

Artificial Intelligence

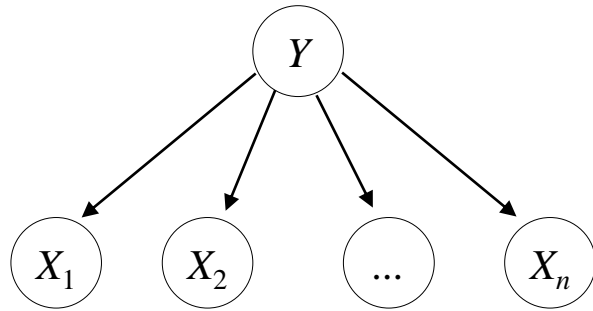
Probabilistic reasoning:
supervised learning
and numerical optimization

Marco Piastra

Prologue: Logistic Regression

Graphical Models Redux

■ Naïve Bayesian Classifier

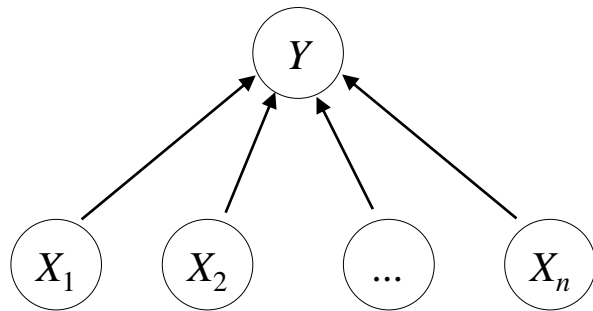


$$P(Y, X_1, \dots, X_n) = P(Y) \prod_{i=1}^n P(X_i|Y)$$

A 'generative' model

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

■ Alternative model*



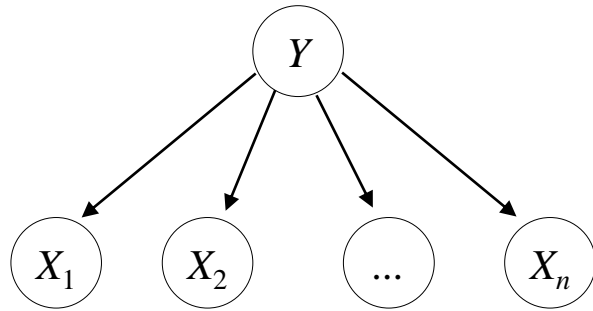
Just reverting the arrows ...

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

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

Graphical Models Redux

■ Naïve Bayesian Classifier

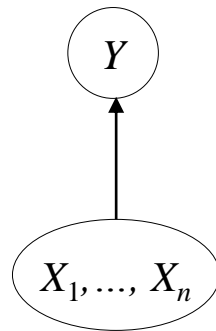


$$P(Y, X_1, \dots, X_n) = P(Y) \prod_{i=1}^n P(X_i|Y)$$

A 'generative' model

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

■ Alternative model*



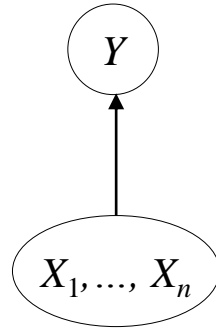
$$P(Y, X_1, \dots, X_n) = P(Y|X_1, \dots, X_n)P(X_1, \dots, X_n)$$

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

Removing any independence hypotheses ...

Graphical Models Redux

- Alternative model*



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

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

It may sound promising...

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

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

The MLE is the relative frequency

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

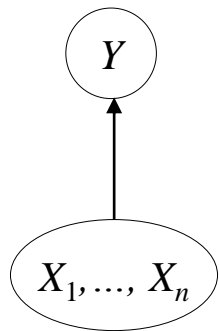
However...

2^n probabilities will have to be learnt

Hardly any real-world dataset will contain all possible combinations ...

Logistic Regression

■ Graphical Model



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

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

For convenience, define:

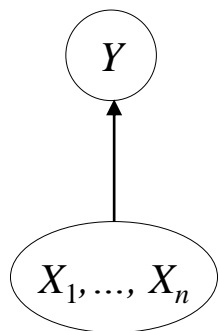
$$p(\mathbf{x}) := P(Y = 1|X_1 = x_1, \dots, X_n = x_n) \quad \text{where} \quad \mathbf{x} := \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} \quad \text{i.e. a vector}$$

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

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

Logistic Regression

■ Graphical Model



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

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

$$\text{Classification } \frac{p(\mathbf{x})}{1 - p(\mathbf{x})} > \lambda$$

Logit transform:

$$\log \frac{p(\mathbf{x})}{1 - p(\mathbf{x})} = f(\mathbf{x}) \quad \Rightarrow \quad p(\mathbf{x}) = \frac{e^{f(\mathbf{x})}}{1 + e^{f(\mathbf{x})}} = \frac{1}{1 + e^{-f(\mathbf{x})}} = \underbrace{\sigma(f(\mathbf{x}))}_{\text{the sigmoid function}}$$

Assume $f(\mathbf{x})$ linear

$$f(\mathbf{x}) := \underbrace{\mathbf{w}\mathbf{x}}_{\text{scalar product of vectors}} + b \quad \Rightarrow \quad p(\mathbf{x}) = \frac{1}{1 + e^{-(\mathbf{w}\mathbf{x} + b)}}$$

Logistic Regression
(i.e. a parametric distribution)

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

Logistic Regression

■ Maximum Likelihood Estimation

Dataset

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

Conditional probability

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

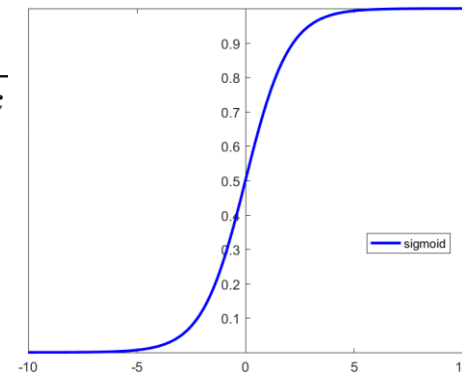
Likelihood

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

Log-likelihood

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

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



A 'discriminative' model

This is a product of conditional probabilities (IID data)

Logistic Regression

■ Maximum Likelihood Estimation

Log-likelihood

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

MLE (a.k.a. Maximum Conditional Likelihood Estimator MCLE in this case)

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

where

negative log-likelihood — $nl(D, \theta) := -l(D, \theta)$

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

Gradient Descent *(and all that)*

Gradient Descent (GD): intuition

■ Objective

Turn this into a minimization problem

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

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

■ Iterative method

Step in the method

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

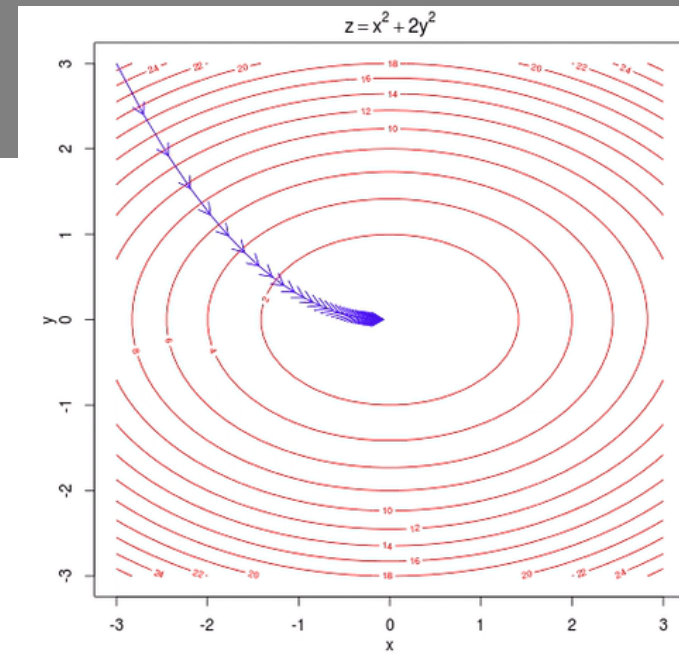
In detail

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

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

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

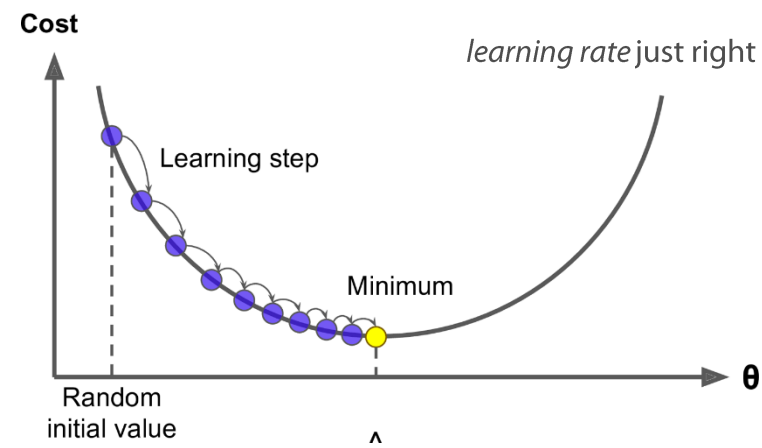
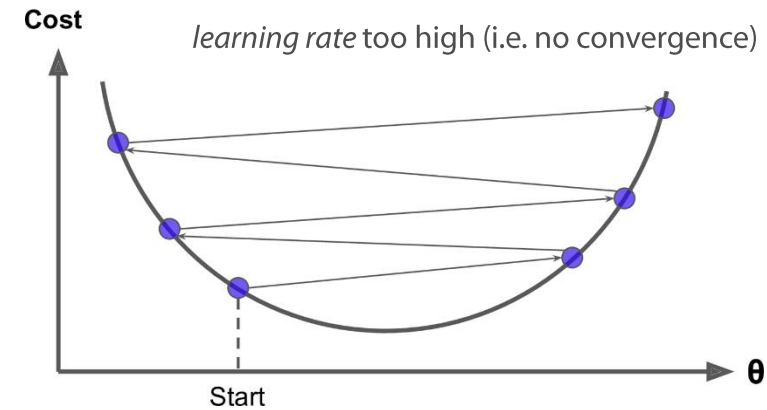
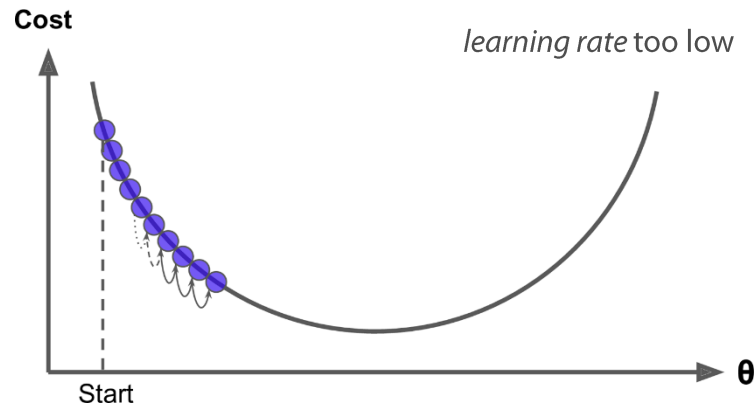
the gradient descent method converges to the optimal θ^* for $t \rightarrow \infty$
provided that $\eta \leq 1/C$

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

Gradient Descent (GD): practicalities

- *Convergence in practice*

The choice of the *learning rate* η is crucial

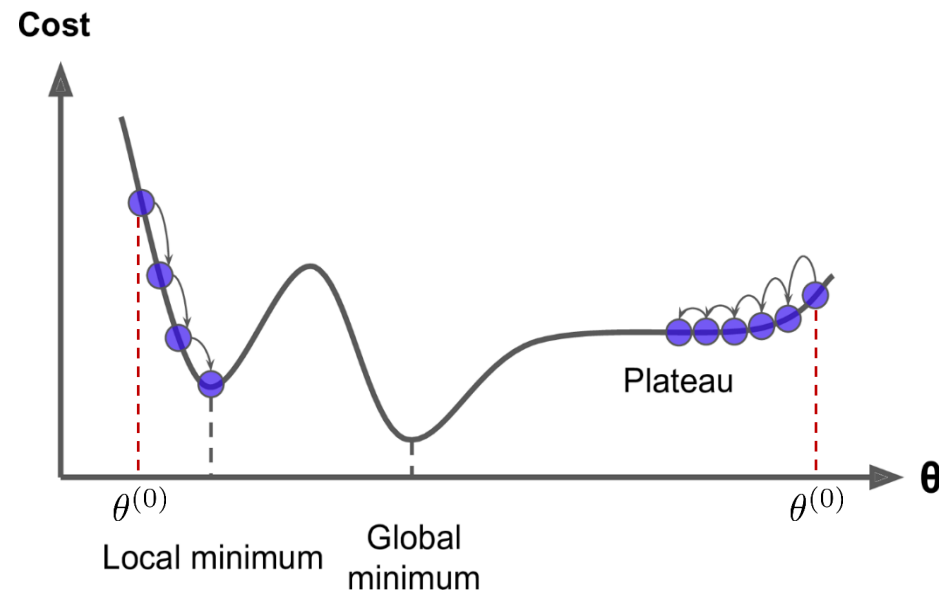


Images from <https://www.safaribooksonline.com/library/view/hands-on-machine-learning/9781491962282/ch04.html>

Gradient Descent (GD): practicalities

- *Convergence in practice*

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



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

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

Stochastic Gradient Descent (SGD): intuition

- *Objective*

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

- *Iterative method*

1. Initialize $\theta^{(0)}$ at random
2. Pick a data item $(\mathbf{x}^{(i)}, y^{(i)}) \in D$ with uniform probability
3. Update $\theta^{(t)} = \theta^{(t-1)} - \eta^{(t)} \nabla_{\theta} nl(\mathbf{x}^{(i)}, y^{(i)}, \theta^{(t-1)})$
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 $nl(D, \theta)$ is *convex, derivable, and Lipschitz continuous*, that is

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

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

$$\eta^{(t)} \leq \frac{1}{Ct} \quad \text{Note that } \eta^{(t)} \rightarrow 0 \text{ for } t \rightarrow \infty$$

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

Accuracy ρ is attained when

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

Define also

$$N := |D|$$

Size of data space

$$d := \dim(\theta)$$

Dimension of parameter space

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

Algorithm	Cost per iteration	Iterations to reach accuracy ρ	Time to reach accuracy ρ
<i>Gradient descent</i> (GD)	$\mathcal{O}(Nd)$	$\mathcal{O}\left(\log \frac{1}{\rho}\right)$	$\mathcal{O}\left(Nd \log \frac{1}{\rho}\right)$
<i>Stochastic gradient descent</i> (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 $nl(D, \theta)$ convex, derivable, and Lipschitz continuous

Accuracy ρ is attained when

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

Define also

$$N := |D|$$

Size of data space

$$d := \dim(\theta)$$

Dimension of parameter space

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

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*

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

- *Iterative method*

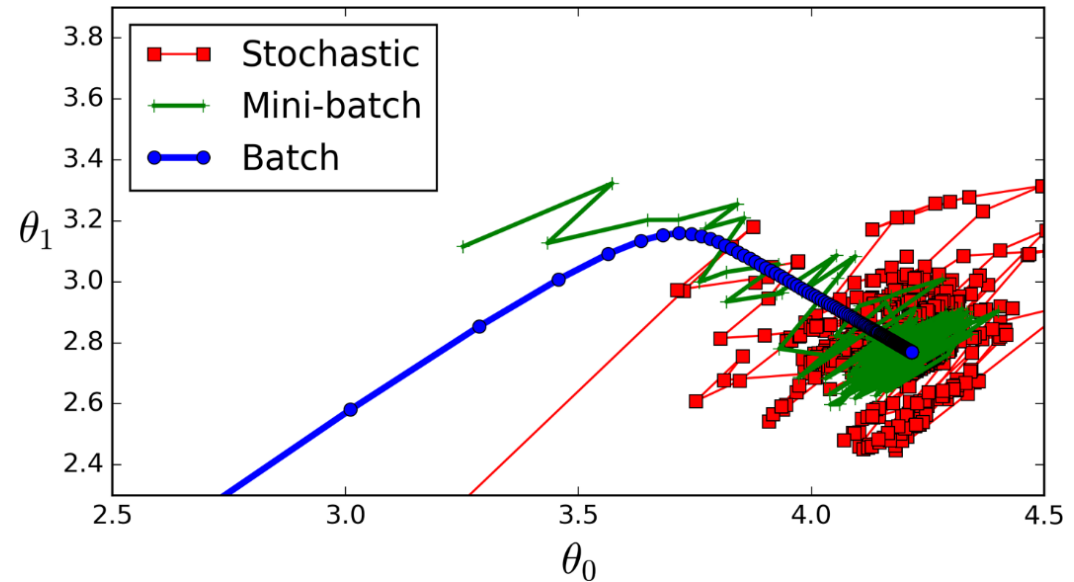
1. Initialize $\theta^{(0)}$ at random
2. Pick a mini batch $B \subseteq D$ with uniform probability
3. Update $\theta^{(t)} = \theta^{(t-1)} - \eta^{(t)} \nabla_{\theta} nl(B, \theta^{(t-1)})$
4. Unless some termination criterion has been met, go back to step 2.

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

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 Likelihood Estimation

Log-likelihood

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

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

This is the fundamental computation in all GD-like methods

Parameters can be expressed as:

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

Hence the gradient can be split into two separate components:

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

Data item indexes dropped, for simplicity

Logistic Regression

- Log-likelihood gradients

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

Logistic Regression

- Log-likelihood gradients

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

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:

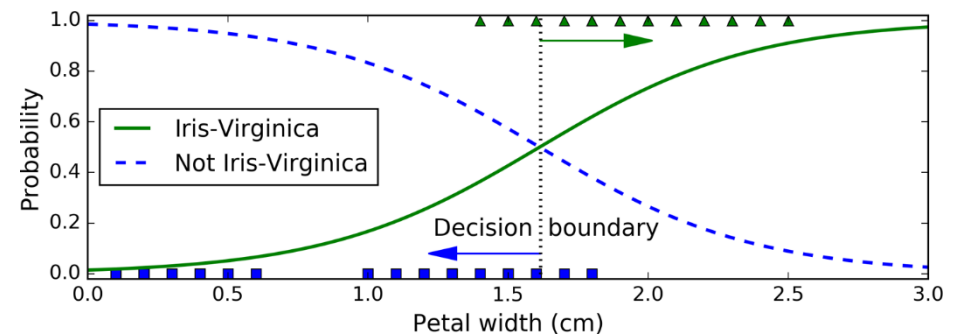


Image from <https://www.safaribooksonline.com/library/view/hands-on-machine-learning/9781491962282/ch04.html>

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*

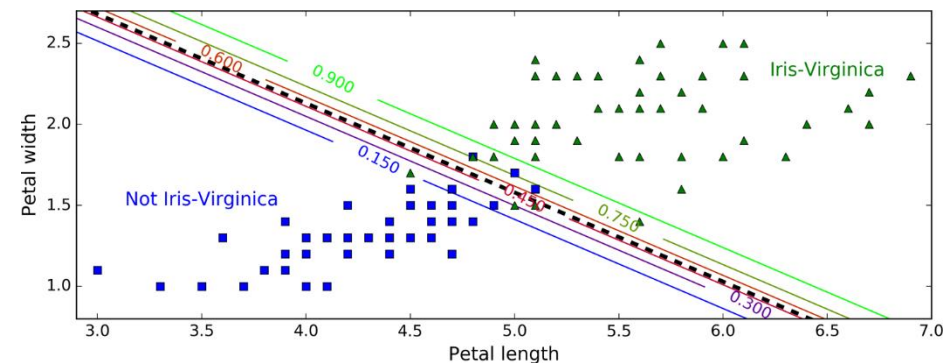


Image from <https://www.safaribooksonline.com/library/view/hands-on-machine-learning/9781491962282/ch04.html>

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?*

Overfitting

When the training process becomes too specific to the training set

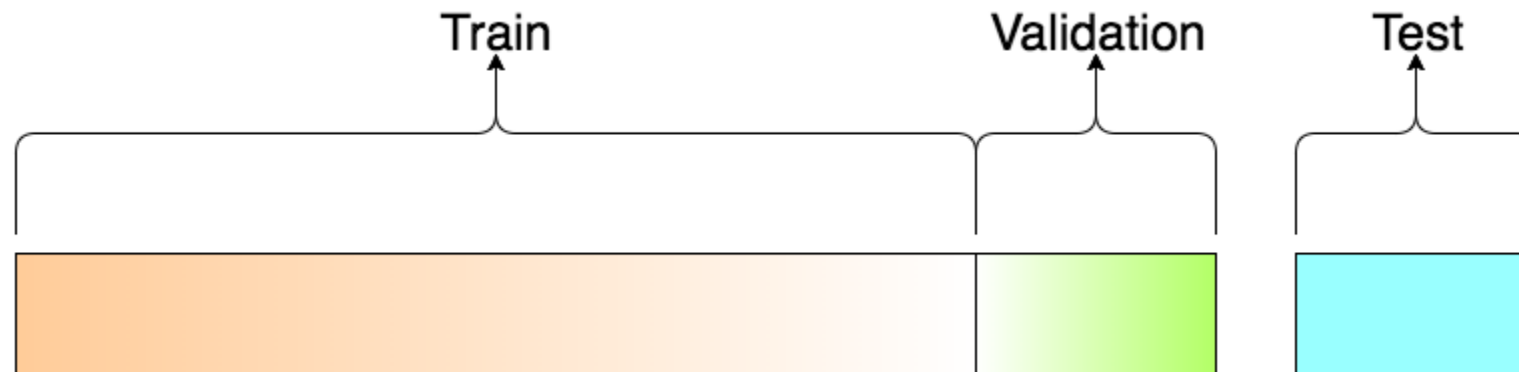
■ **Training set, validation set, test set**

Splitting the dataset

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

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

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



Overfitting

When the training process becomes too specific to the training set

■ Training set, validation set

Splitting the dataset

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

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

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

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

Loss

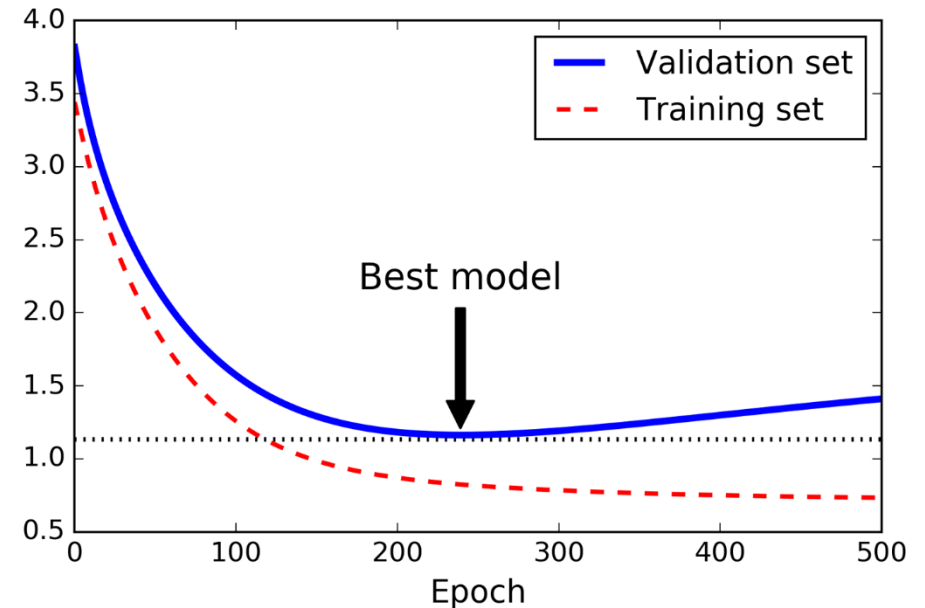


Image from <https://www.safaribooksonline.com/library/view/hands-on-machine-learning/9781491962282/ch04.html>

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 1 fold for testing
- 3) Unless all combinations have been considered, change combination and go back to 2)

Consider the *average test loss* across all possible combinations

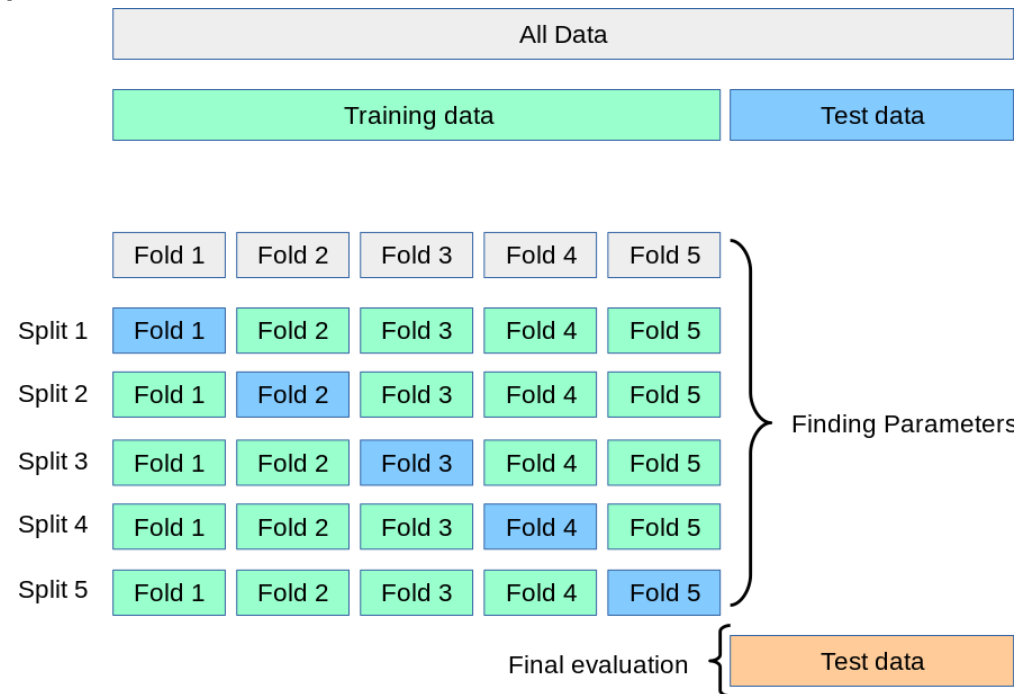


Image from <https://www.kdnuggets.com/2020/01/data-validation-machine-learning.html>