Numerical optimization and logistic regression

Esben Scriver Andersen*

*Australian National University

November 20, 2023

Unconstrained optimization

Unconstrained minimization problem

$$\min_{x \in A} \quad f(x)$$

Unconstrained optimization

Unconstrained minimization problem

$$\min_{x \in A} \quad f(x)$$

Recall, that we can transform any maximization problem into a minimization problem.

Example I

Consider the simple quadratic optimization problem

$$\min_{x \in \mathbb{R}} \quad ax + \frac{1}{2}b(x-c)^2$$

Example I

Consider the simple quadratic optimization problem

$$\min_{x \in \mathbb{R}} \quad ax + \frac{1}{2} b(x-c)^2$$

- $FOC : f'(x) = a + b(x-c) = 0$

Example I

Consider the simple quadratic optimization problem

$$\min_{x \in \mathbb{R}} \quad ax + \frac{1}{2}b(x-c)^2$$

- $FOC : f'(x) = a + b(x-c) = 0 \Leftrightarrow x^* = c - a/b$

Example I

Consider the simple quadratic optimization problem

$$\min_{x \in \mathbb{R}} \quad ax + \frac{1}{2}b(x-c)^2$$

- $FOC : f'(x) = a + b(x-c) = 0 \Leftrightarrow x^* = c - a/b$
- $SOC : f''(x) = b > 0$

Example I

Consider the simple quadratic optimization problem

$$\min_{x \in \mathbb{R}} \quad ax + \frac{1}{2}b(x-c)^2$$

- $FOC: f'(x) = a + b(x-c) = 0 \Leftrightarrow x^* = c - a/b$
- $SOC: f''(x) = b > 0$

As the FOC is linear in x, this optimization problem has a closed form solution

Example II

Now consider this exponential optimization problem

$$\min_{x \in \mathbb{R}} \quad e^{x} - 2e^{-2x} + e^{-3x}$$

Example II

Now consider this exponential optimization problem

$$\min_{x \in \mathbb{R}} \quad e^{x} - 2e^{-2x} + e^{-3x}$$

- $FOC : f'(x) = e^{x} + 4e^{-2x} - 3e^{-3x} = 0$

Example II

Now consider this exponential optimization problem

$$\min_{x\in\mathbb{R}} \quad e^x - 2e^{-2x} + e^{-3x}$$

- $FOC : f'(x) = e^x + 4e^{-2x} - 3e^{-3x} = 0$
- $SOC : f''(x) = e^x - 8e^{-2x} + 9e^{-3x} > 0$

Example II

Now consider this exponential optimization problem

$$\min_{x \in \mathbb{R}} \quad e^{x} - 2e^{-2x} + e^{-3x}$$

- $FOC : f'(x) = e^{x} + 4e^{-2x} - 3e^{-3x} = 0$
- $SOC : f''(x) = e^{x} - 8e^{-2x} + 9e^{-3x} > 0$

As the FOC is none-linear in x, this optimization problem has no closed form solution

Aim for the first half of the lecture

Introduce you to numerical methods used to solve optimization problems

Aim for the first half of the lecture

Introduce you to numerical methods used to solve optimization problems

Two classes of optimizers:

1. Gradient based (our focus)
2. None-gradient based

Aim for the first half of the lecture

Introduce you to numerical methods used to solve optimization problems

Two classes of optimizers:

1. Gradient based (our focus)
2. None-gradient based

Gradient based optimizers include (not conclusive):

- Newton's method
- BFGS
- BHHH
- Gradient descent

Newton's method

- Case: we cannot solve the optimization problem analytically.

$$\min_{x \in \mathbb{R}^k} \quad f(x)$$

Newton's method

- Case: we cannot solve the optimization problem analytically.

$$\min_{x \in \mathbb{R}^k} \quad f(x)$$

- Idea: A second order polynomial has a closed form solution. So, let's approximate $f(x)$ by a 2nd order Taylor polynomial in the point x_0

$$\min_{x \in \mathbb{R}^k} \quad f(x_0) + (x - x_0)^T \nabla f(x_0) + \tfrac{1}{2}(x - x_0)^T \nabla^2 f(x_0)(x - x_0)$$

Newton's method

- Case: we cannot solve the optimization problem analytically.

$$\min_{x \in \mathbb{R}^k} \quad f(x)$$

- Idea: A second order polynomial has a closed form solution. So, let's approximate $f(x)$ by a 2nd order Taylor polynomial in the point x_0

$$\min_{x \in \mathbb{R}^k} \quad f(x_0) + (x - x_0)^T \nabla f(x_0) + \tfrac{1}{2}(x - x_0)^T \nabla^2 f(x_0)(x - x_0)$$

- FOC: $\nabla f(x_0) + \nabla^2 f(x_0)(x - x_0) = 0$

Newton's method

- Case: we cannot solve the optimization problem analytically.

$$\min_{x \in \mathbb{R}^k} \quad f(x)$$

- Idea: A second order polynomial has a closed form solution. So, let's approximate $f(x)$ by a 2nd order Taylor polynomial in the point x_0

$$\min_{x \in \mathbb{R}^k} \quad f(x_0) + (x - x_0)^T \nabla f(x_0) + \frac{1}{2}(x - x_0)^T \nabla^2 f(x_0)(x - x_0)$$

- FOC: $\nabla f(x_0) + \nabla^2 f(x_0)(x - x_0) = 0 \Leftrightarrow x^* = x_0 - [\nabla^2 f(x_0)]^{-1} \nabla f(x_0)$

Newton's method

- Case: we cannot solve the optimization problem analytically.

$$\min_{x \in \mathbb{R}^k} \quad f(x)$$

- Idea: A second order polynomial has a closed form solution. So, let's approximate $f(x)$ by a 2nd order Taylor polynomial in the point x_0

$$\min_{x \in \mathbb{R}^k} \quad f(x_0) + (x - x_0)^T \nabla f(x_0) + \tfrac{1}{2}(x - x_0)^T \nabla^2 f(x_0)(x - x_0)$$

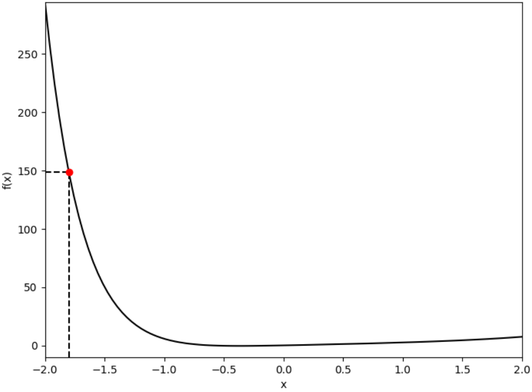
- FOC: $\nabla f(x_0) + \nabla^2 f(x_0)(x - x_0) = 0 \Leftrightarrow x^* = x_0 - [\nabla^2 f(x_0)]^{-1} \nabla f(x_0)$
- SOC: $[\nabla^2 f(x_0)]^{-1} \geq 0$

Example I: Consider the minimization problem without closed-form solution

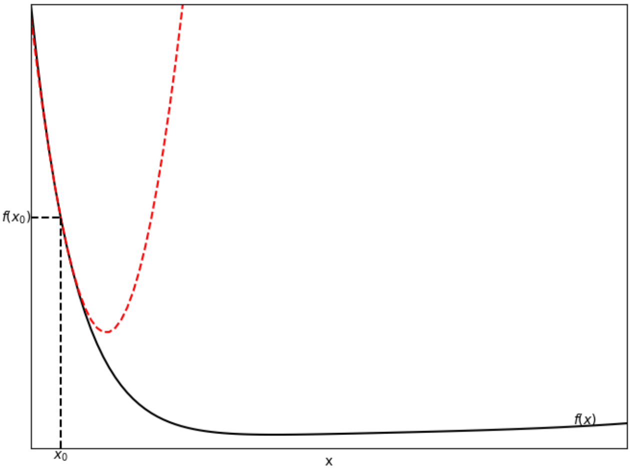
- $f(x) = e^x - 2e^{-2x} + e^{-3x}$
- $f'(x) = e^x + 4e^{-2x} - 3e^{-3x}$
- $f''(x) = e^x - 8e^{-2x} + 9e^{-3x}$

Example I: Consider the minimization problem without closed-form solution

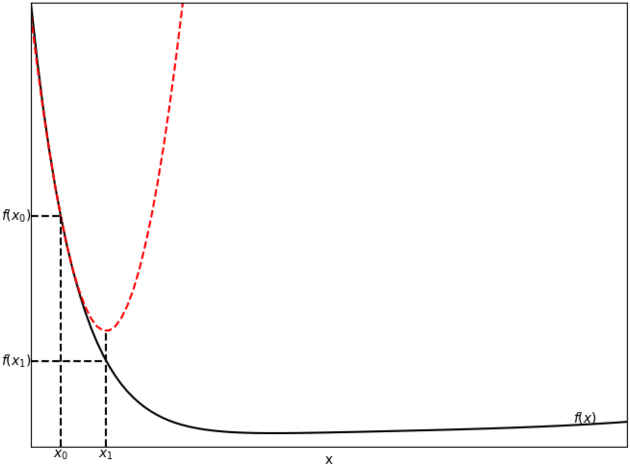
- $f(x) = e^{x} - 2e^{-2x} + e^{-3x}$
- $f'(x) = e^{x} + 4e^{-2x} - 3e^{-3x}$
- $f''(x) = e^{x} - 8e^{-2x} + 9e^{-3x}$



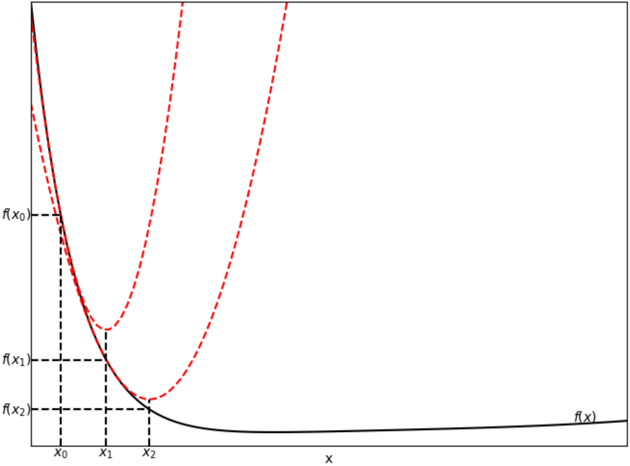
Example I: Approximate the function by the 2nd order Taylor approximation



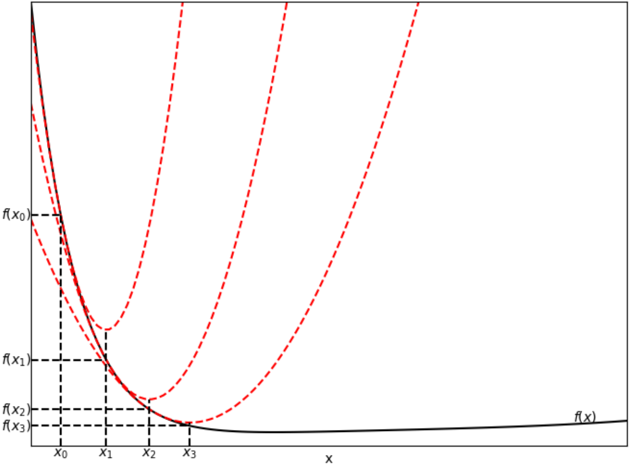
Example I: Find the minimum of the 2nd order Taylor approximation



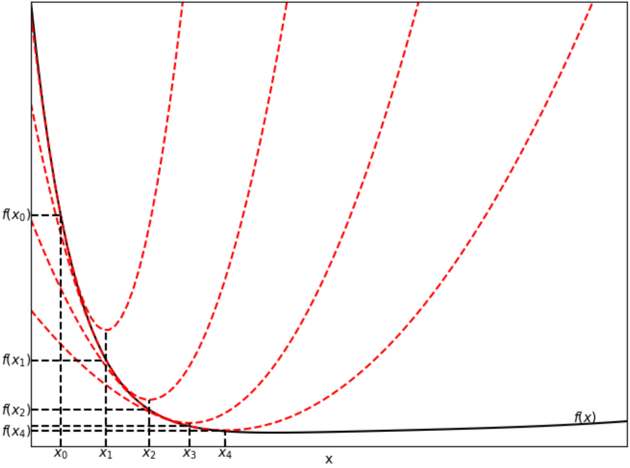
Example I: Repeat



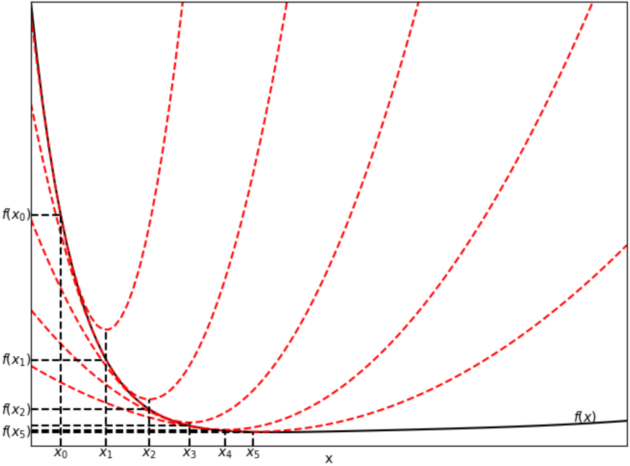
Example I: Repeat, repeat



Example I: Repeat, repeat, repeat



Example I: Repeat, repeat, repeat, ...



Newton's method

The simplest implementation of Newton's method starts from an initial guess, x_0, and then iterative update the solution of the FOC

$$x_{k+1} = x_k - [\nabla^2 f(x_k)]^{-1} \nabla f(x_k),$$

until the norm of the gradient is sufficiently close to zero, $\|\nabla f(x_k)\| < \varepsilon$.

Simple implementation of the Newton's method

```
def NewtonsMethod(x,grad,hess):
  convergence = 'failed'
  for k in range(1000):
    gradx = grad(x) #evaluate the gradient in x_{k}

    norm_grad = np.sum(np.abs(gradx), axis=None) #calculate the norm of the gradient
    if norm_grad < 1e-10: #stop if gradient close to zero
      convergence = 'converged'
      break

    dx =-np.linalg.solve(hess(x), gradx) #calculate the newton step
    x = x + dx #calculate x_{k+1}

  return x, convergence
```

Simple implementation of the Newton's method

```python
def NewtonsMethod(x,grad,hess):
  convergence = 'failed'
  for k in range(1000):
    gradx = grad(x) #evaluate the gradient in x_{k}

    norm_grad = np.sum(np.abs(gradx), axis=None) #calculate the norm of the gradient
    if norm_grad < 1e-10: #stop if gradient close to zero
      convergence = 'converged'
      break

    dx =-np.linalg.solve(hess(x), gradx) #calculate the newton step
    x = x + dx #calculate x_{k+1}

  return x, convergence
```

Let's take a closer look at how this works

Under- and over shooting

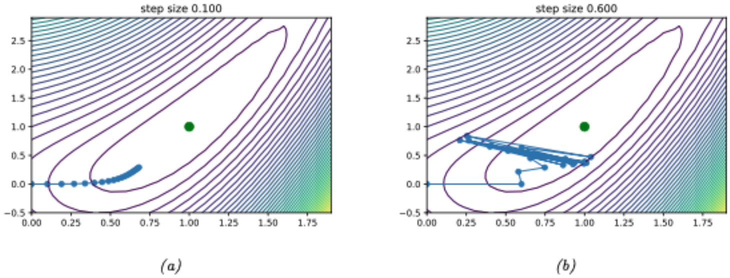


Figure 8.11: Steepest descent on a simple convex function, starting from $(0, 0)$*, for 20 steps, using a fixed step size. The global minimum is at* $(1, 1)$*. (a)* $\eta = 0.1$*. (b)* $\eta = 0.6$*. Generated by steepestDescentDemo.ipynb.*

Line search

For not well behaved objective functions, f, the performance of Newton's method can be improved through line search.

Line search

For not well behaved objective functions, f, the performance of Newton's method can be improved through line search.

Let Δx denotes the newton step

$$\Delta x \equiv -[\nabla^2 f(x_0)]^{-1} \nabla f(x_0).$$

Line search

For not well behaved objective functions, f, the performance of Newton's method can be improved through line search.

Let Δx denotes the newton step

$$\Delta x \equiv -[\nabla^2 f(x_0)]^{-1} \nabla f(x_0).$$

Line search is an algorithm that tries to find a good step length, $t\Delta x$,

$$x_{k+1} = x_k + t\Delta x.$$

Line search

For not well behaved objective functions, f, the performance of Newton's method can be improved through line search.

Let Δx denotes the newton step

$$\Delta x \equiv -[\nabla^2 f(x_0)]^{-1} \nabla f(x_0).$$

Line search is an algorithm that tries to find a good step length, $t\Delta x$,

$$x_{k+1} = x_k + t\Delta x.$$

- Exact line search for the optimal t

$$\min_{t \in \mathbb{R}^+} f(x_k + t\Delta x).$$

Line search

For not well behaved objective functions, f, the performance of Newton's method can be improved through line search.

Let Δx denotes the newton step

$$\Delta x \equiv -[\nabla^2 f(x_0)]^{-1} \nabla f(x_0).$$

Line search is an algorithm that tries to find a good step length, $t\Delta x$,

$$x_{k+1} = x_k + t\Delta x.$$

- Exact line search for the optimal t

$$\min_{t \in \mathbb{R}^+} f(x_k + t\Delta x).$$

- Inexact line search just tries to find an adequately t

Backtracking line search

Backtracking line search is a very simple inexact line search algorithm based on the Armijo–Goldstein condition

$$
\begin{aligned}
& f(x_k) - f(x_k + t\Delta x) > t\gamma, \\
\Leftrightarrow & f(x_k + t\Delta x) < f(x_k) - t\gamma,
\end{aligned}
$$

Backtracking line search

Backtracking line search is a very simple inexact line search algorithm based on the Armijo–Goldstein condition

$$\begin{aligned} &f(x_k) - f(x_k + t\Delta x) > t\gamma, \\ \Leftrightarrow &f(x_k + t\Delta x) < f(x_k) - t\gamma, \end{aligned}$$

where γ is proportional to the directional derivative

$$\gamma = -\alpha \nabla f(x_k)^T \Delta x.$$

Backtracking line search

Backtracking line search is a very simple inexact line search algorithm based on the Armijo–Goldstein condition

$$
\begin{aligned}
&f(x_k) - f(x_k + t\Delta x) > t\gamma, \\
\Leftrightarrow &f(x_k + t\Delta x) < f(x_k) - t\gamma,
\end{aligned}
$$

where γ is proportional to the directional derivative

$$\gamma = -\alpha \nabla f(x_k)^T \Delta x.$$

- If the condition is satisfied the improvement is considered adequate

Backtracking line search

Backtracking line search is a very simple inexact line search algorithm based on the Armijo–Goldstein condition

$$f(x_k) - f(x_k + t\Delta x) > t\gamma,$$
$$\Leftrightarrow f(x_k + t\Delta x) < f(x_k) - t\gamma,$$

where γ is proportional to the directional derivative

$$\gamma = -\alpha \nabla f(x_k)^T \Delta x.$$

- If the condition is satisfied the improvement is considered adequate
- If the condition is not met then reduce t proportionally by β, $t = \beta t$

Backtracking line search

Backtracking line search is a very simple inexact line search algorithm based on the Armijo–Goldstein condition

$$
\begin{aligned}
&f(x_k) - f(x_k + t\Delta x) > t\gamma, \\
\Leftrightarrow &f(x_k + t\Delta x) < f(x_k) - t\gamma,
\end{aligned}
$$

where γ is proportional to the directional derivative

$$\gamma = -\alpha \nabla f(x_k)^T \Delta x.$$

- If the condition is satisfied the improvement is considered adequate
- If the condition is not met then reduce t proportionally by β, $t = \beta t$
- Best practice is to set α between 0.01 and 0.30

Backtracking line search

Backtracking line search is a very simple inexact line search algorithm based on the Armijo–Goldstein condition

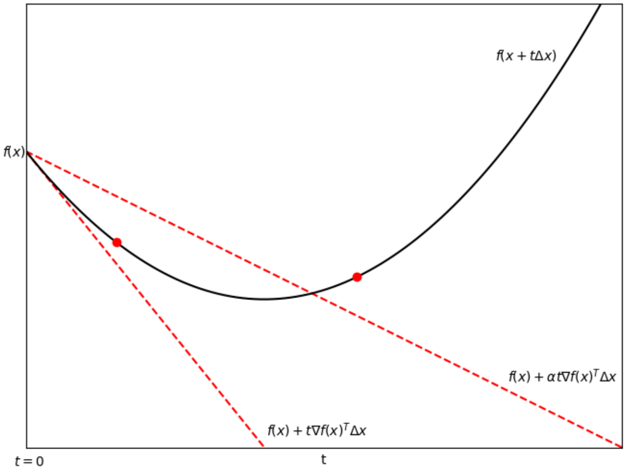
$$
\begin{aligned}
& f(x_k) - f(x_k + t\Delta x) > t\gamma, \\
\Leftrightarrow & f(x_k + t\Delta x) < f(x_k) - t\gamma,
\end{aligned}
$$

where γ is proportional to the directional derivative

$$\gamma = -\alpha \nabla f(x_k)^T \Delta x.$$

- If the condition is satisfied the improvement is considered adequate
- If the condition is not met then reduce t proportionally by β, $t = \beta t$
- Best practice is to set α between 0.01 and 0.30
- Best practice is to set β between 0.10 and 0.80

Backtracking line search with $\alpha < 1$



Implementation of Newton's method with backtracking

```
def NewtonsMethodBacktracking(fun,x0,grad,hess):
  convergence = 'failed'
  a, b = 0.2, 0.6 #backtracking parameters
  for k in range(1000):
    fun0 = fun(x0) #evaluate the function value in x_{k}
    grad0 = grad(x0) #evaluate the gradient in x_{k}

    norm_grad = np.sum(np.abs(grad0), axis=None) #calculate the norm of the gradient
    if norm_grad < 1e-10: #stop if gradient close to zero
      convergence = 'converged'
      break

    dx =-np.linalg.solve(hess(x0), grad0) #calculate the newton step

    t = 1 #initiate t step length
    x1 = x0 + dx #calculate initial x_{k+1}
    while (fun(x1) > fun0 + a * t * grad0 * dx): # Armijo-Goldstein condition
      t = b * t #update t if predicted improvement in f(x) is not adequately large
      x1 = x0 + t * dx #update x_{k+1}

    norm_step = np.sum(np.abs(t * dx), axis=None) #calculate the norm of the step size
    if norm_step < 1e-12: #stop if step is close to zero
      convergence = 'stopped early'
      break
  return x1, convergence
```

Calculating the gradient and hessian

Three ways to obtain the gradient, $\nabla f(x)$:

Calculating the gradient and hessian

Three ways to obtain the gradient, $\nabla f(x)$:

- analytical differentiation

Calculating the gradient and hessian

Three ways to obtain the gradient, $\nabla f(x)$:

- analytical differentiation
- numerical differentiation

$$\tfrac{\partial f(x)}{\partial x} \approx \frac{f(x+h) - f(x-h)}{2h}$$

Calculating the gradient and hessian

Three ways to obtain the gradient, $\nabla f(x)$:

- analytical differentiation
- numerical differentiation

$$\tfrac{\partial f(x)}{\partial x} \approx \frac{f(x+h) - f(x-h)}{2h}$$

- automatic differentiation (e.g. JAX, Pytorch, or Tensorflow)

Calculating the gradient and hessian

Three ways to obtain the gradient, $\nabla f(x)$:

- analytical differentiation
- numerical differentiation

$$\tfrac{\partial f(x)}{\partial x} \approx \frac{f(x+h) - f(x-h)}{2h}$$

- automatic differentiation (e.g. JAX, Pytorch, or Tensorflow)

As the hessian, $\nabla^2 f(x)$, is the second derivative we can also use numerical and automatic differentiation to calculate the hessian by simply applying the method twice.

BFGS

- For Newton's method we need to calculate the hessian (computational costly)

BFGS

- For Newton's method we need to calculate the hessian (computational costly)
- BFGS (Broyden–Fletcher–Goldfarb–Shanno) iteratively approximate the Hessian just from gradients

BFGS

- For Newton's method we need to calculate the hessian (computational costly)
- BFGS (Broyden–Fletcher–Goldfarb–Shanno) iteratively approximate the Hessian just from gradients

$$
\begin{aligned}
H_{k+1} &= H_k + \frac{yy^T}{y^T s} - \frac{H_k s s^T H_k^T}{s^T H_k s}, \\
y &\equiv \nabla f(x_{k+1}) - \nabla f(x_k), \\
s &\equiv x_{k+1} - x_k
\end{aligned}
$$

where H_0 typically is set to the identity matrix, $H_0 = I$

Stochastic Gradient descent

Consider the loss function

$$F(x) = \frac{1}{N}\sum_{i=1}^{N} f_i(x).$$

Stochastic Gradient descent

Consider the loss function

$$F(x) = \frac{1}{N}\sum_{i=1}^{N} f_i(x).$$

The gradient of the loss function has the form

$$\nabla F(x) = \frac{1}{N}\sum_{i=1}^{N} \nabla f_i(x).$$

Stochastic Gradient descent

Consider the loss function

$$F(x) = \frac{1}{N} \sum_{i=1}^{N} f_i(x).$$

The gradient of the loss function has the form

$$\nabla F(x) = \frac{1}{N} \sum_{i=1}^{N} \nabla f_i(x).$$

We can approximate the gradient by random sampling B, where $|B| < N$

$$\nabla F(x, B) \approx \frac{1}{|B|} \sum_{i \in B} \nabla f_i(x).$$

Stochastic Gradient descent

Consider the loss function

$$F(x) = \frac{1}{N}\sum_{i=1}^{N} f_i(x).$$

The gradient of the loss function has the form

$$\nabla F(x) = \frac{1}{N}\sum_{i=1}^{N} \nabla f_i(x).$$

We can approximate the gradient by random sampling B, where $|B| < N$

$$\nabla F(x, B) \approx \frac{1}{|B|}\sum_{i\in B} \nabla f_i(x).$$

The stochastic gradient descent method iteratively updates

$$x_{k+1} = x_k - t\nabla F(x_k, B_k),$$

Multinomial logistic regression

Multinomial logistic regression can be used as a classification model

$$p(y = c|x, \theta) = \frac{e^{a_c}}{\sum_{c'=1}^{C} e^{a_{c'}}},$$
$$a = b + Wx,$$

where $\theta = (b, W)$.

Maximum likelihood estimation

We can then estimate θ by maximum likelihood estimation (MLE) by maximizing the log-likelihood function

$$\hat{\theta} = \underset{\theta \in \mathbb{R}^{(C-1)D}}{\arg\max} \prod_{i=1}^{N} \prod_{c=1}^{C} p(y_i = c | x_i, \theta)^{y_i}.$$

Maximum likelihood estimation

We can then estimate θ by maximum likelihood estimation (MLE) by maximizing the log-likelihood function

$$\hat{\theta} = \underset{\theta \in \mathbb{R}^{(C-1)D}}{\arg\max} \prod_{i=1}^{N} \prod_{c=1}^{C} p(y_i = c | x_i, \theta)^{y_i}.$$

The stated maximization problem can be simplified to a minimization problem of the cross-entropy

$$\hat{\theta} = \underset{\theta \in \mathbb{R}^{(C-1)D}}{\arg\min} -\frac{1}{N} \sum_{i=1}^{N} \sum_{c=1}^{C} y_i \log p(y_i = c | x_i, \theta).$$

Maximum likelihood estimation

We can then estimate θ by maximum likelihood estimation (MLE) by maximizing the log-likelihood function

$$\hat{\theta} = \underset{\theta \in \mathbb{R}^{(C-1)D}}{\arg\max} \prod_{i=1}^{N} \prod_{c=1}^{C} p(y_i = c | x_i, \theta)^{y_i}.$$

The stated maximization problem can be simplified to a minimization problem of the cross-entropy

$$\hat{\theta} = \underset{\theta \in \mathbb{R}^{(C-1)D}}{\arg\min} -\frac{1}{N} \sum_{i=1}^{N} \sum_{c=1}^{C} y_i \log p(y_i = c | x_i, \theta).$$

Let's take a look at a practical example

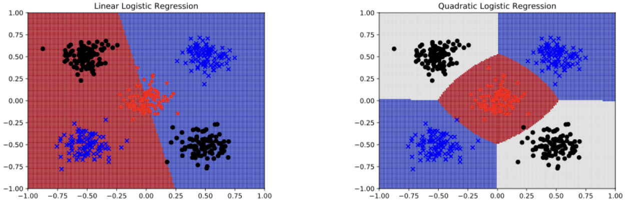
Decision rule

If the loss for misclassifying each class is the same, then the optimal decision rule is to predict $\hat{y} = c$ iff class c is the most likely class

Decision rule

If the loss for misclassifying each class is the same, then the optimal decision rule is to predict $\hat{y} = c$ iff class c is the most likely class

$$\hat{y} = \underset{c=1,\ldots,C}{\arg\max}\, p(y = c|x, \theta)$$



(a) *(b)*

Figure 10.7: Example of 3-class logistic regression with 2d inputs. (a) Original features. (b) Quadratic features. Generated by logreg_multiclass_demo.ipynb.

Overfit

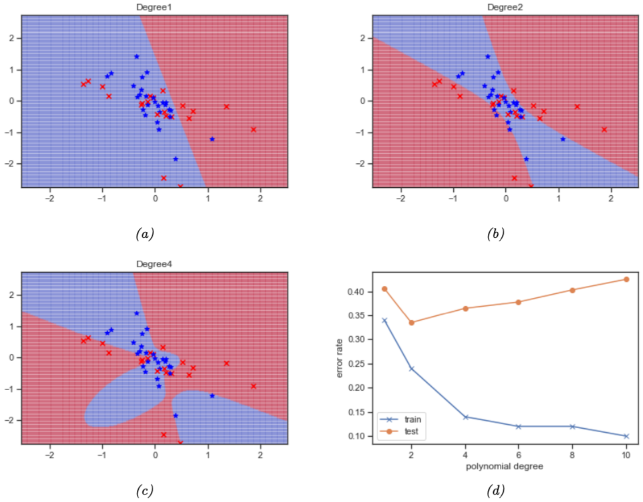


Figure 10.4: Polynomial feature expansion applied to a two-class, two-dimensional logistic regression problem. (a) Degree $K = 1$. (b) Degree $K = 2$. (c) Degree $K = 4$. (d) Train and test error vs degree. Generated by logreg_poly_demo.ipynb.

Regularization

Simplest regularization

$$\hat{\theta} = \underset{\theta \in \mathbb{R}^{(C-1)D}}{\arg\min} -\frac{1}{N} \sum_{i=1}^{N} \sum_{c=1}^{C} y_i \log p(y_i = c | x_i, \theta) + \lambda ||\theta||_2^2.$$

Important to standardize the features (mean 0 and variance 1). Alternatively, use min-max scaling of the features (the features lie in the [0,1]).

Thank you for today :)