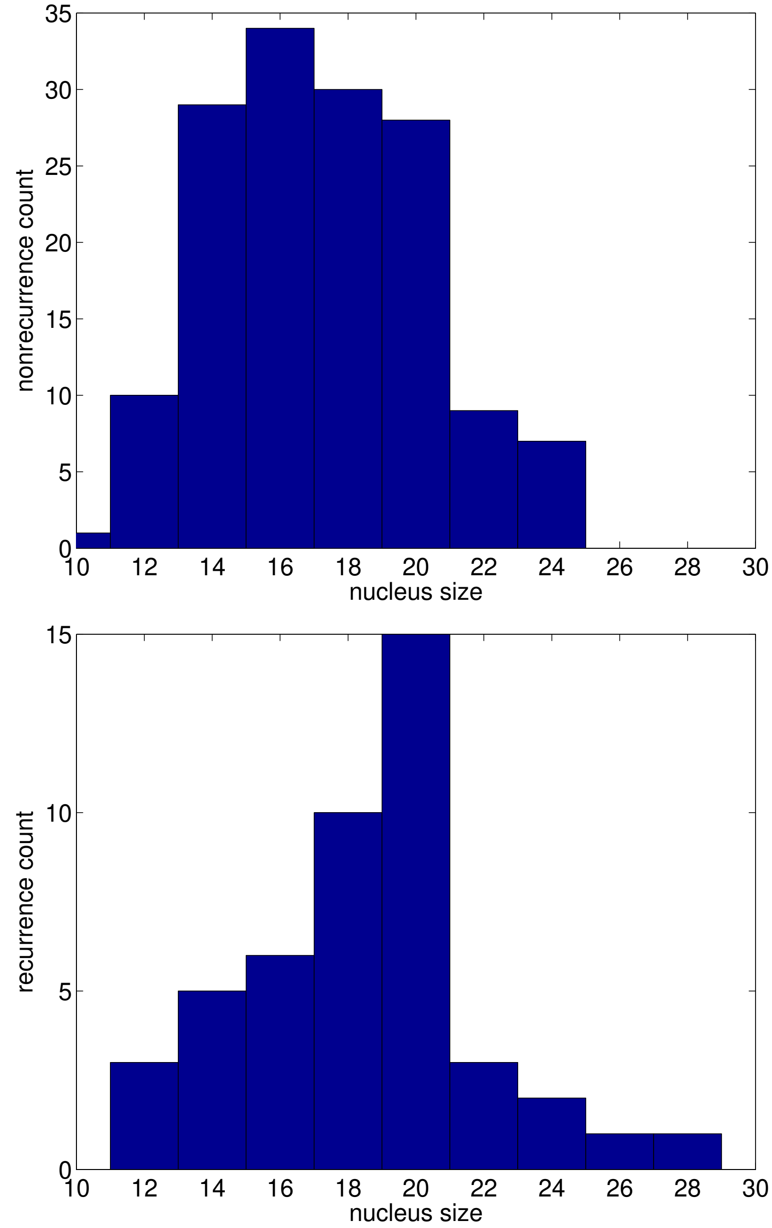
COMP 652: Machine Learning

Lecture 4

Today

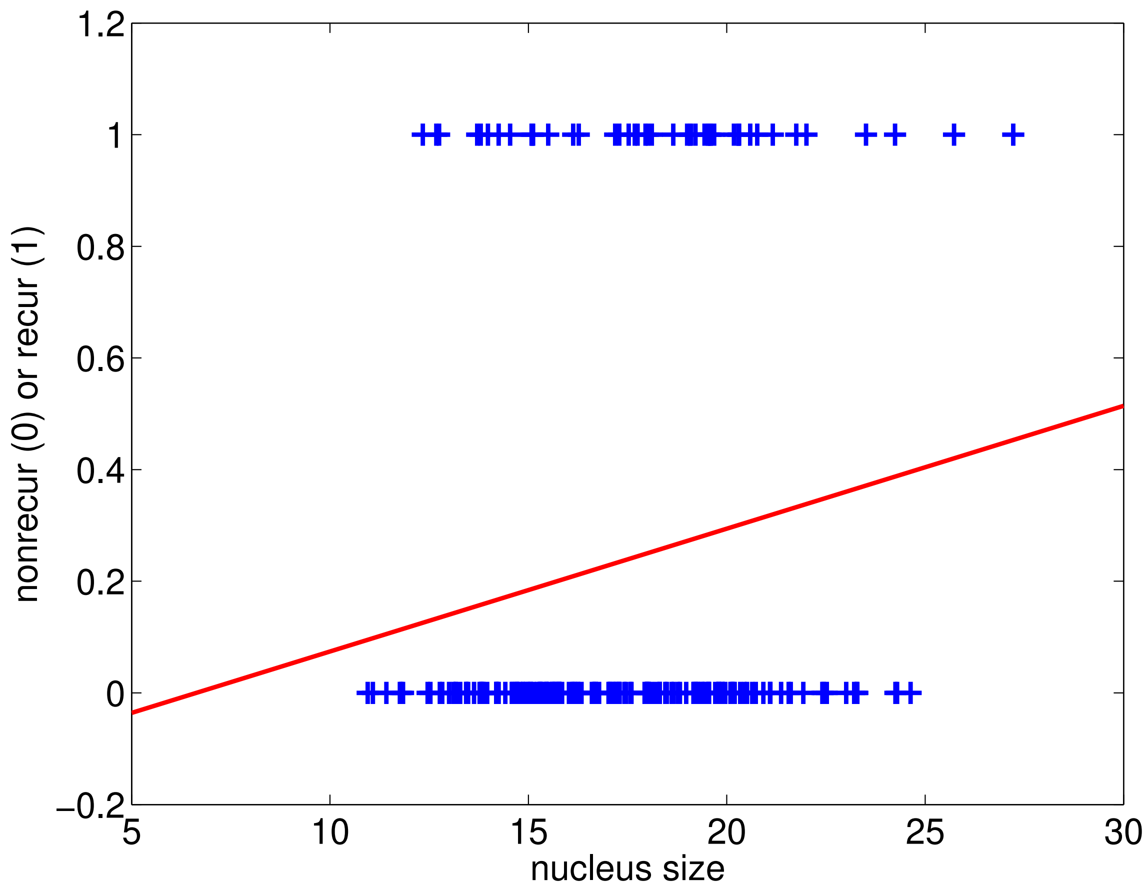
- Logistic regression
- Gradient descent

Classification problem example: Given "nucleus size" predict non/recurrrence



Solution by linear regression

- Univariate real input: nucleus size
- Output coding: non-recurrence = 0, recurrence = 1
- Sum squared error minimized



- Comments?

Comments on linear regression for classification

- The predictor shows an increasing trend towards recurrence with larger nucleus size, as expected.
- Output cannot be directly interpreted as a class prediction.
- Thresholding output (e.g., at 0.5) could be used to predict 0 or 1. (In this case, prediction would be 0 except for extremely large nucleus size.)
- Output could be interpreted as probability. (Except that probabilities above 1 and below 0 may be output.)

$\Rightarrow$ Similar interpretation concerns arise when predicting real values that must be non-negative or fall in a given range. E.g.:

- What's a negative time-to-recurrence?
- Activity of a gene ranges between zero (none) and some maximum (fully on).

The logistic function

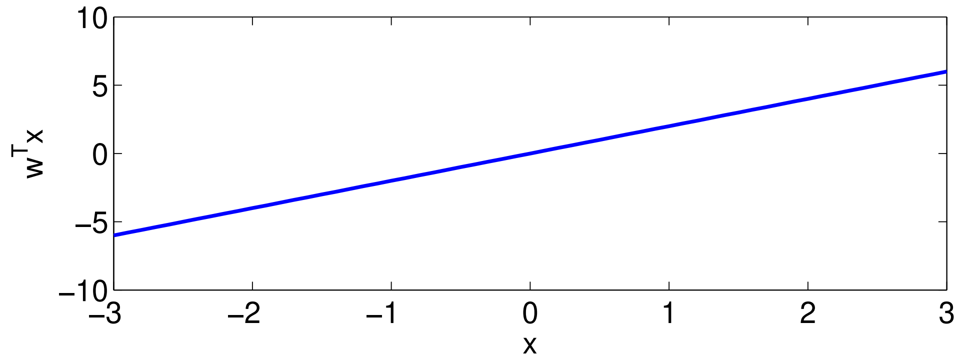
- When the output should be in the range $[0, 1]$, we sometimes turn to a "logistic" classifier or regressor:

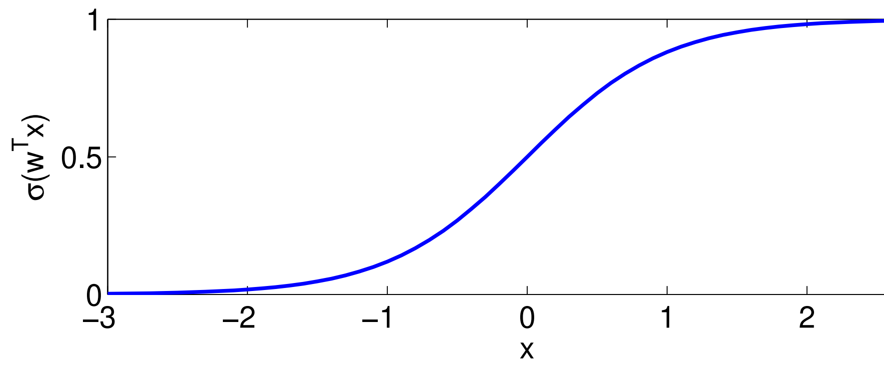
$$h_{\mathbf{w}}(\mathbf{x}) = \frac{1}{1 + e^{-\sum_{j=0}^{n} w_j x_j}}$$

- Defining $\sigma(u) = 1/(1 + e^{-u})$ as the "logistic" function, we can write:

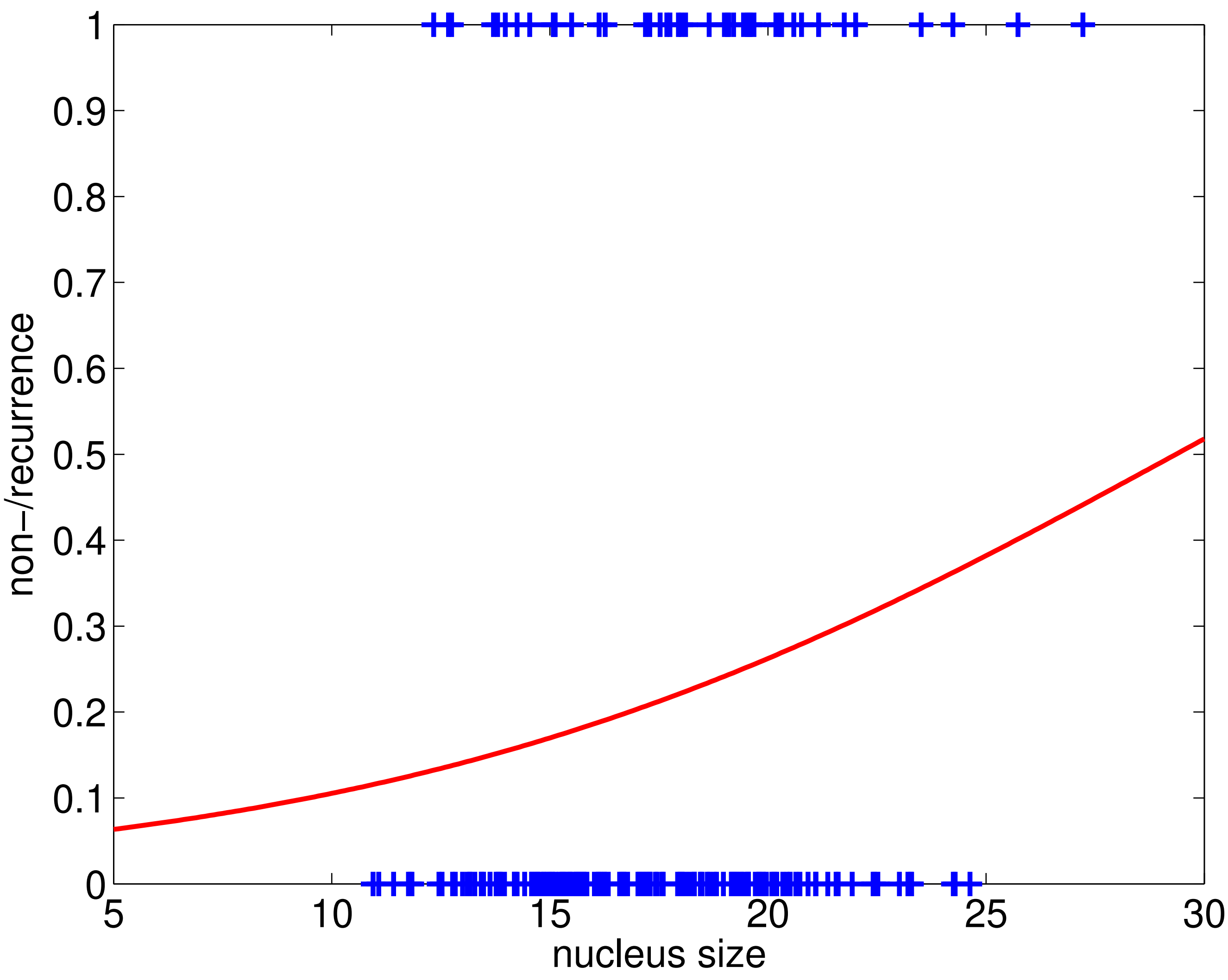
$$h_{\mathbf{w}}(\mathbf{x}) = \sigma\left(\sum_{j=0}^{n} w_j x_j\right) = \sigma(\mathbf{w}^T\mathbf{x})$$

- σ can be viewed as taking a linear predictor and "squashing" it's output between 0 and 1:





Example of a logistic fit



Error functions?

- For either regression ($\mathcal{Y} = [0, 1]$) or classification ($\mathcal{Y} = \{0, 1\}$) problems, we could minimize the sum-squared-error (SSQ):

$$J(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{m}(y_i - \sigma(\mathbf{w}^T\mathbf{x_i}))^2$$

- Question 1: For what probabilistic model of the data is this the maximum likelihood solution?
- Question 2: How to we minimize this J?

Question 1 answer

- We saw that in general, the minimizing the SSQ is equivalent to finding the maximum likelihood hypothesis under the model:

$$y_i = h(\mathbf{x_i}) + \epsilon_i$$

  where the ϵ_i are i.i.d. mean-zero Gaussians with standard deviation σ.
- In this case, we would thus have:

$$y_i = \sigma(\mathbf{w}^T \mathbf{x_i}) + \epsilon_i$$

- If all $y_i \in \{0, 1\}$, this doesn't make sense. Gaussian perturbations (noise) would likely result in real-valued, not-binary, y_i.
- If all $y_i \in [0, 1]$, it might make sense. We'd likely expect to see values above or below the range $[0, 1]$ though.

Question 2 answer?

Question 2 answer

- At a minimum, the gradient must be zero...

First, let's look at $\sigma(u)$

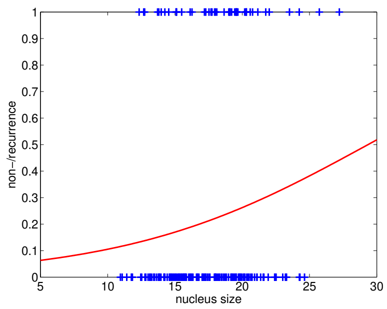
$$
\begin{aligned}
\frac{d}{du}\sigma(u) &= \frac{d}{du}\frac{1}{1+e^{-u}} \\
&= \frac{-1}{(1+e^{-u})^2}\frac{d}{du}(1+e^{-u}) \\
&= \frac{-1}{(1+e^{-u})^2}(-e^{-u}) \\
&= \frac{1}{1+e^{-u}}\frac{e^{-u}}{1+e^{-u}} \\
&= \frac{1}{1+e^{-u}}\frac{e^{-u}+1-1}{1+e^{-u}} \\
&= \frac{1}{1+e^{-u}}\left(1-\frac{1}{1+e^{-u}}\right) \\
&= \sigma(u)(1-\sigma(u))
\end{aligned}
$$

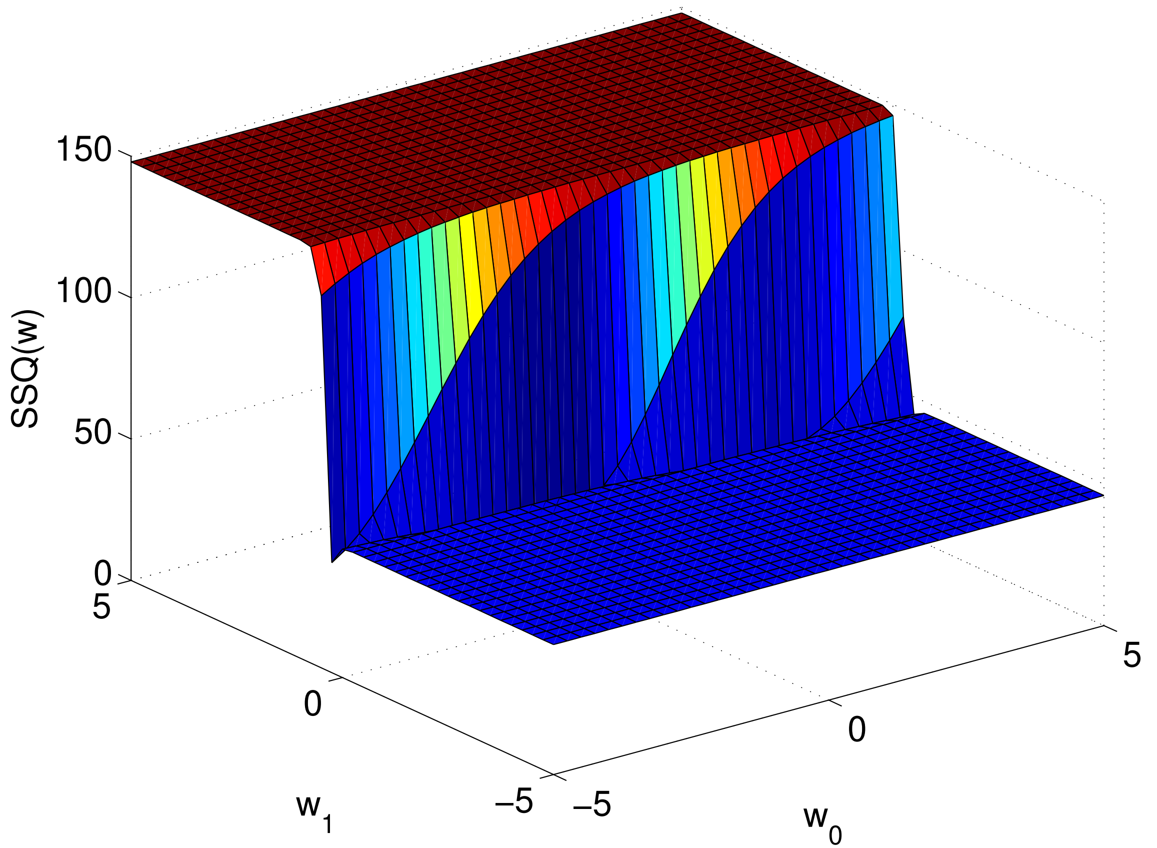
Now, let's look at $J(\mathbf{w})$

$$\begin{aligned}
\frac{\partial}{\partial w_k} J(\mathbf{w}) &= \frac{\partial}{\partial w_k} \frac{1}{2} \sum_{i=1}^{m} (\sigma(\mathbf{w}^T \mathbf{x_i}) - y_i)^2 \\
&= \sum_{i=1}^{m} (\sigma(\mathbf{w}^T \mathbf{x_i}) - y_i) \frac{\partial}{\partial w_k} (\sigma(\mathbf{w}^T \mathbf{x_i}) - y_i) \\
&= \sum_{i=1}^{m} (\sigma(\mathbf{w}^T \mathbf{x_i}) - y_i) \sigma(\mathbf{w}^T \mathbf{x_i})(1 - \sigma(\mathbf{w}^T \mathbf{x_i})) \frac{\partial}{\partial w_k} \mathbf{w}^T \mathbf{x_i} \\
&= \sum_{i=1}^{m} (\sigma(\mathbf{w}^T \mathbf{x_i}) - y_i) \sigma(\mathbf{w}^T \mathbf{x_i})(1 - \sigma(\mathbf{w}^T \mathbf{x_i})) x_{i,k}
\end{aligned}$$

- Thus, $\nabla_{\mathbf{w}} J = \sum_{i=1}^{m} (\sigma(\mathbf{w}^T \mathbf{x_i}) - y_i) \sigma(\mathbf{w}^T \mathbf{x_i})(1 - \sigma(\mathbf{w}^T \mathbf{x_i})) \mathbf{x_i}$.
- Setting $\nabla_{\mathbf{w}} J = 0$ results in a *nonlinear* system of $n+1$ equations in $n+1$ unknowns.
- Gradient descent is often used to solve. But beware: there may be many local optima!

Example error surface





An alternative error function for binary classification

- When output of a hypothesis $h(\mathbf{x_i})$ is interpreted as the probability that $y_i = 1$, we can use the principle of maximum likelihood to derive an error function.

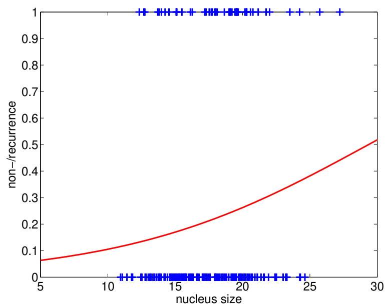
The cross-entropy error function

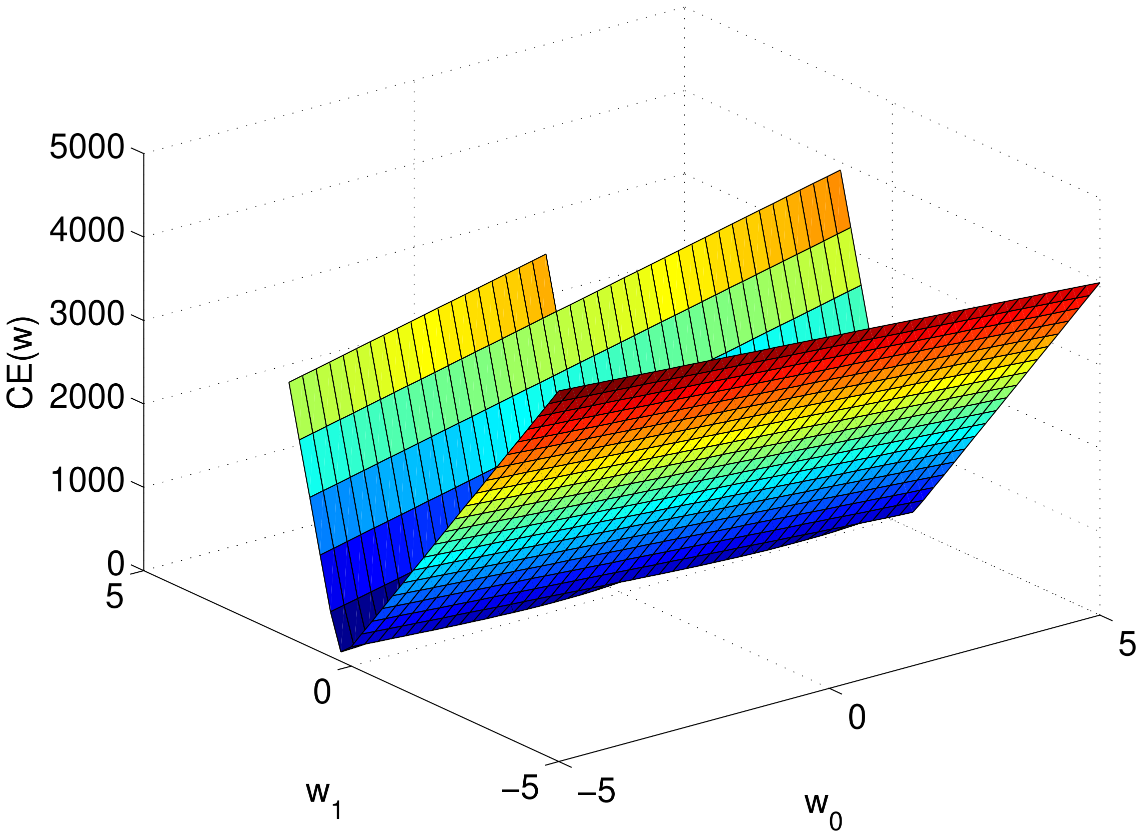
$$\begin{aligned}
l(h) &= P(Y|X, h) \\
&= \Pi_{i=1}^{m} P(y_i|\mathbf{x_i}, h) \\
&= \Pi_{i=1}^{m} \begin{cases} h(\mathbf{x_i}) & \text{if } y_i = 1 \\ 1 - h(\mathbf{x_i}) & \text{if } y_i = 0 \end{cases}
\end{aligned}$$

$$\begin{aligned}
-\log l(h) &= -\sum_{i=1}^{m} \log \begin{cases} h(\mathbf{x_i}) & \text{if } y_i = 1 \\ 1 - h(\mathbf{x_i}) & \text{if } y_i = 0 \end{cases} \\
&= -\sum_{i=1}^{m} y_i \log h(\mathbf{x_i}) + (1 - y_i) \log(1 - h(\mathbf{x}_i))
\end{aligned}$$

- The max likelihood hypothesis minimizes the cross-entropy error function: $-\sum_{i=1}^{m} y_i \log h(\mathbf{x_i}) + (1 - y_i) \log(1 - h(\mathbf{x}_i))$
- For logistic classification, this comes to: $-\sum_{i=1}^{m} y_i \log \sigma(\mathbf{w}^T \mathbf{x_i}) + (1 - y_i) \log(1 - \sigma(\mathbf{w}^T \mathbf{x_i}))$

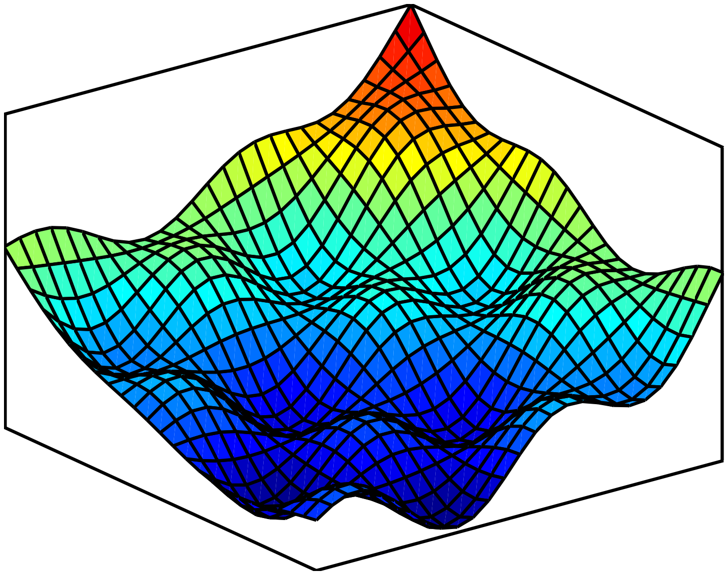
Cross-entropy error surface





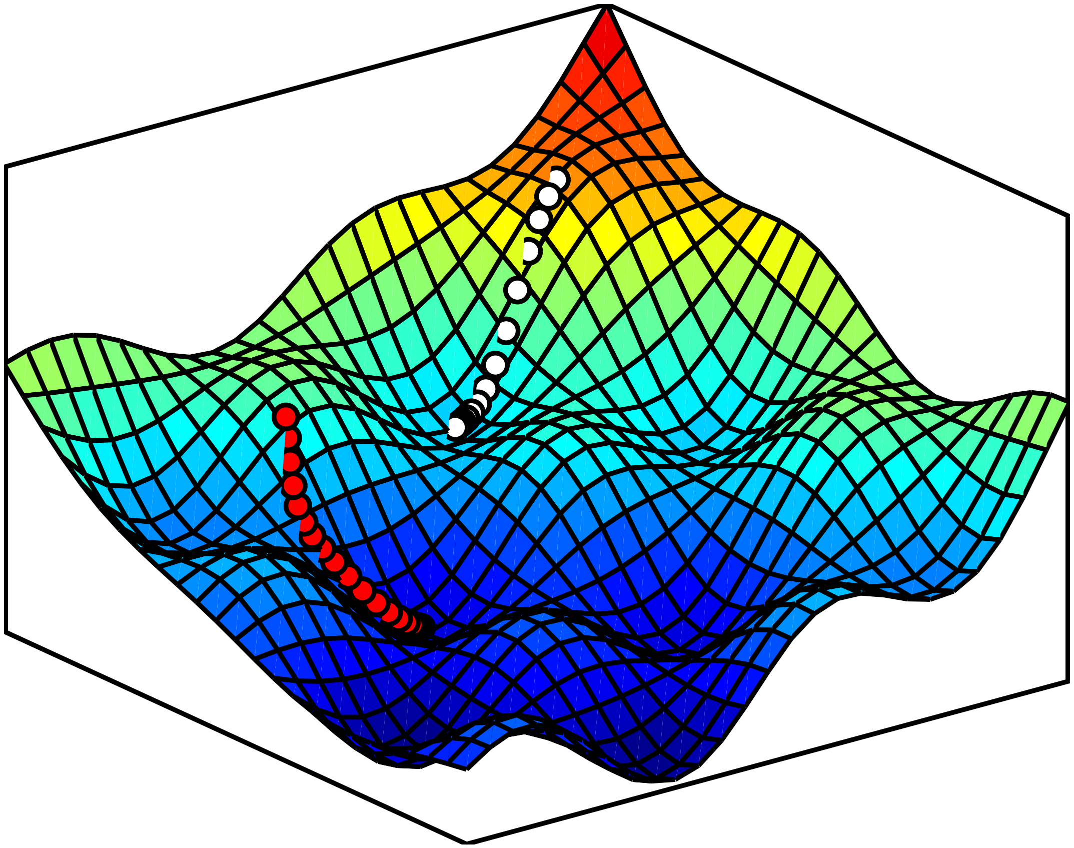
Gradient descent

- The gradient of f at a point $\langle u_1, u_2, \ldots, u_n \rangle$ can be thought of as a vector indicating which way is "uphill".



- If this is an error function, we want to move "downhill" on it, i.e., in the direction opposite to the gradient

Example gradient descent traces



Gradient descent algorithm

- The basic algorithm assumes that ∇f is easily computed
- We want to produce a sequence of vectors $\mathbf{u^1}, \mathbf{u^2}, \mathbf{u^3}, \ldots$ with the goal that:
  - $f(\mathbf{u^1}) > f(\mathbf{u^2}) > f(\mathbf{u^3}) > \ldots$
  - $\lim_{i \to \infty} \mathbf{u^i} = \mathbf{u}$ and $\mathbf{u}$ is locally optimal.
- The algorithm: Given $\mathbf{u^0}$, do for $i = 0, 1, 2, \ldots$

$$\mathbf{u^{i+1}} = \mathbf{u^i} - \alpha_i \nabla f(\mathbf{u^i}) \ ,$$

  where $\alpha_i > 0$ is the *step size* or *learning rate* for iteration i.

Stepsizes and convergence

- Convergence to a local minimum depends in part on the α_i.
- If they are too large (such as constant) oscillation or "bubbling" may occur.
  (This suggests the α_i should tend to zero as $i \to \infty$.)
- If they are too small, the $\mathbf{u^i}$ may not move far enough to reach a local minimum, or may do so very slowly.

Robbins-Monroe conditions

- The α_i are a Robbins-Monroe sequence if:
  - $\sum_{i=0}^{\infty} \alpha_i = +\infty$
  - $\sum_{i=0}^{\infty} \alpha_i^2 < \infty$
- E.g., $\alpha_i = \frac{1}{i+1}$ (averaging)
- E.g., $\alpha_i = \frac{1}{2}$ for $i = 1 \ldots T$, $\alpha_i = \frac{1}{2^2}$ for $i = T+1, \ldots (T+1)+2T$ etc
- These conditions, along with appropriate conditions on f are sufficient to ensure convergence of the $\mathbf{u^i}$ to a point $\mathbf{u}^\infty$ such that $\nabla f(\mathbf{u}^\infty) = 0$.
- Many variants are possible: e.g., we may use at each step *a random vector* with mean $\nabla f(\mathbf{u^i})$; this is *stochastic gradient descent*.

“Batch” versus “On-line” optimization

- Often in machine learning the error function, J, is a sum of errors attributed to each instance: $(J = J_1 + J_2 + \ldots + J_m.)$
- In *batch gradient descent*, the true gradient is computed at each step:

$$\nabla J = \nabla J_1 + \nabla J_2 + \ldots \nabla J_m.$$

- In *on-line gradient descent*, at each iteration one instance, $i \in \{1, \ldots, m\}$, is chosen at random and only ∇J_i is used in the update.
- Why prefer one or the other?

"Batch" versus "On-line" optimization

- Batch is simple, repeatable.
- On-line:
  - Requires less computation per step.
  - Randomization may help escape poor local minima.
  - Allows working with a stream of data, rather than a static set (hence "on-line").

Termination

There are many heuristics for deciding when to stop gradient descent.

1. Run until $\|\nabla f\|$ is smaller than some threshold.
2. Run it for as long as you can stand.
3. Run it for a short time from 100 different starting points, see which one is doing best, goto 2.
4. ...

Back to logistic regression

- Error function is $J(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{m}(y_i - h_{\mathbf{w}}(\mathbf{x_i}))^2$
- The gradient is given by:

$$\nabla h_{\mathbf{w}}(\mathbf{x_i}) = h_{\mathbf{w}}(\mathbf{x_i})(1 - h_{\mathbf{w}}(\mathbf{x_i}))\mathbf{x_i}$$

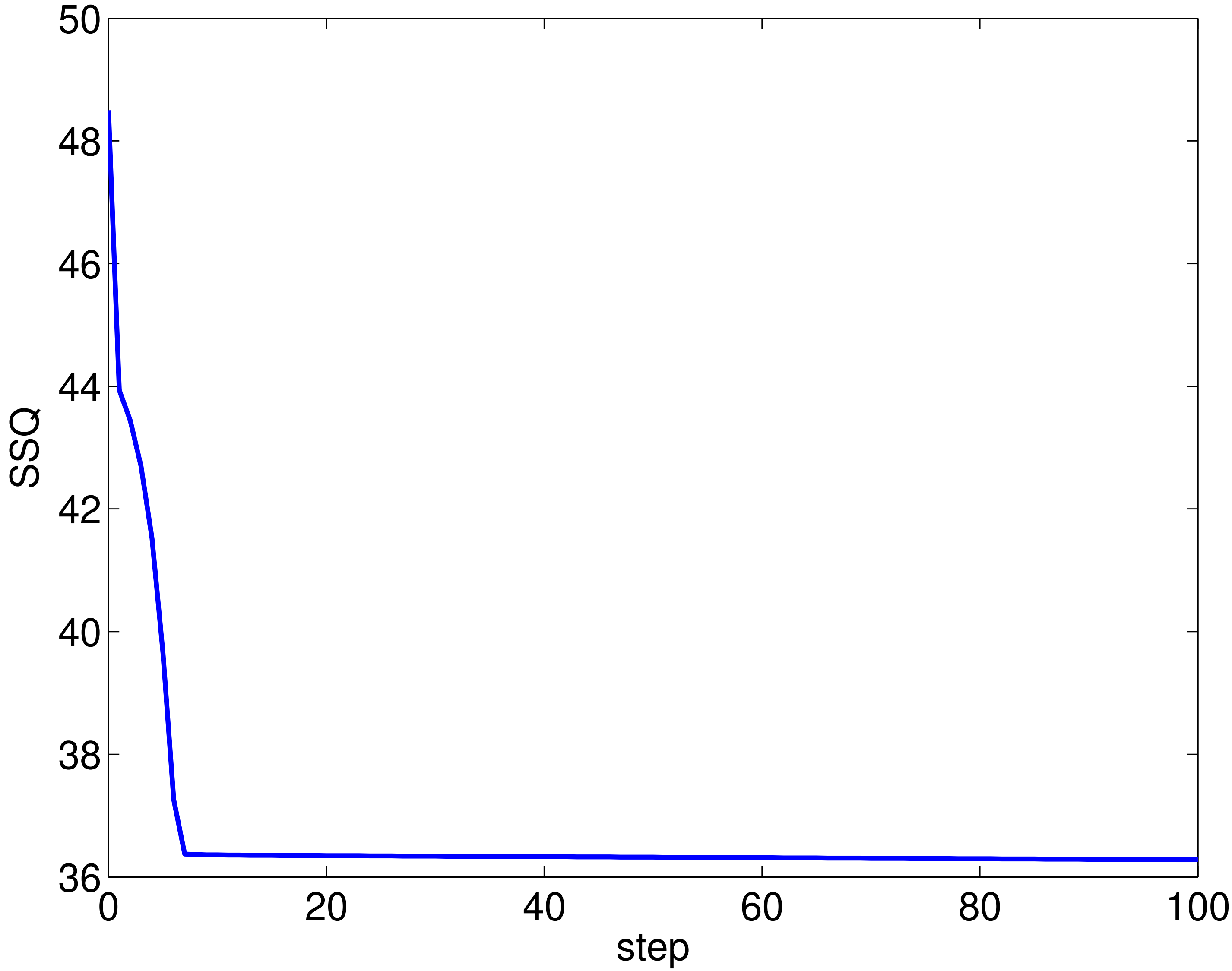
- We obtain the weight update rule:

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha \sum_i (y_i - h_{\mathbf{w}}(\mathbf{x_i}))h_{\mathbf{w}}(\mathbf{x_i})(1 - h_{\mathbf{w}}(\mathbf{x_i}))\mathbf{x_i}$$

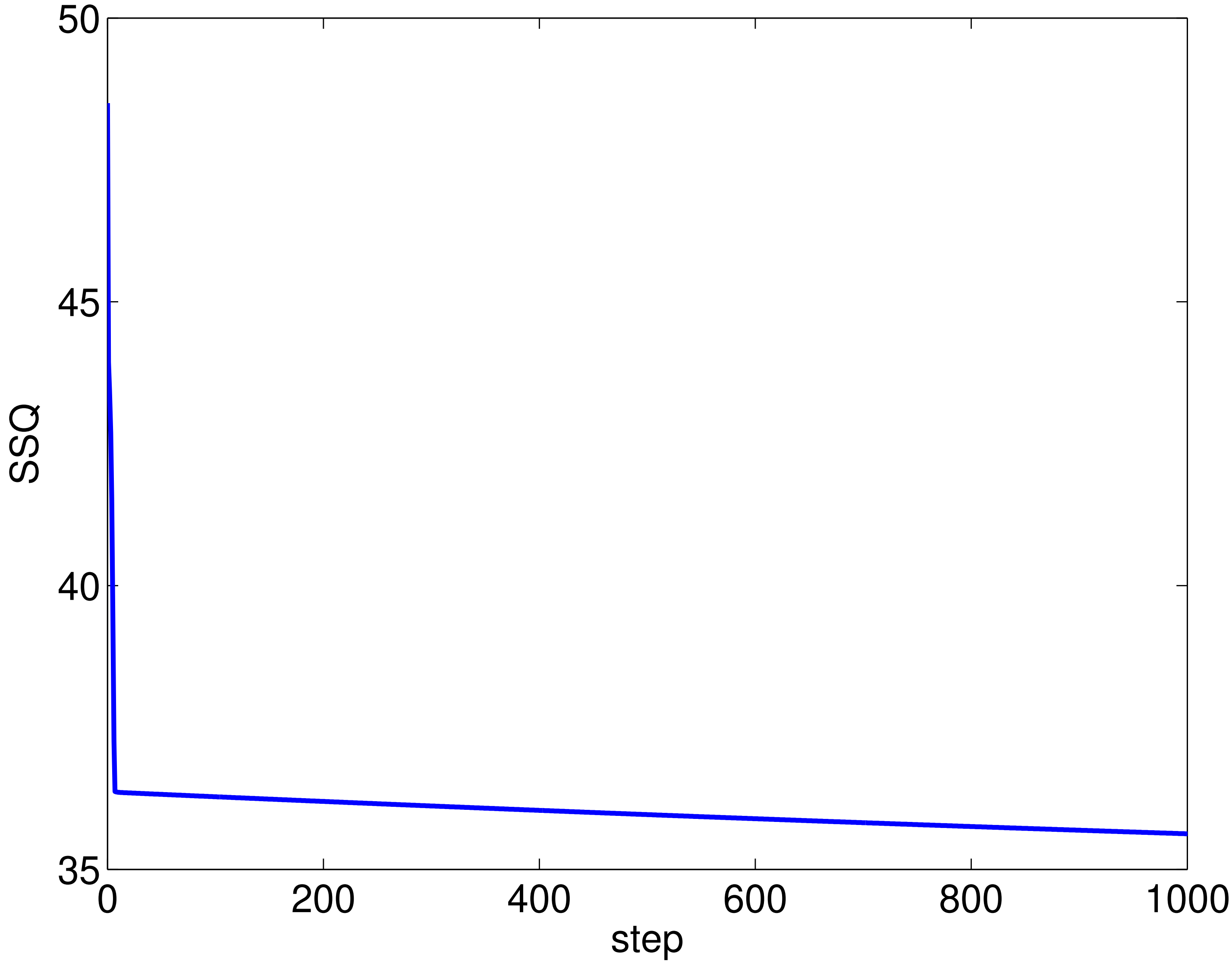
- We can again do batch or on-line updates

Show GradDesc Series 1 Slideshow

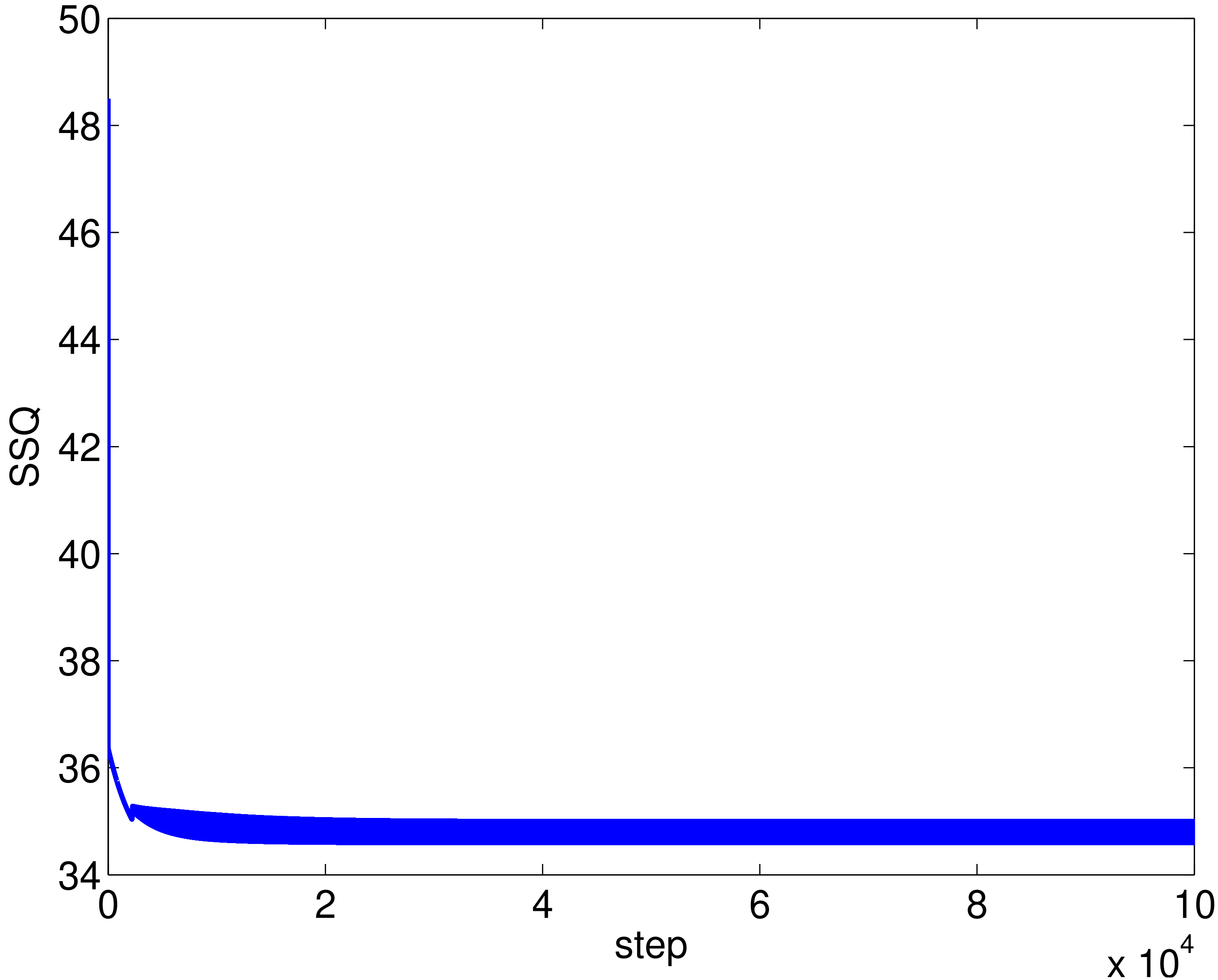
Change in SSQ during procedure



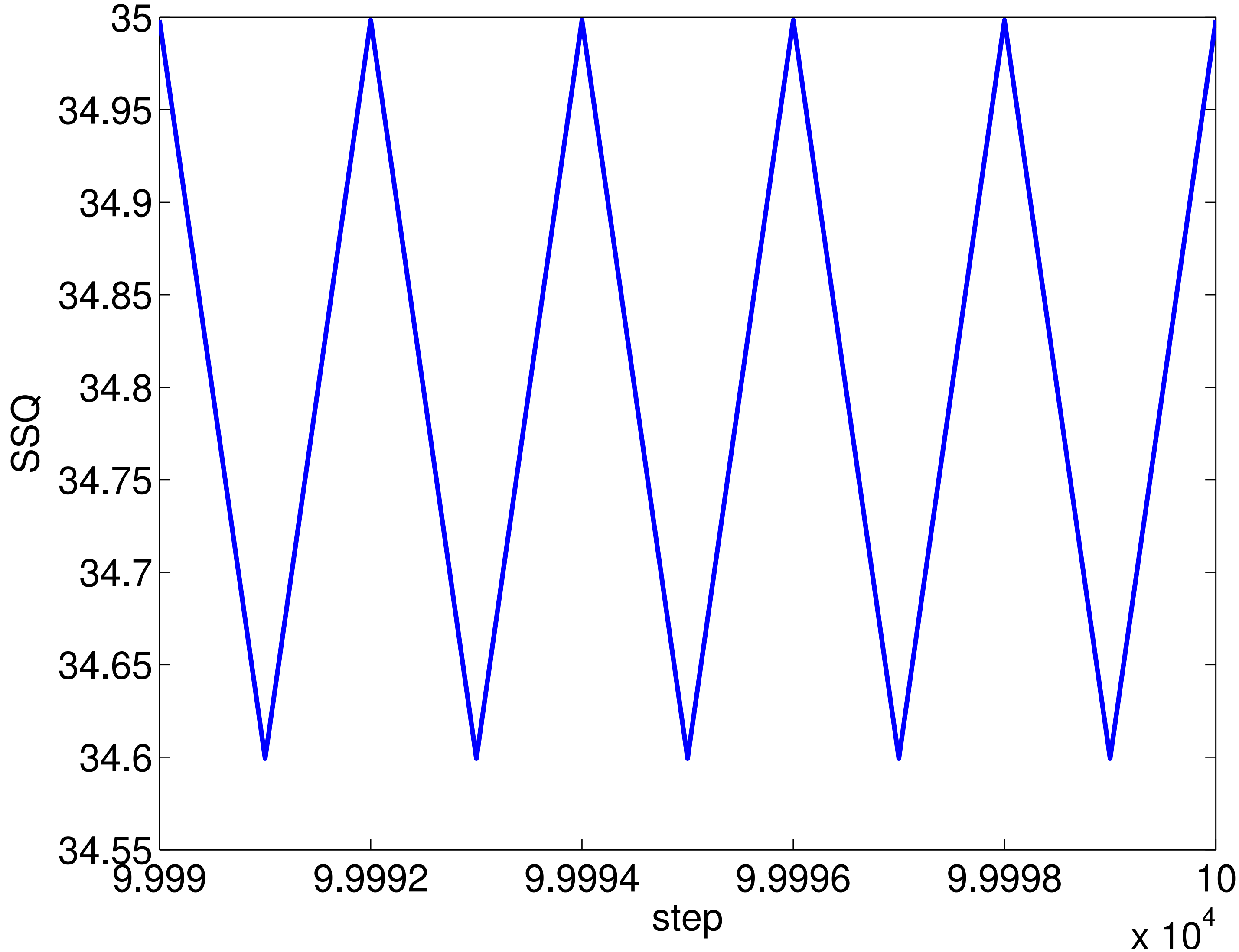
Change in SSQ during procedure



Change in SSQ during procedure



Change in SSQ during procedure



Show GradDesc Series 2 Slideshow

Summary

- Logistic regression (or classification) is suitiable when predicting a binary class probability or a real value in the range $[0, 1]$.
  - Sum-squared-error may be minimized.
  - For classification, however, cross-entropy error function is better motivated.
  - Setting gradient to zero results in nonlinear system of $n = 1$ equations in $n + 1$ unknowns.
- Gradient descent for finding local minima of complicated error surface.
  - Repeatedly adjust parameters in direction opposite of gradient.
  - If step-sizes obey Robbins-Monroe conditons (and error function satisfies some reasonable properties), convergence to local minimum guaranteed.