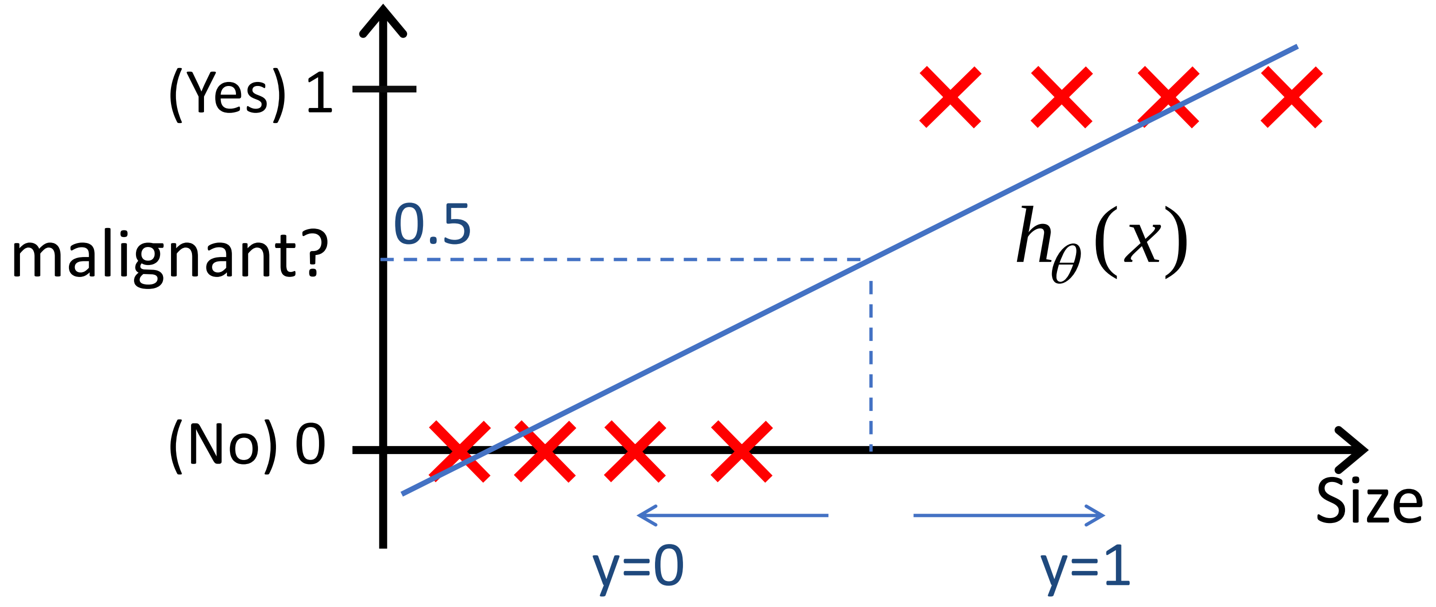
Logistic regression

Introduction

- Generative approach – crate a joint model of the form $p(y, \mathbf{x})$ and then to condition on $\mathbf{x}$, deriving $p(y|\mathbf{x})$
- Discriminative approach - fit a model of the form $p(y|\mathbf{x})$ directly
    - we will assume discriminative models which are linear in the parameters

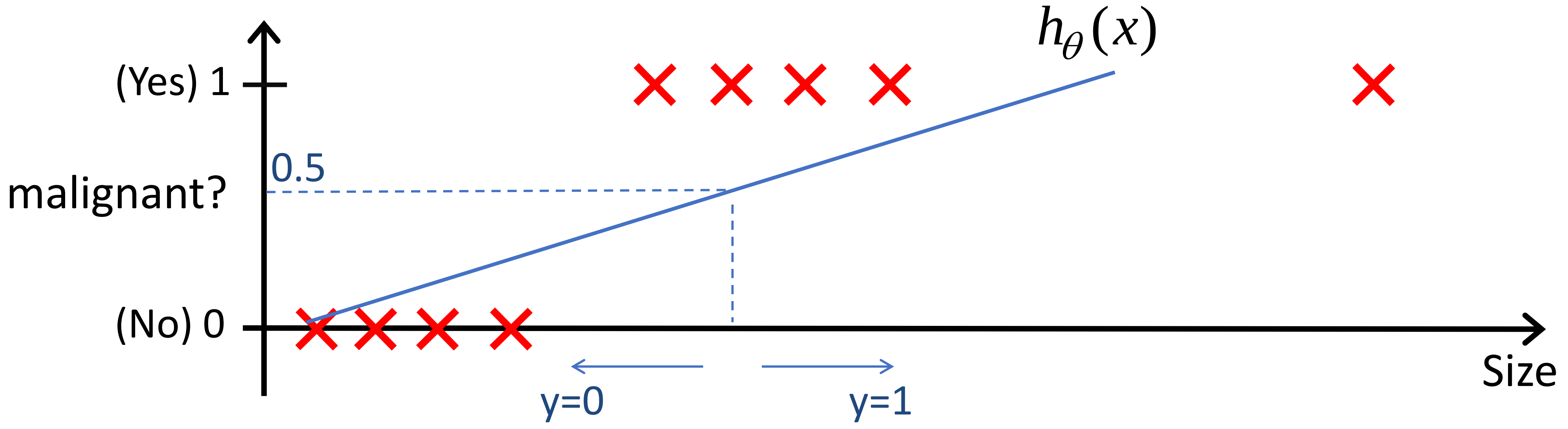


What happens if we try to apply linear regression to this problem?

We set a threshold at the output of the classifier $h_\theta(x)$ equal to 0.5:

If $h_\theta(x) \geq 0.5$, we predict "y = 1"

If $h_\theta(x) < 0.5$, we predict "y = 0"



Again we set the threshold at the output of the classifier $h_\theta(x)$ equal to 0.5:

If $h_\theta(x) \geq 0.5$, we predict "y = 1"

If $h_\theta(x) < 0.5$, we predict "y = 0"

In the previous example we were just lucky, linear regression is generally a bad classifier

Classification: y = 0 or 1

$h_\theta(x)$ can be > 1 or < 0

Logistic regression: $0 \le h_\theta(x) \le 1$

Logistic regression model

We want $0 \le h_\theta(x) \le 1$

$$z = \theta^T x$$

$$g(\theta^T x) = \frac{1}{1+e^{-\theta^T x}}$$

Old regression model:

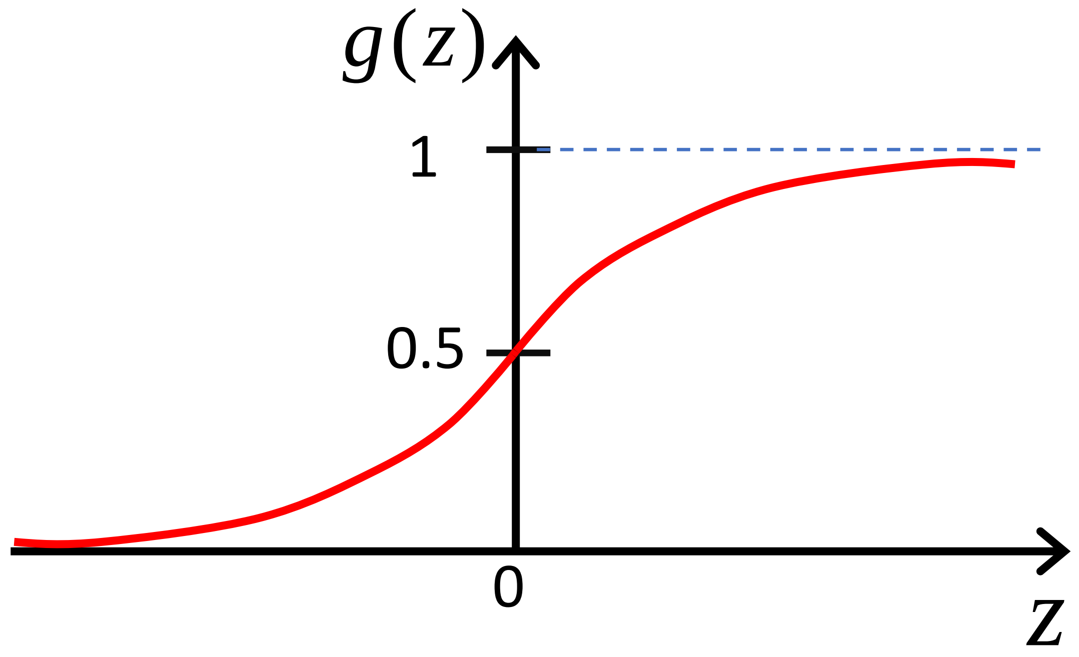
$$h_\theta(x) = \theta^T x$$

New regression model:

$$h_\theta(x) = g(\theta^T x)$$

Uses sigmoid (logistic) function

$$g(z) = \frac{1}{1+e^{-z}}$$



Interpretation of the output of the hypothesis

$h_\theta(x)$ = estimating the probability that y = 1 at a given input x

Example: If $x = \begin{bmatrix} x_0 \\ x_1 \end{bmatrix} = \begin{bmatrix} 1 \\ Size \end{bmatrix}$

And obtain $h_\theta(x) = 0.7$

We need to tell the patient that there is a 70% chance that the tumor will be malign

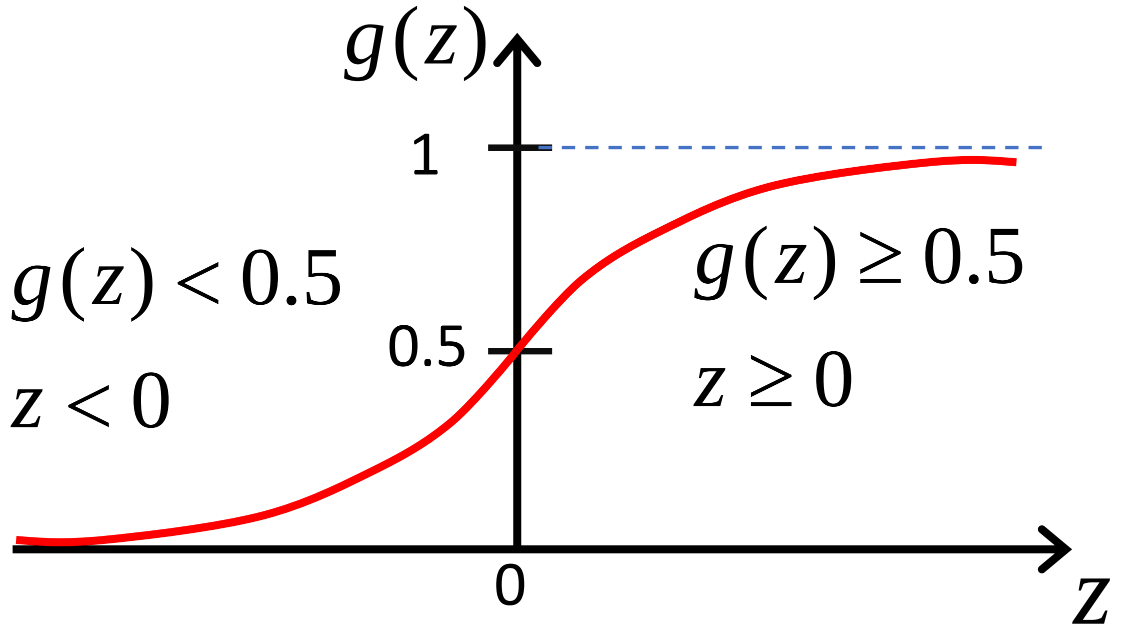
$h_\theta(x) = P(y = 1 | x; \theta)$ “probability that y = 1, for a given x, parameterized with θ ”

$$P(y = 1 | x; \theta) + P(y = 0 | x; \theta) = 1$$

Logistic regression calssification

$$h_\theta(x) = g(\theta^T x)$$

$$g(z) = \frac{1}{1+e^{-z}}$$



Predict "y=1" if $h_\theta(x) \geq 0.5$

$$g(\theta^T x) \geq 0.5$$

$$\Downarrow$$

$$\theta^T x \geq 0$$

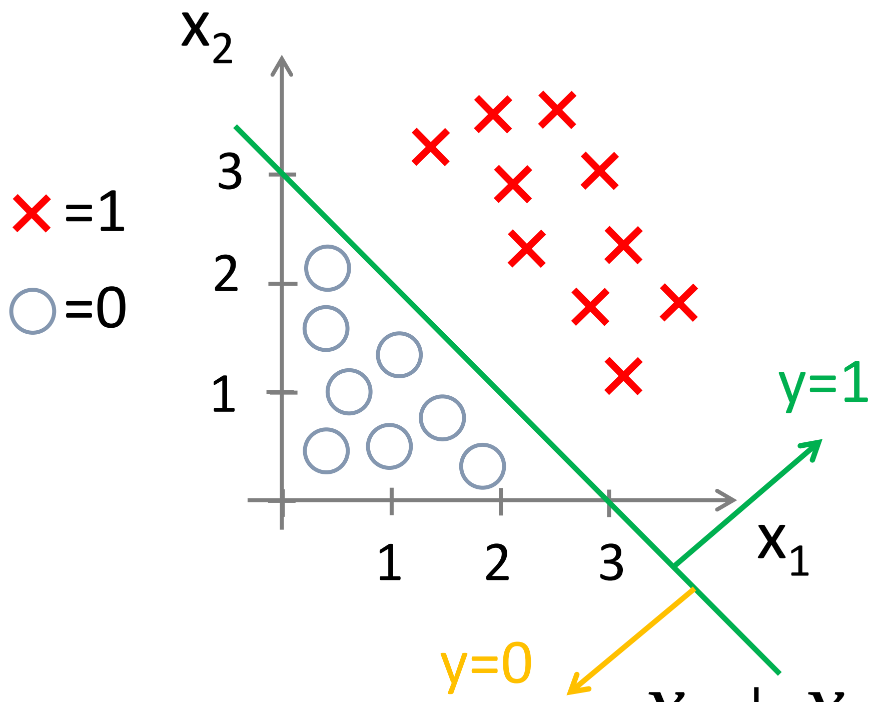
Predict "y=0" if $h_\theta(x) < 0.5$

$$g(\theta^T x) < 0.5$$

$$\Downarrow$$

$$\theta^T x < 0$$

Decision boundary



$$x_1 + x_2 = 3$$

Decision boundary (place of distinction between y = 0 and y = 1)

Hypothesis:

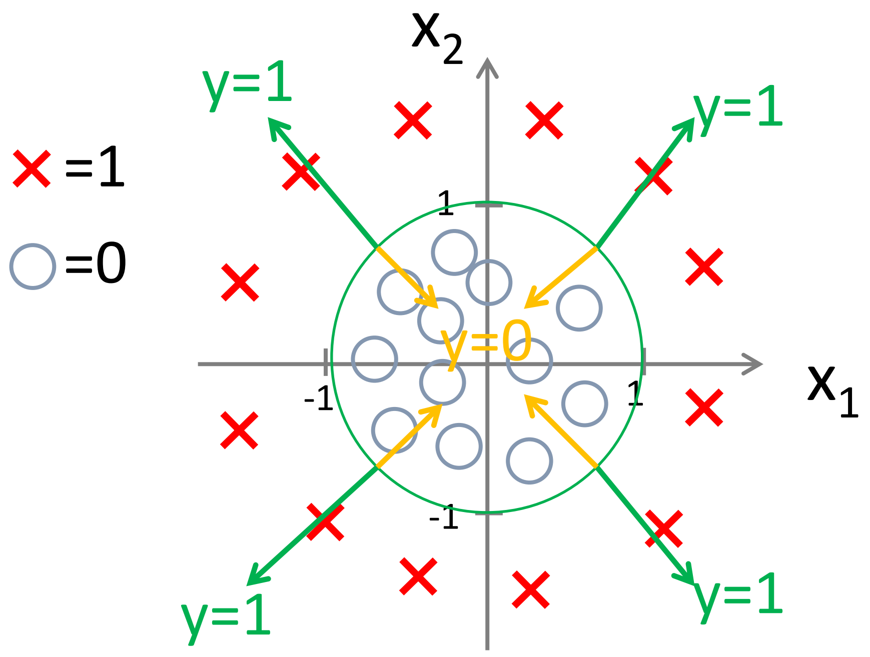
$$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2)$$

Let us suppose: $\theta = \begin{bmatrix} -3 \\ 1 \\ 1 \end{bmatrix}$

Predict “y=1” if $-3 + x_1 + x_2 \geq 0$

$$x_1 + x_2 \geq 3$$

Nonlinear decision boundary



$$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2)$$

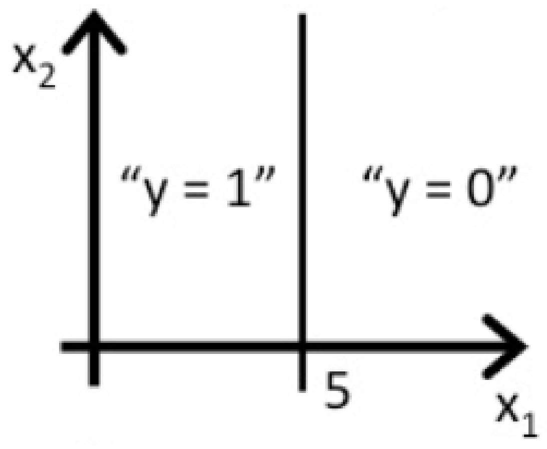
Let us suppose: $\theta = \begin{bmatrix} -1 \\ 0 \\ 0 \\ 1 \\ 1 \end{bmatrix}$

Predict "y=1" if $-1 + x_1^2 + x_2^2 \geq 0$

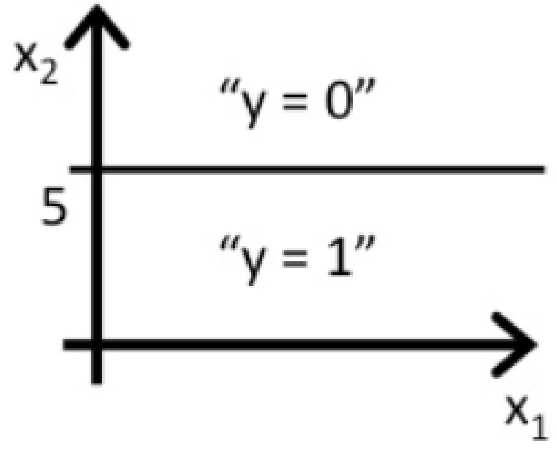
Прашање

- Претпоставете дека имаме логистичка регресија со две карактеристики x_1 и x_2. Претпоставете дека $\theta_0 = 5, \theta_1 = -1$ и $\theta_2 = 0$. Кои од следните графици ја покажуваат границата на одлука на $h_\theta(x)$?

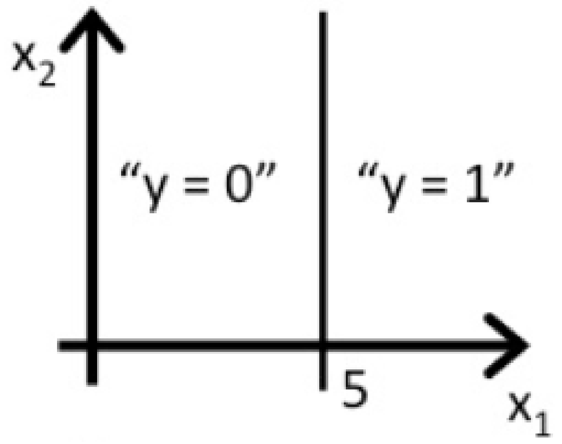
a)



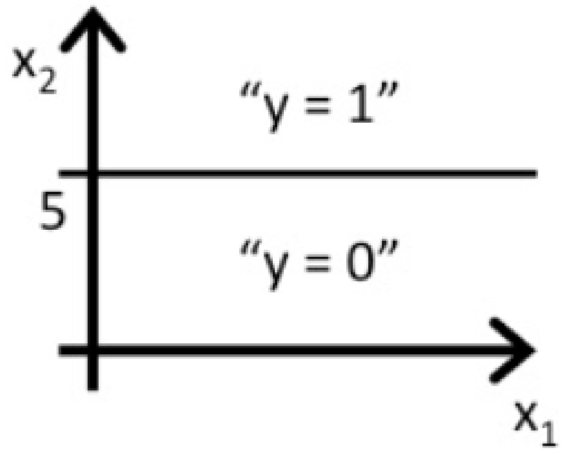
b)



c)



d)



Model specification

- Generalize linear regression to the (binary) classification setting by making **two changes**
- First, we replace the Gaussian distribution for y with a **Bernoulli** distribution (more appropriate for $y \in \{0, 1\}$)

  $$p(y|\mathbf{x},\mathbf{w}) = \mathrm{Ber}(y|\mu(\mathbf{x}))$$

  where $\mu(\mathbf{x}) = \mathrm{E}\,[y|\mathbf{x}] = p(y = 1|\mathbf{x})$.
- Second, we compute a linear combination of the inputs, as before, but then we pass this through a function that ensures $0 \leq \mu(\mathbf{x}) \leq 1$ by defining

  $$\mu(\mathbf{x}) = \mathrm{sigm}(\mathbf{w}^T\,\mathbf{x})$$

  where $\mathrm{sigm}(\eta)$ refers to the **sigmoid** function, also known as the **logistic** or **logit** function

Model specification

- The sigmoid function is defined as

$$\mathrm{sigm}(\eta) \triangleq \frac{1}{1+\exp(-\eta)} = \frac{e^\eta}{e^\eta + 1}$$

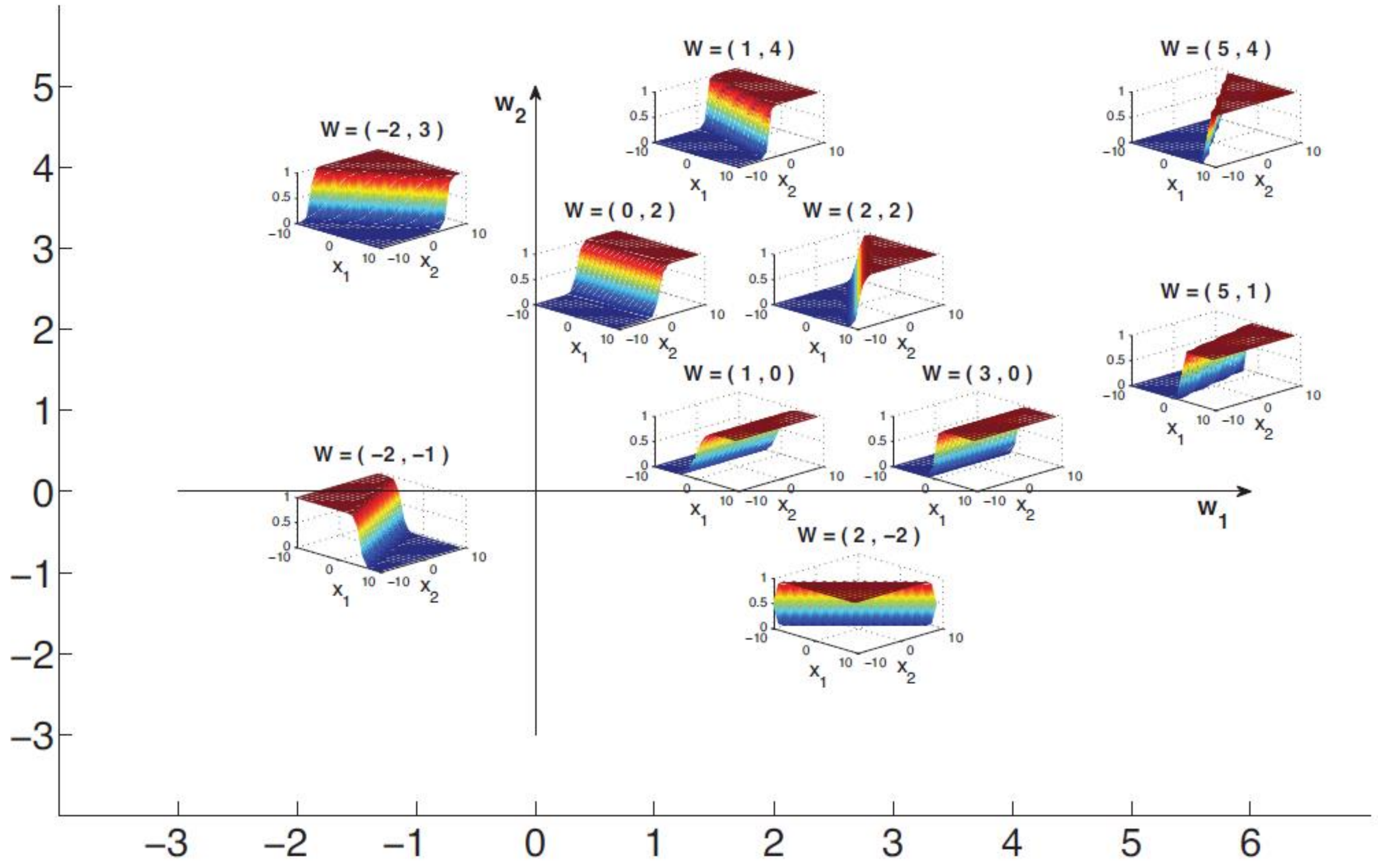
- Putting these two steps together we get

$$p(y|\mathbf{x},\mathbf{w}) = \mathrm{Ber}(y|\mathrm{sigm}(\mathbf{w}T\,\mathbf{x}))$$

- This is called **logistic regression** due to its similarity to linear regression (although it is a form of classification, not regression!).

Model specification

- Plots of $p(y = 1|\mathbf{x},\mathbf{w}) = \mathrm{sigm}(\mathbf{w}^T\mathbf{x})$ for 2d input and different weight vectors **w**. If we threshold these probabilities at 0.5, we induce a linear decision boundary



Model fitting

- Algorithms for estimating the parameters of a logistic regression model.
- MLE – maximum likelihood estimation
  - The negative log-likelihood for logistic regression is given by

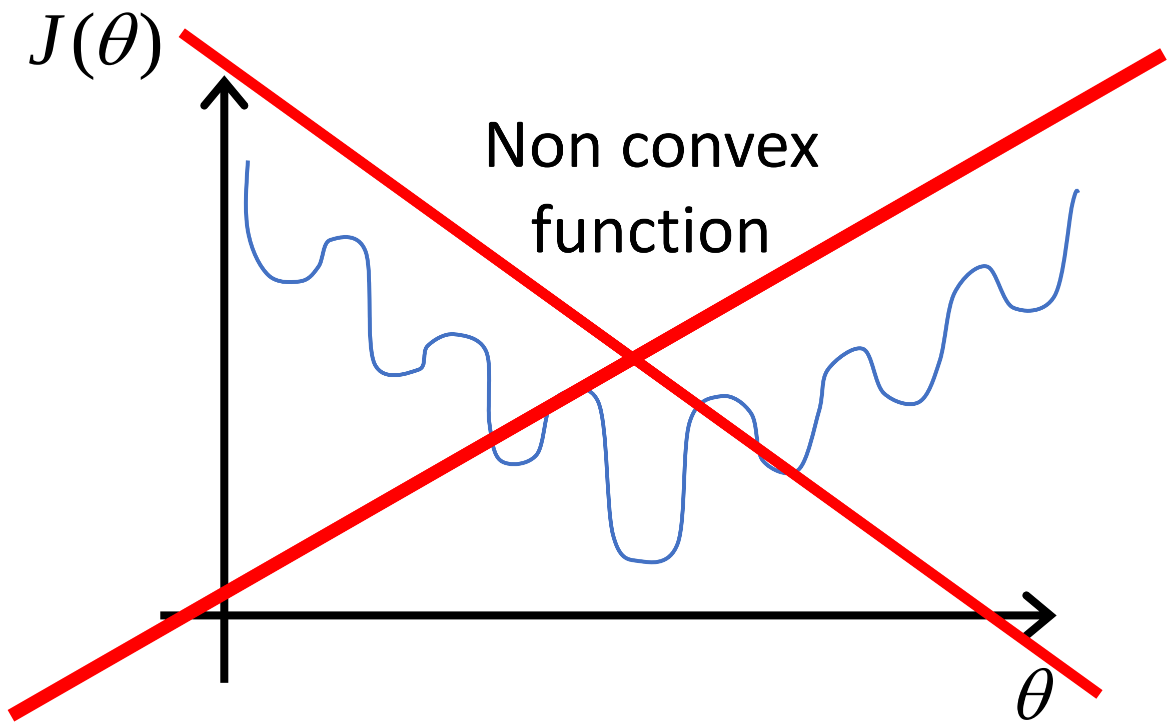
$$\begin{aligned}\mathrm{NLL}(\mathbf{w}) &= -\sum_{i=1}^{N} \log[\mu_i^{\mathbb{I}(y_i=1)} \times (1-\mu_i)^{\mathbb{I}(y_i=0)}] \\ &= -\sum_{i=1}^{N} [y_i \log \mu_i + (1-y_i)\log(1-\mu_i)]\end{aligned}$$

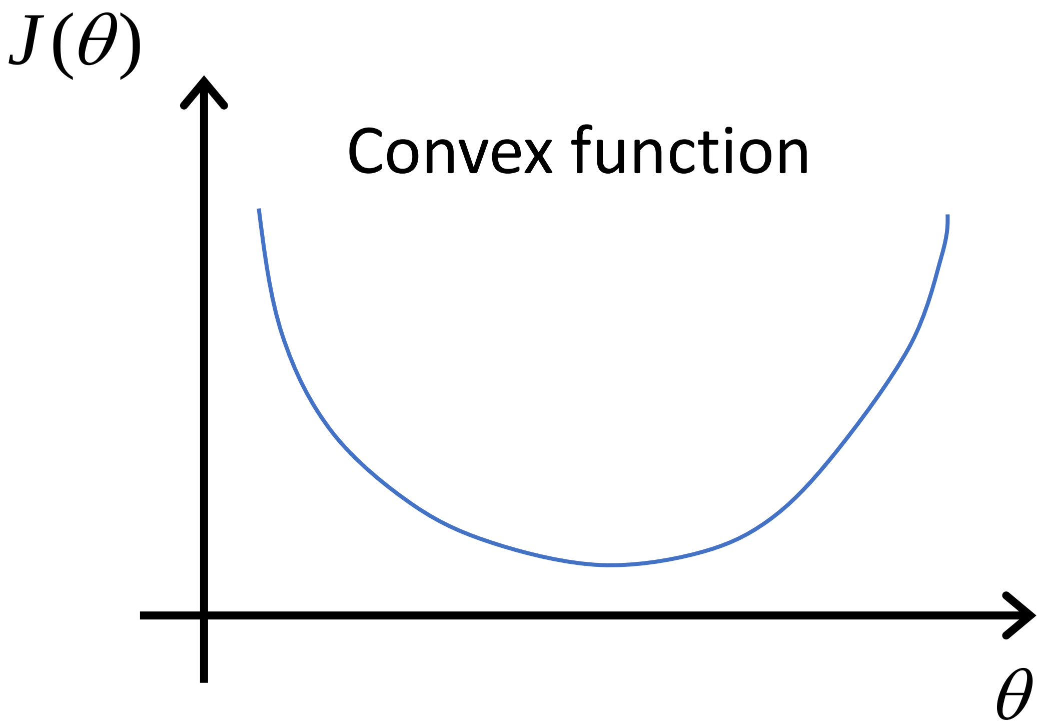
  - This is also called the **cross-entropy** error function
- Unlike linear regression, we can no longer write down the MLE in closed form.

Cost function

Linear regression: $$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\frac{1}{2}(h_\theta(x^{(i)}) - y^{(i)})^2$$

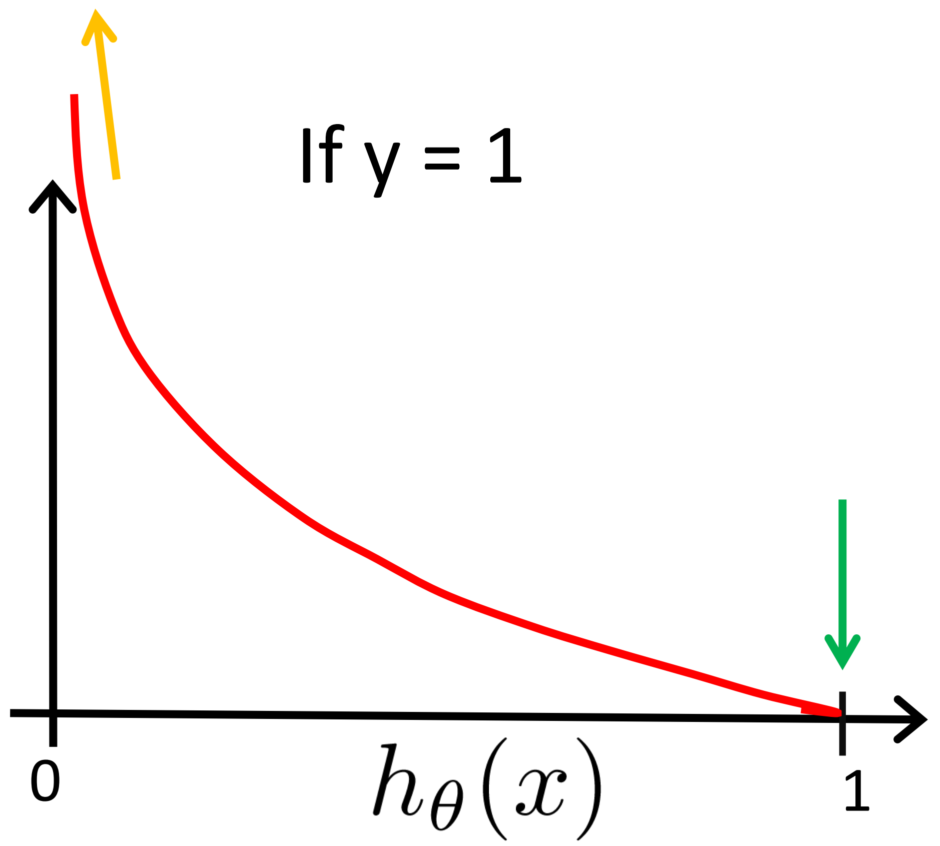
Cost: $$C(h_\theta(x^{(i)}), y^{(i)}) = \frac{1}{2}(h_\theta(x^{(i)}) - y^{(i)})^2$$





Cost function for logistic regression

$$C(h_\theta(x), y) = \begin{cases} \boxed{-\log(h_\theta(x)) \quad \text{ако } y = 1} \\ -\log(1 - h_\theta(x)) \quad \text{ако } y = 0 \end{cases}$$



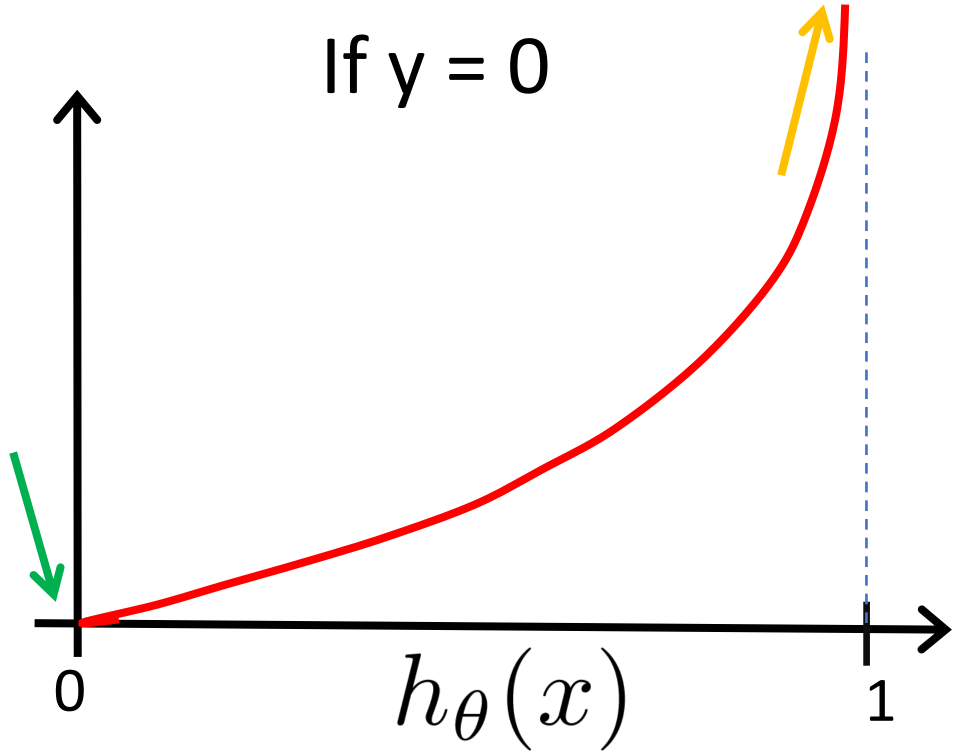
$$\boxed{C = 0 \text{ ако } y = 1, h_\theta(x) = 1}$$

But: $\boxed{\lim_{h_\theta \to 0} C(h_\theta(x), y) = \infty}$

It captures the case that if $h_\theta(x) = 0$, (that is we predict $P(y=1 | x;\theta) = 0$), but $y = 1$, then we will punish the algorithm with a huge cost

Cost function for logistic regression

$$C(h_\theta(x), y) = \begin{cases} -\log(h_\theta(x)) & ако\ y = 1 \\ \boxed{-\log(1 - h_\theta(x)) \quad ако\ y = 0} \end{cases}$$



$$\boxed{C = 0 \text{ ако } y = 0, h_\theta(x) = 0}$$

But: $\boxed{\lim_{h_\theta \to 1} C(h_\theta(x), y) = \infty}$

It captures the case that if $h_\theta(x) = 1$, (that is we predict $P(y = 0 \mid x; \theta) = 0$), but $y = 1$, then we will punish the algorithm with a huge cost

Cost function for logistic regression

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m} C(h_\theta(x^{(i)}), y^{(i)})$$

$$C(h_\theta(x), y) = \begin{cases} -\log(h_\theta(x)) & \text{ако } y = 1 \\ -\log(1 - h_\theta(x)) & \text{ако } y = 0 \end{cases}$$

Прашање

- Како може следната цена да се запише во една равенка?

$$C(h_\theta(x), y) = \begin{cases} -\log(h_\theta(x)) & ако\ y = 1 \\ -\log(1 - h_\theta(x)) & ако\ y = 0 \end{cases}$$

a) $C(h_\theta(x), y) = -y\log(h_\theta(x)) - (1-y)\log(1-h_\theta(x))$

b) $C(h_\theta(x), y) = -(1-y)\log(h_\theta(x)) - y\log(1-h_\theta(x))$

c) $C(h_\theta(x), y) = y\log(h_\theta(x)) + (1-y)\log(1-h_\theta(x))$

Cost function for logistic regression

$$J(\theta) = \frac{1}{m} \sum_{i=1}^{m} C(h_\theta(x^{(i)}), y^{(i)})$$

$$C(h_\theta(x), y) = \begin{cases} -\log(h_\theta(x)) & \text{if } y = 1 \\ -\log(1 - h_\theta(x)) & \text{if } y = 0 \end{cases}$$

$$\Downarrow$$

$$C(h_\theta(x), y) = -y \log(h_\theta(x)) - (1 - y) \log(1 - h_\theta(x))$$

MLE

- Since we can not write down the MLE in closed form, we need to use an optimization algorithm to compute it. For this, we need to derive the gradient and Hessian.

$$\begin{aligned}\mathbf{g} &= \frac{d}{d\mathbf{w}} f(\mathbf{w}) = \sum_i (\mu_i - y_i)\mathbf{x}_i = \mathbf{X}^T(\boldsymbol{\mu} - \mathbf{y}) \\ \mathbf{H} &= \frac{d}{d\mathbf{w}} \mathbf{g}(\mathbf{w})^T = \sum_i (\nabla_{\mathbf{w}} \mu_i)\mathbf{x}_i^T = \sum_i \mu_i(1-\mu_i)\mathbf{x}_i\mathbf{x}_i^T \\ &= \mathbf{X}^T\mathbf{S}\mathbf{X}\end{aligned}$$

where $\mathbf{S} \triangleq \mathrm{diag}(\mu_i(1-\mu_i))$

- **H** is positive definite hence the NLL is convex and has a unique global minimum.

Steepest descent

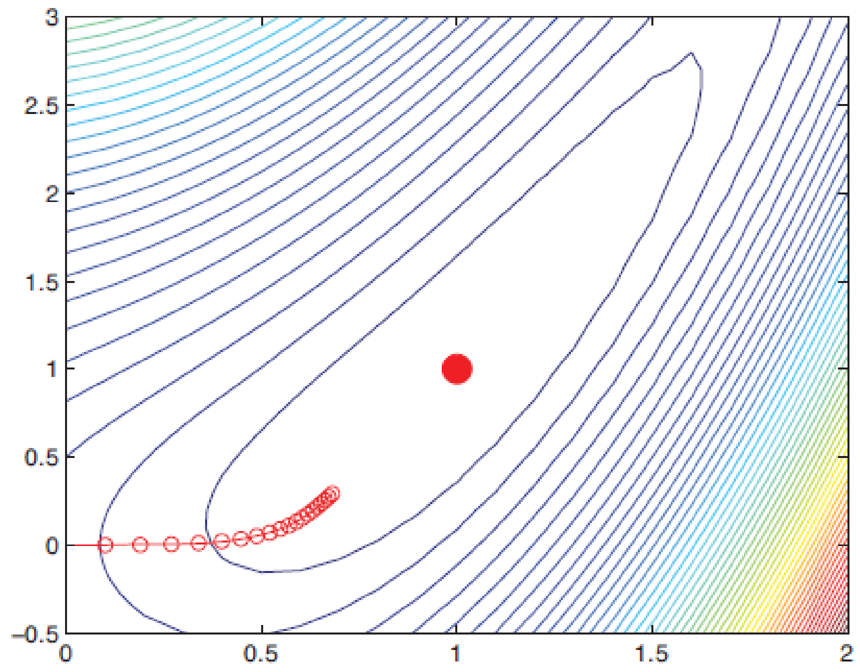
- The simplest algorithm for unconstrained optimization is **gradient descent** or **steepest descent:**

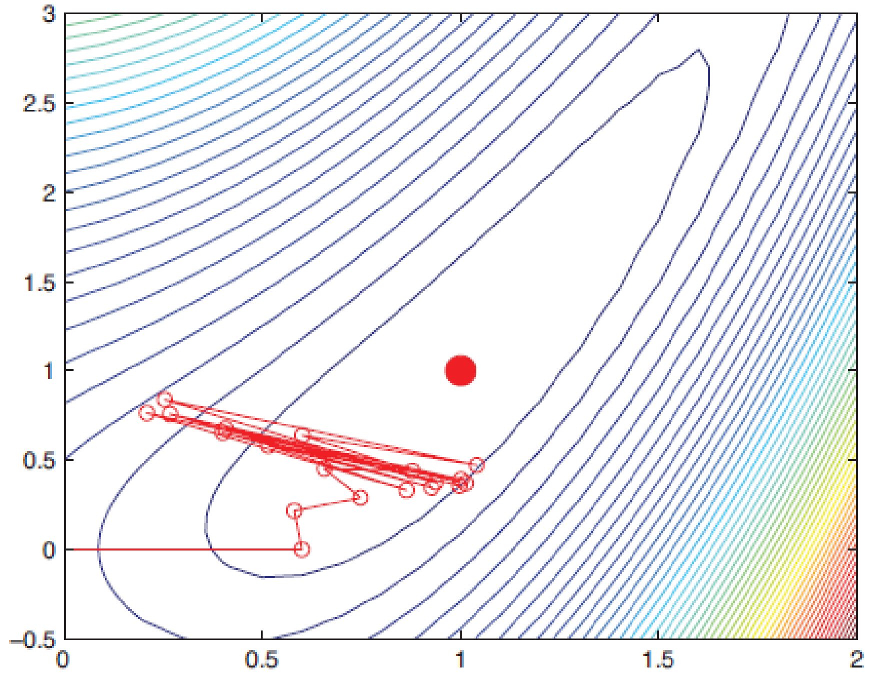
$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \eta_k \mathbf{g}_k$$

where η_k is the step size or **learning rate.**

- The main issue here is: **how to set the step size?**
- Gradient descent on a simple function, starting from (0, 0), for 20 steps, using a fixed learning rate (step size) η. The global minimum is at (1, 1). (a) $\eta = 0.1$. (b) $\eta = 0.6$

$$f(\boldsymbol{\theta}) = 0.5(\theta_1^2 - \theta_2)^2 + 0.5(\theta_1 - 1)^2$$





Gradient Descent

$$J(\theta) = -\frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)} \log h_\theta(x^{(i)}) + (1-y^{(i)})\log(1-h_\theta(x^{(i)}))\right]$$

Сакаме да најдеме $\min_{\theta} J(\theta)$

Повторувај $\{$

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

$\}$ (симултано ажурирање на сите θ_j)

Gradient Descent

$$J(\theta) = -\frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)} \log h_\theta(x^{(i)}) + (1-y^{(i)})\log(1-h_\theta(x^{(i)}))\right]$$

Сакаме да најдеме $\min_\theta J(\theta)$

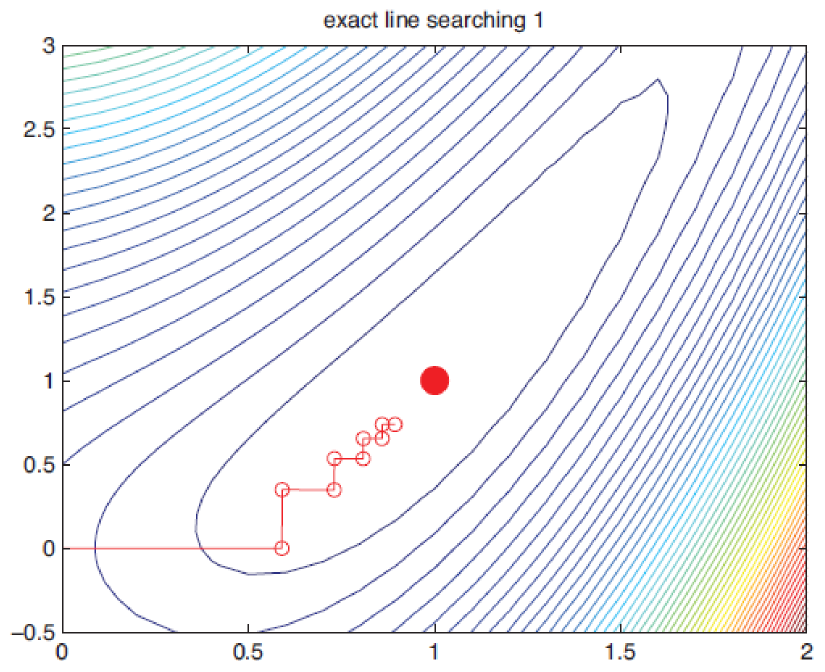
Повторувај $\{$

$$\theta_j := \theta_j - \alpha \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

$\}$ (симултано ажурирање на сите θ_j)

Алгоритамот изгледа идентично како кај линеарната регресија!

Steepest descent

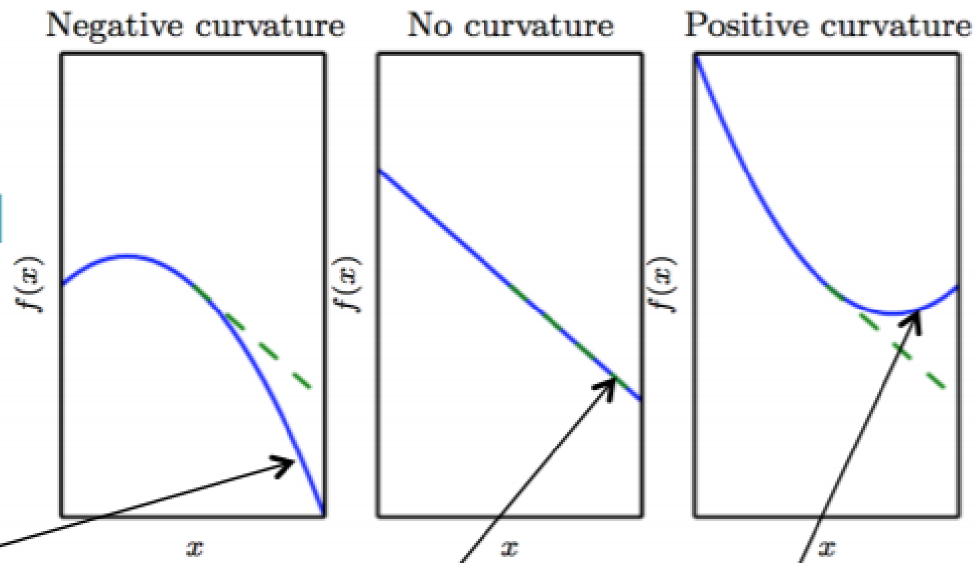


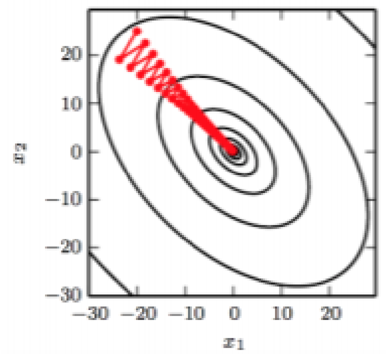
- Picking the step size –more stable method that is guaranteed to converge to a local optimum no matter where we start (global convergence).
- Pick η to minimize $\phi(\eta) = f(\boldsymbol{\theta}_k + \eta \mathbf{d}_k)$ **(line minimization** or **line search)**

Second derivative measures curvature

- Derivative of a derivative
- Quadratic functions with different curvatures

Dashed line is value of cost function predicted by gradient alone





Decrease is faster than predicted by Gradient Descent

Gradient Predicts decrease correctly

Decrease is slower than expected Actually increases

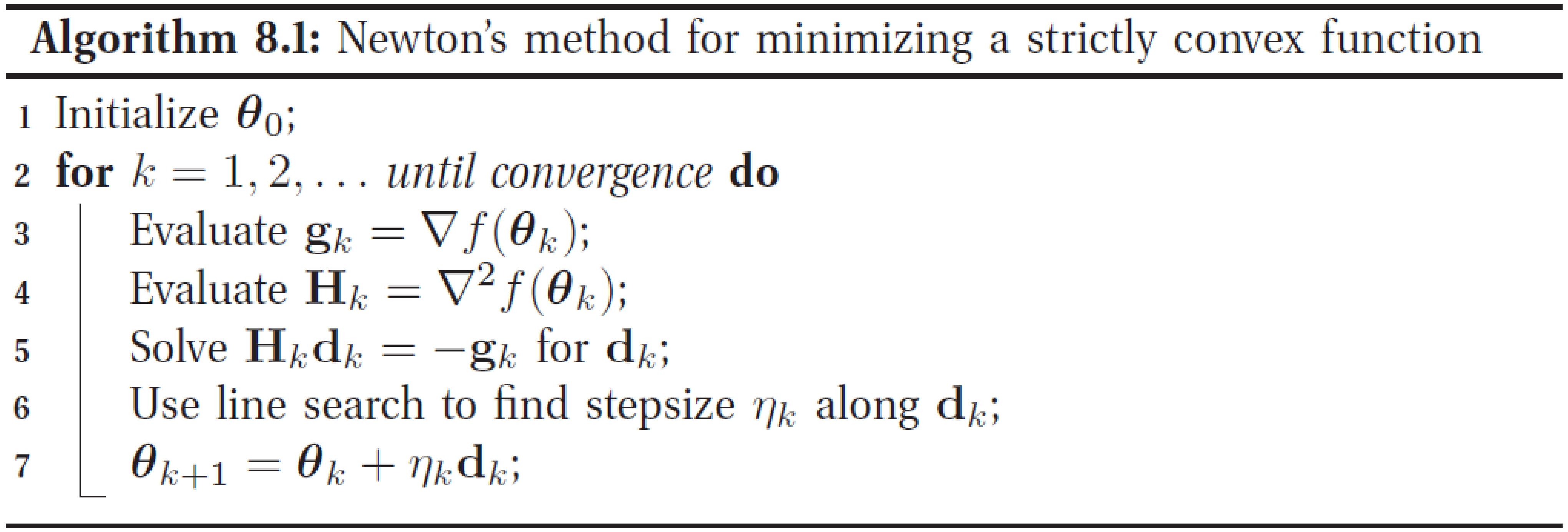
Newton’s method

- One can derive faster optimization methods by taking the curvature of the space (i.e. the Hessian) into account
- These are called **second order** optimization methods
- The primary example is the **Newton’s algorithm** (mother of all second-order optimization algorithms)
- This is an iterative algorithm which consists of updates of the form

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \eta_k \mathbf{H}_k^{-1} \mathbf{g}_k$$

- The derivation of the algorithm considers second-order Taylor series approximation of $f(\theta)$ around θ_k and finding Newton step d_k that minimizes the function

Newton’s method

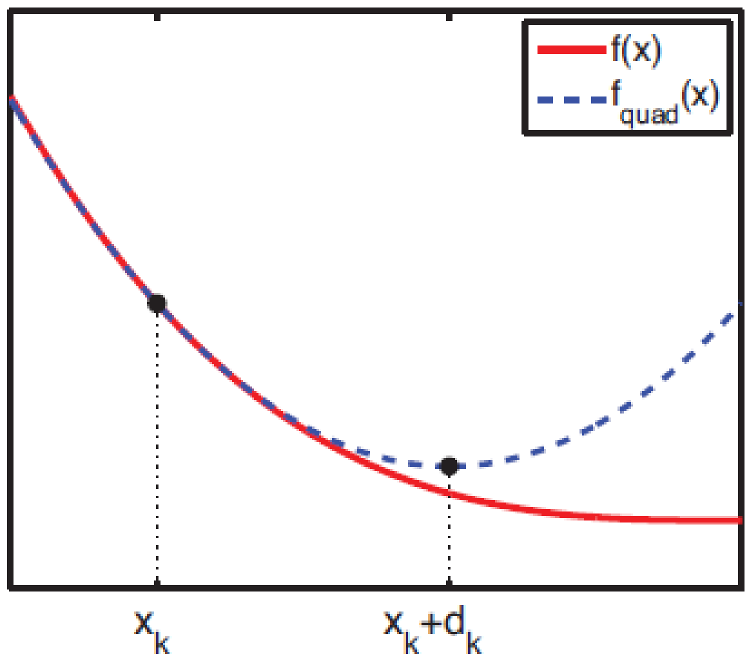


Algorithm 8.1: Newton's method for minimizing a strictly convex function

```
Initialize $\boldsymbol{\theta}_0$;
for $k = 1, 2, \ldots$ until convergence do
    Evaluate $\mathbf{g}_k = \nabla f(\boldsymbol{\theta}_k)$;
    Evaluate $\mathbf{H}_k = \nabla^2 f(\boldsymbol{\theta}_k)$;
    Solve $\mathbf{H}_k \mathbf{d}_k = -\mathbf{g}_k$ for $\mathbf{d}_k$;
    Use line search to find stepsize $\eta_k$ along $\mathbf{d}_k$;
    $\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k + \eta_k \mathbf{d}_k$;
```

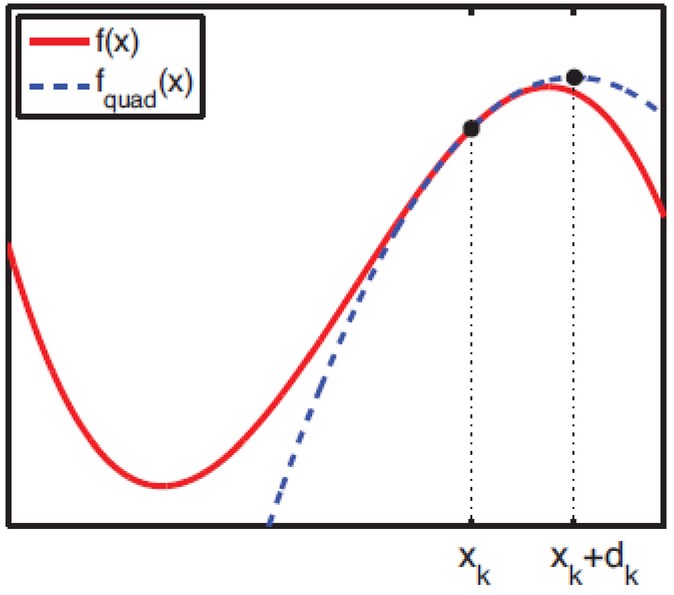
Newton's method – illustration for minimizing 1d function

- (a) The solid curve is the function $f(x)$. The dotted line $f_{quad}(x)$ is its second order approximation at x_k. The Newton step d_k is what must be added to x_k to get to the minimum of $f_{quad}(x)$.



Newton's method – illustration for minimizing 1d function

- (b) Illustration of Newton's method applied to a nonconvex function. We fit a quadratic around the current point x_k and move to its stationary point, $x_{k+1} = x_k + d_k$. Unfortunately, this is a local maximum, not minimum.
- In its simplest form Newton's method requires that $\mathbf{H}_k$ be positive definite, which will hold if the function is strictly convex. If not, the objective function is not convex, then $\mathbf{d}_k$ may not be a descent direction. In this case, one simple strategy is to revert to steepest descent, $\mathbf{d}_k = -\mathbf{g}_k$. The **Levenberg Marquardt** algorithm is an adaptive way to blend between Newton steps and steepest descent steps.



Other methods

- Iteratively reweighted least squares (IRLS) – Newton’s algorithm applied to find MLE for binary logistic regression
- Quasi-Newton method – when it is too expensive to compute **H** explicitly. This method iteratively builds up an approximation of the Hessian using information extracted from the gradient vector at each step

Online learning

- Traditionally machine learning is performed **offline**, which means we have a **batch** of data, and we optimize an equation of the following form

$$f(\boldsymbol{\theta}) = \frac{1}{N}\sum_{i=1}^{N} f(\boldsymbol{\theta}, \mathbf{z}_i)$$

- where $\mathbf{z}_i = (\mathbf{x}_i, y_i)$ in the supervised case, or just $\mathbf{x}_i$ in the unsupervised case, and $f(\boldsymbol{\Theta}, \mathbf{z}_i)$ is some kind of loss function, for example

$$f(\boldsymbol{\theta}, \mathbf{z}_i) = L(y_i, h(\mathbf{x}_i, \boldsymbol{\theta}))$$

- where $h(\mathbf{x}_i, \boldsymbol{\Theta})$ is a prediction function, and $L(y, \hat{y})$ is some loss function such as squared error

Online learning

- However, if we have **streaming data**, we need to perform **online learning**, so we can update our estimates as each new data point arrives rather than waiting until “the end” (which may never occur).
- And even if we have a batch of data, we might want to treat it like a stream if it is too large to hold in main memory

Stochastic gradient descent

- Suppose we receive an infinite stream of samples from the distribution. One way to optimize stochastic objectives is to perform the update in the previous equation at each step.
- Instead of going through all examples, Stochastic Gradient Descent (SGD) performs the parameters update on each example
- It adds noise to the learning process that helps improving generalization error. However, this would increase the run time.

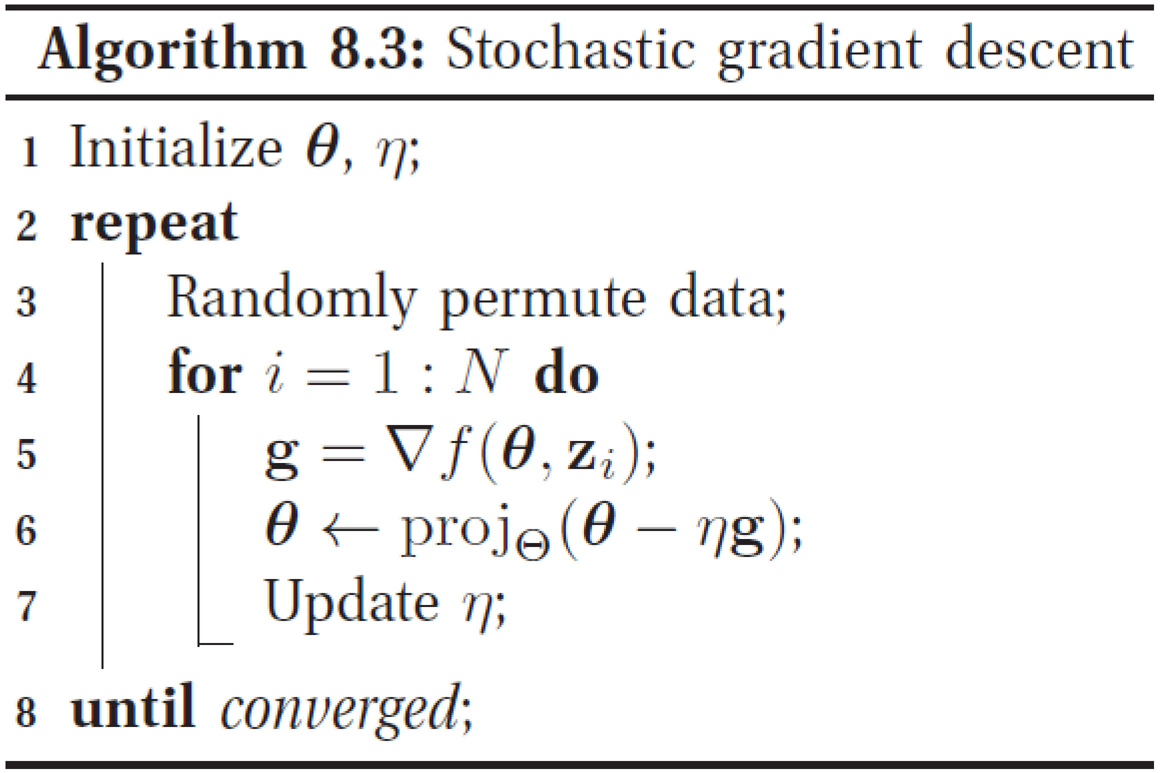
```
for i in range(num_epochs):
    np.random.shuffle(data)
    for example in data:
        grad = compute_gradient(example, params)
        params = params - learning_rate * grad
```

Setting the step size

- The set of values of η_k over time is called the learning rate **schedule**
- Various formulas are used such as $\eta_k = 1/k$
- The need to adjust these tuning parameters is one of the main drawback of stochastic optimization.
- One simple heuristic is as follows: store an initial subset of the data, and try a range of η values on this subset; then choose the one that results in the fastest decrease in the objective and apply it to all the rest of the data

SGD compared to batch learning

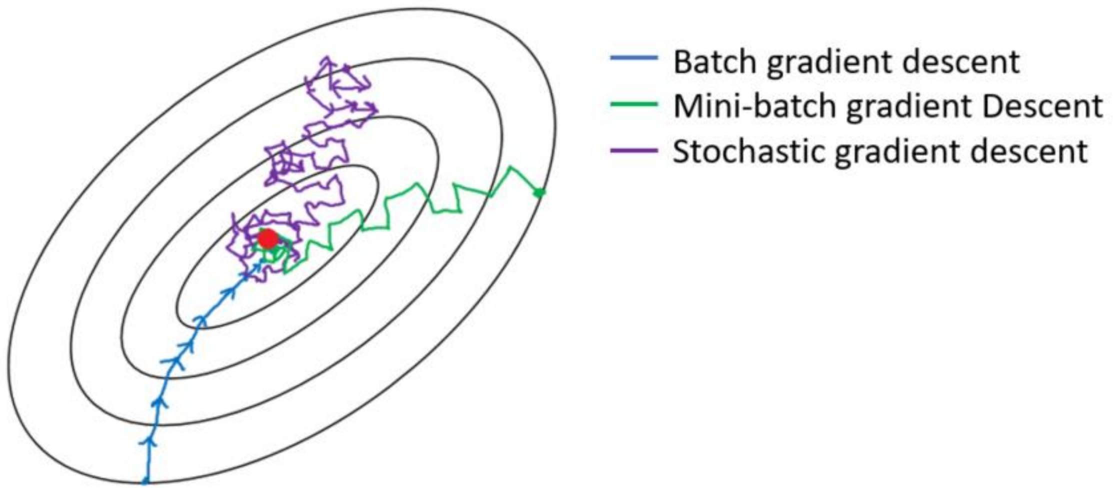
- If we don't have an infinite data stream, we can "simulate" one by sampling data points at random from our training set.
- A single pass over the entire data set is called an **epoch**.
- In this offline case, it is often better to compute the gradient of a **mini-batch** of *B* data cases. If *B* = 1, this is standard SGD, and if *B* = *N*, this is standard **steepest descent**. Typically *B* $\sim$100 is used.



Algorithm 8.3: Stochastic gradient descent

```
Initialize $\boldsymbol{\theta}$, $\eta$;
repeat
    Randomly permute data;
    for $i = 1 : N$ do
        $\mathbf{g} = \nabla f(\boldsymbol{\theta}, \mathbf{z}_i)$;
        $\boldsymbol{\theta} \leftarrow \text{proj}_{\Theta}(\boldsymbol{\theta} - \eta \mathbf{g})$;
        Update $\eta$;
until converged;
```

SGD compared to batch learning



- Although a simple first-order method, SGD performs surprisingly well on some problems, especially ones with large data sets
- The intuitive reason for this is that one can get a **fairly good estimate** of the gradient by looking at **just a few examples**. Carefully evaluating precise gradients using large datasets is often a waste of time, since the algorithm will have to recompute the gradient again anyway at the next step. It is often a better use of computer time to have a noisy estimate and to move rapidly through parameter space.
- As an extreme example, suppose **we double the training set** by duplicating every example. Batch methods will take twice as long, but online methods will be unaffected, since the direction of the gradient has not changed
- In addition to enhanced speed, SGD is often **less prone to getting stuck in shallow local minima, because it adds a certain amount of "noise"**. Consequently it is quite popular in the machine learning community for fitting models with non-convex objectives, such as neural networks and deep belief networks

The LMS algorithm

- As an example of SGD, let us consider how to compute the MLE for linear regression in an online fashion. The online gradient at iteration *k* is given by

$$\mathbf{g}_k = \mathbf{x}_i(\theta_k^T \mathbf{x}_i - y_i)$$

- where *i* = *i*(*k*) is the training example to use at iteration *k*. If the data set is streaming, we use *i*(*k*) = *k*; we shall assume this from now on, for notational simplicity.
- The equation is easy to interpret: it is the feature vector $\mathbf{x}_k$ weighted by the difference between what we predicted, $\widehat{y_k} = \theta_k^T x_k$, and the true response,$y_k$; hence the gradient acts like an error signal.

The LMS algorithm

- After computing the gradient, we take a step along it as follows:

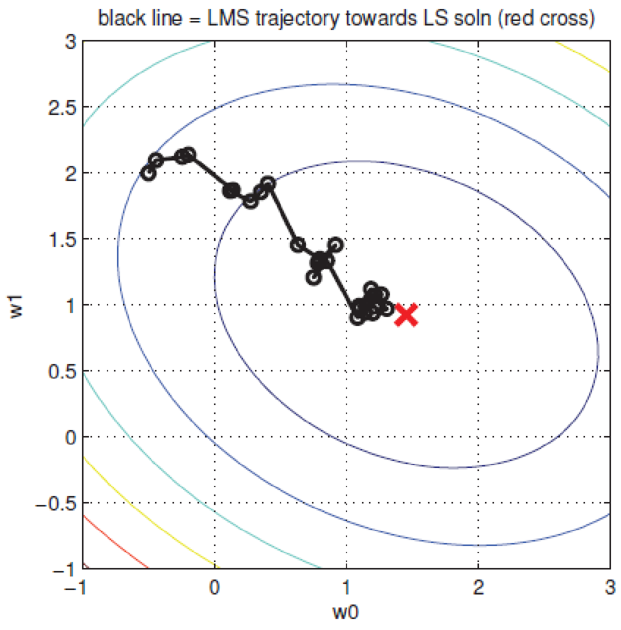
$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \eta_k(\hat{y}_k - y_k)\mathbf{x}_k$$

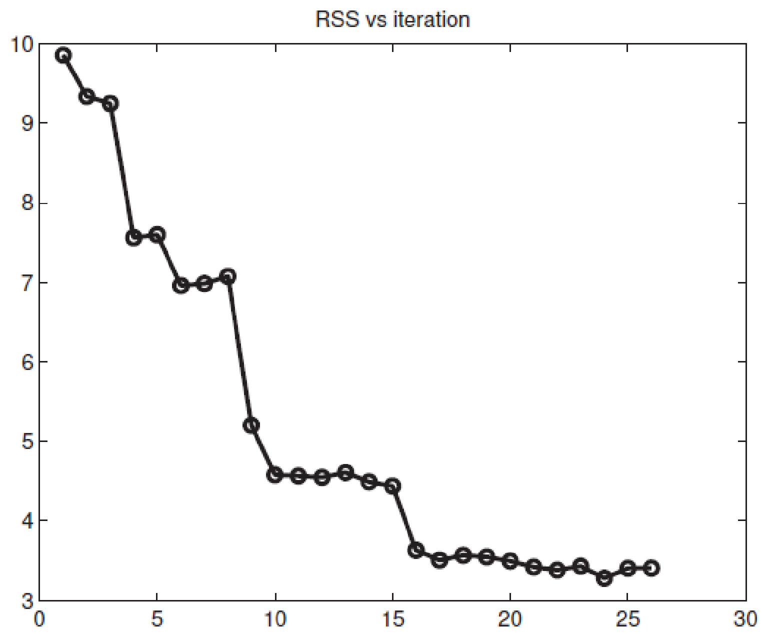
- This algorithm is called the **least mean squares** or **LMS** algorithm

- Left: we start from $\boldsymbol{\theta} = (-0.5, 2)$ and slowly converging to the least squares solution of

$$\hat{\boldsymbol{\theta}} = (1.45, 0.92)$$

- Right: plot of objective function over time. Note that it does not decrease monotonically.





Online case for binary logistic regression

- Now let us consider how to fit a binary logistic regression model in an online manner.
- In the online case, the weight update has the simple form

$$\boldsymbol{\theta}_k = \boldsymbol{\theta}_{k-1} - \eta_k \mathbf{g}_i = \boldsymbol{\theta}_{k-1} - \eta_k(\mu_i - y_i)\mathbf{x}_i$$

where $\mu_i = p(y_i = 1|\mathbf{x}_i, \boldsymbol{\theta}_k) = \mathbb{E}\left[y_i|\mathbf{x}_i, \boldsymbol{\theta}_k\right]$

- We see that this has exactly the same form as the LMS algorithm.

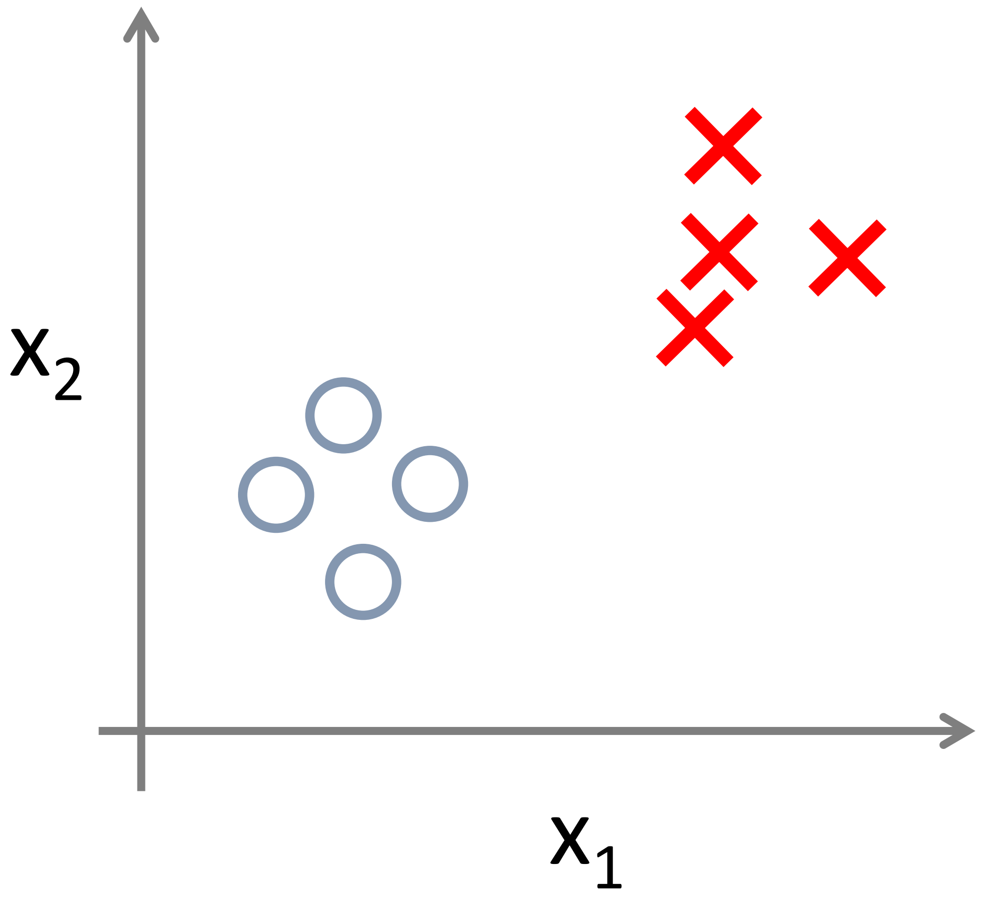
Повеќекласна класификација

Означување на електронски пораки: Работа, Пријатели, Фамилија, Хоби

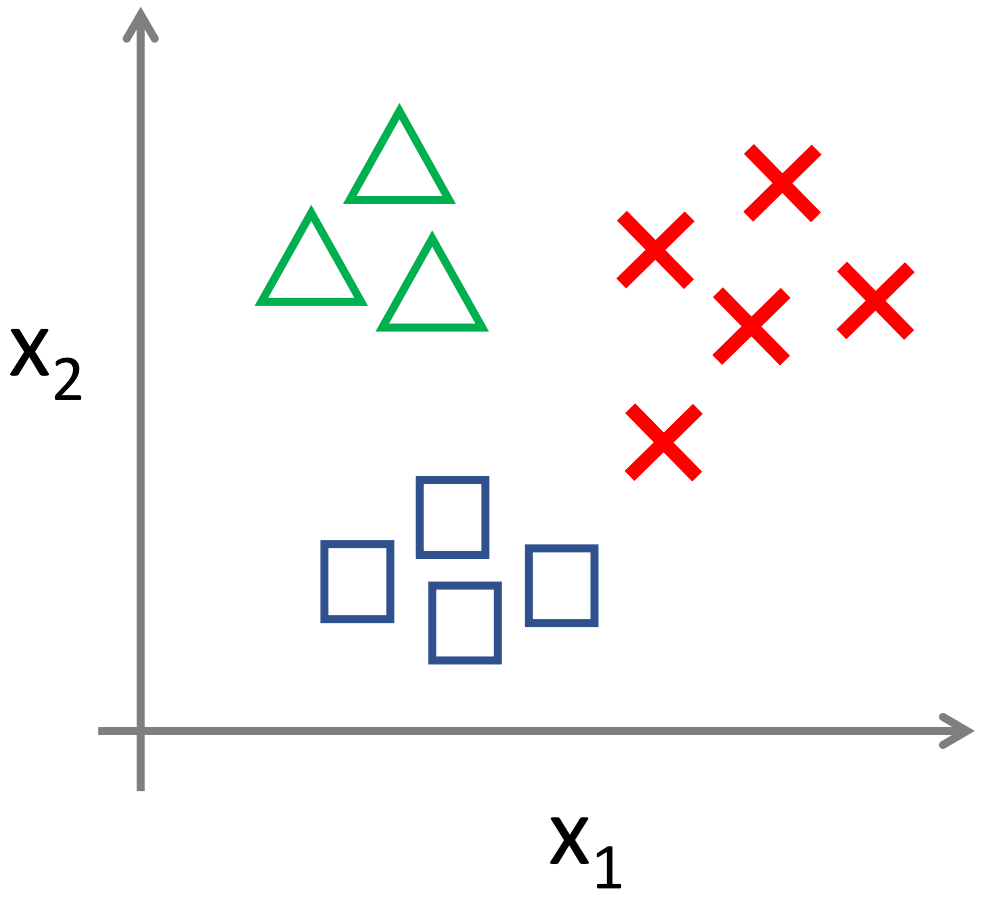
Медицинска дијагноза: Здрав, Настинка, Грип

Временски прилики: Сончево, Облачно, Дождливо, Снежно

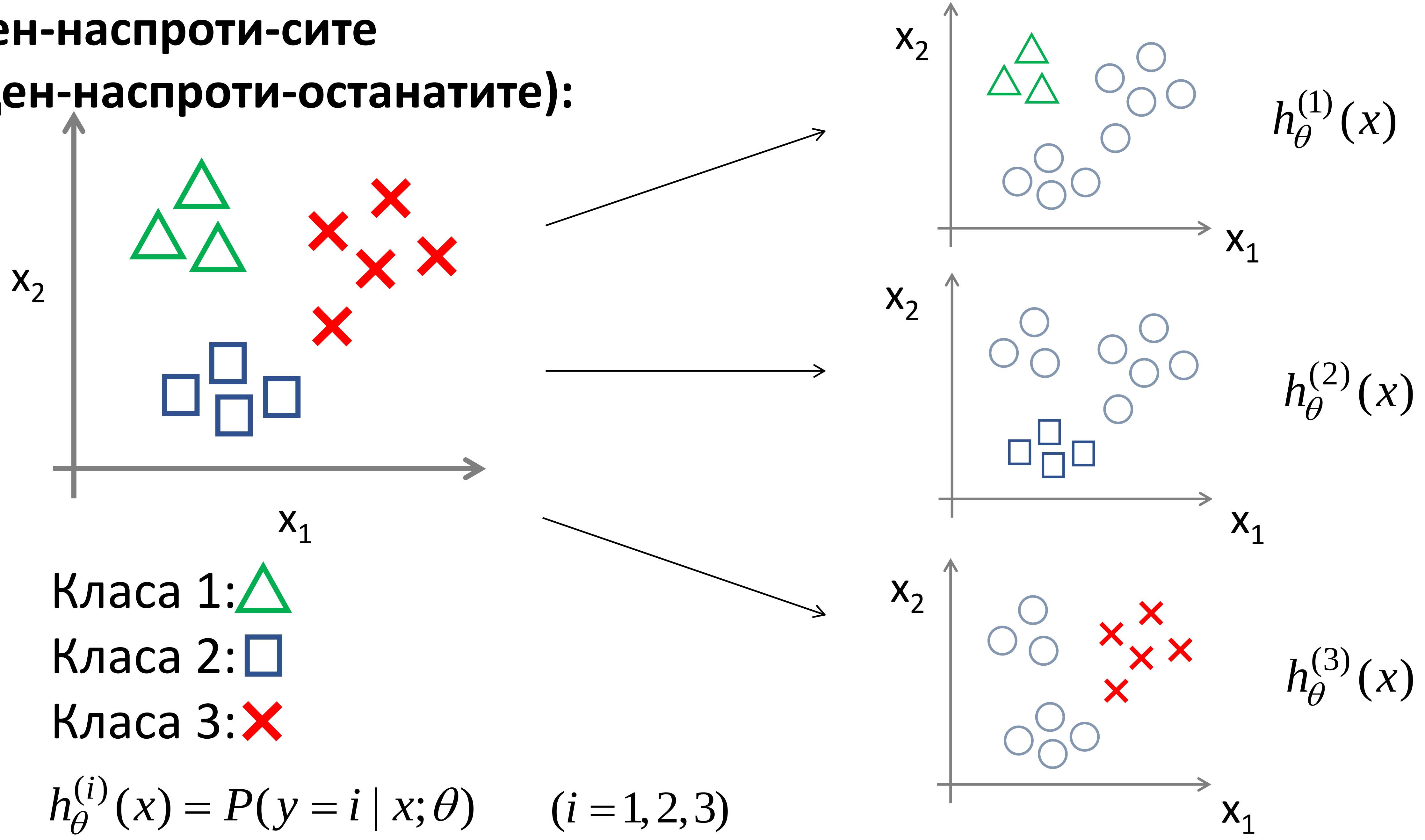
Бинарна класификација:



Повеќе-класна класификација:



Еден-наспроти-сите (еден-наспроти-останатите):



Класа 1: △
Класа 2: □
Класа 3: ✕

$$h_\theta^{(i)}(x) = P(y = i \mid x; \theta) \qquad (i = 1, 2, 3)$$

Еден-наспроти-сите

Обучуваме класификатор со логистичка регресија $h_\theta^{(i)}(x)$ за секоја класа i за да ја пресметаме веројатноста дека $y = i$.

За нов влез x, за да направиме предвидување, ја избираме класата i за која се максимизира

$$\max_i h_\theta^{(i)}(x)$$

L_2 Regularization

- In Linear Regression, to avoid overfitting and extreme values of the coefficients, we added a Normal prior to the coefficients which resulted in a L_2 norm term
  - Preference of MAP estimate instead of MLE
- We can do the same thing for Logistic Regression, resulting in new equations
  - Equations for Logistic Ridge Regression

$$\begin{aligned} f'(\mathrm{w}) &= \mathrm{NLL}(\mathrm{w}) + \lambda \mathrm{w}^T \mathrm{w} \\ \mathrm{g}'(\mathrm{w}) &= \mathrm{g}(\mathrm{w}) + \lambda \mathrm{w} \\ \mathrm{H}'(\mathrm{w}) &= \mathrm{H}(\mathrm{w}) + \lambda \mathrm{I} \end{aligned}$$

Bayesian Logistic Regression

- Instead of using a point estimate for the coefficients w in Logistic regression, compute the full posterior over the parameters, $p(\mathbf{w}|D)$
- This can be useful for any situation where we want to associate confidence intervals with our predictions
- Solutions:
  - Gaussian (Laplace) approximation
  - Markov Chain Monte Carlo (MCMC)