

# Linear methods of classification

Victor Kitov  
[v.v.kitov@yandex.ru](mailto:v.v.kitov@yandex.ru)

Yandex School of Data Analysis



# Orthogonal vector to hyperplane

## Theorem 1

*Vector  $w$  is orthogonal to hyperplane  $w^T x + w_0 = 0$*

*Proof.* Consider arbitrary  $x_A, x_B \in \{x : w^T x + w_0 = 0\}$ :

$$w^T x_A + w_0 = 0 \quad (1)$$

$$w^T x_B + w_0 = 0 \quad (2)$$

By subtracting (2) from (1), obtain  $w^T(x_A - x_B) = 0$ , so  $w$  is orthogonal to hyperplane. □

## Distance from point to hyperplane

### Theorem 2

*Distance from point  $x$  to hyperplane  $w^T x + w_0 = 0$  is equal to  $\frac{w^T x + w_0}{\|w_0\|}$ .*

*Proof.* Project  $x$  on the hyperplane, let the projection be  $p$  and complement  $h = x - p$ , orthogonal to hyperplane. Then

$$x = p + h$$

Since  $p$  lies on the hyperplane,

$$w^T p + w_0 = 0$$

Since  $h$  is orthogonal to hyperplane and according to theorem 1

$$h = r \frac{w}{\|w\|}, \quad r \in \mathbb{R} \text{ - distance to hyperplane.}$$

## Distance from point to hyperplane

$$x = p + r \frac{w}{\|w\|}$$

After multiplication by  $w$  and addition of  $w_0$ :

$$w^T x + w_0 = w^T p + w_0 + r \frac{w^T w}{\|w\|} = r \|w\|$$

because  $w^T p + w_0 = 0$  and  $\|w\| = \sqrt{w^T w}$ . So we get, that

$$r = \frac{w^T x + w_0}{\|w\|}$$

### Comments:

- From one side of hyperplane  $r > 0 \Leftrightarrow w^T x + w_0 > 0$
- From the other side  $r < 0 \Leftrightarrow w^T x + w_0 < 0$ .
- Distance from hyperplane to origin 0 is  $\frac{w_0}{\|w\|}$ . So  $w_0$  accounts for hyperplane offset.

# Binary linear classifier geometric interpretation

Binary linear classifier:

$$\hat{y}(x) = \text{sign} \left( w^T x + w_0 \right)$$

divides feature space by hyperplane  $w^T x + w_0 = 0$ .

- $w^T x + w_0$  is the score of positive class vs. negative.
  - proportional to distance to hyperplane

# Table of Contents

- 1 Multiclass classification with binary classifiers
- 2 Gradient descent optimization
- 3 Regularization
- 4 Logistic regression

# Multiclass classification with binary classifiers

- Task - make  $C$ -class classification using many binary classifiers.
- Approaches:
  - **one-versus-all**
    - for each  $c = 1, 2, \dots, C$  train binary classifier on all objects and output  $\mathbb{I}[y_n = c]$ ,
    - assign class, getting the highest score in resulting  $C$  classifiers.
  - **one-versus-one**
    - for each  $i, j \in [1, 2, \dots, C]$ ,  $i \neq j$  learn on objects with  $y_n \in \{i, j\}$  with output  $y_n$
    - assign class, getting the highest score in resulting  $C(C - 1)/2$  classifiers.
- Ambiguity may occur.

# Linear classifier

- Classification among classes  $1, 2, \dots, C$ .
- Use  $C$  discriminant functions  $g_c(x) = w_c^T x + w_{c0}$
- Decision rule:

$$\hat{y}(x) = \arg \max_c g_c(x)$$

- Decision boundary between classes  $y = i$  and  $y = j$  is linear:

$$(w_i - w_j)^T x + (w_{i0} - w_{j0}) = 0$$

- Decision regions are convex<sup>1</sup>.

---

<sup>1</sup>why? prove that.



## Binary linear classifier

- For two classes  $y \in \{+1, -1\}$  classifier becomes

$$\hat{y}(x) = \begin{cases} +1, & w_{+1}^T x + w_{+1,0} > w_{-1}^T x + w_{-1,0} \\ -1 & \text{otherwise} \end{cases}$$

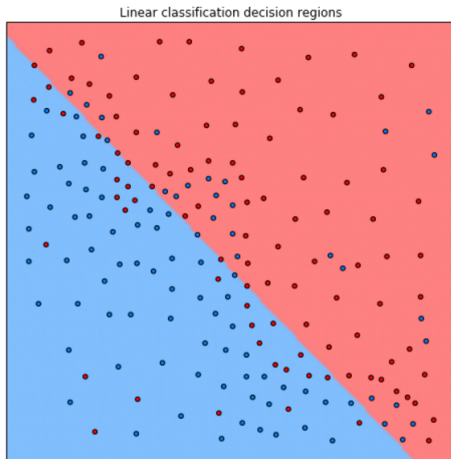
- This decision rule is equivalent to

$$\begin{aligned} \hat{y}(x) &= \text{sign}(w_{+1}^T x + w_{+1,0} - w_{-1}^T x + w_{-1,0}) = \\ &= \text{sign} \left( (w_{+1}^T - w_{-1}^T) x + (w_{+1,0} - w_{-1,0}) \right) \\ &= \text{sign} (w^T x + w_0) \end{aligned}$$

for  $w = w_{+1} - w_{-1}$ ,  $w_0 = w_{+1,0} - w_{-1,0}$ .

- Decision boundary  $w^T x + w_0 = 0$  is linear.
- Multiclass case can be solved using multiple binary classifiers with one-vs-all, one-vs-one or error correcting codes schemes.

## Example: linear decision region



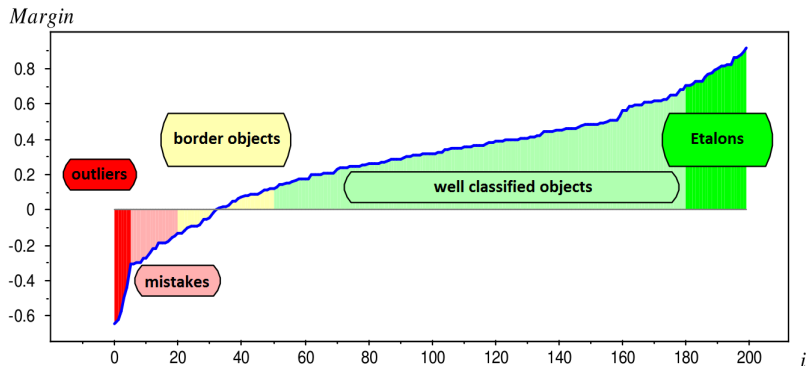
## Margin of binary linear classifier

$$\begin{aligned}M(x, y) &= g_y(x) - g_{-y}(x) = w_y^T x + w_{y,0} - w_{-y}^T x - w_{-y,0} \\&= [w_y - w_{-y}]^T x + [w_{y,0} - w_{-y,0}] \\&= y \left( [w_{+1} - w_{-1}]^T x + [w_{+1,0} - w_{-1,0}] \right) \\&= y \left( w^T x + w_0 \right)\end{aligned}$$

- Margin=score, how well classifier predicted true  $y$  for object  $x$ .
- $M(x, y|w) > 0 \iff$  object  $x$  is correctly classified as  $y$ 
  - signs of  $w^T x + w_0$  and  $y$  coincide
- $|M(x, y|w)| = |w^T x + w_0|$  - confidence of decision
  - proportional to distance from  $x$  to hyperplane  $w^T x + w_0 = 0$ .

# Margin

Objects, ordered by margin



## Redefinitions

- Add  $w_0$  to  $w = [w_1, \dots, w_D]^T$ :

$$w = [w_0, w_1, \dots, w_D]^T$$

- Add constant feature  $x_0 \equiv 1$  to  $x = [x^1, \dots, x^D]^T$ :

$$x = [1, x^1, \dots, x^D]^T$$

- Binary linear classifier becomes:

$$\hat{y}(x) = \text{sign}(w^T x)$$

- Margin becomes:

$$M(x, y|w) = w^T xy$$

# Weights optimization

- Margin=score, how well classifier predicted true  $y$  for object  $x$ .
- Task: select such  $w$  to increase  $M(x_n, y_n|w)$  for all  $n$ .
- Formalization:

$$\frac{1}{N} \sum_{n=1}^N \mathcal{L}(M(x_n, y_n|w)) \rightarrow \min_w$$

## Misclassification rate optimization

- Misclassification rate optimization:

# Misclassification rate optimization

- Misclassification rate optimization:

$$\frac{1}{N} \sum_{n=1}^N \mathbb{I}[M(x_n, y_n | w) < 0] \rightarrow \min_w$$



# Misclassification rate optimization

- Misclassification rate optimization:

$$\frac{1}{N} \sum_{n=1}^N \mathbb{I}[M(x_n, y_n | w) < 0] \rightarrow \min_w$$

is not recommended:

- discontinuous function, can't use numerical optimization!
- continuous margin is more informative than binary error indicator.

# Misclassification rate optimization

- Misclassification rate optimization:

$$\frac{1}{N} \sum_{n=1}^N \mathbb{I}[M(x_n, y_n | w) < 0] \rightarrow \min_w$$

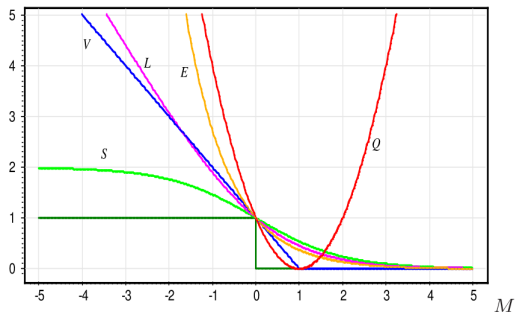
is not recommended:

- discontinuous function, can't use numerical optimization!
- continuous margin is more informative than binary error indicator.
- If we select loss function  $\mathcal{L}(M)$  such that  $\mathbb{I}[M] \leq \mathcal{L}(M)$  then we can optimize upper bound on misclassification rate:

$$\text{MISCLASSIFICATION RATE} = \frac{1}{N} \sum_{n=1}^N \mathbb{I}[M(x_n, y_n | w) < 0]$$

$$\leq \frac{1}{N} \sum_{n=1}^N \mathcal{L}(M(x_n, y_n | w)) = L(w)$$

# Common loss functions



$$\begin{aligned}
 Q(M) &= (1 - M)^2 \\
 V(M) &= (1 - M)_+ \\
 S(M) &= 2(1 + e^M)^{-1} \\
 L(M) &= \log_2(1 + e^{-M}) \\
 E(M) &= e^{-M}
 \end{aligned}$$

# Table of Contents

- 1 Multiclass classification with binary classifiers
- 2 Gradient descent optimization
- 3 Regularization
- 4 Logistic regression

# Gradient

- For any function  $f(x)$ , depending from  $x = (x_1, \dots, x_D)^T$  gradient

$$\nabla f(x) := \begin{pmatrix} \frac{\partial f(x)}{\partial x_1} \\ \frac{\partial f(x)}{\partial x_2} \\ \dots \\ \frac{\partial f(x)}{\partial x_D} \end{pmatrix}$$

- If function  $f(x, y)$  depends on other variables  $y$  gradient  $\nabla_x$  considers only derivatives with respect to  $x$ :

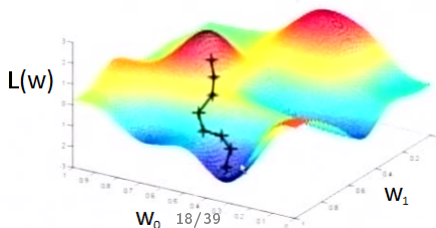
$$\nabla_x f(x, y) := \begin{pmatrix} \frac{\partial f(x)}{\partial x_1} \\ \frac{\partial f(x)}{\partial x_2} \\ \dots \\ \frac{\partial f(x)}{\partial x_D} \end{pmatrix}$$

## Gradient descend optimization

- Optimization task to obtain the weights:

$$L(w) = \sum_{i=1}^N \mathcal{L}(w^T x_i y_i) \rightarrow \min_w$$

- For convex  $\mathcal{L}(u)$   $L(w)$  will also be convex  $\Rightarrow$  method will converge to global optimum from any starting conditions.
- Gradient descend - iterative movement in direction of  $-\nabla_w F(w)$ .
- Example for  $w = (w_0, w_1)^T$ :



# Gradient descend optimization

**INPUT:**

$\eta$ : parameter, controlling the speed of convergence  
stopping rule

**ALGORITHM:**

initialize  $w_0$  randomly

**WHILE** stopping rule is not satisfied:

$$w_{n+1} \leftarrow w_n - \eta \nabla_w L(w_n)$$

$$n \leftarrow n + 1$$

**RETURN**  $w_n$

Any computational issues for big data?

# Gradient descend optimization

**INPUT:**

$\eta$ : parameter, controlling the speed of convergence  
stopping rule

**ALGORITHM:**

initialize  $w_0$  randomly

**WHILE** stopping rule is not satisfied:

$$w_{n+1} \leftarrow w_n - \eta \frac{1}{N} \sum_{i=1}^N \nabla_w \mathcal{L}(x_i, y_i | w_n)$$

$$n \leftarrow n + 1$$

**RETURN**  $w_n$

Gradient calculation requires  $O(N)$  operations!



# Stochastic gradient descent optimization

**INPUT:**

$\eta$ : parameter, controlling the speed of convergence  
stopping rule

**ALGORITHM:**

initialize  $w_0$  randomly

**WHILE** stopping rule is not satisfied:

    randomly sample  $I = \{i_1, \dots, i_K\}$  from  $\{1, 2, \dots, N\}$

$w_{n+1} \leftarrow w_n - \eta \frac{1}{K} \sum_{i \in I} \nabla_w \mathcal{L}(x_i, y_i | w_n)$

$n \leftarrow n + 1$

**RETURN**  $w_n$

## Stochastic gradient descent optimization

- Main idea: for random subsample  $I = \{i_1, \dots, i_K\}$ , called minibatch,

$$\frac{1}{N} \sum_{i=1}^N \mathcal{L}(x_i, y_i | w) \approx \frac{1}{K} \sum_{i \in I} \mathcal{L}(x_i, y_i | w), \quad K \ll N$$

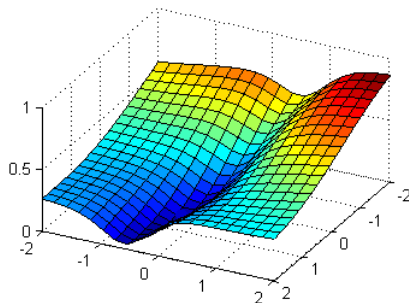
- Original method used  $K = 1$ .
- $K > 1$  gives smoother gradient.  $\frac{1}{K} \sum_{i \in I} \nabla_w \mathcal{L}(x_i, y_i | w_n)$  can still be computed in  $O(1)$  because processors internally perform vector arithmetics.
- SGD converges almost surely when  $\eta_n \rightarrow 0$  as  $n \rightarrow \infty$  at an appropriate rate.
- In practice  $\eta = \text{small const}$  or  $\eta_n = \frac{1}{n}$
- Indices generation: before each pass through the training set, it is randomly shuffled and then passed sequentially.

# Gradient descend optimization

- Possible stopping rules:
  - $|w_{n+1} - w_n| < \varepsilon$
  - $|L(w_{n+1}) - L(w_n)| < \varepsilon$
  - $n > n_{max}$
- For regression GD and SGD are also applicable:  
 $\mathcal{L}(M(x_n, y_n|w))$  replace with  $\mathcal{L}(w^T x_n - y_n)$ .

## Recommendations for use

- Convergence is faster for normalized features
  - feature normalization solves the problem of «elongated valleys»

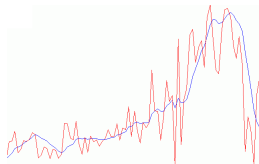


## Tracking convergence of SGD

- Estimation of  $L(w_n) = \frac{1}{N} \sum_{i=1}^N \mathcal{L}(x_i, y_i | w_n)$  on each iteration takes  $O(N)$  and is impractical.
- For series  $z_1, \dots, z_N$  exponentially smoothed series is obtained by

$$\begin{cases} s_1 = z_1 \\ s_{n+1} = \alpha z_{n+1} + (1 - \alpha) s_n \end{cases} \quad \begin{array}{l} \alpha \in (0, 1) - \text{hyperparameter} \\ \text{recalculation takes } O(1) \end{array}$$

Example: original (red) and exp-smoother (blue) time series:



## Tracking convergence of SGD

Exponential smoothing of loss enables loss reestimation in  $O(1)$ :

$$L_0^{smooth} = \sum_{i=1}^N \mathcal{L}(M(x_i, y_i | w_0))$$
$$L_{n+1}^{smooth} = \alpha \mathcal{L}(M(x_i, y_i | w_0)) + (1 - \alpha) L_n^{smooth}$$

# Discussion of SGD

## Advantages

- Simple
- Works online
- A small subset of learning objects may be sufficient for accurate estimation

# Discussion of SGD

## Advantages

- Simple
- Works online
- A small subset of learning objects may be sufficient for accurate estimation

## Drawbacks

- Optimization using 2nd order derivatives converges faster.
- Needs selection of  $\eta_n$ :
  - too big: divergence
  - too small: very slow convergence
- When  $\mathcal{L}(u)$  has horizontal asymptotes (e.g. sigmoid), may «get stuck» for large values of  $w^T x_i$ .

- If  $\mathcal{L}(\cdot)$  is convex  $\Rightarrow$  convergence to global min from any starting point.
- If  $\mathcal{L}(\cdot)$  is non-convex  $\Rightarrow$  convergence to different local min, depending on starting point.



# Table of Contents

- 1 Multiclass classification with binary classifiers
- 2 Gradient descent optimization
- 3 Regularization**
- 4 Logistic regression

# Regularization

- Insert additional requirement for regularizer  $R(\beta)$  to be small:

$$\sum_{n=1}^N \mathcal{L}(M(x_n, y_n|w) + \lambda R(\beta) \rightarrow \min_{\beta}$$

- $\lambda > 0$  - hyperparameter.
- $R(\beta)$  penalizes complexity of models.

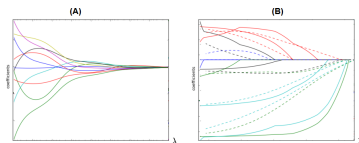
$$R(\beta) = \|\beta\|_1 \quad L_1 \text{ regularization}$$

$$R(\beta) = \|\beta\|_2^2 \quad L_2 \text{ regularization}$$

- Not only *accuracy* matters for the solution but also *model simplicity*!
- $\lambda$  controls complexity of the model:  $\uparrow \lambda \Leftrightarrow \text{complexity} \downarrow$ .

# Comments

- Dependency of  $\beta$  from  $\lambda$  for  $L_2$  (A) and  $L_1$  (B) regularization:



- $L_1$  can be used for automatic feature selection.
- $\lambda$  is usually found using cross-validation on exponential grid, e.g.  $[10^{-6}, 10^{-5}, \dots, 10^5, 10^6]$ .
- It's always recommended to use regularization because
  - it gives smooth control over model complexity.
  - reduces ambiguity for multiple solutions case.

## $L_1$ regularization

- $\|w\|_1$  regularizer will do feature selection.
- Consider

$$L(w) = \sum_{n=1}^N \mathcal{L}(M(x_n, y_n|w)) + \lambda \sum_{d=1}^D |w_d|$$

$$\frac{\partial}{\partial w_i} L(w) = \sum_{n=1}^N \frac{\partial}{\partial w_i} \mathcal{L}(M(x_n, y_n|w)) + \lambda \operatorname{sign} w_i$$

$$\lambda \operatorname{sign} w_i \nrightarrow 0 \text{ when } w_i \rightarrow 0$$

- If  $\lambda > \max_w \left| \sum_{n=1}^N \frac{\partial}{\partial w_i} \mathcal{L}(M(x_n, y_n|w)) \right|$ , then it becomes optimal to set  $w_i = 0$
- For higher  $\lambda$  more weights become zero.

## $L_2$ regularization

$$L(w) = \sum_{n=1}^N \mathcal{L}(M(x_n, y_n|w)) + \lambda \sum_{d=1}^D w_d^2$$

$$\frac{\partial}{\partial w_i} L(w) = \sum_{n=1}^N \frac{\partial}{\partial w_i} \mathcal{L}(M(x_n, y_n|w)) + 2\lambda w_i$$

$$2\lambda w_i \rightarrow 0 \text{ when } w_i \rightarrow 0$$

- Strength of regularization  $\rightarrow 0$  as weights  $\rightarrow 0$ .
- So  $L_2$  regularization will not set weights exactly to 0.

# Table of Contents

- 1 Multiclass classification with binary classifiers
- 2 Gradient descent optimization
- 3 Regularization
- 4 Logistic regression**

# Binary classification

- Linear classifier:

$$\text{score}(\omega_1|x) = w^T x$$

- +relationship between score and class probability is assumed:

$$p(\omega_1|x) = \sigma(w^T x)$$

where  $\sigma(z) = \frac{1}{1+e^{-z}}$  - sigmoid function

## Binary classification: estimation

Using the property  $1 - \sigma(z) = \sigma(-z)$  obtain that

$$p(y = +1|x) = \sigma(w^T x) \implies p(y = -1|x) = \sigma(-w^T x)$$

So for  $y \in \{+1, -1\}$

$$p(y|x) = \sigma(y\langle w, x \rangle)$$

Therefore ML estimation can be written as:

$$\prod_{i=1}^N \sigma(\langle w, x_i \rangle y_i) \rightarrow \max_w$$



## Loss function for 2-class logistic regression

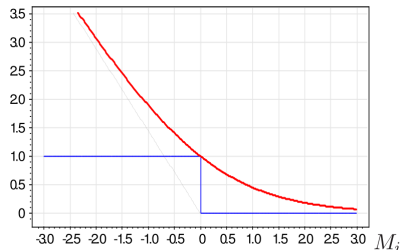
For binary classification  $p(y|x) = \sigma(\langle w, x \rangle y)$   $w = [\beta'_0, \beta]$ ,  
 $x = [1, x_1, x_2, \dots, x_D]$ .

Estimation with ML:

$$\prod_{i=1}^n \sigma(\langle w, x_i \rangle y_i) \rightarrow \max_w$$

which is equivalent to

$$\sum_i^n \ln(1 + e^{-\langle w, x_i \rangle y_i}) \rightarrow \min_w$$



It follows that logistic regression is linear discriminant estimated with loss function  $\mathcal{L}(M) = \ln(1 + e^{-M})$ .

## Multiple classes

Multiple class classification:

$$\begin{cases} \text{score}(\omega_1|x) = w_1^T x \\ \text{score}(\omega_2|x) = w_2^T x \\ \dots \\ \text{score}(\omega_C|x) = w_C^T x \end{cases}$$

+relationship between score and class probability is assumed:

$$p(\omega_c|x) = \text{softmax}(w_c^T x | x_1^T x, \dots, x_C^T x) = \frac{\exp(w_c^T x)}{\sum_i \exp(w_i^T x)}$$

## Multiple classes

### Weights ambiguity:

$w_c$ ,  $c = 1, 2, \dots, C$  defined up to shift  $v$ :

$$\frac{\exp((w_c - v)^T x)}{\sum_i \exp((w_i - v)^T x)} = \frac{\exp(-v^T x) \exp(w_c^T x)}{\sum_i \exp(-v^T x) \exp(w_i^T x)} = \frac{\exp(w_c^T x)}{\sum_i \exp(w_i^T x)}$$

To remove ambiguity usually  $v = w_C$  is subtracted.

### Estimation with ML:

$$\begin{cases} \prod_{n=1}^N \text{softmax}(w_{y_n}^T x_n | x_1^T x, \dots, x_C^T x) \rightarrow \max_{w_1, \dots, w_{C-1}} \\ w_C = 0 \end{cases}$$

## Summary

- Linear classifier - classifier with linear discriminant functions.
- Binary linear classifier:  $\hat{y}(x) = \text{sign}(w^T x + w_0)$ .
- Perceptron, logistic, SVM - linear classifiers estimated with different loss functions.
- Weights are selected to minimize total loss on margins.
- Gradient descent iteratively optimizes  $L(w)$  in the direction of maximum descent.
- Stochastic gradient descent approximates  $\nabla_w L$  by averaging gradients over small subset of objects.
- Regularization gives smooth control over model complexity.
- $L_1$  regularization automatically selects features.