«Машинное обучение и анализ данных»

Нелинейные методы

Александр Дьяконов

16 ноября 2020 года

План

Что делать, если хотим искать нелинейные закономерности линейными методами

«Трюки с ядрами (kernel tricks)»

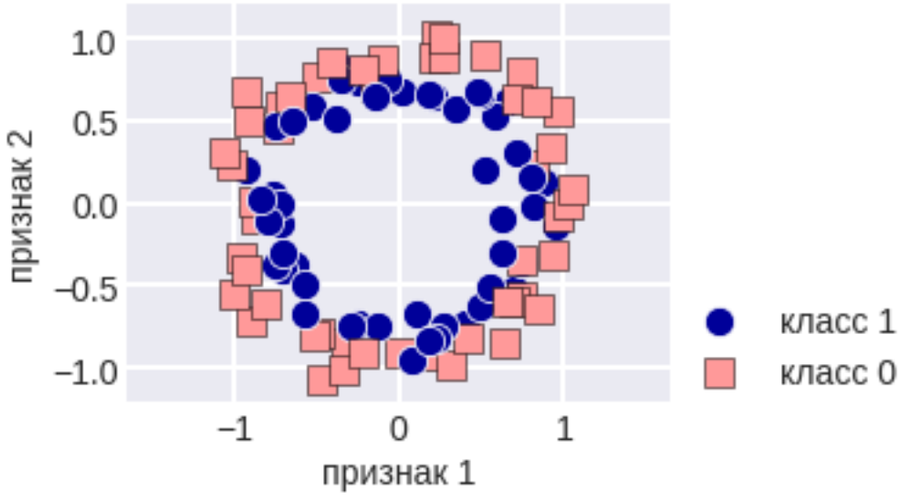
Вывод нелинейного SVM

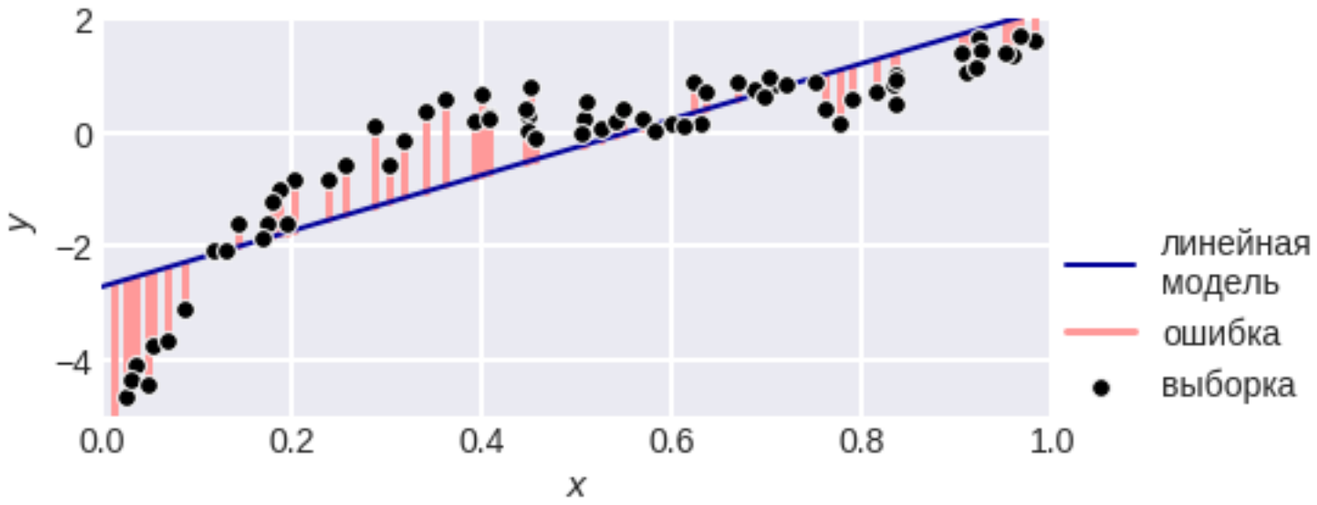
«Кернализация» других методов

Решение задач произвольной природы

Проблема линейности

Некоторые задачи не решаются линейными методами





Когда линейной модели не хватает

- **добавление «деформаций признаков»**

- **добавление любых других «базисных функций»**
GAM (Generalized Additive Models)

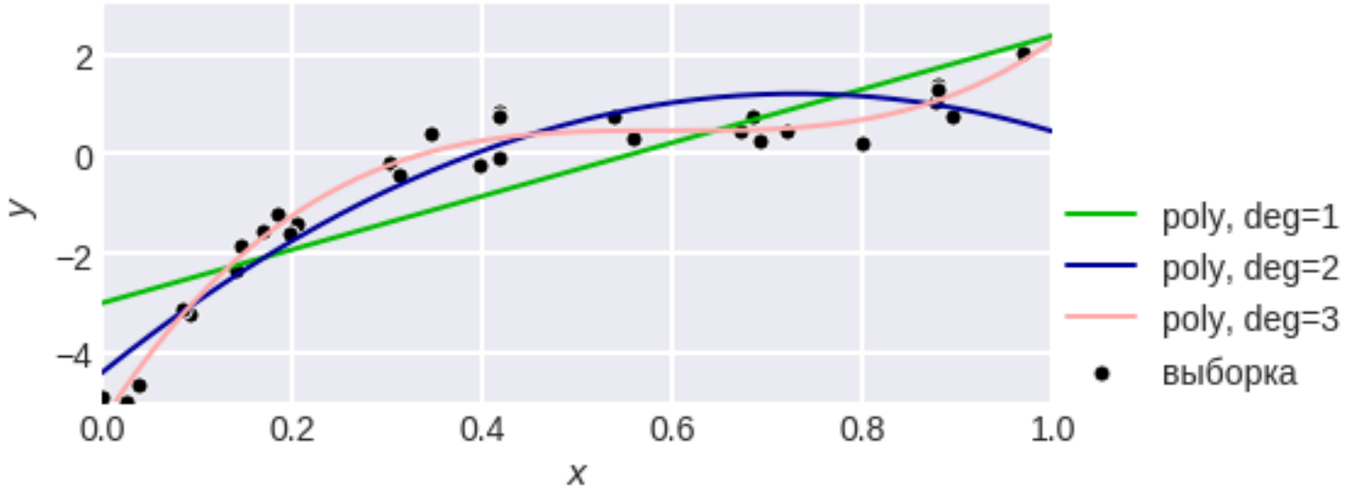
можно совмещать подходы: кусочно-полиномиальная модель

- **сплайны**

- **локально линейные методы**
Local Regression

- **ядерные методы (Kernel Tricks)**

Деформация: полиномиальная модель

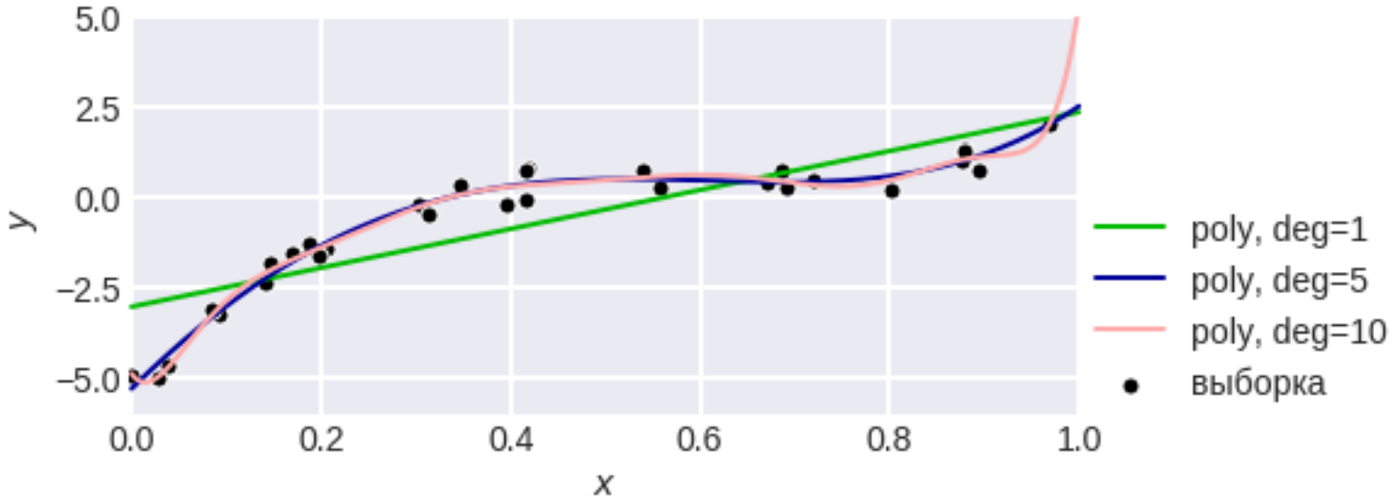


добавляем признаки-мономы

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

с несколькими $(z_1, \ldots, z_n) \to (\ldots, \prod_{t \in T} z_t, \ldots)$

Деформация: полиномиальная модель



подводные камни:

большое число признаков (+ неестественные признаки)
приводит к переобучению

потом в материале «сложность»

Минутка кода: полиномиальная модель

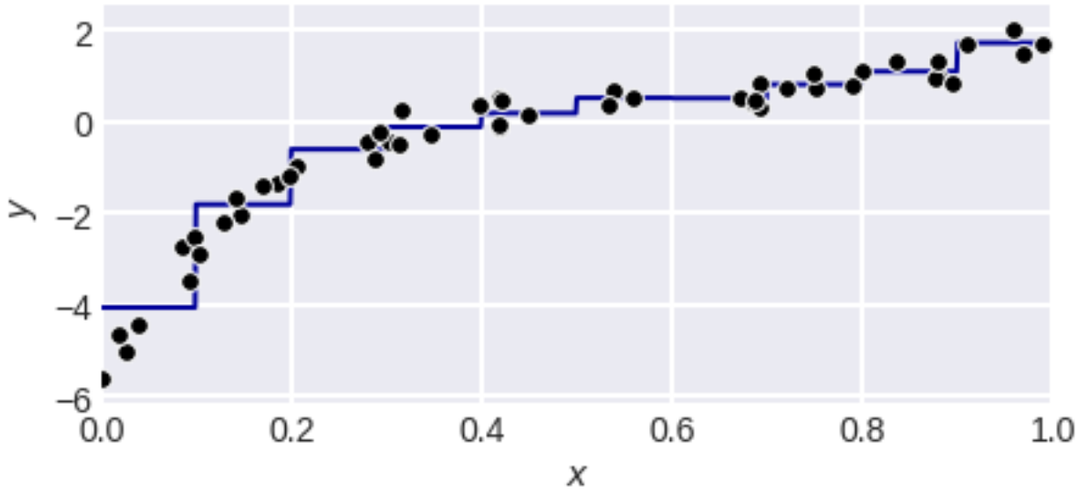
```
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import Ridge

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!

Использование других базисных функций

$$\begin{cases} 1, & x \geq \theta_i, \\ 0, & x < \theta_i, \end{cases}$$



интересно, что можно просто составить новую признаковую матрицу и «запихнуть» её в функцию регрессии

часто: характеристические функции интервалов (Step Functions)

$$I[\theta_i \leq x < \theta_{i+1}]$$

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

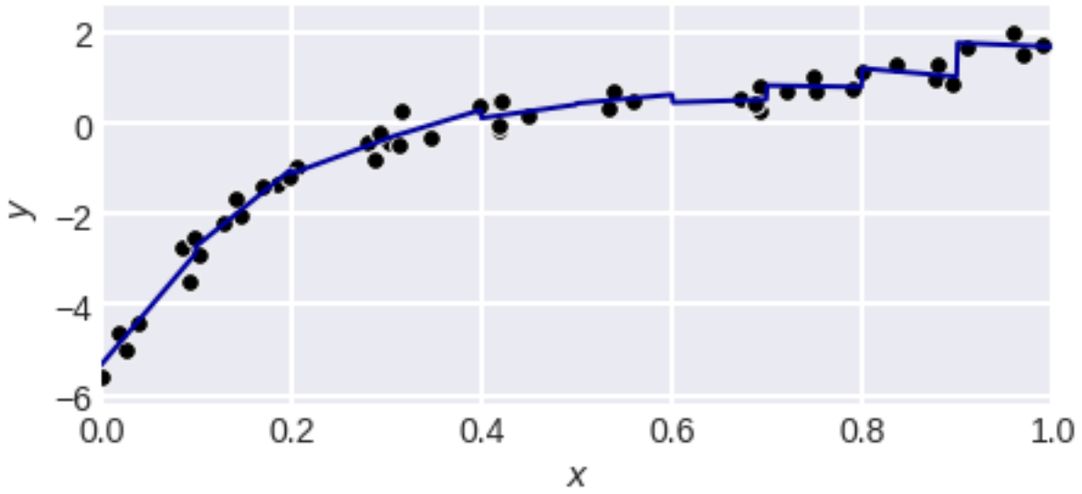
- **квантили**
- **особые точки**

(например, круглые суммы в признаке «зарплата»)

Использование других базисных функций

Кусочно-линейная регрессия

$$I[\theta_i \le x < \theta_{i+1}],$$
$$x \cdot I[\theta_i \le x < \theta_{i+1}]$$



аналогично кусочно-полиномиальная

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

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

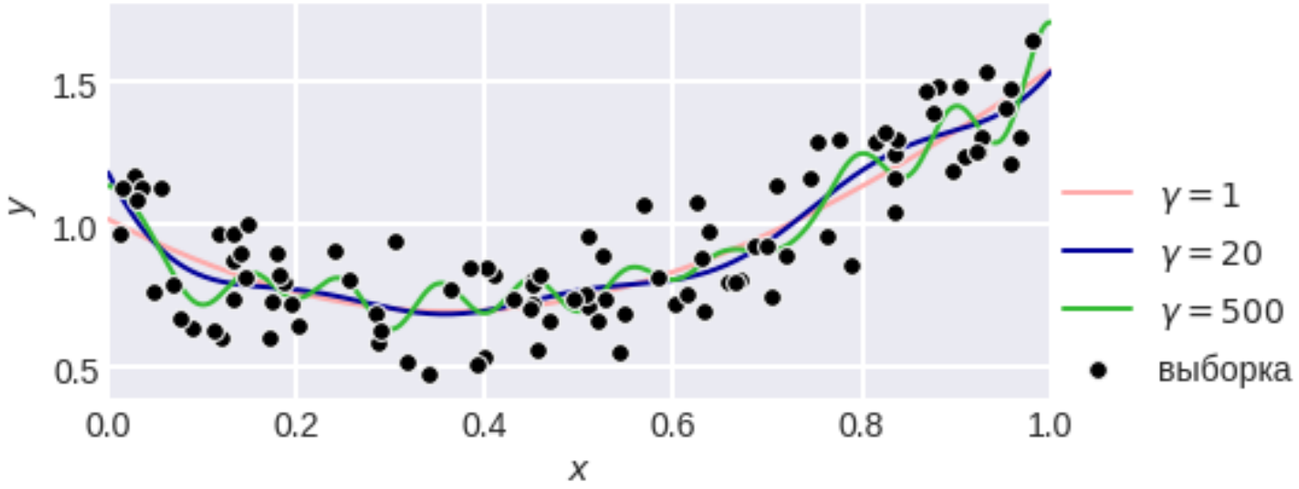
Inverse multiquadric

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

Polyharmonic spline

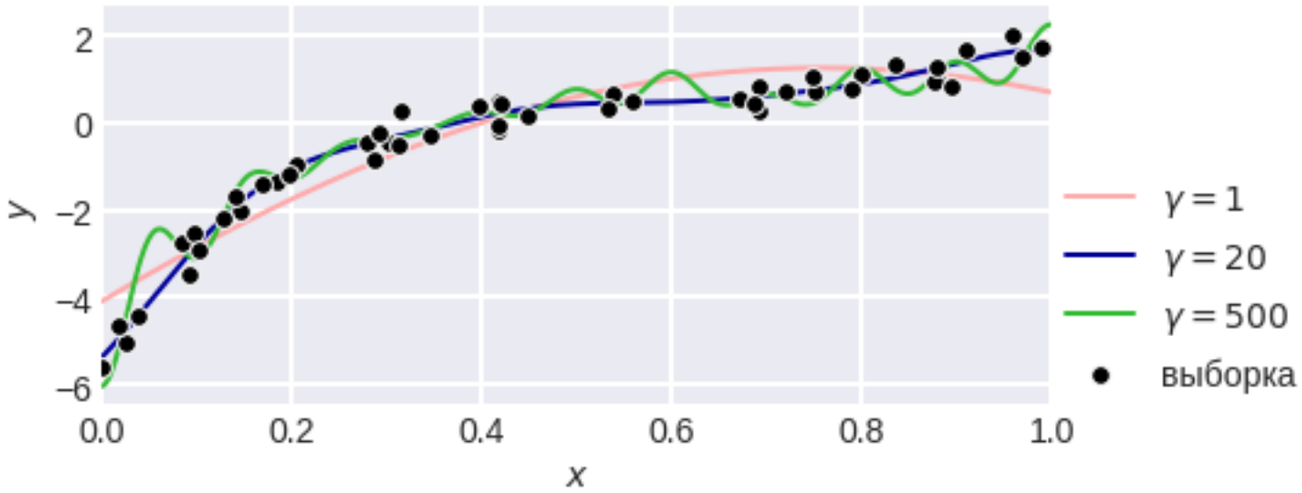
$$f(r) = \begin{cases} r^k, & k = 1,3,5,\dots \\ r^k \ln r, & k = 2,4,6,\dots \end{cases}$$

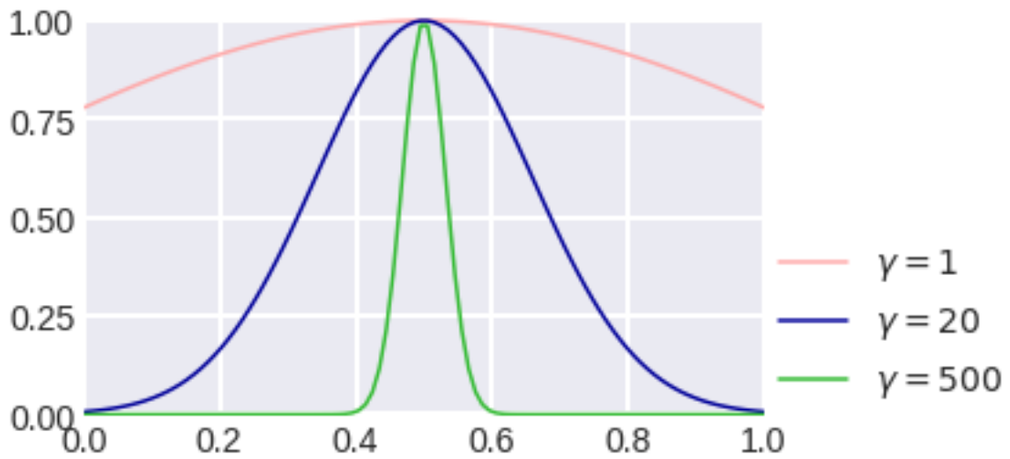
RBF-ядро: пример для регрессии



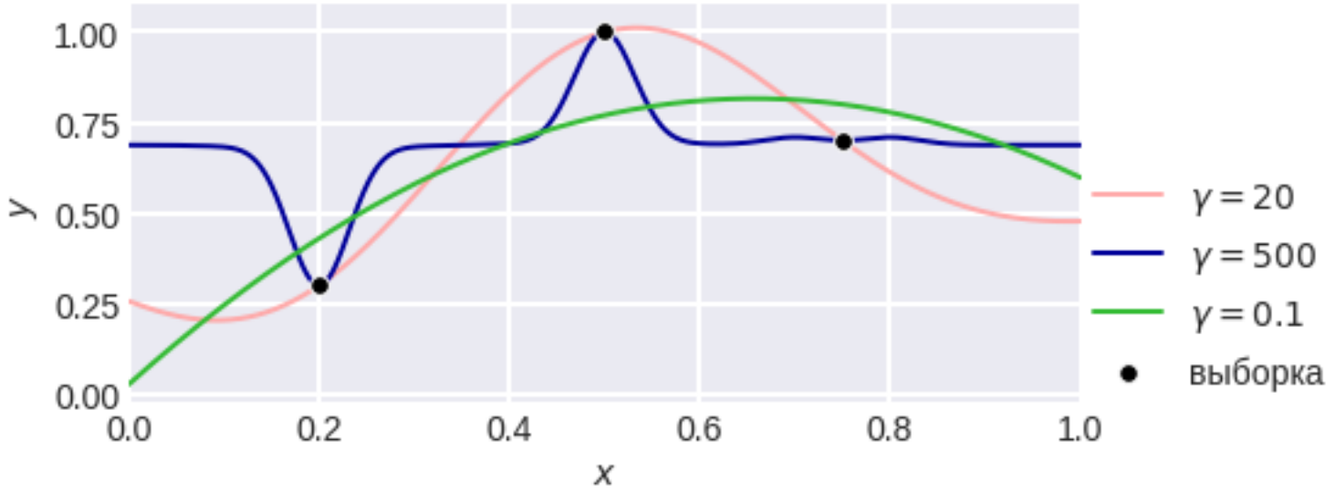
$$\varphi(x) = \exp(-\gamma \parallel x - z \parallel^2)$$

тут, правда, выбраны равномерные эталонные точки

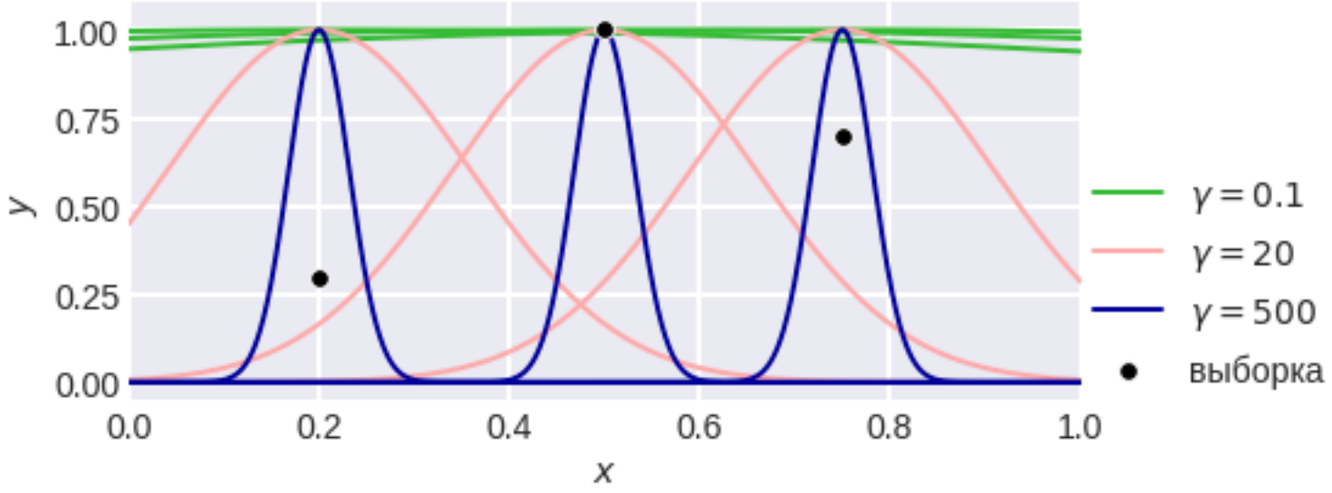




RBF-ядро: эффект от ширины ядра



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)

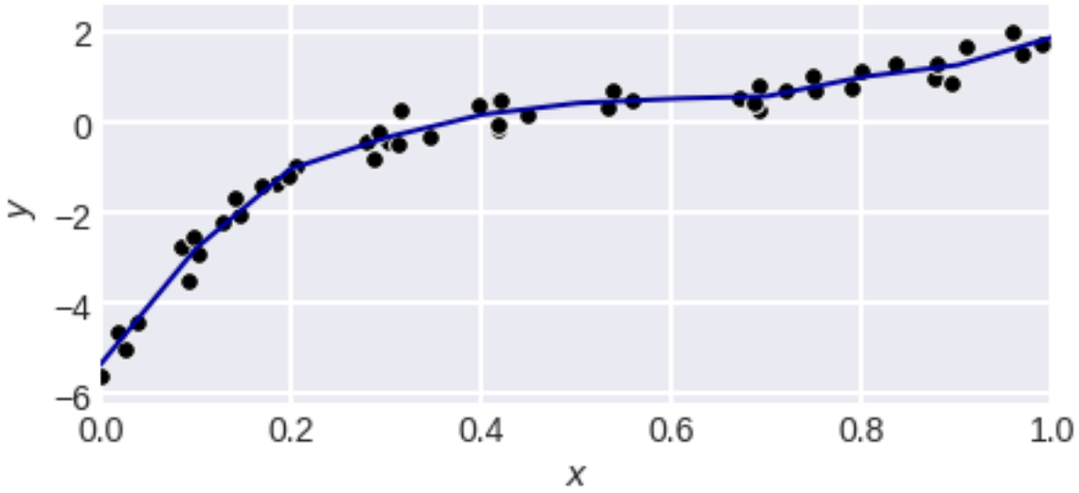
https://towardsdatascience.com/loess-373d43b03564

вернёмся в предобработке данных

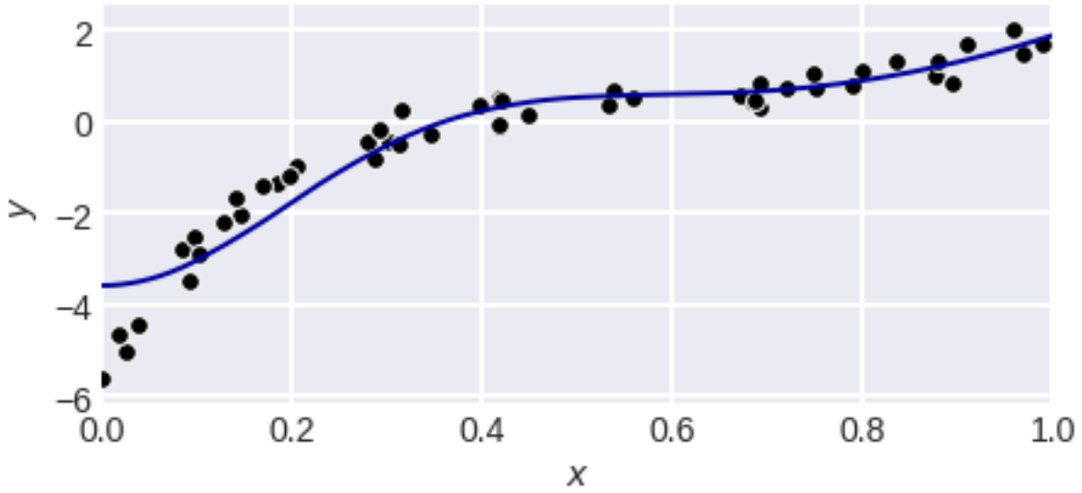
Сплайны

можно добавляя функции вида

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



$$\begin{cases} |\, x-\theta_i \,|, & x \ge \theta_i, \\ 0, & x < \theta_i, \end{cases}$$



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

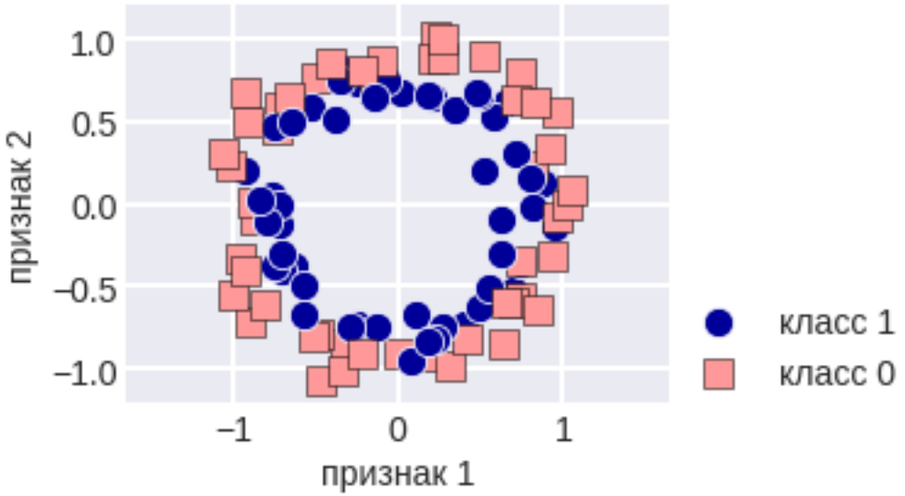
Ядерные методы (Kernel Tricks)

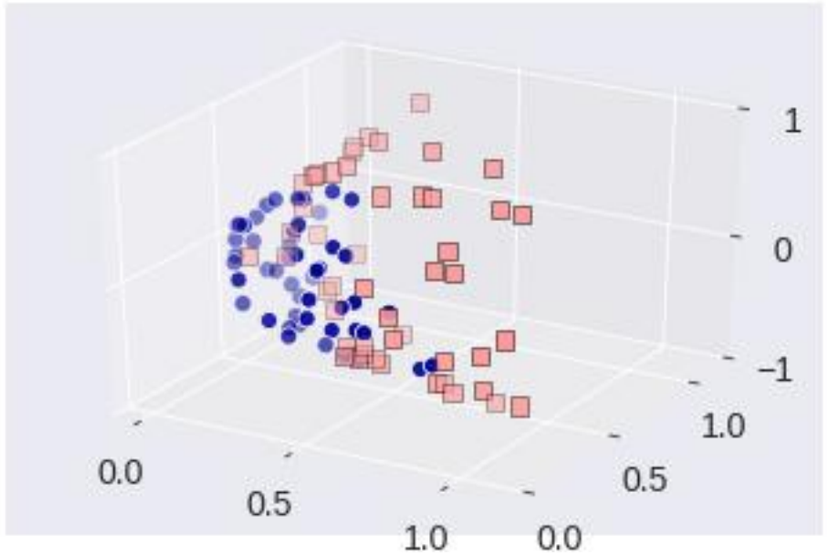
Идея «искривить/деформировать пространство» –
перейти в пространство признаков,
где уже можно решить линейными методами

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

нетрудно видеть

$$K(x,z) = (x_1 z_1 + \ldots + x_n z_n)^2 =$$

$$= x_1^2 z_1^2 + \ldots + x_n^2 z_n^2 + \sum_{ij} \sqrt{2} x_i x_j \sqrt{2} z_i z_j =$$

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

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

Чтобы перейти в пространство всех мономов степени 2 не надо явно строить признаки. Достаточно возвести в квадрат скалярное произведение в изначальном признаковом пространстве.

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 кернализированным (kernelized), если информации из обучения достаточно в таком виде

Ядро (Kernel) – определение формальное

функция $K : X \times X \to \mathbb{R}$ **– неотрицательно определённое ядро**
(positive semidefinite kernel),
если это симметричная функция:

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

для любой обучающей выборки $\{x_1,\ldots,x_m\}$ **матрица ядра неотрицательно определена:**

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

(~ 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

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

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

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

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

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

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

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

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…

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

теперь…

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

Надо знать опорные векторы для классификации

S **– множество индексов опорных векторов**

Теперь получили нелинейную модель с помощью линейной!!!

- **Есть специальные ядра**
- **Можно учить ядро по данным!**

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

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

Наблюдение

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

Запись напоминает простейшую нейронную сеть [Воронцов]

+ при настройке единственное решение

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

– непонятно, как выбрать ядро, параметры метода, признаки

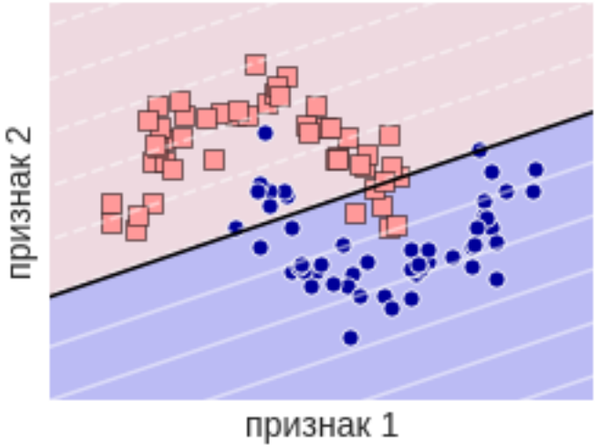
(это не автоматизировано)

Подводные камни

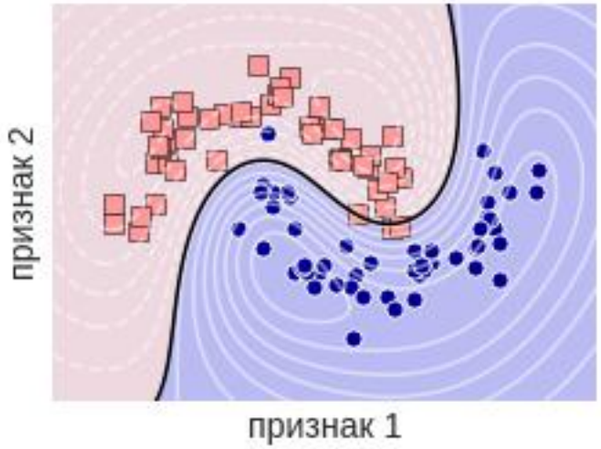
- использование ядер может вызывать переполнения…
- если мы сами придумаем пространство, в которое хотим перейти, то, скорее всего, не найдём подходящего ядра
- геометрия в новом пространстве нам, на самом деле, не совсем интуитивно ясна

SVM с разными ядрами

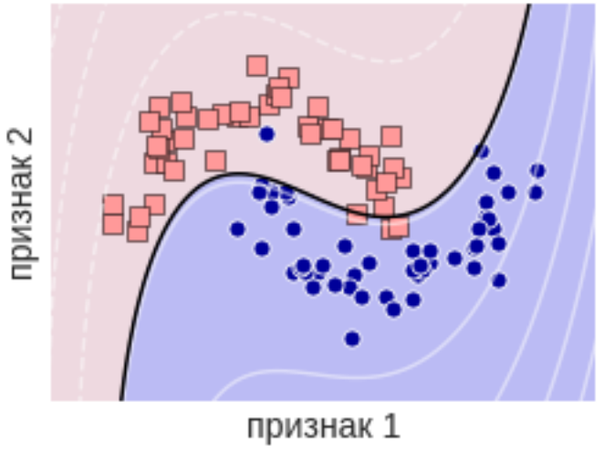
linear



RBF



poly

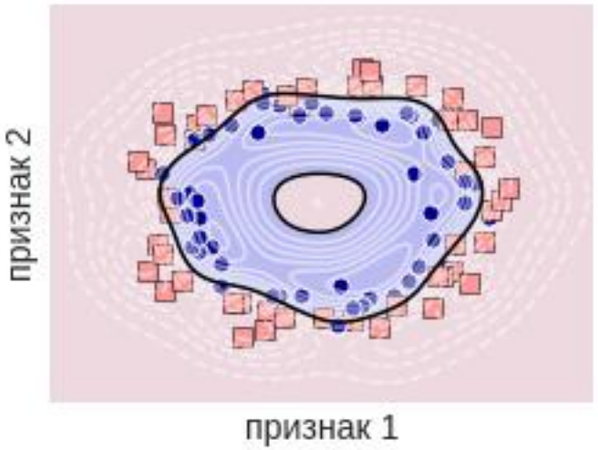


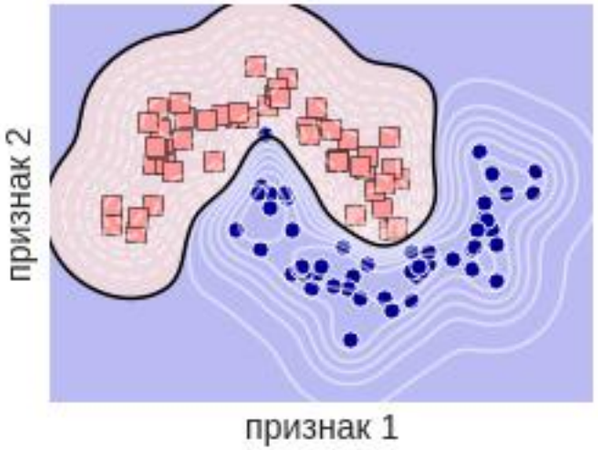
Минутка кода

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

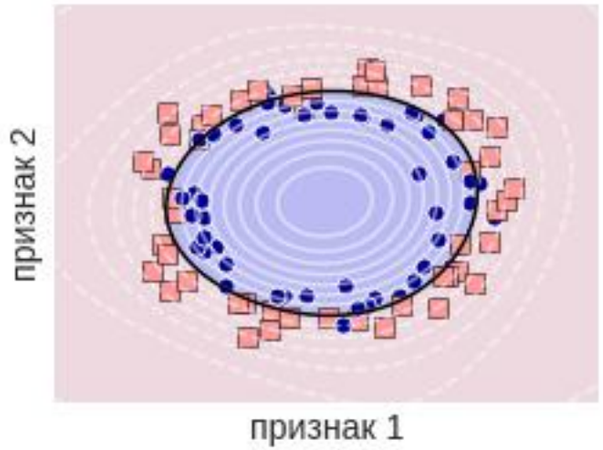
SVM с RBF-ядрами разной ширины

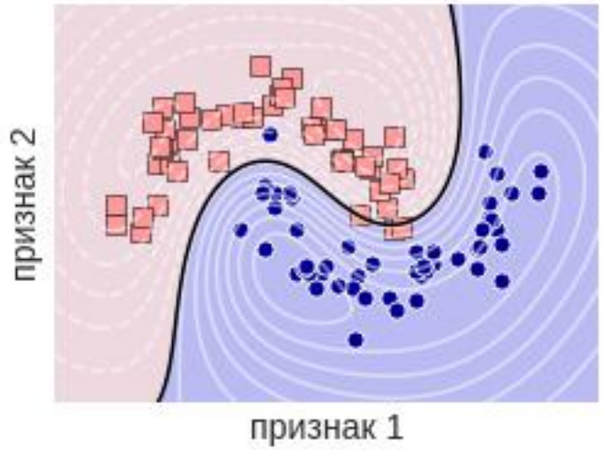
$\gamma = 10$



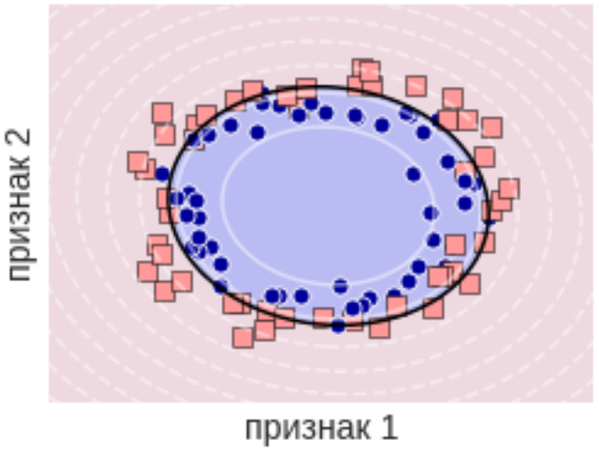


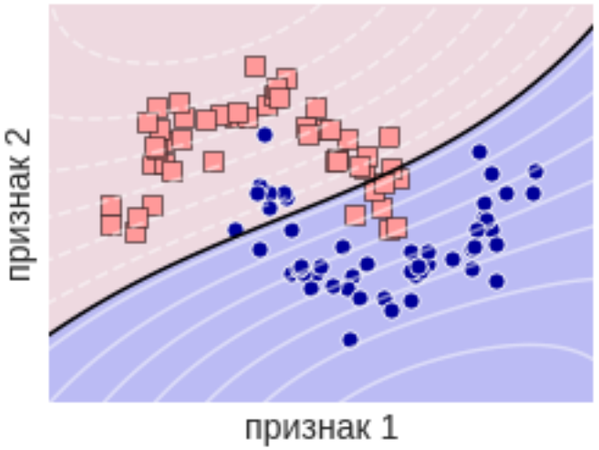
$\gamma = 1$





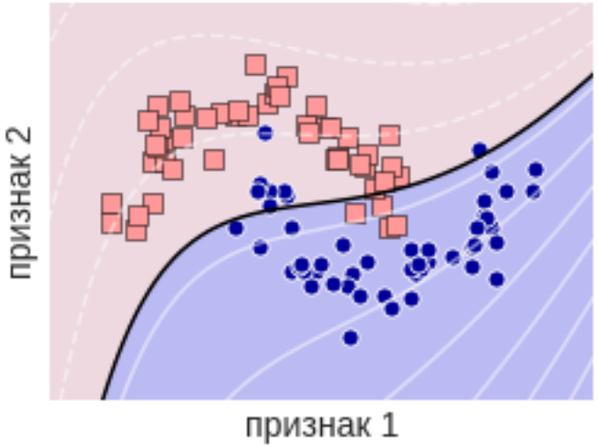
$\gamma = 0.1$



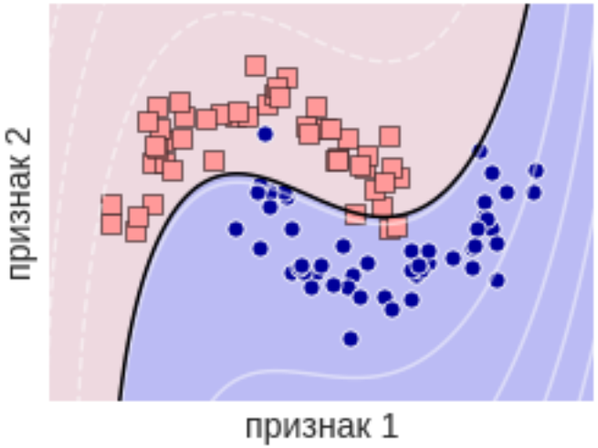


SVM с регуляризацией

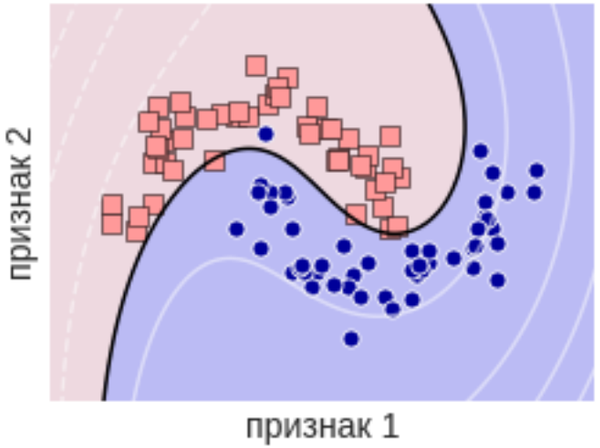
$C = 0.1$

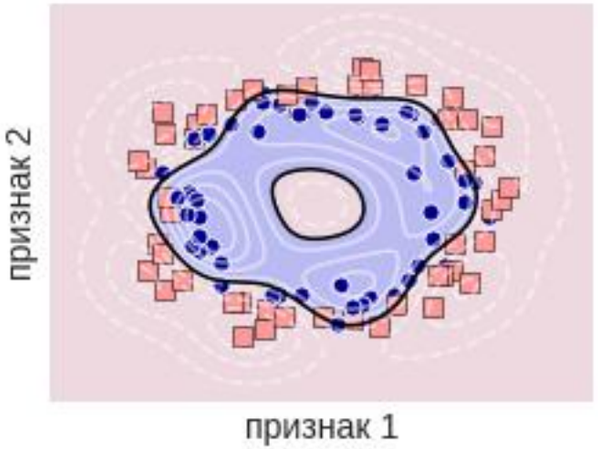


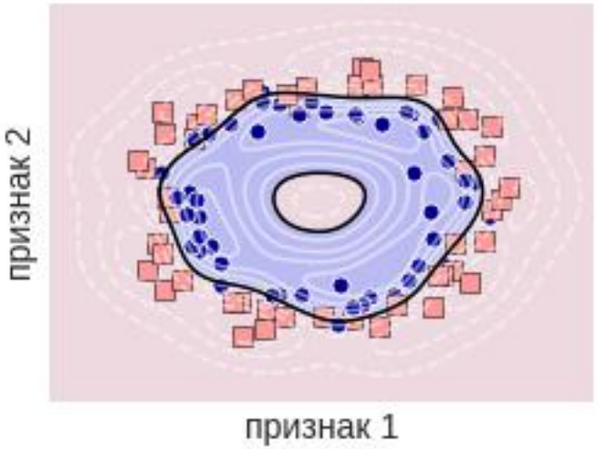
$C = 1$

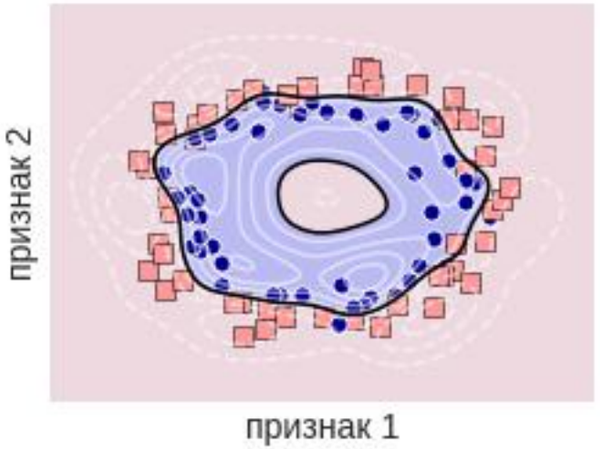


$C = 10$









Кернализация гребневой регресии

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

Тогда делается замена:

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

Гребневая регрессия

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

решение может быть записано как

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

Вроде здесь не используем попарные скалярные произведения (элементы матрицы XX^{T}), но… решение можно переписать!

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

Магия: Representer theorem

Доказательство (попробуйте сами)

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

SVD: $X = U\Lambda V$ **(чтобы не запутаться тут V без транспонирования)**

$$((U\Lambda V)^{\mathrm{T}}U\Lambda V + \lambda I)w = (U\Lambda V)^{\mathrm{T}}y$$

$$(V^{\mathrm{T}}\Lambda^{\mathrm{T}}U^{\mathrm{T}}U\Lambda V + \lambda I)w = V^{\mathrm{T}}\Lambda^{\mathrm{T}}U^{\mathrm{T}}y$$

$$(V^{\mathrm{T}}\Lambda^{2}V + \lambda I)w = V^{\mathrm{T}}\Lambda^{\mathrm{T}}U^{\mathrm{T}}y$$

$$U\Lambda^{-1}V(V^{\mathrm{T}}\Lambda^{2}V + \lambda I)w = U\Lambda^{-1}VV^{\mathrm{T}}\Lambda^{\mathrm{T}}U^{\mathrm{T}}y$$

$$(U\Lambda^{-1}VV^{\mathrm{T}}\Lambda^{2}V + \lambda U\Lambda^{-1}V)w = y$$

$$(U\Lambda V + \lambda U\Lambda^{-1}V)w = y$$

$$(U\Lambda VV^{\mathrm{T}}\Lambda U^{\mathrm{T}} + \lambda I)U\Lambda^{-1}Vw = y$$

$$U\Lambda^{-1}Vw = (U\Lambda VV^{\mathrm{T}}\Lambda U^{\mathrm{T}} + \lambda I)^{-1}y$$

$$V^{\mathrm{T}}\Lambda UU\Lambda^{-1}Vw = V^{\mathrm{T}}\Lambda U(U\Lambda VV^{\mathrm{T}}\Lambda U^{\mathrm{T}} + \lambda I)^{-1}y$$

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

Кернализация гребневой регресии

сравним число операций…

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

время (time)

$$O(mn^2 + n^3) \qquad O(m^2 n + m^3)$$

память (space)

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

Кстати

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

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 регресии

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

тогда $w = X^{\mathrm{T}}\alpha = \sum_{i=1}^{m} \alpha_i x_i \to \sum_{i=1}^{m} \alpha_i \underbrace{\varphi(x_i)}_{\equiv k(x_i,\cdot)}$, **где**

$$\alpha = (XX^{\mathrm{T}} + \lambda I_{m\times m})^{-1} y \to (K + \lambda I_{m\times m})^{-1} y,$$

Ответ нашей kernel-ridge-регрессии:

$$y = w^{\mathrm{T}} x \to \sum_{i=1}^{m} \alpha_i K(x_i, x)$$

матрицу попарных скалярных произведений и ядро обозначаем одной буквой

для справки

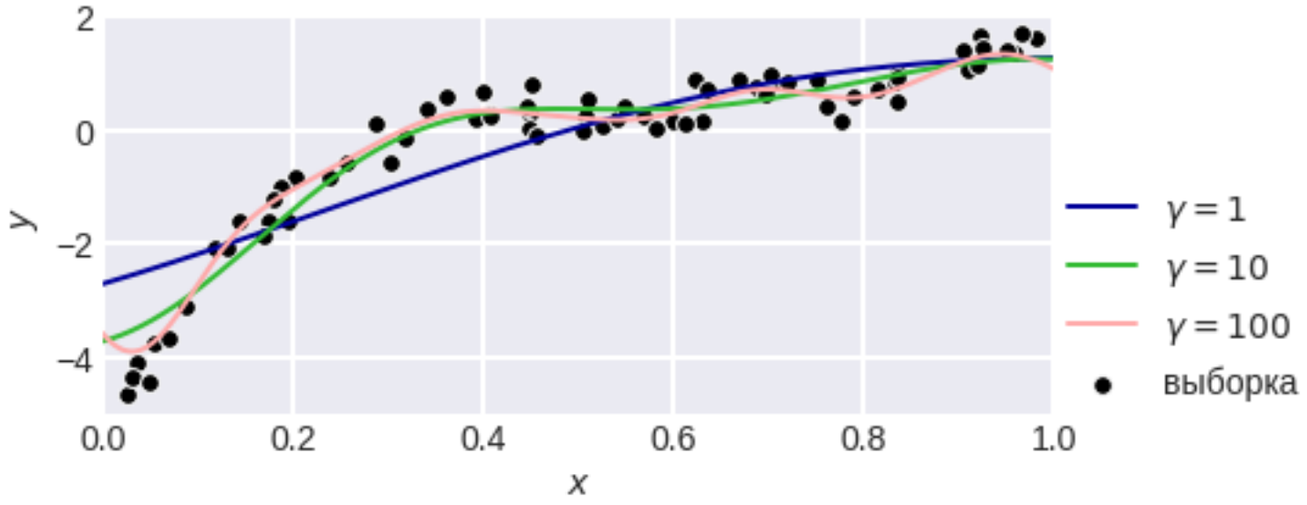
Hal Daume' «From Zero to Reproducing Kernel Hilbert Spaces in Twelve Pages or Less»

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

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

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

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

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

Математика ядер – операции над ядрами

поэтому разброс…

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

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

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

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

аналогично, квадраты расстояний (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

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

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

можно имитировать модификацией матрицы ядра:

$K \to W^{-1/2} K W^{1/2}$, **где**

$$W = \text{diag}(K(x_1, x_1), \ldots, K(x_m, x_m))$$

Проблема выбора ядра

часто надо настраивать гиперпараметры ядра
(скользящий контроль)

Есть теория выбора и настройки самого ядра… пример

$$K[i,:] = [k(x_i, x_1), k(x_i, x_2), \ldots, k(x_i, x_m)]$$

– 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, матрицу $K_{m \times m}$

есть методы увеличения скорости kernel-методов

RBF

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

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

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

система m уравнений с m неизвестными

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

$$j \in \{1, 2, \ldots, m\}$$

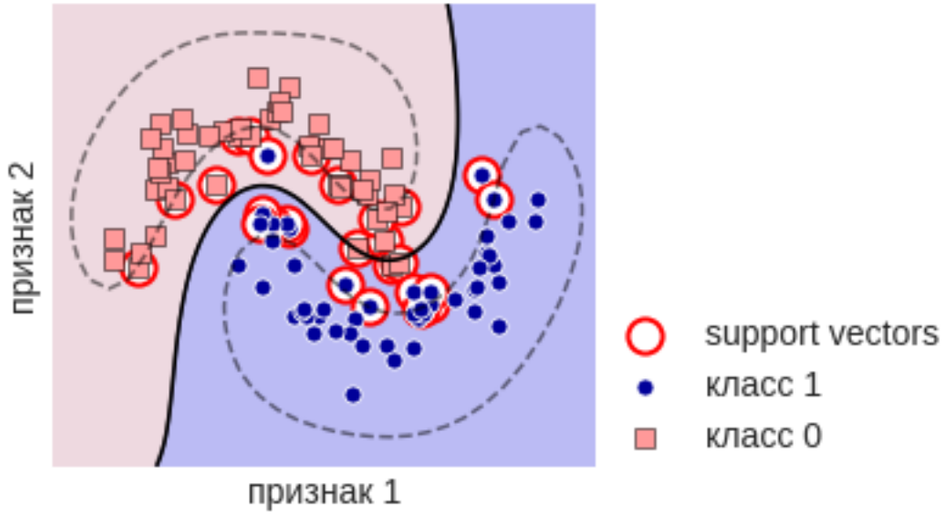
$$a(x) = \sum_{i=1}^{m} w_i \exp(-\gamma \| x - x_i \|^2)$$

RBF – проблема выбора эталонных точек

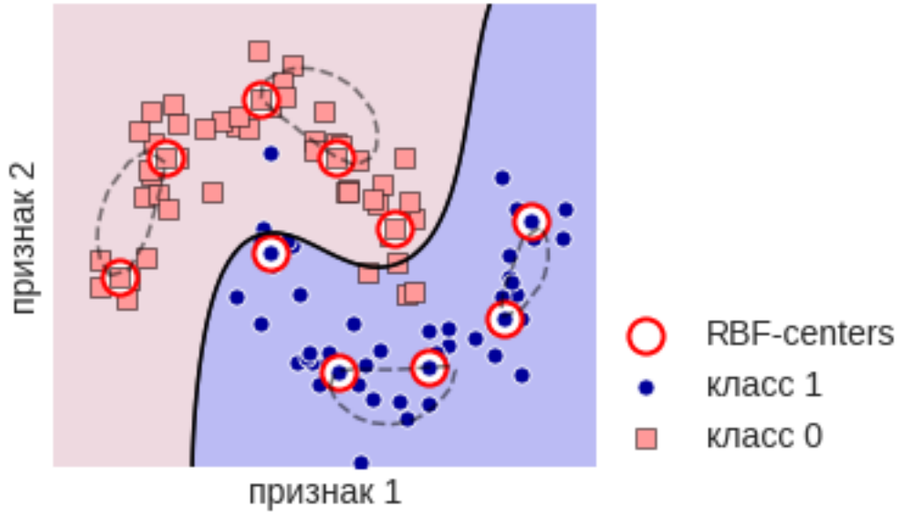
лучше в качестве центров использовать k эталонных точек
Как выбрать?

один из способов: кластеризация → центры кластеров

Support vectors



RBF centers



RBF – проблема настройки весов

если точек будет меньше…

$$\sum_{i \in S} w_i \exp(-\gamma \| x_j - x_i \|^2) = y_j$$

$$j \in \{1, 2, \ldots, m\}$$

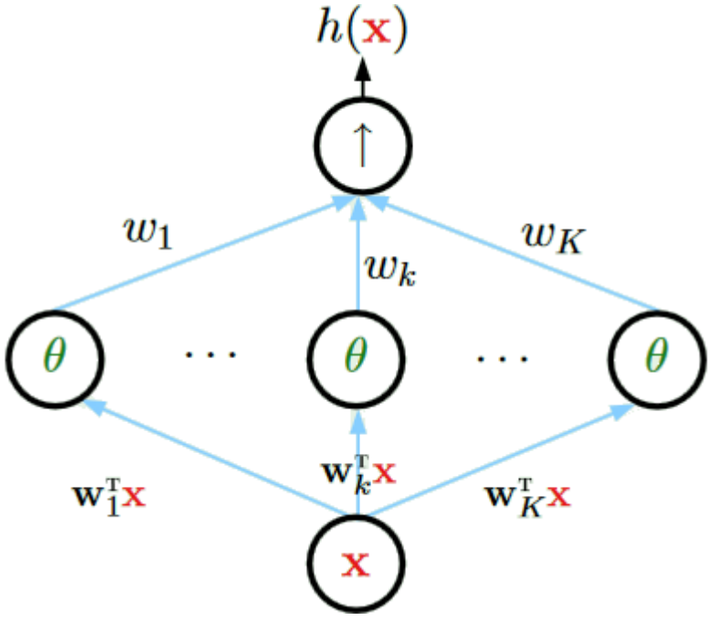
$$K_{m \times |S|} w = y$$

решение через псевдообратную

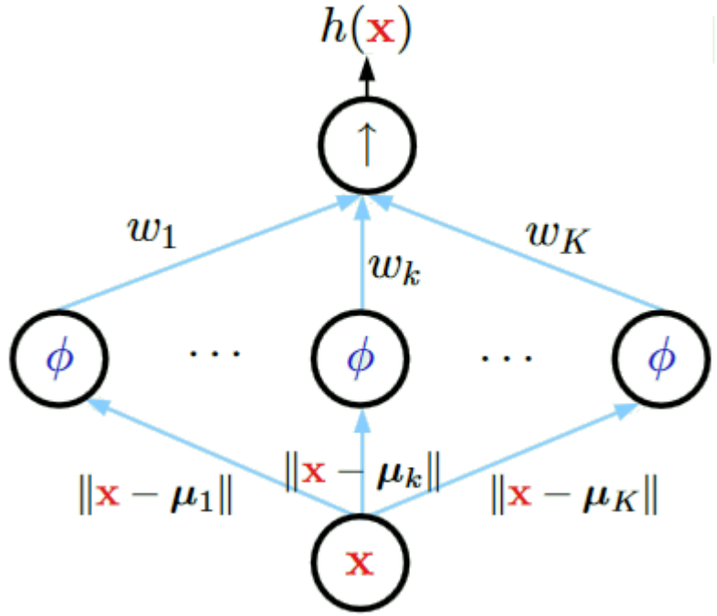
$$w = (K^{\mathrm{T}} K)^{-1} K^{\mathrm{T}} y$$

или RBF-сеть

RBF-сеть (Radial basis function network)

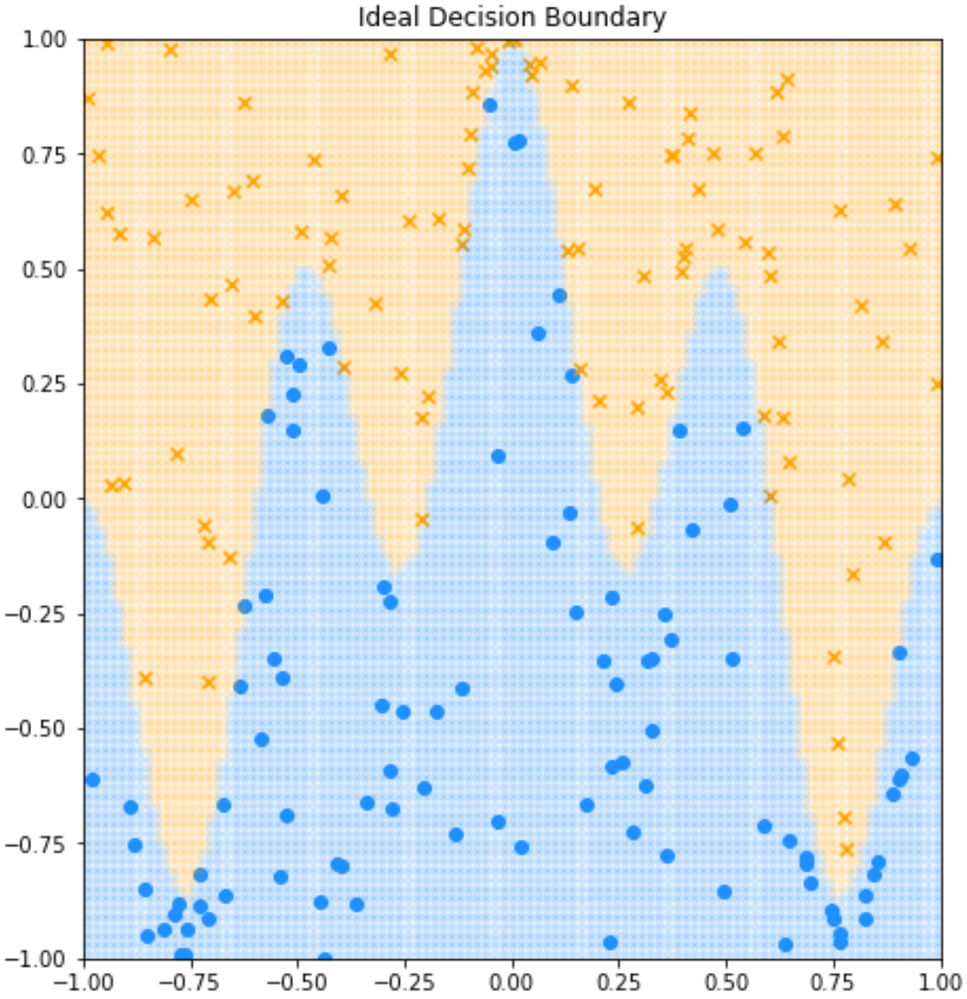


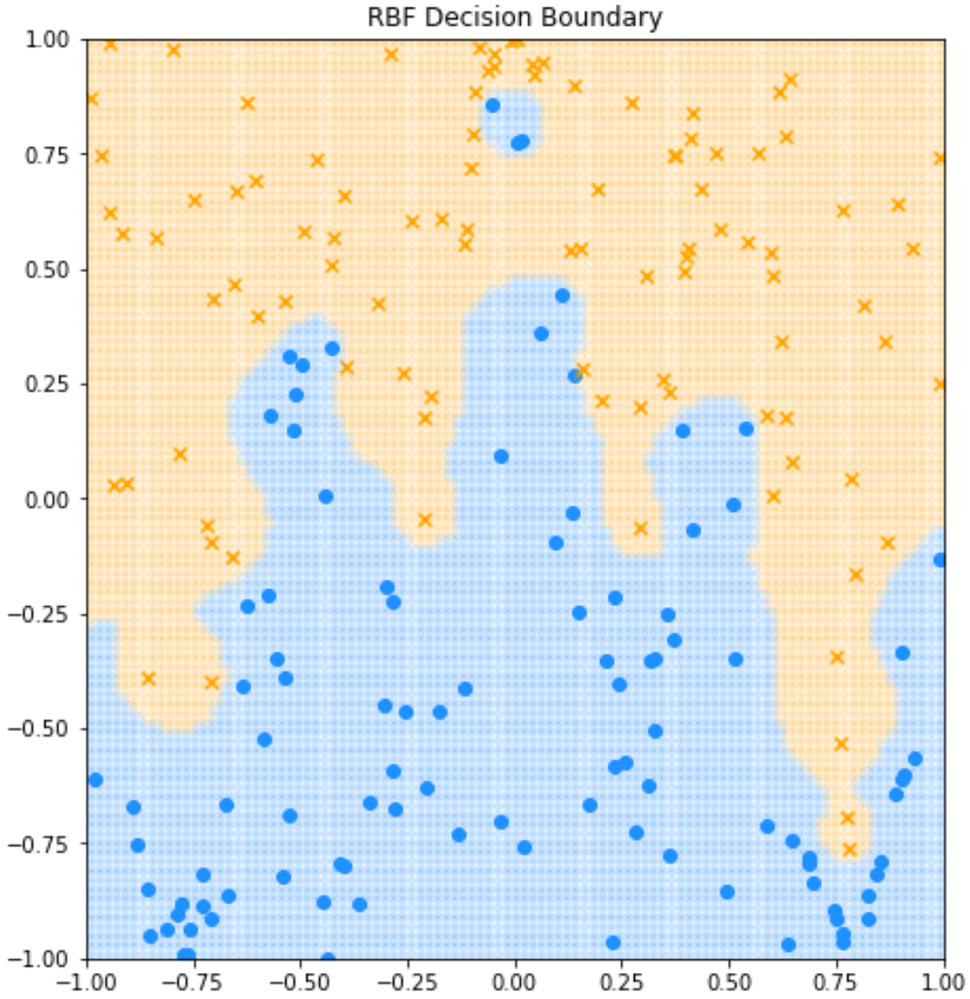
нейросеть



RBF-сеть

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

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

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

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

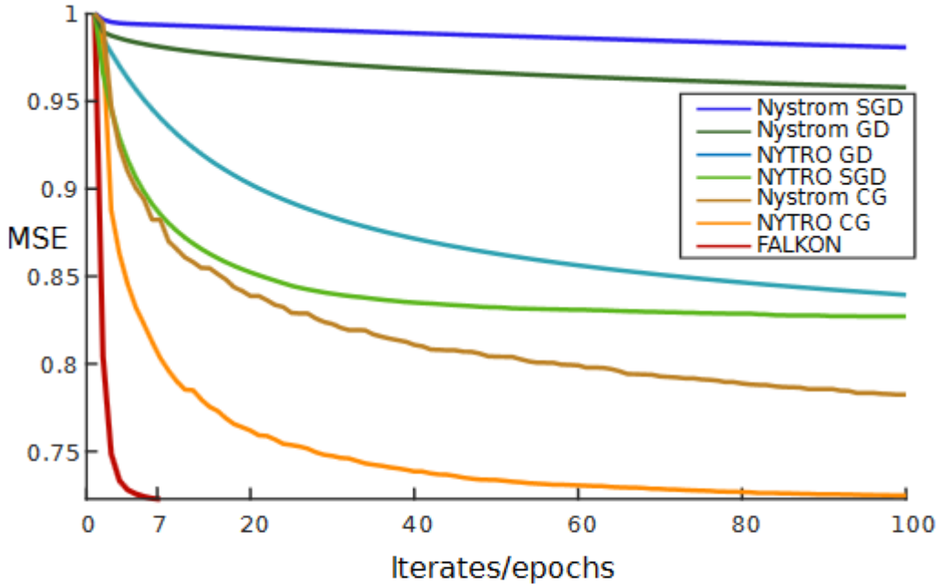


Figure 1: Falkon is compared to stochastic gradient, gradient descent and conjugate gradient applied to Problem (8), while NYTRO refer to the variants described in [23]. The graph shows the test error on the HIGGS dataset (1.1×10^7 examples) with respect to the number of iterations (epochs for stochastic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

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

Дальше

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

Также есть Kernel ICA, Kernel CCA

Ссылки

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

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

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

RBF https://en.wikipedia.org/wiki/Radial_basis_function

Итог

Можно решать сложные задачи линейными методами
~ пополнение признакового пространства

Есть «автоматические способы пополнения»

Многие методы можно «кернализовать»