

**«Машинное обучение»**

# **Нелинейные методы**

**Александр Дьяконов**



## План

**Что делать, если хотим искать нелинейные закономерности линейными методами**

**«Трюки с ядрами (kernel tricks)»**

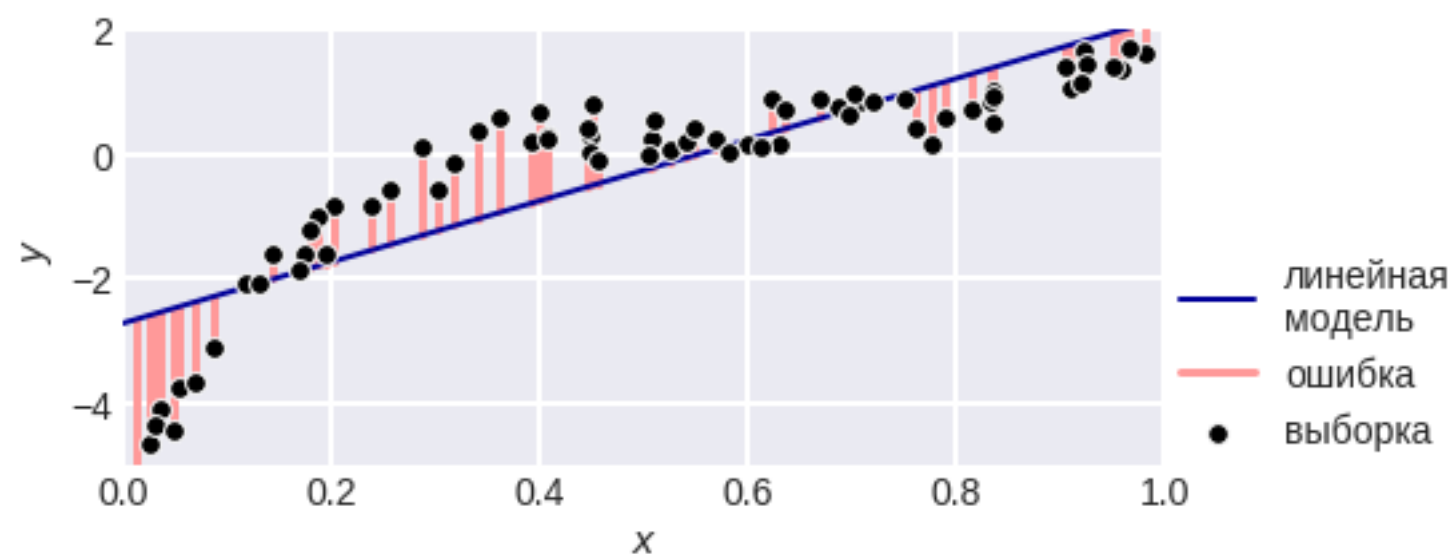
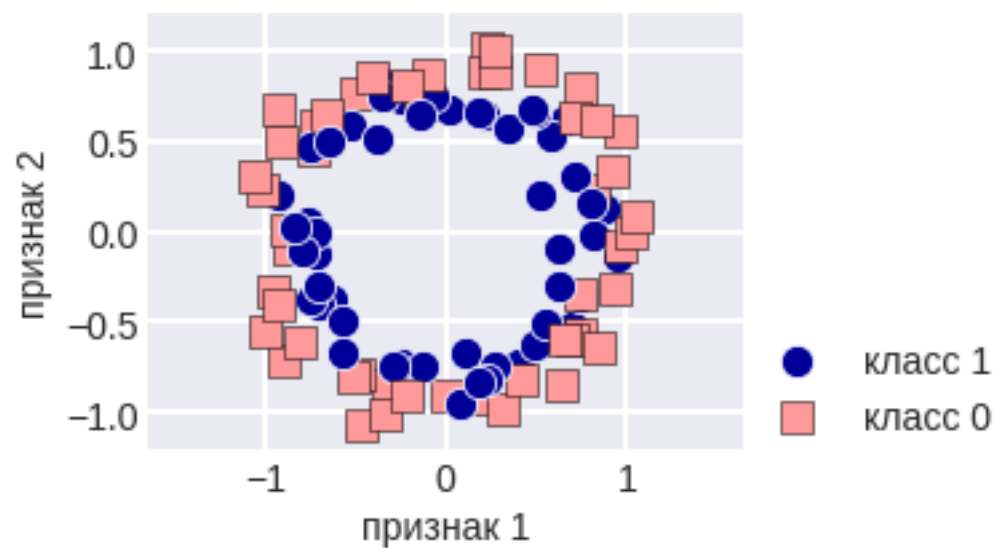
**Вывод нелинейного SVM**

**«Кернализация» других методов**

**Решение задач произвольной природы**

## Проблема линейности

### Некоторые задачи не решаются линейными методами



## Когда линейной модели не хватает

### линейная модель + новые признаки

- деформации существующих / базисные функции

**GAM (Generalized Additive Models)**

Пример: кусочно-полиномиальная модель / RBF

- локально линейные методы

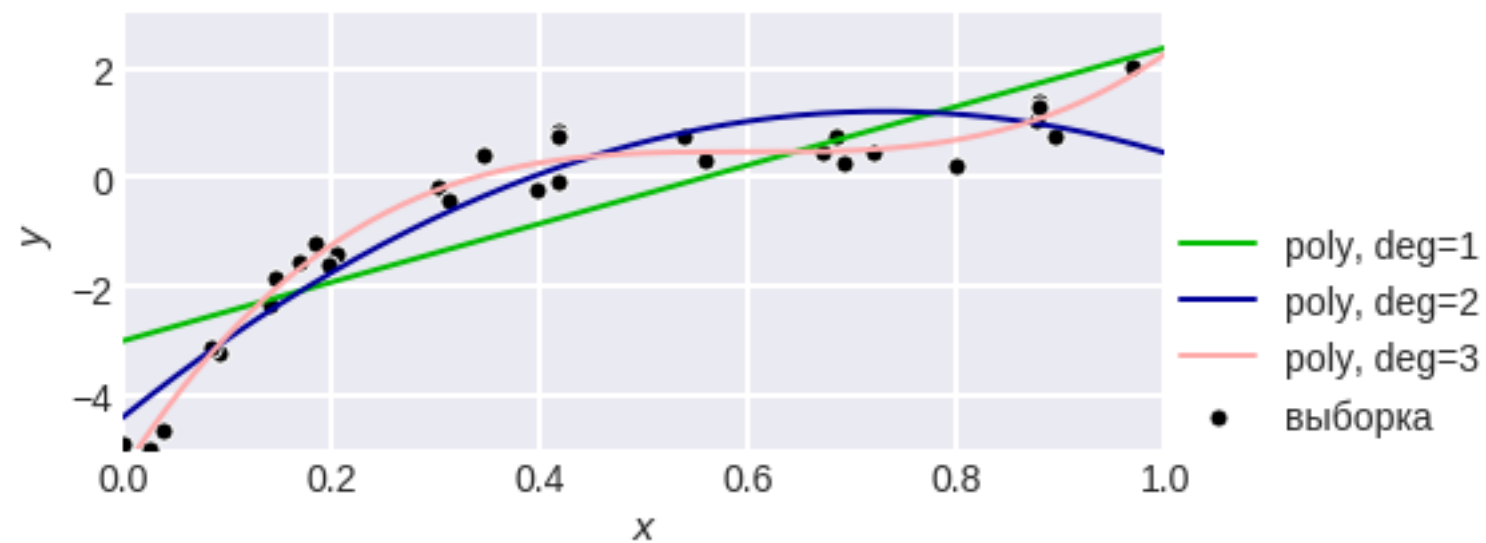
Пример: Local Regression

- ядерные методы (Kernel Tricks)

### использование нелинейной модели

- метрические алгоритмы
- деревья
- ансамбли
  - RF
  - GBM
  - NN

## Деформация: полиномиальная модель



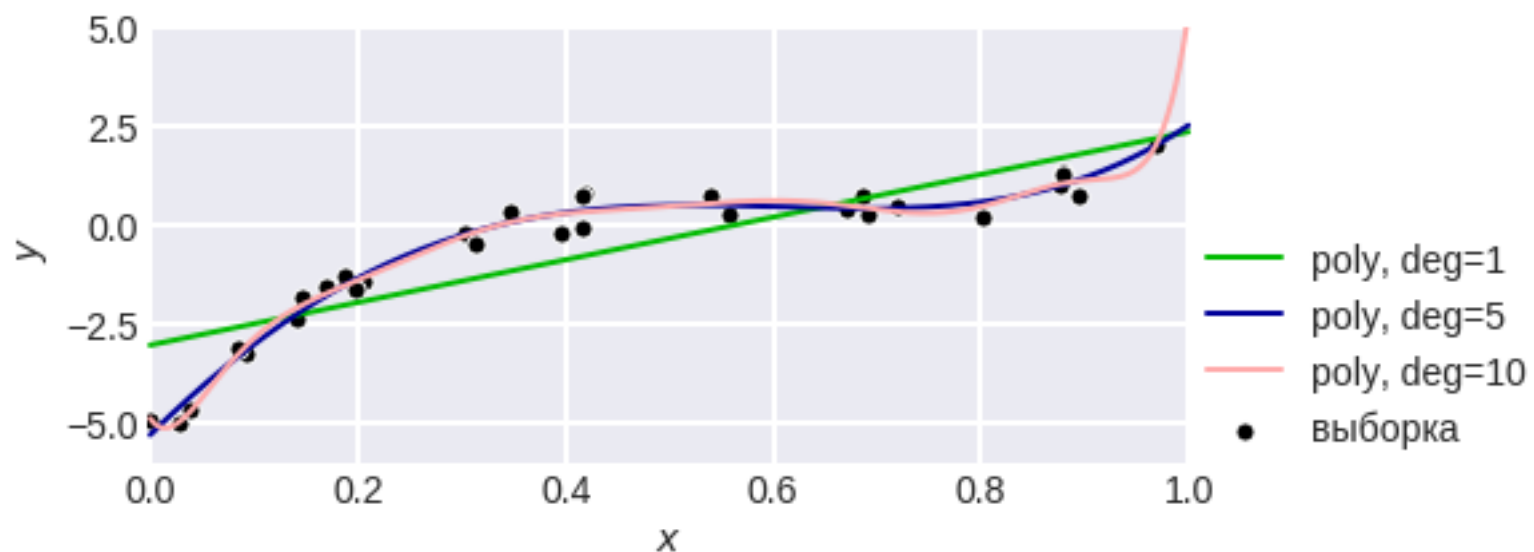
**добавляем признаки-мономы**

**в задаче с одним признаком  $(x) \rightarrow (1, x, x^2, \dots, x^k)$**

**теперь линейная регрессия превращается в  $a(x) = w_0 + w_1x + w_2x^2 + \dots + w_kx^k$**

**с несколькими  $(X_1, \dots, X_n) \rightarrow (\dots, \prod_{t \in T} X_t, \dots)$**

## Деформация: полиномиальная модель



**подводные камни – очень много признаков добавлять:**

- 1) переобучение
- 2) неестественные признаки

**потом в материале «сложность»**

## Минутка кода: полиномиальная модель

```
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import Ridge

poly = PolynomialFeatures(degree=degree)

X = poly.fit_transform(X)
XX = poly.fit_transform(XX)

clf = Ridge(alpha=alpha,
            fit_intercept=True,
            normalize=True)

clf.fit(X, y)
a = clf.predict(XX)
```

## Деформация

**Важно: можно деформировать и целевой признак!**

```
clf.fit(X, np.expm1(y))  
a = np.log1p(clf.predict(X))
```

$$e^y - 1 = w^T x \Rightarrow y = \log(w^T x + 1)$$



## Использование других базисных функций

**Просто составляем новую признаковую матрицу  
и «запихиваем» её в функцию регрессии / классификации**

$$(X_1, \dots, X_n) \rightarrow (\dots, f_j(X_1, \dots, X_n), \dots)$$

**например, характеристические функции интервалов**

$$\varphi(X_t) = I[\theta_i \leq X_t < \theta_{i+1}]$$

**выбор хороших порогов (cutpoints / knots) – отдельная задача, можно:**

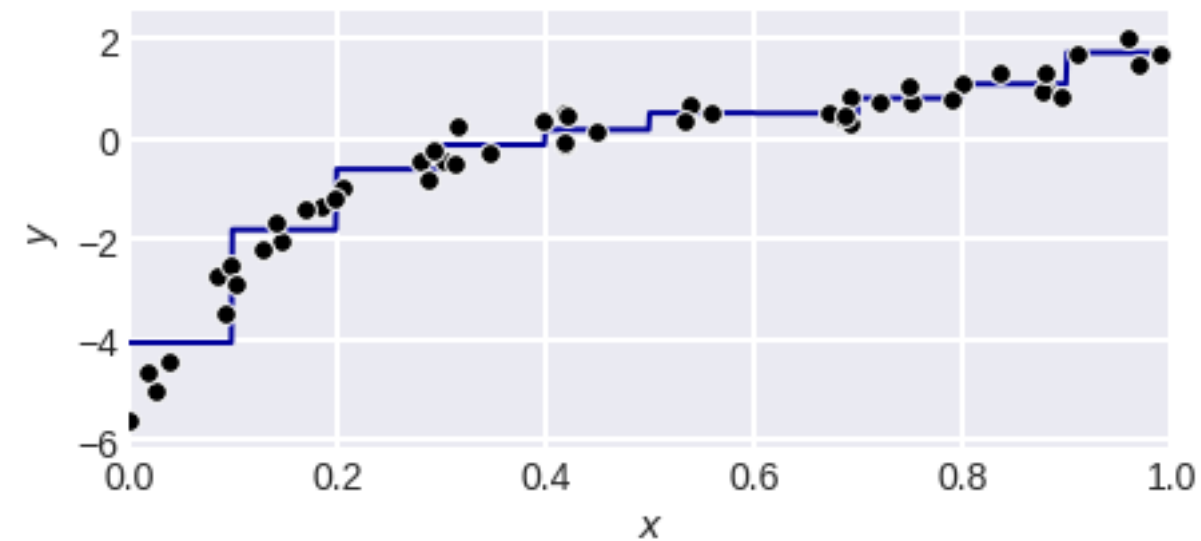
- **квантили**
- **«межкластерные» точки**
- **особые точки**

**(например, круглые суммы в признаке «зарплата»)**

## Использование других базисных функций

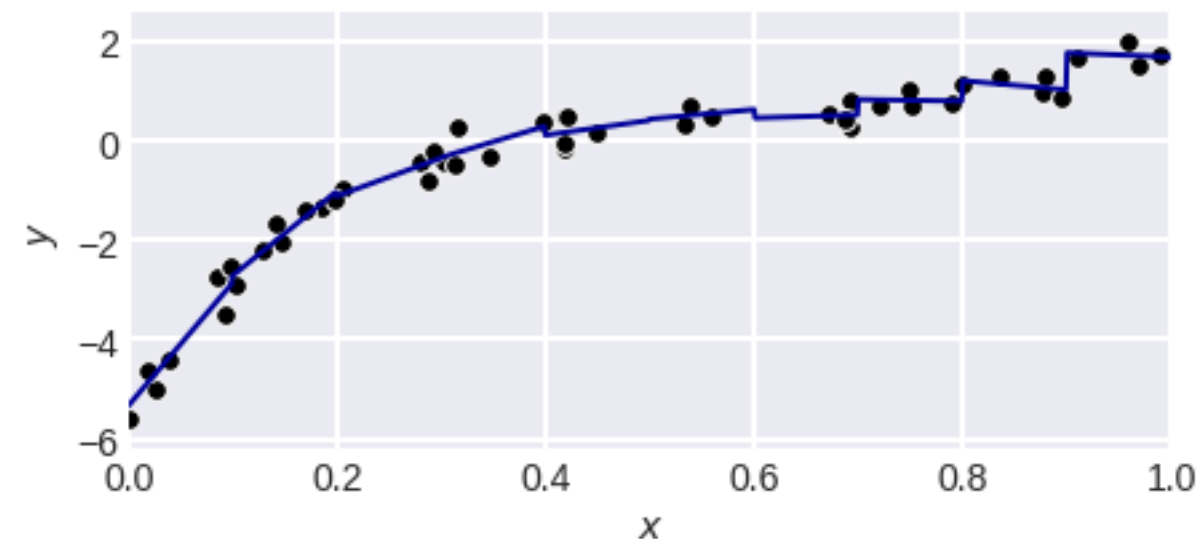
**сравнение с порогом**

$$I[X_t \geq \theta_i] = \begin{cases} 1, & X_t \geq \theta_i, \\ 0, & X_t < \theta_i, \end{cases}$$



**кусочно-линейная регрессия**

$$I[\theta_i \leq X_t < \theta_{i+1}], \\ X_t \cdot I[\theta_i \leq X_t < \theta_{i+1}]$$



аналогично кусочно-полиномиальная

## Использование других базисных функций

«Сплайны» получаются при добавлении функций вида

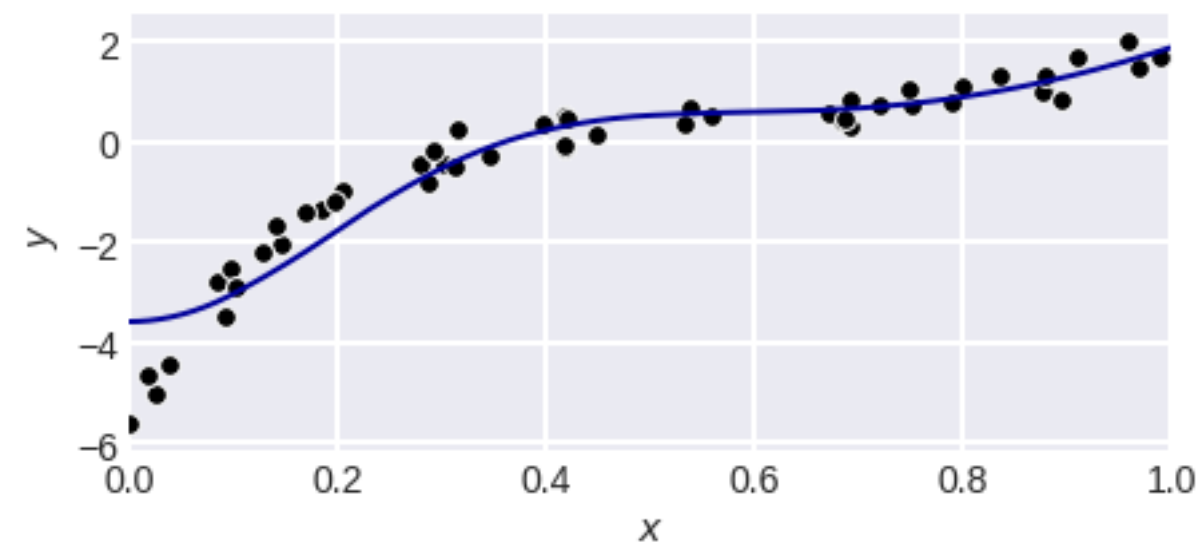
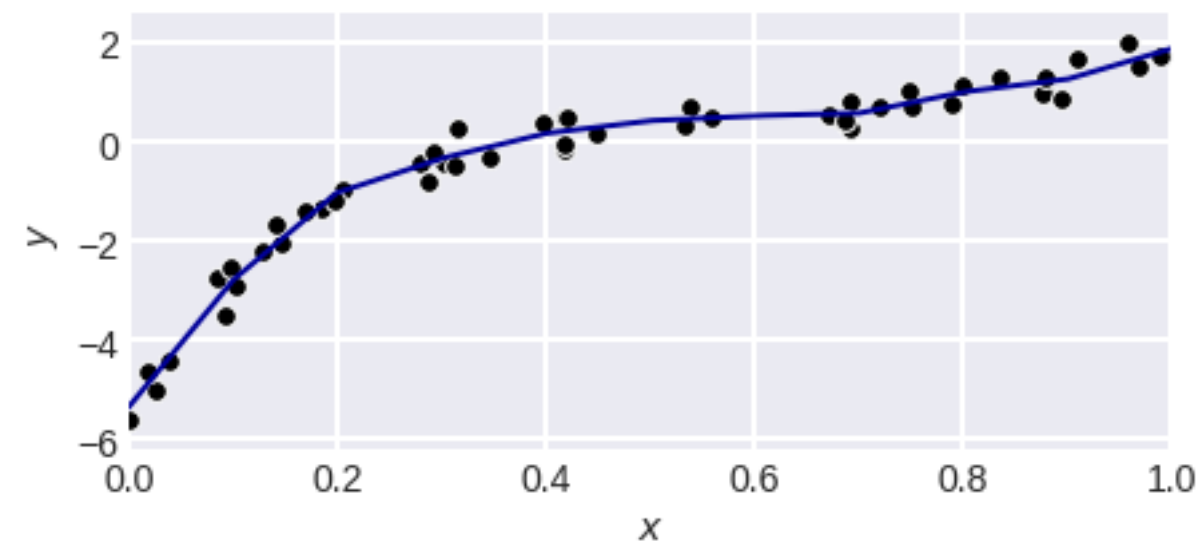
$$\begin{cases} |X_t - \theta_i|^k, & X_t \geq \theta_i, \\ 0, & X_t < \theta_i, \end{cases}$$

Здесь (вверху) –

$$\begin{cases} |X_t - \theta_i|, & X_t \geq \theta_i, \\ 0, & X_t < \theta_i, \end{cases}$$

(внизу) –

$$\begin{cases} (X_t - \theta_i)^2, & X_t \geq \theta_i, \\ 0, & X_t < \theta_i, \end{cases}$$



## Радиально-базисная функция (Radial basis function, RBF)

**Радиальная функция (Radial function) – функция вида**

$$\varphi_z(x) = f(\|x - z\|) : \mathbb{R}^n \rightarrow \mathbb{R}$$

т.е. зависящая от расстояния (в более общем случае – любого) до какой-то точки

**Радиальная функция называется радиально-базисной,  
если для любого набора попарно различных точек  $\{x_1, \dots, x_m\}$ ,  
функции  $\varphi_{x_1}(x), \dots, \varphi_{x_m}(x)$  линейно независимы  
и матрица  $\|\varphi_{x_j}(x_i)\|_{m \times m}$  невырождена**

$$\|\varphi_{x_j}(x_i)\|_{m \times m} = \begin{bmatrix} \varphi_{x_1}(x_1) & \dots & \varphi_{x_m}(x_1) \\ \dots & \dots & \dots \\ \varphi_{x_1}(x_m) & \dots & \varphi_{x_m}(x_m) \end{bmatrix}$$

**Радиально-базисная функция (Radial basis function, RBF)****Gaussian**

$$f(r) = \exp(-\varepsilon r^2)$$

**Thin plate spline**

$$f(r) = r^2 \ln r$$

**Multiquadric**

$$f(r) = \sqrt{1 + \varepsilon r^2}$$

**Inverse multiquadric**

$$f(r) = \frac{1}{\sqrt{1 + \varepsilon r^2}}$$

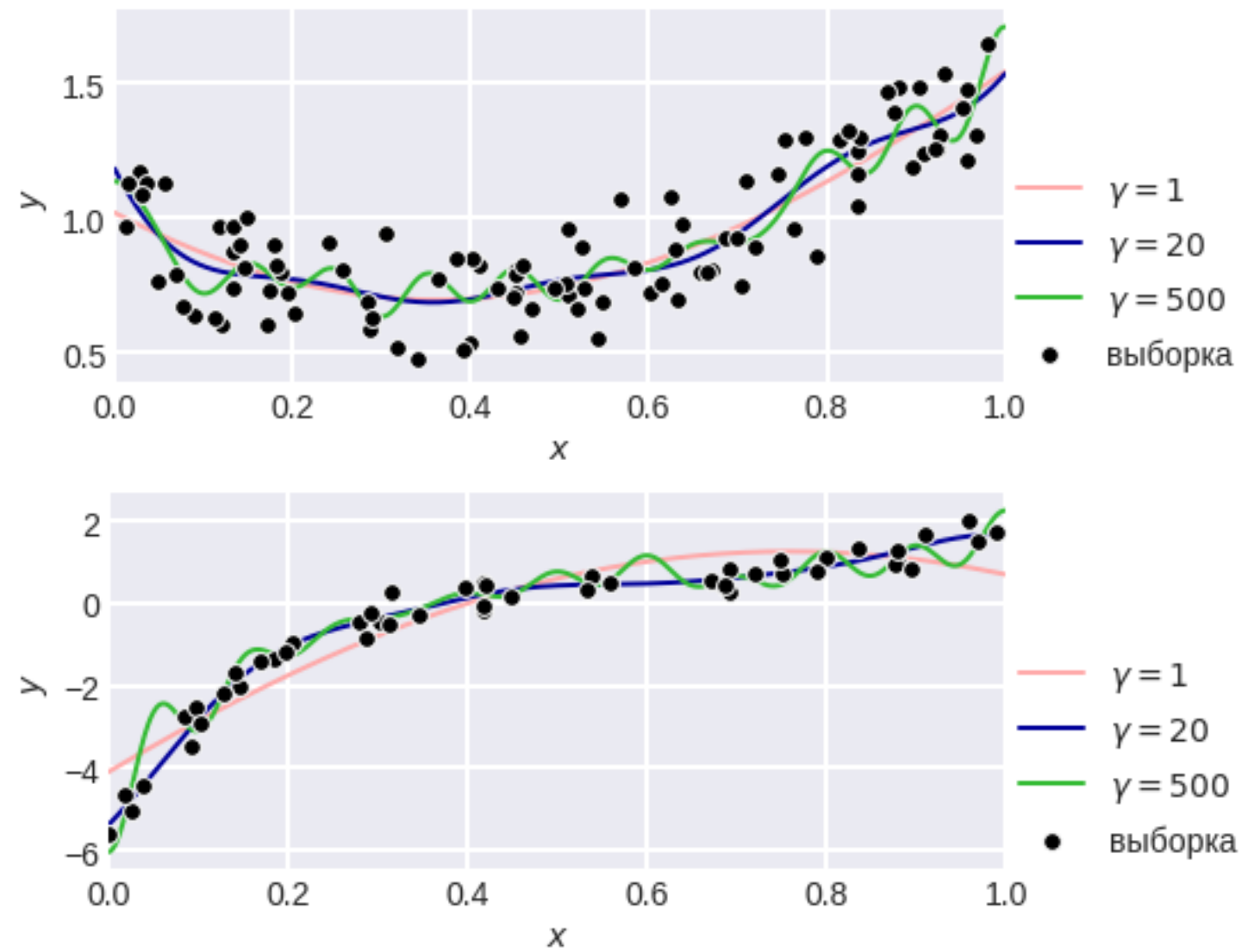
**Inverse quadratic**

$$f(r) = \frac{1}{1 + \varepsilon r^2}$$

**Polyharmonic spline**

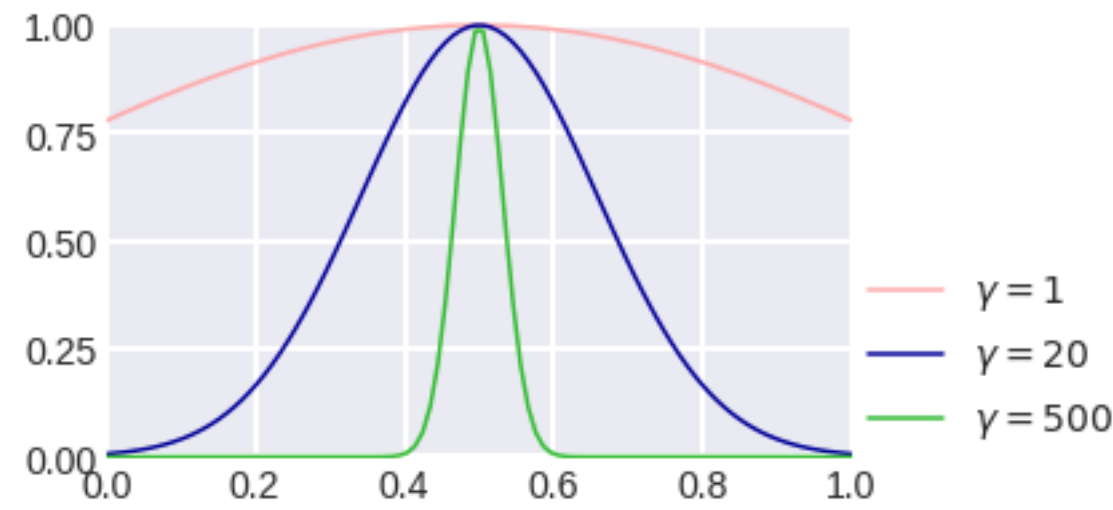
$$f(r) = \begin{cases} r^k, & k = 1, 3, 5, \dots \\ r^k \ln r, & k = 2, 4, 6, \dots \end{cases}$$

RBF-ядро: пример для регрессии

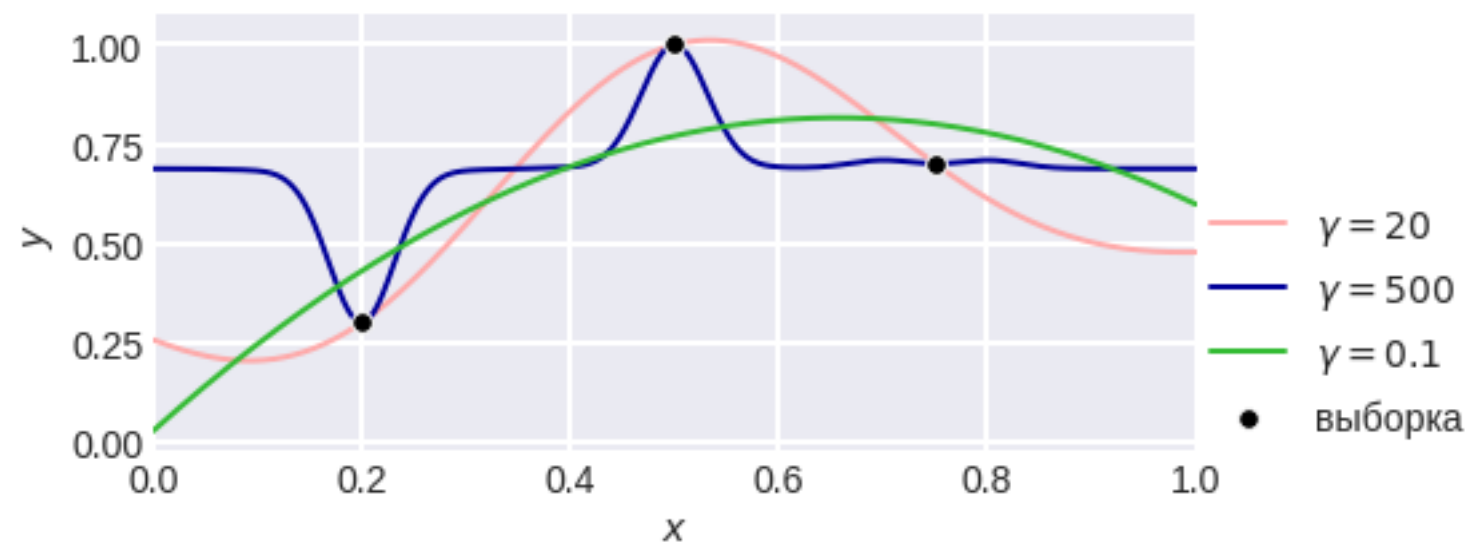


$$\varphi(x) = \exp(-\gamma \|x - z\|^2)$$

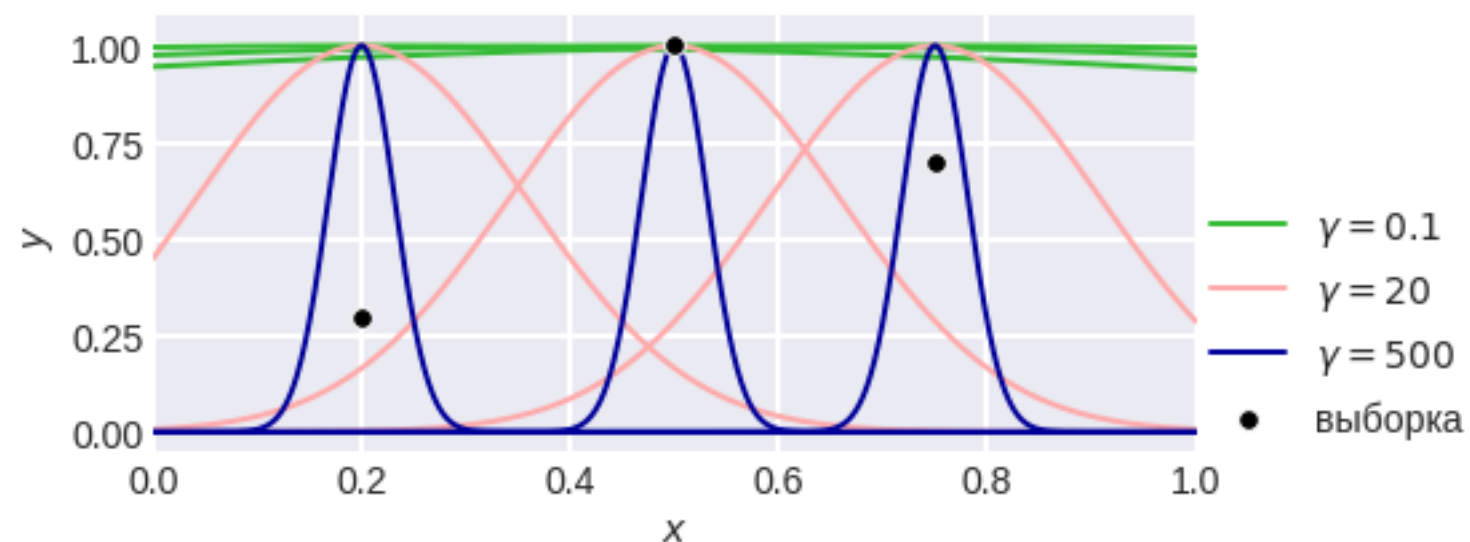
тут, правда, выбраны равномерные эталонные точки



## RBF-ядро: эффект от ширины ядра



## Ядра с разной шириной – центры в выборке



## Локальные методы

уже была регрессия Надарая-Ватсона (называют Kernel regression)

**Аналогичная идея: прогноз в точке**

- по окрестности точки
- по взвешенной выборке (веса  $\sim 1 / \text{расстояние до точки}$ )

**Можно использовать линейный метод**

**или полиномиальный**

**Local Regression = Local Polynomial Regression = Moving Regression**

**см. LOESS – locally estimated scatterplot smoothing (Savitzky–Golay filter)**

**LOWESS – locally weighted scatterplot smoothing (locally weighted polynomial regression)**

<https://towardsdatascience.com/loess-373d43b03564>

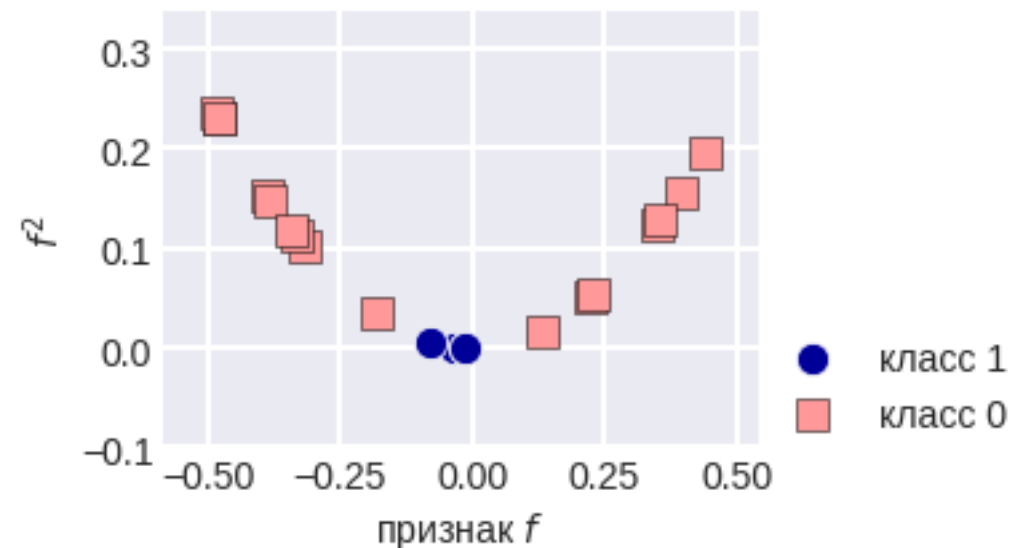
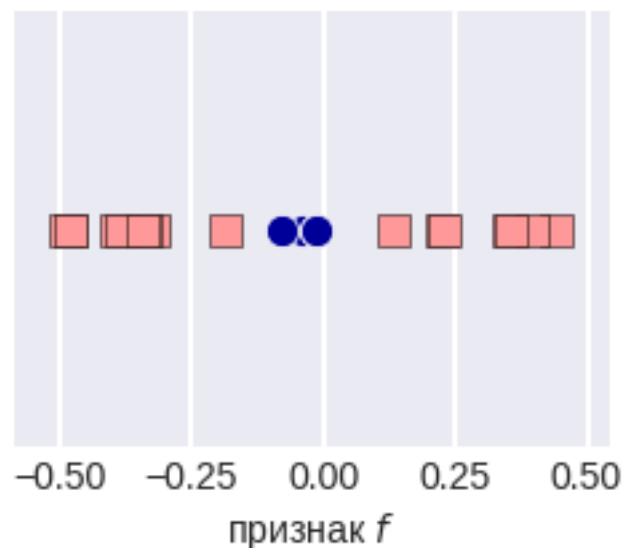
**вернёмся в **предобработке данных****



## Ядерные методы (Kernel Tricks)

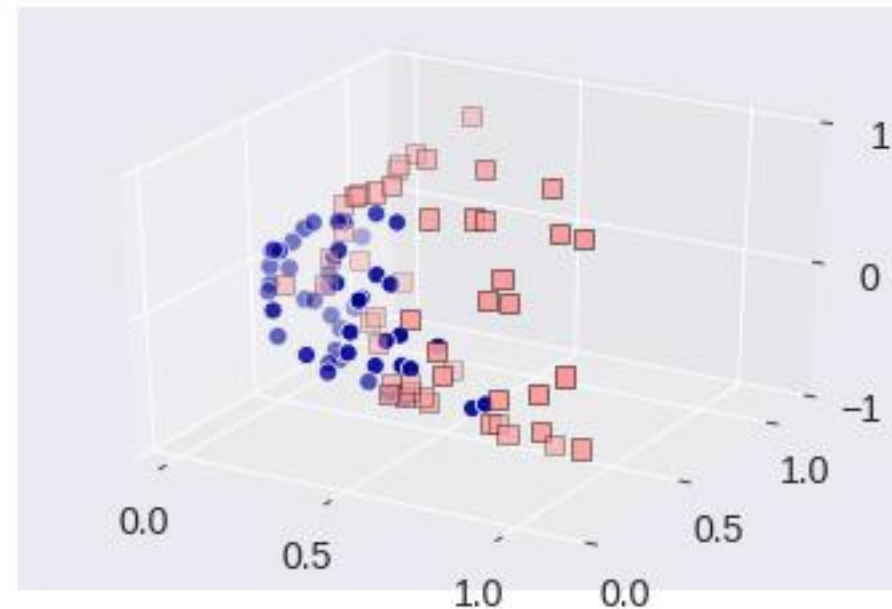
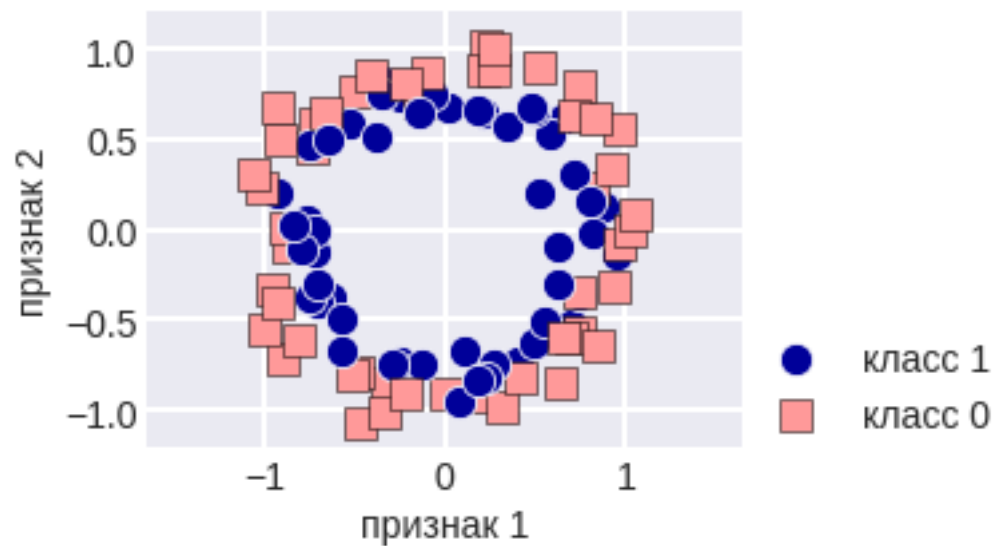
**Идея «искривить/деформировать пространство» –  
перейти в пространство признаков,  
где уже можно решить линейными методами**

**Пример перехода  $(x) \rightarrow (x, x^2)$**



## Ядерные методы (Kernel Tricks)

Пример перехода  $(x_1, x_2) \rightarrow (x_1^2, \sqrt{2}x_1x_2, x_2^2)$



**Переход к пространству мономов ограниченной степени  
может быть очень трудоёмким,  
тут и спасают kernel tricks**

**Пример: SVM**

$$\sum_{i=1}^m \alpha_i - \frac{1}{2} \sum_{i=1}^m \sum_{j=1}^m \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j \rightarrow \max_{0 \leq \alpha \leq C}$$
$$\sum_{i=1}^m \alpha_i y_i = 0$$

**от обучающей выборки нужны только попарные скалярные произведения!**

**В некоторых методах надо знать не значения признаков,  
а уметь вычислять скалярные произведения признаковов описаний некоторых объектов**

## Kernel Tricks

**Пусть**  $x = (x_1, \dots, x_n)^T$ ,  $z = (z_1, \dots, z_n)^T$   
**рассмотрим функцию**  $K(x, z) = (x^T z)^2$

$$K(x, z) = (x_1 z_1 + \dots + x_n z_n)^2 = x_1^2 z_1^2 + \dots + x_n^2 z_n^2 + \sum_{ij} \sqrt{2} x_i x_j \sqrt{2} z_i z_j = \varphi(x)^T \varphi(z)$$

**где**  $\varphi(x) = (x_1^2, \dots, x_n^2, \dots, \sqrt{2} x_i x_j, \dots),$   
 $\varphi(z) = (z_1^2, \dots, z_n^2, \dots, \sqrt{2} z_i z_j, \dots)$

**Чтобы перейти в пространство всех мономов степени 2 не надо явно строить признаки,  
достаточно возвести в квадрат скалярное произведение...**

## Ядро (Kernel) – определение по сути

**Ядро (Kernel) – функция  $K : X \times X \rightarrow \mathbb{R}$  такая, что существует функция**

$$\varphi : X \rightarrow F, \text{ что } K(x, z) = \varphi(x)^T \varphi(z)$$

**$F$  – гильбертово пространство**

**Ядра позволяют делать переходы  
в многомерное пространство неявно!**

## Матрица ядра (kernel matrix):

$$\| K(x_i, x_j) \|_{m \times m} = \begin{bmatrix} K(x_1, x_1) & \cdots & K(x_1, x_m) \\ \cdots & \cdots & \cdots \\ K(x_m, x_1) & \cdots & K(x_m, x_m) \end{bmatrix}$$

**метод называется кернализованным (kernelized),  
если для его реализации достаточно только знания значений скалярных произведений  
(на признаковые описания объектов обучения)**

## Ядро (Kernel) – определение формальное

функция  $K : X \times X \rightarrow \mathbb{R}$  является неотрицательно определённым ядром (positive semidefinite kernel) тогда и только тогда, когда

1) это симметричная функция:

$$K(x, z) = K(z, x),$$

2) для любой обучающей выборки  $\{x_1, \dots, x_m\}$  матрица ядра неотрицательно определена:

$$\forall w \in \mathbb{R}^m \quad w^T \parallel K(x_i, x_j) \parallel w \geq 0$$

(~ неотрицательные с.з.)

– условия Мерсера (Mercer's conditions)

**ДЗ Попробуйте доказать эквивалентность приведённых определений – это и есть теорема Мерсера (хотя бы в одну сторону ;)**

## Примеры ядер

**Однородные полиномиальные**  
**(homogeneous polynomial kernel)**  
степени  $d$

$$K(x, z) = (x^T z)^d$$

**Неоднородные полиномиальные**  
**(inhomogeneous polynomial kernel)**  
степени  $d$

$$K(x, z) = (x^T z + 1)^d$$

**можно + const > 0**

**большие  $d$  ничем не сложнее  $d = 1$**

**«Универсальное»**  
**полиномиальное ядро**

$$K(x, z) = \frac{1}{1 - \alpha^2 x^T z}$$

**RBF-ядро**  
**(Gaussian Radial Basis Function)**

$$K(x, z) = \exp(-\gamma \|x - z\|^d)$$

**константа 1, сумма, произведение ядер и умножение ядра на положительное число  
также будет ядром**

**Обоснование «универсального» ядра**

$$K(x, z) = \frac{1}{1 - \alpha^2 x^T z}$$

$$\begin{aligned} \frac{1}{1 - \alpha^2 x^T z} &= \sum_{i=0}^{+\infty} (\alpha^2 x^T z)^i = 1 + \alpha^2 x^T z + \alpha^4 (x^T z)^2 + \dots = \\ &= 1 + \alpha^2 (x_1 z_1 + \dots + x_n z_n) + \alpha^4 \left( \sum c_* x_i x_j z_i z_j \right) + \dots = \\ &= \varphi(x)^T \varphi(z) \end{aligned}$$

$$\varphi(x) = [1, \alpha x_1, \alpha x_2, \dots, \alpha x_n, \dots, \alpha^2 \sqrt{c_*} x_i x_j, \dots]^T$$

**Пространство бесконечномерное,  
а вычисляем за конечное время!**



**RBF-ядро соответствует переходу в многомерное пространство**

Пусть для простоты  $\gamma = 1$ ,  $d = 2$ ,  $x, z \in \mathbb{R}$

$$\begin{aligned} K(x, z) &= \exp(-(x - z)^2) = \\ &= \exp(-x^2 + 2xz - z^2) = \\ &= \exp(-x^2) \left( \sum_{k=0}^{\infty} \frac{2^k x^k z^k}{k!} \right) \exp(-z^2) = \\ &= \left( \sum_{k=0}^{\infty} \frac{2^{k/2} \exp(-x^2) x^k}{\sqrt{k!}} \cdot \frac{2^{k/2} \exp(-z^2) z^k}{\sqrt{k!}} \right) \end{aligned}$$

## Пример использования в SVM

в классике...

$$a(x) = \text{sgn}(w^T x) = \text{sgn} \left( \sum_{i \in S} \alpha_i y_i x_i^T x + b \right)$$

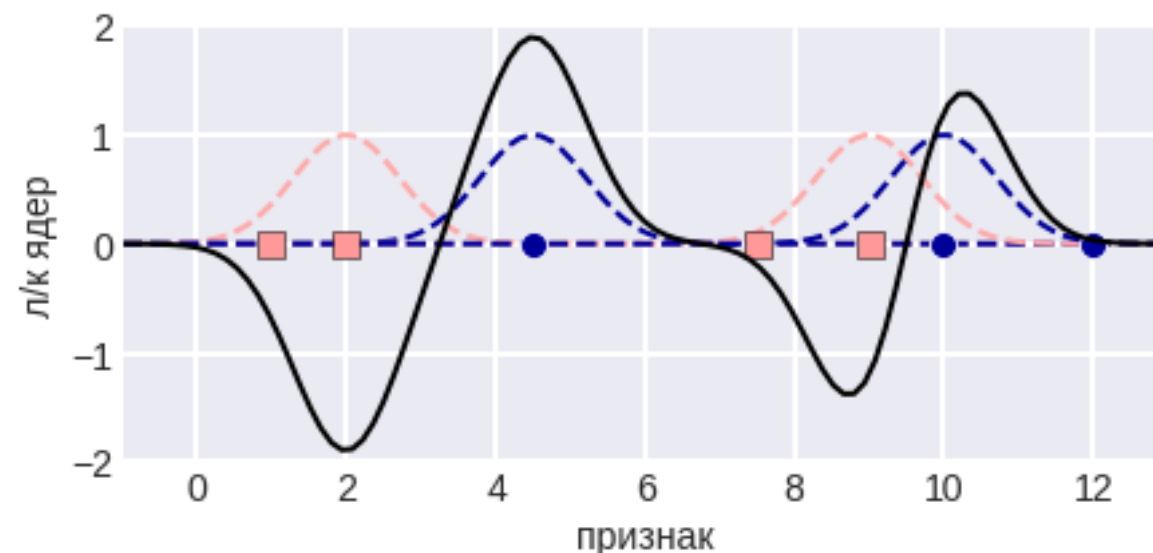
теперь...

$$\begin{aligned} a(x) &= \text{sgn} \left( \sum_{i \in S} \alpha_i y_i \varphi(x_i)^T \varphi(x) \right) = \\ &= \text{sgn} \left( \sum_{i \in S} \alpha_i y_i K(x_i, x) \right) \end{aligned}$$

надо знать опорные векторы  
для классификации

$S$  – множество их индексов

Теперь получили нелинейную модель с  
помощью линейной!!!



кстати, если использовать смещение

$$\begin{aligned} b &= \frac{1}{|S|} \sum_{i \in S} \left( y_i - \sum_{j \in S} \alpha_j y_j x_j^T x_i \right) \rightarrow \\ &\frac{1}{|S|} \sum_{i \in S} \left( y_i - \sum_{j \in S} \alpha_j y_j K(x_j, x_i) \right) \end{aligned}$$

## Нелинейный метод SVM

Построить  $H = \| y_i y_j K(x_i, x_j) \|_{m \times m}$

Решить

$$-\frac{1}{2} \alpha^T H \alpha + \tilde{1}^T \alpha \rightarrow \max$$

при условиях

$$0 \leq \alpha \leq C, y^T \alpha = 0$$

здесь везде векторная запись

Решение

$$w = \sum_{i=1}^m \alpha_i y_i \varphi(x_i)$$

Можно не выписывать в явном  
виде

т.к. наш классификатор

$$a(x) = \operatorname{sgn} \left( \sum_{i \in S} \alpha_i y_i K(x_i, x) \right)$$

**Наблюдение: запись напоминает простейшую нейронную сеть [Воронцов]**

$$a(x) = \operatorname{sgn} \left( \sum_{i \in S} \alpha_i y_i K(x_i, x) \right)$$

**+ при настройке единственное решение**

**+ автоматическое определение числа нейронов в скрытом слое  $|S|$**

**– непонятно, как выбрать ядро, параметры метода, признаки (это не автоматизировано)**

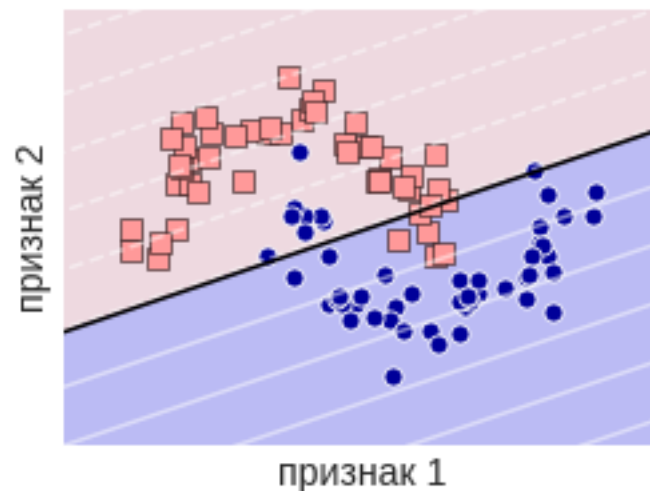
**– использование ядер может вызывать переполнения...**

**– если мы сами придумаем пространство, в которое хотим перейти,  
то, скорее всего, не найдём подходящего ядра**

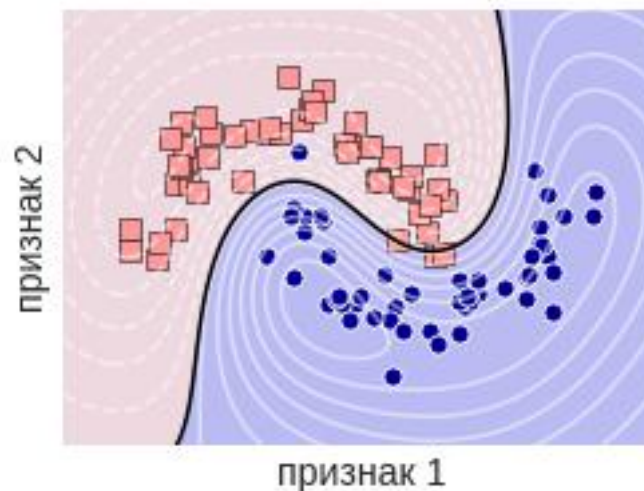
**– геометрия в новом пространстве нам, на самом деле,  
не совсем интуитивно ясна**

## SVM с разными ядрами

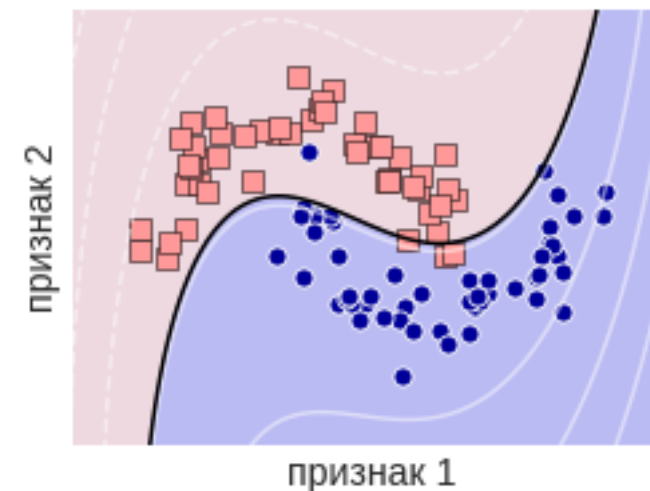
linear



RBF



poly



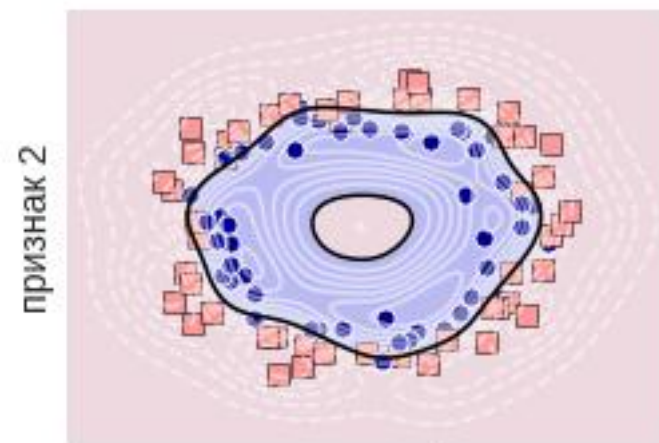
## Минутка кода

```
from sklearn.svm import SVC  
svm = SVC(kernel='rbf', gamma=1.0)  
svm.fit(X, y)
```

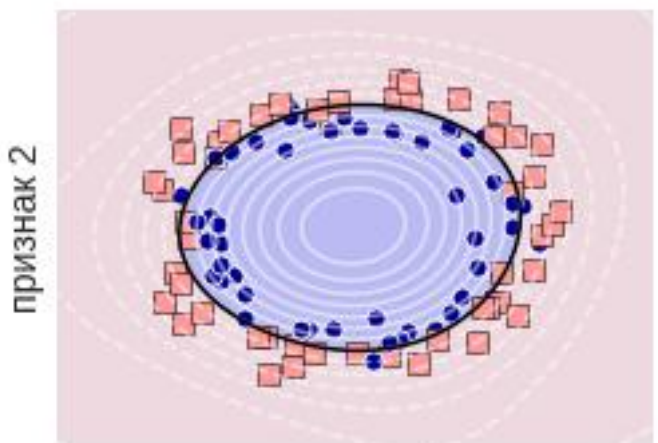
**Ниже обратите внимание на нелогичности разделений и возможность «обмана алгоритма»**

SVM с RBF-ядрами разной ширины

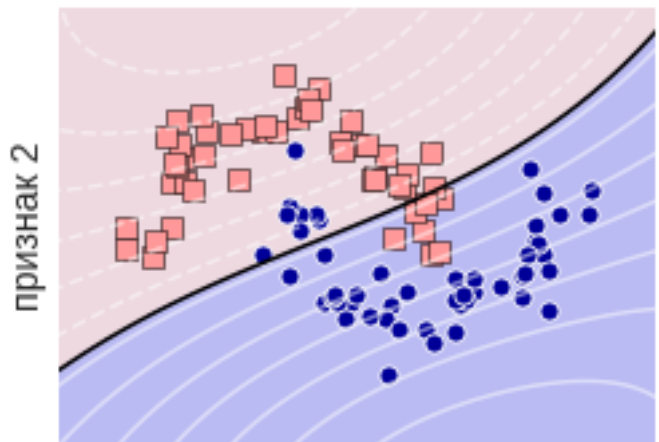
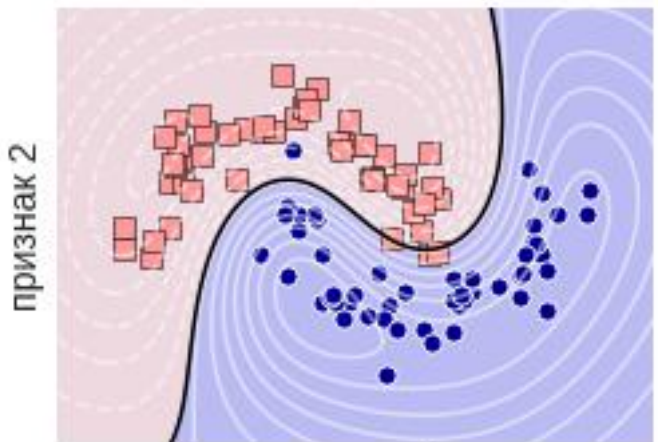
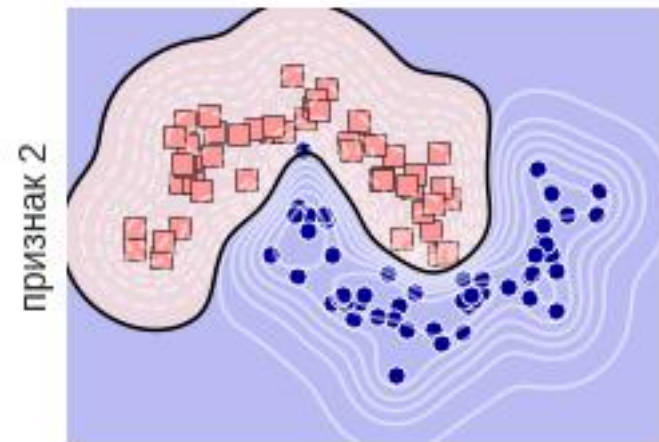
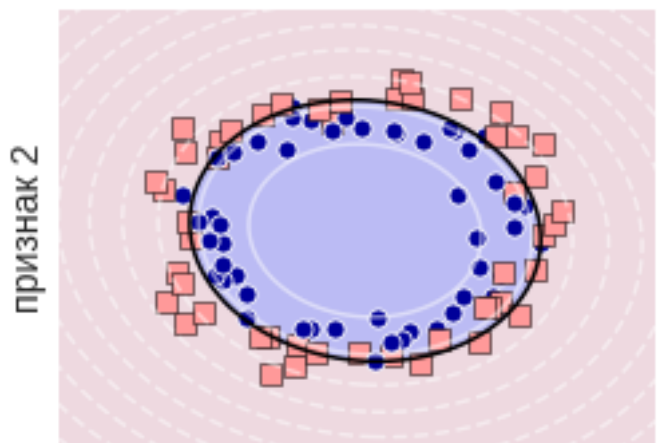
$\gamma = 10$



$\gamma = 1$



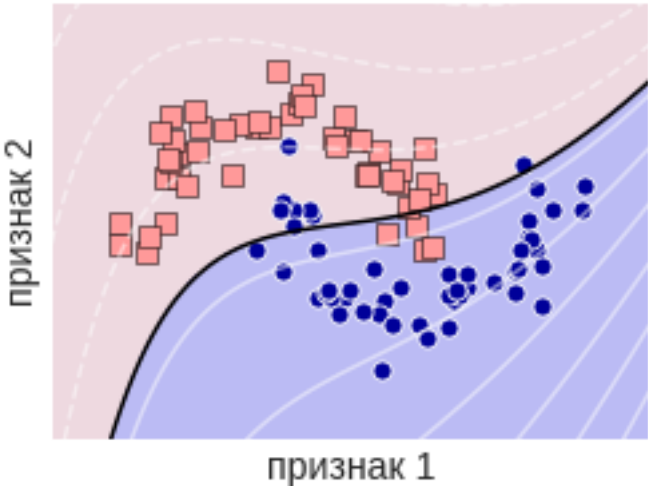
$\gamma = 0.1$



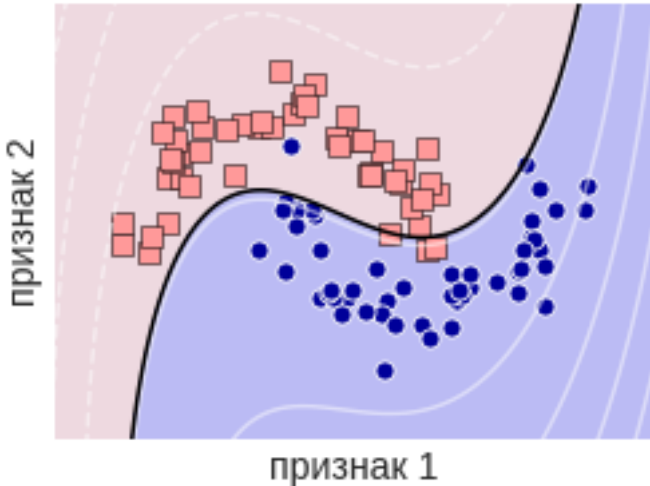


SVM с регуляризацией

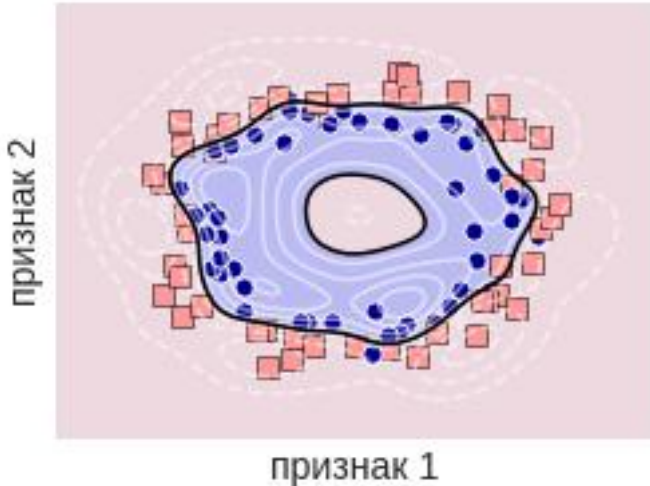
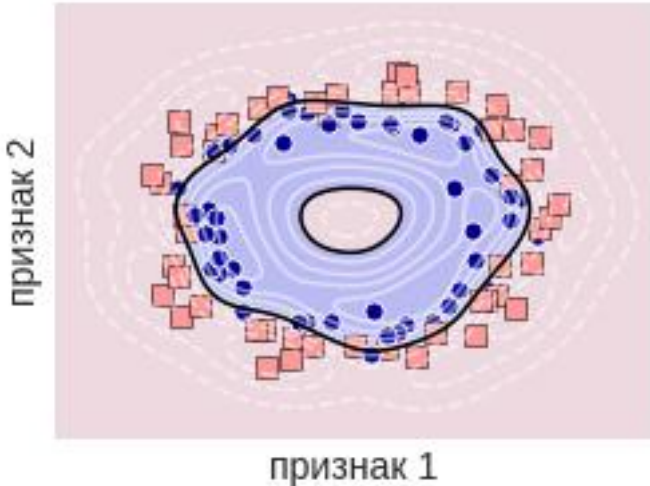
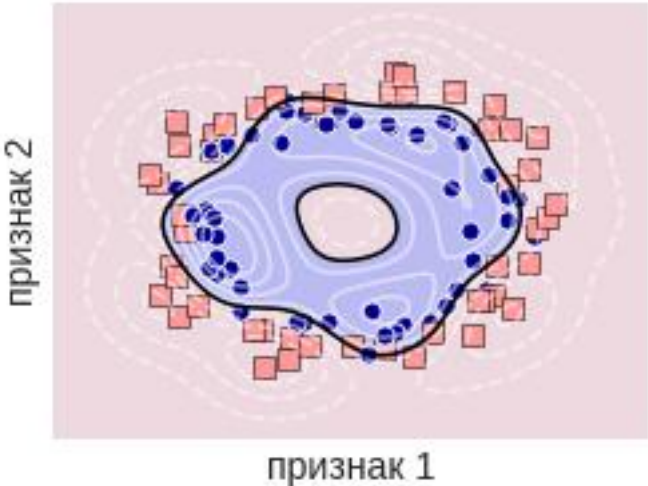
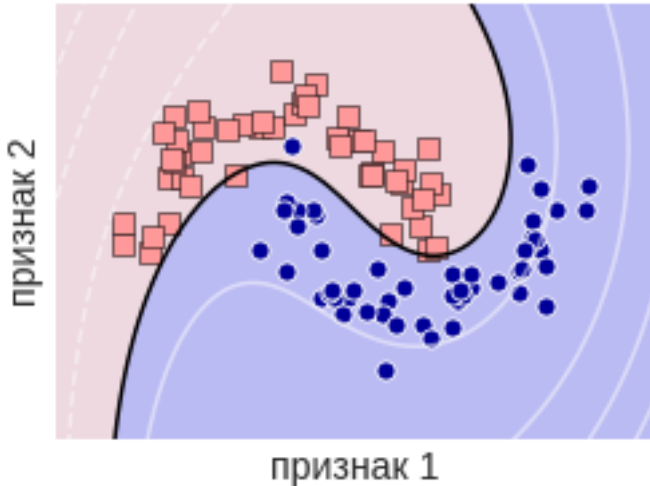
$C = 0.1$



$C = 1$

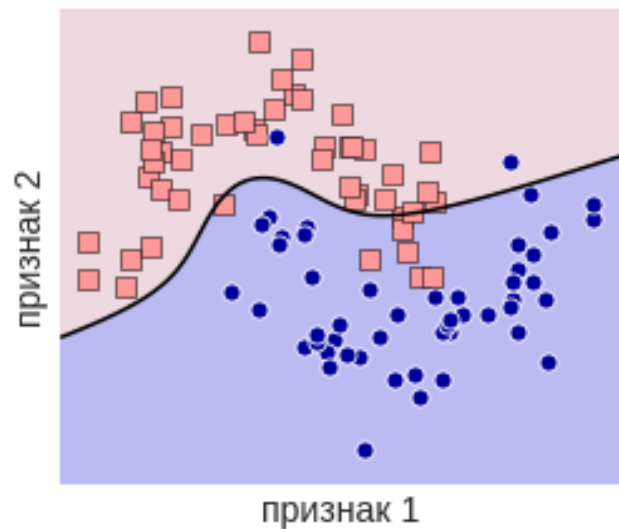


$C = 10$

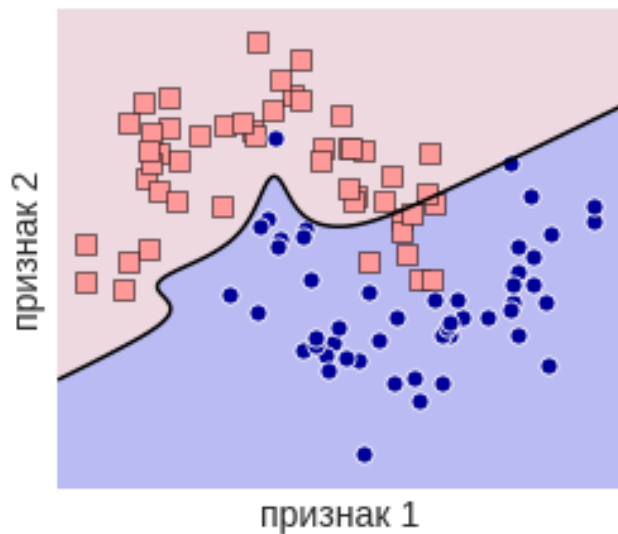


SVM с разной степенью полинома

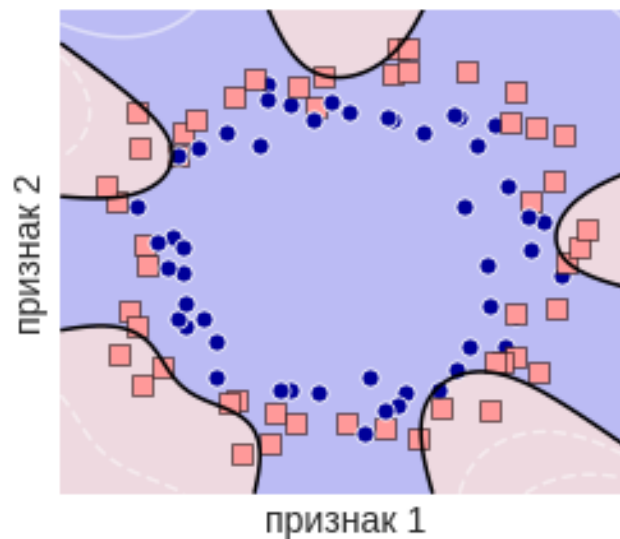
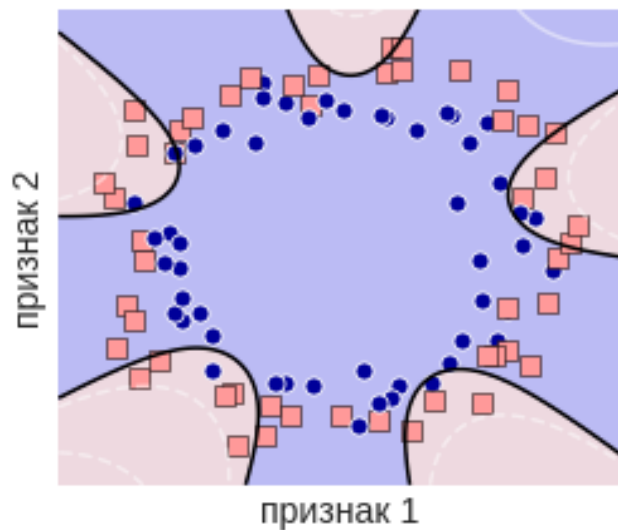
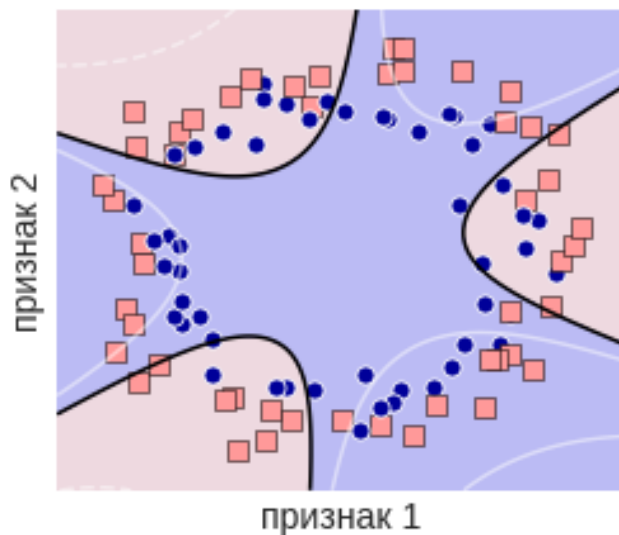
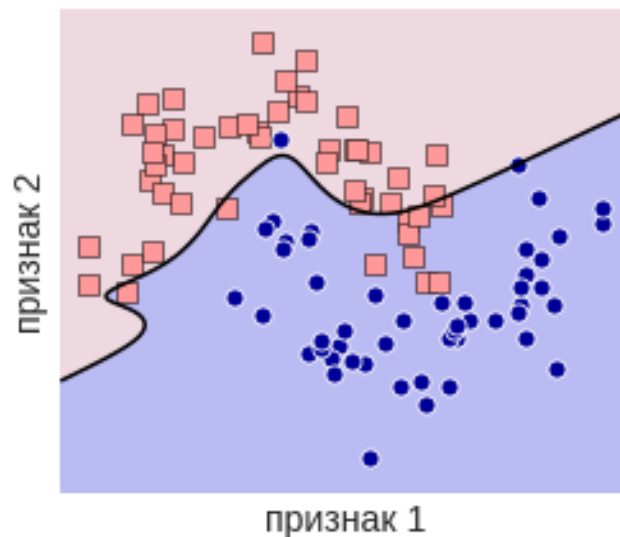
$d = 3$



$d = 5$



$d = 7$





## Кернализация гребневой регрессии

**Метод можно «кернализовать», если он использует только попарные скалярные произведения признаков описаний объектов.**

**Тогда делается замена:**

$$x^T z \rightarrow K(x, z)$$

### Гребневая регрессия

$$w = \arg \min_w \sum_{i=1}^m (y_i - w^T x_i)^2 + \lambda w^T w$$

**решение может быть записано как**

$$w = (X^T X + \lambda I_{n \times n})^{-1} X^T y$$

**Вроде здесь не используем попарные скалярные произведения (элементы матрицы  $XX^T$ ), но... решение можно переписать!**

$$w = X^T (X X^T + \lambda I_{m \times m})^{-1} y$$

## доказательство Representer theorem (попробуйте сами)

$$(X^T X + \lambda I_{n \times n})w = X^T y \Leftrightarrow w = (X^T X + \lambda I_{n \times n})^{-1} X^T y$$

**SVD:**  $X = U \Lambda V^T$  (используем полное разложение  $(m \times m)(m \times n)(n \times n)$ )

$$\begin{aligned} (X^T X + \lambda I_{n \times n})w &= X^T y \\ ((U \Lambda V^T)^T U \Lambda V^T + \lambda I)w &= (U \Lambda V^T)^T y \\ (V \Lambda^T U^T U \Lambda V^T + \lambda I)w &= V \Lambda^T U^T y \\ (V \Lambda^T \Lambda V^T + \lambda I)w &= V \Lambda^T U^T y \\ V^T (V \Lambda^T \Lambda V^T + \lambda I)w &= V^T V \Lambda^T U^T y \\ (\Lambda^T \Lambda V^T + \lambda V^T)w &= \Lambda^T U^T y \\ (\Lambda^T \Lambda + \lambda I)V^T w &= \Lambda^T U^T y \\ V^T w &= (\Lambda^T \Lambda + \lambda I)^{-1} \Lambda^T U^T y \\ V V^T w &= V (\Lambda^T \Lambda + \lambda I)^{-1} \Lambda^T U^T y \\ w &= V (\Lambda^T \Lambda + \lambda I_{n \times n})^{-1} \Lambda^T U^T y \end{aligned}$$

$$\begin{aligned} w &= X^T (X X^T + \lambda I_{m \times m})^{-1} y \\ w &= V \Lambda^T U^T (U \Lambda V^T V \Lambda^T U^T + \lambda I_{m \times m})^{-1} y \\ w &= V \Lambda^T U^T (U \Lambda \Lambda^T U^T + \lambda U U^T)^{-1} y \\ w &= V \Lambda^T U^T (U (\Lambda \Lambda^T + \lambda I_{m \times m}) U^T)^{-1} y \\ w &= V \Lambda^T U^T U (\Lambda \Lambda^T + \lambda I_{m \times m})^{-1} U^T y \\ w &= V \Lambda^T (\Lambda \Lambda^T + \lambda I_{m \times m})^{-1} U^T y \end{aligned}$$

Осталось доказать,  
что синие матрицы равны;  
 $\text{diag}(\lambda_1 / (\lambda_1^2 + \lambda), \dots, \lambda_k / (\lambda_k^2 + \lambda))_{n \times m}$   
 $k = \min(m, n)$

## Кернализация гребневой регрессии

**сравним число операций...**

$$w = (X^T X + \lambda I_{n \times n})^{-1} X^T y \qquad w = X^T (X X^T + \lambda I_{m \times m})^{-1} y$$

**время (time)**

$$O(mn^2 + n^3)$$

$$O(m^2 n + m^3)$$

**память (space)**

$$O(mn \vee n^2)$$

$$O(mn \vee m^2)$$

**Кстати**

$$w = X^T \underbrace{(X X^T + \lambda I_{m \times m})^{-1} y}_{\equiv \alpha \in \mathbb{R}^m} = \sum_{t=1}^m \alpha_t x_t$$

**отдельный смысл: коэффициенты = л/к объектов**

## Representer theorem

- возможность кернализовать
- другие затраты по времени и памяти
- опять видим, что коэффициенты = л/к объектов
- можно и **SGD** использовать!!! (пока это опустим)

при доказательстве также можно

**Matrix inversion lemma**

$$(FH^{-1}G - E)^{-1}FH^{-1} = E^{-1}F(GE^{-1}F - H)^{-1}$$

## Кернализация гребневой регрессии

$$w = X^T (X X^T + \lambda I_{m \times m})^{-1} y$$

**тогда**  $w = X^T \alpha = \sum_{i=1}^m \alpha_i x_i \rightarrow \sum_{i=1}^m \alpha_i \varphi(x_i)$ , где

$$\alpha = (X X^T + \lambda I_{m \times m})^{-1} y \rightarrow (\| K(x_i, x_j) \|_{m \times m} + \lambda I_{m \times m})^{-1} y,$$

**Ответ нашей kernel-ridge-регрессии:**

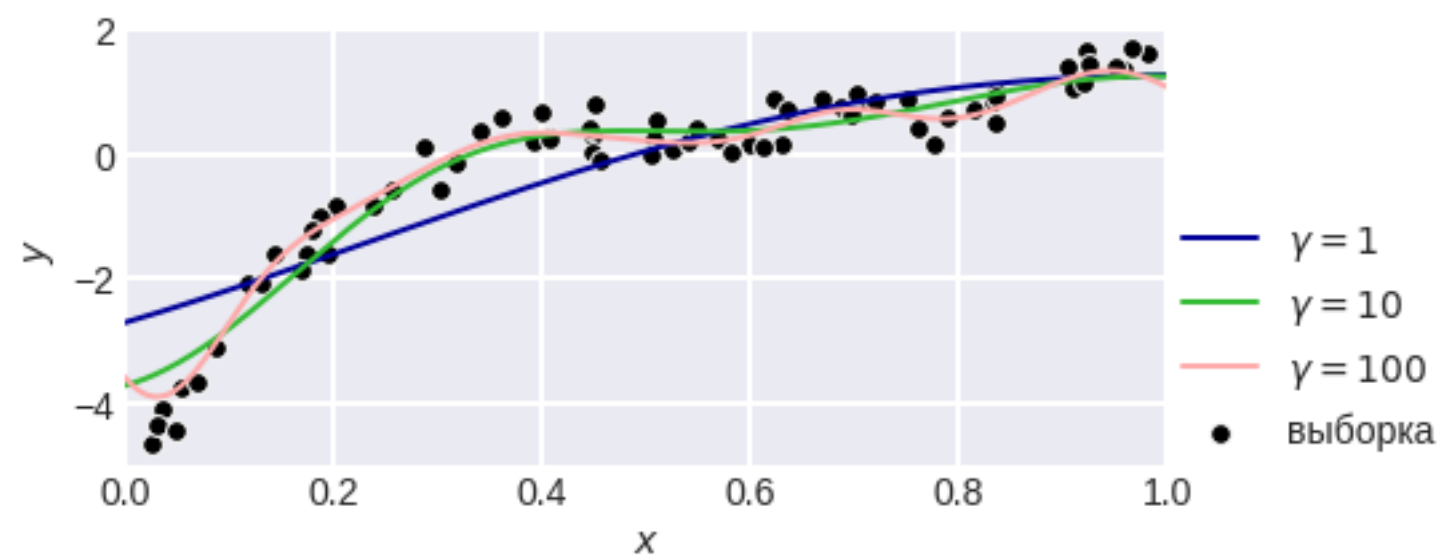
$$a(x) = w^T x \rightarrow w^T \varphi(x) = \sum_{i=1}^m \alpha_i \varphi(x_i)^T \varphi(x) = \sum_{i=1}^m \alpha_i K(x_i, x)$$

**для справки:** Hal Daume' «From Zero to Reproducing Kernel Hilbert Spaces in Twelve Pages or Less» <http://users.umiacs.umd.edu/~hal/docs/daume04rkhs.pdf>

Минутка кода

```
sklearn.kernel_ridge.KernelRidge
```

kernel="linear"	Ядро
alpha=1	Коэффициент регуляризации
gamma=None	Параметр для ядер RBF-типа
degree=3	Параметр для полиномиального ядра
coef0=1	Параметр для полиномиального ядра и сигмоиды
kernel_params	Параметр для пользовательского ядра



## Кернализации с нестандартными данными

**kernels on probability distributions**

**kernels on strings**

**kernels on functions**

**kernels on groups**

**kernels on graphs**

## Кернализация в анализе последовательностей

```
x = "ACAGCAGTA"
z = "AGCAAGCGAG"

from collections import Counter
def phi(x):
    """
    пространство 3-подпоследовательностей
    """
    cnt = Counter()
    for i in range(len(x)-2):
        cnt[x[i: i+3]] += 1
    return dict(cnt)

phix = phi(x) #{'ACA': 1, 'CAG': 2, 'AGC': 1, 'GCA': 1, 'AGT': 1, 'GTA': 1}
phiz = phi(z) #{'AGC': 2, 'GCA': 1, 'CAA': 1, 'AAG': 1, 'GCG': 1, 'CGA': 1, 'GAG': 1}

def phidot(phix, phiz):
    """
    скалярное произведение в новом пространстве
    """
    return ({name: phix[name] * phiz[name] for name in set(phix.keys()) & set(phiz.keys())})

x_dot_y = phidot(phix, phiz) # {'GCA': 1, 'AGC': 2}
sum(list(x_dot_y.values())) # 3
```



## Математика ядер – операции над ядрами

**С ядрами можно делать много «неявных» операций**

**Пример: вычислить разброс в новом пространстве**

$$\sigma_{\varphi}^2 = \frac{1}{m} \sum_{i=1}^m \|\varphi(x_i) - \mu_{\varphi}\|^2$$

**сначала вычислим**  $\|\varphi(x_i) - \mu_{\varphi}\|^2 =$

$$= (\varphi(x_i) - \mu_{\varphi})^T (\varphi(x_i) - \mu_{\varphi}) = \varphi(x_i)^T \varphi(x_i) - 2\varphi(x_i)^T \mu_{\varphi} + \mu_{\varphi}^T \mu_{\varphi} =$$

$$= K(x_i, x_i) - \frac{2}{m} \sum_{j=1}^m \underbrace{\varphi(x_i)^T \varphi(x_j)}_{K(x_i, x_j)} + \left( \frac{1}{m} \sum_{j=1}^m \varphi(x_j) \right)^T \left( \frac{1}{m} \sum_{j=1}^m \varphi(x_j) \right) =$$

$$= K(x_i, x_i) - \frac{2}{m} \sum_{j=1}^m K(x_i, x_j) + \frac{1}{m^2} \sum_{j=1}^m \sum_{t=1}^m K(x_j, x_t)$$

## Математика ядер – операции над ядрами

поэтому разброс...

$$\begin{aligned}
\sigma_{\varphi}^2 &= \frac{1}{m} \sum_{i=1}^m \left( K(x_i, x_i) - \frac{2}{m} \sum_{j=1}^m K(x_i, x_j) + \frac{1}{m^2} \sum_{j=1}^m \sum_{t=1}^m K(x_j, x_t) \right) = \\
&= \frac{1}{m} \sum_{i=1}^m K(x_i, x_i) - \frac{2}{m^2} \sum_{i=1}^m \sum_{j=1}^m K(x_i, x_j) + \frac{1}{m^2} \sum_{j=1}^m \sum_{t=1}^m K(x_j, x_t) = \\
&= \frac{1}{m} \sum_{i=1}^m K(x_i, x_i) - \frac{1}{m^2} \sum_{i=1}^m \sum_{j=1}^m K(x_i, x_j)
\end{aligned}$$

**Разброс вычисляется через значения ядер,  
такие сущности как «средние» не надо определять в явном виде**

**аналогично(доказать):**  $\| \varphi(x_i) - \varphi(x_j) \|^2 = K(x_i, x_i) + K(x_j, x_j) - 2K(x_i, x_j)$

## Математика ядер – операции над ядрами

**нормировка в новом признаковом пространстве**

$$\varphi(x) \rightarrow \frac{\varphi(x)}{\|\varphi(x)\|}$$

$$K(x_i, x_j) \rightarrow K(x_i, x_j) = \frac{\varphi(x_i)^T \varphi(x_j)}{\|\varphi(x_i)\| \cdot \|\varphi(x_j)\|} = \frac{K(x_i, x_j)}{\sqrt{K(x_i, x_i)K(x_j, x_j)}}$$

**можно имитировать модификацией матрицы ядра:**

$$K \rightarrow W^{-1/2} K W^{-1/2}, \text{ где } W = \text{diag}(K(x_1, x_1), \dots, K(x_m, x_m))$$

## Проблема выбора ядра

**Выбор ядра – экспертно / перебор**

**Настройка гиперпараметров ядра – скользящий контроль**

**Интерпретация ядра:**

$$(K(x_i, x_1), K(x_i, x_2), \dots, K(x_i, x_m))$$

**– можно рассматривать как новое признаковое описание,  
т.е. «сходства» с объектами обучения**

**Но почему именно с этими объектами???**

**⇒ настройка (определение) объектов**

**Не забывать**

- **храним много информации  
например,  $m \times m$ -матрицу**

**есть методы увеличения скорости kernel-методов**

**RBF – суть метода**

$$\| K(x_i, x_j) \|_{m \times m} = \begin{bmatrix} K(x_1, x_1) & \dots & K(x_1, x_m) \\ \dots & \dots & \dots \\ K(x_m, x_1) & \dots & K(x_m, x_m) \end{bmatrix}$$

**Помним... базисность (в RBF) нужна для невырожденности этой матрицы**

$$\| K(x_i, x_j) \| w = y$$

**– система  $m$  уравнений с  $m$  неизвестными**

$$\sum_{i=1}^m w_i \exp(-\gamma \| x_j - x_i \|^2) = y_j, j \in \{1, 2, \dots, m\}$$

$$a(x) = \sum_{i=1}^m w_i \exp(-\gamma \| x - x_i \|^2)$$

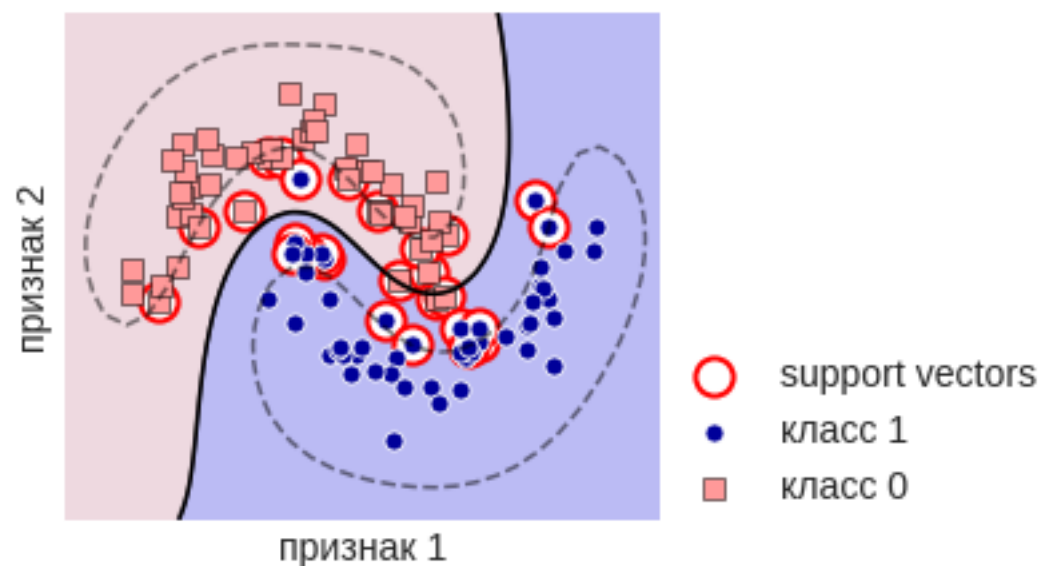
**помним, что аналогично было в SVM...  $a(x) = \text{sgn} \left( \sum_{i \in S} \alpha_i y_i K(x_i, x) \right)$**

## RBF – проблема выбора эталонных точек

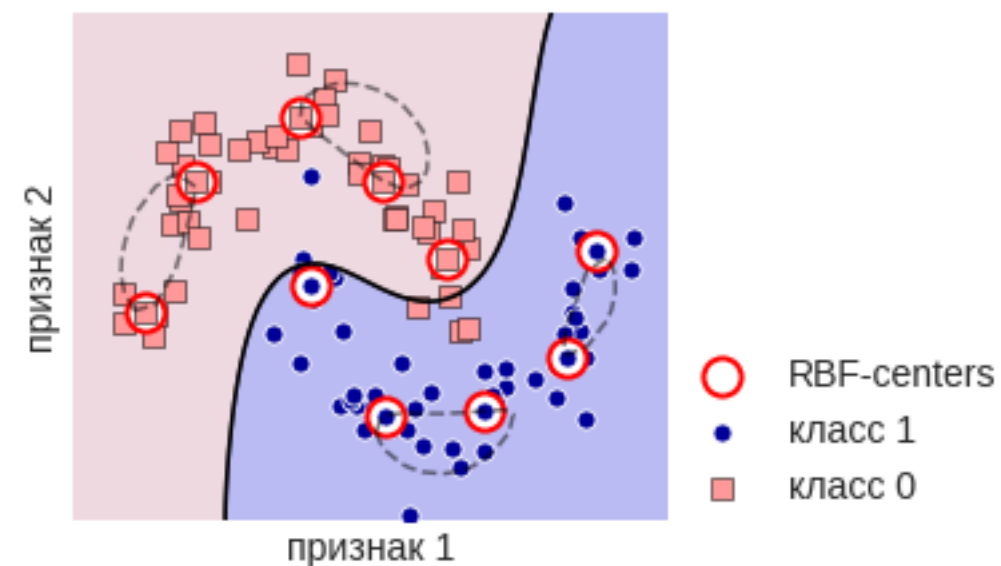
**лучше в качестве центров использовать  $k$  эталонных точек**  
**Как выбрать?**

**один из способов: кластеризация → центры кластеров**

**Support vectors**



**RBF centers**



## RBF – проблема настройки весов

**если точек будет меньше... (собственно, как в SVM)**

$$\sum_{i \in S} w_i \exp(-\gamma \|x_j - x_i\|^2) = y_j, \quad j \in \{1, 2, \dots, m\}$$

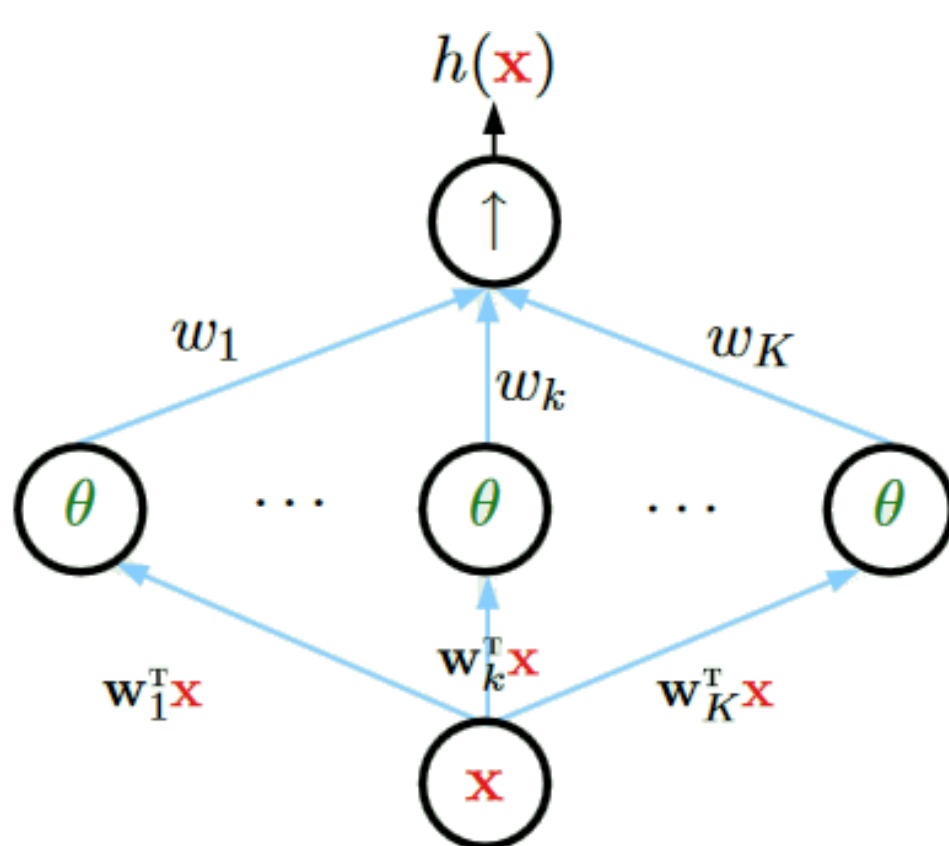
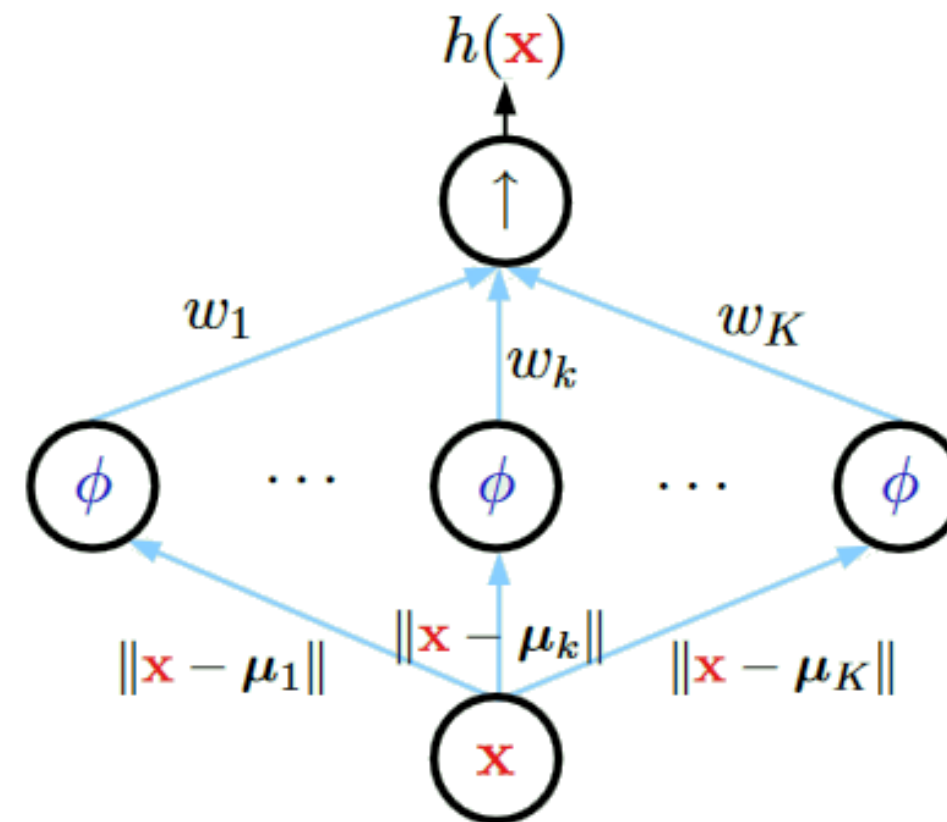
$$\|K(x_i, x_j)\|_{m \times |S|} w = y$$

**решение через псевдообратную (если матрицу обозначить через K):**

$$w = (K^T K + \lambda I)^{-1} K^T y$$

**а практичнее – SGD**

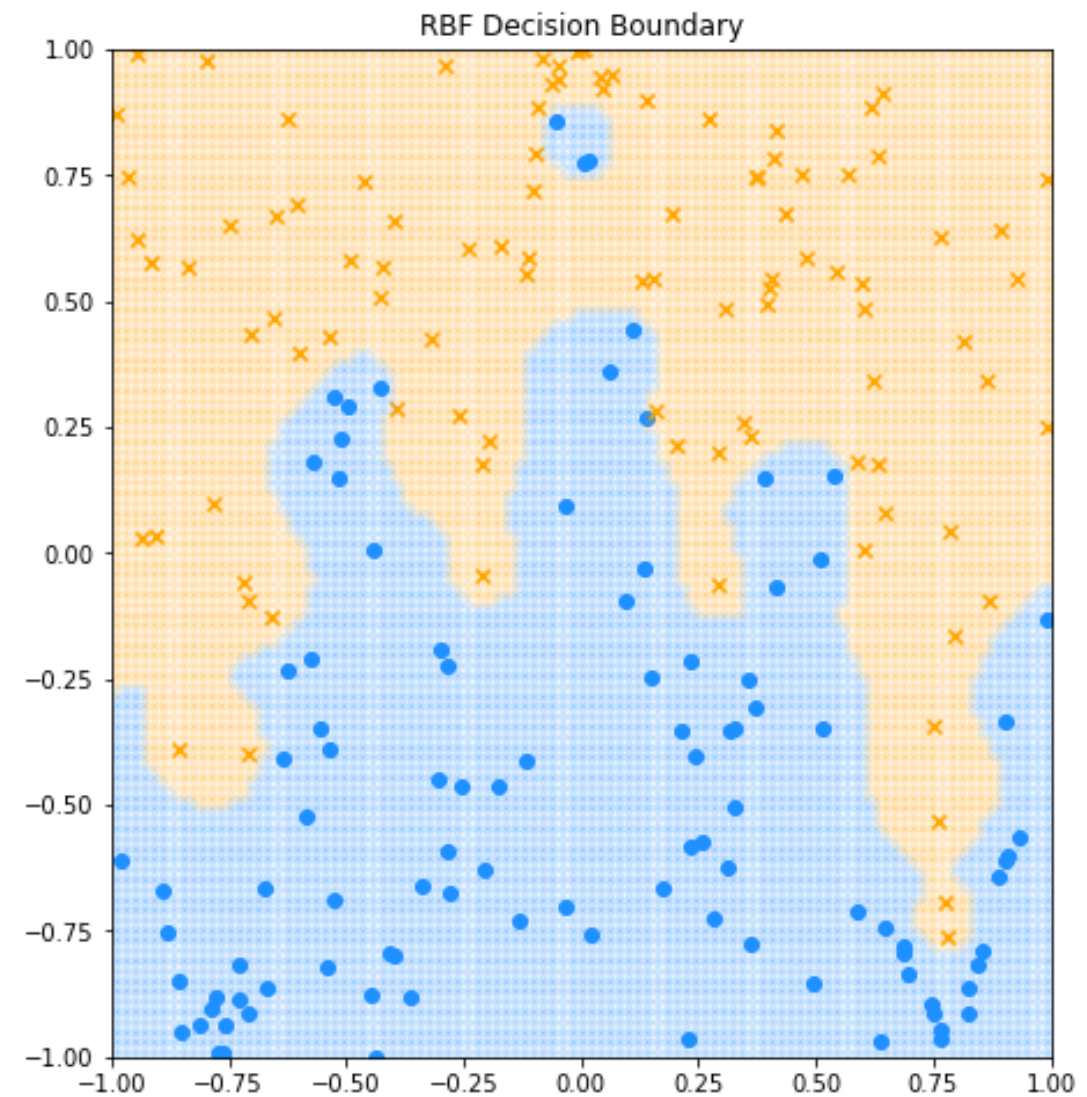
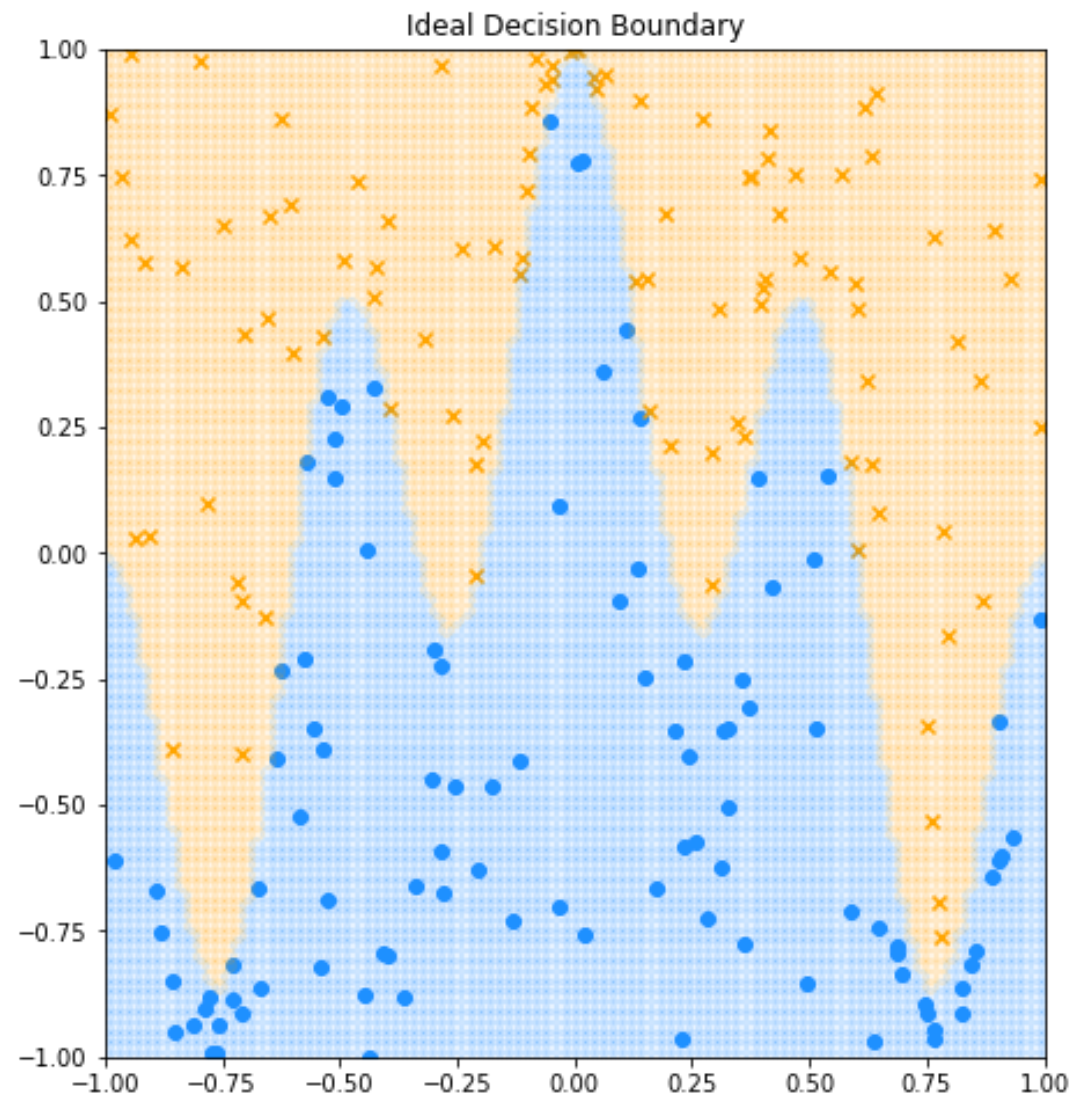
**или RBF-сеть**

**RBF-сеть (Radial basis function network)****нейросеть****RBF-сеть**

<https://www.youtube.com/watch?v=O8CfrnOPtLc>



## RBF-сеть (Radial basis function network)



<https://github.com/JeremyLinux/PyTorch-Radial-Basis-Function-Layer>

Реализации методов, основанных на ядрах

FALKON, 2017

$O(m)$  – память

$O(m\sqrt{m})$  – время

идея: случайные подмножества

$$S \subseteq \{1,2,\dots,m\}$$

+ SGD

$$\sum_{i=1}^m \alpha_i K(x_i, x) \rightarrow \sum_{i \in S} \alpha_i K(x_i, x)$$

Algorithm	train time	kernel evaluations	memory	test time
SVM / KRR + direct method	$n^3$	$n^2$	$n^2$	$n$
KRR + iterative [1, 2]	$n^2 \sqrt[4]{n}$	$n^2$	$n^2$	$n$
Doubly stochastic [22]	$n^2 \sqrt{n}$	$n^2 \sqrt{n}$	$n$	$n$
Pegasos / KRR + sgd [27]	$n^2$	$n^2$	$n$	$n$
KRR + iter + precondition [3, 28, 4, 5, 6]	$n^2$	$n^2$	$n$	$n$
Divide & Conquer [29]	$n^2$	$n\sqrt{n}$	$n$	$n$
Nystrom, random features [7, 8, 9]	$n^2$	$n\sqrt{n}$	$n$	$\sqrt{n}$
Nystrom + iterative [23, 24]	$n^2$	$n\sqrt{n}$	$n$	$\sqrt{n}$
Nystrom + sgd [20]	$n^2$	$n\sqrt{n}$	$n$	$\sqrt{n}$
FALKON (see Thm. 3)	$n\sqrt{n}$	$n\sqrt{n}$	$n$	$\sqrt{n}$

Table 1: Computational complexity required by different algorithms, for optimal generalization. Logarithmic terms are not showed.

## Реализации методов, основанных на ядрах

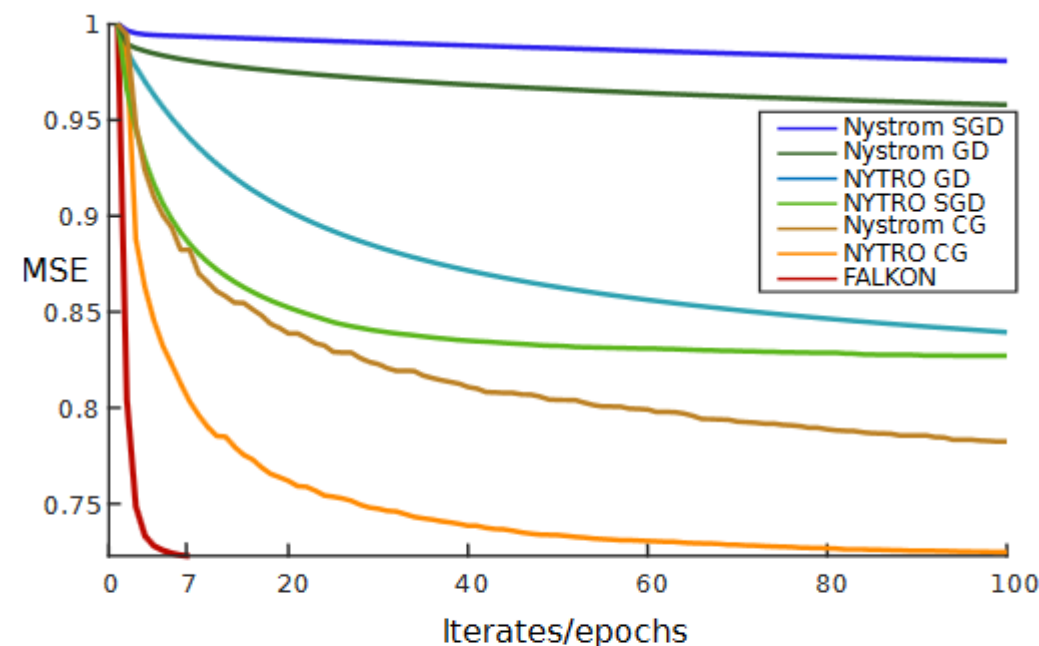


Figure 1: Falkon is compared to stochastic gradient, gradient descent and conjugate gradient applied to Problem (8), while NYTRO refer to the variants described in [23]. The graph shows the test error on the HIGGS dataset ( $1.1 \times 10^7$  examples) with respect to the number of iterations (epochs for stochastic algorithms).

**Alessandro Rudi, Luigi Carratino, Lorenzo Rosasco «FALKON: An Optimal Large Scale Kernel Method» // <https://arxiv.org/pdf/1705.10958.pdf>**

## Дальше

В обучении без учителя будет метод **kernel PCA, kernel k-means (?)**

Также есть **Kernel ICA, Kernel CCA**

**Пример деформации пространства на практике:  $(m \times k) * (k \times n)$** 

**Дано:**  $m$ ,  $n$ ,  $k$ , характеристики ЭВМ

**Целевой признак:** Время перемножения матриц размеров  $m \times k$  и  $k \times n$

**В контроле:** ~5000 Записей, ~950 признаков

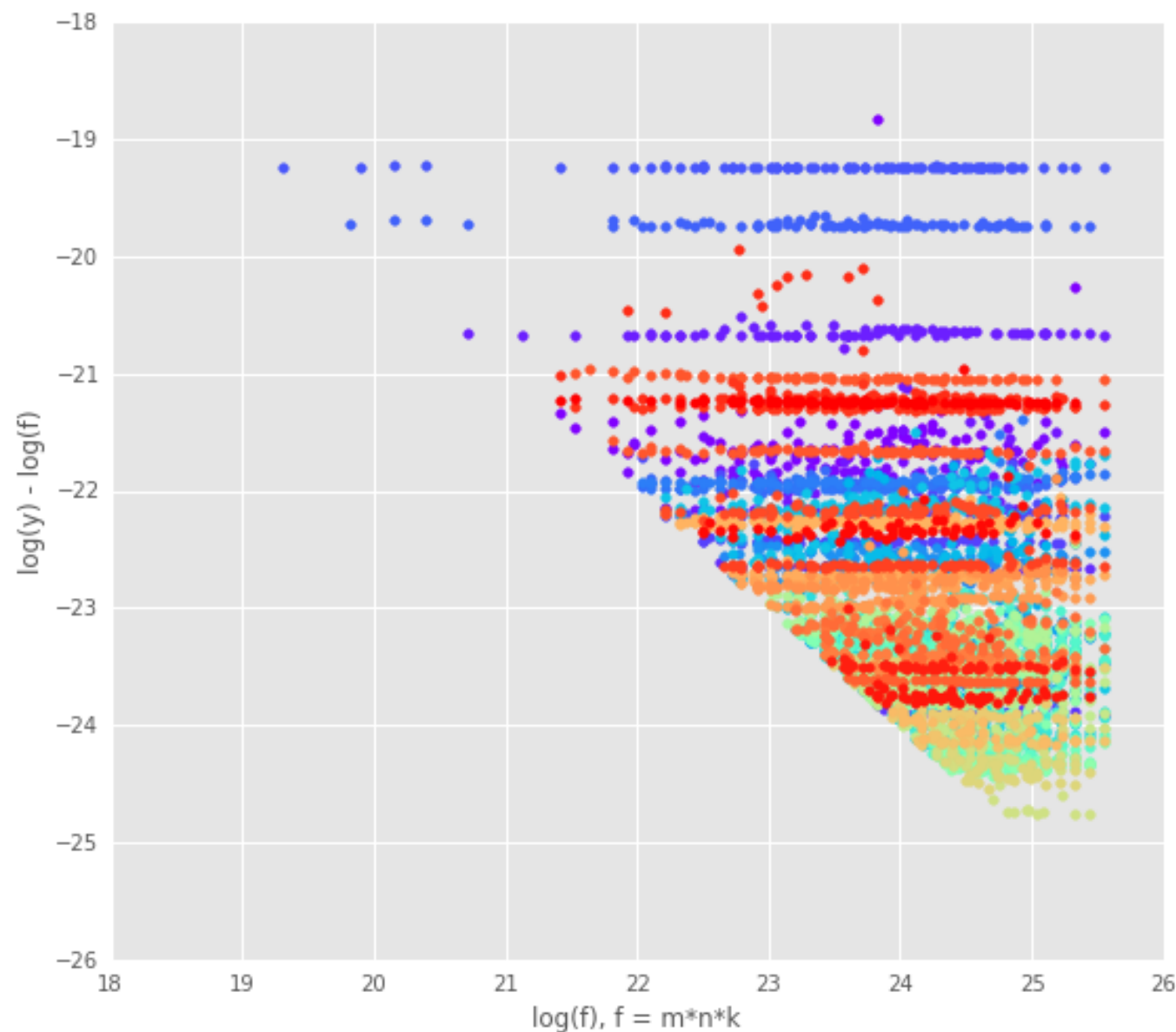
**В тесте:**  $= * =$

**Функционал качества:** MAPE

**Прагматика:** прогнозирование времени вычислений

**Простая логика:** результат произведения –  $mn$  элементов, вычисление каждого –  $k$  умножений, общее число умножений =  $m \times n \times k$

## Пример деформации пространства на практике: $(m \times k) \times (k \times n)$



**Регрессия:**

$$\log(y) - \log(mnk) \approx \text{const}$$

**Пусть**

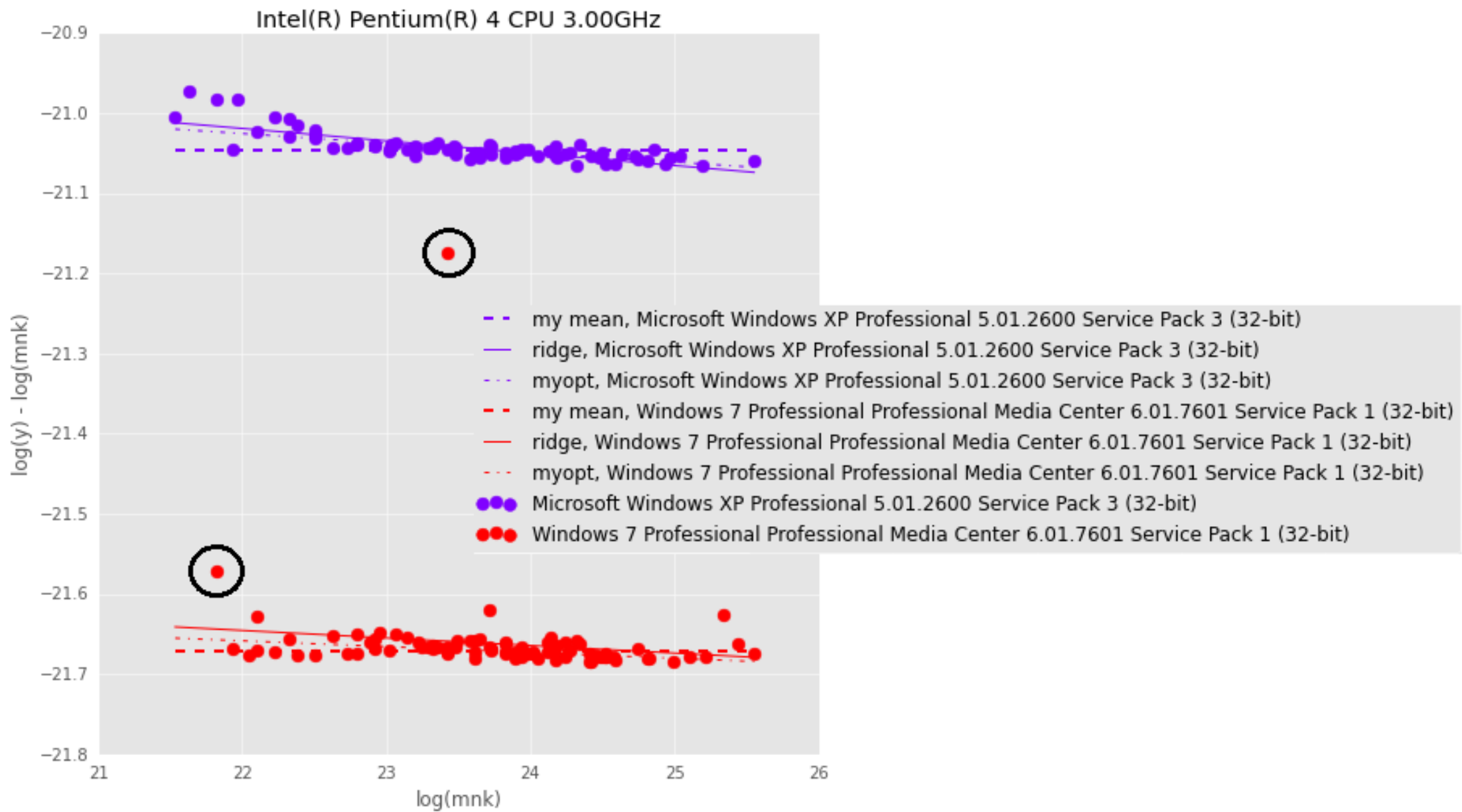
$$\log(y) - \log(mnk) = w_1 \log(mnk) + w_0$$

**потом покажем, почему не const**

**Ответ задачи:**

$$a = e^{w_0} \cdot (mnk)^{w_1+1}$$

Пример деформации пространства на практике:  $(m \times k) * (k \times n)$



## Пример деформации пространства на практике: биология

**Есть предположение**

$$y \approx \frac{\alpha x}{\beta + x}$$

(некоторые биологические процессы так описываются)

**Как решать стандартами методами?**



## Пример деформации пространства на практике: биология

Пусть

$$y = \frac{\alpha x}{\beta + x}$$

**Это линейная регрессия для обратных признаков:**

$$\frac{1}{y} = \frac{\beta}{\alpha} \frac{1}{x} + \frac{1}{\alpha}$$

## Итог

**Можно решать сложные задачи линейными методами  
~ пополнение признакового пространства**

**Есть «автоматические способы пополнения»**

**Многие методы можно «кернализовать»**

## Ссылки

**Scholkopf, B. and Smola, A. J. «Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond» Cambridge, MA: MIT Press, 2002.**

**Scholkopf, B., Tsuda, K., and Vert, J.-P. «Kernel methods in computational biology» Cambridge, MA: MIT press, 2004.**

**Shawe-Taylor, J. and Cristianini, N. «Kernel Methods for Pattern Analysis» New York: Cambridge University Press, 2004.**

**RBF [https://en.wikipedia.org/wiki/Radial\\_basis\\_function](https://en.wikipedia.org/wiki/Radial_basis_function)**