«Машинное обучение»

Нелинейные методы

Александр Дьяконов

25 октября 2021 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

Как применять линейные методы

Проблема категориальных признаков…

	city	sex	income
0	Moscow	M	110
1	London	F	200
2	London	M	140
3	Paris	M	120
4	Moscow	F	190

как эти признаки использовать в алгоритме?

Как применять линейные методы: кодирование категорий

по номеру категории Label Encoding

	city	sex	income	city_le	city_fz	city_rnd	sex_le	sex_fz	sex_rnd
0	Moscow	M	110	1	0	0.63	1	0	0.22
1	London	F	200	0	1	0.75	0	1	0.20
2	London	M	140	0	1	0.75	1	0	0.22
3	Paris	M	120	2	2	0.50	1	0	0.22
4	Moscow	F	190	1	0	0.63	0	1	0.20

```
from sklearn import preprocessing
le = preprocessing.LabelEncoder()

for name in cols:
    # лексикографический порядок
    data[name + '_le'] = le.fit_transform(data[name])
    # в порядке упоминания
    data[name + '_fz'] = pd.factorize(data[name])[0]
    # случайно
    dct = dict(zip(data[name].unique(),
               np.random.rand(data[name].nunique()).round(2)))
    data[name + '_rnd'] = data[name].map(dct)
```

Как применять линейные методы: кодирование категорий

Dummy-кодирование / One-hot-encoding

	city	sex	income	city=Moscow	city=London	city=Paris	sex=M	sex=F
0	Moscow	M	110	1	0	0	1	0
1	London	F	200	0	1	0	0	1
2	London	M	140	0	1	0	1	0
3	Paris	M	120	0	0	1	1	0
4	Moscow	F	190	1	0	0	0	1

OneHotEncoder (по умолчанию sparse, раньше – только с числами)

```
from sklearn import preprocessing
ohe = preprocessing.OneHotEncoder(sparse=False)
tmp = ohe.fit_transform(data[cols]).astype(int)
tmp = pd.DataFrame(tmp,
                   columns=['OHE_' + str(i) for i in range(tmp.shape[1])])
data = pd.concat([data, tmp], axis=1)
```

Самый простой способ

```
pd.get_dummies(data)
```

Как применять линейные методы: кодирование категорий

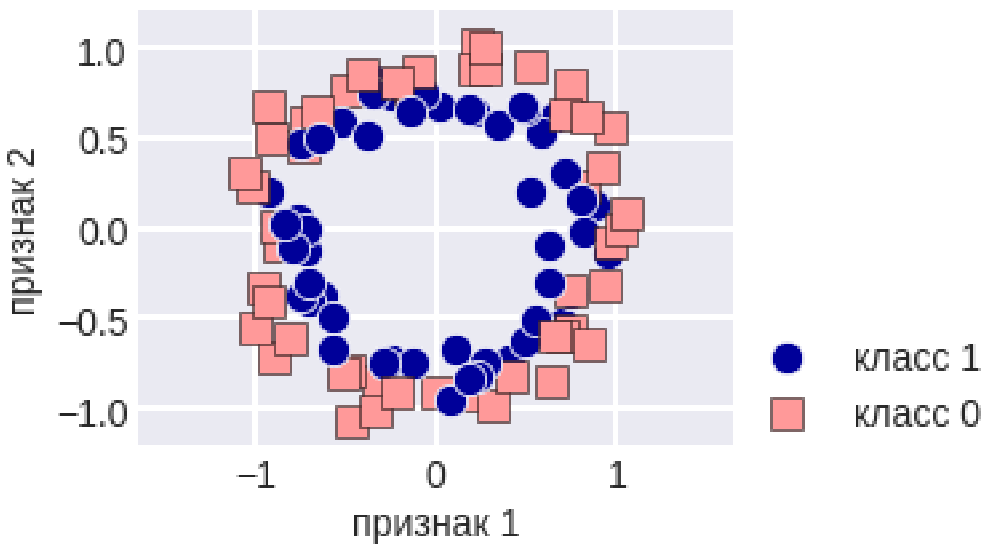
По значению целевого – Target Encoding

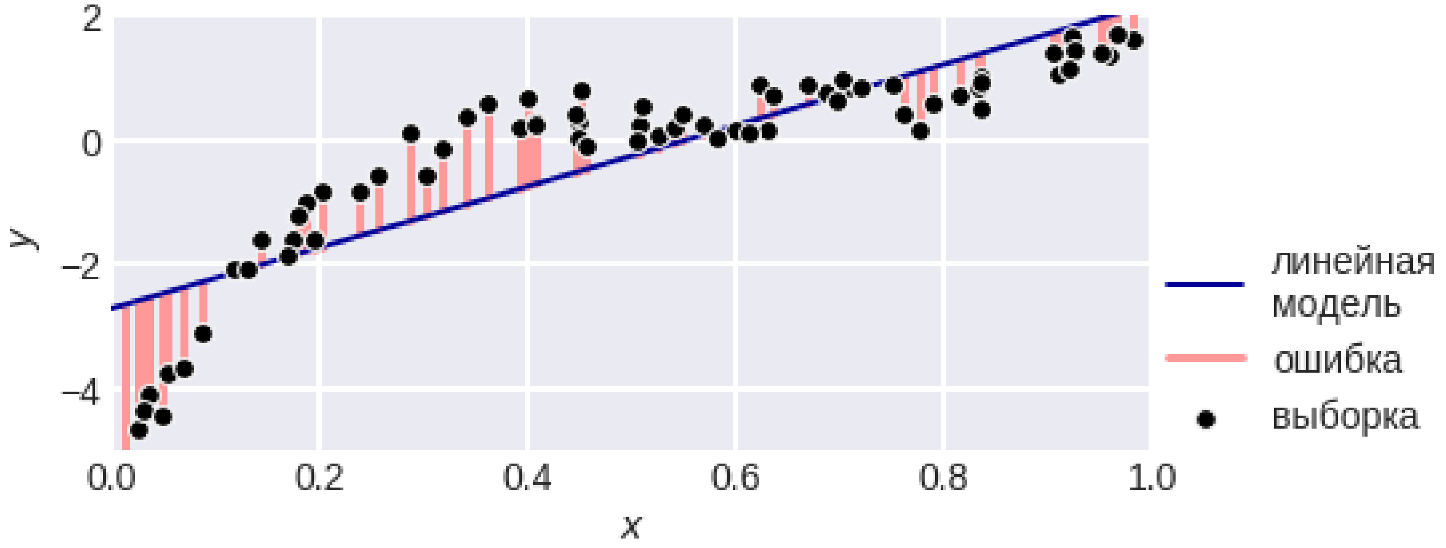
- Mean Target Encoding
- Std Target Encoding
- ...

	city	sex	income	city_mt	sex_mt	target
0	Moscow	M	110	0.5	0.67	1
1	London	F	200	0.5	0.50	1
2	London	M	140	0.5	0.67	0
3	Paris	M	120	1.0	0.67	1
4	Moscow	F	190	0.5	0.50	0

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

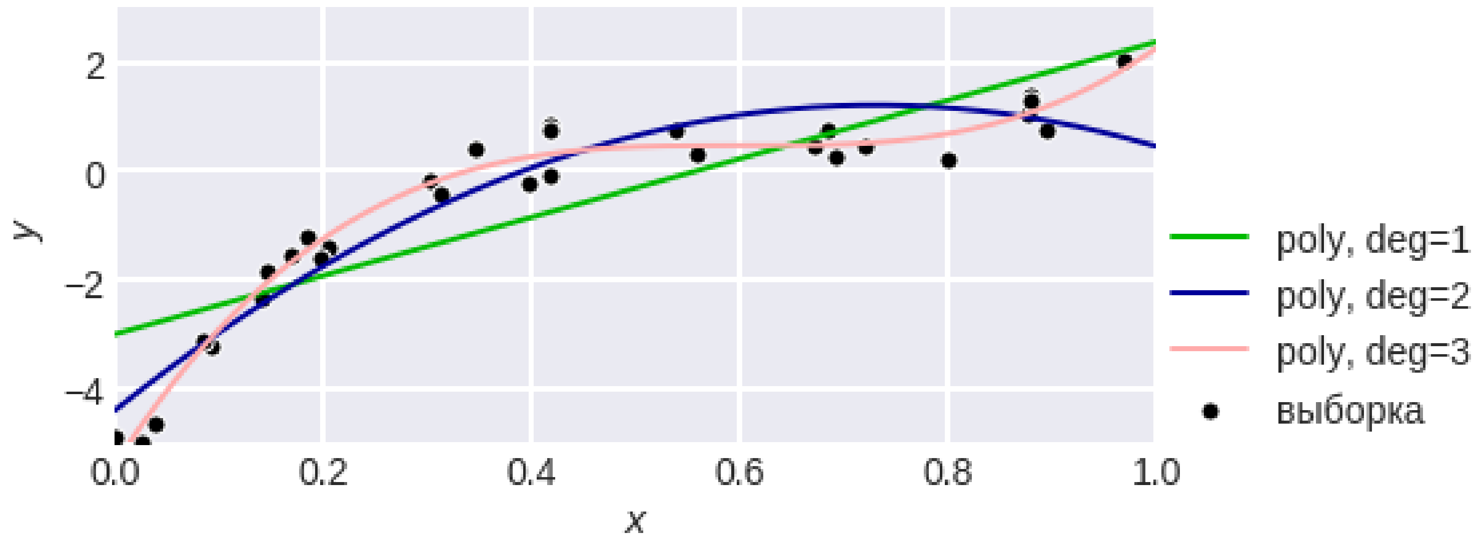
- **локально линейные методы**
  Пример: Local Regression

- **ядерные методы (Kernel Tricks)**

использование нелинейной модели

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

Деформация: полиномиальная модель

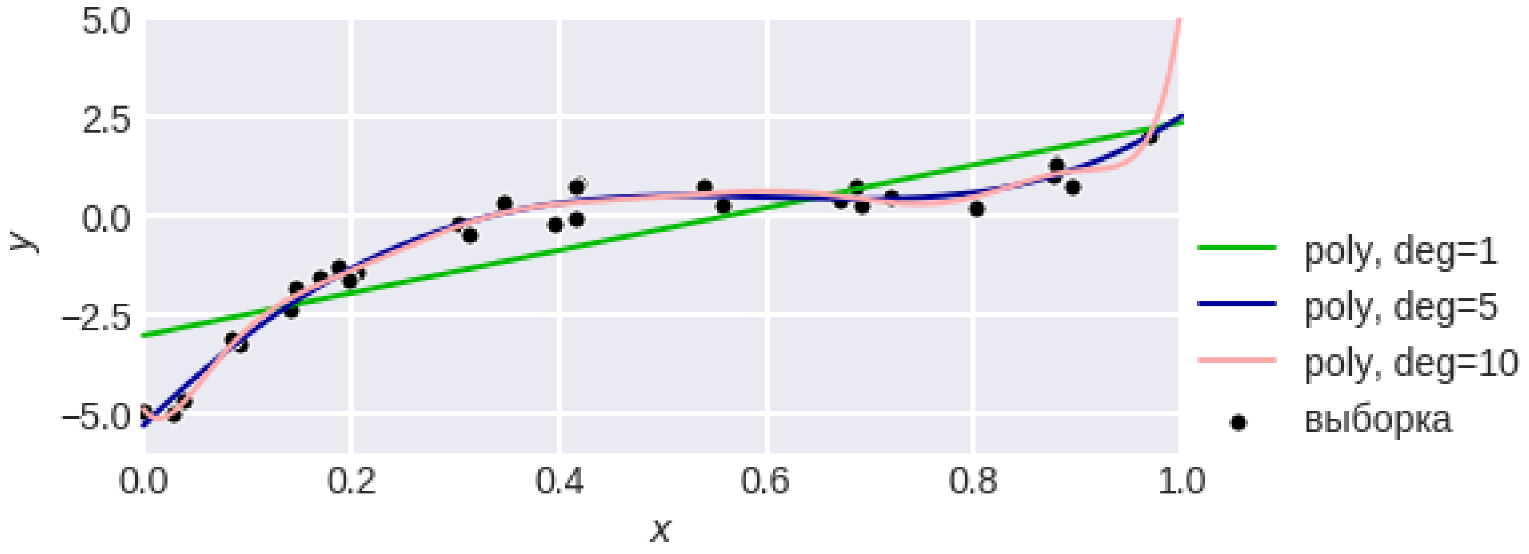


добавляем признаки-мономы

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

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

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!

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

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:

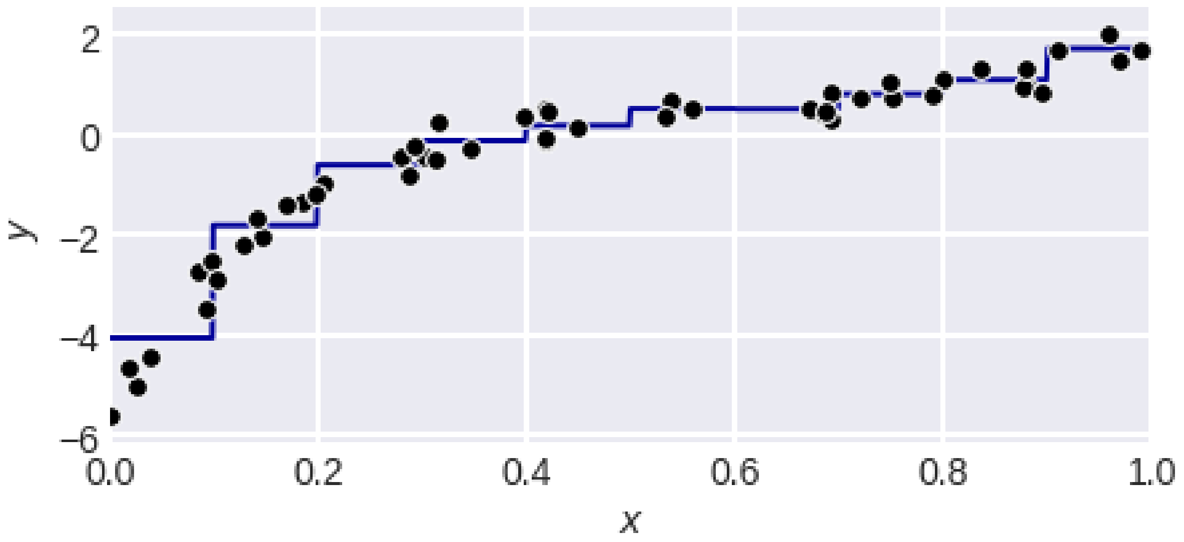
- **квантили**
- **«межкластерные» точки**
- **особые точки**

(например, круглые суммы в признаке «зарплата»)

Использование других базисных функций

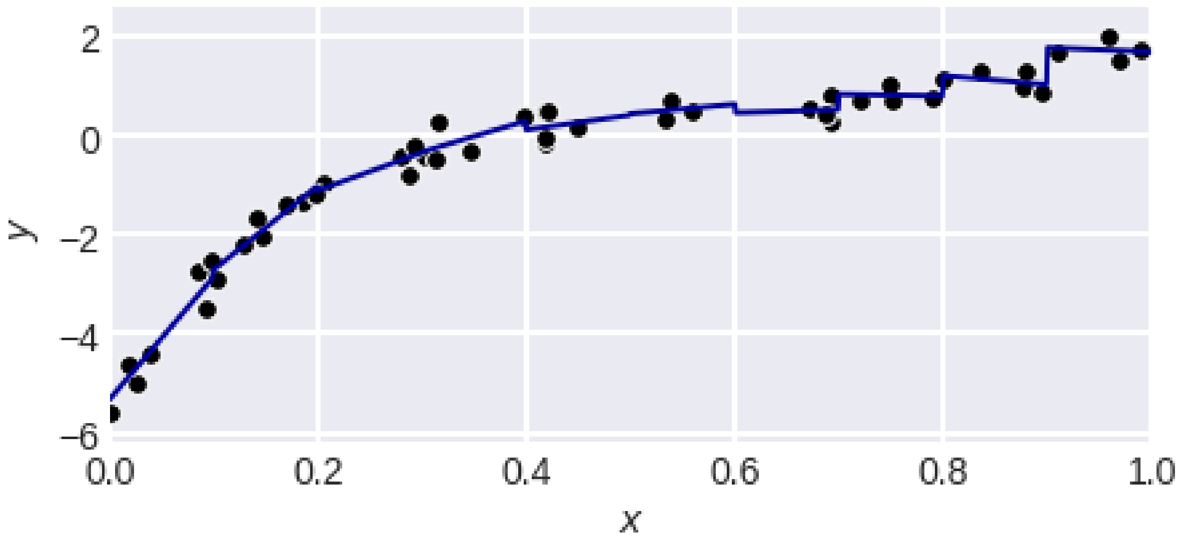
сравнение с порогом

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



кусочно-линейная регрессия

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



аналогично кусочно-полиномиальная

Использование других базисных функций

«Сплайны» получаются при добавлении функций вида

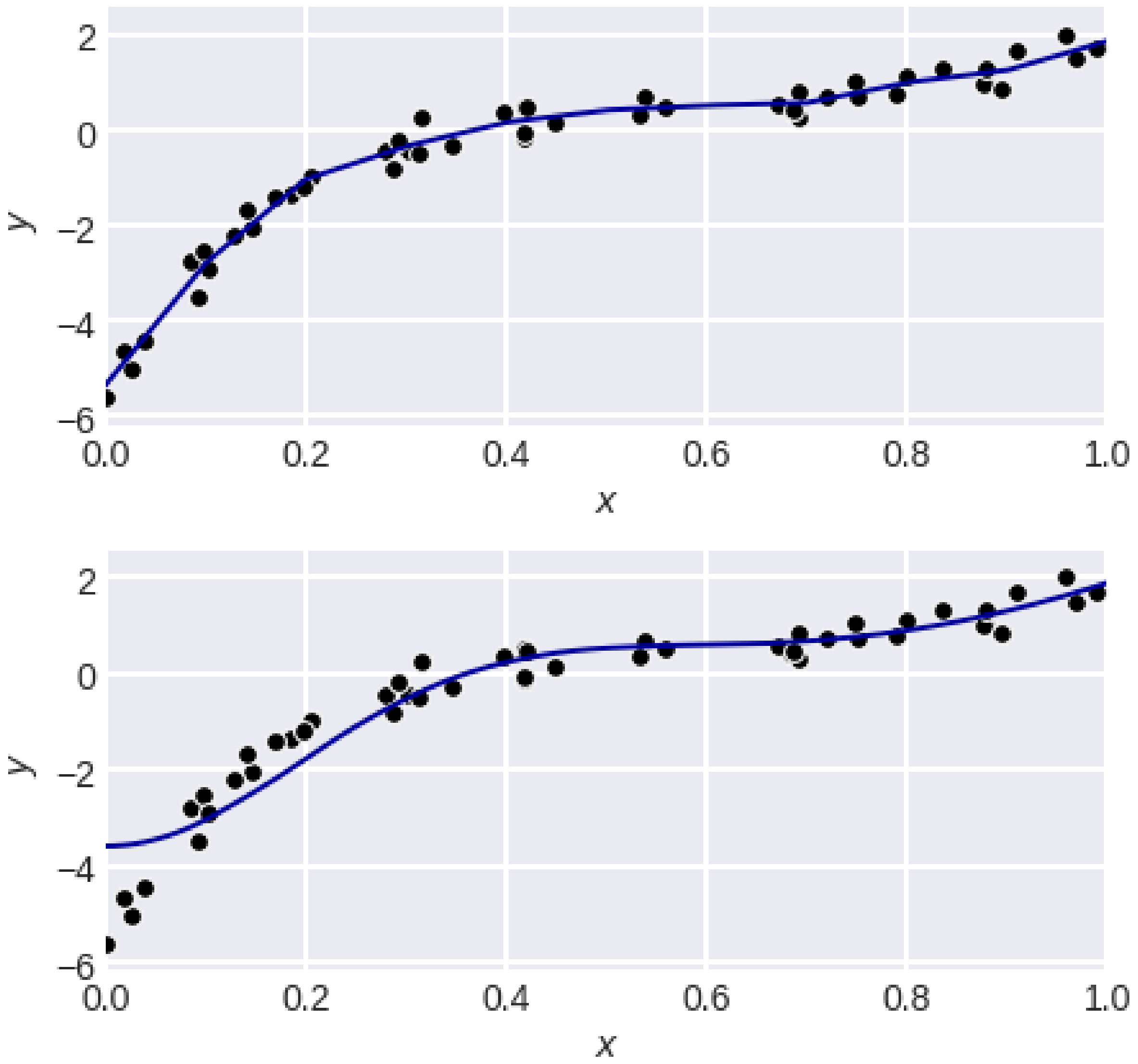
$$\begin{cases} |x-\theta_i|^k, & x \ge \theta_i, \\ 0, & x < \theta_i, \end{cases}$$

Здесь (вверху) –

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

(внизу) –

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



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

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

Thin plate spline

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

Multiquadric

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

Inverse multiquadric

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

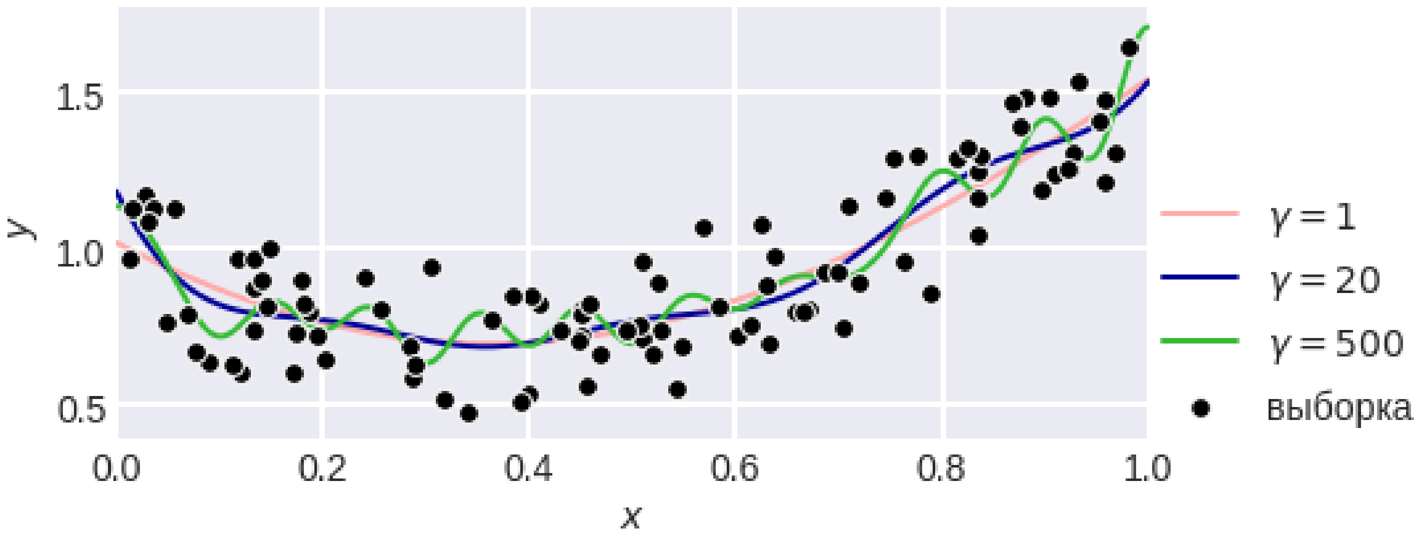
Inverse quadratic

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

Polyharmonic spline

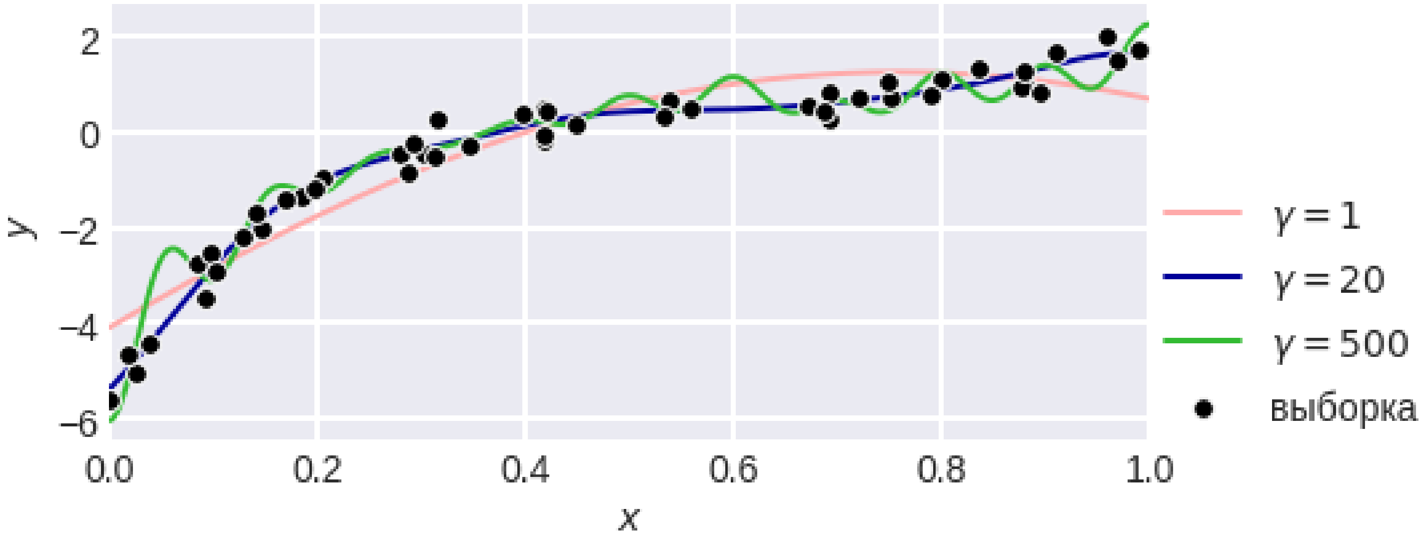
$$f(r)=\begin{cases} r^k, & k=1,3,5,\ldots \\ r^k\ln r, & k=2,4,6,\ldots \end{cases}$$

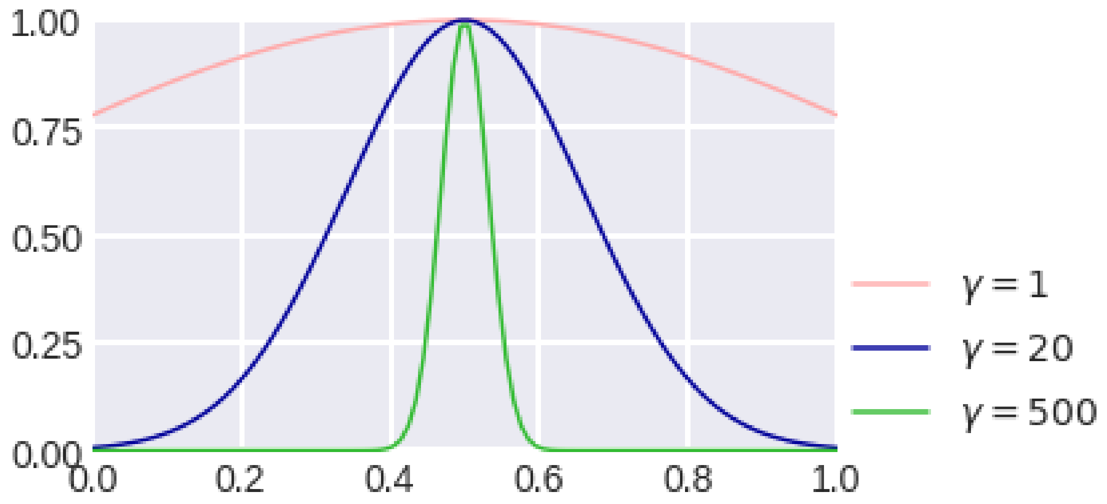
RBF-ядро: пример для регрессии



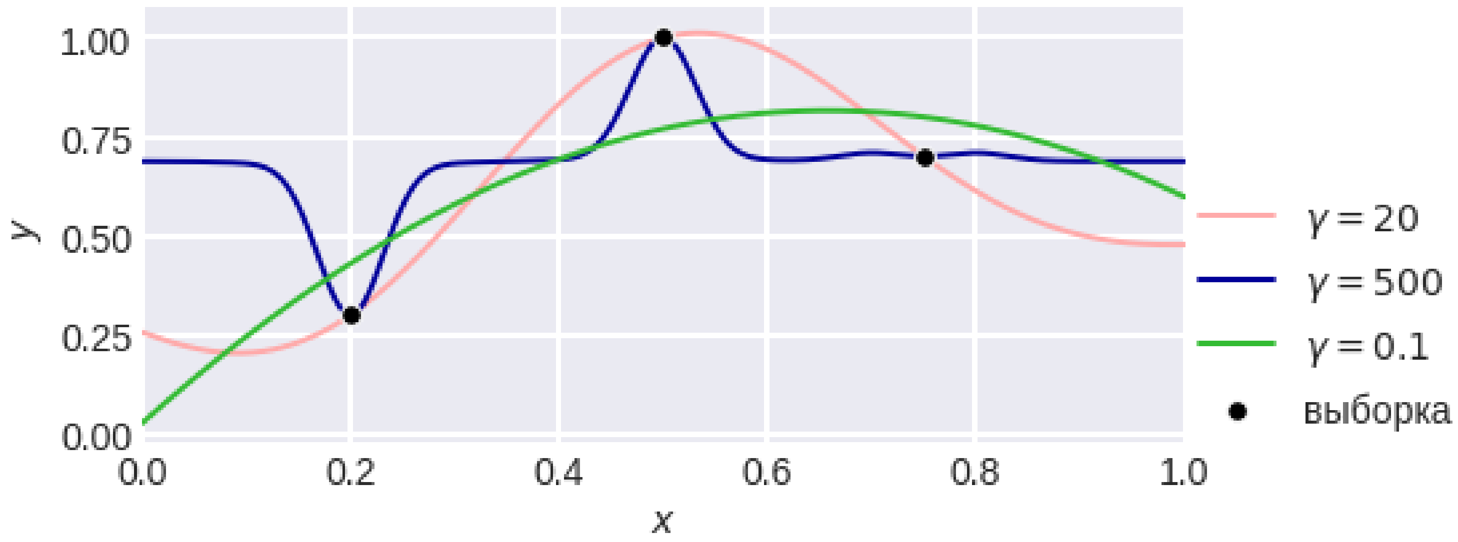
$$\varphi(x) = \exp(-\gamma \| x - z \|^2)$$

тут, правда, выбраны равномерные эталонные точки

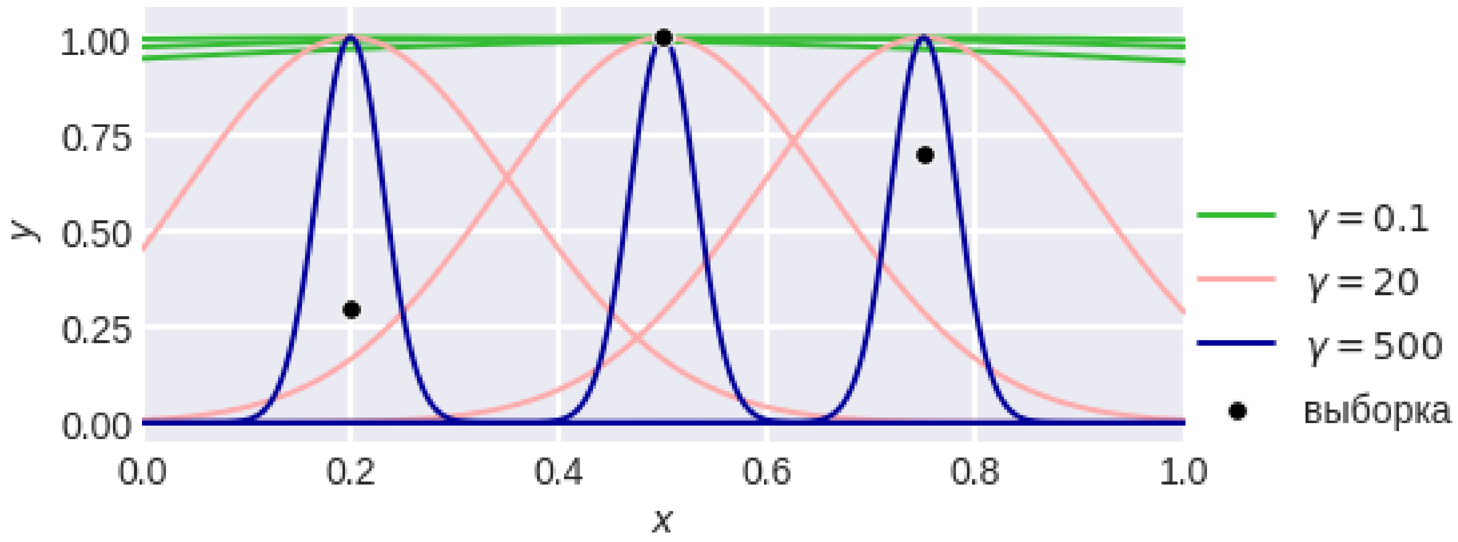




RBF-ядро: эффект от ширины ядра



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 ~ 1 / расстояние до точки)**

Можно использовать линейный метод
или полиномиальный
Local Regression = Local Polynomial Regression = Moving Regression

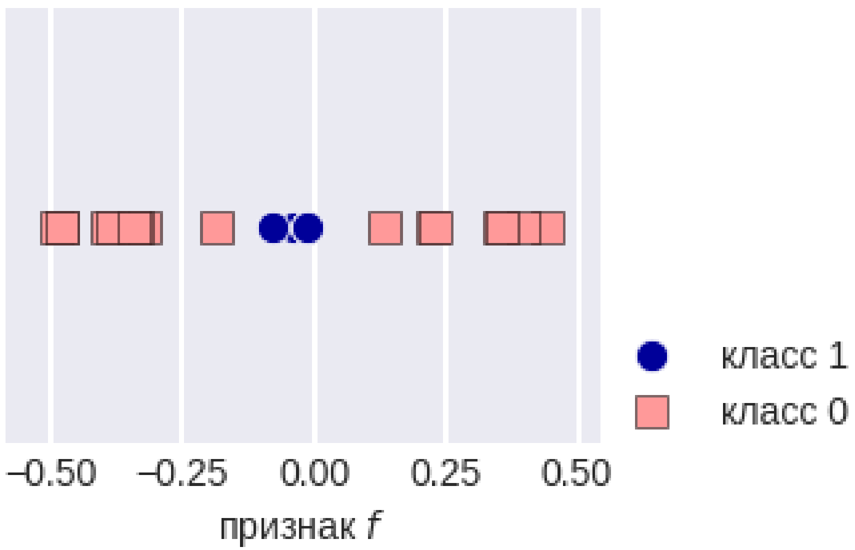
см. LOESS – locally estimated scatterplot smoothing (Savitzky–Golay filter)
LOWESS – locally weighted scatterplot smoothing (locally weighted polynomial regression)
https://towardsdatascience.com/loess-373d43b03564

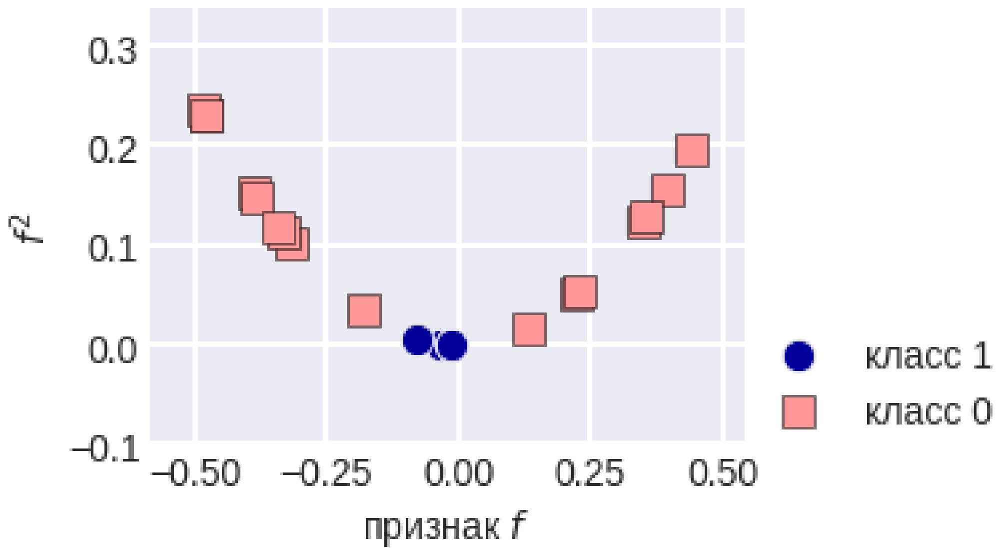
вернёмся в предобработке данных

Ядерные методы (Kernel Tricks)

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

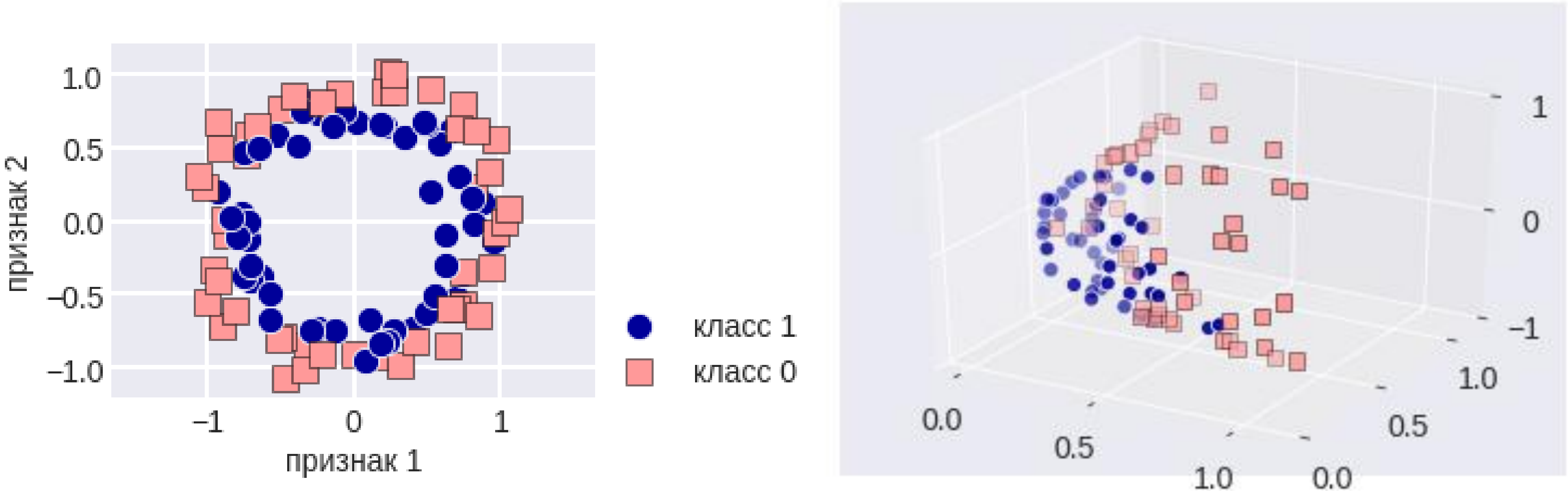
Пример перехода $(x) \to (x, x^2)$





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 $K(x,z)=(x^{\mathrm{т}}z)^2$

$$K(x,z)=(x_1z_1+\ldots+x_nz_n)^2=x_1^2z_1^2+\ldots+x_n^2z_n^2+\sum_{ij}\sqrt{2}x_ix_j\sqrt{2}z_iz_j=\varphi(x)^{\mathrm{т}}\varphi(z)$$

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

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

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** $K(x,z)=\varphi(x)^{\mathrm{т}}\varphi(z)$
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 $K : X \times X \to \mathbb{R}$ **–** **<u>неотрицательно определённое ядро</u>**
(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

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

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

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

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

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

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…

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

теперь…

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

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

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

Наблюдение

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

Запись напоминает простейшую нейронную сеть [Воронцов]

+ при настройке единственное решение
+ автоматическое определение числа нейронов в скрытом слое $|S|$

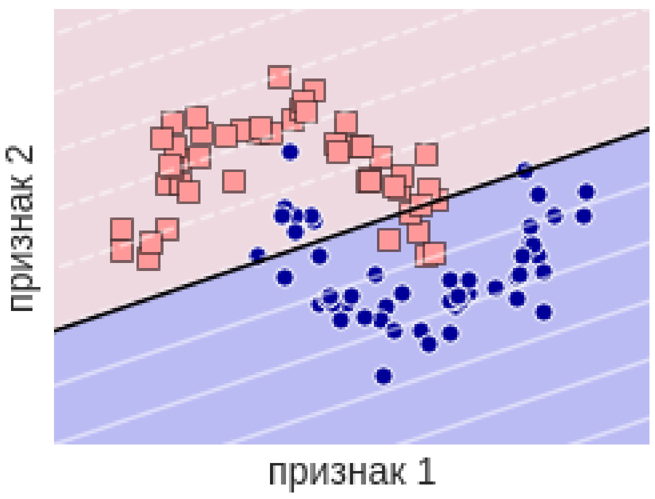
– непонятно, как выбрать ядро, параметры метода, признаки
(это не автоматизировано)

Подводные камни

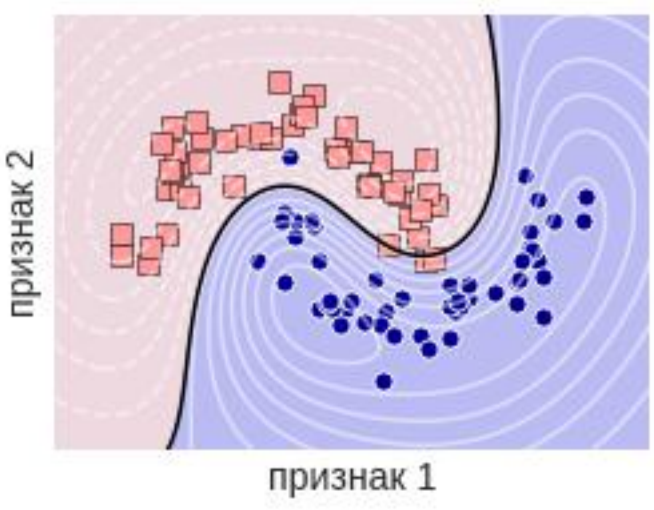
- использование ядер может вызывать переполнения…
- если мы сами придумаем пространство, в которое хотим перейти, то, скорее всего, не найдём подходящего ядра
- геометрия в новом пространстве нам, на самом деле, не совсем интуитивно ясна

SVM с разными ядрами

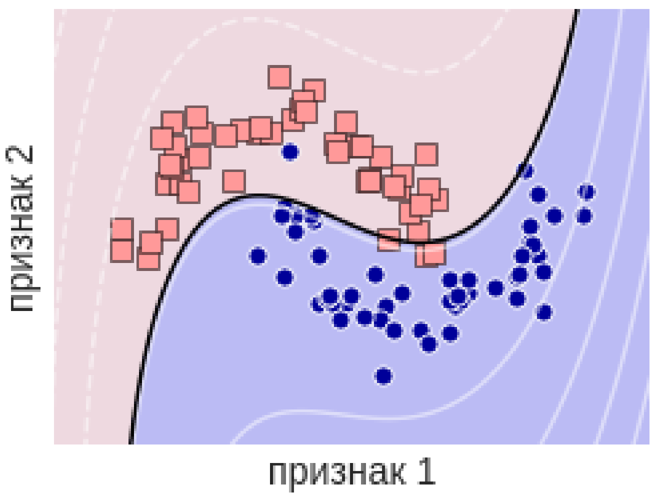
linear



RBF



poly



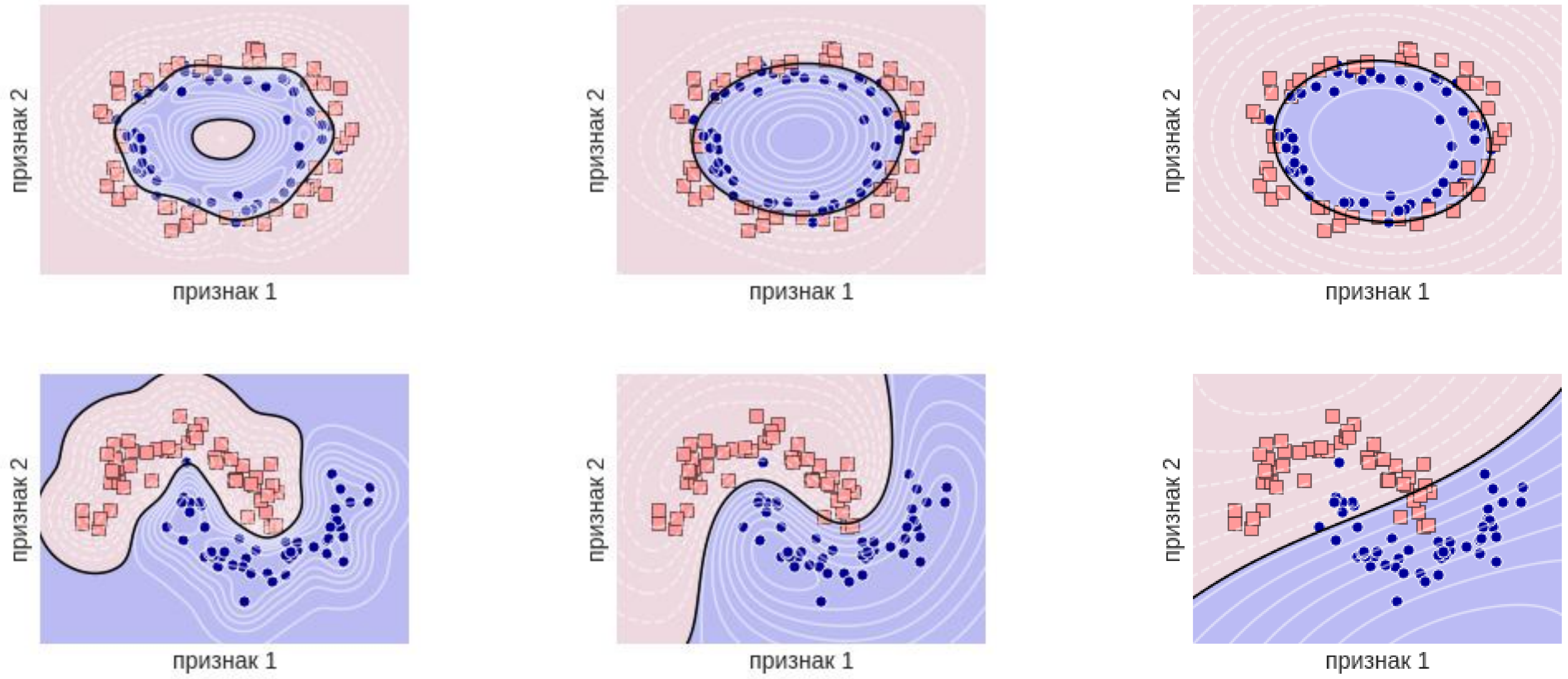
Минутка кода

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

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

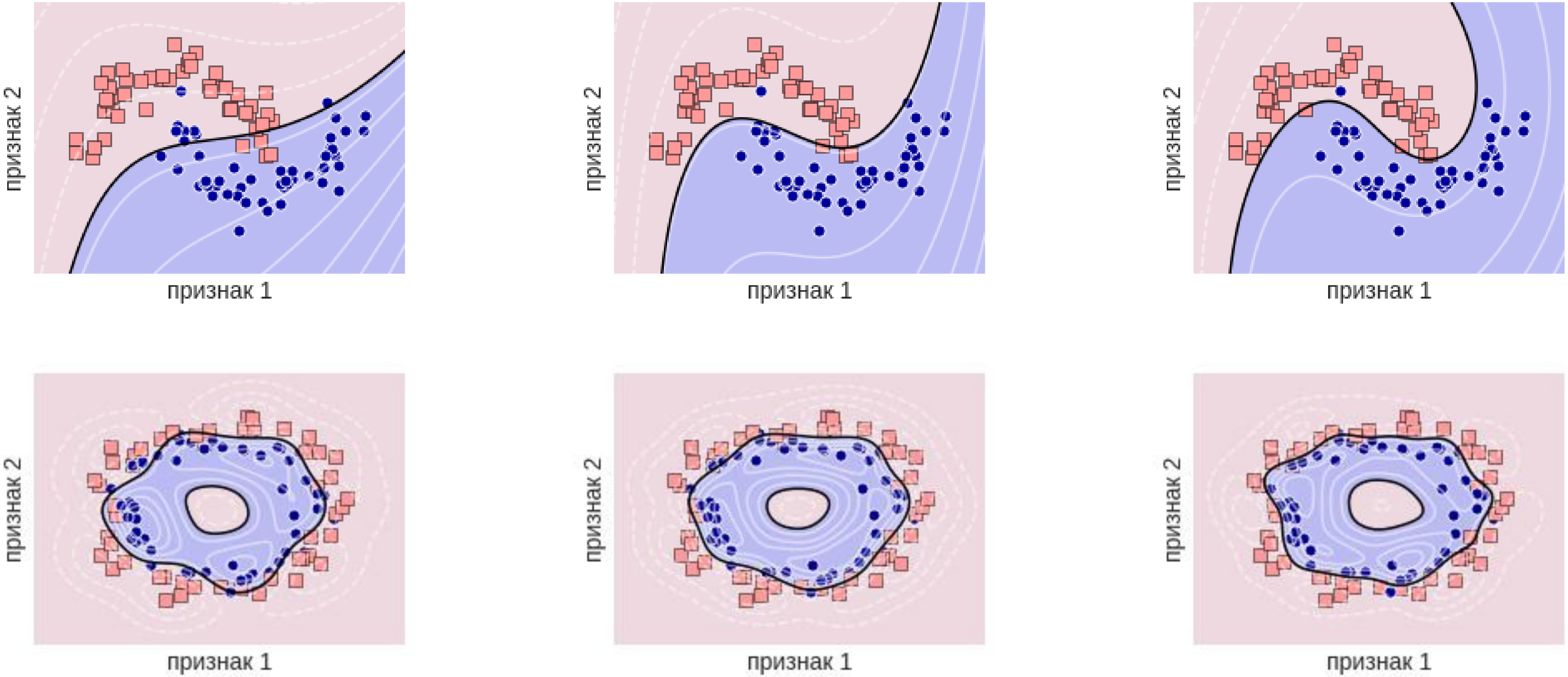
SVM с RBF-ядрами разной ширины

$\gamma = 10$ $\gamma = 1$ $\gamma = 0.1$



SVM с регуляризацией

$C = 0.1$ $\qquad C = 1$ $\qquad C = 10$



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!

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

доказательство Representer theorem (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: $X=U\Lambda V^{\mathrm{T}}$ (используем полное разложение $(m\times m)(m\times n)(n\times n)$)

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

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

Осталось доказать,
что синие матрицы равны;)

$$\mathrm{diag}(\lambda_1/(\lambda_1^2+\lambda),\ldots,\lambda_k/(\lambda_k^2+\lambda))_{n\times m}$$
$$k=\mathrm{rank}(X)$$

Кернализация гребневой регресии

сравним число операций...

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

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

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

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:

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

матрицу попарных скалярных произведений и ядро обозначаем одной буквой

для справки

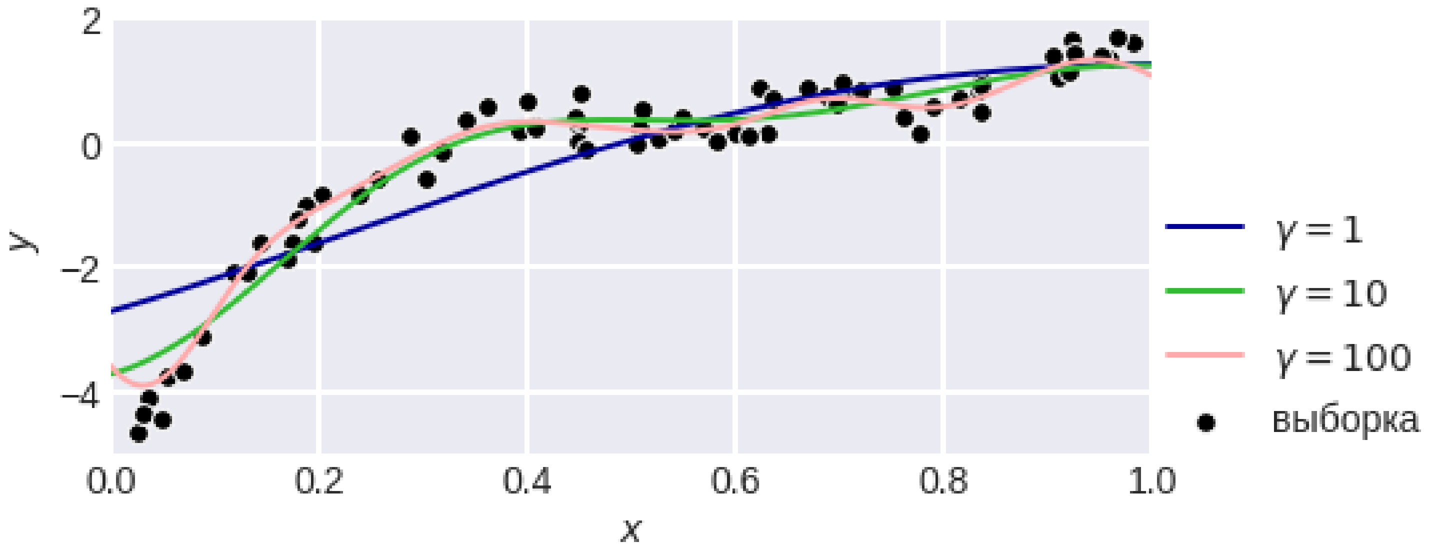
Hal Daume' «From Zero to Reproducing Kernel Hilbert Spaces in Twelve Pages or Less»

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(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

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

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

можно имитировать модификацией матрицы ядра:

$K \to W^{-1/2}KW^{-1/2}$, **где** $W = \mathrm{diag}(K(x_1, x_1), \ldots, K(x_m, x_m))$

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»