Lecture 3

Linear methods

Machine Learning

Ivan Smetannikov

03.07.2019

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 $a_w(x, T^{\ell}) = \operatorname{sign}\big(f(x, w)\big)$.

$f(x, w)$ 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.

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?

Similar idea, different implementation.

Loss function smoothing

Empirical risk:

$$Q(a_w, T^\ell) = Q(w) = \sum_{i}^{\ell} [M_i(w) < 0],$$

it is just the number of errors.

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

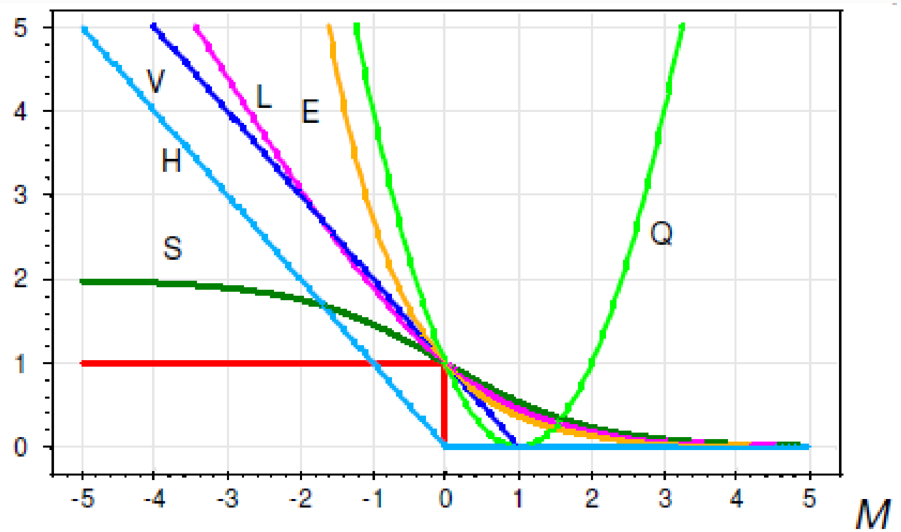
Approximation:

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

where $L(M_i(w)) = L(a_w(x_i, T^\ell), x_i)$ 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

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

Linear classifier:

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

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

Equivalent notation:

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

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

Beyond sign

Can we use something except sign?

What for, we are doing classification, don't we?

In fact, we can. That would help to get more rich prediction. We will talk about that when we discuss logistic regression.

How to learn a linear classifier?

We have to choose parameter vector w.

We can use almost any optimization algorithm capable of optimizing empirical risk in the corresponding space.

What is better to use?

How to learn a linear classifier?

We have to choose parameter vector w.

We can use almost any optimization algorithm capable of optimizing empirical risk in the corresponding space.

What is better to use?

The empirical risk is not a black-box function.

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

We can try to use **gradient descent**!

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, which should be estimated on each step.

Stochastic gradient descent:

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

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

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

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

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 **batch size**

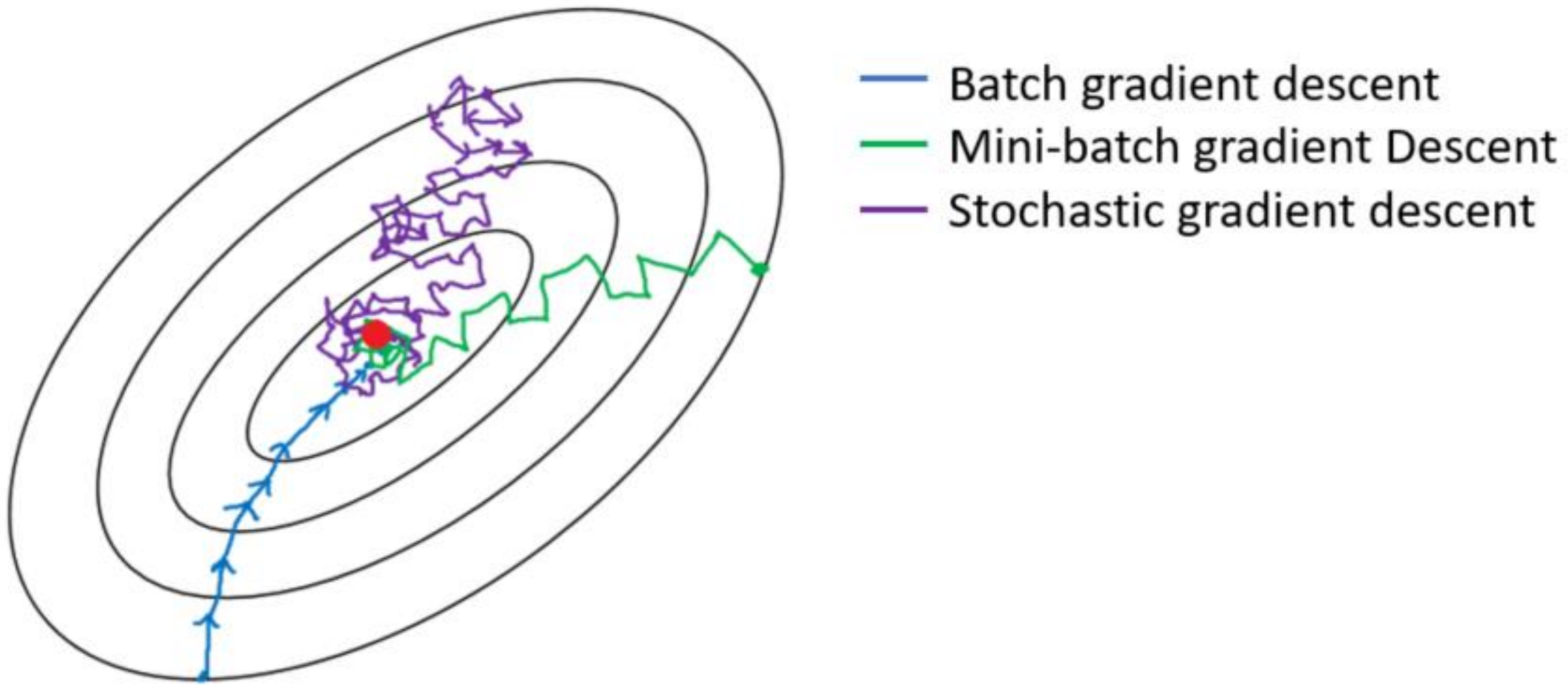
$x_{(1)}, \dots, x_{(\ell)}$ is **an objects order**;

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

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

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

Comparison



Batch gradient descent
Mini-batch gradient Descent
Stochastic gradient descent

Mini-batch gradient descent

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

Mini-batch gradient descent:

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

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

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

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

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

Novikov's theorem

Theorem (Novikov)

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

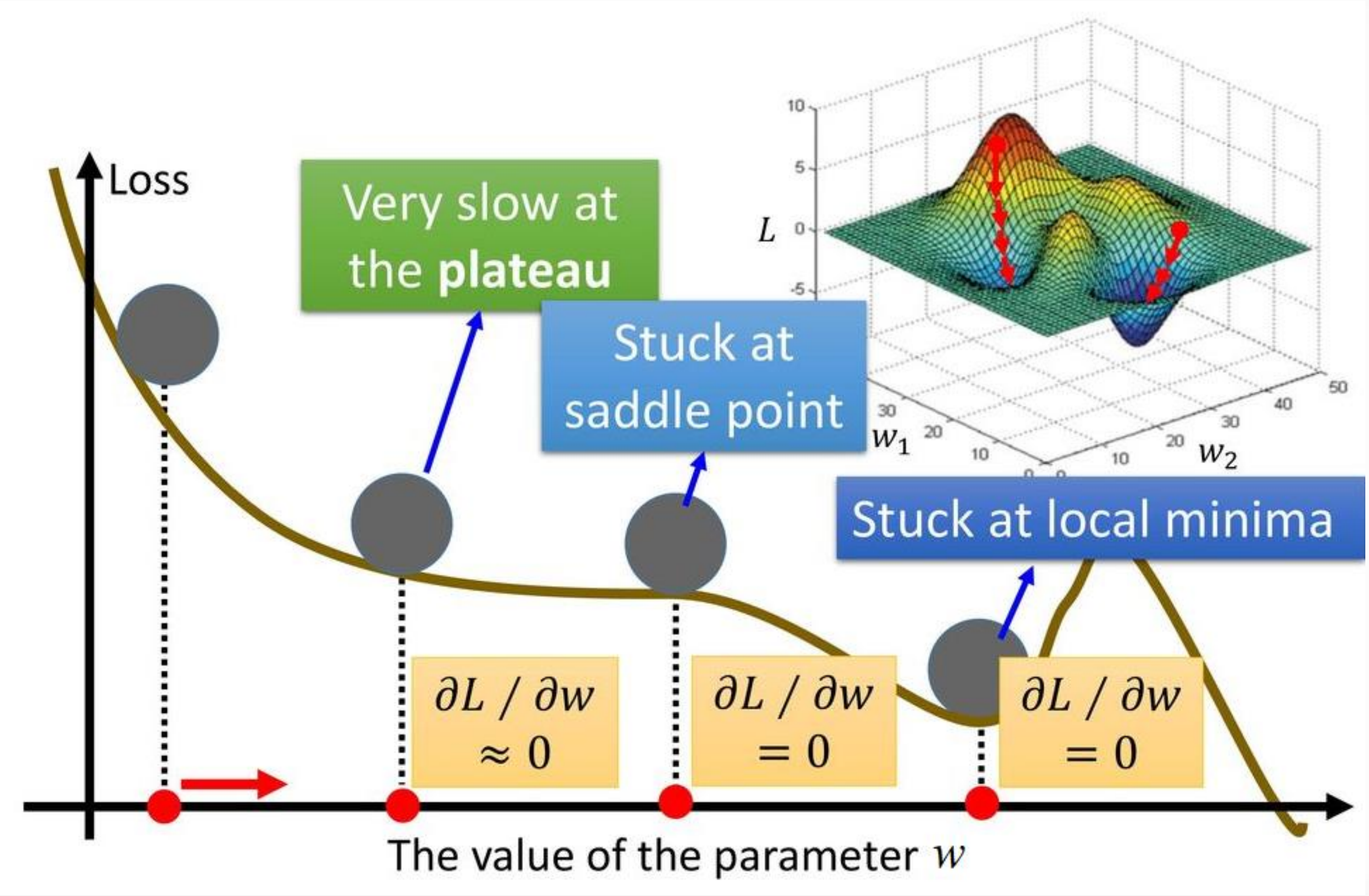
$\langle \widetilde{w}, x_i \rangle y_i > \delta$ for all $i = 1, \ldots, \ell$.

Them 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

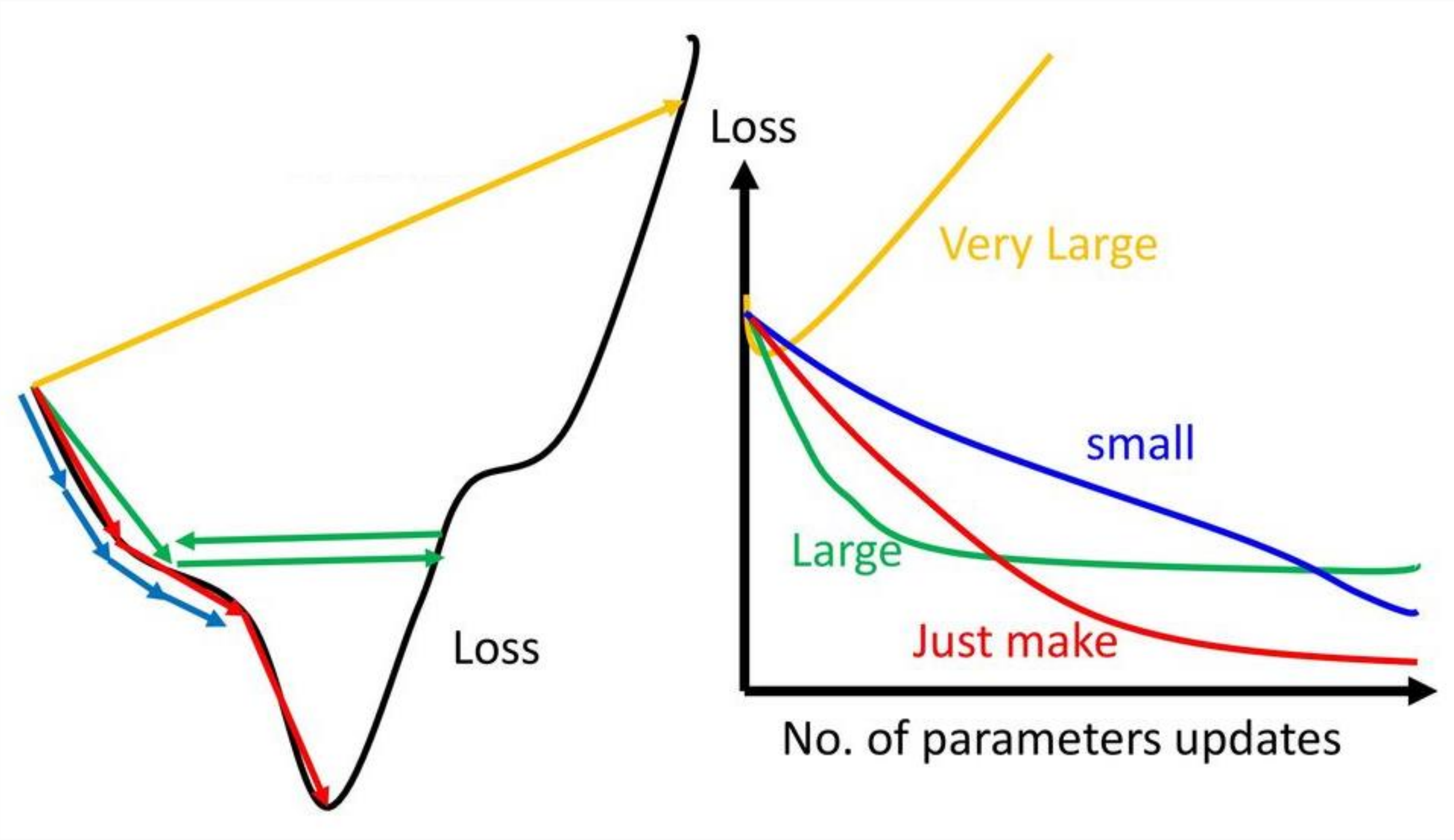


Heuristics for initial guesses

Important for non-convex functions

- $w_j = 0$ for all $j = 0, \dots, n$;
- 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;
- multiply 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 to “jog of” local minima.

Heuristics for object ordering

- take objects from different classes by turns;
- take misclassified objects more frequently;
- do not take “good” objects, 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.

SVD 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.

Linear method discussion

Advantages:

- Simple to implement
- Fast
- Interpretable and provide feature importance

Disadvantages:

- Do not represent complex relationships
- Prone to get overfitted

Lecture plan

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

Regularization for regression

Key hypothesis: *w* "swings" during overfitting. This is because of multicollinearity which arises between different features with the growth of the number of features.

Main idea: clip *w* norm.

Add regularization penalty for weights norm:

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

33

Regularization examples

For linear models $A = \{a(x) = \langle w, x\rangle\}$ (regression) and $A = \{a(x) = \operatorname{sign}\langle w, x\rangle\}$ (classification).

L_2-regularization (ridge regression, weight decay):

$$\text{penalty}(A) = \tau\|w\|_2^2 = \tau\sum w_i^2.$$

L_1−regularization (LASSO):

$$\text{penalty}(A) = \tau\|w\|_1 = \tau\sum|w_i|.$$

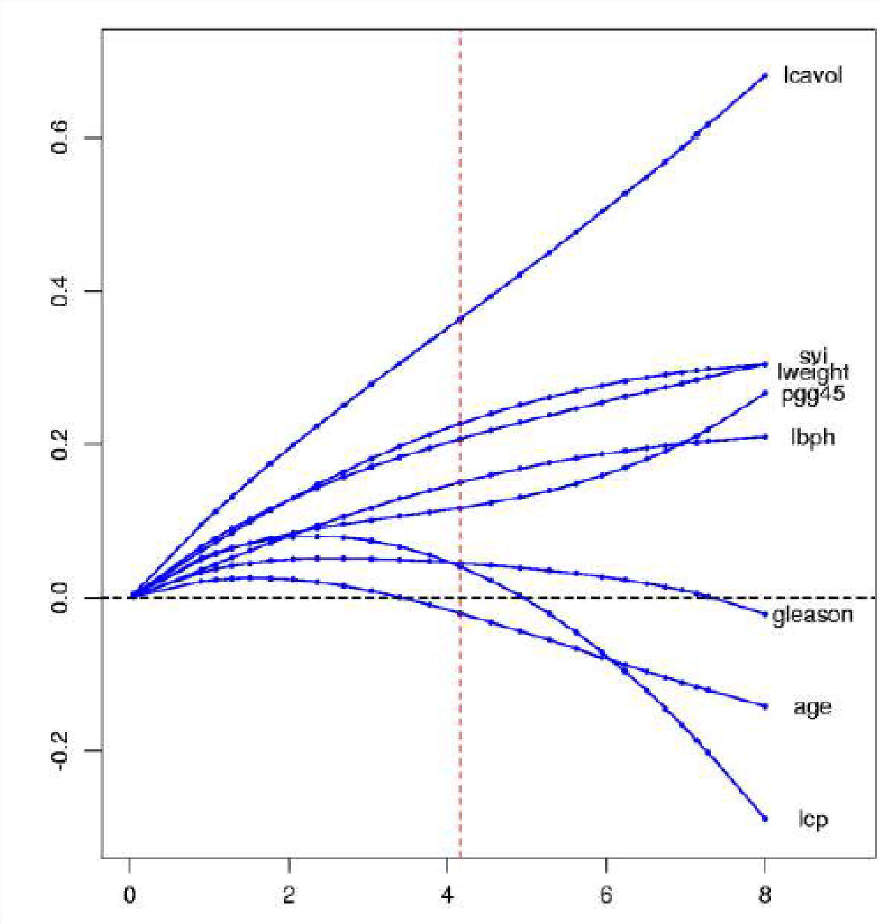
L_0-regularization (AIC, BIC):

$$\text{penalty}(A) = \tau\|w\|_o = \tau\sum[w_i \neq 0].$$

34

Comparison

Ridge regression



Lasso