«Машинное обучение»

Нелинейные методы

Александр Дьяконов

План

Что делать, если хотим искать нелинейные закономерности линейными методами

«Трюки с ядрами (kernel tricks)»

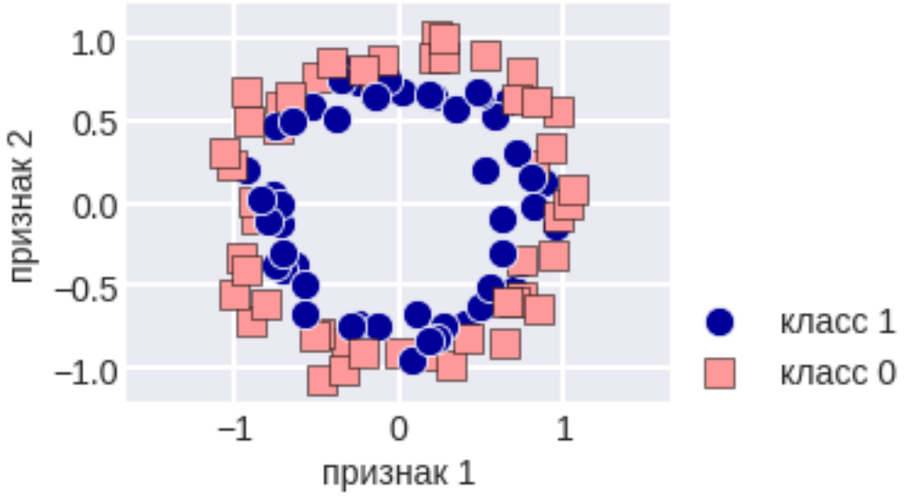
Вывод нелинейного SVM

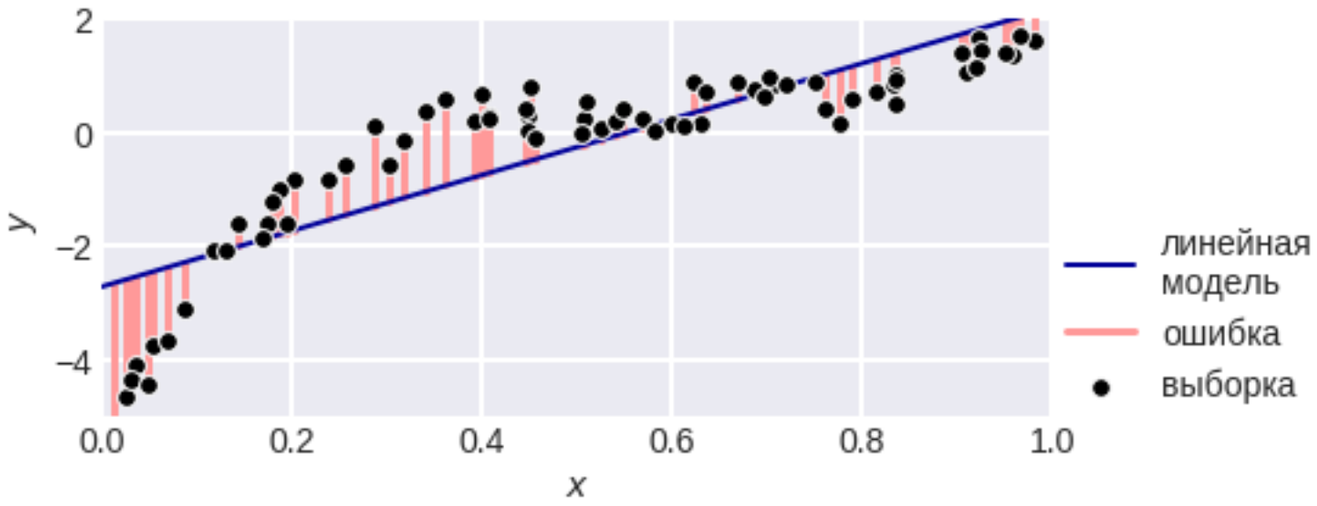
«Кернализация» других методов

Решение задач произвольной природы

Проблема линейности

Некоторые задачи не решаются линейными методами





Когда линейной модели не хватает

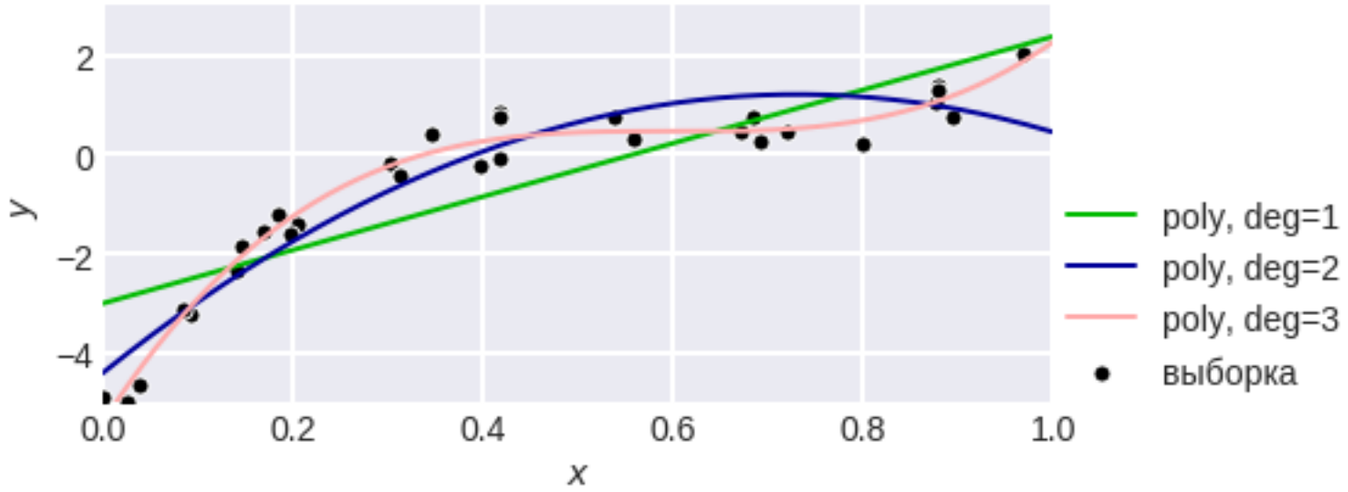
линейная модель + новые признаки

- **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



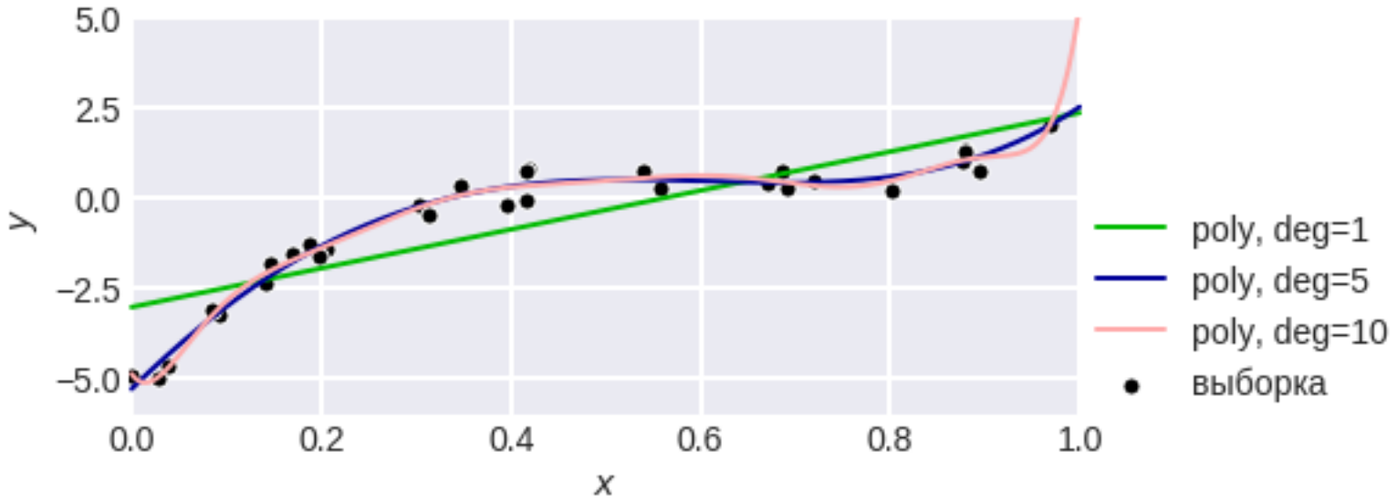
добавляем признаки-мономы

в задаче с одним признаком $(x) \to (1, x, x^2, \ldots, x^k)$

теперь линейная регрессия превращается в $a(x) = w_0 + w_1 x + w_2 x^2 + \ldots + w_k x^k$

с несколькими $(X_1, \ldots, X_n) \to (\ldots, \prod_{t \in T} X_t, \ldots)$

Деформация: полиномиальная модель



подводные камни – очень много признаков добавлять:

1) переобучение

2) неестественные признаки

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(XX))
```

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

Использование других базисных функций

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

$$(X_1,\ldots,X_n) \to (\ldots, f_j(X_1,\ldots,X_n),\ldots)$$

например, характеристические функции интервалов

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

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

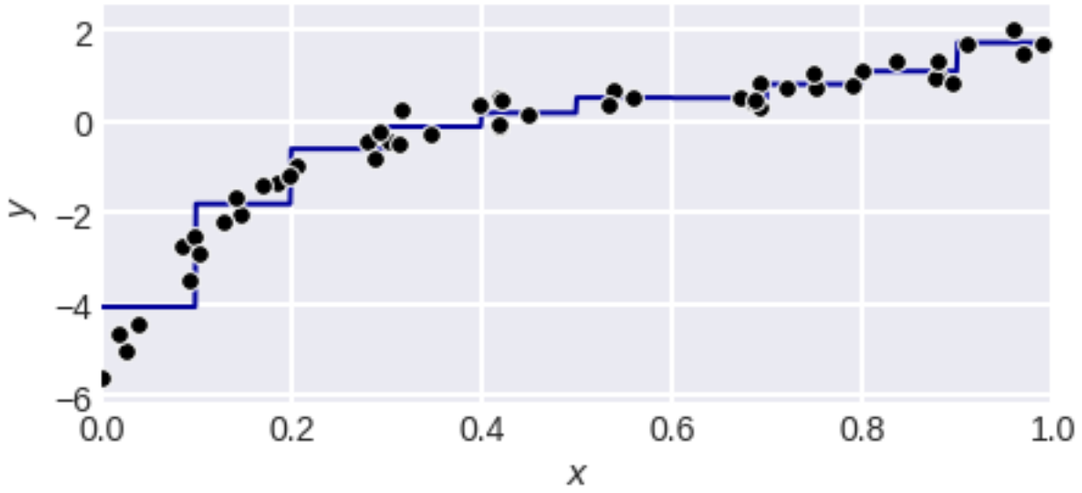
- **квантили**
- **«межкластерные» точки**
- **особые точки**

(например, круглые суммы в признаке «зарплата»)

Использование других базисных функций

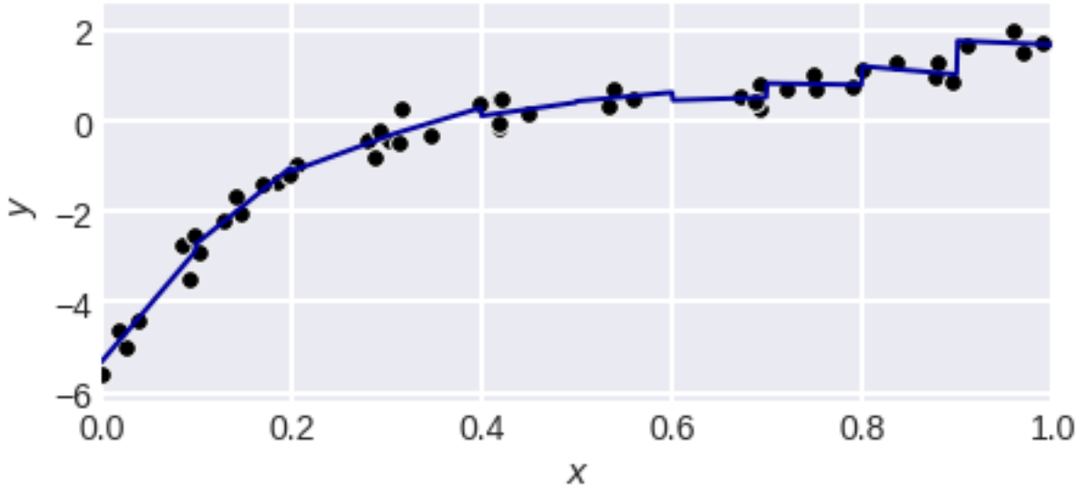
сравнение с порогом

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



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

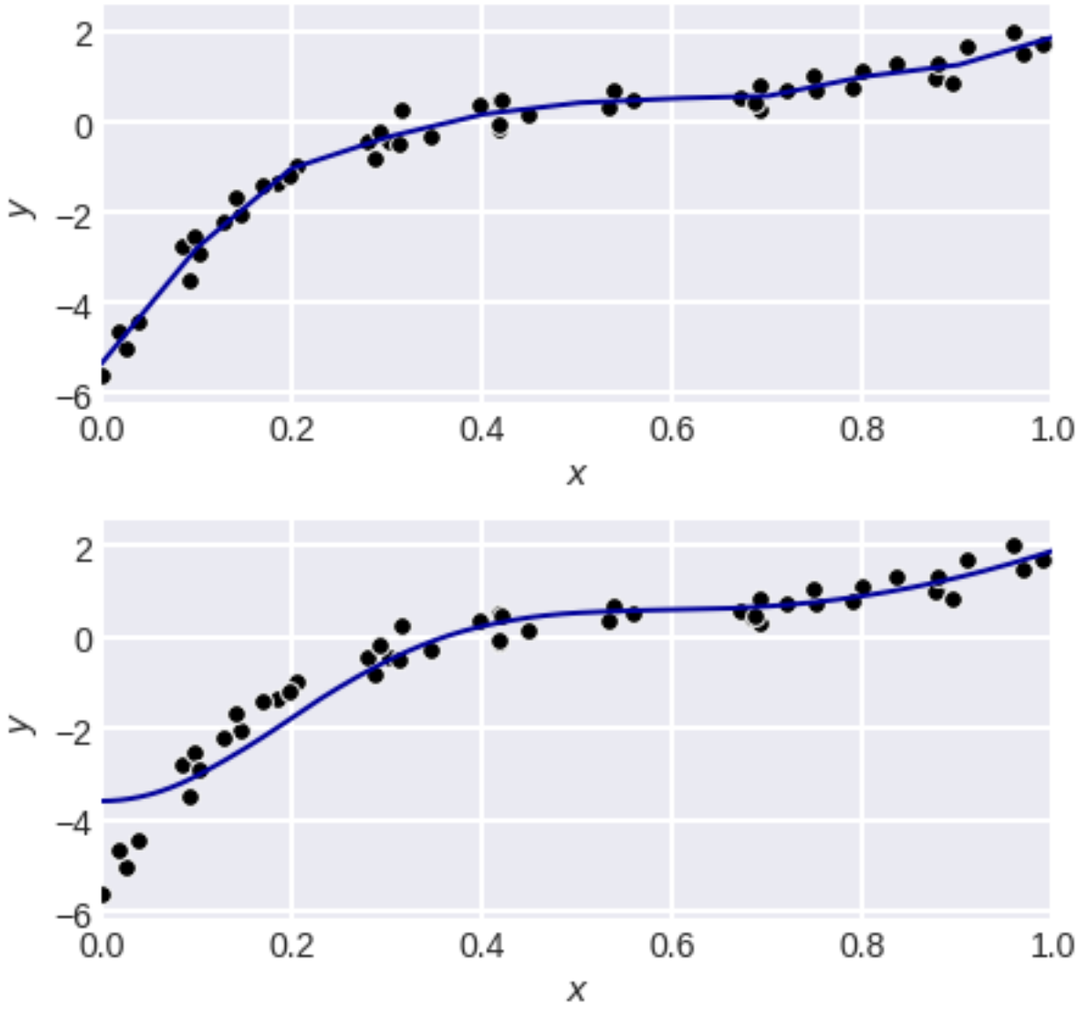
$$\begin{cases} |X_t - \theta_i|^k, & X_t \geq \theta_i, \\ 0, & X_t < \theta_i, \end{cases}$$

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

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

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 $\{x_1, \ldots, x_m\}$,
функции $\varphi_{x_1}(x), \ldots, \varphi_{x_m}(x)$ линейно независимы
и матрица $\| \varphi_{x_j}(x_i) \|_{m \times m}$ невырождена

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

Радиально-базисная функция (Radial basis function, RBF)

Gaussian

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

Multiquadric

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

Inverse quadratic

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

Thin plate spline

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

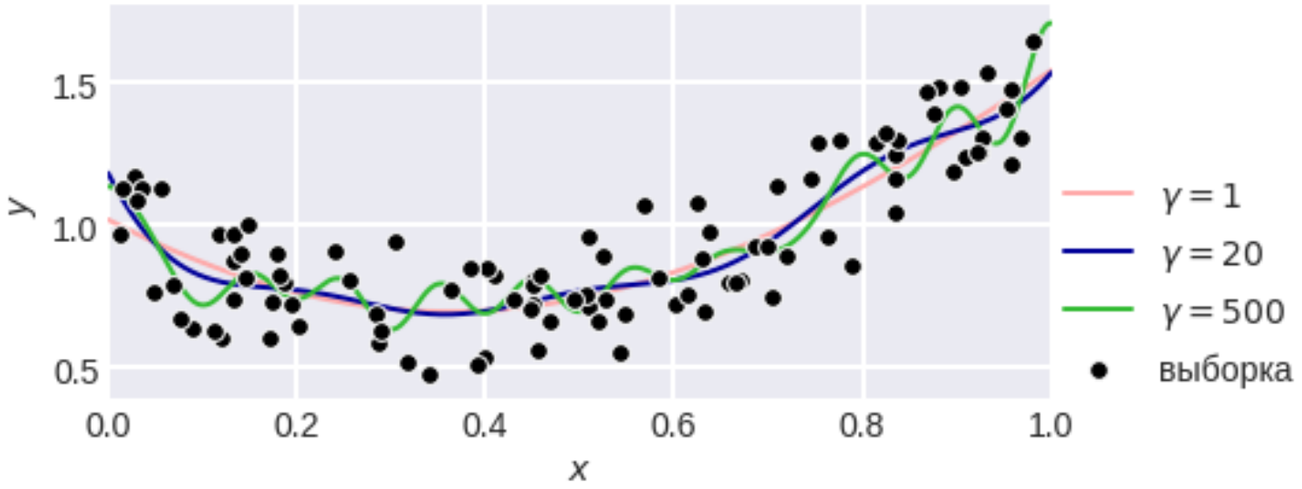
Inverse multiquadric

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

Polyharmonic spline

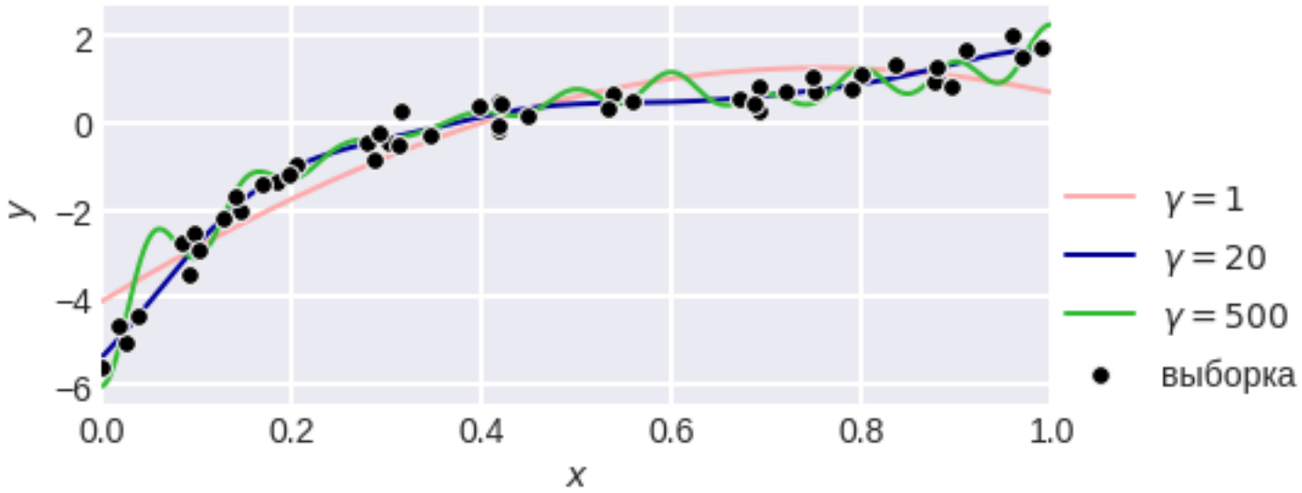
$$f(r) = \begin{cases} r^k, & k = 1,3,5,\dots \\ r^k \ln r, & k = 2,4,6,\dots \end{cases}$$

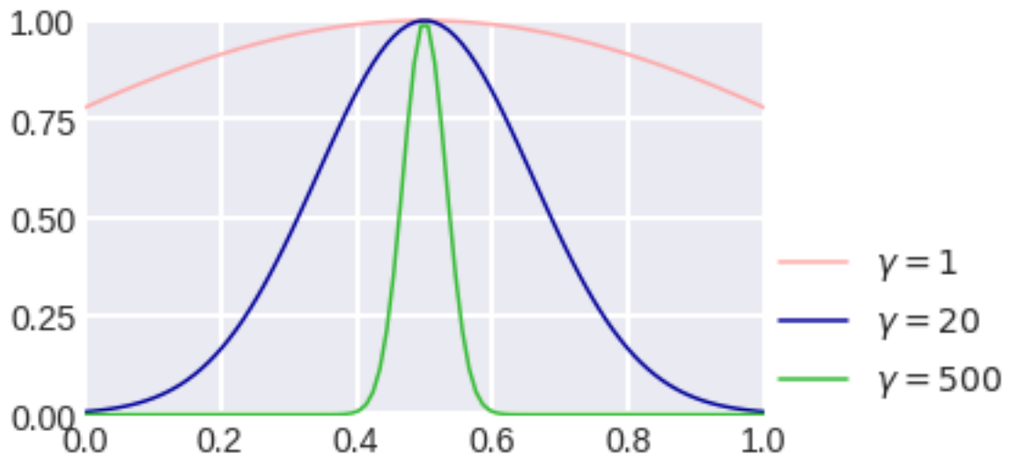
RBF-ядро: пример для регрессии



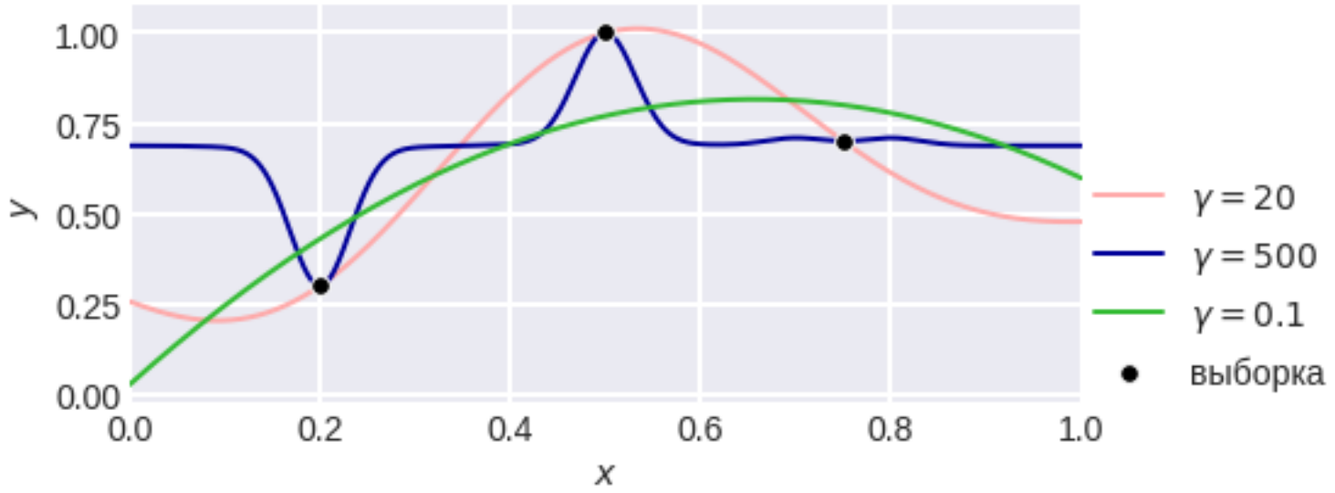
$$\varphi(x) = \exp(-\gamma \parallel x - z \parallel^2)$$

тут, правда, выбраны равномерные эталонные точки

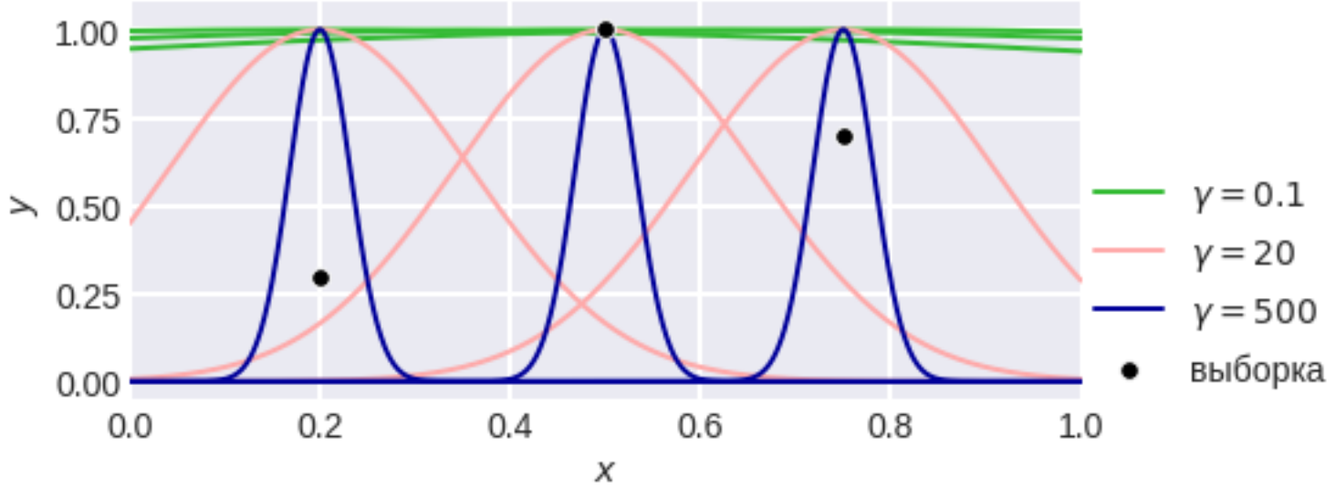




RBF-ядро: эффект от ширины ядра



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 ~ 1 / 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

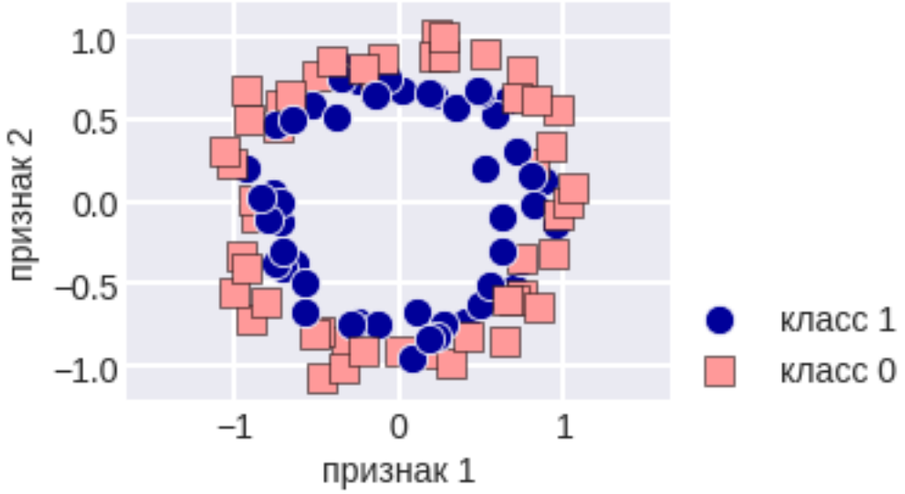
Ядерные методы (Kernel Tricks)

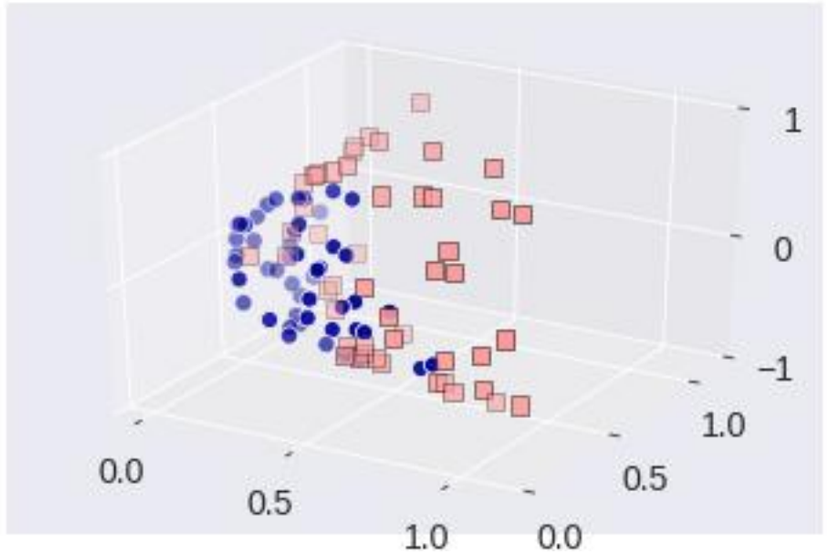
Идея «искривить/деформировать пространство» –
перейти в пространство признаков,
где уже можно решить линейными методами

Пример перехода $(x) \to (x, x^2)$

Ядерные методы (Kernel Tricks)

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





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

Пример: SVM

$$\sum_{i=1}^{m}\alpha_i - \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m}\alpha_i\alpha_j y_i y_j x_i^{\mathrm{T}} x_j \to \max_{0\le\alpha\le C}$$

$$\sum_{i=1}^{m}\alpha_i y_i = 0$$

от обучающей выборки нужны только попарные скалярные произведения!

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

Kernel Tricks

Пусть $x = (x_1, \ldots, x_n),\ z = (z_1, \ldots, z_n)$

рассмотрим функцию $K(x,z) = (x^{\mathrm{T}} z)^2$

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

где $\varphi(x) = (x_1^2, \ldots, x_n^2, \ldots, \sqrt{2} x_i x_j, \ldots),$

$$\varphi(z) = (z_1^2, \ldots, z_n^2, \ldots, \sqrt{2} z_i z_j, \ldots)$$

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 $K : X \times X \to \mathbb{R}$ **такая,**
что существует функция
$\varphi : X \to F$, **что** $K(x,z) = \varphi(x)^{\mathrm{T}} \varphi(z)$
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 <u>кернализированным (kernelized)</u>,
если для его реализации достаточно только знания значений скалярных произведений
(на признаковые описания объектов обучения)

Примеры ядер

Однородные полиномиальные (homogeneous polynomial kernel) степени d

$$K(x,z) = (x^{\mathrm{T}} z)^d$$

Неоднородные полиномиальные (inhomogeneous polynomial kernel) степени d

$$K(x,z) = (x^{\mathrm{T}} z + 1)^d$$

можно + const > 0

большие d ничем не сложнее $d = 1$

«Универсальное» полиномиальное ядро

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

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

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

$$\frac{1}{1-\alpha^2 x^{\mathrm{T}} z}=\sum_{i=0}^{+\infty}(\alpha^2 x^{\mathrm{T}} z)^i=1+\alpha^2 x^{\mathrm{T}} z+\alpha^4 (x^{\mathrm{T}} z)^2+\ldots=$$

$$=1+\alpha^2(x_1 z_1+\ldots+x_n z_n)+\alpha^4\left(\sum c_* x_i x_j z_i z_j\right)+\ldots=$$

$$=\varphi(x)^{\mathrm{T}}\varphi(z)$$

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

Пространство бесконечномерное,
а вычисляем за конечное время!

RBF-ядро соответствует переходу в многомерное пространство

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

$$K(x,z) = \exp(-(x-z)^2) =$$

$$= \exp(-x^2 + 2xz - z^2) =$$

$$= \exp(-x^2)\left(\sum_{k=0}^{\infty} \frac{2^k x^k z^k}{k!}\right)\exp(-z^2) =$$

$$= \left(\sum_{k=0}^{\infty} \frac{2^{k/2}\exp(-x^2)x^k}{\sqrt{k!}} \cdot \frac{2^{k/2}\exp(-z^2)z^k}{\sqrt{k!}}\right)$$

Пример использования в SVM

в классике…

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

теперь…

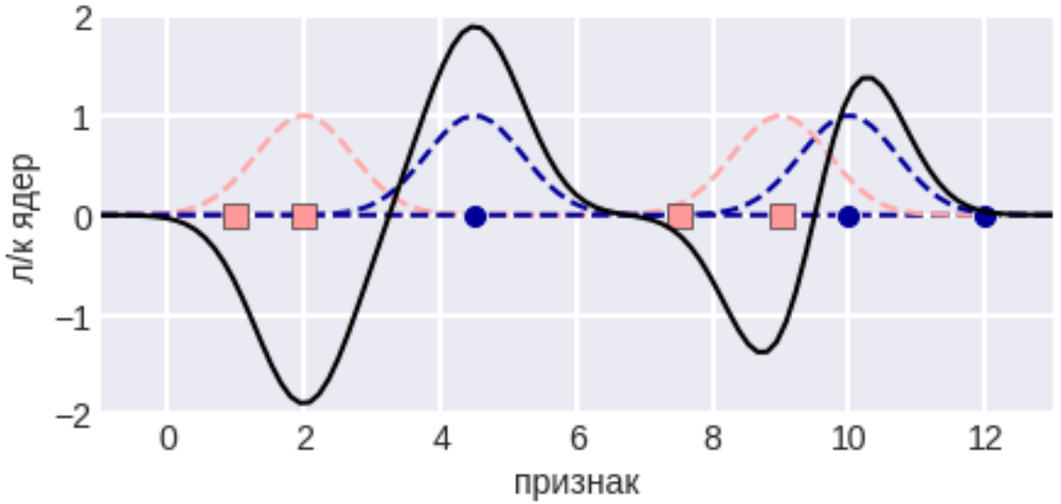
$$a(x) = \text{sgn}\left(\sum_{i \in S} \alpha_i y_i \varphi(x_i)^{\text{T}} \varphi(x) \right) =$$

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

надо знать опорные векторы для классификации

S **– множество их индексов**

Теперь получили нелинейную модель с помощью линейной!!!



кстати, если использовать смещение

$$b = \frac{1}{|S|} \sum_{i \in S} \left(y_i - \sum_{j \in S} \alpha_j y_j x_j^{\text{T}} x_i \right) \to$$

$$\frac{1}{|S|} \sum_{i \in S} \left(y_i - \sum_{j \in S} \alpha_j y_j K(x_j, x_i) \right)$$

Нелинейный метод SVM

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

Решить

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

при условиях

$$0 \le \alpha \le C,\ y^{\mathrm{T}} \alpha = 0$$

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]

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

+ при настройке единственное решение

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

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

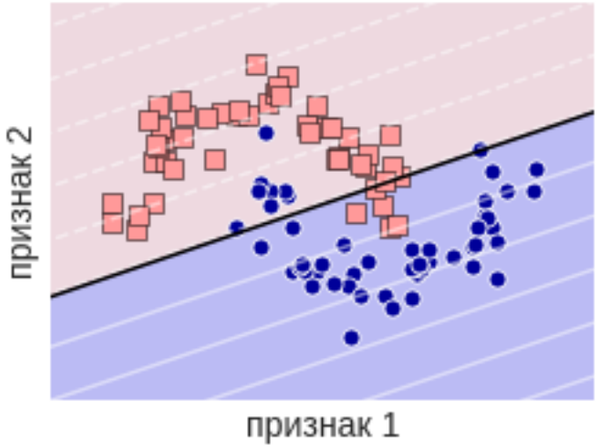
– использование ядер может вызывать переполнения…

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

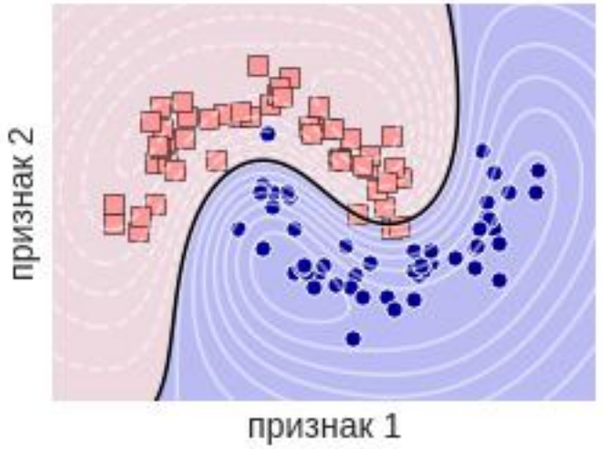
**– геометрия в новом пространстве нам, на самом деле,
не совсем интуитивно ясна**

SVM с разными ядрами

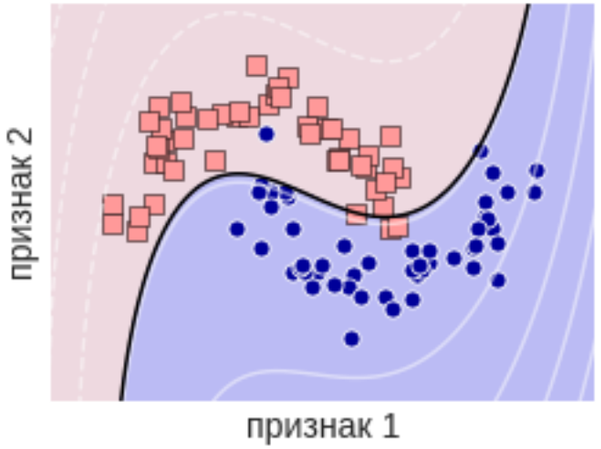
linear



RBF



poly



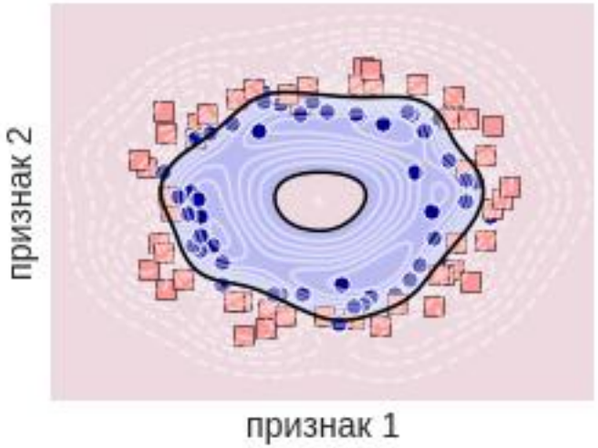
Минутка кода

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

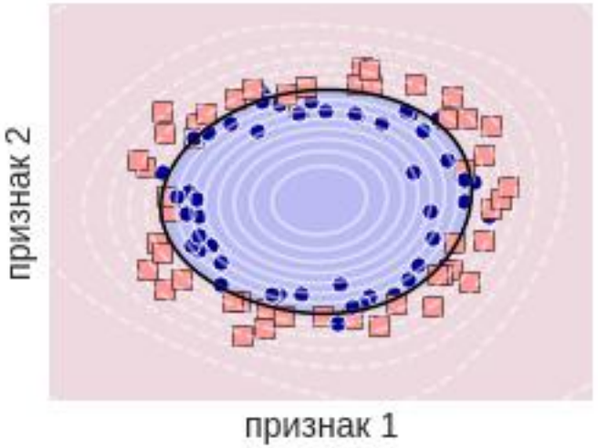
Ниже обратите внимание на нелогичности разделений и возможность «обмана алгоритма»

SVM с RBF-ядрами разной ширины

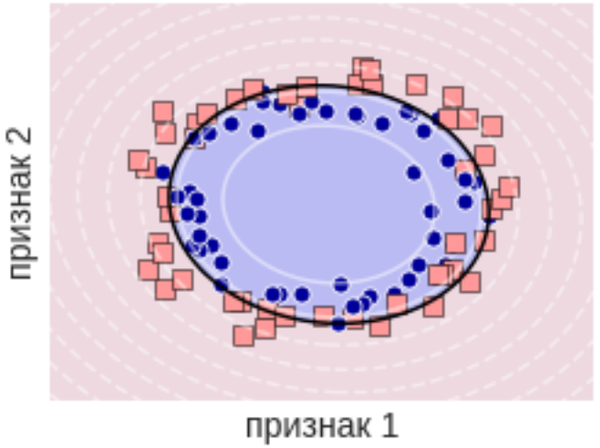
$\gamma = 10$

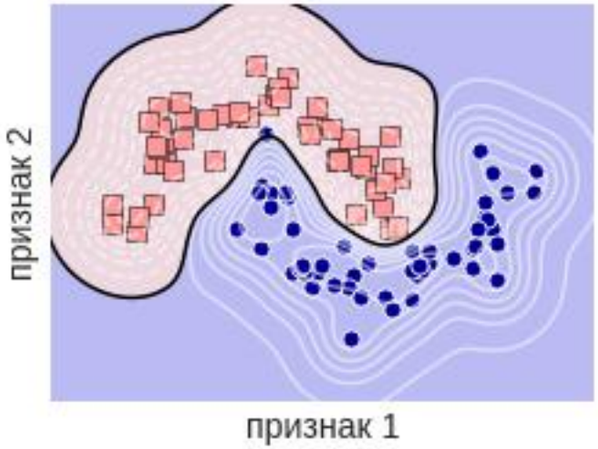


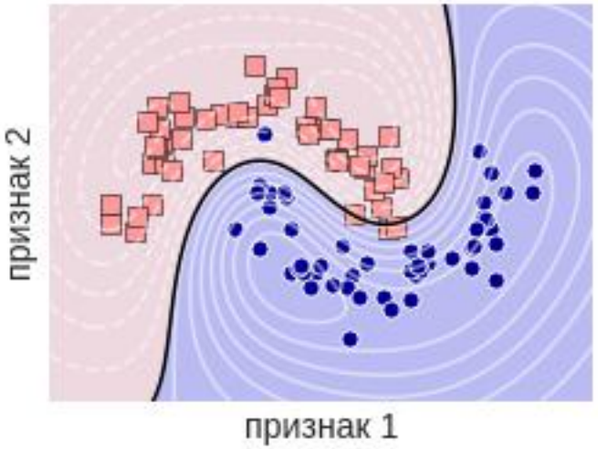
$\gamma = 1$

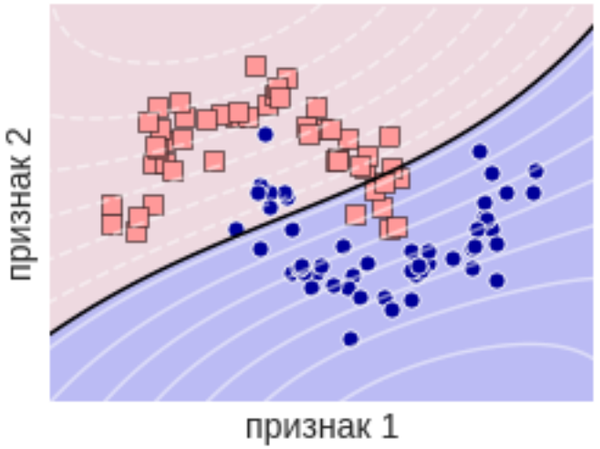


$\gamma = 0.1$



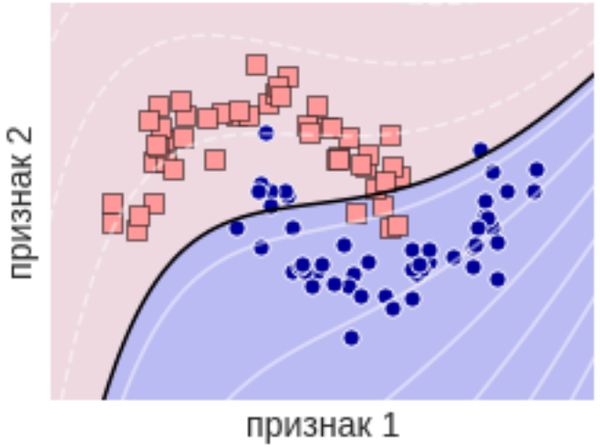




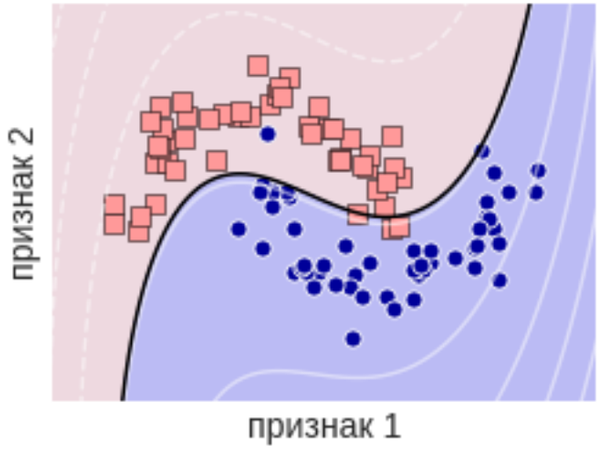


SVM с регуляризацией

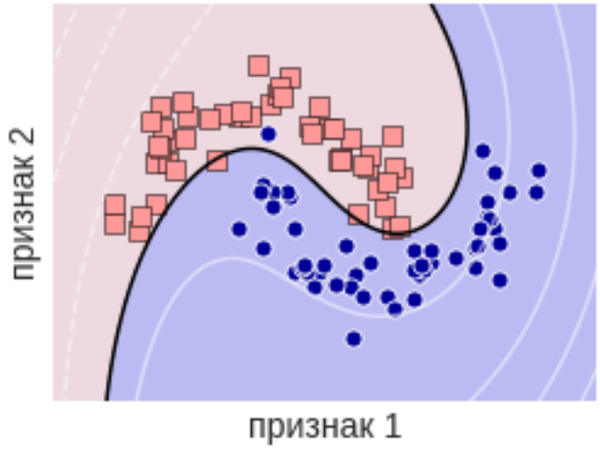
$C = 0.1$

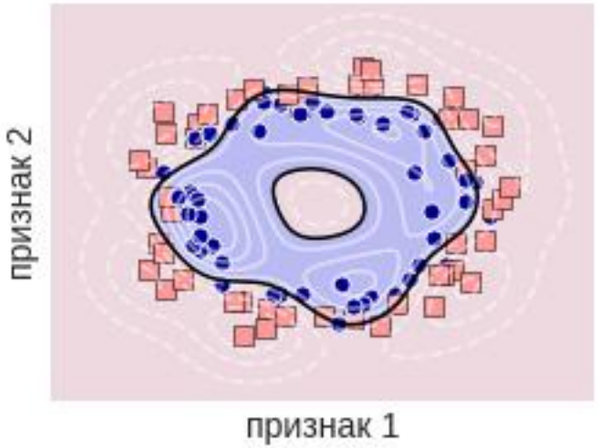


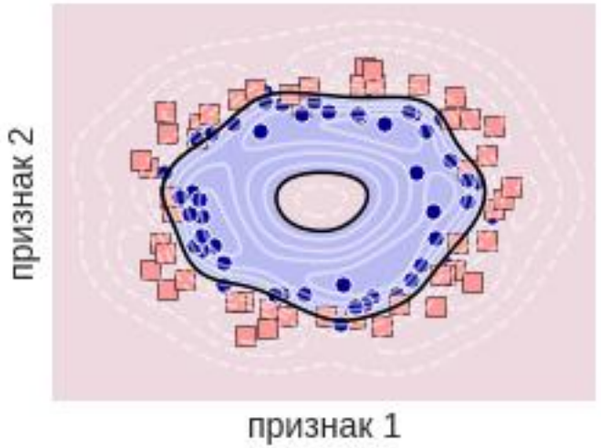
$C = 1$

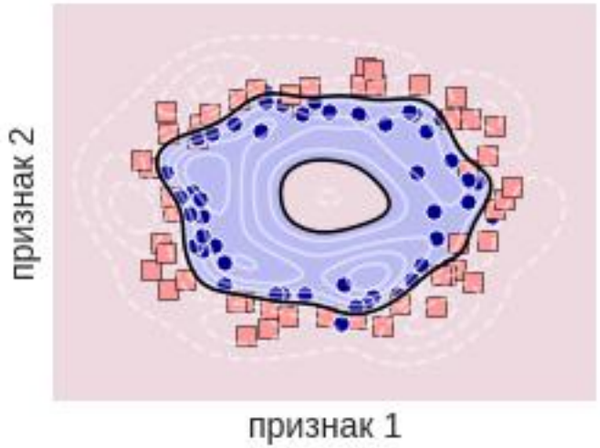


$C = 10$



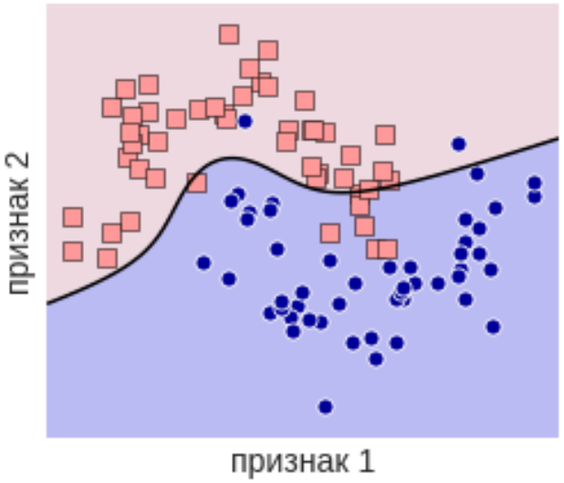




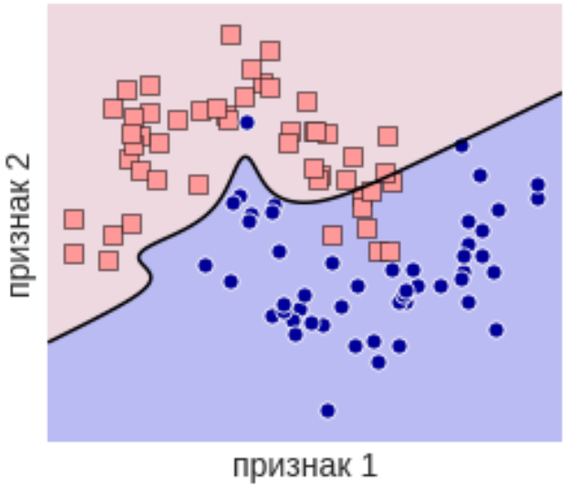


SVM с разной степенью полинома

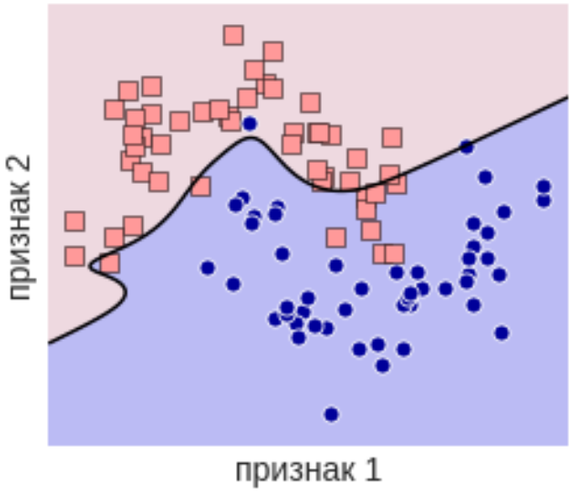
$d = 3$

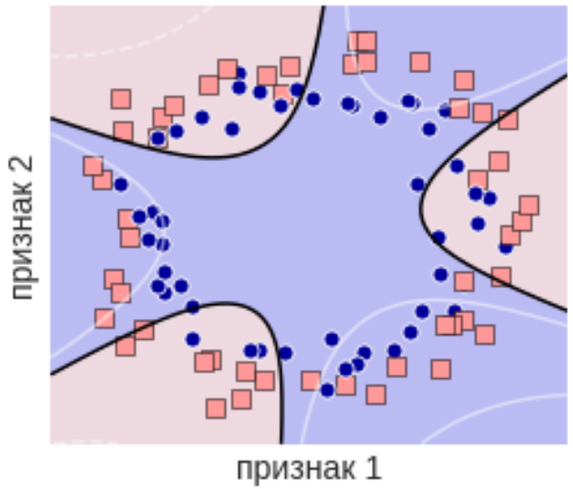


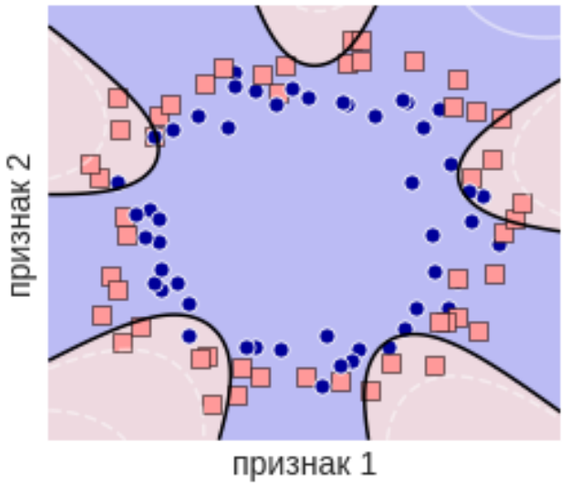
$d = 5$

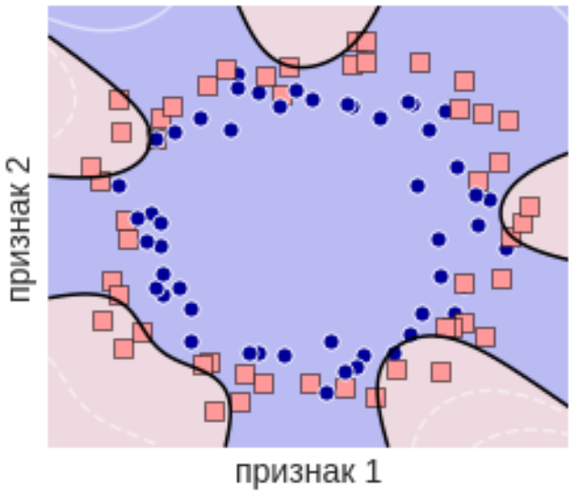


$d = 7$





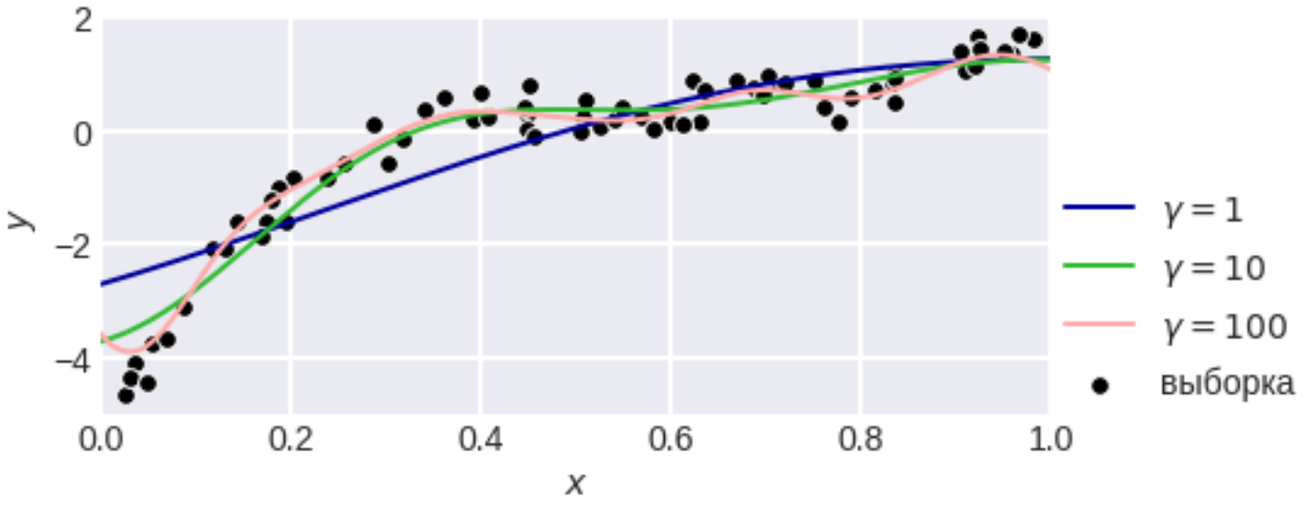




Кернализация гребневой регрессии: просто код

`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**



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
```

Проблема выбора ядра

Выбор ядра – экспертно / перебор

Настройка гиперпараметров ядра – скользящий контроль

Интерпретация ядра:

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

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

Но почему именно с этими объектами???

$\Rightarrow$ настройка (определение) объектов

Не забывать

- **храним много информации**

например, $m \times m$-матрицу

есть методы увеличения скорости kernel-методов

RBF – суть метода

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

Помним… базисность (в RBF) нужна для невырожденности этой матрицы

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

– система m уравнений с m неизвестными

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

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

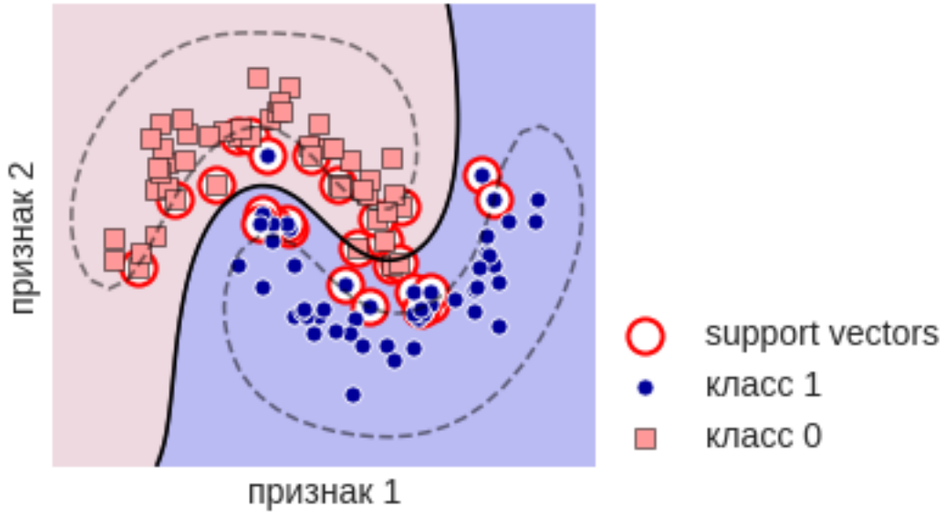
помним, что аналогично было в SVM… $a(x) = \text{sgn}\left(\sum_{i \in S} \alpha_i y_i K(x_i, x) \right)$

RBF – проблема выбора эталонных точек

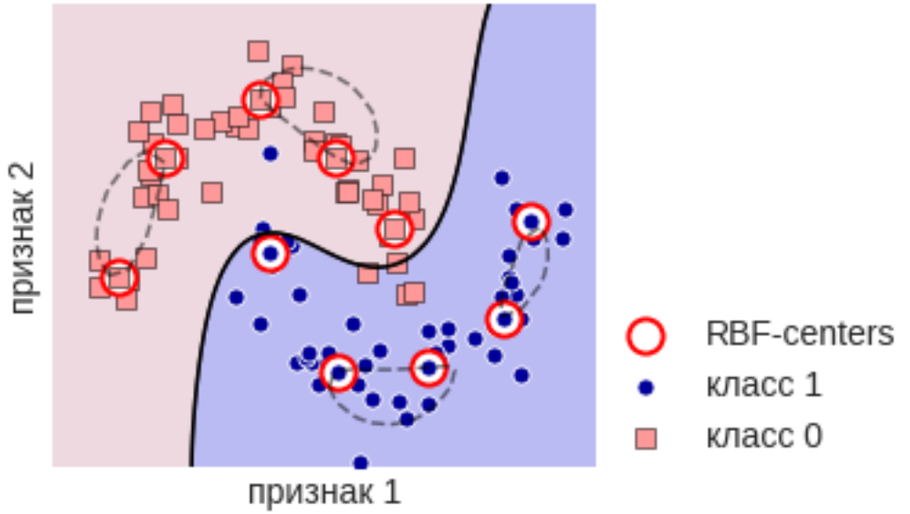
лучше в качестве центров использовать k эталонных точек
Как выбрать?

один из способов: кластеризация → центры кластеров

Support vectors



RBF centers



RBF – проблема настройки весов

если точек будет меньше… (собственно, как в SVM)

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

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

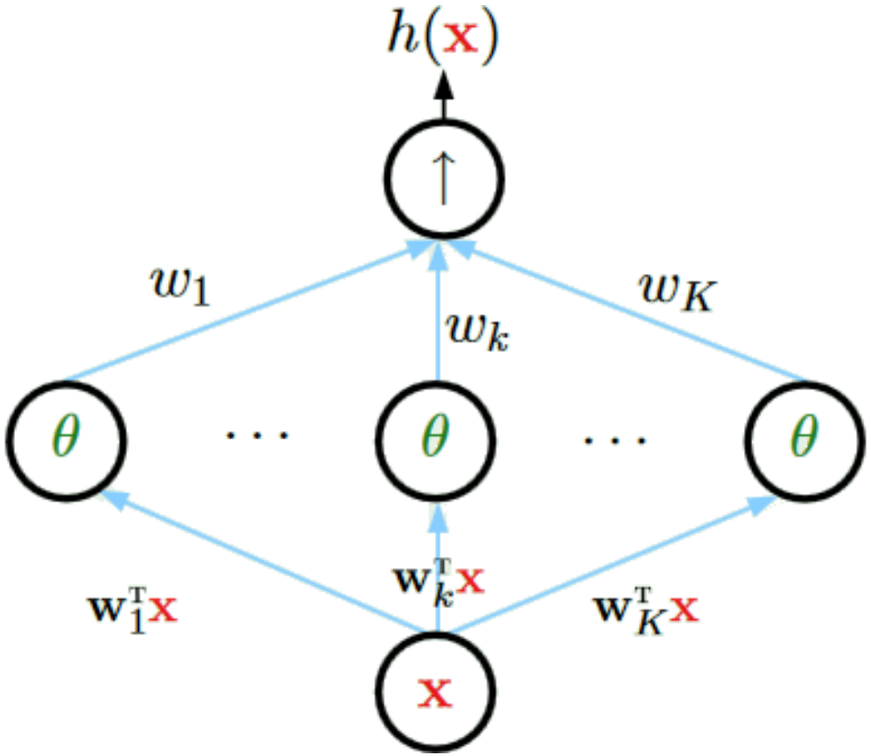
решение через псевдообратную (если матрицу обозначить через K):

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

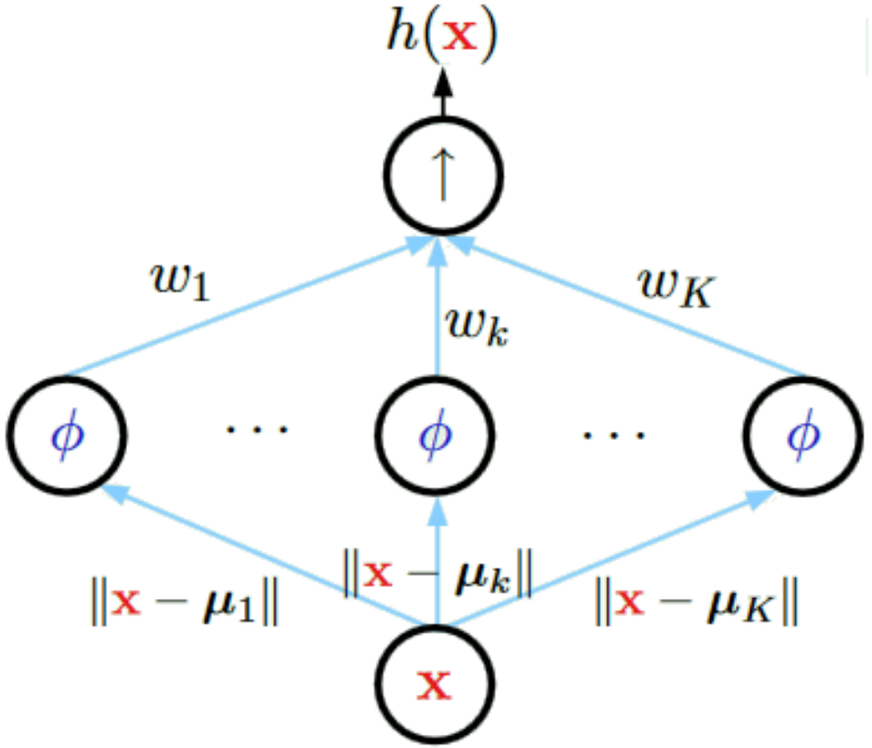
а практичнее – SGD

или RBF-сеть

RBF-сеть (Radial basis function network)



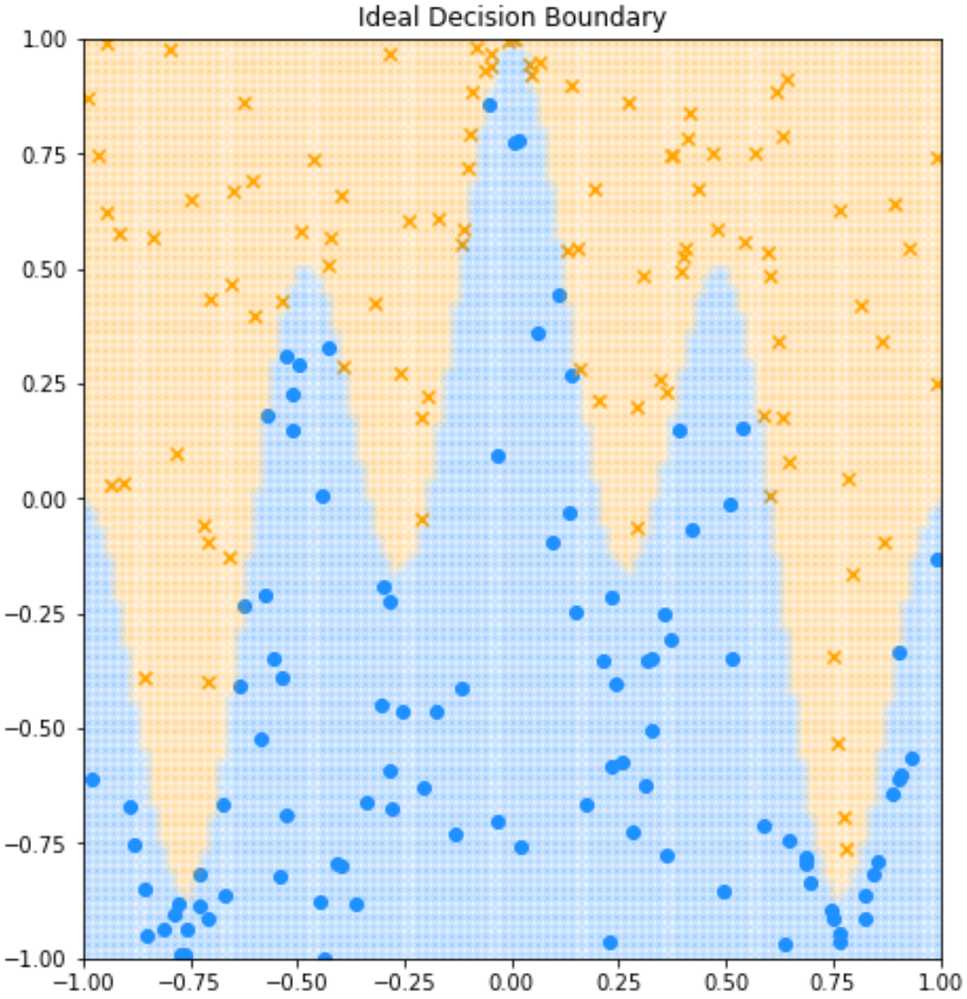
нейросеть

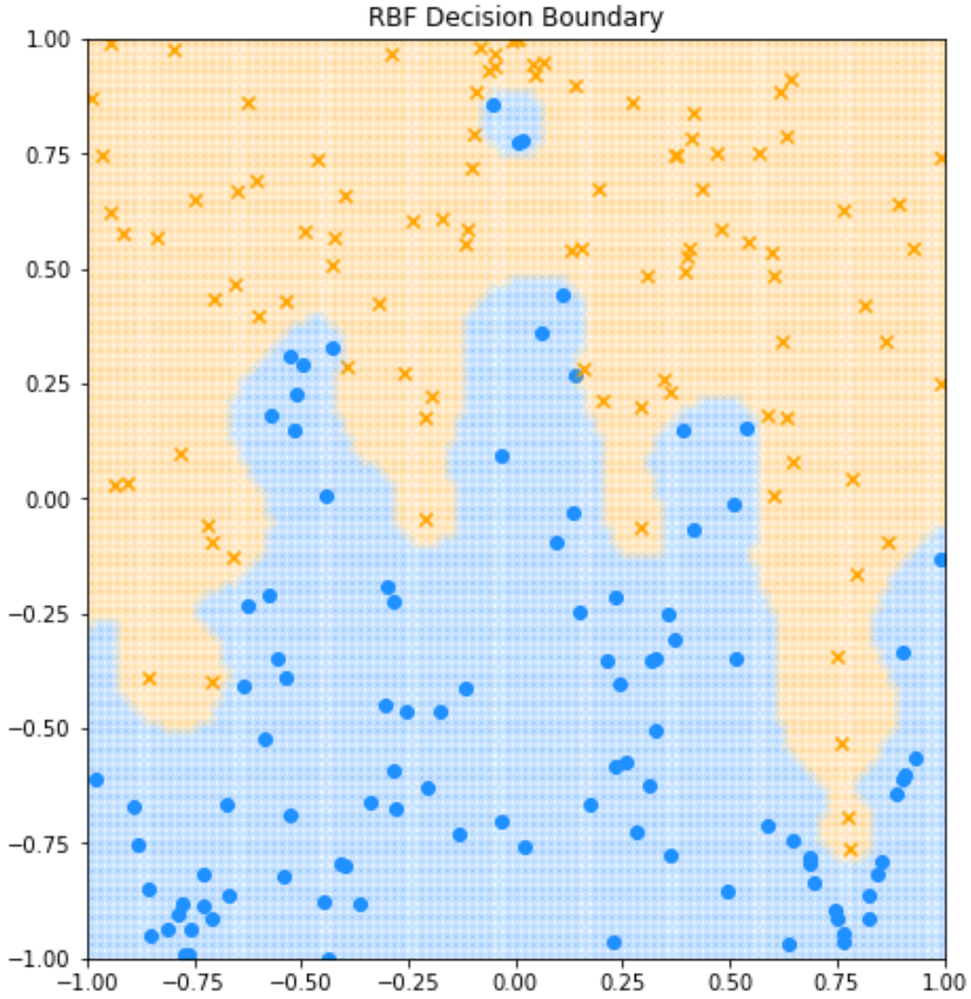


RBF-сеть

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

$S \subseteq \{12,\ldots,m\}$

+ SGD

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

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 + precond [3, 28, 4, 5, 6]	n^2	n^2	n	n
Divide & Conquer [29]	n^2	$n\sqrt{n}$	n	n
Nyström, random features [7, 8, 9]	n^2	$n\sqrt{n}$	n	$\sqrt{n}$
Nyström + iterative [23, 24]	n^2	$n\sqrt{n}$	n	$\sqrt{n}$
Nyström + sgd [20]	n^2	$n\sqrt{n}$	n	$\sqrt{n}$
FALKON (see Thm. 3)	$\boldsymbol{n\sqrt{n}}$	$\boldsymbol{n\sqrt{n}}$	$\boldsymbol{n}$	$\boldsymbol{\sqrt{n}}$

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

Реализации методов, основанных на ядрах

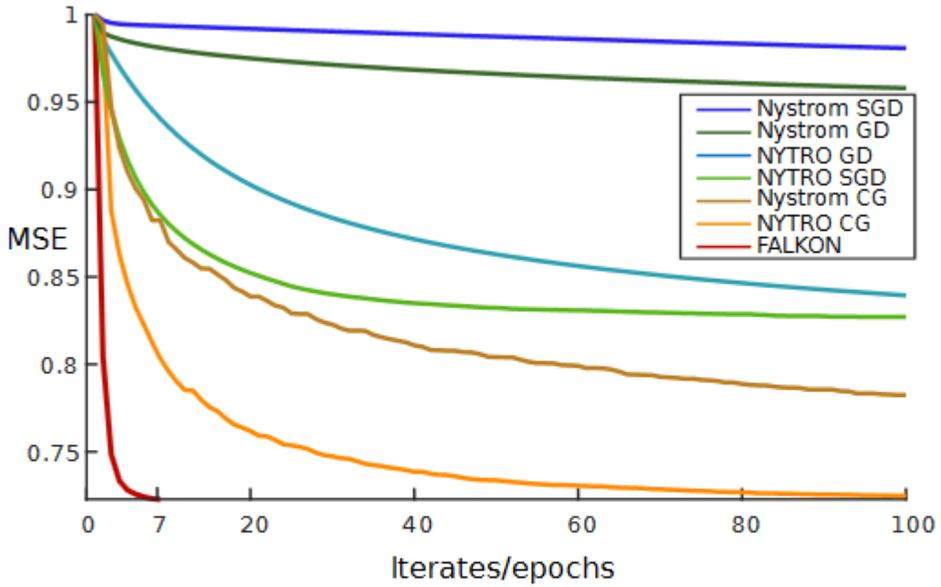


Figure 1: Falkon is compared to stochastic gradient, gradient descent and conjugate gradient applied to Problem (8), while NYTRO refer to the variants described in [23]. The graph shows the test error on the HIGGS dataset (1.1×10^7 examples) with respect to the number of iterations (epochs for stochastic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

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

Пример деформации пространства на практике: (m×k)*(k×n)

Дано: m, n, k, характеристики ЭВМ

Целевой признак: Время перемножения матриц размеров m×k и k×n

В контроле: ~5000 Записей, ~950 признаков

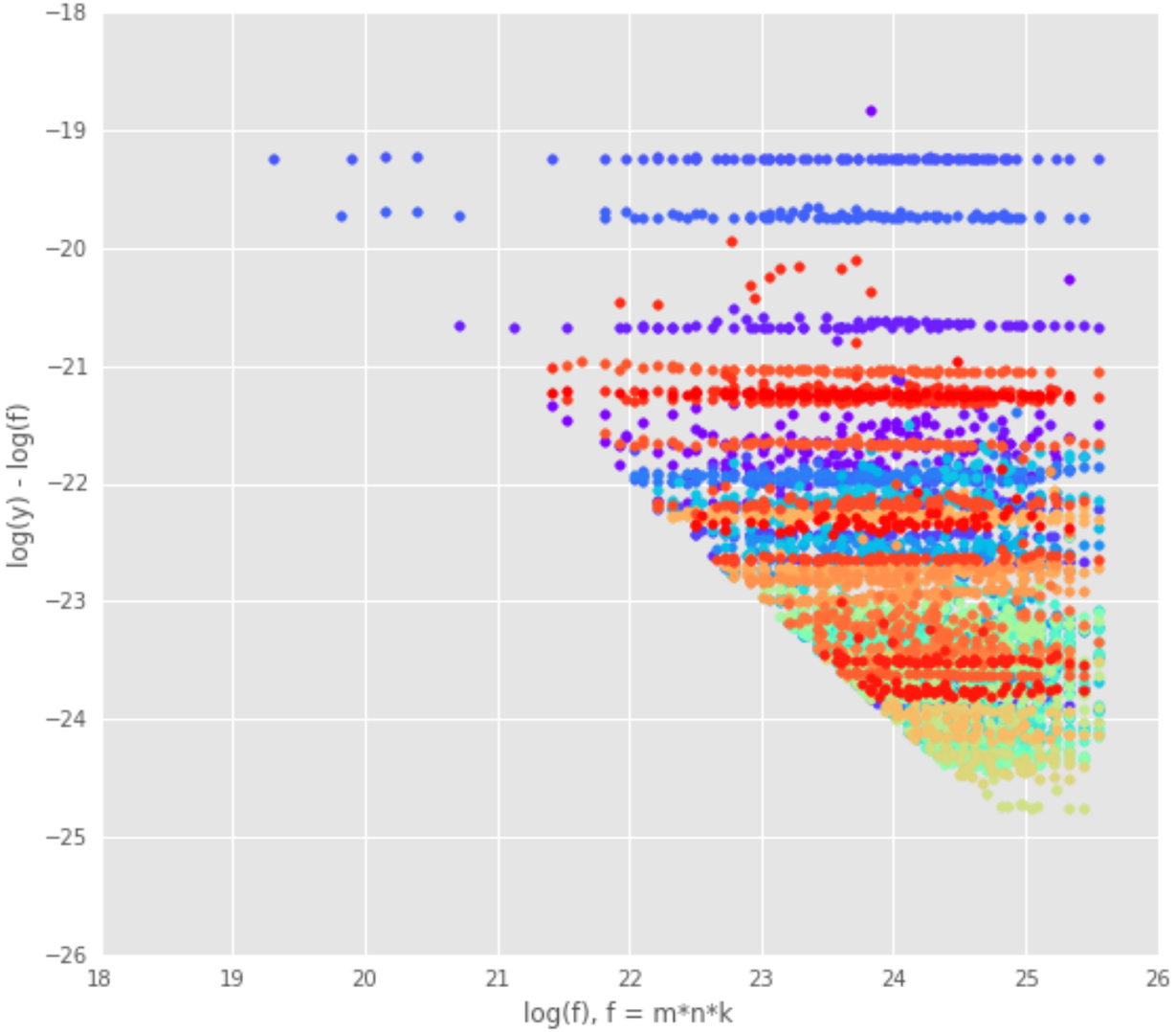
В тесте: =*=

Функционал качества: MAPE

Прагматика: прогнозирование времени вычислений

Простая логика: результат произведения – mn элементов, вычисление каждого – k умножений, общее число умножений = m×n×k

Пример деформации пространства на практике: (m×k)*(k×n)



Регрессия:

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

Пусть

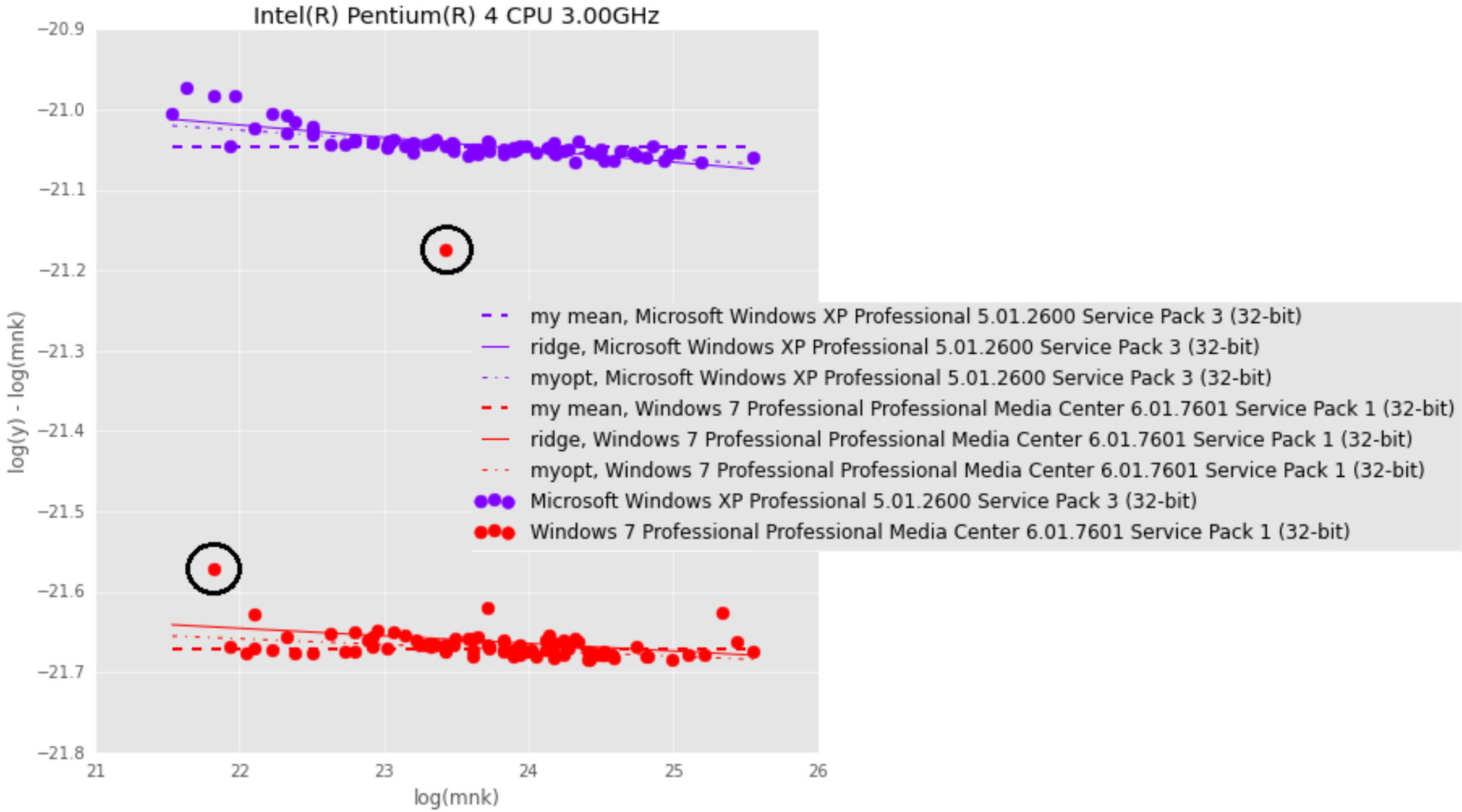
$$\log(y) - \log(mnk) = w_1 \log(mnk) + w_0$$

потом покажем, почему не const

Ответ задачи:

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

Пример деформации пространства на практике: (m×k)*(k×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