4	1·5	0·4	6·7	3·1	4·4	1·4	6·4	3·2	5·3	2·3
5·4	3·9	1·3	0·4	5·6	3·0	4·5	1·5	6·5	3·0	5·5	1·8
5·1	3·5	1·4	0·3	5·8	2·7	4·1	1·0	7·7	3·8	6·7	2·2
5·7	3·8	1·7	0·3	6·2	2·2	4·5	1·5	7·7	2·6	6·9	2·3
5·1	3·8	1·5	0·3	5·6	2·5	3·9	1·1	6·0	2·2	5·0	1·5
5·4	3·4	1·7	0·2	5·9	3·2	4·8	1·8	6·9	3·2	5·7	2·3
5·1	3·7	1·5	0·4	6·1	2·8	4·0	1·3	5·6	2·8	4·9	2·0

Multiple Categories: Softmax

We can generalize logistic regression to multiple categories with the *softmax* function
softmax takes a vector $\mathbf{z} = (z_1, z_2, ..., z_C)$ as input and returns a normalized vector whose ith dimension is:

$$\sigma(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{C} e^{z_j}}.$$

Note: In physics, Boltzmann's original *softmax* is defined as:

$$\sigma(\mathbf{z})_i = \frac{e^{-z_i}}{\sum_{j=1}^{C} e^{-z_j}}.$$

See here:
`https://en.wikipedia.org/wiki/Boltzmann_distribution`

Multiple Categories: Last activation

We use softmax and trainable parameters as last activation of the network.
We can then estimate the probability of an observation represented by $\mathbf{x}$ to belong to class i:

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

Again, we use stochastic gradient descent to compute the weights: $\mathbf{w}$.

Crossentropy Loss Example

Intuitive presentation of the crossentropy loss with the irises



Model				**Eval**
Annotator	Iris virginica	Iris setosa	Iris versicolor	Perfect
Ann. code	(1, 0, 0)	(0, 1, 0)	(0, 0, 1)	Perfect
Model 1	(0.4, 0.3, 0.3)	(0.3, 0.3, 0.4)	(0.2., 0.1, 0.7)	?
Model 2	(0.8, 0.1, 0.1)	(0.1, 0.8, 0.1)	(0.2., 0.1, 0.7)	?
Model 3	(0.8, 0.1, 0.1)	(0.1, 0.8, 0.1)	(0.2, 0.5, 0.3)	?
Model 4	(0.8, 0.1, 0.1)	(0.3, 0.4, 0.3)	(0.3, 0.3, 0.4)	?

1. How to predict the flower type?
2. How to rank the models?

Flower Type



Model				**Eval**
Annotator	Iris virginica	Iris setosa	Iris versicolor	Perfect
Ann. code	(1, 0, 0)	(0, 1, 0)	(0, 0, 1)	Perfect
Model 1	(0.4, 0.3, 0.3) **Virginica**	(0.3, 0.3, 0.4) Versicolor	(0.2., 0.1, 0.7) **Versicolor**	?
Model 2	(0.8, 0.1, 0.1) **Virginica**	(0.1, 0.8, 0.1) **Setosa**	(0.2., 0.1, 0.7) **Versicolor**	?
Model 3	(0.8, 0.1, 0.1) **Virginica**	(0.1, 0.8, 0.1) **Setosa**	(0.2, 0.5, 0.3) Setosa	?
Model 4	(0.8, 0.1, 0.1) **Virginica**	(0.3, 0.4, 0.3) **Setosa**	(0.3, 0.3, 0.4) **Versicolor**	?

We use $\arg\max_i(x_1, x_2, .., x_i, ..., x_n)$ to determine the predicted class

Model Ranking



Model				**Eval**
Annotator	Iris virginica	Iris setosa	Iris versicolor	Perfect
Ann. code	(1, 0, 0)	(0, 1, 0)	(0, 0, 1)	Perfect
Model 1	(0.4, 0.3, 0.3) **Virginica**	(0.3, 0.3, 0.4) Versicolor	(0.2., 0.1, 0.7) **Versicolor**	$0.4 \times 0.3 \times 0.7 = 0.084$
Model 2	(0.8, 0.1, 0.1) **Virginica**	(0.1, 0.8, 0.1) **Setosa**	(0.2., 0.1, 0.7) **Versicolor**	$0.8 \times 0.8 \times 0.7 = 0.448$
Model 3	(0.8, 0.1, 0.1) **Virginica**	(0.1, 0.8, 0.1) **Setosa**	(0.2, 0.5, 0.3) Setosa	$0.8 \times 0.8 \times 0.3 = 0.192$
Model 4	(0.8, 0.1, 0.1) **Virginica**	(0.3, 0.4, 0.3) **Setosa**	(0.3, 0.3, 0.4) **Versicolor**	$0.8 \times 0.4 \times 0.4 = 0.128$

This corresponds to $\prod_i \mathbf{y}_i \cdot \hat{\mathbf{y}}_i$

In practice, we use the logarithms and a sum

Logistic Loss

1. For one observation, logistic regression yields the predicted probabilities of the classes
2. The logistic loss is defined as the opposite of the logarithm of predicted probability of the true class.
3. For a dataset, it can be reformulated as a cross entropy:

$$-\frac{1}{N}\sum_{i=1}^{N}\mathbf{y}_i \cdot \log \hat{\mathbf{y}}_i,$$

where $\mathbf{y}_i$ is a one-hot vector giving the position of the true value:

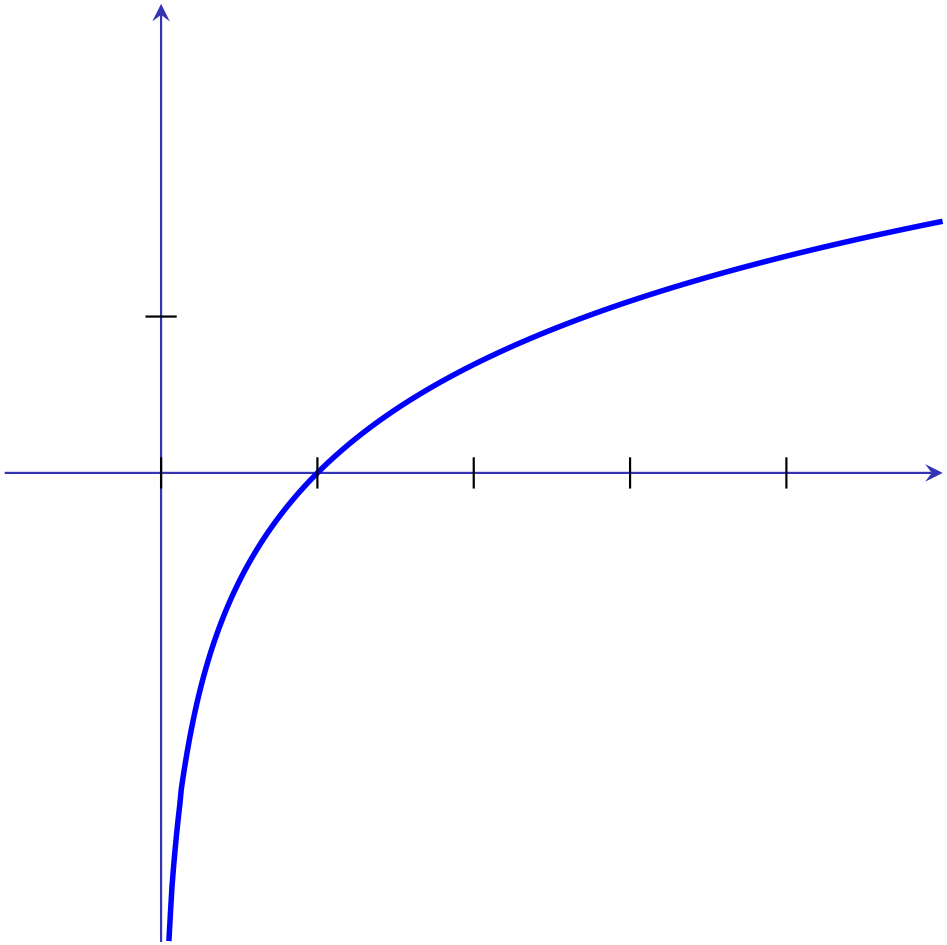
$$\mathbf{y}_i = (0, 0, ..., 0, \mathbf{1}, 0, ...0)$$

and $\hat{\mathbf{y}}_i$ the vector of estimated probabilities of the observations for all the classes

$$\hat{\mathbf{y}}_i = (0.01, 0.005, ..., \mathbf{0.70}, 0.10, ...0.001)$$

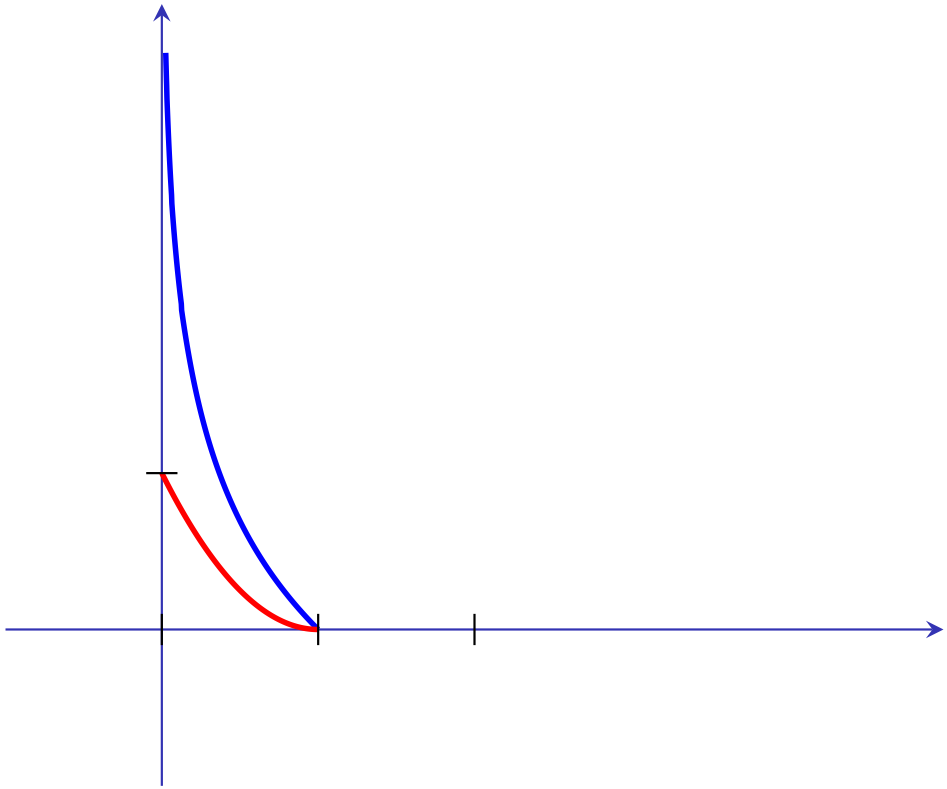
4. For this term: $-1 \times \log 0.7 = -0.36$

Logarithm Function



Loss

The logistic loss is defined as: $L(\hat{y}) = -\log \hat{y}$ in blue compared with the squared error loss in red $(1-\hat{y})^2$.



PyTorch Implementation

Binary and multiclass logistic regressions have a few differences in PyTorch. Notably:

- Representing y:
  In PyTorch, y is a vector of indices (long integers) and $\hat{y}$, a probability distribution

  ```
  y[:5]
  > tensor([2, 1, 0, 2, 0])
  ```

- Crossentropy loss
  `CrossEntropyLoss` includes softmax. We do not have to add it to the model description.
- The model output is just the matrix product with no activation.
- We call this output the logits as the logit function is the inverse of softmax. (weird name though)

Code Example

Experiment: Jupyter Notebook: https://github.com/pnugues/edan96/blob/main/programs/4-Salammbo_multi_torch.ipynb

Loss

The original categories:

```
y[121:126]
 [2 0 0 2 0]
```

The predicted probabilities:

```
model(X[121:126])
 [[9.4238410e-12 2.8314255e-03 9.9716860e-01]
 [9.9939132e-01 6.0863607e-04 2.5036247e-11]
 [9.9859804e-01 1.4019267e-03 3.5701425e-10]
 [1.2004078e-09 2.8088816e-02 9.7191113e-01]
 [9.9938595e-01 6.1400887e-04 2.7445022e-11]]
```

The predicted classes:

```
torch.argmax(model(X[121:126]), dim=-1)
 [2, 0, 0, 2, 0]
```

The loss, probability of the truth.

$$-\frac{1}{N}(\log(9.9716860 \cdot 10^{-1}) + \log(9.9939132 \cdot 10^{-1}) + \log(9.9859804 \cdot 10^{-1}) + ...)$$

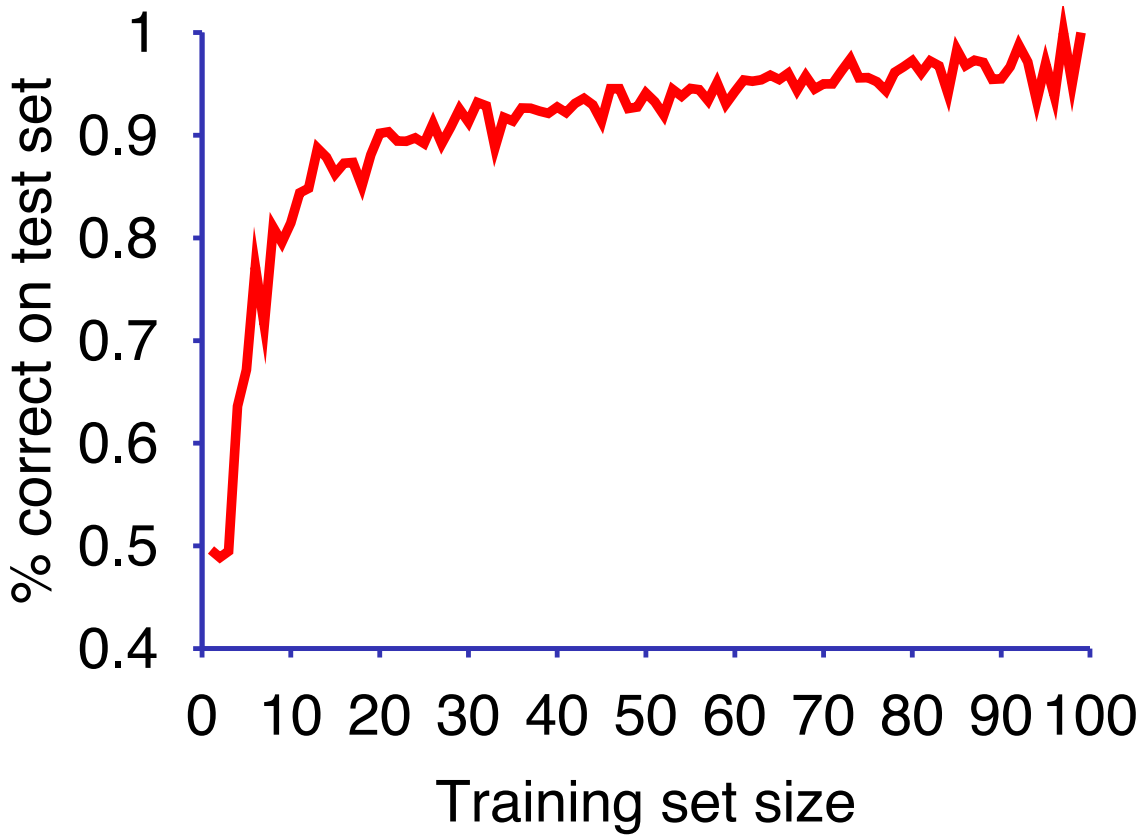
In the example, the prediction is also the truth. This is not always the case.

Code Example

Experiment: Jupyter Notebook: https://github.com/pnugues/edan96/blob/main/programs/4-Salammbo_multi_torch.ipynb

Learning Curve

The classical evaluation technique uses a training set and a test set. Generally, the larger the training set, the better the performance. This can be visualized with a learning curve. From the textbook, Stuart Russell and Peter Norvig, *Artificial Intelligence*, 3rd ed., 2010, page 703.



Overfitting

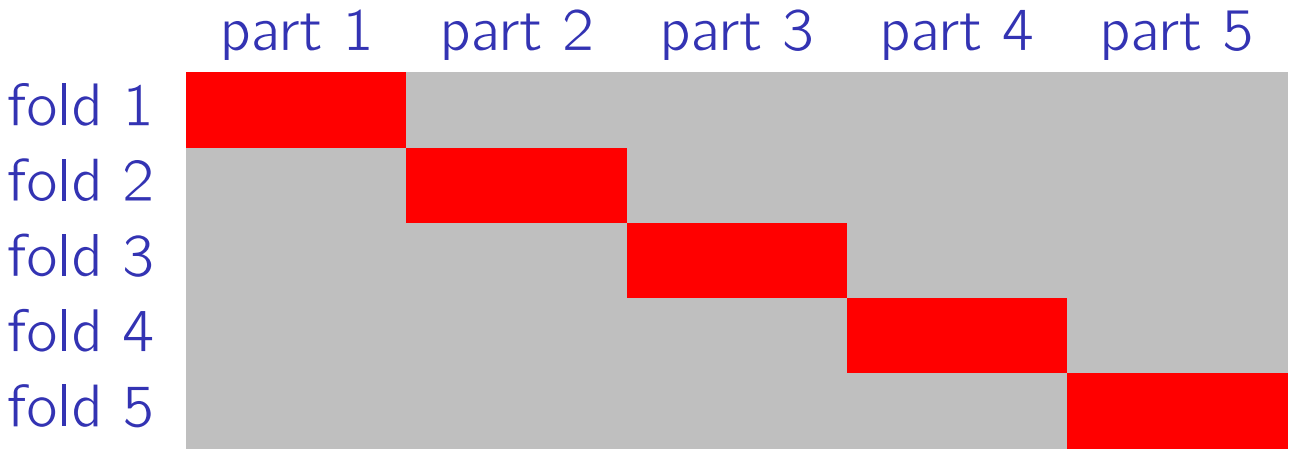
- When two classifiers have equal performances on a specific test set, the simplest one is supposed to be more general
- Complex classifiers may show an overfit to the training data and have poor performance when the data set changes.
- We assess the overfit by drawing the loss and accuracy curves for the training set and a separate validation set.

Evaluation

- The standard evaluation procedure is to train the classifier on a training set and evaluate the performance on a test set.
- When we have only one set, we divide it in two subsets: the training set and the test set (or holdout data).
- The split can be 90–10 or 80–20
- This often optimizes the classifier for a specific test set and creates an overfit

Cross Validation

- A N-fold cross validation mitigates the overfit
- The set is partitioned into N subsets, $N = 5$ for example, one of them being the test set (red) and the rest the training set (gray).
- The process is repeated N times with a different test set: N folds



At the extreme, leave-one-out cross-validation

Model Selection

- Validation can apply to one classification method
- We can use it to select a classification method and its parametrization.
- Needs three sets: training set, validation set (also called development set) , and test set.

Measuring Quality: The Confusion Matrix

A task in natural language processing: Identify the parts of speech (POS) of words.

Example: *The can rusted*

- The human: *The*/art (DT) *can*/noun (NN) *rusted*/verb (VBD)
- The POS tagger: *The*/art (DT) *can*/modal (MD) *rusted*/verb (VBD)

↓Correct	Tagger →									
	DT	IN	JJ	NN	RB	RP	VB	VBD	VBG	VBN
DT	99.4	0.3	–	–	0.3	–	–	–	–	–
IN	0.4	97.5	–	–	1.5	0.5	–	–	–	–
JJ	–	0.1	93.9	1.8	0.9	–	0.1	0.1	0.4	1.5
NN	–	–	2.2	95.5	–	–	0.2	–	0.4	–
RB	0.2	2.4	2.2	0.6	93.2	1.2	–	–	–	–
RP	–	24.7	–	1.1	12.6	61.5	–	–	–	–
VB	–	–	0.3	1.4	–	–	96.0	–	–	0.2
VBD	–	–	0.3	–	–	–	–	94.6	–	4.8
VBG	–	–	2.5	4.4	–	–	–	–	93.0	–
VBN	–	–	4.6	–	–	–	–	4.3	–	90.6

After Franz (1996, p. 124)