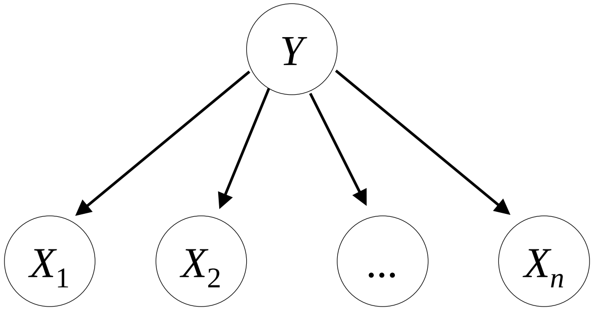
Probabilistic reasoning: *supervised learning and numerical optimization*

Marco Piastra

Prologue:
Logistic Regression

Graphical Models Redux

- *Naïve Bayesian Classifier*

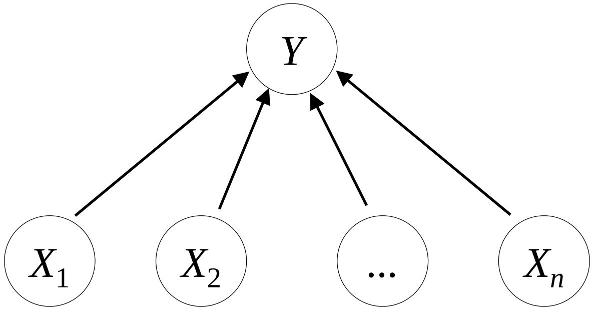


$$P(Y, X_1, \ldots, X_n) = P(Y) \prod_{i=1}^{n} P(X_i|Y)$$

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**



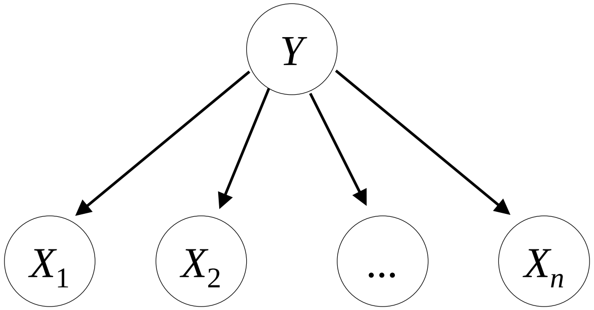
Just reverting the arrows …

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

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*

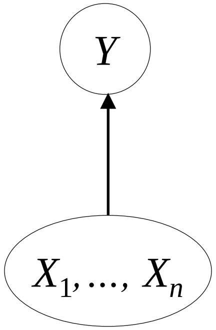


$$P(Y, X_1, \ldots, X_n) = P(Y) \prod_{i=1}^{n} P(X_i|Y)$$

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**



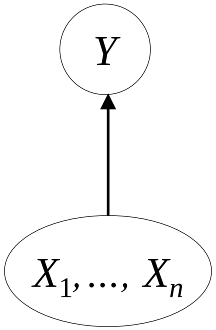
Removing any independence hypotheses …

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

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

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

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

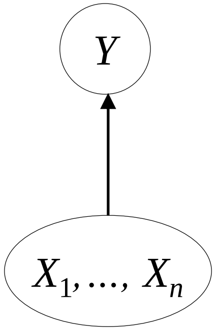
However…

2^n probabilities will have to be learnt

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

Logistic Regression

- *Graphical Model*



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

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

For convenience, define:

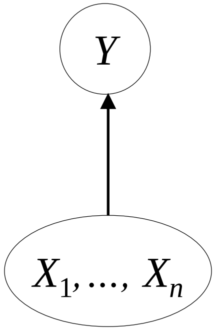
$$p(\boldsymbol{x}) := P(Y = 1|X_1 = x_1, \dots, X_n = x_n)$$ where $$\boldsymbol{x} := \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}$$ i.e. *a vector*

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

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

a vector of parameters

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

scalar product of vectors

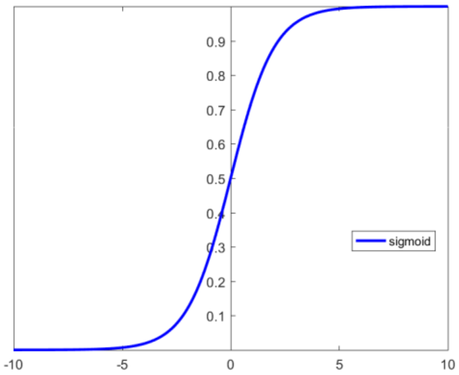
Logistic Regression
(i.e. a parametric distribution)

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

Logistic Regression

- *Maximum Likelihood Estimation*

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



Dataset

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

Conditional probability

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

Likelihood

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

A 'discriminative' model

This is a product of conditional probabilities (IID data)

Log-likelihood

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

$$= \sum_{i=1}^{N} y^{(i)} \log p(\boldsymbol{x}^{(i)}) + (1 - y^{(i)}) \log(1 - p(\boldsymbol{x}^{(i)}))$$

Logistic Regression

- *Maximum Likelihood Estimation*

Log-likelihood

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

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

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

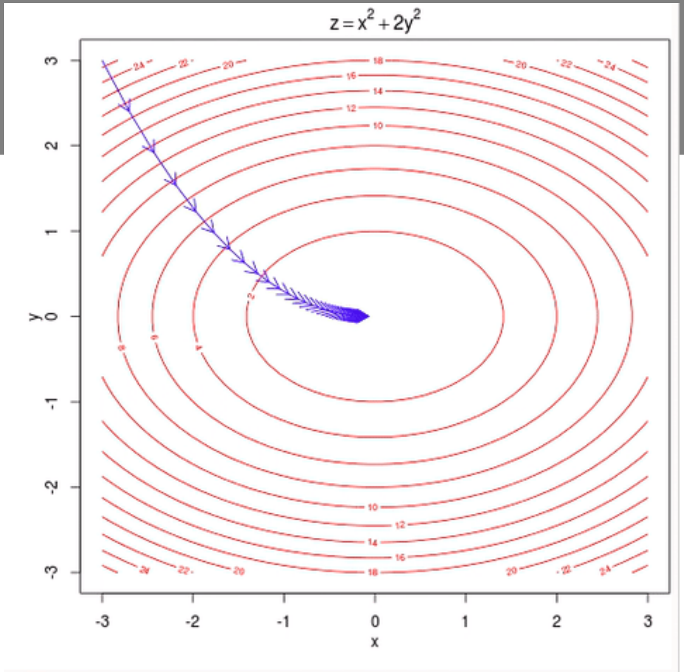
where

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

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

Gradient Descent
(and all that)

Gradient Descent (GD): intuition



- *Objective*

Turn this into a *<u>minimization</u>* *problem*

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

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

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

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

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

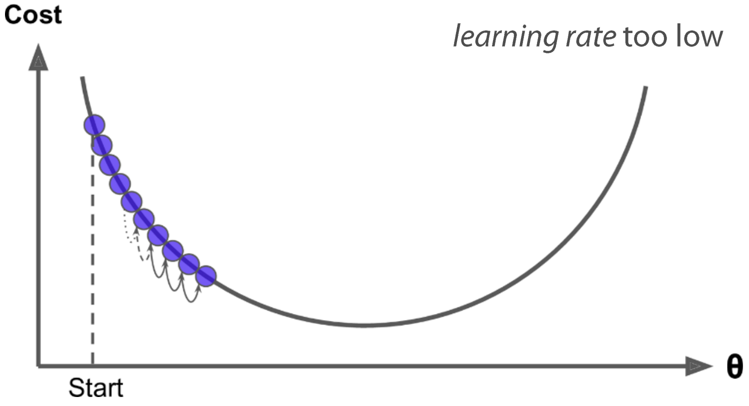
the gradient descent method converges to the optimal θ^* for $t \to \infty$
provided that $\eta \leq 1/C$

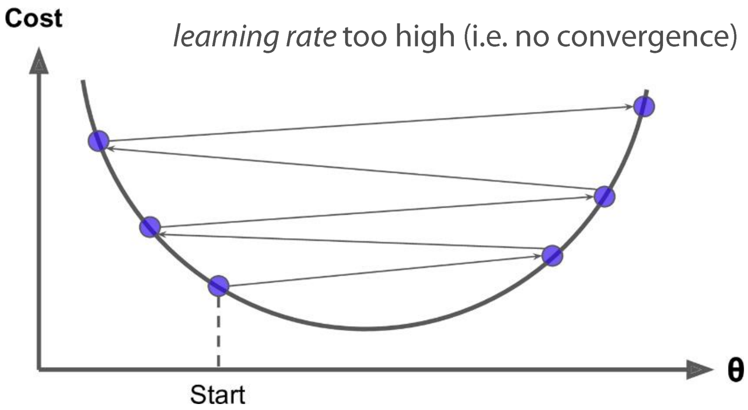
When $nl(D, \theta)$ is *derivable,* and *Lipschitz continuous* but <u>not</u> *convex*
the gradient descent method converges to a <u>local minimum</u> of $nl(D, \theta)$
under the same conditions

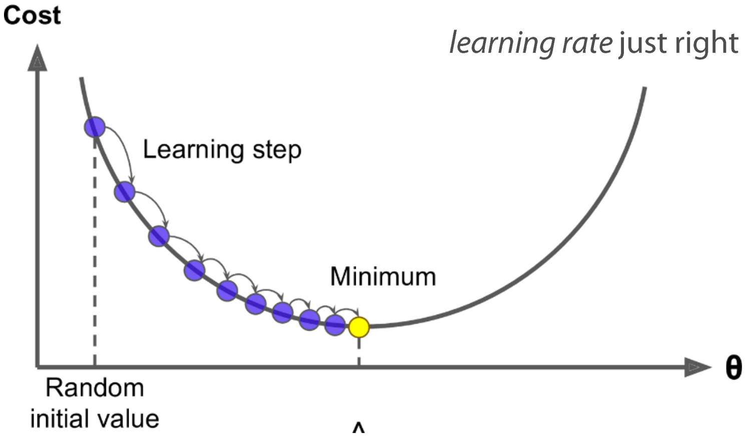
Gradient Descent (GD): practicalities

- *Convergence in practice*

  The choice of the *learning rate* η is crucial


learning rate too low


learning rate too high (i.e. no convergence)

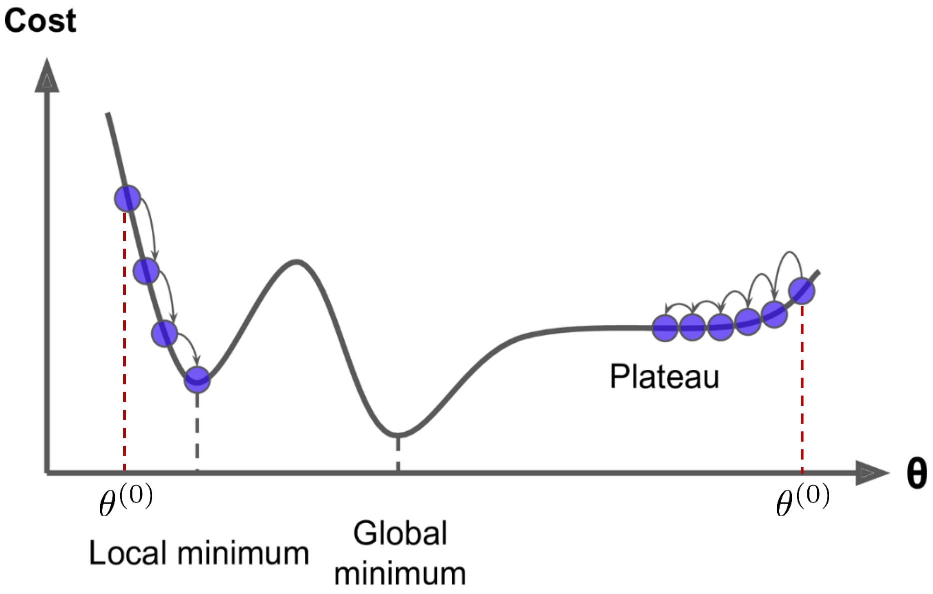

learning rate just right

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*

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

Stochastic Gradient Descent (SGD): intuition

- *Objective*

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

- *Iterative method*

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

Accuracy ρ is attained when

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

Define also $N := |D|$ $d := \dim(\vartheta)$

Size of data space

Dimension of parameter space

Time $:=$ time required to compute $\dfrac{\partial}{\partial \theta_j} \, nl(\boldsymbol{x}^{(i)}, y^{(i)}, \theta)$

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 $nl(D,\theta)$ *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 $\frac{\partial}{\partial\theta_j} nl(\boldsymbol{x}^{(i)}, y^{(i)}, \theta)$

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*

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

- *Iterative method*

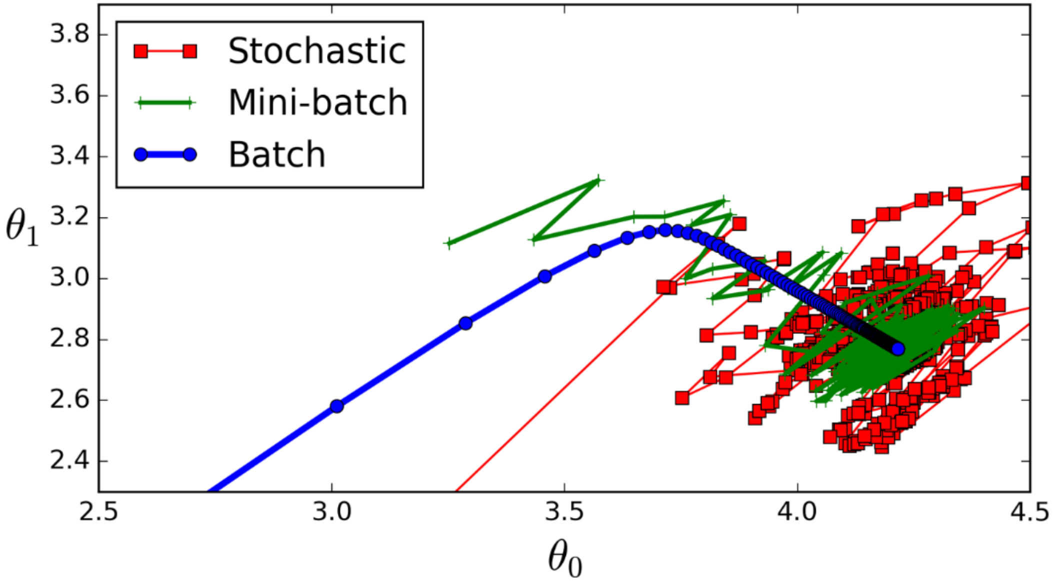
  1. Initialize $\theta^{(0)}$ at random
  2. Pick a mini batch $B \subseteq D$ with uniform probability
  3. Update $\theta^{(t)} = \theta^{(t-1)} - \eta^{(t)} \nabla_\theta \, nl(B, \theta^{(t-1)})$
  4. Unless some termination criterion has been met, go back to step 2.

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

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

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

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

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:

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

Data item indexes dropped, for simplicity

Logistic Regression

- *Log-likelihood gradients*

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

Logistic Regression

- *Log-likelihood gradients*

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

Logistic Regression: qualitative example

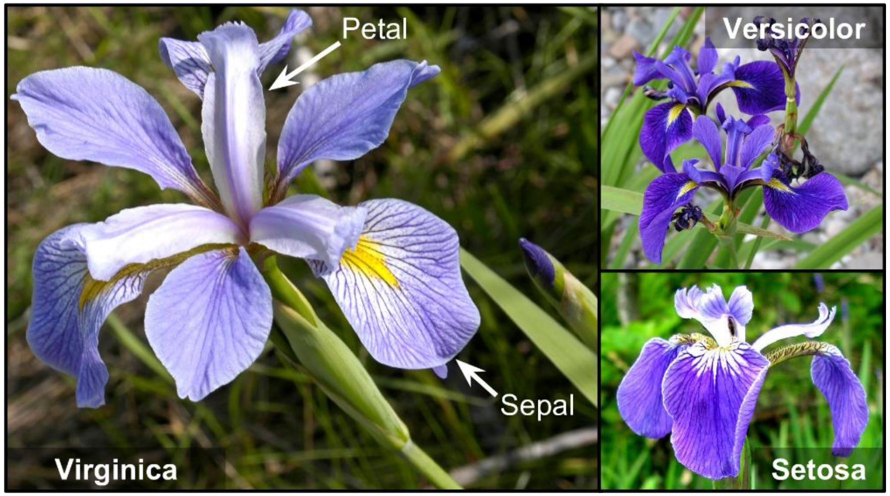
- *IRIS dataset*

  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:

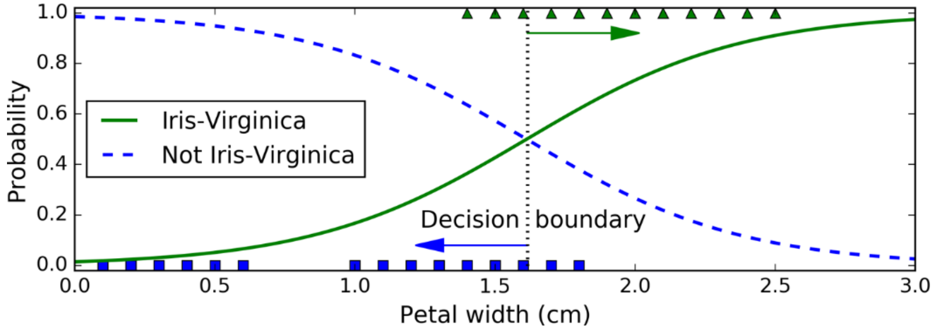


Image from https://www.safaribooksonline.com/library/view/hands-on-machine-learning/9781491962282/ch04.html

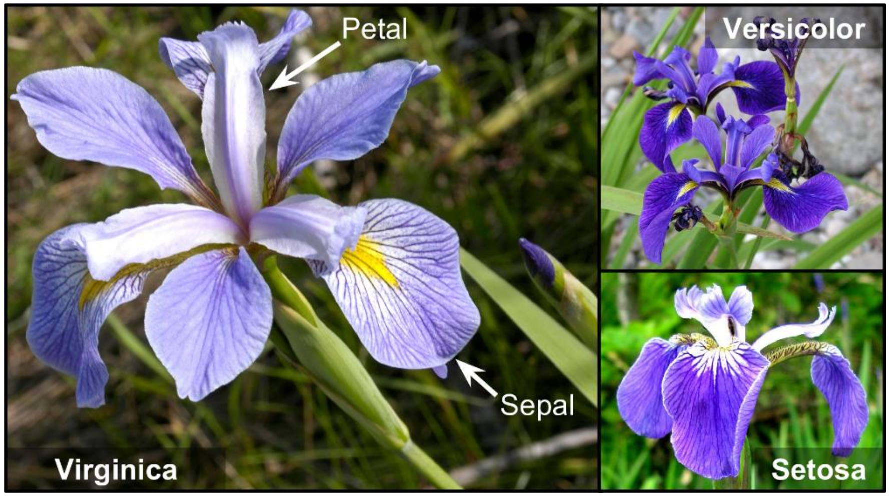
Logistic Regression: qualitative example

- *IRIS dataset*
  [*https://archive.ics.uci.edu/ml/datasets/iris*](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 <u>petal width</u> and <u>petal length</u> 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*

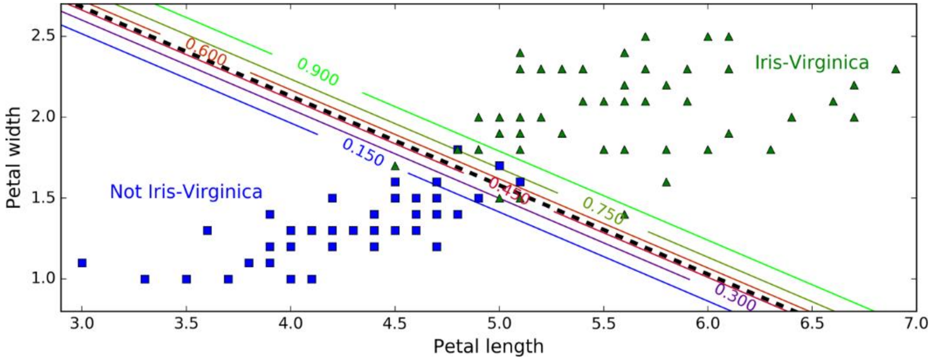


Image from https://www.safaribooksonline.com/library/view/hands-on-machine-learning/9781491962282/ch04.html