COMP 790-125: Goals for today

- Regression – recap
- Classification
- A mixture model
- Sigmoid
- Logistic regression
- Newton method

Regression – recap

Data was given as $\mathcal{D} = \{(\mathbf{x}_i, y_i) : i = 1..n\}$ where $y_i \in \mathbf{R}$ and $\mathbf{x}_i \in \mathbf{R}^p$.

We wanted to find a function $f : \mathbf{R}^p \to \mathbf{R}$ such that for a new pair $(\mathbf{x}^{\text{new}}, y^{\text{new}}) \notin \mathcal{D}$

$$f(\mathbf{x}^{\text{new}}) \approx y^{\text{new}}.$$

Regression – Maximum Likelihood

We assumed a model

$$p(y|\mathbf{x}, \beta_0, \boldsymbol{\beta}) = \frac{1}{\sqrt{2\pi}} \exp\{-\frac{1}{2}(y - f(\mathbf{x}; \beta_0, \boldsymbol{\beta}))^2\}$$

where

$$f(\mathbf{x}; \boldsymbol{\beta}_0, \beta) = \beta_0 + \langle \mathbf{x}, \boldsymbol{\beta} \rangle \,.$$

This gave us an objective, log-likelihood

$$\mathrm{LL}(\beta_0, \boldsymbol{\beta}; \mathcal{D}) = -\frac{1}{2} \sum_i (y_i - f(\mathbf{x}_i; \beta_0, \boldsymbol{\beta}))^2 + \mathrm{const.},$$

and by optimizing this objective we obtained *maximum likelihood* estimates β_0^{ML} and $\boldsymbol{\beta}^{\mathrm{ML}}$.

Regression – Maximum a posteriori

We made observations about a need to make further assumptions about $\boldsymbol{\beta}$ and imposed priors. This resulted in a number of models.

One example is Lasso regression

$$\begin{aligned} p(\beta_j) &= \frac{\lambda}{2}\exp\{-\lambda|\beta_j|\}, j = 1, \ldots, p \\ p(y|\mathbf{x}, \beta_0, \boldsymbol{\beta}) &= \frac{1}{\sqrt{2\pi}}\exp\{-\frac{1}{2}(y - f(\mathbf{x}; \beta_0, \boldsymbol{\beta}))^2\} \end{aligned}$$

and this give us a new objective, log posterior probability,

$$\mathrm{LP}(\beta_0, \boldsymbol{\beta}; \mathcal{D}) = -\frac{1}{2}\sum_{i=1}^{n}(y_i - f(\mathbf{x}_i; \beta_0, \boldsymbol{\beta}))^2 - \sum_{j=1}^{p}\lambda|\beta_j| + \mathrm{const.}$$

and by optimizing this objective we obtained *maximum-a-posteriori* estimates $\beta_0^{\mathrm{MAP}}, \boldsymbol{\beta}^{\mathrm{MAP}}$.

Notation – proportional to

You probably know this already

$$f(x) \propto g(x) \equiv \exists c, f(x) = cg(x).$$

We will use $\propto$, almost exclusively, to talk about unnormalized probabilities[1]

$$\underbrace{p(x)}_{\text{probability}} \quad \propto \quad \underbrace{f(x)}_{\text{unnormalized probability}}$$

$$p(x) = \frac{1}{Z} f(x)$$

$$Z = \int_x f(x) dx$$

$\frac{1}{Z}$ is a *normalization constant*.

[1] unnormalized probability does not sum to 1, but it is non-negative

Notation

So we might write

$$p(x; \mu, \sigma) \propto \exp\{-\frac{1}{2\sigma^2}(x-\mu)^2\}$$

and in this case

$$Z = \int_x \exp\{-\frac{1}{2\sigma^2}(x-\mu)^2\} dx = \sqrt{2\pi\sigma^2}$$

and so

$$p(x; \mu, \sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\{-\frac{1}{2\sigma^2}(x-\mu)^2\}$$

We shorten the math when the normalization constant is known.

Classification – discrete y

Frequently we care about predicting a discrete variable (label) with 2,3,4 ... different *unordered* states

Today we deal with modeling a binary target variable:

- healthy vs. cancer
- responsive vs. nonresponsive
- binding vs. nonbinding peptide

Sometimes a binary variable is the result of a discretization (high/low).

Binary Classification

Data is given as $\mathcal{D} = \{(\mathbf{x}_i, y_i) : i = 1..n\}$ where $y_i \in \{-1, +1\}$ and $\mathbf{x}_i \in \mathbf{R}^p$.

We now need to specify a model for

$$p(y|\mathbf{x})$$

A mixture model

There are many ways to arrive at the model for $p(y|\mathbf{x})$ we will follow generative one.

Suppose we have two classes 1 and 2 and suppose we knew probabilistic models for both of them. For example

$$\begin{aligned} p(\mathbf{x}|y=-1) &\propto \exp\{-\frac{1}{2}(\mathbf{x}-\mu_1)^T\Sigma_1^{-1}(\mathbf{x}-\mu_1)\} \\ p(\mathbf{x}|y=+1) &\propto \exp\{-\frac{1}{2}(\mathbf{x}-\mu_2)^T\Sigma_2^{-1}(\mathbf{x}-\mu_2)\} \end{aligned}$$

further let us assume that we have a biased coin that gives heads with probability q

$$\begin{aligned} p(y=-1) &= q \\ p(y=+1) &= 1-q \end{aligned}$$

A mixture model

How would we generate dataset like $\mathcal{D} = \{(\mathbf{x}_i, y_i) : i = 1..n\}$.

Repeat n times

- sample class from $p(y)$
- given class y, sample $\mathbf{x}$ from $p(\mathbf{x}|y)$

Querying the mixture model

Suppose a new data point $\mathbf{x}^{\text{new}}$ was generated by someone else and they did not tell us the class y^{new}, so what is

$$p(y^{\text{new}} = -1|\mathbf{x}^{\text{new}})$$

Querying the mixture model

Bayes' rule

$$
\begin{aligned}
p(y=-1|\mathbf{x}) &= \frac{p(y=-1,\mathbf{x})}{p(\mathbf{x})} \\
&= \frac{p(y=-1)p(\mathbf{x}|y=-1)}{p(y=-1)p(\mathbf{x}|y=-1)+p(y=+1)p(\mathbf{x}|y=+1)}
\end{aligned}
$$

and a bit more simplified

$$
p(y=-1|\mathbf{x}) = \frac{1}{1+\frac{p(y=+1)p(\mathbf{x}|y=+1)}{p(y=-1)p(\mathbf{x}|y=-1)}} = \frac{1}{1+\frac{p(\mathbf{x},y=+1)}{p(\mathbf{x},y=-1)}}
$$

Sigmoid

For computation of $p(y = -1|\mathbf{x})$ the exact parameters involved in computing $p(\mathbf{x}, y = +1)$ and $p(\mathbf{x}, y = -1)$ are not required, we only need their ratio – odds.

In fact we can consider just log-odds, since log is a monotone function

$$z = \log \frac{p(\mathbf{x}, y = -1)}{p(\mathbf{x}, y = +1)}$$

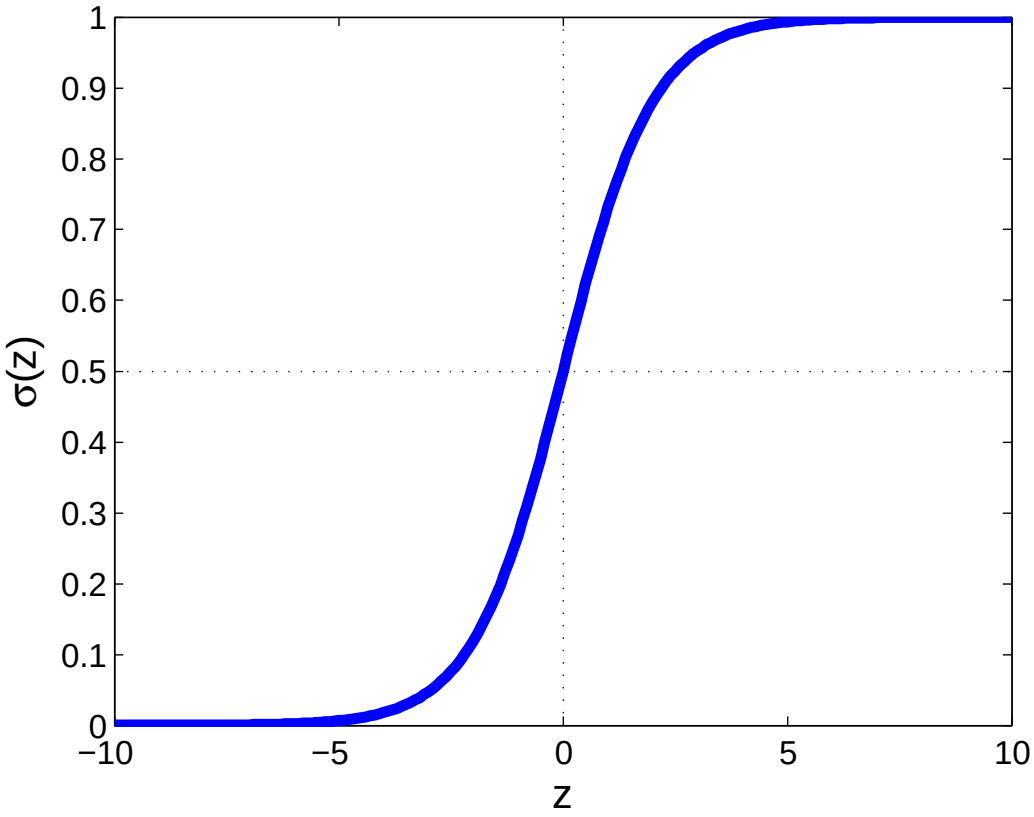
So we can write

$$p(y = -1|z) = \frac{1}{1 + \exp\{-z\}}$$

Sigmoid

Sigmoid function

$$\sigma(z) = \frac{1}{1 + \exp\{-z\}}$$



Classification – $p(y|\mathbf{x})$

Data is given as $\mathcal{D} = \{(\mathbf{x}_i, y_i) : i = 1..n\}$ where $y_i \in \{-1, +1\}$ and $\mathbf{x}_i \in \mathbf{R}^p$.

We can specify a model

$$p(y|\mathbf{x}) = \frac{1}{1 + \exp\{-f(y, \mathbf{x})\}}$$

y is the label and $f(y, \mathbf{x})$ corresponds to log-odds.

The remaining choice to be made

$$f(y, \mathbf{x}; \beta_0, \boldsymbol{\beta}) = y(\beta_0 + \langle \boldsymbol{\beta}, \mathbf{x} \rangle)$$

Decision boundary

Given our model

$$p(y|\mathbf{x}) = \frac{1}{1 + \exp\{-f(y, \mathbf{x})\}}$$

with

$$f(y, \mathbf{x}; \beta_0, \boldsymbol{\beta}) = y(\beta_0 + \langle \boldsymbol{\beta}, \mathbf{x} \rangle)$$

when is the probability $p(y = -1|\mathbf{x}) = p(y = +1|\mathbf{x}) = 0.5$?

Decision boundary

In case our model $p(y = -1|\mathbf{x}) = p(y = +1|\mathbf{x}) = 0.5$ when

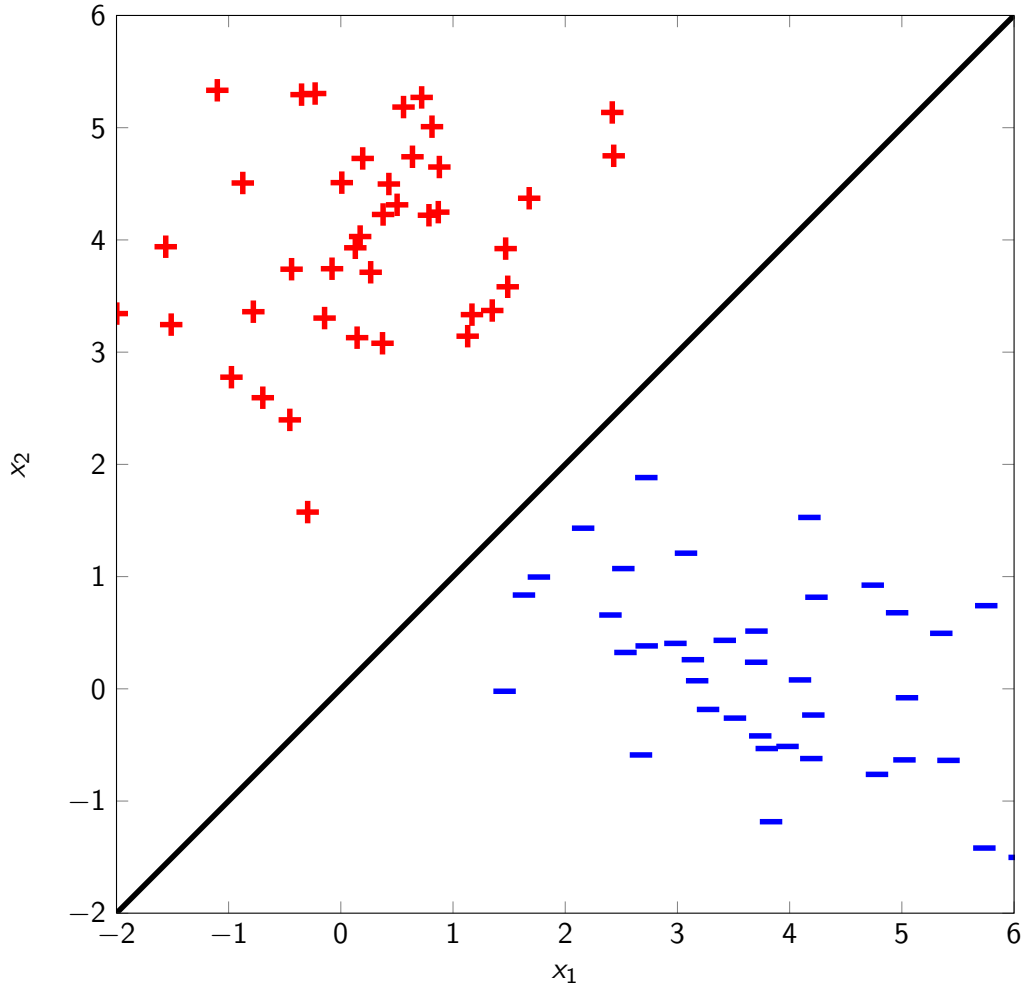
$$f(y, \mathbf{x}; \beta_0, \boldsymbol{\beta}) = y(\beta_0 + \langle \boldsymbol{\beta}, \mathbf{x} \rangle) = 0$$

and since $y \in \{-1, +1\}$

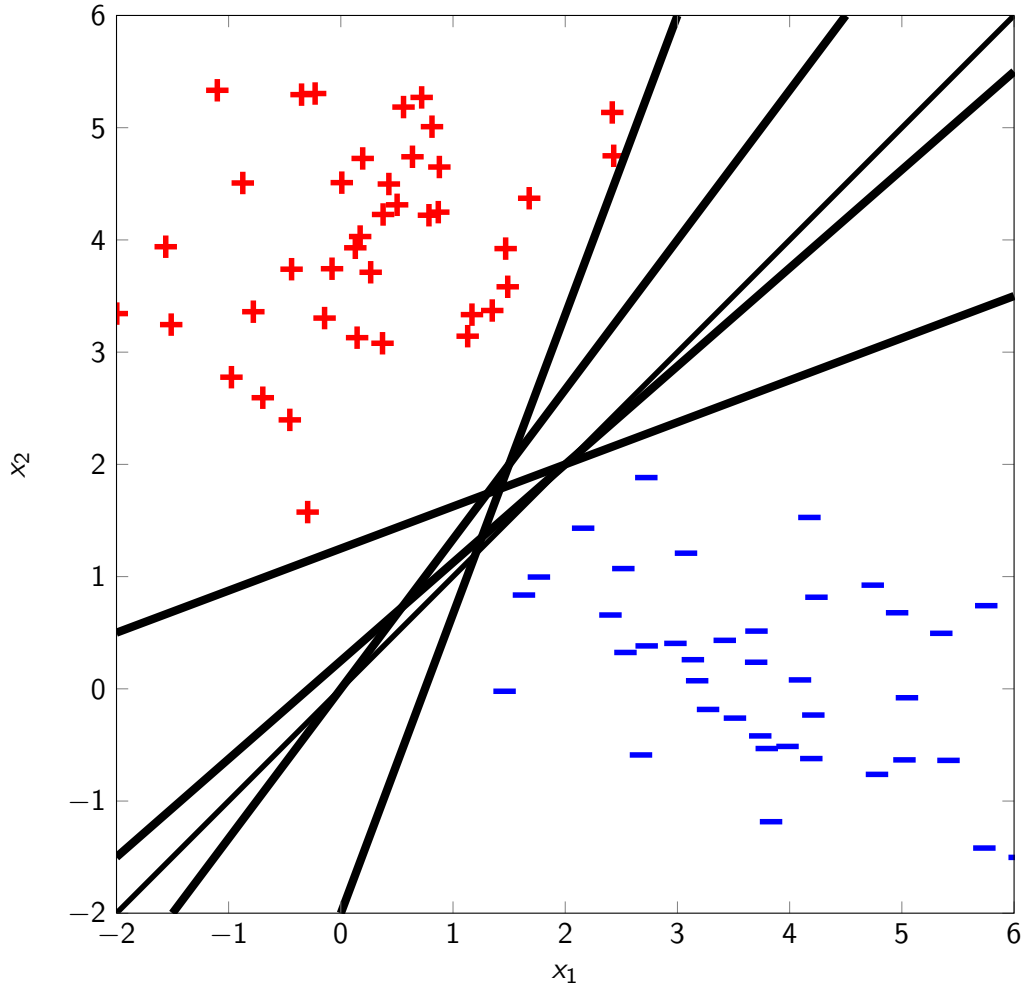
$$\langle \boldsymbol{\beta}, \mathbf{x} \rangle = -\beta_0.$$

This is a linear equation that defines a hyperplane.

Separating hyperplane



Separating hyperplanes



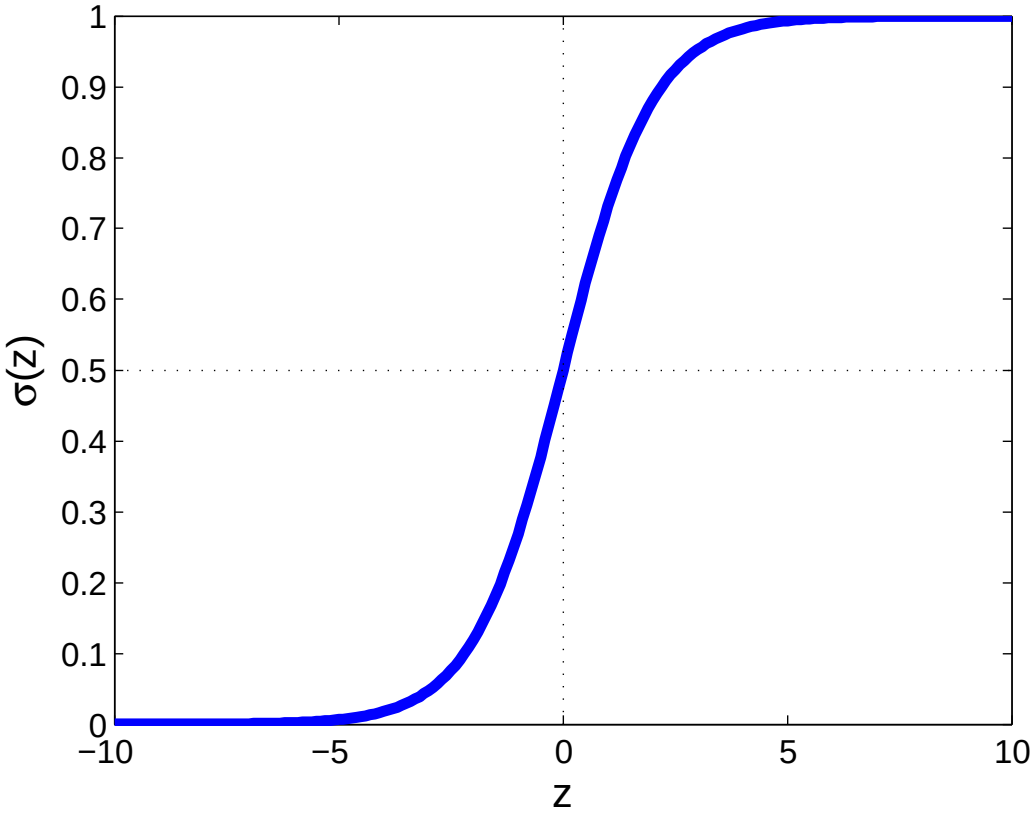
Sigmoid

Our model

$$p(y|\mathbf{x}) = \frac{1}{1 + \exp\{-f(y, \mathbf{x})\}}$$

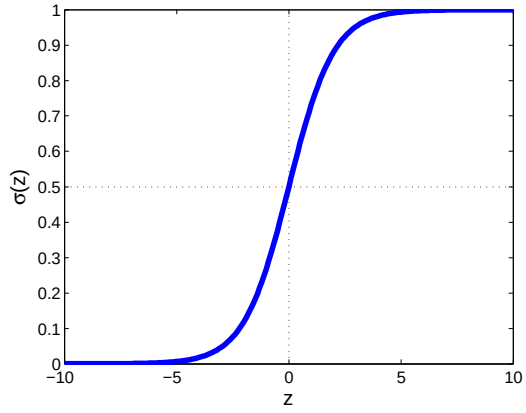
Sigmoid function

$$\sigma(z) = \frac{1}{1 + \exp\{-z\}}$$

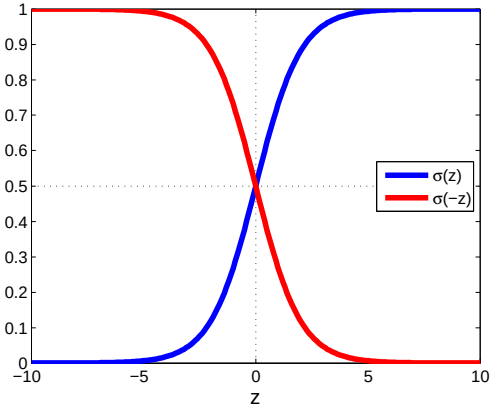


Sigmoid:Transforming your scalars into probabilities since...

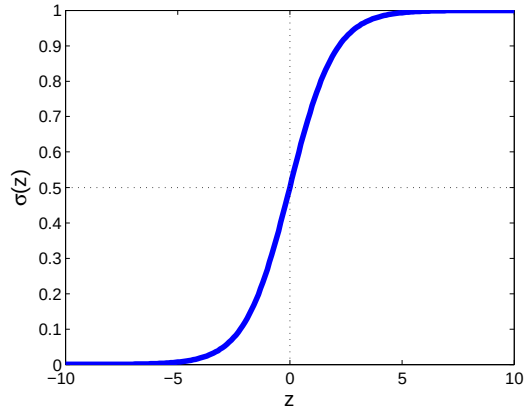
$$\sigma(z) = \frac{1}{1+\exp\{-z\}}$$



$$1 - \sigma(z) = \sigma(-z)$$

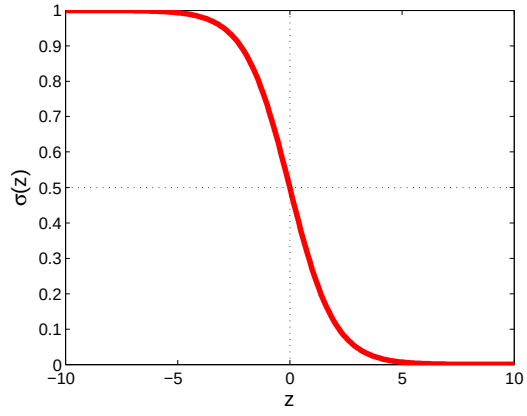


Sigmoid



$$p(y = +1|\mathbf{x}, \beta_0, \boldsymbol{\beta}) = \sigma(+(\beta_0 + \langle \mathbf{x}, \boldsymbol{\beta} \rangle)) = \frac{1}{1 + \exp\{-(\beta_0 + \langle \mathbf{x}, \boldsymbol{\beta} \rangle)\}}$$

Sigmoid



$$p(y = -1|\mathbf{x}, \beta_0, \boldsymbol{\beta}) = \sigma(-(\beta_0 + \langle \mathbf{x}, \boldsymbol{\beta} \rangle)) = \frac{1}{1 + \exp\{+(\beta_0 + \langle \mathbf{x}, \boldsymbol{\beta} \rangle)\}}$$

Logistic regression

Probability of y

$$p(y|\mathbf{x}, \beta_0, \boldsymbol{\beta}) = \frac{1}{1 + \exp\{-y(\beta_0 + \langle \mathbf{x}, \boldsymbol{\beta} \rangle)\}}$$

Likelihood function

$$L(\beta_0, \boldsymbol{\beta}) = \prod_i p(y_i|\mathbf{x}_i, \beta_0, \boldsymbol{\beta}) = \prod_i \frac{1}{1 + \exp\{-y_i(\beta_0 + \langle \mathbf{x}_i, \boldsymbol{\beta} \rangle)\}}$$

Log-Likelihood function

$$\mathrm{LL}(\beta_0, \boldsymbol{\beta}) = -\sum_i \log\{1 + \exp\{-y_i(\beta_0 + \langle \mathbf{x}_i, \boldsymbol{\beta} \rangle)\}\}$$

Maximum likelihood

We are again pursuing maximum likelihood estimate for $\beta_0, \boldsymbol{\beta}$ this time for logistic regression.

We need to ascend the likelihood surface to find optimal $\beta_0, \boldsymbol{\beta}$.

We will again use partial derivatives to accomplish this.

$$\nabla \mathrm{LL}(\beta_0, \boldsymbol{\beta}) = \begin{bmatrix} \frac{\partial LL(\beta_0, \boldsymbol{\beta})}{\partial \beta_0} \\ \frac{\partial LL(\beta_0, \boldsymbol{\beta})}{\partial \beta_1} \\ \cdots \\ \frac{\partial LL(\beta_0, \boldsymbol{\beta})}{\partial \beta_p} \end{bmatrix}$$

Optimization

$$\mathrm{LL}(\beta_0, \boldsymbol{\beta}) = -\sum_i \log\{1 + \exp\{-y_i(\beta_0 + \langle \mathbf{x}_i, \boldsymbol{\beta}\rangle)\}\}$$

We can take partial derivatives

$$\begin{aligned}
\frac{\partial \mathrm{LL}(\beta_0, \boldsymbol{\beta})}{\partial \beta_j} &= \sum_i \frac{\exp\{-y_i(\beta_0 + \langle \mathbf{x}_i, \boldsymbol{\beta}\rangle)\}}{1 + \exp\{-y_i(\beta_0 + \langle \mathbf{x}_i, \boldsymbol{\beta}\rangle)\}} y_i x_{i,j} \\
&= \sum_i \left(1 - \frac{1}{1 + \exp\{-y_i(\beta_0 + \langle \mathbf{x}_i, \boldsymbol{\beta}\rangle)\}}\right) y_i x_{i,j} \\
&= \sum_i \left(1 - p(y_i | \mathbf{x}_i, \beta_0, \boldsymbol{\beta})\right) y_i x_{i,j}
\end{aligned}$$

Optimization: Cannot get closed form coordinate ascent

We have partial derivatives

$$\frac{\partial \mathrm{LL}(\beta_0, \boldsymbol{\beta})}{\partial \beta_j} = \sum_i \left(1 - p(y_i | \mathbf{x}_i, \beta_0, \boldsymbol{\beta})\right) y_i x_{i,j}$$

but we cannot solve for optimal β_j by setting the partial derivatives to 0.

Optimization

We cannot do coordinate ascent so we will use gradient a little differently

$$\begin{aligned}
\nabla \mathrm{LL}(\beta_0, \boldsymbol{\beta}) &= \begin{bmatrix} \sum_i (1 - p(y_i|\mathbf{x}_i, \beta_0, \boldsymbol{\beta}))y_i & (\beta_0) \\ \sum_i (1 - p(y_i|\mathbf{x}_i, \beta_0, \boldsymbol{\beta}))y_i x_{i,1} & (\beta_1) \\ \dots & \\ \sum_i (1 - p(y_i|\mathbf{x}_i, \beta_0, \boldsymbol{\beta}))y_i x_{i,p} & (\beta_p) \end{bmatrix} \\
&= \sum_i (1 - p(y_i|\mathbf{x}_i, \beta_0, \boldsymbol{\beta}))y_i \begin{bmatrix} 1 \\ \mathbf{x}_i \end{bmatrix}
\end{aligned}$$

Check your gradients – finite differences

Remember how derivatives are defined. Here is a numerical analog of that definition

$$\frac{\mathrm{LL}(\beta_0, \beta_1, ..., \beta_j + h, ...\beta_n) - \mathrm{LL}(\beta_0, \beta_1, ..., \beta_j, ...\beta_n)}{h}$$

approximate

$$\frac{\partial \mathrm{LL}}{\partial \beta_j}(\beta_0, \boldsymbol{\beta})$$

for small h (1e-10)

Check your gradients – symbolically

A lot of effort has gone into building good symbolic systems, let it not go to waste.

Silly demo of Mathematica

$$\mathrm{LL}(\beta_0, \boldsymbol{\beta}) = -\sum_i \log\{1 + \exp\{-y_i(\beta_0 + \langle \mathbf{x}_i, \boldsymbol{\beta} \rangle)\}\}$$

Easy optimization: gradient ascent

$$\begin{aligned}
\beta_0^{\text{new}} &= \beta_0 + s \sum_i (1 - p(y_i|\mathbf{x}_i, \beta_0, \boldsymbol{\beta}))y_i \\
\boldsymbol{\beta}^{\text{new}} &= \boldsymbol{\beta} + s \sum_i (1 - p(y_i|\mathbf{x}_i, \beta_0, \boldsymbol{\beta}))y_i x_i
\end{aligned}$$

s is a small step size (learning rate) chosen in an adhoc manner (1e-1,1e-2,1e-3).

Too large a step results in worse likelihood.
Too small a step results in slow algorithm.

Instead of a fixed step size, we can use *backtracking*.

Backtracking idea

We are given objective $f(\mathbf{x})$, a current set of parameters $\mathbf{x}$ and a direction $\mathbf{g}$ (e.g. $\mathbf{g} = \nabla f(\mathbf{x})$).

$$\mathbf{x}^{\text{new}} = \mathbf{x} + s\mathbf{g}$$

but we do not know a good step size s

You can try a schedule of step sizes, for example

$$1, 0.95, 0.95^2, \ldots 0.95^n$$

until you find one for which $f(\mathbf{x} + s\mathbf{g})$ is better than $f(\mathbf{x})$.

Backtracking because it looks like you are backtracking from your first ambitious step.

Gradient ascent method with backtracking

- Start at some $\beta_0, \boldsymbol{\beta}$
- While LL is sufficienty changing repeat
  1. compute $\nabla \mathrm{LL}(\beta_0, \boldsymbol{\beta})$
  2. use backtracking along direction $\nabla \mathrm{LL}(\beta_0, \boldsymbol{\beta})$ to get s
  3. $$\begin{bmatrix} \beta_0 \\ \boldsymbol{\beta} \end{bmatrix} = \begin{bmatrix} \beta_0 \\ \boldsymbol{\beta} \end{bmatrix} + s \nabla \mathrm{LL}(\beta_0, \boldsymbol{\beta})$$

Taylor – single variable

Given a function $f : \mathbf{R} \to \mathbf{R}$ and its derivatives $f', f'', f^{(3)}...$ evaluated at x_0 we can approximate $f(x_0 + d)$ with

$$f(x_0) + f'(x_0)d + \frac{1}{2!}f''(x_0)d^2 + \frac{1}{3!}f^{(3)}(x_0)d^3 + ... \qquad (1)$$

Frame of mind: the expresion (1) is a polynomial in d

If d is small then $d^2, d^3, ...$ drop off quickly, further $\frac{1}{n!}$ also drops off quickly so the contribution from the higher order derivatives is scaled down.

Taylor – single variable

$$f(x_0) + f'(x_0)d + \frac{1}{2!}f''(x_0)d^2 + \underbrace{\frac{1}{3!}f^{(3)}(x_0)d^3 + \ldots}_{\text{higher order terms}} \qquad (2)$$

In general you can find some interval $[-d_{\max}, d_{\max}]$ such that the contribution from higher order terms is smaller in absolute value than some preset ϵ.

$$\left|\frac{1}{3!}f^{(3)}(x_0)d^3 + \frac{1}{4!}f^{(4)}(x_0)d^4 + \ldots\right| \leq \epsilon$$

Taylor – single variable

So for a given x_0 and a $d \in [-d_{\max}, d_{\max}]$ you can define a quadratic polynomial in d

$$q(d) = f(x_0) + f'(x_0)d + \frac{1}{2!}f''(x_0)d^2$$

and guarantee that

$$|f(x_0 + d) - q(d)| < \epsilon$$

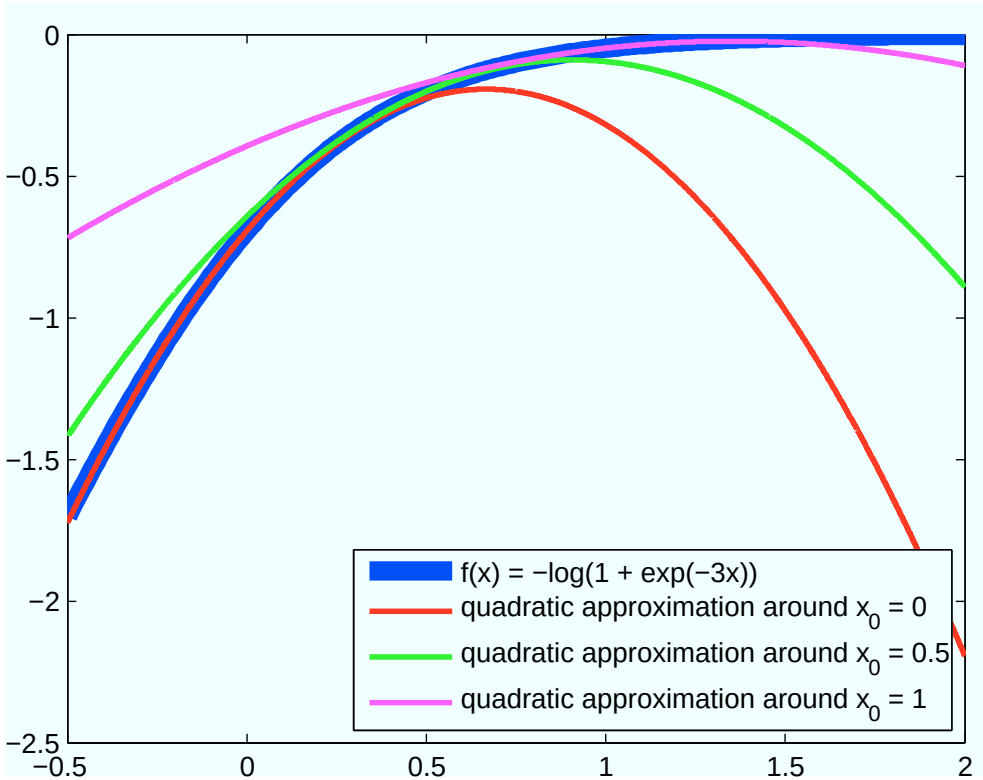
Upshot: If we are willing to accept the ϵ error and restrict d we can work with the quadratic polynomial $q(d)$ instead of the possibly hairy $f(x_0 + d)$

Fun with quadratics

A quadratic approximation of $f(x)$ around x_0

$$q_{x_0}(d) = f(x_0) + f'(x_0)d + \frac{1}{2!}f''(x_0)d^2$$

I am using notation q_{x_0} to emphasise that this is a quadratic approximation around x_0, because ...



Fun with quadratics

We know how to construct quadratic approximations of functions around a given x_0.

We can use them as proxies for original function in some region $[x_0 - d_{\max}, x_0 + d_{\max}]$

Putting quadratic approximation to use

Well then, let's find its maximizer. This is the point where $q'_{x_0}(d) = 0$ and since

$$q'_{x_0}(d) = f'(x_0) + f''(x_0)d$$

we can set

$$d^* = \operatorname*{argmax}_d q_{x_0}(d) = -\frac{f'(x_0)}{f''(x_0)}$$

This would take us to the optimum of our quadratic approximation.

Putting quadratic approximation to use

$$d^* = \operatorname*{argmax}_d q_{x_0}(d) = -\frac{f'(x_0)}{f''(x_0)}$$

If $d^* \in [-d_{\max}, d_{\max}]$ we can suggest a reasonable guess of the maximizer of f

$$x_0 + d^*$$

We can say that $f(x_0 + d^*)$ is within ϵ of local maximum of $f(x)$.

Putting quadratic approximation to use

We can also take any d between 0 and d^* and guarantee that $q(d) \geq q(0)$. This is a step towards the optimum.

Further, we can compute the improvement $q(d) - q(0)$ under our quadratic approximation and compare to $f(x_0 + d) - f(x_0)$

Remember, as we shrink d these two improvements will converge.

You should be reminded of the backtracking: We shrink d until our approximation is satisfactory.

Demo of Newton on Regularized Logistic Regression cost (1D)

The optimization problem:

$$\underset{x \in \mathbf{R}^n}{\text{maximize}} \quad -\log\{1 + \exp\{-3x\}\} - (1/2)x^2$$

We will start from $x_0 = 2$, approximate the objective with a quadratic and choose a step in maximizing direction such that our approximation is within $\epsilon = 0.01$ of the objective.

Questions?

Generalization to higher dimensional problems

We can still use Taylor expansion:

$$f(\mathbf{x}_0 + \mathbf{d}) = f(\mathbf{x}_0) + \mathbf{d}^T \nabla f(\mathbf{x}_0) + \frac{1}{2}\mathbf{d}^T \nabla^2 f(\mathbf{x}_0)\mathbf{d} + ...$$

where gradient

$$\nabla f(\mathbf{x}_0) = \begin{bmatrix} \frac{\partial}{\partial \mathbf{x}_1} f(\mathbf{x}_0) \\ \dots \\ \frac{\partial}{\partial \mathbf{x}_n} f(\mathbf{x}_0) \end{bmatrix}$$

and Hessian

$$\nabla\nabla f(\mathbf{x}_0) = \begin{bmatrix} \frac{\partial^2}{\partial^2 \mathbf{x}_1} f(\mathbf{x}_0) & \frac{\partial^2}{\partial \mathbf{x}_1 \partial \mathbf{x}_2} f(\mathbf{x}_0) & \dots & \frac{\partial^2}{\partial \mathbf{x}_1 \partial \mathbf{x}_n} f(\mathbf{x}_0) \\ \frac{\partial^2}{\partial \mathbf{x}_2 \partial \mathbf{x}_1} f(\mathbf{x}_0) & \frac{\partial^2}{\partial^2 \mathbf{x}_2} f(\mathbf{x}_0) & \dots & \frac{\partial^2}{\partial \mathbf{x}_2 \partial \mathbf{x}_n} f(\mathbf{x}_0) \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2}{\partial \mathbf{x}_n \partial \mathbf{x}_1} f(\mathbf{x}_0) & \frac{\partial^2}{\partial \mathbf{x}_n \partial \mathbf{x}_2} f(\mathbf{x}_0) & \dots & \frac{\partial^2}{\partial^2 \mathbf{x}_n} f(\mathbf{x}_0) \end{bmatrix}$$

Hessian bowls

One interesting classification of matrices revolves around their definiteness.

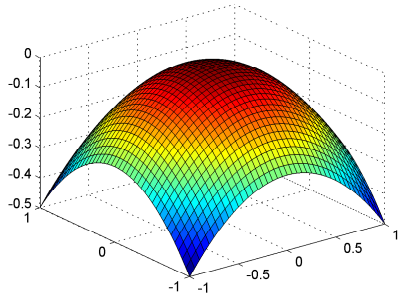
Matrix $\mathbf{A} \in \mathbf{R}^{n \times n}$ is

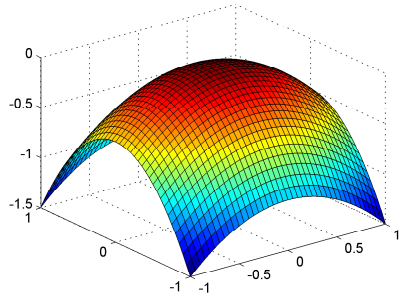
- positive semi definite (psd) if $\mathbf{x}^T \mathbf{A}\mathbf{x} \geq 0$ for all $\mathbf{x} \in \mathbf{R}^n$
- negative semi definite (nsd) if $\mathbf{x}^T \mathbf{A}\mathbf{x} \leq 0$ for all $\mathbf{x} \in \mathbf{R}^n$

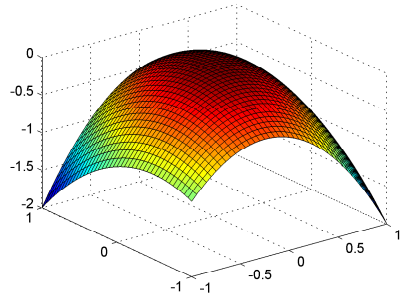
This is relevant to us in the context of Hessian matrices because positive (negative) definite Hessian matrices guarantee convexity (concavity) of objective.

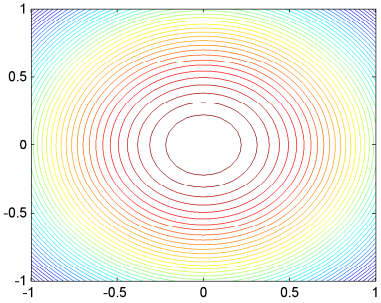
Examples of functions with negative semi definite Hessian

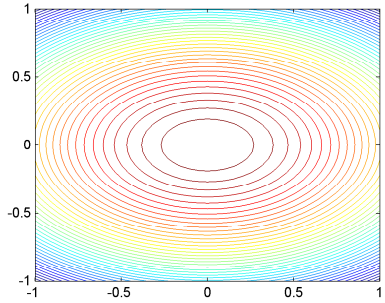
In all examples below $f(\mathbf{x}) = (1/2)\mathbf{x}^T H\mathbf{x}$

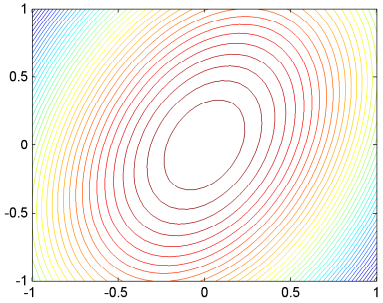








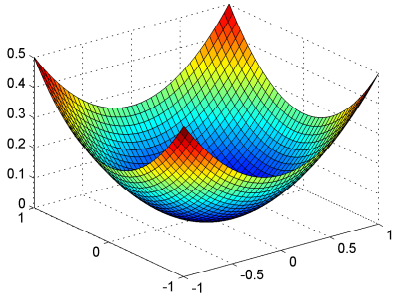


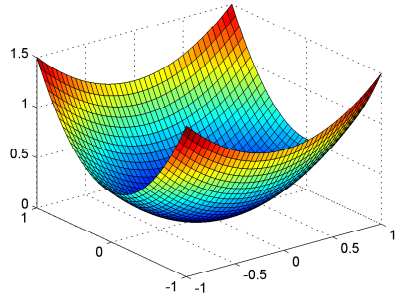


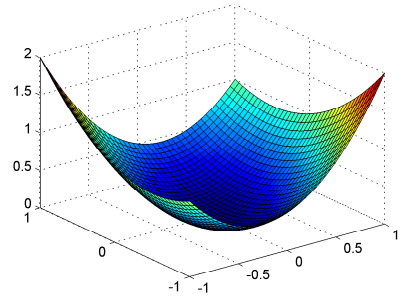
Hessian $\begin{bmatrix} -0.5 & 0 \\ 0 & -0.5 \end{bmatrix}$ $\begin{bmatrix} -1 & 0 \\ 0 & -2 \end{bmatrix}$ $\begin{bmatrix} -2 & 0.5 \\ 0.5 & -1 \end{bmatrix}$

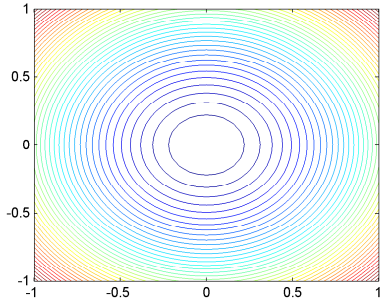
Examples of functions with positive semi definite Hessian

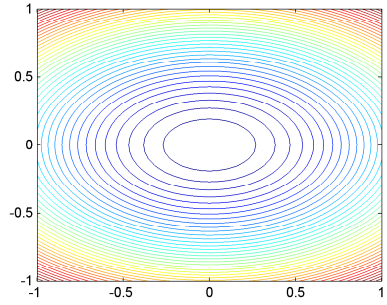
In all examples below $f(\mathbf{x}) = (1/2)\mathbf{x}^T\mathbf{H}\mathbf{x}$

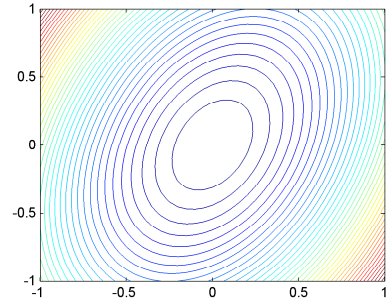












Hessian $\begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix}$ $\begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix}$ $\begin{bmatrix} 2 & -0.5 \\ -0.5 & 1 \end{bmatrix}$

Question

We know how to numerically check if our analytically derived gradients are right – finite differences.

How do we check an analytically derived Hessian?

Back to Taylor and higher dimensional problems

As we did in case of 1D optimization, we can adopt a quadratic approximation of f around some $\mathbf{x}_0$

$$q_{\mathbf{x}_0}(\mathbf{d}) = f(\mathbf{x}_0) + \mathbf{d}^T \nabla f(\mathbf{x}_0) + \frac{1}{2}\mathbf{d}^T \nabla^2 f(\mathbf{x}_0)\mathbf{d}$$

and optimize this function instead of f

Newton again

Optimum of $q_{\mathbf{x}_0}(d)$ is achieved at

$$\mathbf{d}^* = \operatorname*{argmax}_{\mathbf{d}} q_{\mathbf{x}_0}(\mathbf{d}) = -(\nabla^2 f(\mathbf{x}_0))^{-1} \nabla f(\mathbf{x}_0)$$

and you can compare this to the 1D version

$$d^* = \operatorname*{argmax}_{d} q_{x_0}(d) = -\frac{f'(x_0)}{f''(x_0)}$$

Step sizes

Again, as in the 1D case, quadratic function is only a reasonable approximation locally.

Usually we cannot take a full step ($s = 1$) as we may not trust the quadratic approximation that far away

$$\mathbf{x}^{\text{new}} = \mathbf{x}^{\text{old}} - s[\nabla^2 f(\mathbf{x}^{\text{old}})]^{-1} \nabla f(\mathbf{x}^{\text{old}})$$

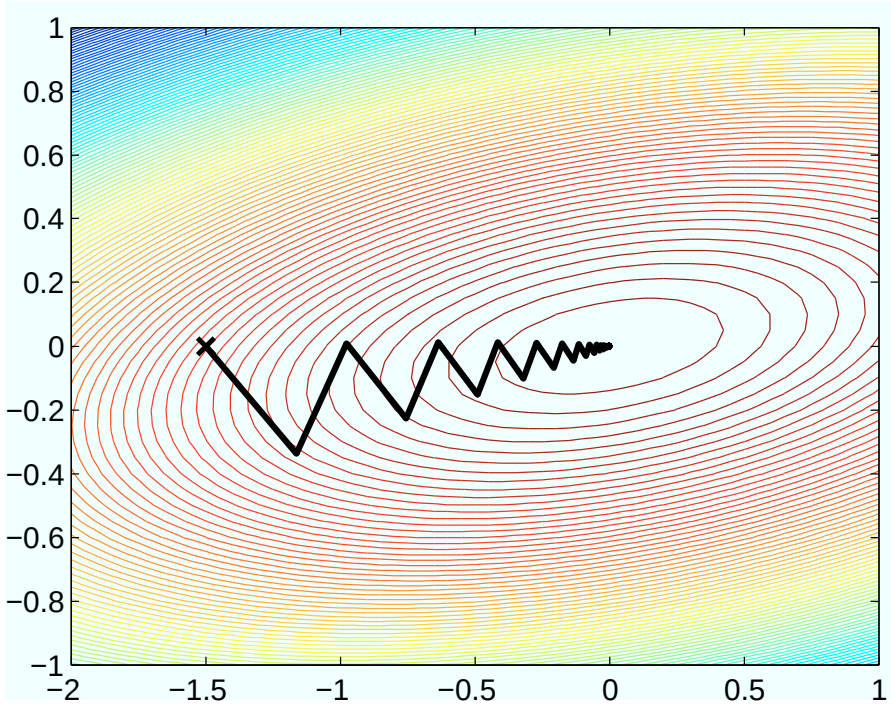
so we have to employ some sort of step-size search algorithm[2], usually a backtracking variant.

[2] also called line-search since we search along direction $(\nabla^2 f(\mathbf{x}_0))^{-1} \nabla f(\mathbf{x}_0)$

Obligatory steepest descent/ascent slide

Hessians seem like a bother, can't we just use gradients?

$$f(\mathbf{x}) = (1/2)\mathbf{x}^T \begin{bmatrix} -0.5 & 0.5 \\ 0.5 & 4 \end{bmatrix} \mathbf{x}$$



With Newton's method this problem is solved in a single step.

Standard optimization techniqus

- Steepest descent: cheap steps, but too many of them.
- Newton: expensive step, but smaller number of them

Alternatives

- BFGS (BroydenFletcherGoldfarbShanno) approximates $[\nabla^2 f]^{-1}$ (inverse of Hessian)
- Conjugate gradients avoid zig-zagging of steepest descent by avoiding undoing previous iterations

Solid implementations for both available. A nice tutorial on conjugate gradients:
`http://www.cs.cmu.edu/~quake-papers/painless-conjugate-gradient.pdf`

Also look at:
`http://www.di.ens.fr/~mschmidt/Software/minFunc.html`

We did ...

- Regression – recap
- Classification
- A mixture models
- Sigmoid
- Logistic regression
- Newton method