Lecture 3

Linear methods

Machine Learning

Sergey Muravyov

29.01.2024

Lecture plan

- Linear classification
- Gradient descent
- Linear regression and matrix decomposition
- Regularization

Lecture plan

- Linear classification
- Gradient descent
- Linear regression and matrix decomposition
- Regularization

Problem formulation

Constraint: $Y = \{-1, +1\}$

$T^{\ell} = \{(x_i, y_i)\}_{i=1}^{\ell}$ is given

Find classifier $a_w(x, T^{\ell})$ in form $\text{sign}(f(x, w))$,

where $f(x, w)$ is a discernment function,

w is a parameter vector.

Key hypothesis: objects are (well-)separable.

Main idea: search among separating surfaces described with $f(x, w) = 0$.

Margin

Margin of object x_i:

$$M_i(w) = y_i f(x_i, w),$$

$M_i(w) < 0$ is an evidence of misclassification.

Margin

Margin of object x_i:

$$M_i(w) = y_i f(x_i, w),$$

$M_i(w) < 0$ is an evidence of misclassification.

We have previously defined **margin** of object x_i as

$$M(x_i) = C_{y_i}(x_i) - \max_{y \in Y \setminus \{y_i\}} C_y(x_i),$$

where $C_y(u) = \sum_{i=1}^{\ell} [y(u,i) = y] w(i,u)$, $w(i,u)$ is function of u's ith neighbor importance.

What is their relation?

Loss function smoothing

Empirical risk:

$$Q\left(a_w, T^\ell\right) = Q(w) = \sum_i^\ell [M_i(w) < 0],$$

is simply the number of errors classifier a_w shows.

The function is not smooth, so it is hard to find optima.

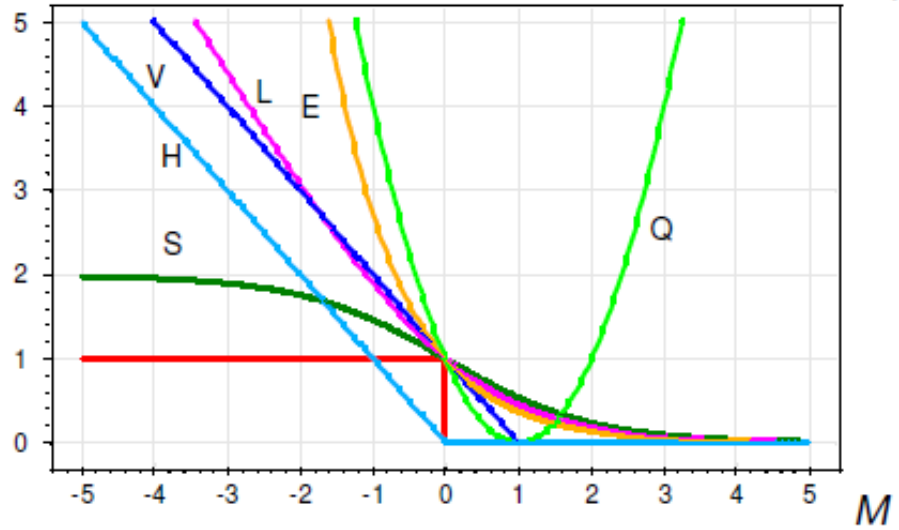
Approximation:

$$\tilde{Q}(w) = \sum_i^\ell L(M_i(w)),$$

where $L\left(M_i(w)\right) = L(a_w\left(x_i, T^\ell\right), x_i)$ is a loss function.

Smooth loss functions

We want L to be non-negative, non-increasing, and smooth:



$H(M) = (-M)_+$ — piecewise linear (Hebb's rule);
$V(M) = (1 - M)_+$ — piecewise linear (SVM);
$L(M) = \log_2(1 + e^{-M})$ — logarithmic (LR);
$Q(M) = (1 - M)^2$ — square (LDA);
$S(M) = 2(1 + e^M)^{-1}$ — sigmoid (ANN);
$E(M) = e^{-M}$ — exponential (AdaBoost).

Linear classifier

$f_j: X \to \mathbb{R}, j = 1, \ldots, n$ are numeric features.

Linear classifier:

$$a_w\left(x, T^\ell\right) = \operatorname{sign}\left(\sum_{i=1}^{n} w_i f_i(x) - w_0\right).$$

$w_1, \ldots w_n \in \mathbb{R}$ are feature **weights**.

Equivalent notation:

$$a_w\left(x, T^\ell\right) = \operatorname{sign}(\langle w, x \rangle),$$

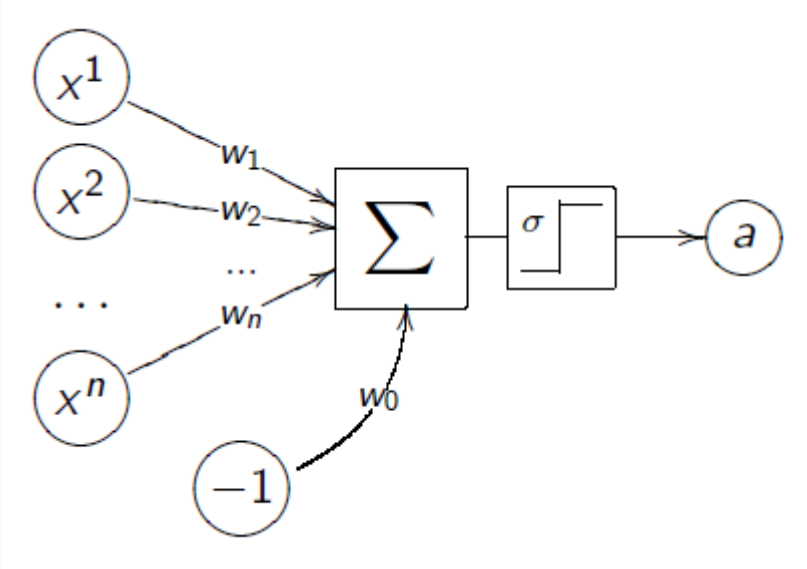
if a feature $f_0(x) = -1$ is added.

Neuron

McCulloch-Pitts neuron:

$$a_w\left(x, T^{\ell}\right) = \sigma\left(\sum_{i=1}^{n} w_i f_i(x) - w_0\right),$$

where σ is an activation function.



Set of algorithms

That assumption of how the classifier should look like specified the algorithm set A_{linear} from which we must choose an algorithm.

How does this algorithm set look like?

Set of algorithms

That assumption of how the classifier should look like specified the algorithm set A_{linear} from which we must choose an algorithm.

How does this algorithm set look like?

$$A_{\text{linear}} = \{a_w(x) = \text{sign}(\langle w, x \rangle) | w \in \mathbb{R}^n\}$$

How to learn a linear classifier?

We should find parameter vector w.

We can use almost any optimization algorithm capable to optimize empirical risk in the corresponding space.

The empirical risk is not black-box function.

Even more, we have guaranteed that it is a smooth function.

Lecture plan

- Linear classification
- Gradient descent
- Linear regression and matrix decomposition
- Regularization

Gradient descent

Empirical risk minimization problem

$$\tilde{Q}(w) = \sum_i^{\ell} L\big(M_i(w)\big) = \sum_i^{\ell} L(\langle w, x_i\rangle y_i) \to \min_w.$$

Gradient descent (a.k.a Batch gradient descent):

$w^{[0]} =$ **an initial guess value**;

$w^{[k+1]} = w^{[k]} - \mu\nabla Q\big(w^{[k]}\big)$,

where μ is a **gradient step** a.k.a **learning rate.**

$$w^{[k+1]} = w^{[k]} - \mu \sum_i^{\ell} L'(\langle w, x_i\rangle y_i)\, x_i y_i.$$

Stochastic gradient descent

Problem is that there are too many objects function of which should be reestimated on each step.

Stochastic gradient descent:

$w^{[0]}$ is **an initial guess values**;

$x_{(1)}, \ldots, x_{(\ell)}$ is **an objects order**;

$$w^{[k+1]} = w^{[k]} - \mu L'\left(\langle w^{[k]}, x_{(k)}\rangle y_{(k)}\right),$$

$$Q^{[k+1]} = (1-\alpha)Q^{[k]} + \alpha L\left(\langle w^{[k]}, x_{(k)}\rangle y_{(k)}\right).$$

Stop when values of Q and/or w do not change much.

Mini-batch gradient descent

Problem is that it is a bit too random because it depends only on a single object.

Mini-batch gradient descent:

$w^{[0]}$ is **an initial guess values**; b is **a batch size**

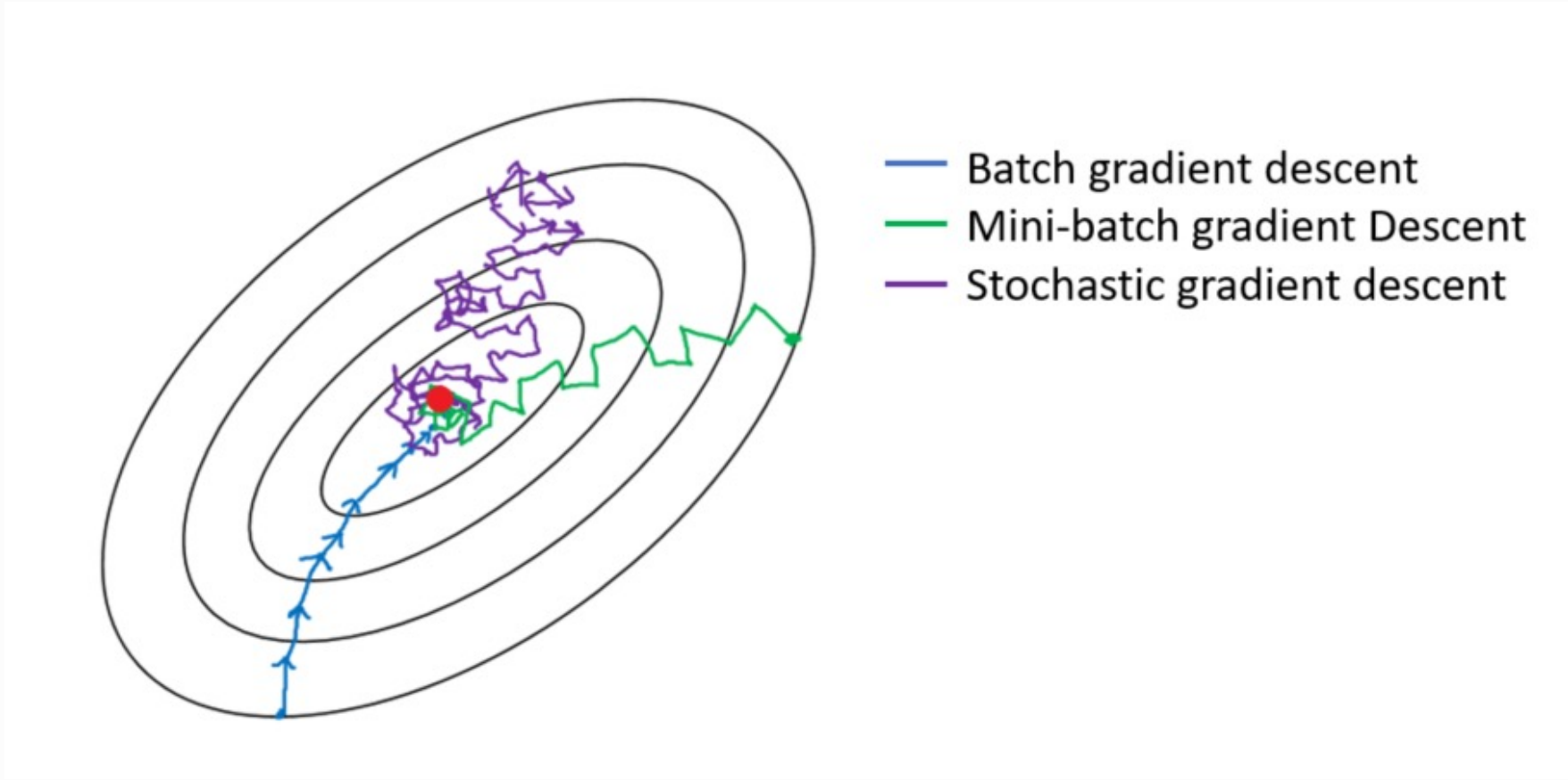
$x_{(1)}, \dots, x_{(\ell)}$ is **an objects order**;

$$w^{[K+1]} = w^{[K]} - \mu \sum_{k=Kb}^{k=(K+1)b} L'\left(\langle w^{[K]}, x_{(k)}\rangle y_{(k)}\right),$$

$$Q^{[K+1]} = (1-\alpha)Q^{[K]} + \alpha \sum_{k=Kb}^{k=(K+1)b} L\left(\langle w^{[K]}, x_{(k)}\rangle y_{(k)}\right).$$

Stop when values of Q and/or w do not change much.

Comparison


Batch gradient descent
Mini-batch gradient Descent
Stochastic gradient descent

Rosenblatt's rule and Hebb's rule

Rosenblatt's rule for $\{1; 0\}$ classification case for weight learning: for each object $x_{(k)}$ change the weight vector:

$$w^{[k+1]} := w^{[k]} - \eta(a_w(x_{(k)}) - y_{(k)}).$$

Hebb's rule for $\{1; -1\}$ classification case for weight learning: for each object $x_{(k)}$ change the weight vector:

if $\langle w^{[k]} x_{(k)} \rangle y_{(k)} < 0$ then $w^{[k+1]} := w^{[k]} + \eta x_{(k)} y_{(k)}$.

Novikov's theorem

Theorem (Novikov)

Let sample T^ℓ be linearly separable: $\exists \widetilde{w}, \exists \delta > 0$:

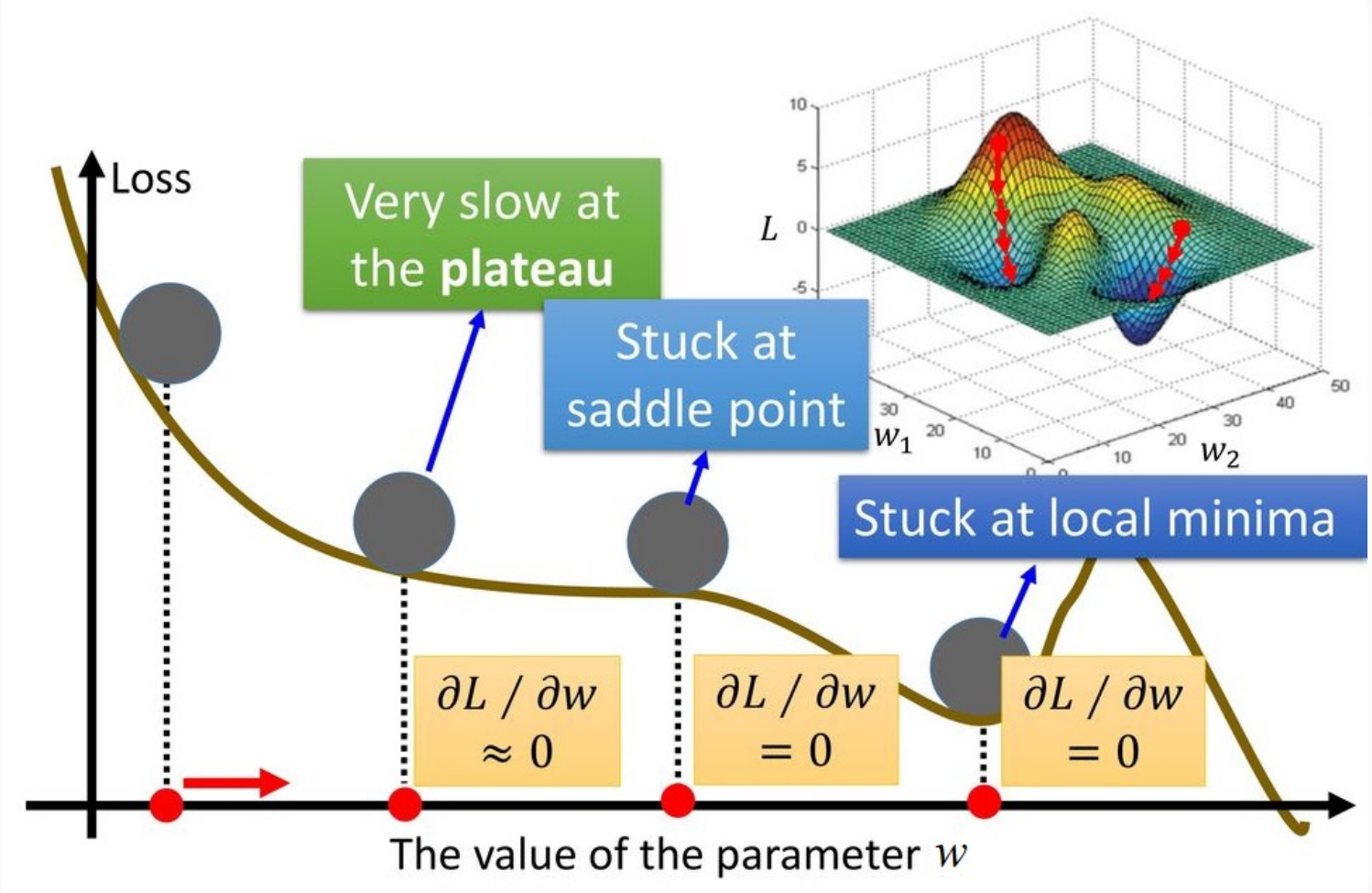
$\langle \widetilde{w}, x_i \rangle y_i > \delta$ for all $i = 1, \ldots, \ell$.

Then the stochastic gradient descent with Hebb's rule will find weight vector w, which:

- splits sample without error;
- with any initial guess $w^{[0]}$;
- with any learning rate $\mu > 0$;
- independently on objects ordering $x_{(i)}$;
- with finite numbers of changing vector w;
- if $w^{[0]} = 0$, then the number of changes in vector w is

$t_{\max} \leq \frac{1}{\delta^2} \max ||x_j||$.

Convergence problems

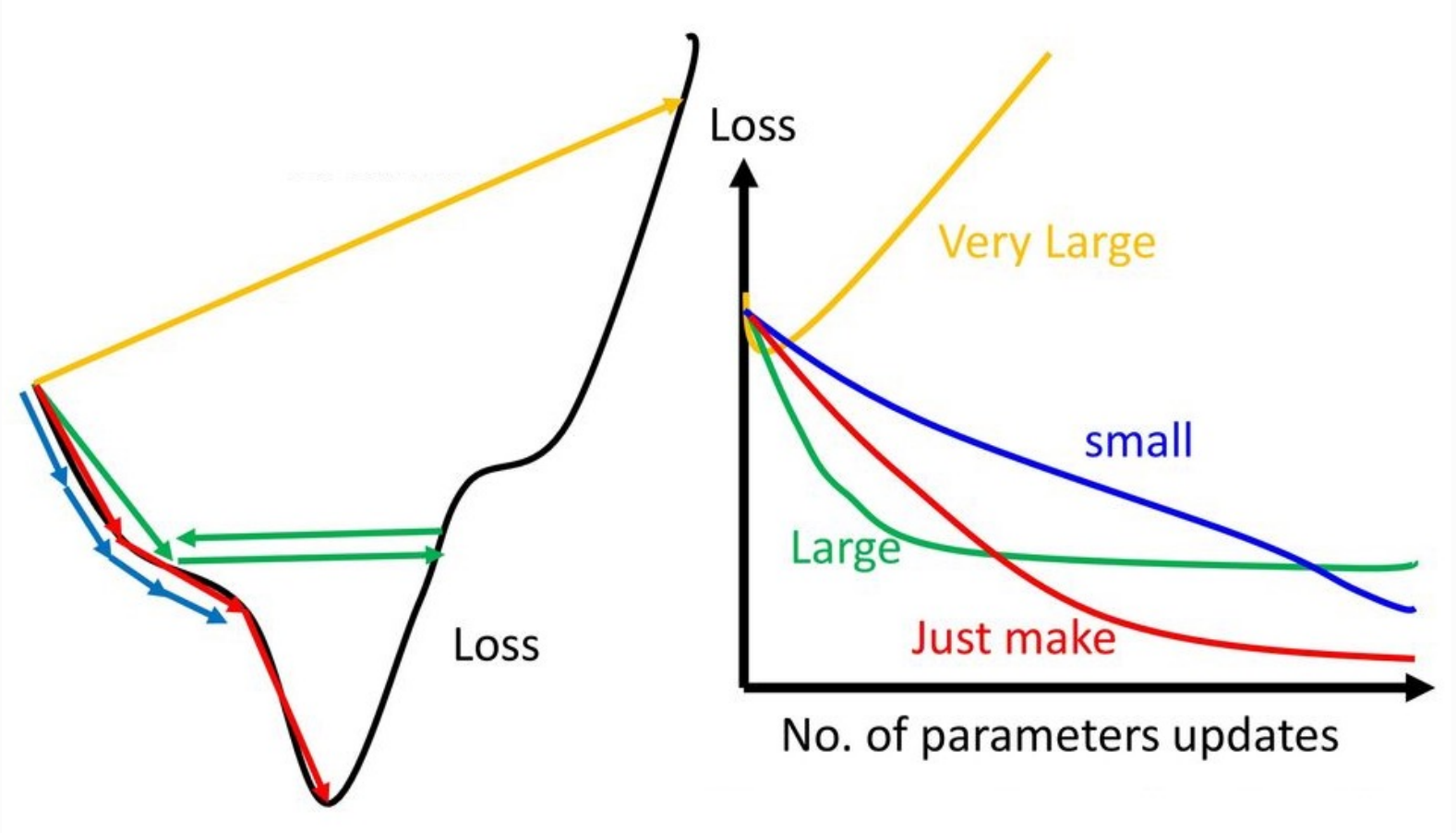


Heuristics for initial guesses

Important for non-convex functions

- $w_j = 0$ for all $j = 0, \ldots, n$;
- small random values: $w_j \in \left[-\frac{1}{2n}, \frac{1}{2n}\right]$;
- $w_j = \frac{\langle y, f_j \rangle}{\langle f_j, f_j \rangle}$;
- learn it with a small random subsample;
- multiple runs with different initial guesses.

Learning rates comparison



Heuristics for learning rate

- Convergence is achieved for convex functions when

$$\mu^{[k]} \to 0, \sum \mu^{[k]} = \infty, \sum \left(\mu^{[k]}\right)^2 < \infty$$

- **Steepest gradient descent**:

$$Q\left(w^{[k]} - \mu^{[k]} \nabla Q\left(w^{[k]}\right)\right) \to \min_{\mu^{[k]}}$$

- Steps for "jog of" local minima
- Second order methods
- Usage of mean vector of recent steps

Heuristics for object ordering

- take objects from different classes each step;
- take misclassified objects more frequently;
- do not take "good" object, such that $M_i > \kappa_+$;
- do not take noisy objects, such that $M_i < \kappa_-$.

SG algorithm discussion

Advantages:

- it is easy to implement;
- it is easy to generalize for any f and L;
- dynamical learning;
- can handle small samples.

Disadvantages:

- slow convergence or even divergence is possible;
- can stuck in local minima, saddle points;
- proper heuristic choice is very important;
- overfitting.

Lecture plan

- Linear classification
- Gradient descent
- Linear regression and matrix decomposition
- Regularization

Linear regression model

Model of multidimensional linear regression:

$$f(x, \theta) = \sum_{j=1}^{n} \theta_j f_j(x), \quad \theta \in \mathbb{R}^n.$$

Matrix notation:

$$F = \begin{pmatrix} f_1(x_1) & \dots & f_n(x_1) \\ \dots & \dots & \dots \\ f_1(x_\ell) & \dots & f_n(x_\ell) \end{pmatrix}, y = \begin{pmatrix} y_1 \\ \dots \\ y_\ell \end{pmatrix}, \theta = \begin{pmatrix} \theta_1 \\ \dots \\ \theta_n \end{pmatrix}.$$

Quality in matrix notation:

$$Q(\theta, T^\ell) = \sum_{i=1}^{\ell} (f(x_i, \theta) - y_i)^2 = ||F\theta - y||^2 \to \min_{\theta \in \mathbb{R}}.$$

Matrix decomposition

There are plenty of ways how it can be solved.

One of the most popular way is to apply singular **vector** decomposition, which is a **matrix decomposition** (a.k.a. **matrix factorization**) method.

Normal equation system

Minimum condition:

$$\frac{\partial Q(\theta)}{\partial \theta} = 2F^{\top}(F\theta - y) = 0.$$

$$\theta^* = (F^{\top}F)^{-1}F^{\top}y$$

$F^+ = (F^{\top}F)^{-1}F^{\top}$ is **pseudo reverse matrix** (**Moore–Penrose inverse**)

$P_F = FF^+$ is **projection matrix**

Solution:

$$\theta^* = F^+y.$$

Minimum approximation:

$$Q(\theta^*) = ||P_Fy - y||^2.$$

Singular vector decomposition

Theorem: any matrix F size of $\ell \times n$ can be represented with singular decomposition

$$F = VDU^{\top}.$$

With

- $V = (v_1, \dots, v_n)$ is size of $\ell \times n$ and orthogonal $V^{\top}V = I_n$, columns v_j are eigenvectors of matrix $FF^{\top}$;
- $U = (u_1, \dots, u_n)$ is size of $n \times n$ and ortogonal $U^{\top}U = I_n$, rows u_j are eigenvectors of matrix $F^{\top}F$;
- $D = \operatorname{diag}\left(\sqrt{\lambda_1}, \dots, \sqrt{\lambda_n}\right)$ is size of $n \times n$, $\sqrt{\lambda_j}$ are **singular numbers**, squares of eigenvalues of matrices $FF^{\top}$ and $F^{\top}F$.

SVD interpretation

You can think of some latent space, to which we want to project data.

D represents importance of each basis vector

V represents how objects correspond to the basis vectors

U represents how features correspond to the basis vectors

OLS with SVD

$$F^+ = (UDV^\top VDU^\top)^{-1}UDV^\top = UD^{-1}V^\top = \sum_{j=1}^{n} \frac{1}{\sqrt{\lambda_j}} u_j v_j^\top;$$

$$\theta^* = F^+ y = UD^{-1}V^\top y = \sum_{j=1}^{n} \frac{1}{\sqrt{\lambda_j}} u_j (v_j^\top y);$$

$$F\theta^* = P_F y = (VDU^\top)UD^{-1}V^\top y = VV^\top y = \sum_{j=1}^{n} v_j (v_j^\top y);$$

$$||\theta^*||^2 = ||D^{-1}V^\top y||^2 = \sum_{j=1}^{n} \frac{1}{\lambda_j} \left(v_j^\top y\right)^2.$$

Discussion

- When we can compute SVD, we can easily find solution for OLS.
- SVD is computations are quite heavy still, its complexity is $O(\ell^2 n + n^3)$

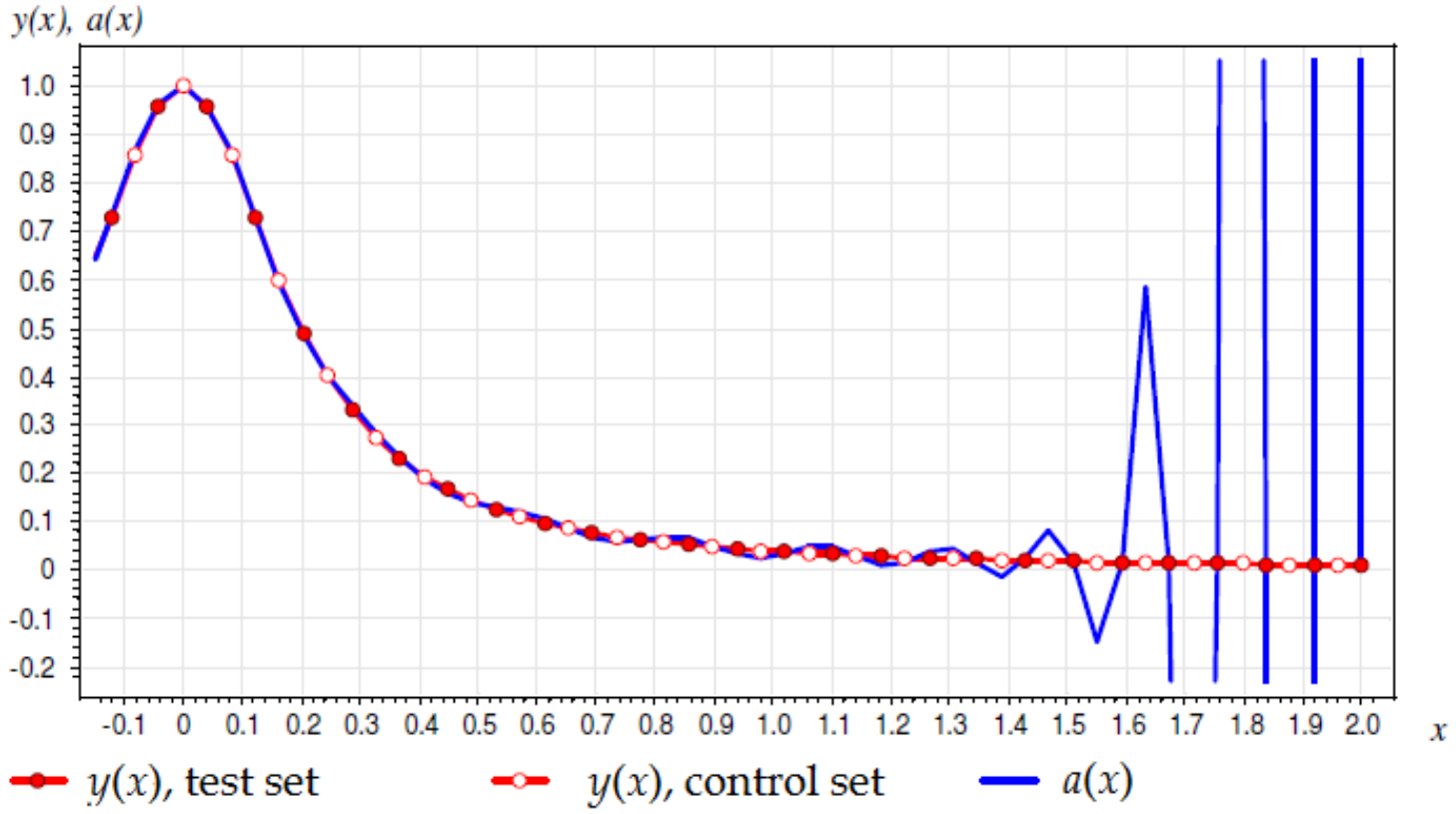
- SVD is important in many other machine learning tasks, first of all, in dimensionality reduction.

Lecture plan

- Linear classification
- Gradient descent
- Linear regression and matrix decomposition
- Regularization

Overfitted algorithm (reminder)

$y(x) = \dfrac{1}{1 + 25x^2}$; $a(x)$ — polynomial of degree $n = 38$



Regularization

Key hypothesis: w "swings" during training causing overfitting

Main idea: clip w norm.

Add regularization penalty for weights norm:

$$Q_\tau(a_w, T^\ell) = Q(a_w, T^\ell) + \frac{\tau}{2}\|w\|^2 \to \min_w.$$

τ is a coefficient representing the strength of model regularization, or, equally, is a trade-off between model performance and generalizability.

Ridge regression

Assumption: values of θ have Gaussian distribution with covariance matrix σI_n:

$$Q_\tau(\theta) = ||F\theta - y||^2 + \frac{1}{2\sigma}||\theta||^2 \to \min_\theta,$$

where $\tau = 1/\sigma$ is regularization penalty.

It can be simply added to OLS solution:

$$\theta_\tau^* = (F^\top F + \tau I_n)^{-1} F^\tau y.$$

Solution for ridge regression

$$\theta_\tau^* = U(D^2 + \tau I_n)^{-1} D V^\top y = \sum_{j=1}^{n} \frac{\sqrt{\lambda_j}}{\lambda_j + \tau} u_j (v_j^\top y);$$

$$F\theta_\tau^* = (VDU^\top)\theta_\tau^* = V \operatorname{diag}\left(\frac{\lambda_j}{\lambda_j + \tau}\right) V^\top y =$$

$$= \sum_{j=1}^{n} \frac{\lambda_j}{\lambda_j + \tau} v_j (v_j^\top y);$$

$$||\theta^*||^2 = ||D^2(D^2 + \tau I_n)^{-1} D^{-1} V^\top y||^2 = \sum_{j=1}^{n} \frac{1}{\lambda_j + \tau} (v_j^\top y)^2.$$

Tibshirani lasso

Assumption: values of vector θ has Laplacian distribution:

$$\begin{cases} Q_\tau(\theta) = ||F\theta - y||^2 \to \min\limits_{\theta}; \\ \sum\limits_{i=1}^{n} |\theta_i| \le \kappa. \end{cases}$$

LASSO (least absolute shrinkage and selection operator). Will lead to feature selection.

LASSO regression

The resulting optimization problem

$$Q_\tau(\theta) = ||F\theta - y||^2 + \tau\|\theta\|_1 \to \min_\theta,$$

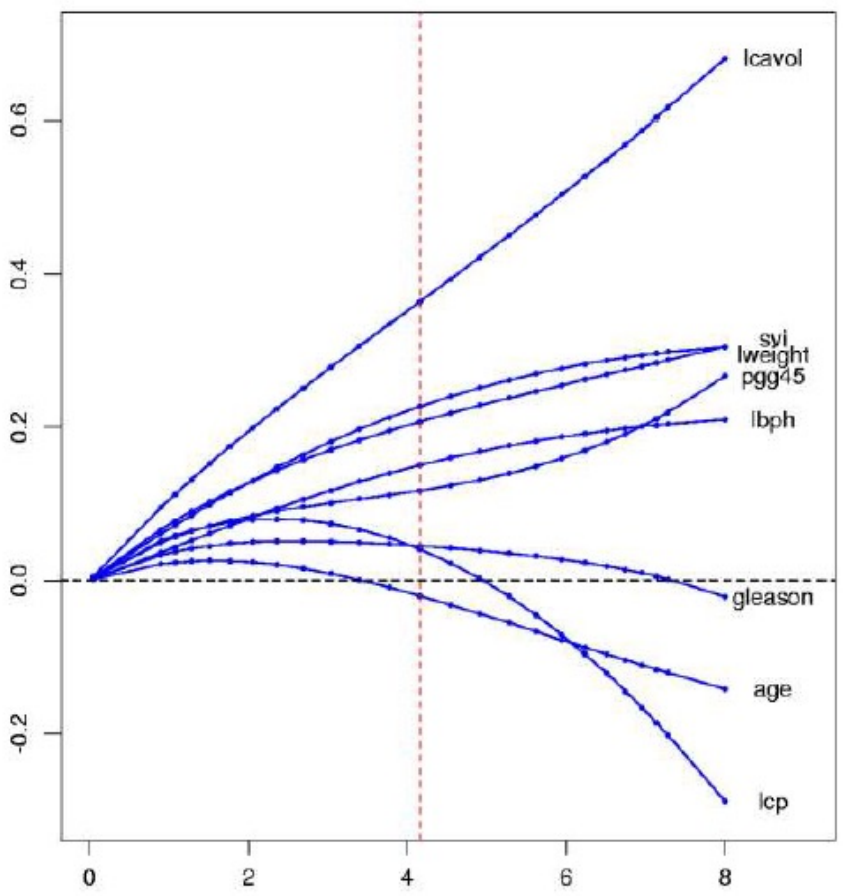
where $\|\theta\|_1$ is l_1-norm:$\|\theta\|_1 = \sum|\theta_i|$.

No nice analytical solution exist.

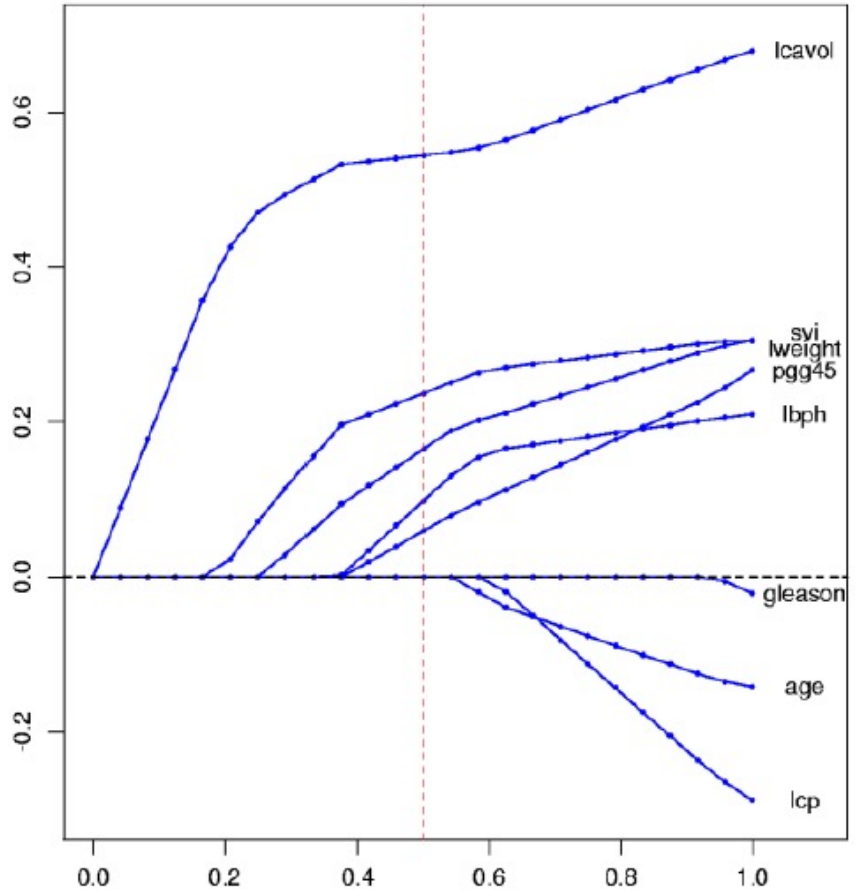
However, a nice computational solution exist.

Comparison

Ridge regression



Lasso



Regularization for gradient descent

Regularization is simply adopted to any gradient descent:

$$\nabla Q_\tau(w) = \nabla Q(w) + \tau w,$$
$$w^{[k+1]} = w^{[k]}(1 - \mu\tau) - \mu\nabla Q(w).$$

Regularization discussion (1/2)

- l_1-norm and l_2-norm regularizers are the most popular
- **ElasticNet**, which is sum of the previous two is also popular
- Many other may be used with respect to initial assumptions
- Some techniques are de-facto regularization or can be interpreted as regularization

Regularization discussion (2/2)

- One of two general approaches to fight overfitting
- Required to solve ill-stated problems
- Regularization penalties can be simply added to represent properties you expect from model

- Regularization coefficient should be tuned
- Not always clean, how to choose regularization