Artificial Intelligence

An Advanced Course About Foundations

UNIVERSITÀ DI PAVIA

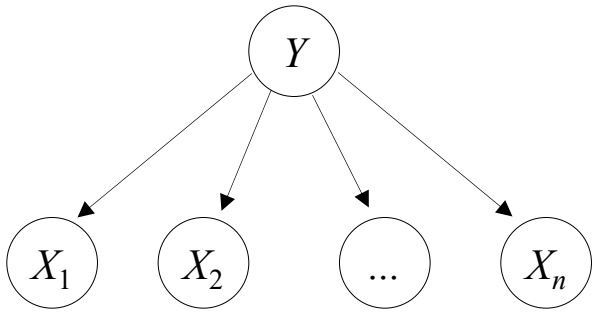
Probabilistic Reasoning: Numerical Supervised Learning

Marco Piastra

Prologue:
Logistic Regression

Graphical Models Redux

- *Naïve Bayesian Classifier*

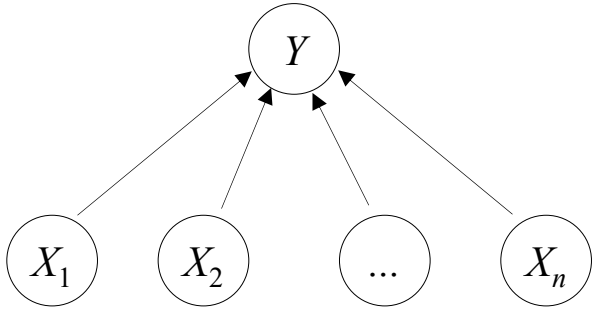


$$P(Y, X_1, \ldots, X_n) = P(Y) \prod_{i=1}^{n} P(X_i|Y)$$

A 'generative' model

Classification $$\frac{P(Y=1)}{P(Y=0)} \prod_{i=1}^{n} \frac{P(X_i|Y=1)}{P(X_i|Y=0)} > \lambda$$

- *Alternative model**



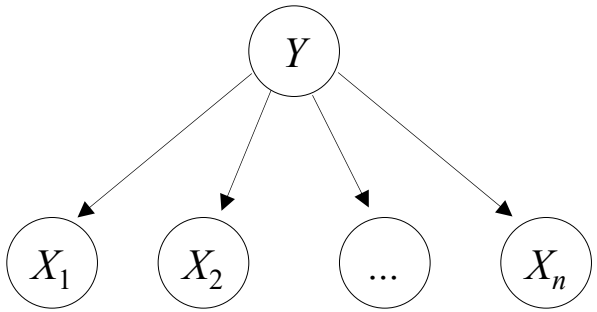
Just reverting the arrows …

$$P(Y, X_1, \ldots, X_n) = P(Y|X_1, \ldots, X_n) \prod_{i=1}^{n} P(X_i)$$

Classification $$\frac{P(Y=1|X_1, \ldots, X_n)}{P(Y=0|X_1, \ldots, X_n)} > \lambda$$

Graphical Models Redux

- *Naïve Bayesian Classifier*

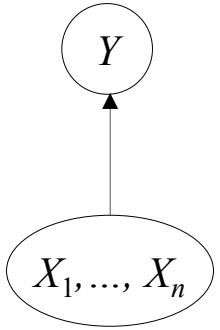


$$P(Y, X_1, \ldots, X_n) = P(Y) \prod_{i=1}^{n} P(X_i|Y)$$

A 'generative' model

Classification $$\frac{P(Y=1)}{P(Y=0)} \prod_{i=1}^{n} \frac{P(X_i|Y=1)}{P(X_i|Y=0)} > \lambda$$

- *Alternative model**



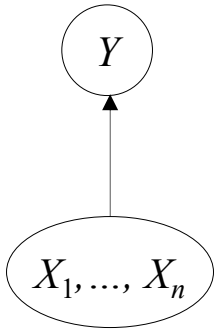
Removing any independence hypotheses …

$$P(Y, X_1, \ldots, X_n) = P(Y|X_1, \ldots, X_n) P(X_1, \ldots, X_n)$$

Classification $$\frac{P(Y=1|X_1, \ldots, X_n)}{P(Y=0|X_1, \ldots, X_n)} > \lambda$$

Graphical Models Redux

- *Alternative model**



$$P(Y, X_1, \ldots, X_n) = P(Y|X_1, \ldots, X_n)P(X_1, \ldots, X_n)$$

Classification: $\dfrac{P(Y = 1\,|X_1, \ldots, X_n)}{P(Y = 0\,|X_1, \ldots, X_n)} > \lambda$

It may sound promising…

No counter-intuitive independence assumptions (*as compared to Naïve Bayesian Classifier*)

It is enough to learn one conditional distribution $P(Y|X_1, \ldots, X_n)$

The MLE is the relative frequency

$$P(Y = y\,|X_1 = x_1, \ldots, X_n = x_n) = \frac{N_{Y=y, X_1=x_1, \ldots, X_n=x_n}}{N_{X_1=x_1, \ldots, X_n=x_n}}$$

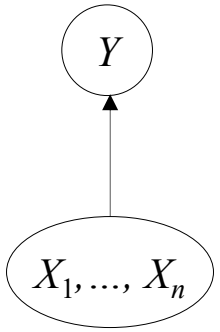
However

2^n probability spaces need to be considered:

hardly any real-world dataset will contain all possible combinations …

Logistic Regression

- *Graphical Model*



$$P(Y, X_1, \ldots, X_n) = P(Y|X_1, \ldots, X_n)P(X_1, \ldots, X_n)$$

Classification $\dfrac{P(Y=1|X_1, \ldots, X_n)}{P(Y=0|X_1, \ldots, X_n)} > \lambda$

For convenience, define:

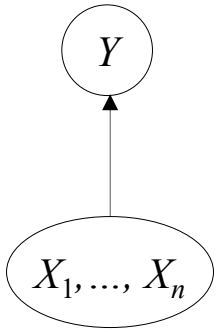
$$p(\boldsymbol{x}) := P(Y=1|X_1=x_1, \ldots, X_n=x_n) \quad \text{where} \quad \boldsymbol{x} := \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} \quad \text{i.e. } \textit{a vector}$$

$$\frac{P(Y=1|X_1=x_1, \ldots, X_n=x_n)}{P(Y=0|X_1=x_1, \ldots, X_n=x_n)} = \frac{p(\boldsymbol{x})}{1-p(\boldsymbol{x})}$$

OK. How can we define $p(\boldsymbol{x})$ *then?*

Logistic Regression

- *Graphical Model*



$$P(Y, X_1, \ldots, X_n) = P(Y|X_1, \ldots, X_n)P(X_1, \ldots, X_n)$$

$$p(\boldsymbol{x}) := P(Y = 1|X_1 = x_1, \ldots, X_n = x_n)$$

Classification $\dfrac{p(\boldsymbol{x})}{1 - p(\boldsymbol{x})} > \lambda$

Logit transform:

$$\log \frac{p(\boldsymbol{x})}{1 - p(\boldsymbol{x})} = f(\boldsymbol{x}) \quad \Rightarrow \quad p(\boldsymbol{x}) = \frac{e^{f(\boldsymbol{x})}}{1 + e^{f(\boldsymbol{x})}} = \frac{1}{1 + e^{-f(\boldsymbol{x})}} = \sigma(f(\boldsymbol{x}))$$

the sigmoid function

Assume $f(\boldsymbol{x})$ *is linear*

$$f(\boldsymbol{x}) := \boldsymbol{w}\boldsymbol{x} + b \quad \Rightarrow \quad p(\boldsymbol{x}) = \frac{1}{1 + e^{-(\boldsymbol{w}\boldsymbol{x}+b)}}$$

a vector of parameters

scalar product of vectors

Logistic Regression
(i.e. a parametric distribution)

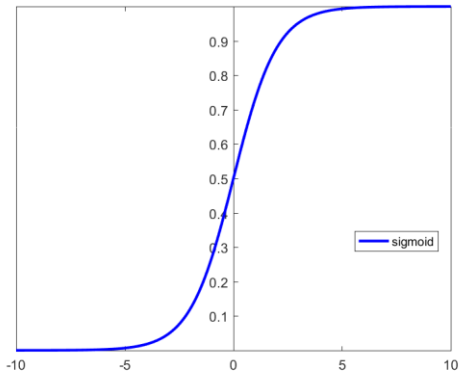
$$\theta := \{\boldsymbol{w}, b\}$$

Logistic Regression

- *Maximum Likelihood Estimation*

sigmoid function

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



Dataset

$$D = \{(\boldsymbol{x}^{(i)}, y^{(i)})\}_{i=1}^{N}$$

Conditional probability

$$P(Y=1|\boldsymbol{x}) = p(\boldsymbol{x}) = \frac{1}{1+e^{-(\boldsymbol{w}\boldsymbol{x}+b)}}$$

A 'discriminative' model:
only a conditional probability is being considered

Conditional Likelihood

$$L(\theta, D) := \prod_{i=1}^{N} p(\boldsymbol{x}^{(i)})^{y^{(i)}} (1-p(\boldsymbol{x}^{(i)}))^{(1-y^{(i)})}$$

Conditional Log-likelihood

$$\begin{aligned} l(\theta, D) := \log l(\theta, D) &= \log \prod_{i=1}^{N} p(\boldsymbol{x}^{(i)})^{y^{(i)}} (1-p(\boldsymbol{x}^{(i)}))^{(1-y^{(i)})} \\ &= \sum_{i=1}^{N} y^{(i)} \log p(\boldsymbol{x}^{(i)}) + (1-y^{(i)}) \log(1-p(\boldsymbol{x}^{(i)})) \end{aligned}$$

Logistic Regression

- *Maximum Likelihood Estimation*

Conditional Log-likelihood

$$
\begin{aligned}
l(\theta, D) &= \sum_{i=1}^{N} y^{(i)} \log p(\boldsymbol{x}^{(i)}) + (1 - y^{(i)}) \log(1 - p(\boldsymbol{x}^{(i)})) \\
&= \sum_{i=1}^{N} \log(1 - p(\boldsymbol{x}^{(i)})) + \log \frac{p(\boldsymbol{x}^{(i)})}{1 - p(\boldsymbol{x}^{(i)})} \\
&= \sum_{i=1}^{N} \log(1 - p(\boldsymbol{x}^{(i)})) + (\boldsymbol{w}\boldsymbol{x}^{(i)} + b) \\
&= \sum_{i=1}^{N} -\log(1 + e^{\boldsymbol{w}\boldsymbol{x}^{(i)} + b}) + (\boldsymbol{w}\boldsymbol{x}^{(i)} + b)
\end{aligned}
$$

recall that, by definition

$$\log \frac{p(\boldsymbol{x})}{1 - p(\boldsymbol{x})} = f(\boldsymbol{x}) := \boldsymbol{w}\boldsymbol{x} + b$$

Maximum Conditional Likelihood Estimator (MCLE in this case)

where:

$$\theta^* := \text{argmax}_\theta \; l(\theta, D) = \text{argmin}_\theta \; nl(\theta, D)$$

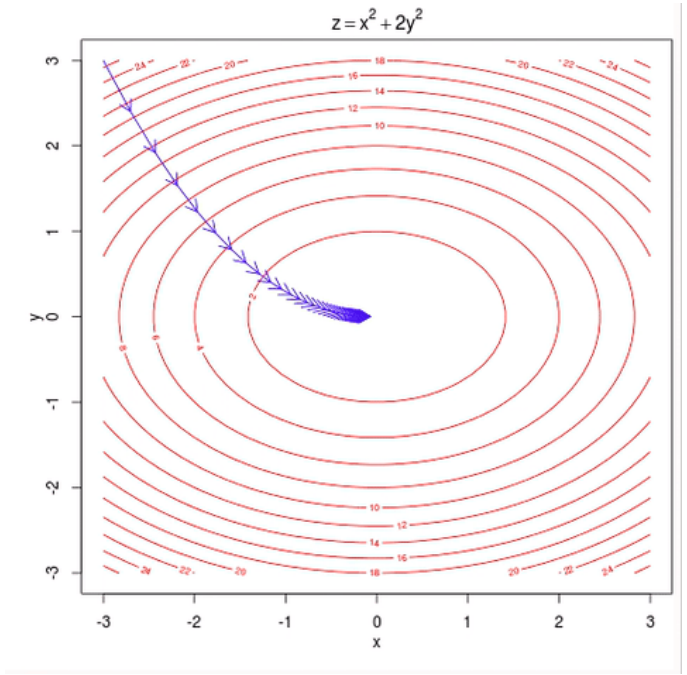
$$nl(\theta, D) := -l(\theta, D)$$

negative log-likelihood

$nl(\theta, D)$ *is convex for* θ *but it cannot be minimized analytically …*

Gradient Descent
(and all that)

Gradient Descent (GD): intuition



- *Objective*

$$\theta^* := \text{argmin}_\theta \; nl(D, \theta)$$

negative log-likelihood: $nl(\theta, D) := -l(\theta, D)$
Makes this a minimization problem

- *Iterative method*

iteration step

1. Initialize $\theta^{(0)}$ at random
2. Update $\theta^{(t)} = \theta^{(t-1)} - \eta \nabla_\theta \; nl(\theta^{(t-1)}, D)$
3. Unless some termination criterion has been met, go back to step 2.

In detail

$$\nabla_\theta \; nl(\theta, D) := \sum_D \nabla_\theta \; nl(\theta, \boldsymbol{x}^{(i)}, y^{(i)})$$

The gradient of the loss over the dataset D is the sum of gradients over each data item

$$\eta \ll 1$$

A *learning rate*, it is arbitrary (i.e. an *hyperparameter*)

Gradient Descent (GD): convergence

- *Convergence*

  When $nl(\theta, D)$ is *convex*, *derivable,* and *Lipschitz continuous*, that is

  $$\|\nabla_\theta\, nl(\theta_1, D) - \nabla_\theta\, nl(\theta_2, D)\| \leq C\, \|\theta_1 - \theta_2\|, \quad C > 0$$

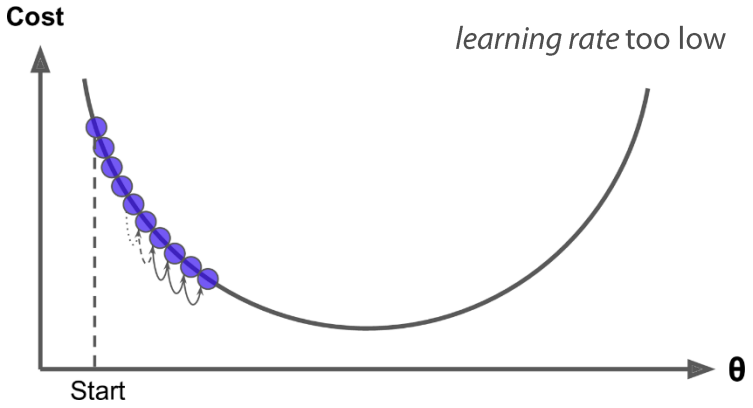
  the gradient descent method converges to the optimal θ^* for $t \to \infty$
  provided that $\eta \leq 1/C$

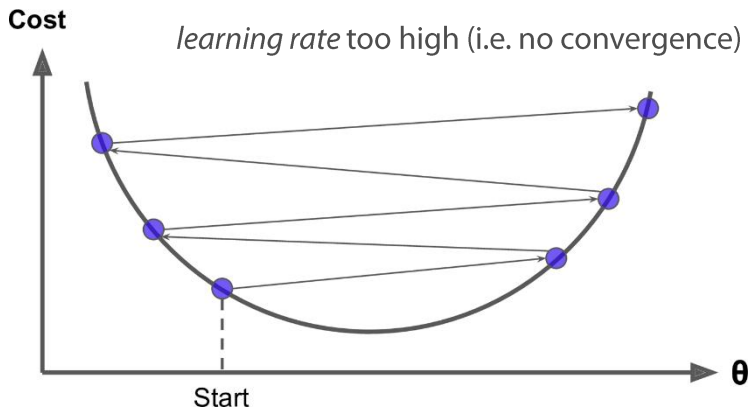
  When $nl(\theta, D)$ is *derivable,* and *Lipschitz continuous* but not *convex*
  the gradient descent method converges to a local minimum of $nl(\theta, D)$
  under the same conditions

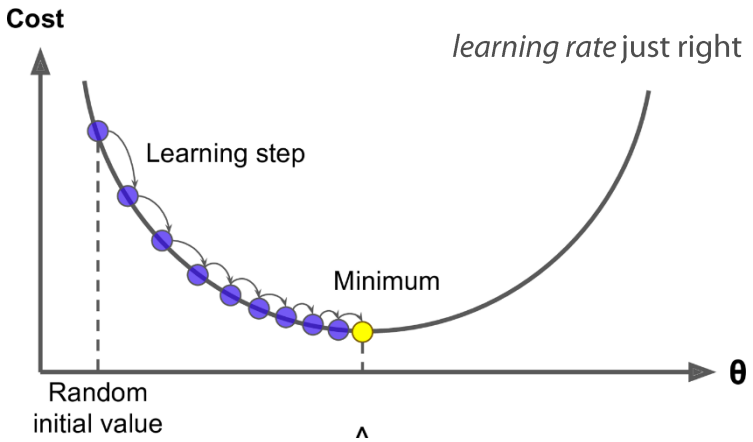
Gradient Descent (GD): practicalities

- *Convergence in practice*

  The choice of the *learning rate* η is crucial





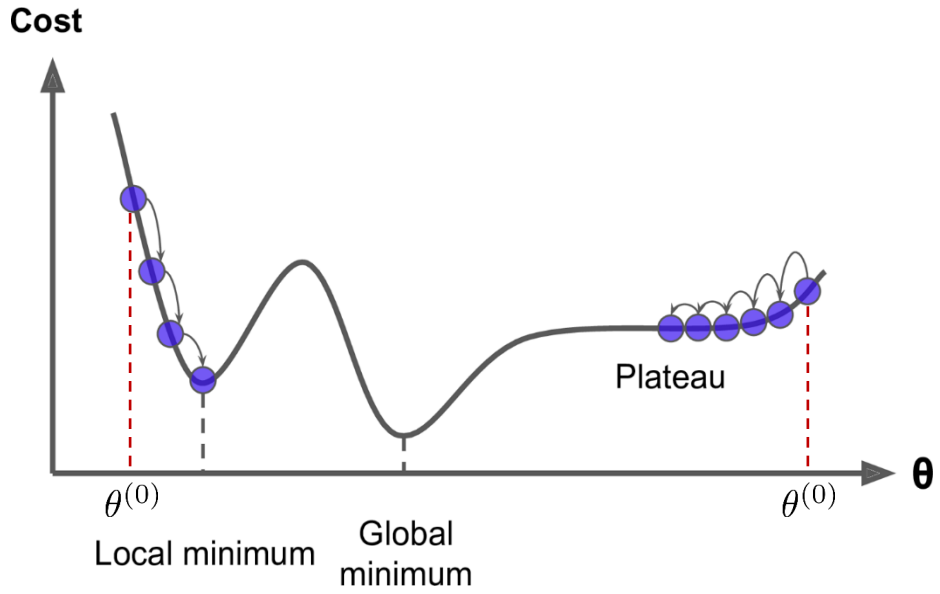


Images from https://www.safaribooksonline.com/library/view/hands-on-machine-learning/9781491962282/ch04.html

Gradient Descent (GD): practicalities

- *Convergence in practice*

  When $nl(D, \theta)$ is <u>not</u> convex, the initial estimate $\theta^{(0)}$ is crucial



The outcome of the method will depend on which $\theta^{(0)}$ is picked

Images from https://www.safaribooksonline.com/library/view/hands-on-machine-learning/9781491962282/ch04.html

Stochastic Gradient Descent (SGD): intuition

- *Objective*

$$\theta^* := \text{argmin}_\theta \; nl(\theta, D)$$

- *Iterative method*

1. Initialize $\theta^{(0)}$ at random
2. Pick a data item $(\boldsymbol{x}^{(i)}, y^{(i)}) \in D$ with *uniform probability*
3. Update $\theta^{(t)} = \theta^{(t-1)} - \eta^{(t)} \nabla_\theta \; nl(\theta^{(t-1)}, \boldsymbol{x}^{(i)}, y^{(i)})$
4. Unless some termination criterion has been met, go back to step 2.

$$\eta^{(t)} \ll 1$$

Note that the *learning rate* may *vary* across iterations…

Stochastic Gradient Descent (SGD): convergence

- *Convergence*

  When $nl(\theta, D)$ is *convex*, *derivable*, and *Lipschitz continuous*, that is

  $$\|\nabla_\theta\, nl(\theta_1, D) - \nabla_\theta\, nl(\theta_2, D)\| \leq C\, \|\theta_1 - \theta_2\|, \quad C > 0$$

  the stochastic gradient descent method converges to the optimal θ^* for $t \to \infty$ provided that

  $$\eta^{(t)} \leq \frac{1}{Ct}$$

  Note that $\eta^{(t)} \to 0$ for $t \to \infty$

  When $nl(D, \theta)$ is *derivable*, and *Lipschitz continuous* but not *convex* the gradient descent method converges to a local minimum of $l(D, \theta)$ under the same conditions

Convergence rate comparison

Assume $nl(\theta, D)$ *convex, derivable,* and *Lipschitz continuous*

Accuracy ρ is attained when

$$| nl(\theta^{(t)}, D) - nl(\theta^*, D) | \leq \rho$$

Define also

$N := |D|$

Size of data space

$d := \dim(\theta)$

Dimension of parameter space

Time := time required to compute each gradient *component*: $\frac{\partial}{\partial \theta_j} nl(\theta, \boldsymbol{x}^{(i)}, y^{(i)})$

Algorithm	***Cost per iteration***	***Iterations to reach accuracy*** ρ	***Time to reach accuracy*** ρ
Gradient descent (GD)	$\mathcal{O}(Nd)$	$\mathcal{O}\left(\log \frac{1}{\rho}\right)$	$\mathcal{O}\left(Nd \log \frac{1}{\rho}\right)$
Stochastic gradient descent (SGD)	$\mathcal{O}(d)$	$\mathcal{O}\left(\frac{1}{\rho}\right)$	$\mathcal{O}\left(d\frac{1}{\rho}\right)$

[*from Bottou & Bousquet, 2007*]

Convergence rate comparison

Assume $nl(\theta, D)$ *convex, derivable,* and *Lipschitz continuous*

Accuracy ρ is attained when

$$| nl(\theta^{(t)}, D) - nl(\theta^*, D) | \leq \rho$$

Define also

$N := |D|$

Size of data space

$d := \dim(\theta)$

Dimension of parameter space

Time := time required to compute each gradient *component*: $\frac{\partial}{\partial \theta_j} nl(\theta, \boldsymbol{x}^{(i)}, y^{(i)})$

SGD can be much faster with large datasets !

Algorithm	***Cost per iteration***	***Iterations to reach accuracy*** ρ	***Time to reach accuracy*** ρ
Gradient descent (GD)	$\mathcal{O}(Nd)$	$\mathcal{O}\left(\log \frac{1}{\rho}\right)$	$\mathcal{O}\left(Nd \log \frac{1}{\rho}\right)$
Stochastic gradient descent (SGD)	$\mathcal{O}(d)$	$\mathcal{O}\left(\frac{1}{\rho}\right)$	$\mathcal{O}\left(d\frac{1}{\rho}\right)$

[*from Bottou & Bousquet, 2007*]

Mini-batch Gradient Descent (MBGD): intuition

- *Objective*

$$\theta^* := \text{argmin}_\theta \; nl(\theta, D)$$

- *Iterative method*

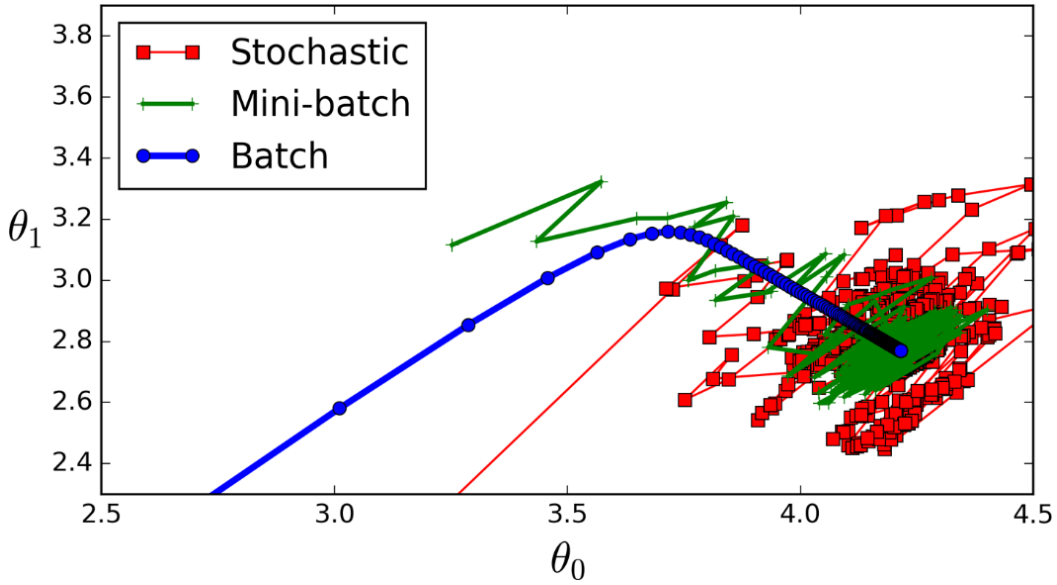
1. Initialize $\theta^{(0)}$ at random
2. Pick a mini batch $B \subseteq D$ with *uniform probability*
3. Update $\theta^{(t)} = \theta^{(t-1)} - \eta^{(t)} \nabla_\theta \; nl(\theta^{(t-1)}, B)$
4. Unless some termination criterion has been met, go back to step 2.

$$\nabla_\theta \; nl(\theta, B) := \sum_B \nabla_\theta \; nl(\theta, \boldsymbol{x}^{(i)}, y^{(i)})$$

This method has the same convergence properties of SGD

Qualitative methods comparison

Typical traces
of the three methods
(batch = GD)



In general:

- GD is more regular but slower (with large datasets)
- SGD is faster (with large datasets) but noisy
- MBGD is often the right compromise in practice…

Image from https://www.safaribooksonline.com/library/view/hands-on-machine-learning/9781491962282/ch04.html

Back to Logistic Regression

Logistic Regression

- *Maximum Conditional Likelihood Estimation*

Conditional Log-likelihood

$$l(\theta, D) = \sum_{i=1}^{N} -\log(1 + e^{\boldsymbol{w}\boldsymbol{x}^{(i)}+b}) + (\boldsymbol{w}\boldsymbol{x}^{(i)} + b)$$

$$l(\theta, \boldsymbol{x}^{(i)}, y^{(i)}) := -\log(1 + e^{\boldsymbol{w}\boldsymbol{x}^{(i)}+b}) + y^{(i)}(\boldsymbol{w}\boldsymbol{x}^{(i)} + b)$$

This is the fundamental computation in all GD-like methods

Parameters can be expressed as:

$$\theta = (\boldsymbol{w}, b)$$

Hence the gradient can be split into two separate components:

$$\nabla_\theta \, l(\theta, \boldsymbol{x}, y) = \left(\frac{\partial}{\partial \boldsymbol{w}} l(\theta, \boldsymbol{x}, y), \;\; \frac{\partial}{\partial b} l(\theta, \boldsymbol{x}, y) \right)$$

Data item indexes dropped, for simplicity

For logistic regression, the negative log-likelihood is convex;
hence gradient descent is guaranteed to reach its global minimum

Logistic Regression

- *Log-likelihood gradients*

$$\frac{\partial}{\partial \boldsymbol{w}} l(\theta, \boldsymbol{x}, y) = \frac{\partial}{\partial \boldsymbol{w}} \left(-\log(1 + e^{\boldsymbol{w}\boldsymbol{x}+b}) + y(\boldsymbol{w}\boldsymbol{x} + b) \right)$$

$$= -\frac{\partial}{\partial \boldsymbol{w}} \log(1 + e^{\boldsymbol{w}\boldsymbol{x}+b}) + y \frac{\partial}{\partial \boldsymbol{w}} (\boldsymbol{w}\boldsymbol{x} + b)$$

$$= -\frac{1}{1 + e^{\boldsymbol{w}\boldsymbol{x}+b}} \frac{\partial}{\partial \boldsymbol{w}} (1 + e^{\boldsymbol{w}\boldsymbol{x}+b}) + y\boldsymbol{x}$$

$$= -\frac{e^{\boldsymbol{w}\boldsymbol{x}+b}}{1 + e^{\boldsymbol{w}\boldsymbol{x}+b}} \frac{\partial}{\partial \boldsymbol{w}} (\boldsymbol{w}\boldsymbol{x} + b) + y\boldsymbol{x}$$

$$= -\frac{e^{\boldsymbol{w}\boldsymbol{x}+b}}{1 + e^{\boldsymbol{w}\boldsymbol{x}+b}} \boldsymbol{x} + y\boldsymbol{x}$$

$$= -\sigma(\boldsymbol{w}\boldsymbol{x} + b)\boldsymbol{x} + y\boldsymbol{x}$$

Logistic Regression

- *Log-likelihood gradients*

$$\begin{aligned}
\frac{\partial}{\partial b} l(\theta, \boldsymbol{x}, y) &= \frac{\partial}{\partial b} \left(-\log(1 + e^{\boldsymbol{w}\boldsymbol{x}+b}) + y(\boldsymbol{w}\boldsymbol{x} + b) \right) \\
&= -\frac{\partial}{\partial b} \log(1 + e^{\boldsymbol{w}\boldsymbol{x}+b}) + y \frac{\partial}{\partial b} (\boldsymbol{w}\boldsymbol{x} + b) \\
&= -\frac{1}{1 + e^{\boldsymbol{w}\boldsymbol{x}+b}} \frac{\partial}{\partial b} (1 + e^{\boldsymbol{w}\boldsymbol{x}+b}) + y \\
&= -\frac{e^{\boldsymbol{w}\boldsymbol{x}+b}}{1 + e^{\boldsymbol{w}\boldsymbol{x}+b}} \frac{\partial}{\partial b} (\boldsymbol{w}\boldsymbol{x} + b) + y \\
&= -\frac{e^{\boldsymbol{w}\boldsymbol{x}+b}}{1 + e^{\boldsymbol{w}\boldsymbol{x}+b}} + y \\
&= -\sigma(\boldsymbol{w}\boldsymbol{x} + b) + y
\end{aligned}$$

A Practical Example:
Logistic Regression is Linear, Anyway

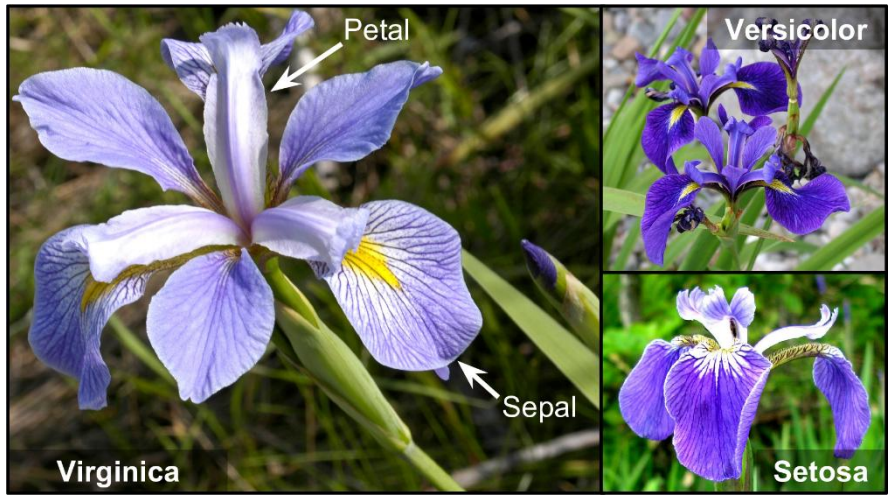
Logistic Regression: qualitative example

- *IRIS dataset*
  https://archive.ics.uci.edu/ml/datasets/iris

  Three classes
  (Iris Setosa, Iris Versicolour, Iris Virginica)

  Numerical data
  (petal length & width, sepal length & width)

  150 data items (50 per each class)



Consider just one class: Iris Virginica
(the other class is the complement)
and petal width as unique input feature

Apply logistic regression (with any GD-like method)

This will be the result:

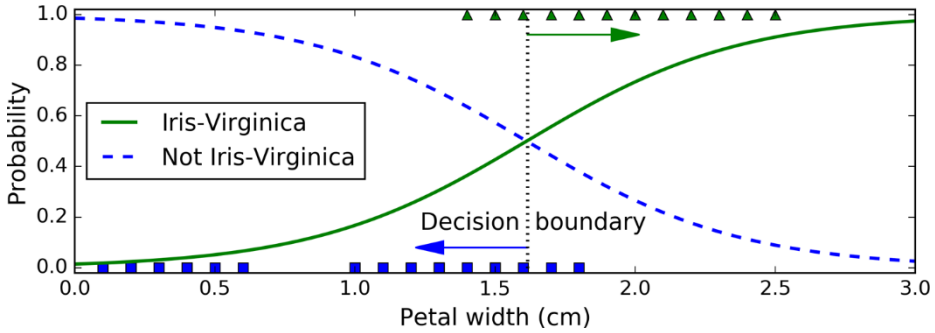


Image from https://www.safaribooksonline.com/library/view/hands-on-machine-learning/9781491962282/ch04.html

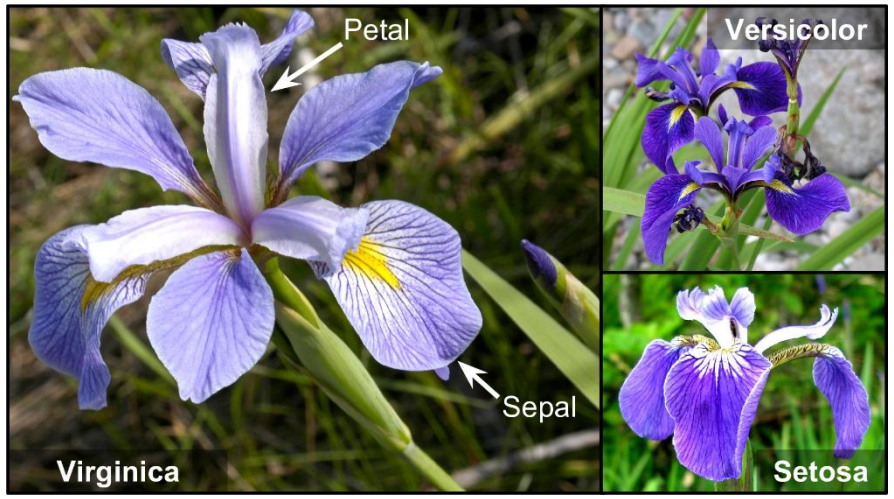
Logistic Regression: qualitative example

- *IRIS dataset*
  https://archive.ics.uci.edu/ml/datasets/iris

  Three classes
  (Iris Setosa, Iris Versicolour, Iris Virginica)

  Numerical data
  (petal length & width, sepal length & width)

  150 data items (50 per each class)



Consider just one class: Iris Virginica
(the other class is the complement)
with petal width and petal length as input features

Apply logistic regression (with any GD-like method)

This will be the result:

The separation improves

The *linearity* of the parametrization is evident:
the two classes must be *linearly separable*

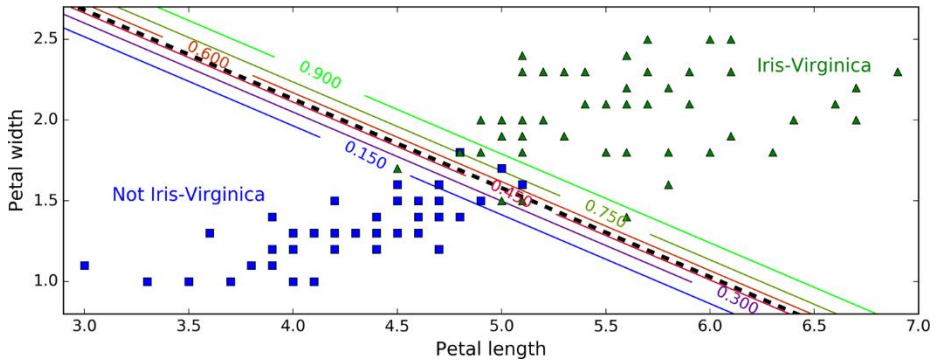


Image from https://www.safaribooksonline.com/library/view/hands-on-machine-learning/9781491962282/ch04.html

Probabilistic Models as *Predictors*

Predictors!

From a known dataset to predicting further possible outcomes

- ***Probabilistic inference*** *(redux)*

  In general, given a probabilistic model, the problem is finding:

$$P(\{X_r\}|\{X_o\}) = \sum_{\{X_i\}} P(\{X_r\}, \{X_i\}|\{X_o\})$$

  where:

  1) $\{X_o\}$ is the set of observations, i.e. *what is known*: *partial knowledge*
  2) $\{X_r\}$ is what *we want to know*
  3) $\{X_i\}$ is any other variable in the model

Fundamental question:

How good is a probabilistic model when applied to data items that are *not* in the dataset?

E.g. suppose we have a data item which is fully observed but not part of the dataset D
will the probabilistic model, given with partial observations, predict the remaining observations?

Training, validation and test

When the training process becomes too specific to the training set

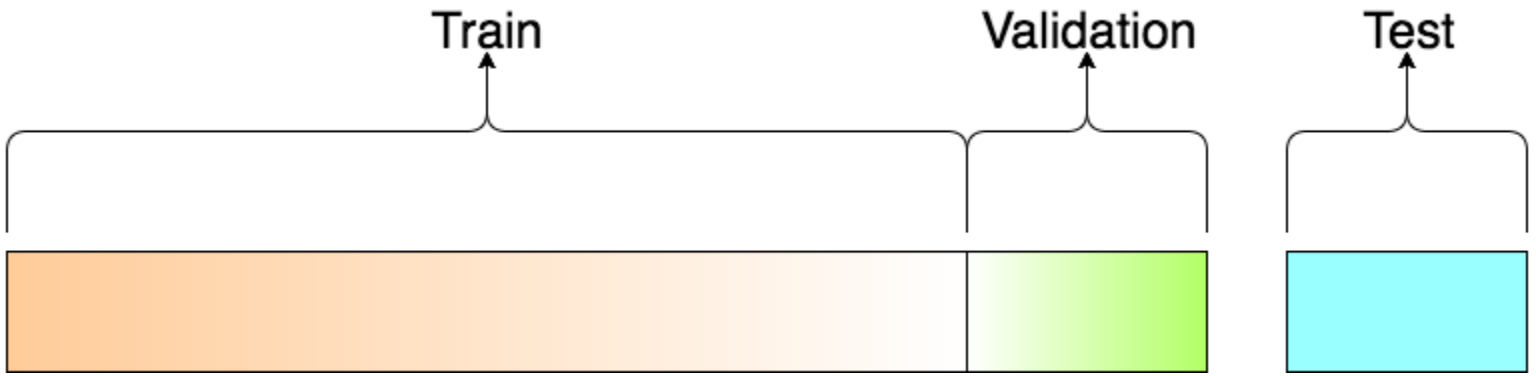
- ***Training set, validation set, test set***

  Splitting the dataset into a partition

$$D = D_{train} \cup D_{val} \cup D_{test}$$

$$\{(\boldsymbol{x}^{(i)}, y^{(i)})\}_{i=1}^{N} = \{(\boldsymbol{x}^{(j)}, y^{(j)})\}_{j=1}^{N_{train}} \cup \{(\boldsymbol{x}^{(k)}, y^{(k)})\}_{k=1}^{N_{val}} \cup \{(\boldsymbol{x}^{(l)}, y^{(l)})\}_{l=1}^{N_{test}}$$

$$N_{train} \gg N_{val}, N_{test}$$



Overfitting

When the training process becomes too specific to the training set

- ***Training set, validation set, test set***

  Splitting the dataset

  $$D = D_{train} \cup D_{val} \cup D_{test}$$

  $$\{(\boldsymbol{x}^{(i)}, y^{(i)})\}_{i=1}^{N} = \{(\boldsymbol{x}^{(j)}, y^{(j)})\}_{j=1}^{N_{train}} \cup \{(\boldsymbol{x}^{(k)}, y^{(k)})\}_{k=1}^{N_{val}} \cup \{(\boldsymbol{x}^{(l)}, y^{(l)})\}_{l=1}^{N_{test}}$$

  $$N_{train} \gg N_{val}, N_{test}$$

  Training is made on D_{train} only

  At each *epoch* — *when the whole* D_{train} *has been processed*

  the loss function is evaluated on D_{val}

  After some epochs, the performance on D_{val} *might get worse*

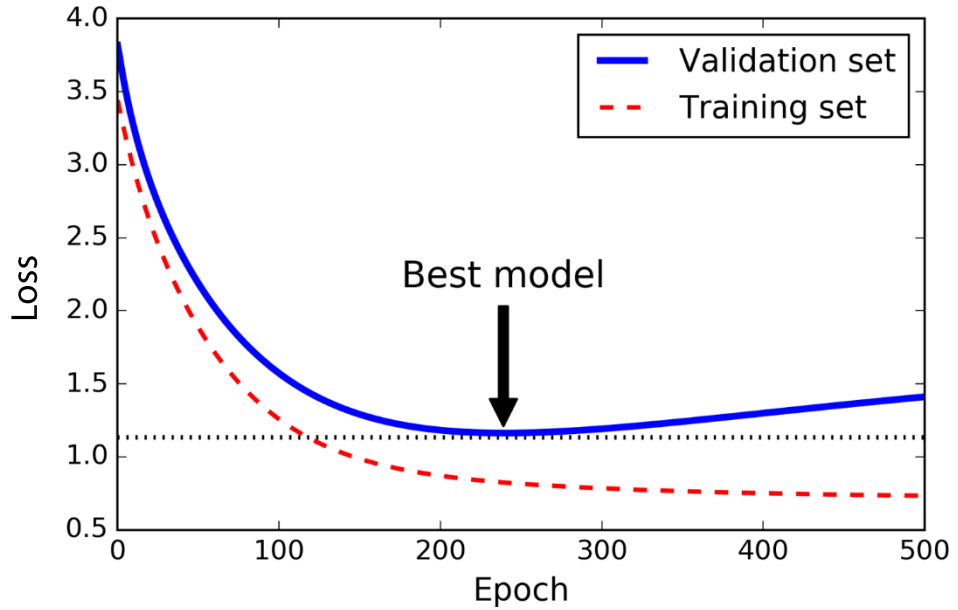


Image from https://www.safaribooksonline.com/library/view/hands-on-machine-learning/9781491962282/ch04.html

k-Fold Cross-Validation

- ***One dataset, multiple splits***
  1) Divide the dataset into k splits (i.e. *folds*)
  2) Use k - 1 folds for training and one fold for testing
  3) Unless all combinations have been considered, change combination and go back to 2)

The test result is the *average test loss* across all possible combinations

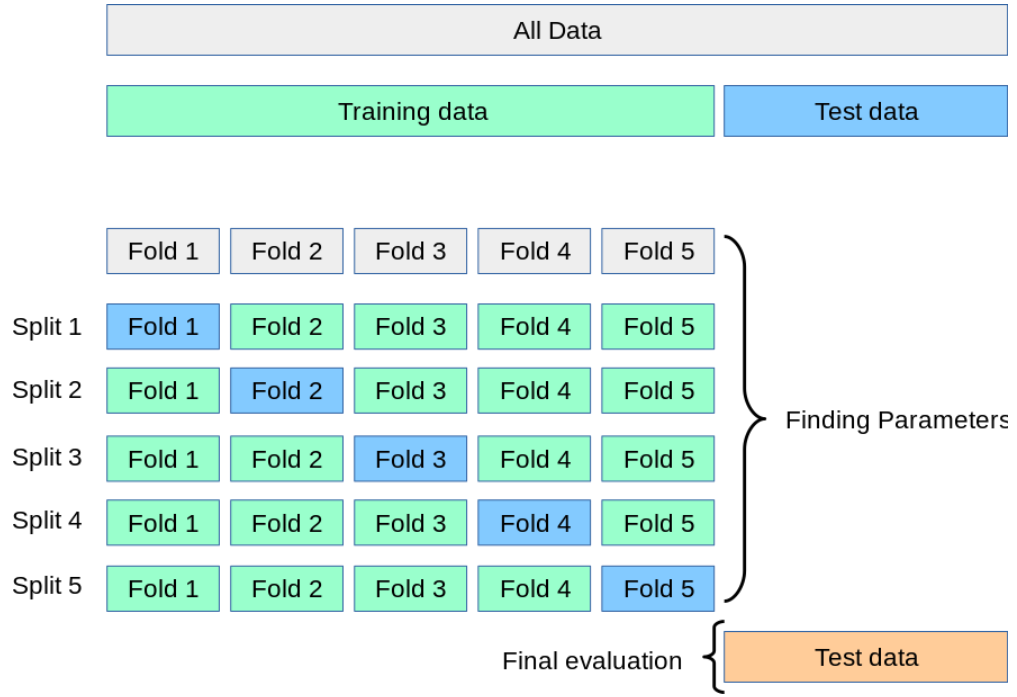


Image from https://www.kdnuggets.com/2020/01/data-validation-machine-learning.html