Lecture 3

Linear models

Machine Learning

Sergey Muravyov

17.10.2017

Lecture plan

- Linear problems
- Gradient descent
- Linear regression and perceptron
- Regularization
- Support vector machine
- Kernel trick
- Support Vector Regression

- The presentation is prepared with materials of the K.V. Vorontsov's course "Machine Leaning".

Lecture plan

- Linear problems
- Gradient descent
- Linear regression and perceptron
- Regularization
- Support vector machine
- Kernel trick
- Support Vector Regression

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:

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

Problem formulation

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

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

Find classifier $a_w\left(x, T^\ell\right) = \text{sign}\left(f(x, w)\right)$.

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

w — parameters 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.

We have already 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 weight function of u's ith neighbor importance.

Empirical risk

Empirical risk:

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

It is just the number of errors.

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

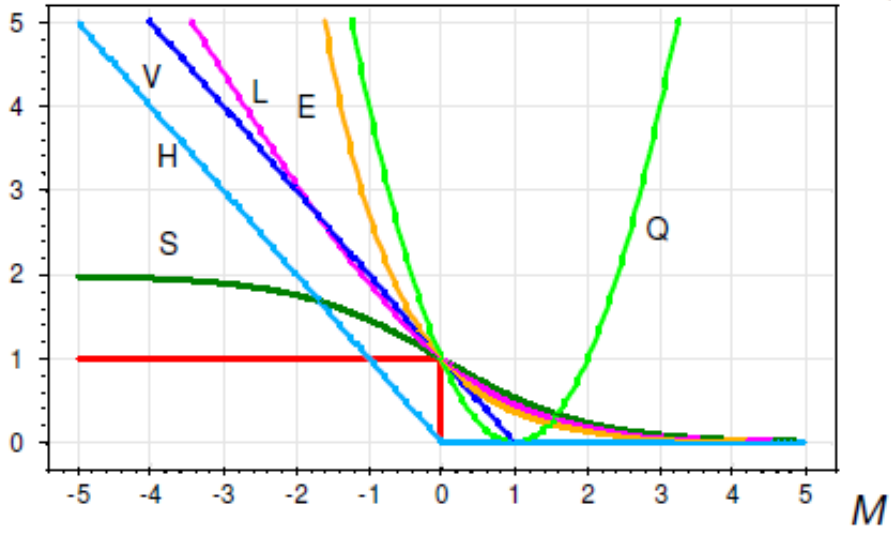
Approximation:

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

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

Loss function

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 model

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

Linear classifier:

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

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

Equivalent notation:

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

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

Lecture plan

- Linear problems
- Gradient descent
- Linear regression and perceptron
- Regularization
- Support vector machine
- Kernel trick
- Support Vector Regression

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:

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

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

where μ is **a gradient step**.

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

Stochastic gradient descent

Problem is that there are too many objects with should be estimated on each step.

Stochastic gradient descent:

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

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

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

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

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

Implementations of SGD

Python: **SGDClassifier** with loss stands for a loss function

WEKA: **SGD** with –F for a loss function,
–L for learning rate,
–E for the number of epochs

Heuristics for initial guesses

- $w_j = 0$ 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;
- multiply runs with different initial guesses.

Heuristics for object ordering

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

Heuristics for gradient descent step

- 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.

SG algorithm discussion

Advantages:

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

Disadvantages:

- slow convergence or even divergence is possible;
- can be “stuck” in local minima;
- proper heuristic choice is very important;
- overfitting.

Lecture plan

- Linear problems
- Gradient descent
- Linear regression and perceptron
- Regularization
- Support vector machine
- Kernel trick
- Support Vector Regression

Linear regression

Python: **LinearRegression**

WEKA: none, see the regularized version

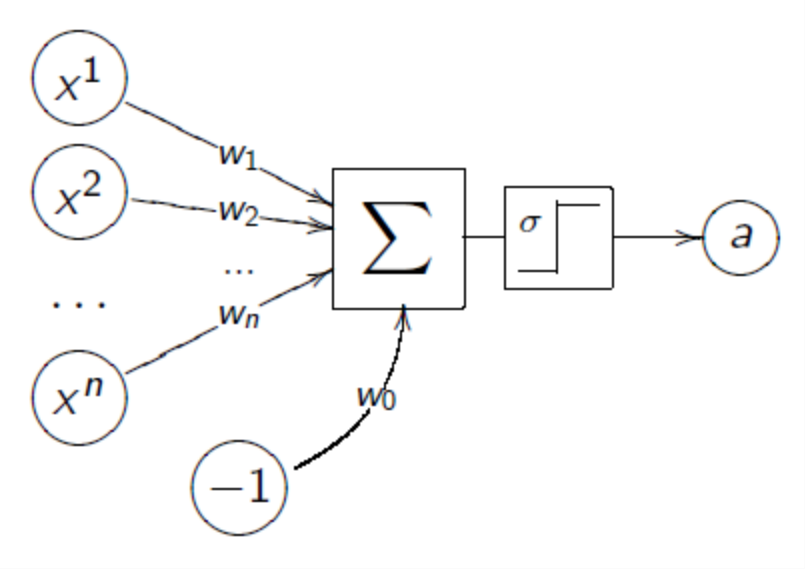
In fact, it is evaluated by applying Singular Vector Decomposition (SVD).

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.

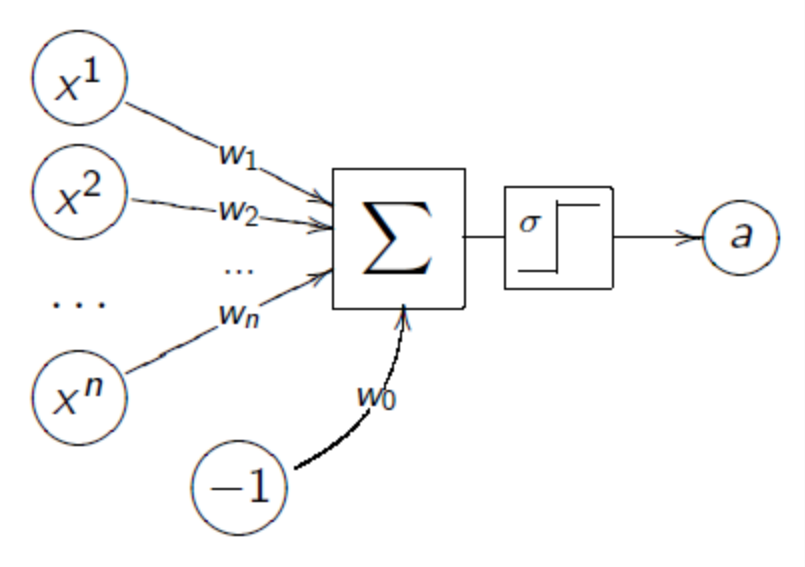


Perceptron

Rosenblatt's perceptron:

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

where σ is an activation function.



What is the difference?

Historically, neuron was created earlier than perceptron, the latter was more general model.

Mathematically, neuron solved binary classification problem with labels $\{-1, +1\}$ while perceptron solved binary classification problem with labels {0,1}.

Today, they are synonyms.

Learning neuron and perceptron

Special case of gradient descent

$$L(a_w, x) = (-\langle w, x\rangle y)_+,$$

where $(s)_+ = s \cdot [s < 0]$.

Hebb's rule (delta rule):

gradient descent step is

if $-\langle w^{[k]}, x_i\rangle y_i > 0$, then $w^{[k]} = w^{[k]} + \mu x_i y_i$.

Rosenblatt perceptron:

$$w^{[k]} = w^{[k]} + \mu(\text{sign}(\langle w, x_i\rangle) - y_i)x_i$$

(the same, when $Y = \{0,1\}$).

Perceptron implementation

Python: **Perceptron**

Weka: none, can be used **MultilayerPerceptron** with –H set to be 0.

Lecture plan

- Linear problems
- Gradient descent
- Linear regression and perceptron
- Regularization
- Support vector machine
- Kernel trick
- Support Vector Regression

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:

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

Regularization examples

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

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

Ridge regression

$$Q\left(a_w, T^{\ell}\right) + \frac{1}{2\sigma}||w||^2 \to \min_w$$

Python: **Ridge**, where alpha stands for $\frac{1}{2\sigma}$.

Weka: **LinearRegression**, where -R stands for $\frac{1}{2\sigma}$.

LASSO regression

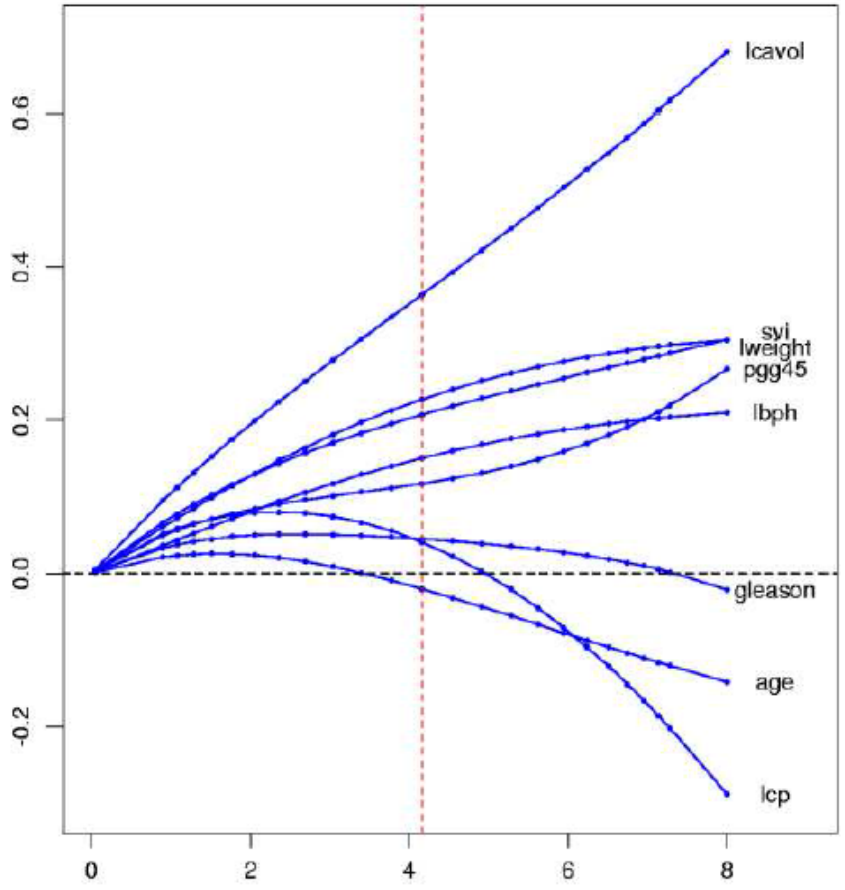
$$Q\left(a_w, T^\ell\right) + \kappa|w| \to \min_w$$

Python: **Lasso**, where alpha stands for κ.

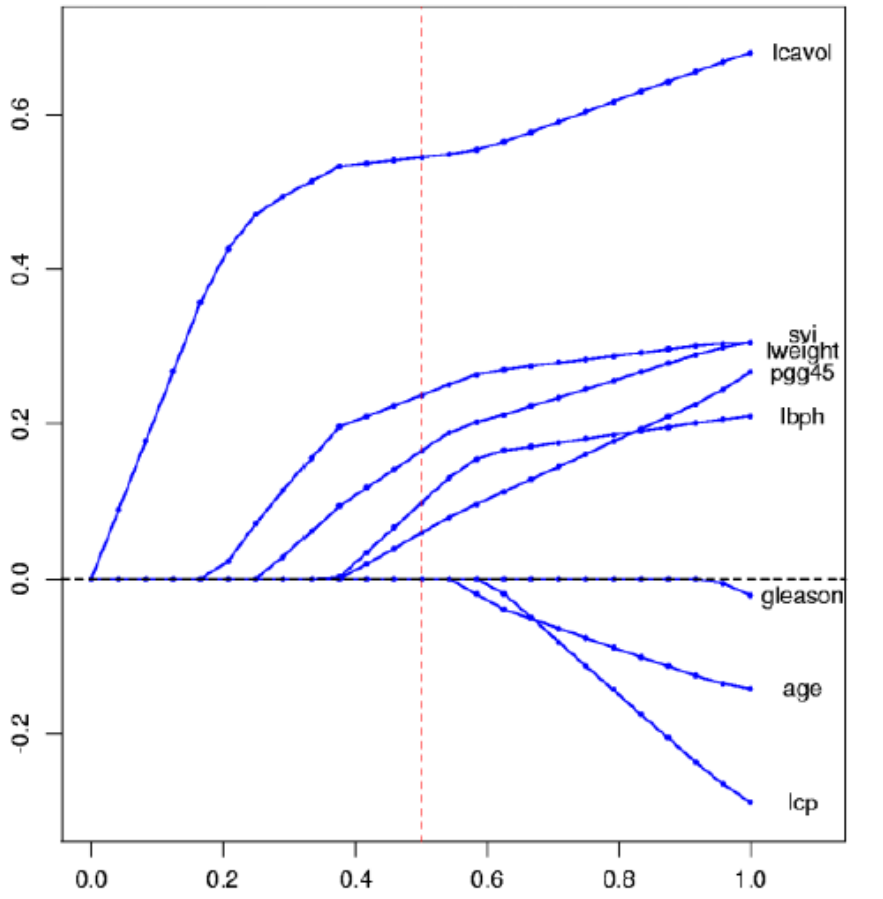
Weka: none, but has implemented greedy feature selection.

Comparison

Ridge regression



Lasso



Regularization for SGD

Add regularization penalty for weights norm:

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

For gradient:

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

Python: **SGDClassifier** with penalty stands for penalty.

WEKA: **SDG** with **–R** for regularization constant.

Lecture plan

- Linear problems
- Gradient descent
- Linear regression and perceptron
- Regularization
- Support vector machine
- Kernel trick
- Support Vector Regression

Basic idea

Basic idea: if we say that classifier is linear, what then is the best way to define it?

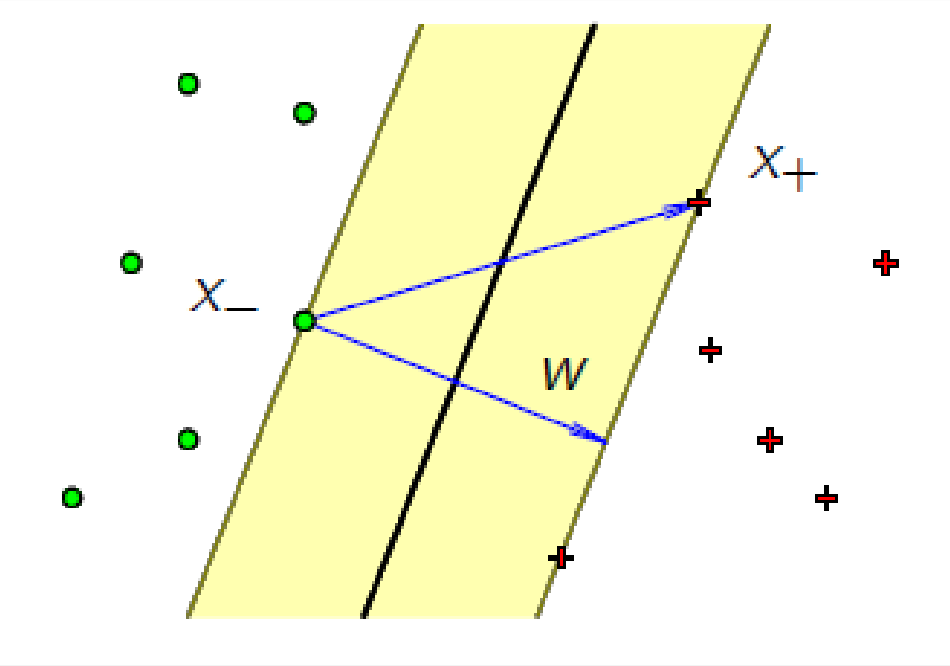
Main idea: search for a surface that is the most distant from the classes (large margin classification).

Linearly separable case

Key hypothesis: sample is linearly separable:

$$\exists w, w_0 : M_i(w, w_0) = y_i(\langle w, x_i \rangle - w_0) > 0, i = 1, \ldots, \ell.$$

Separating lines exist, therefore such a line that has maximum distance from both the classes also exists.



Separating stripe

Normalize margin:

$$\min_i M_i(w, w_0) = 1.$$

Separating stripe:

$$\{x: -1 \le \langle w, x \rangle - w_0 \le 1\}.$$

Stripe width:

$$\frac{\langle x_+ - x_-, w \rangle}{||w||} = \frac{(\langle x_+, w \rangle - w_0) - (\langle x_-, w \rangle - w_0)}{||w||} = \frac{2}{||w||}.$$

It turns to be a minimization problem:

$$\begin{cases} ||w||^2 \to \min_{w, w_0}; \\ M_i(w, w_0) \ge 1, i = 1, \dots, \ell. \end{cases}$$

Linearly inseparable case

Key hypothesis: sample is not linearly separable:

$$\forall w, w_0 \; \exists x_d \colon M_d(w, w_0) = y_d(\langle w, x_d \rangle - w_0) < 0$$

There is no such separating lines, but we can still try to find a line with the smallest margins for each object.

Linearly inseparability

In case of linearly inseparable sample:

$$\begin{cases} \frac{1}{2}||w||^2 + C\sum_{i=1}^{\ell}\xi_i \to \min_{w,w_0,\xi}; \\ M_i(w,w_0) \geq 1 - \xi_i, i = 1,\dots,\ell; \\ \xi_i \geq 0, \qquad i = 1,\dots,\ell. \end{cases}$$

ξ_i are **slack variables.**

Equivalent problem of unconditional optimization problem:

$$\sum_{i=1}^{\ell}\left(1 - M_i(w,w_0)\right)_+ + \frac{1}{2C}||w||^2 \to \min_{w,w_0}.$$

This is approximated empirical risk.

Dual problem

Lagrangian

$$\mathcal{L}(w, w_0; \mu, \lambda) = \frac{1}{2}||w||^2 - \sum_{i=1}^{m} \mu_i (M_i(w, w_0) - 1) - \sum_{j=1}^{k} \xi_j (\mu_i + \lambda_i - C)$$

λ_i are variables, dual for constraints $M_i \geq 1 - \xi_i$;

μ_i are variables, dual for constraints $\xi_i \geq 0$.

Condition of minimum:

$$\begin{cases} \frac{\delta \mathcal{L}}{\delta w} = 0; \frac{\delta \mathcal{L}}{\delta w_0} = 0; \frac{\delta \mathcal{L}}{\delta \xi} = 0; \\ \xi_i \geq 0; \lambda_i \geq 0; \mu_i \geq 0; \\ \lambda_i = 0 \text{ or } M_i(w, w_0) = 1 - \xi_i; \\ \mu_i = 0 \text{ or } \xi_i = 0; \end{cases}$$

$i = 1, \ldots, m$.

Support vectors

Object types:

1. $\lambda_i = 0; \mu_i = C; \xi_i = 0; M_i > 1$

peripheral objects.

2. $0 < \lambda_i < C; 0 < \mu_i < C; \xi_i = 0; M_i = 1$

support boundary objects.

3. $\lambda_i = C; \mu_i = 0; \xi_i > 0; M_i < 1$

support-disturbers.

Object x_i is **support** object, if $\lambda_i \neq 0$.

Non-linear programming problem

$$-\mathcal{L}(\lambda) = -\sum_{i=1}^{\ell}\lambda_i + \frac{1}{2}\sum_{i=1}^{\ell}\sum_{j=1}^{\ell}\lambda_i\lambda_j y_i y_j \langle x_i, x_j\rangle \to \min_{\lambda}$$

$$\begin{cases} 0 \le \lambda_i \le C; \\ \sum_{j=1}^{\ell}\lambda_i y_i = 0. \end{cases}$$

Primal problem solution can be expressed with dual problem solution:

$$\begin{cases} w = \sum_{i=1}^{\ell}\lambda_i y_i x_i\,; \\ w_0 = \langle w, x_i\rangle - y_i. \end{cases} \quad \forall i\colon \lambda_i > 0, M_i = 1.$$

Linear classifier:

$$a(x) = \operatorname{sign}\left(\sum_{i=1}^{\ell}\lambda_i y_i \langle x_i, x\rangle - w_0\right).$$

Lecture plan

- Linear problems
- Gradient descent
- Linear regression and perceptron
- Regularization
- Support vector machine
- Kernel trick
- Support Vector Regression

Kernel trick

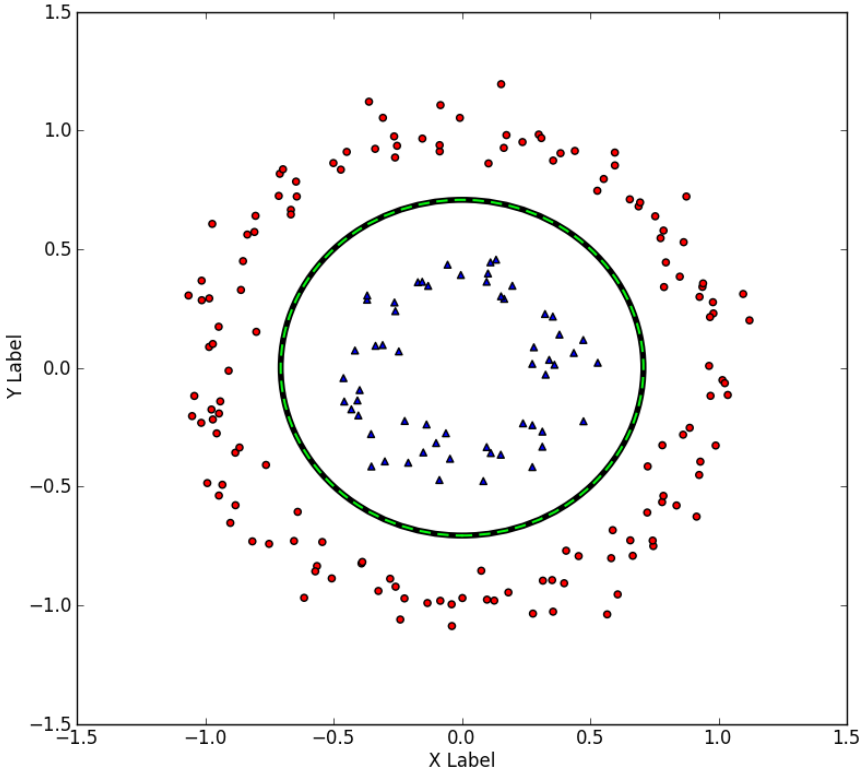
Main idea: find such a mapping to higher-dimensional space, that the points in new space will be linearly separable.

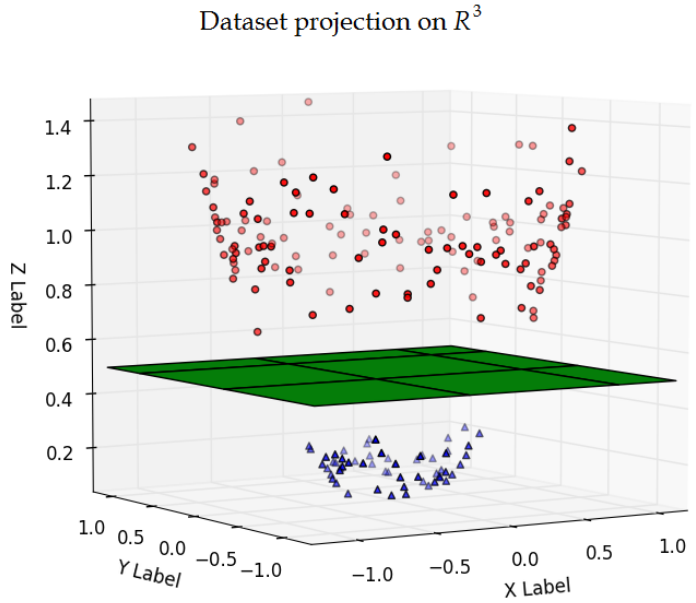
Idea basis: let separating surface can be well approximated by sum of functions depend on $x_1, \dots, x_n$:

$$c_1 x_1 + \cdots + c_n x_n + f_1(x_1, \dots, x_n) + \cdots + f_k(x_1, \dots, x_n)$$

If we add features $f_1(x_1, \dots, x_n)$, …, $f_k(x_1, \dots, x_n)$, than we would have new space $x_1, \dots, x_n, x_{n+1}, \dots, x_{n+k}$, in which it points would be linearly separable.

Example


1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
Y Label
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
X Label


Dataset projection on R^3
1.4
1.2
1.0
0.8
0.6
0.4
0.2
Z Label
1.0
0.5
0.0
−0.5
−1.0
Y Label
−1.0
−0.5
0.0
0.5
1.0
X Label

Why kernels?

We can build distance-based classifier for support objects (vectors). Usage of kernel function is equal to using a certain mapping.

Main problem is to find kernel which maps our space into linearly separable.

Function $K: X \times X \to \mathbb{R}$ is **kernel function**, if it can be represented as $K(x, x') = \langle \psi(x), \psi(x') \rangle$ with a mapping $\psi: X \to H$, where H — space with a scalar product.

Typical kernels

- Linear:

$$\langle x, x' \rangle$$

- Polynomial:

$$(\gamma \langle x, x' \rangle + r)^d$$

- RFB:

$$\exp(-\gamma \, |x - x'|^2)$$

- Sigmoid:

$$\tanh(\gamma \langle x, x' \rangle + r)$$

- Pearson VII universal function kernel:

$$\frac{1}{1 + \left(2\sqrt{(x - x')^2 \sqrt{2^{1/\omega} - 1}/\delta} \right)^{2\omega}}$$

SVM discussion

Advantages:

- Convex quadratic programming problem has a single solution
- Any separating surface
- Small number of support object used for learning

Disadvantages:

- Sensitive to noise
- No common rules for kernel function choice
- The constant C should be chosen
- No feature selection

SVM implementations

Python: **SVC** with kernel standing for kernel

NuSVC: has additional parameter nu, which is lower bound of the fraction of support vectors

LinearSVC: only linear kernels

Weka: **SMO** with –C corresponding to nu in NuSVC

–K standing for kernel

Lecture plan

- Linear problems
- Gradient descent
- Linear regression and perceptron
- Regularization
- Support vector machine
- Kernel trick
- Support Vector Regression

Support vector regression

Minimization problem

$$\begin{cases} \frac{1}{2}||w||^2 \to \min_{w,w_0}; \\ |M_i(w,w_0)| \leq \varepsilon. \end{cases}$$

Now we want a curve to be very close to all the objects and be regularized. ε is a threshold.

We can add slack variables and solve dual problem in the same way.

SVR implementation

Python: **SVR**

NuSVR

LinearSVR

KernelRigde (ridge regression + kernel trick)

Weka: **SMOreg** with the same parameters as **SMO** and

–I standing for optimization algorithm to be used.