# Линейный классификатор

# Разделяющая поверхность

- Задача классификации с двумя классами $Y = \{-1, +1\}$ по обучающей выборке $X^l$ построить алгоритм классификации $a(x, w) = \text{sign } f(x, w)$, $f(x, w)$ – разделяющая (дискриминантная) функция, $w$ – вектор параметров.

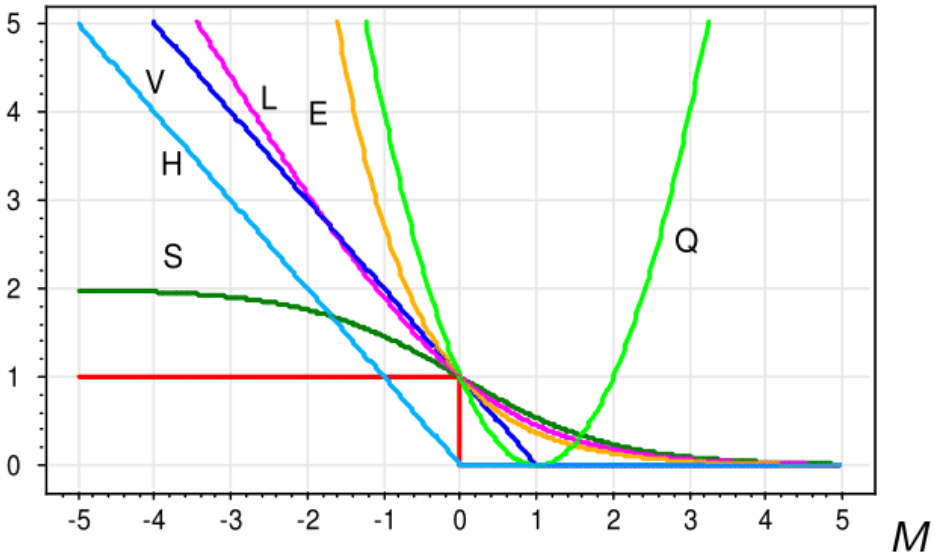
- $f(x, w) = 0$ – разделяющая поверхность;

- $M_i(w) = y_i f(x_i, w)$ – отступ объекта $x_i$;

- $M_i(w) < 0$ – ошибка алгоритма.

$$Q(w) = \sum_{i=1}^{l} [M_i(w) < 0] \le \tilde{Q}(w) = \sum_{i=1}^{l} L\big(M_i(w)\big) \to \min_w$$

# Непрерывные аппроксимации пороговой функции потерь




$V(M) = (1 - M)_+$ — кусочно-линейная (SVM);
$H(M) = (-M)_+$ — кусочно-линейная (Hebb's rule);
$L(M) = \log_2(1 + e^{-M})$ — логарифмическая (LR);
$Q(M) = (1 - M)^2$ — квадратичная (FLD);
$S(M) = 2(1 + e^M)^{-1}$ — сигмоидная (ANN);
$E(M) = e^{-M}$ — экспоненциальная (AdaBoost);
$[M < 0]$ — пороговая функция потерь.

# Линейный классификатор

$f_j$: $j = 1..n$ – числовые признаки

Линейный алгоритм классификации:

$$a(x, w) = sign\left(\sum_{j=1}^{n} w_j f_j(x) - w_0\right)$$

Векторная запись:

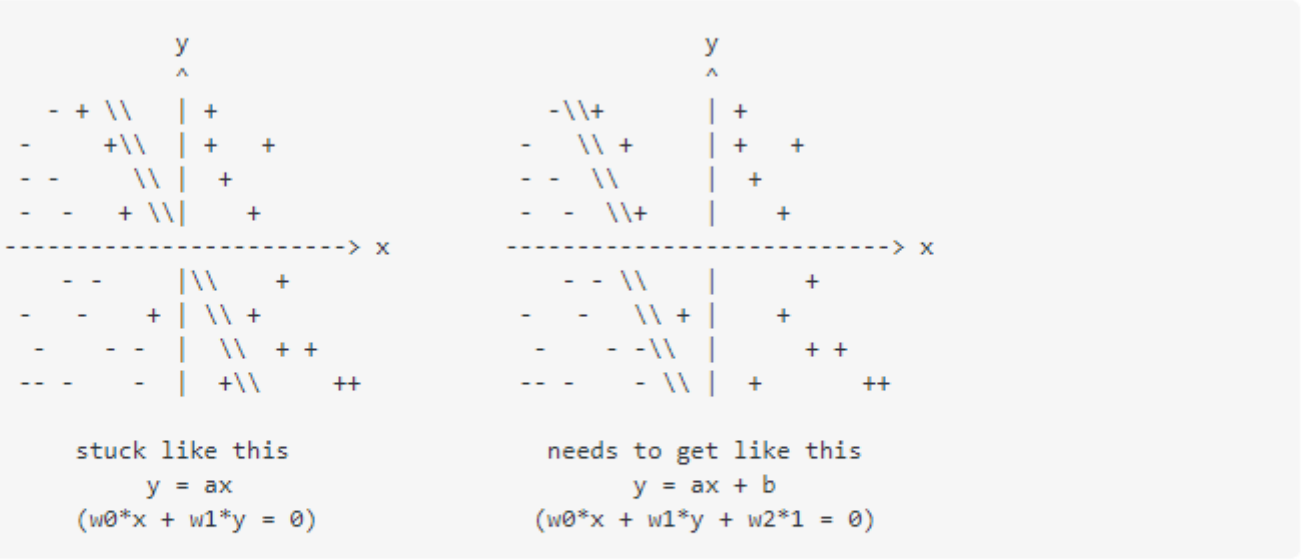
$$a(x, w) = \text{sign}(< w, x >)$$

Отступы объектов:

$$M_i(w) = \langle w, x_i \rangle y_i$$

8

Alright, so perhaps the original ASCII graph was not 100% accurate! Let me try to depict this again:




Share Edit Follow Flag

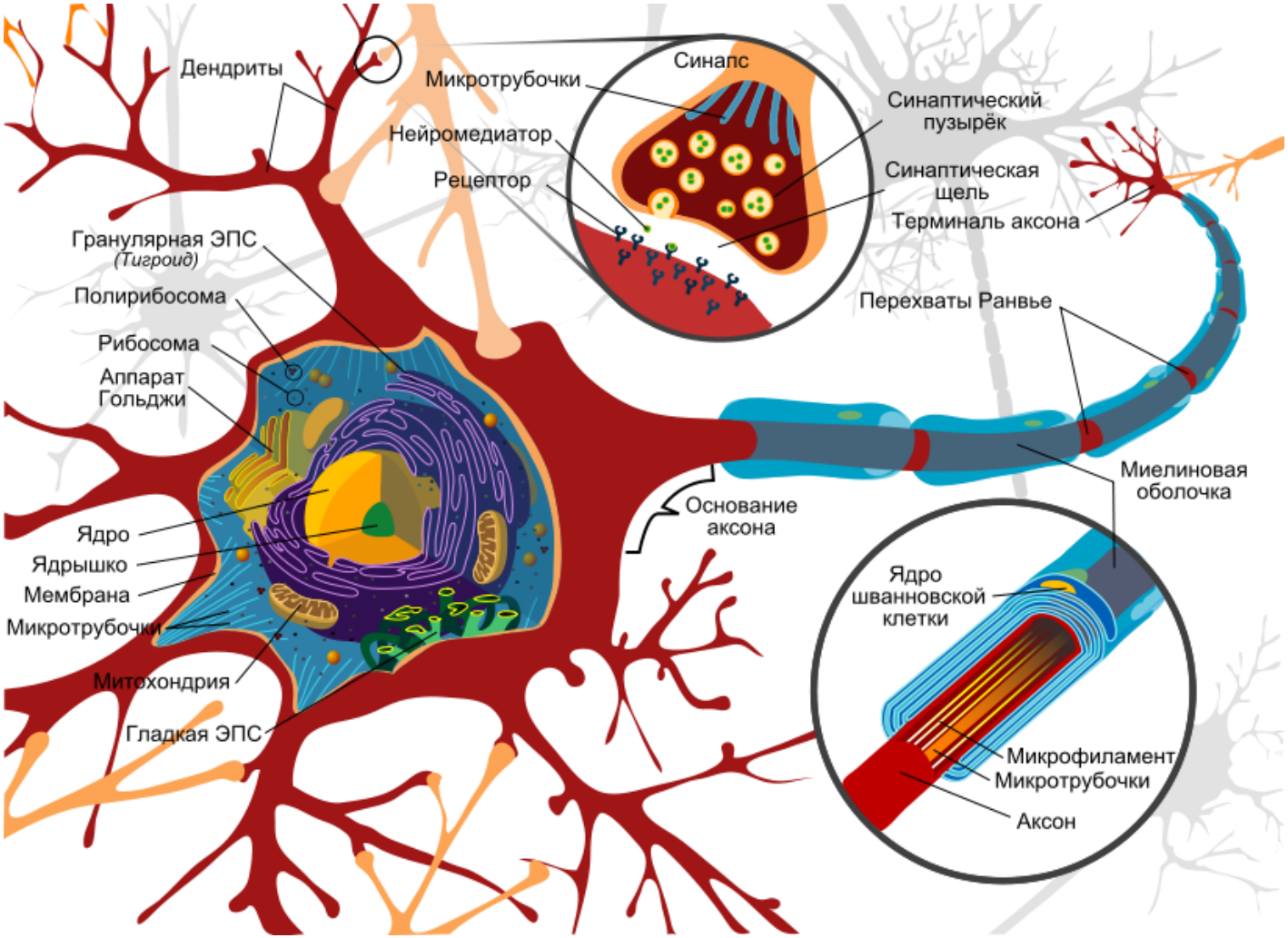
edited May 23, 2017 at 11:57
Community Bot
1 • 1

answered Oct 28, 2014 at 11:27
Amro
122k • 25 • 235 • 441

Add a comment

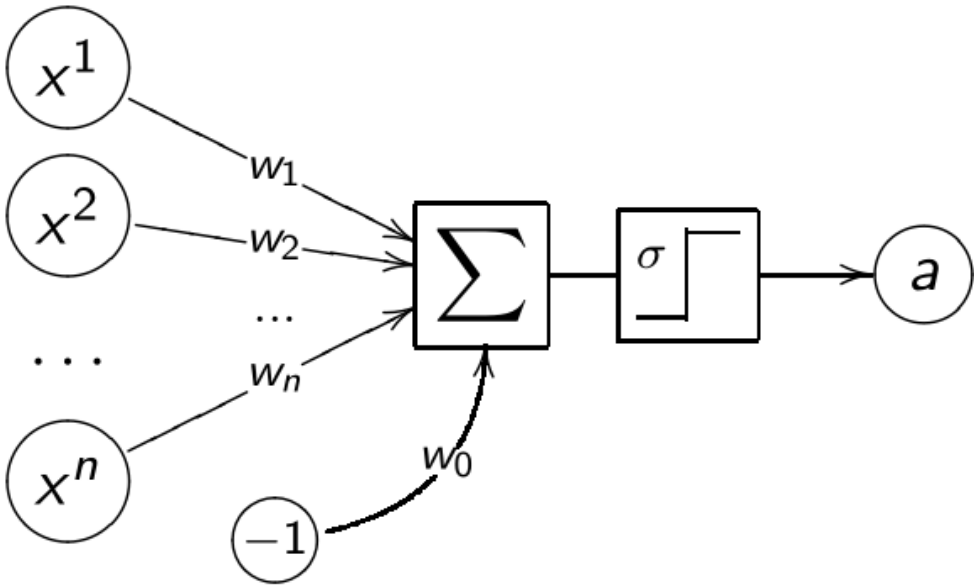
https://stackoverflow.com/questions/26399233/clarification-on-bias-of-a-perceptron

# Линейный классификатор – модель нейрона 1

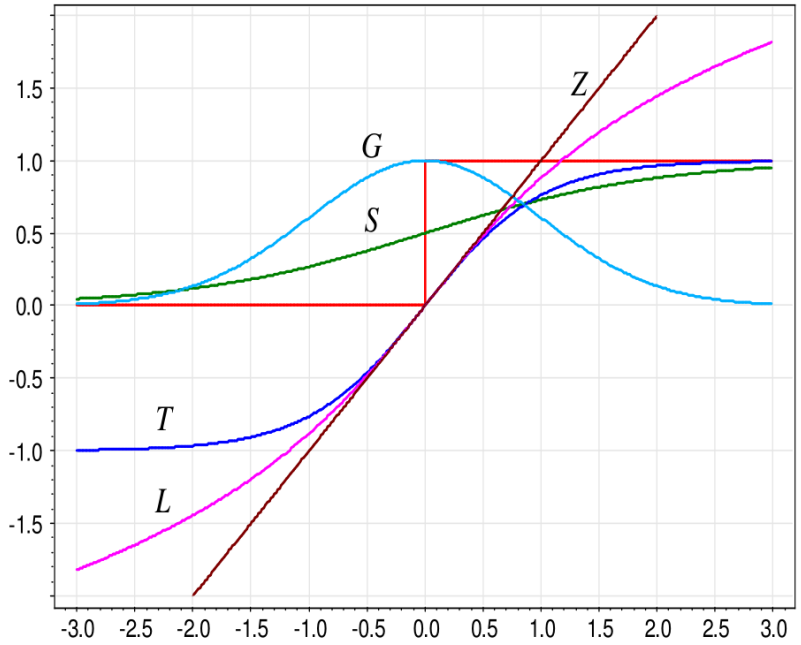


# Линейный классификатор – модель нейрона 2

Линейная модель нейрона Маккалока-Питтса (1943):
$a(x, w) = \sigma(< w, x >) = \sigma\left(\sum_{j=1}^{n} w_j f_j(x) - w_0\right)$,
где $\sigma$ – функция активации (sign).



# Функции активации




| | |
|---|---|
| $\theta(z) = [z \geqslant 0]$ | пороговая функция Хевисайда; |
| $\sigma(z) = (1 + e^{-z})^{-1}$ | сигмоидная функция (S); |
| $\mathrm{th}(z) = 2\sigma(2z) - 1$ | гиперболический тангенс (T); |
| $\ln(z + \sqrt{z^2 + 1})$ | логарифмическая функция (L); |
| $\exp(-z^2/2)$ | гауссовская функция (G); |
| $z$ | линейная функция (Z); |

# Градиентный метод 1

Минимизация аппроксимированного эмпирического риска:

$$Q\left(w, X^l\right) = \sum_{i=1}^{l} L(\langle w, x_i \rangle \cdot y_i) \to \min_w$$

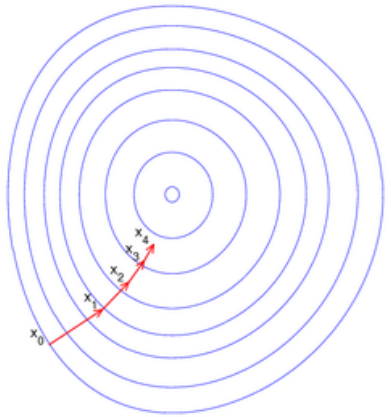
Численная минимизация методом градиентного спуска

$w^{(0)}$ – начальное приближение

$$w^{(t+1)} = w^{(t)} - \eta \cdot \nabla Q(w^{(t)}), \nabla Q\left(w^{(t)}\right) = \left(\frac{\partial Q(w)}{\partial w_j}\right)_{j=1}^{n}$$

$\eta$ – скорость обучения

$$w^{(t+1)} = w^{(t)} - \eta \cdot \sum_{i=1}^{l} (L'(\langle w, x_i \rangle \cdot y_i) \cdot x_i \cdot y_i)$$



$$\nabla f = \frac{\partial f}{\partial x_1} e_1 + \cdots + \frac{\partial f}{\partial x_n} e_n$$

# Частная производная 1

Производная сложной функции:

$$f\big(g(x)\big)' = f'\big(g(x)\big) \cdot g'(x)$$

$$f(x) = (2x+1)^2$$

$f(x)$ – возведение в квадрат

$g(x) = 2x + 1$

$f\big(g(x)\big)' = (g(x)^2)' = 2 \cdot g(x) = 2(2 \cdot x + 1)$

$g'(x) = (2x+1)' = 2$

$f\big(g(x)\big)' = f'\big(g(x)\big) \cdot g'(x) = 2 \cdot 2 \cdot (2 \cdot x + 1)$

# Частная производная 2

$$\nabla Q\left(w^{(t)}\right) = \left(\frac{\partial Q(w)}{\partial w_j}\right)_{j=1}^{n} = \left(\frac{\partial \sum_{i=1}^{l} L(\langle w, x_i\rangle y_i)}{\partial w_j}\right)_{j=1}^{n} =$$

$$\left(\frac{\sum_{i=1}^{l} \partial\left(L(\langle w, x_i\rangle y_i)\right) * \partial(\langle w, x_i\rangle y_i)}{\partial w_j}\right)_{j=1}^{n}$$

$\langle w, x_i\rangle y_i = \left(\sum_{j=1}^{n} w_j x_{i,j}\right) y_i, i = \overline{1, l}, l = 2, n = 2$:

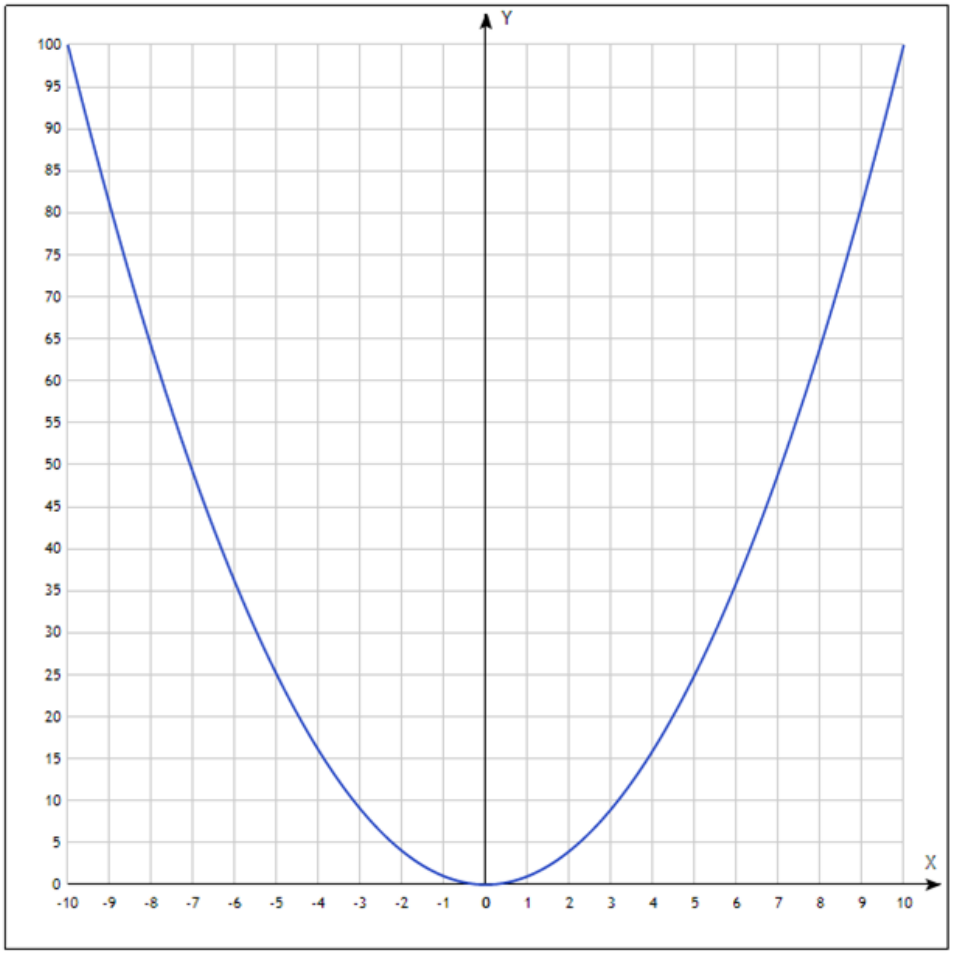
$n -$ количество параметров $w, l$ $-$ количество объектов в выборке

$$\frac{\partial(\langle w, x_i\rangle y_i)}{\partial w_j} = (l = 2, n = 2) = \frac{\partial(w_1 x_{1,1} y_1 + w_2 x_{1,2} y_1 + w_1 x_{2,1} y_2 + w_2 x_{2,2} y_2)}{\partial w_1} +$$

$$\frac{\partial(w_1 x_{1,1} y_1 + w_2 x_{1,2} y_1 + w_1 x_{2,1} y_2 + w_2 x_{2,2} y_2)}{\partial w_2} =$$

$$x_{1,1} y_1 + x_{1,2} y_1 + x_{2,1} y_2 + x_{2,2} y_2 = \sum x_i y_i$$

# Парабола


Y
X
100
95
90
85
80
75
70
65
60
55
50
45
40
35
30
25
20
15
10
5
0
-10 -9 -8 -7 -6 -5 -4 -3 -2 -1 0 1 2 3 4 5 6 7 8 9 10

# Градиентный метод 2

$f(x) = x^2, f'(x) = 2 \cdot x$

Начальное приближение $x_0 = -3, \eta = 0.2;$

$x_1 = -3 - 0.2 \cdot \big(2 \cdot (-3)\big) = -3 + 1.2 = -1.8$

$x_2 = -1.8 - 0.2 \cdot \big(2 \cdot (-1.8)\big) = -1{,}08$

$x_3 = -1.08 - 0.2 \cdot \big(2 \cdot (-1.08)\big) = -0{,}648$

$x_4 = -0.648 - 0.2 \cdot \big(2 \cdot (-0.648)\big) = -0{,}388$

…

Пока $|x_i - x_{i+1}| \geq \varepsilon$ или $|f'(x_i) - f'(x)_{i+1}| \geq \varepsilon$

# Метод стохастического градиента

**Вход:**

$X^\ell$ — обучающая выборка; $\eta$ — темп обучения; $\lambda$ — параметр сглаживания.

**Выход:**

Синаптические веса $w_1, \ldots, w_n$;

---

инициализировать веса $w_j$, $j = 1, \ldots, n$;
инициализировать текущую оценку функционала:
$Q := \sum_{i=1}^{\ell} \mathscr{L}\big(\langle w, x_i\rangle\, y_i\big)$;
**повторять**
  выбрать объект $x_i$ из $X^\ell$ (например, случайным образом);
  вычислить выходное значение алгоритма $a(x_i, w)$ и ошибку:
  $\varepsilon_i := \mathscr{L}\big(\langle w, x_i\rangle\, y_i\big)$;
  сделать шаг градиентного спуска:
  $w := w - \eta \mathscr{L}'\big(\langle w, x_i\rangle\, y_i\big) x_i y_i$;
  оценить значение функционала:
  $Q := (1 - \lambda)Q + \lambda\varepsilon_i$;
**пока** значение $Q$ не стабилизируется и/или веса $w$ не перестанут изменяться;

---

# Адаптивный линейный элемент, ADALINE

Задача регрессии: $X = R^{n+1}, Y \subseteq R$

Видроу и Хофф (1960):

$$L(a, y) = (a - y)^2, a(x, w) = \langle w, x \rangle$$

Градиентный шаг – **дельта-правило**:

$$w = w - \eta(\langle w, x_i \rangle - y_i)x_i$$

$$L(a, y) = (\langle w, x \rangle - y)^2.$$

$$\frac{\partial(\langle w, x \rangle - y)^2}{\partial w} =$$

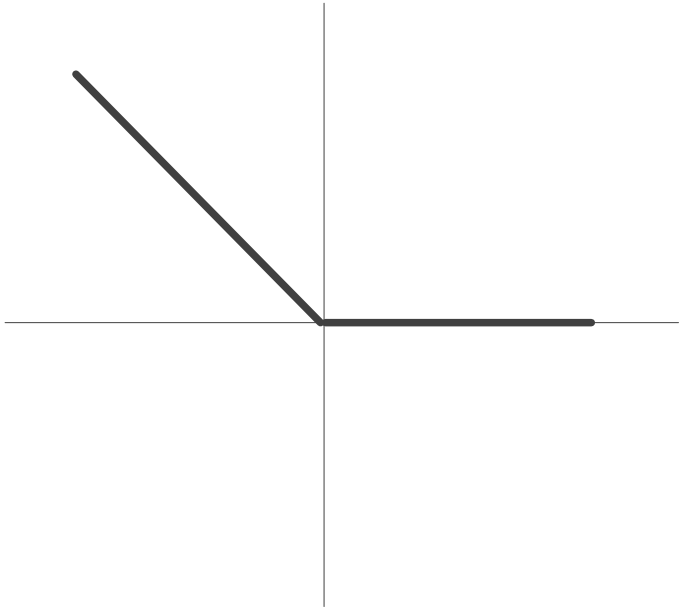
$$= \big(2 \cdot (\langle w, x \rangle - y)\big) \cdot (\langle w, x \rangle - y)' =$$

$$= 2 \cdot (\langle w, x \rangle - y) \cdot x$$

# Правило Хэбба

Задача классификации: $X = R^{n+1}, Y = \{-1, +1\}$

$L(a, y) = -\langle w, x \rangle y$



Линейный классификатор:

$a(x, w) = sign\langle w, x \rangle$

**Правило Хэбба (1949):**

Если $\langle w, x_i \rangle y_i < 0$ то $w = w + \eta x_i y_i$

# Вопрос

Посчитать градиент $L(a, y) = -\langle w, x \rangle y$.

# Инициализация весов

1) $w_j = 0, j = \overline{1, n}$;

2) небольшие случайные значения (или нули)

$w_j = random(-\frac{1}{2n}; \frac{1}{2n})$;

4) обучение по небольшой случайной подвыборке объектов (когда данных много);

5) многократный запуск из разных случайных начальных решений.

# Порядок предъявления объектов

1) случайный порядок;
2) объекты с большой ошибкой брать чаще;
3) не брать «хорошие объекты»;
4) не брать объекты-«выбросы».

# Эвристика

Что такое эвристика?

# Хитрости обучения

- Выбивание из локальных минимумов
- Ранний останов

# Нормализация признаков

$$x = \frac{x - x_{min}}{x_{max} - x_{min}}$$

$$x = \frac{x - x_{avg}}{x_{std}}$$

# Переобучение

1) Слишком мало объектов, слишком много признаков;
2) Мультиколлинеарность.

Идентификация:

1) Слишком большие веса $w$;
2) Неустойчивость $a(x, w)$.

Решение:

1) Сокращение весов;
2) Ранний останов.

# Мультиколлинеарность 1

Для любого объекта обучающей выборки

$$u_1 x_i^1 + \cdots + u_l x_i^l = 0 \text{ или } \langle u, x_i \rangle = 0$$

Система векторов называется линейно зависимой, если из этих векторов можно составить нулевую линейную комбинацию.

$$a = (3,1,2,0), b = (0,-2,1,5), c = (3,-3,4,10)$$

$$c = a + 2b$$

$$a + 2b + (-1)c = (0,0,0,0)$$

# Мультиколлинеарность 2

$$w^* = \arg\min_w \sum_{i=1}^{l} (\langle w, x_i \rangle - y_i)^2$$

$$w_1 = w^* + tu$$

$$\langle w_1, x \rangle = \langle w^* + tu, x \rangle = \langle w^*, x \rangle + t\langle u, x \rangle = \langle w^*, x \rangle$$

# Регуляризация

$$Q(w) = \sum_{i=1}^{l} L\big(M_i(w)\big) + \alpha\|w\|^p \to \min_w$$

$L1$ – регуляризация ($p = 1$), ;

$L2$ – регуляризация ($p = 2$).

# Норма вектора

- Гёльдеровы нормы $n$-мерных векторов (семейство): $\|x\|_p = \left(\sum_i |x_i|^p\right)^{\frac{1}{p}}$,

где $p \geqslant 1$ (обычно подразумевается, что это натуральное число). В частности:

- $\|x\|_1 = \sum_i |x_i|$, что также имеет название *метрика L1*, *норма* $\ell_1$ или манхэттенское расстояние. Для вектора представляет собой сумму модулей всех его элементов.
- $\|x\|_2 = \sqrt{\sum_i |x_i|^2}$, что также имеет название *метрика L2*, *норма* $\ell_2$ или евклидова норма. Является геометрическим расстоянием между двумя точками в многомерном пространстве, вычисляемым по теореме Пифагора.
- $\|x\|_\infty = \max |x_i|$ (это предельный случай $p \to \infty$).

Запишем задачу настройки вектора параметров $\beta$:

$$Q(\beta) = \sum_{i=1}^{l} \mathcal{L}_i(\beta) + \lambda \sum_{j=1}^{n} |\beta_j|,$$

где $\mathcal{L}_i(\beta) = \mathcal{L}(y_i, g(x_i, \beta))$ — некоторая ограниченная гладкая функция потерь. Сделаем замену переменных, чтобы функционал стал гладким. Каждой переменной $\beta_j$ поставим в соответствие две новые неотрицательные переменные:

$$\begin{cases} u_j = \frac{1}{2}(|\beta_j| + \beta_j) \\ v_j = \frac{1}{2}(|\beta_j| - \beta_j) \end{cases}$$

Тогда:

$$\begin{cases} \beta_j = u_j - v_j \\ |\beta_j| = u_j + v_j \end{cases}$$

В новых переменных функционал становится гладким, но добавляются ограничения-неравенства:

$$\begin{cases} Q(u, v) = \sum_{i=1}^{l} \mathcal{L}_i(u - v) + \lambda \sum_{j=1}^{n} (u_j + v_j) \to \min\limits_{u,v} \\ u_j \geq 0, v_j \geq 0, \ j = 1, \ldots, n \end{cases}$$

Для любого $j$ хотя бы одно из ограничений $u_j \geq 0$ и $v_j \geq 0$ обращается в равенство, иначе второе слагаемое в $Q(u, v)$ можно было бы уменьшить, не изменив первое. Если гиперпараметр $\lambda$ устремить к $\infty$, в какой-то момент все $2n$ ограничений обратятся в равенство. Постепенное увеличение гиперпараметра $\lambda$ приводит к увеличению числа таких $j$, для которых $u_j = v_j = 0$, откуда следует, что $\beta_j = 0$. Как говорилось ранее, в линейных моделях это означает, что значения $j$-го признака игнорируются, и его можно исключить из модели.

# Персептрон

## sklearn.linear_model.Perceptron

*class* `sklearn.linear_model.` **`Perceptron`** (*penalty=None, alpha=0.0001, fit_intercept=True, max_iter=None, tol=None, shuffle=True, verbose=0, eta0=1.0, n_jobs=1, random_state=0, class_weight=None, warm_start=False, n_iter=None*)

[source]

Read more in the User Guide.

**Parameters:**

**penalty** : None, 'l2' or 'l1' or 'elasticnet'

The penalty (aka regularization term) to be used. Defaults to None.

**alpha** : float

Constant that multiplies the regularization term if regularization is used. Defaults to 0.0001

**fit_intercept** : bool

Whether the intercept should be estimated or not. If False, the data is assumed to be already centered. Defaults to True.

**max_iter** : int, optional

The maximum number of passes over the training data (aka epochs). It only impacts the behavior in the `fit` method, and not the partial_fit. Defaults to 5. Defaults to 1000 from 0.21, or if tol is not None.

*New in version 0.19.*

**tol** : float or None, optional

The stopping criterion. If it is not None, the iterations will stop when (loss > previous_loss - tol). Defaults to None. Defaults to 1e-3 from 0.21.

*New in version 0.19.*

**shuffle** : bool, optional, default True

Whether or not the training data should be shuffled after each epoch.

**verbose** : integer, optional

The verbosity level

**eta0** : double

Constant by which the updates are multiplied. Defaults to 1.

**n_jobs** : integer, optional

The number of CPUs to use to do the OVA (One Versus All, for multi-class problems) computation. -1 means 'all CPUs'. Defaults to 1.

**random_state** : int, RandomState instance or None, optional, default None

The seed of the pseudo random number generator to use when shuffling the data. If int, random_state is the seed used by the random number generator; If RandomState instance, random_state is the random number generator; If None, the random number generator is the RandomState instance used by np.random.

**class_weight** : dict, {class_label: weight} or "balanced" or None, optional

Preset for the class_weight fit parameter.

Weights associated with classes. If not given, all classes are supposed to have weight one.

The "balanced" mode uses the values of y to automatically adjust weights inversely proportional to class frequencies in the input data as `n_samples / (n_classes * np.bincount(y))`

**warm_start** : bool, optional

When set to True, reuse the solution of the previous call to fit as initialization, otherwise, just erase the previous solution.

**n_iter** : int, optional

The number of passes over the training data (aka epochs). Defaults to None. Deprecated, will be removed in 0.21.

*Changed in version 0.19:* Deprecated

# Набор данных

Attribute Information:
1.sepal length in cm
2. sepal width in cm
3. petal length in cm
4. petal width in cm
5. class:
-- Iris Setosa
-- Iris Versicolour
-- Iris Virginica

```
5.1,3.5,1.4,0.2,Iris-setosa
4.9,3.0,1.4,0.2,Iris-setosa
4.7,3.2,1.3,0.2,Iris-setosa
4.6,3.1,1.5,0.2,Iris-setosa
5.0,3.6,1.4,0.2,Iris-setosa
5.4,3.9,1.7,0.4,Iris-setosa
4.6,3.4,1.4,0.3,Iris-setosa
5.0,3.4,1.5,0.2,Iris-setosa
4.4,2.9,1.4,0.2,Iris-setosa
4.9,3.1,1.5,0.1,Iris-setosa
5.4,3.7,1.5,0.2,Iris-setosa
4.8,3.4,1.6,0.2,Iris-setosa
4.8,3.0,1.4,0.1,Iris-setosa
4.3,3.0,1.1,0.1,Iris-setosa
5.8,4.0,1.2,0.2,Iris-setosa
5.7,4.4,1.5,0.4,Iris-setosa
5.4,3.9,1.3,0.4,Iris-setosa
5.1,3.5,1.4,0.3,Iris-setosa
5.7,3.8,1.7,0.3,Iris-setosa
5.1,3.8,1.5,0.3,Iris-setosa
5.4,3.4,1.7,0.2,Iris-setosa
```

# Пример работы персептрона

```
import pandas as pd
import numpy as np
from sklearn import datasets
from sklearn.linear_model import perceptron
from sklearn.model_selection import train_test_split

iris = datasets.load_iris()

X = iris.data[:,:]
y = iris.target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=.3, random_state=0)

net = perceptron.Perceptron(n_iter=100, verbose=0, random_state=None, fit_intercept=True, eta0=0.001)
net.fit(X_train,y_train)

print('The accuracy of the classifier on training data is {:.2f} out of 1'.format(net.score(X_train, y_train)))
print('The accuracy of the classifier on test data is {:.2f} out of 1'.format(net.score(X_test, y_test)))

print(net.coef_)
```

```
The accuracy of the classifier on training data is 0.89 out of 1
The accuracy of the classifier on test data is 0.89 out of 1
[[ 0.0016  0.0046 -0.0075 -0.0036]
 [ 0.0105 -0.0536  0.0239 -0.0663]
 [-0.0785 -0.0479  0.105   0.0799]]
```

# Пример L1 регуляризации

```
import pandas as pd
import numpy as np
from sklearn import datasets
from sklearn.linear_model import perceptron
from sklearn.model_selection import train_test_split

iris = datasets.load_iris()

s = [0,1]
X = iris.data[:,:]
y = iris.target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=.3, random_state=0)

net = perceptron.Perceptron(n_iter=100, verbose=0, random_state=None, fit_intercept=True, eta0=0.001, penalty ='l1', alpha = 0.03
net.fit(X_train,y_train)

print('The accuracy of the classifier on training data is {:.2f} out of 1'.format(net.score(X_train, y_train)))
print('The accuracy of the classifier on test data is {:.2f} out of 1'.format(net.score(X_test, y_test)))

print(net.coef_)
```

```
The accuracy of the classifier on training data is 0.43 out of 1
The accuracy of the classifier on test data is 0.44 out of 1
[[ 0.      0.     -0.0034  0.    ]
 [ 0.     -0.0018  0.      0.    ]
 [ 0.      0.      0.0187  0.    ]]
```

# Пример L2 регуляризации

```
import pandas as pd
import numpy as np
from sklearn import datasets
from sklearn.linear_model import perceptron
from sklearn.model_selection import train_test_split

iris = datasets.load_iris()

s = [0,1]
X = iris.data[:,:]
y = iris.target

print(X.shape)
print(y.shape)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=.3, random_state=0)

net = perceptron.Perceptron(n_iter=100, verbose=0, random_state=None, fit_intercept=True, eta0=0.001, penalty ='l2', alpha = 0.03
net.fit(X_train,y_train)

print('The accuracy of the classifier on training data is {:.2f} out of 1'.format(net.score(X_train, y_train)))
print('The accuracy of the classifier on test data is {:.2f} out of 1'.format(net.score(X_test, y_test)))

print(net.coef_)
```

```
(150, 4)
(150,)
The accuracy of the classifier on training data is 0.85 out of 1
The accuracy of the classifier on test data is 0.84 out of 1
[[ 0.00087619  0.00255496 -0.00664195 -0.00291955]
 [ 0.00591879 -0.04933992  0.02608472 -0.06212774]
 [-0.06367701 -0.04977843  0.08653533  0.07180048]]
```

# L1 vs L2

The **key difference** between these techniques is that L1 shrinks the less important feature's coefficient to zero thus, removing some feature altogether. So, this works well for **feature selection** in case we have a huge number of features.

L1-norm does not have an analytical solution, but L2-norm does (due to swuare).

# Пример добавления корреляций

```
import numpy as np
# генерация признаков
a = np.random.rand(2000, 7)
c = a[np.where((a[:,0] > 0.5) * (a[:,1] > 0.5))]
d = a[np.where((a[:,0] < 0.7) * (a[:,1] < 0.7))]
z = np.concatenate((c, d), axis=0)
x  = (z[:,0] + 3*z[:,1])
x1 = (z[:,2] -   z[:,3])
x2 = (2*z[:,4] -   z[:,5] + z[:,1])
x3 = (2*z[:,2] -   z[:,4] + z[:,0])
# создание классов
c1 = np.repeat(0, np.size(c, 0))
c2 = np.repeat(1, np.size(d, 0))
cls = np.concatenate((c1, c2), axis=0)

linf = z
#linf = np.column_stack([z, x])
#linf = np.column_stack([linf, x1])
#linf = np.column_stack([linf, x2])
#linf = np.column_stack([linf, x3])

import pandas as pd
import numpy as np
from sklearn import datasets
from sklearn.linear_model import perceptron
from sklearn.model_selection import train_test_split

X = linf
y = cls
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=.3, random_state=0)
net = perceptron.Perceptron(n_iter=100, verbose=0, random_state=None, fit_intercept=True, eta0=0.5, penalty=None)
net.fit(X_train,y_train)

print('The accuracy of the classifier on training data is {:.2f} out of 1'.format(net.score(X_train, y_train)))
print('The accuracy of the classifier on test data is {:.2f} out of 1'.format(net.score(X_test, y_test)))

print(np.std(net.coef_))
```

```
The accuracy of the classifier on training data is 0.93 out of 1
The accuracy of the classifier on test data is 0.92 out of 1
3.6969946155
```

# Увеличение СКО коэффициентов

```
import numpy as np
# генерация признаков
a = np.random.rand(2000, 7)
c = a[np.where((a[:,0] > 0.5) * (a[:,1] > 0.5))]
d = a[np.where((a[:,0] < 0.7) * (a[:,1] < 0.7))]
z = np.concatenate((c, d), axis=0)
x  = (z[:,0] + 3*z[:,1])
x1 = (z[:,2] -   z[:,3])
x2 = (2*z[:,4] -   z[:,5] + z[:,1])
x3 = (2*z[:,2] -   z[:,4] + z[:,0])
# создание классов
c1 = np.repeat(0, np.size(c, 0))
c2 = np.repeat(1, np.size(d, 0))
cls = np.concatenate((c1, c2), axis=0)

linf = z
linf = np.column_stack([z, x])
linf = np.column_stack([linf, x1])
linf = np.column_stack([linf, x2])
linf = np.column_stack([linf, x3])

import pandas as pd
import numpy as np
from sklearn import datasets
from sklearn.linear_model import perceptron
from sklearn.model_selection import train_test_split

X = linf
y = cls
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=.3, random_state=0)
net = perceptron.Perceptron(n_iter=100, verbose=0, random_state=None, fit_intercept=True, eta0=0.5, penalty=None)
net.fit(X_train,y_train)

print('The accuracy of the classifier on training data is {:.2f} out of 1'.format(net.score(X_train, y_train)))
print('The accuracy of the classifier on test data is {:.2f} out of 1'.format(net.score(X_test, y_test)))

print(np.std(net.coef_))
```

```
The accuracy of the classifier on training data is 0.94 out of 1
The accuracy of the classifier on test data is 0.92 out of 1
7.33888696613
```