

## Лекция 5: поиск аномалий с обучением без учителя

# Проблемы поиска аномалий в «обучении без учителя»

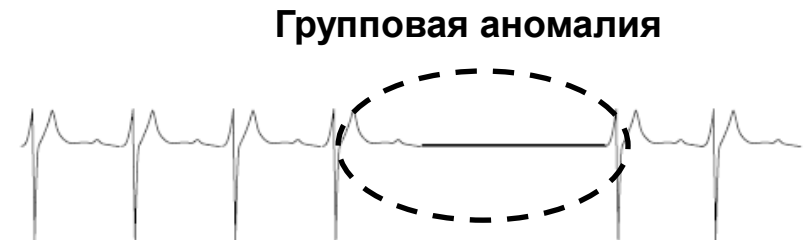
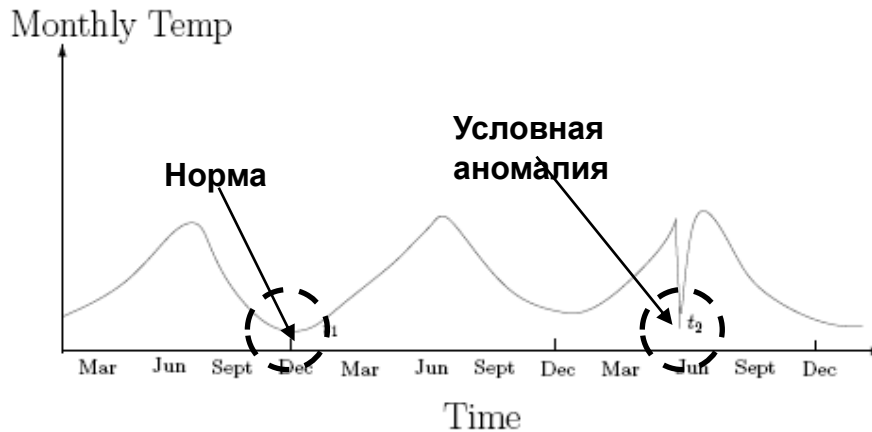
- Выявление аномалий – сложная задача:
  - Граница между нормой и аномалией не всегда очевидна и не всегда однозначна – вопрос эффективности алгоритма?
  - Нет общепринятого понятия аномальности, зависит от предметной области и от алгоритма поиска
  - «Шум» в данных – «интересная» аномалия или случайный выброс?
  - Нормальное поведение может меняться во времени - «интересная» аномалия или изменение поведения?
  - Интерпретируемость – почему аномалия?
- Типы аномалий:
  - Точечные (выбросы), «Условные», «Групповые»
- «Разметка» аномалий:
  - Степень аномальности или бинарная метка

# Практические приложения

- Безопасность (финансовая, компьютерная, гражданская)
- Медицина
- Промышленные нештатные ситуации
- Обработка мультимедиа
- Поиск новых тем в Text mining
- Выявлением редких событий
- ...

# НЕ точечные = условные и групповые аномалии

- «Условные» аномалии:
  - Точка является аномалией при определенных условиях
  - Одна и та же точка может быть нормальной и аномальной в разном окружении (контексте)
  - Требуется определения понятия контекста
  - Чаще всего – во временных рядах
- «Групповые» аномалии (каждая может быть нормальной):
  - Требуется какая-либо зависимость между точками, например:
  - Временные ряды
  - Географические и пространственные данные

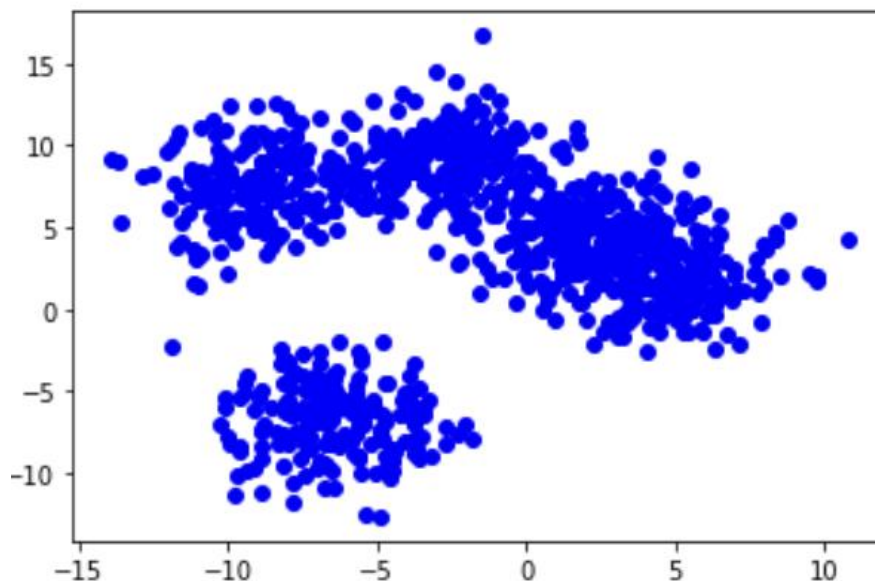


# Типы точечных аномалий и методы их поиска

- Одномерные (вероятностные)
  - Пользовательские интервалы
  - Отклонение от мат. ожидания или медианы
  - Экстремальные перцентили
- Метрические (KNN и кластеризация)
- Статистические (параметрические и нет)
- Анализ отклонений (ошибка реконструкции)
  - Матричные разложения (в том числе робастные и нелинейные)
  - Нейросетевые автокодировщики, SOM

# Демонстрационный пример

```
# Генерация данных гауссового облака точек  
from sklearn.datasets import make_blobs  
  
n_samples = 1000  
X_one, y_one = make_blobs(n_samples=n_samples,  
                           centers=5, cluster_std=2,  
                           random_state=42)  
plt.scatter(X_one[:, 0], X_one[:, 1], c='blue')
```



# Метрические методы

## ■ Основной постулат:

- «рядом с нормальными точками много соседей, рядом с аномалиями – нет»

## ■ Два этапа:

1. Расчет расстояний и числа соседей (как правило, строятся поисковые индексы)
2. Анализ структуры ближайших соседей и «разметка» точек

## ■ Категории:

- На основе расстояний – аномалии наиболее удалены от остальных
- На основе плотности – аномалии лежат в области с невысокой плотностью
- Для одномерного случая и известного распределения почти всегда можно доказать эквивалентность

# Определение аномалий в методах ближайших соседей

## ■ Метрические методы

- Точка  $x$  есть  $DB(p, D)$  аномалия, если по меньшей мере  $p$ -я часть остальных точек лежит дальше чем  $D$  от нее:

$$|\{x_i \mid x_i \in X, d(x, x_i) > D\}| \geq n * p$$

## ■ На основе оценки плотности

- Расчет локальной плотности распределения вокруг каждой точки
- Области с невысокой плотностью - аномалии

## ■ Методы:

- Глобальные метрические методы
- Local Outlier Factor (LOF)
- Connectivity Outlier Factor (COF)
- Multi-Granularity Deviation Factor (MDEF)



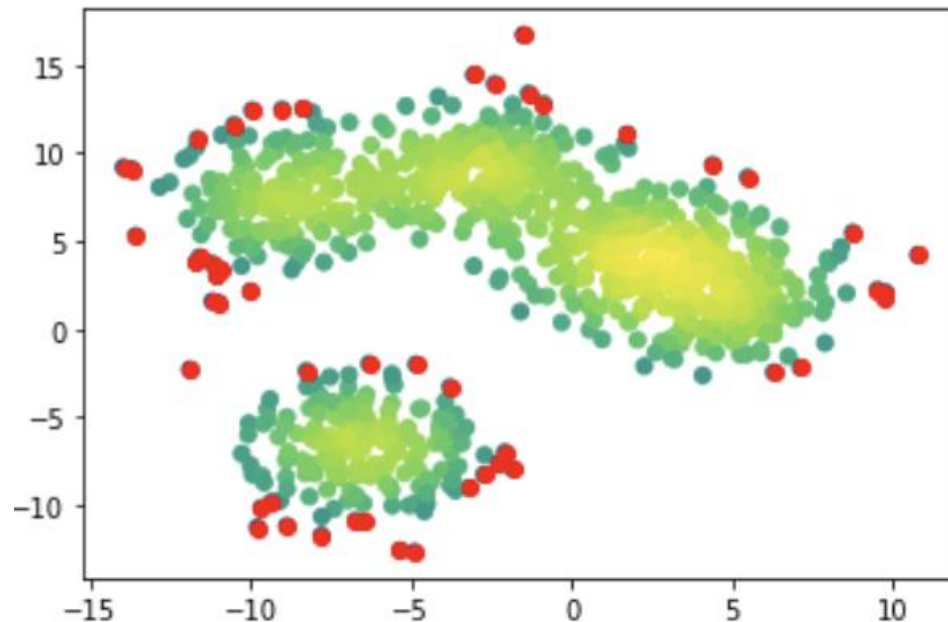
# Глобальные метрические методы

## ■ Общая процедура:

- Для каждой точки  $d$  считается расстояние до  $k$ -го ближайшего соседа  $d_k$
- Все точки упорядочиваются по  $d_k$
- Исключения – точки с наибольшим  $d_k$  (а значит, с наименьшим числом соседей)
- Обычно берется  $n\%$  точек с наибольшим  $d_k$  как исключения
- Не годится для данных со сложными распределениями

# Демо пример

```
from sklearn.neighbors import NearestNeighbors
nbrs = NearestNeighbors(n_neighbors = 100)
nbrs.fit(X_one)
distances, indexes = nbrs.kneighbors(X_one)
E=-distances.max(axis=1)
X_anom=X_one[E<pd.DataFrame(E).quantile(q=0.05)[0]]
plt.scatter(X_one[:, 0], X_one[:, 1], c=E)
plt.scatter(X_anom[:, 0], X_anom[:, 1], c="red")
```



# Local Outlier Factor (LOF)

## ■ Общая процедура:

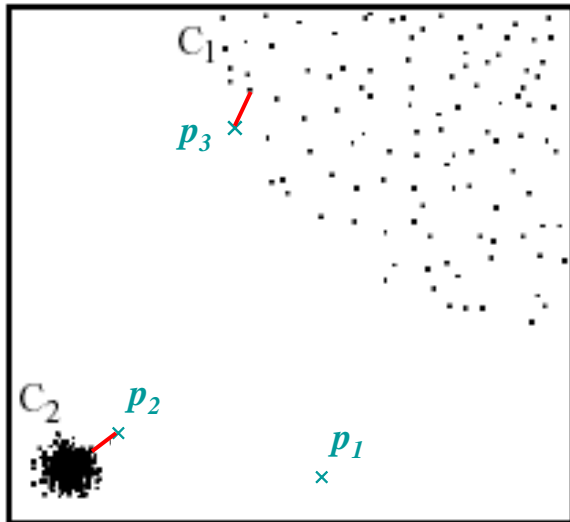
- Для каждой точки  $q$  считается расстояние до  $k$ -го соседа ( $k$ -distance)
- Считается достижимое расстояние ( $reach-dist$ ) для каждого  $q$  по образцу  $p$ :  
 $reach-dist(q, p) = \max\{k-distance(p), d(q, p)\}$
- Считается локальная достижимая плотность *local reachability density* ( $lrd$ ) для  $q$  как среднее достижимое расстояние по  $MinPts$  соседям

$$lrd_{MinPts}(x) = \left( \frac{\sum_{y \in N_{MinPts}(x)} reach-dist_{MinPts}(x, y)}{|N_{MinPts}(x)|} \right)$$

- Считается  $LOF(q)$  как отношение:

$$LOF_{MinPts}(x) = \frac{\sum_{y \in N_{MinPts}(x)} \frac{lrd_{MinPts}(y)}{lrd_{MinPts}(x)}}{|N_{MinPts}(x)|}$$

# Свойства LOF (пример)



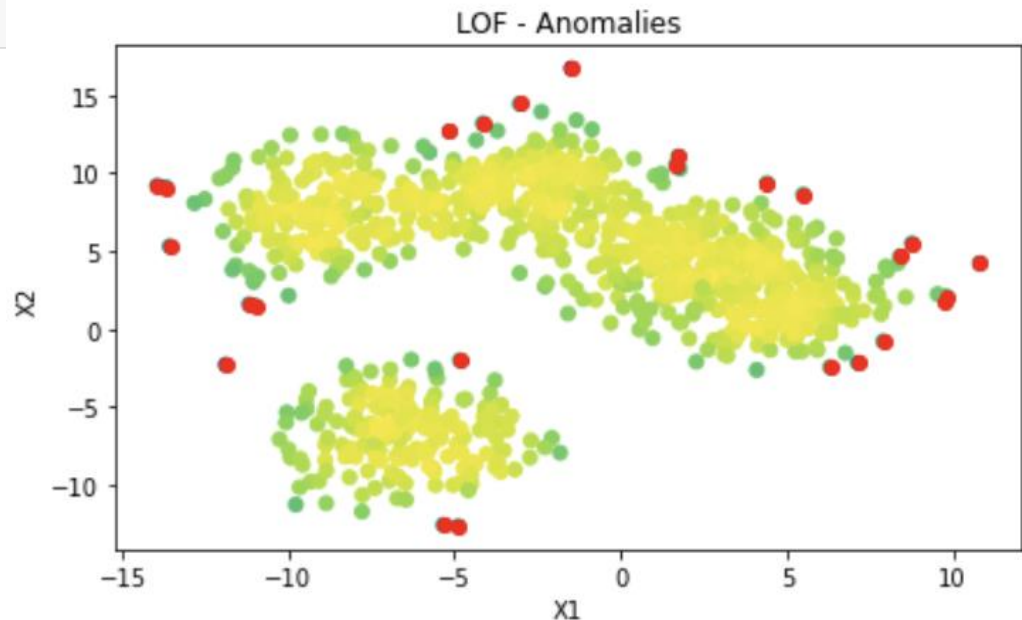
- В глобальном метрическом подходе  $p_2$  не исключение (много соседей, но на пределе расстояния), а в *LOF* и  $p_1$ , и  $p_2$  исключения
- В глобальном подходе  $p_3$  может быть исключением, а в *LOF* - нет

# Демо пример

```
from sklearn.neighbors import LocalOutlierFactor

lof = LocalOutlierFactor(n_neighbors=15, novelty=False)
clf = lof.fit(X_one)
if_outlier = lof.fit_predict(X_one)
X_anom = X_one[if_outlier==-1]

plt.figure(figsize = (7, 4))
plt.scatter(X_one[:, 0], X_one[:, 1], c=clf.negative_outlier_factor_)
plt.scatter(X_anom[:, 0], X_anom[:,1], c='red')
plt.xlabel('X1')
plt.ylabel('X2')
plt.title('LOF - Anomalies')
```



# Connectivity Outlier Factor (COF)

## ■ Процедура:

- В рамках  $k$ -окрестности объекта  $x$  строится последовательность вложенных подмножеств, отличающихся на один объект:

$$\{x\} = G_1 \subset G_2 \subset G_3 \subset \dots \subset G_m = N_k(x)$$

где  $|G_i| = i$

т.е. поочередно добавляется объект, ближайший к уже добавленным (single-link или Complete-link)

- Таким образом формируется последовательность расстояний:

$$\{0, dist(G_1, x_2), \dots, dist(G_{i-1}, x_i), \dots, dist(G_{m-1}, x_m)\} = \{d_m\}$$

- И определяется среднее связывающее расстояние (average chaining distance) :

$$ac - dist_G(x) = \frac{1}{m-1} \sum_{i=2}^m \frac{2(m+1-i)}{m} * d_i$$

- Оно учитывает структуру объектов и расстояний между ними внутри множества, соседей - чем дальше объект отстоит от  $x$ , тем меньше его вклад в значение функции

# Connectivity Outlier Factor (COF)

- Точка  $p$  – исключение, если среднее связывающее расстояние (chaining distance)  $ac-dist_{kNN(p)}(p)$  больше среднего связывающего расстояния ( $ac-dist$ ) среди всех  $k$  ближайших соседей  $p$   $kNN(p)$
- Вычисляется мера связности:

$$COF_k(x) = \frac{|N_k(x)| ac-dist_{N_k(x)}(x)}{\sum_{y \notin N_k(x)} ac-dist_{N_k(y)}(y)}$$

- Доказывается, что COF и LOF в определенных случаях дают близкие результаты, а также, что они имеют квадратичную сложность

# Методы на основе кластеризации

## ■ Основная идея:

- Нормальные данные лежат в больших кластерах с высокой плотностью, остальные - аномалии

## ■ Общая процедура:

- Кластеризация
- Расчет характеристик кластеров (диаметр, среднее внутрикластерное расстояние и т.п.)
- Поиск аномалий

## ■ Аномалии:

- Объекты вне кластеров, маленькие кластеры, кластеры с низкой плотностью

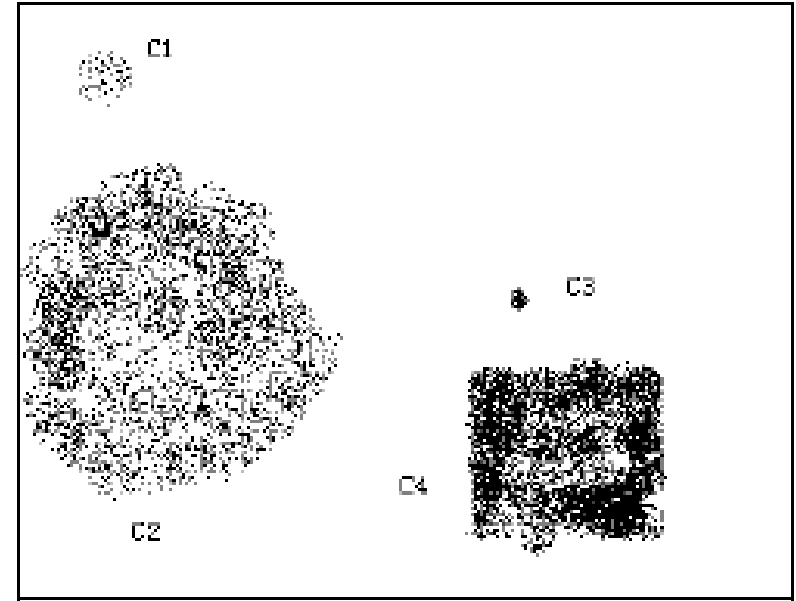
## ■ Недостатки:

- Вычислительно сложны, зависят от метода кластеризации, не всегда можно выделить кластеры, а аномалии искать нужно



# Cluster based Local Outlier Factor (CBLOF)

- CBLOF считается для каждой точки с учетом размера кластера и расстояний:
  - Если точка в маленьком кластере, то CBLOF есть произведение размера кластера на расстояние до центроида ближайшего большого кластера
  - Если точка лежит в большом кластере, то CBLOF - произведение размера кластера на расстояние до центроида этого кластера

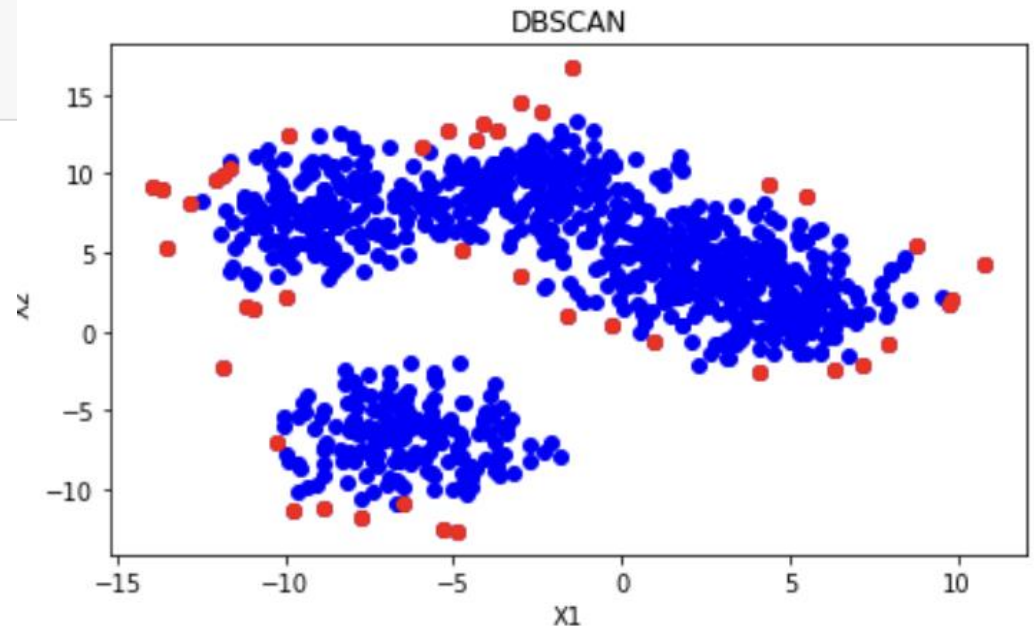


# Демо пример

```
from sklearn.cluster import DBSCAN

dbscn = DBSCAN(eps=1, min_samples=5)
dbscn.fit(X_one)
X_anom = X_one[dbscn.labels_!=-1]

plt.figure(figsize = (7, 4))
plt.scatter(X_one[:, 0], X_one[:, 1], c='blue')
plt.scatter(X_anom[:, 0], X_anom[:,1], c='red')
plt.xlabel('X1')
plt.ylabel('X2')
plt.title('DBSCAN')
```



# Свойства методов ближайших соседей

## ■ Достоинства:

- Простые в реализации
- Достаточно интуитивно понятные
- Не нужно априорных знаний

## ■ Недостатки:

- Вычислительно сложны
- В задачах с большой размерностью и разреженной матрицей данных – проблема расстояний
- Сложно адекватно определить расстояние для сложных разнородных данных
- «Критические» параметры надо задавать априори

# Статистические методы

## ■ Основная идея:

- Строится вероятностная модель (параметрическая или нет), описывающая вероятностное распределение данных
- Может строится модель только нормальных данных или смесь нормальных и аномалий (обычно требуется знать пропорцию аномалий)
- Для каждой точки оценивается правдоподобие, что она сгенерирована данной вероятностной моделью или отношение правдоподобий нормального и аномального распределений

## ■ Достоинства:

- Аппарат статистики и теории вероятностей

## ■ Проблемы

- Для большой размерности тяжело оценить распределения и параметры

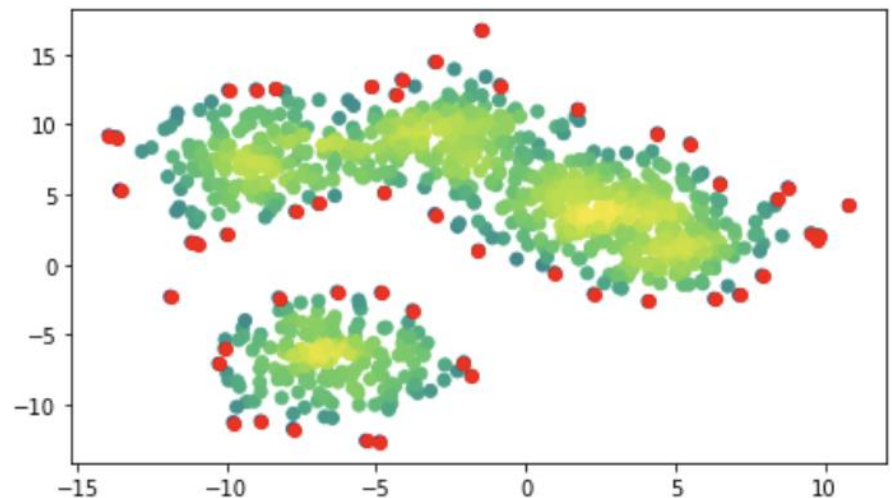
# Демо пример

```
from sklearn.mixture import GaussianMixture

fig, axes = plt.subplots(ncols=1, figsize=(7,4))
gmm_0 = GaussianMixture(n_components=25)
gmm_0.fit(X_one)

E=gmm_0.score_samples(X_one)

X_anom=X_one[E<pd.DataFrame(E).quantile(q=0.05)[0]]
plt.scatter(X_one[:, 0], X_one[:, 1], c=E)
plt.scatter(X_anom[:, 0], X_anom[:, 1], c="red")
```



# Оценка плотности распределения с помощью ядерных функций

- Оценка плотности распределения:

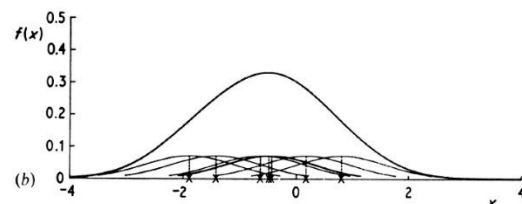
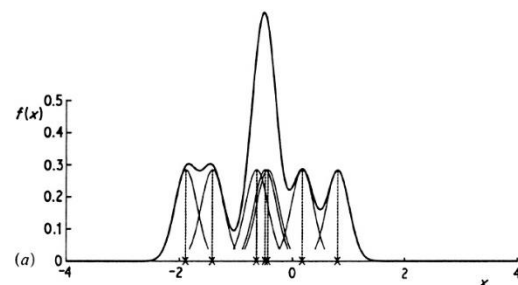
$$\hat{f}_h(x) = \frac{1}{n} \sum_{i=1}^n K_h(x - x_i) = \frac{1}{nh} \sum_{i=1}^n K\left(\frac{x - x_i}{h}\right),$$

- где  $K$ - ядерная функция (положительно-определенная функция), такая что  $K(x)=K(-x)$  и:

$$\int_{-\infty}^{\infty} K(x)dx = 1.$$

- например  $K(u) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}u^2}$
- $x_i$  – значения из выборки,
- $h$  – параметр сглаживания, автоматический выбор  $h$  по MISE

$$\text{MISE}(\lambda) = \int_x \{E(\hat{f}_\lambda(x)) - f(x)\}^2 dx + \int_x \text{var}(\hat{f}_\lambda(x)) dx$$

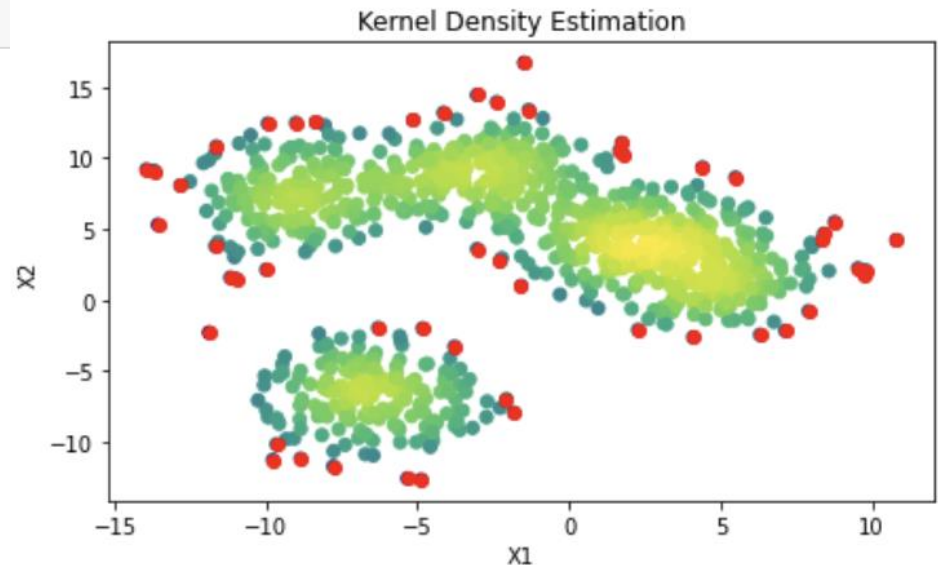


# Демо пример

```
from sklearn.neighbors import KernelDensity
kde = KernelDensity(kernel='gaussian', bandwidth=1).fit(X_one)
scr=kde.score_samples(X_one)

X_anom=X_one[scr<pd.DataFrame(scr).quantile(q=0.05)[0]]

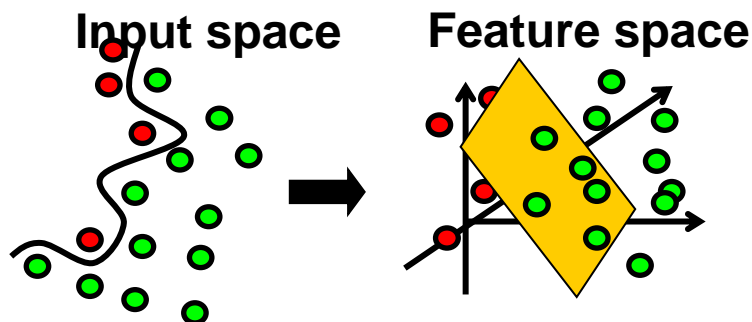
plt.figure(figsize = (7, 4))
plt.scatter(X_one[:, 0], X_one[:, 1], c=scr)
plt.scatter(X_anom[:, 0], X_anom[:,1], c='red')
plt.xlabel('X1')
plt.ylabel('X2')
plt.title('Kernel Density Estimation')
```



# Непараметрические методы на основе ядерных функций

- **Основная идея** – отобразить признаковое пространство  $X$  в индуцированное гильбертово пространство большой размерности  $H$  (RKHS) с образами  $\varphi(x)$  и  $\varphi(y)$ , связанными с наблюдениями  $x$  и  $y$  через замену скалярного произведения потенциальной функцией (ядром)  $K(x, y) = \langle \varphi(x), \varphi(y) \rangle_H$ ,

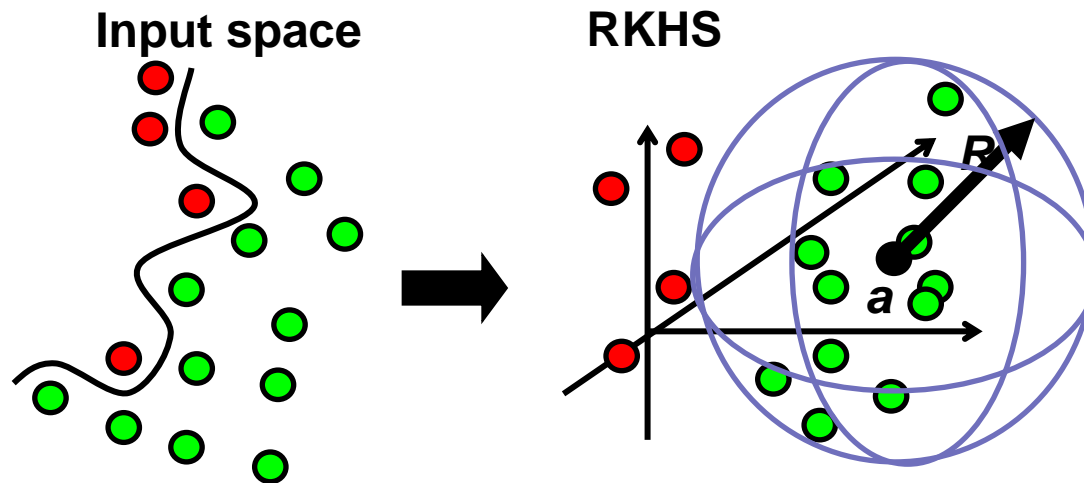
**Цель отображения** - дать возможность использовать в RKHS более простые геометрические структуры для описания основных зависимостей .



- **Популярные алгоритмы:**
  - Support vector clustering (Single Class SVM), Kernel PCA.



# Support vector clustering



- С помощью ядра (обычно гуссовского) строится RKHS, ширина ядра влияет на «гладкость» границ областей при обратном отображении
- В RKHS ищется гиперсфера с центром в  $a$  и минимальным радиусом  $R$ , содержащую не менее  $1 - \nu$  часть образов всех наблюдений
- Аномалии – наблюдения, чьи образы вне гиперсферы, чем дальше, тем аномальнее
- Обратное отображение границы задает в пространстве признаков контуры плотности распределения с заданным порогом

# Обнаружение аномалий с помощью SVM

- Формулировка задачи оптимизации:

$$\min_{\xi \in \mathbb{R}^m, R \in \mathbb{R}, a \in H} \left[ R^2 + \frac{1}{\nu N} \sum_{i=1}^N \xi_i \right]$$
$$\text{subject to } \|\varphi(x_i) - a\|^2 \leq R^2 + \xi_i, \forall i \in [1, N]$$

- Решающая функция:

$$f(z) = \text{sgn} \left( R^2 - \sum_{i,j=1}^N \beta_i \beta_j K(x_i, x_j) + 2 \sum_{i=1}^N \beta_i K(x_i, z) - K(z, z) \right),$$

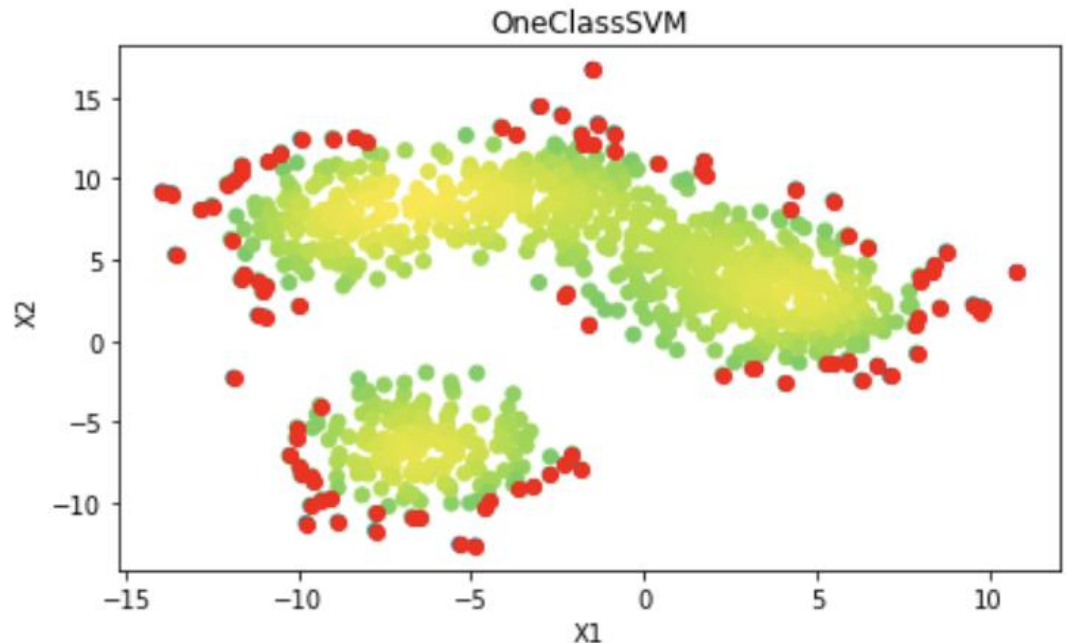
где  $\beta_i$  — множители Лагранжа,  $\beta_i = \frac{1}{\nu N}$  для выбросов,  $0 < \beta_i < \frac{1}{\nu N}$  для наблюдений на границе и  $\beta_i = 0$  для точек внутри сферы,  $z$  — проверяемое наблюдений

# Демо пример

```
from sklearn.svm import OneClassSVM

osvm = OneClassSVM(nu=0.1).fit(X_one)
X_anom = X_one[osvm.predict(X_one)==-1]

plt.figure(figsize = (7, 4))
plt.scatter(X_one[:, 0], X_one[:, 1], c=osvm.score_samples(X_one))
plt.scatter(X_anom[:, 0], X_anom[:, 1], c='red')
plt.xlabel('X1')
plt.ylabel('X2')
plt.title('OneClassSVM')
```



# Kernel PCA

- Проекция наблюдений на выбранные  $k$  главных нелинейных компонент в RKHS.
- Проекция образа наблюдения  $\varphi(x)$  на  $k$ th главную компоненту  $V^k$ :

$$V^{kT} \varphi(x) = \left( \sum_{i=1}^N a_i^l \varphi(x_i) \right)^T \varphi(x)$$

- $\alpha$  собственные вектора центрированной матрицы ядер  $\tilde{K}$ :

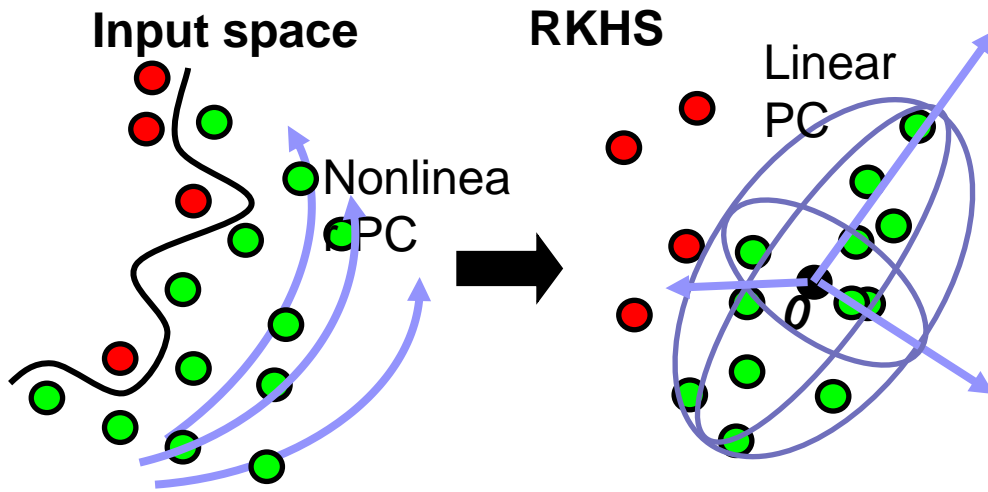
$$N\lambda\alpha = \tilde{K},$$

$$\text{где } \tilde{K} = K - 1_N K - K 1_N + 1_N K 1_N;$$

$\lambda$  с.зн.  $N$  – размер выборки.

- В результате в RKHS строится гиперэллипсоид (не гиперсфера) с фиксированным центром (т.к. матрица центрирована), содержащий основную часть образов наблюдений

# Kernel PCA



- $a=0$  центр  
центрированного  
гиперэллипсоида в  
RKHS
- главные компоненты  
линейны в RKHS и  
нелинейны в  
пространстве  
признаков

**Ошибка реконструкции** – уровень аномальности наблюдения  $z$ :

$$err(z) = K(z, z) - \frac{2}{N} \sum_{i=1}^N K(z, x_i) + \frac{1}{N^2} \sum_{i,j=1}^N K(x_i, x_j) - \sum_{l=1}^q \left( \sum_{i=1}^N a_i^l (K(z, x_i) - \frac{1}{N} \sum_{r=1}^N K(x_i, x_r) - \frac{1}{N} \sum_{r=1}^N K(z, x_r) + \frac{1}{N^2} \sum_{r,s=1}^N K(x_r, x_s)) \right)^2,$$

где  $N$  размер выборки;  $a_i^l$  коэф. КРСА;  $q$  – число главных компонент

# Демо пример

```
from sklearn.decomposition import KernelPCA

kernel_pca = KernelPCA(n_components=2, kernel="cosine",
                       fit_inverse_transform=True)
kernel_pca.fit(X_one)
X_one_pca = kernel_pca.transform(X_one)

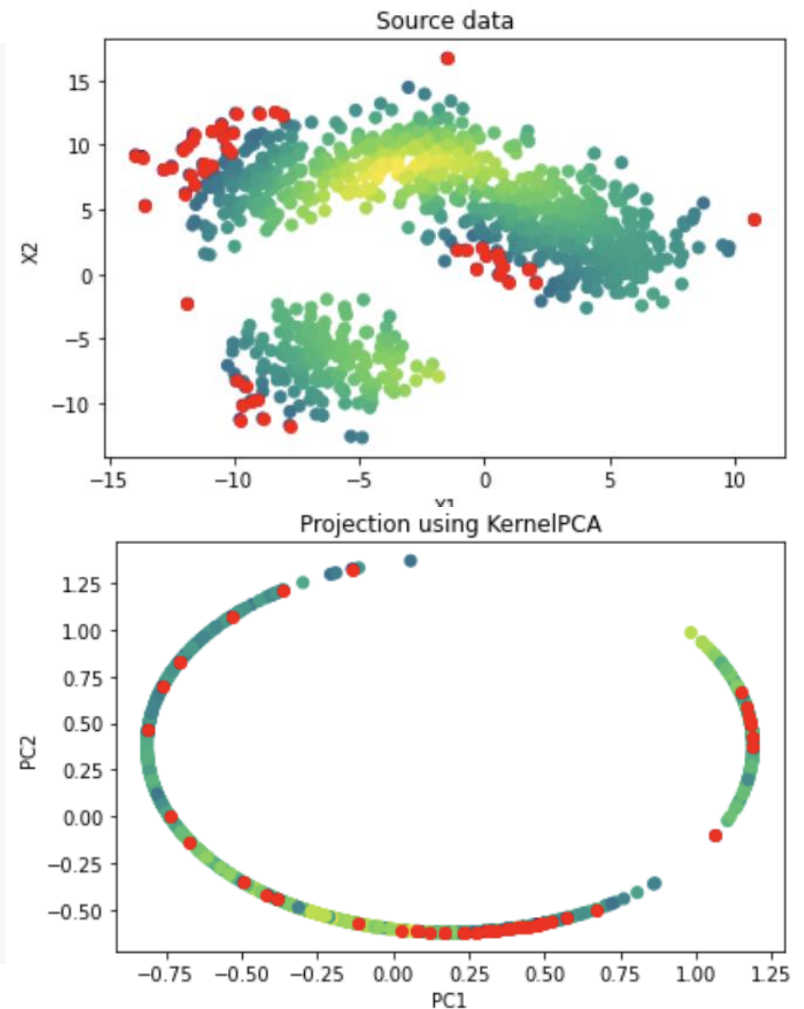
fig, axes = plt.subplots(ncols=2, figsize=(14, 4))

#reconstruction error
E=abs(X_one-kernel_pca.inverse_transform(X_one_pca))

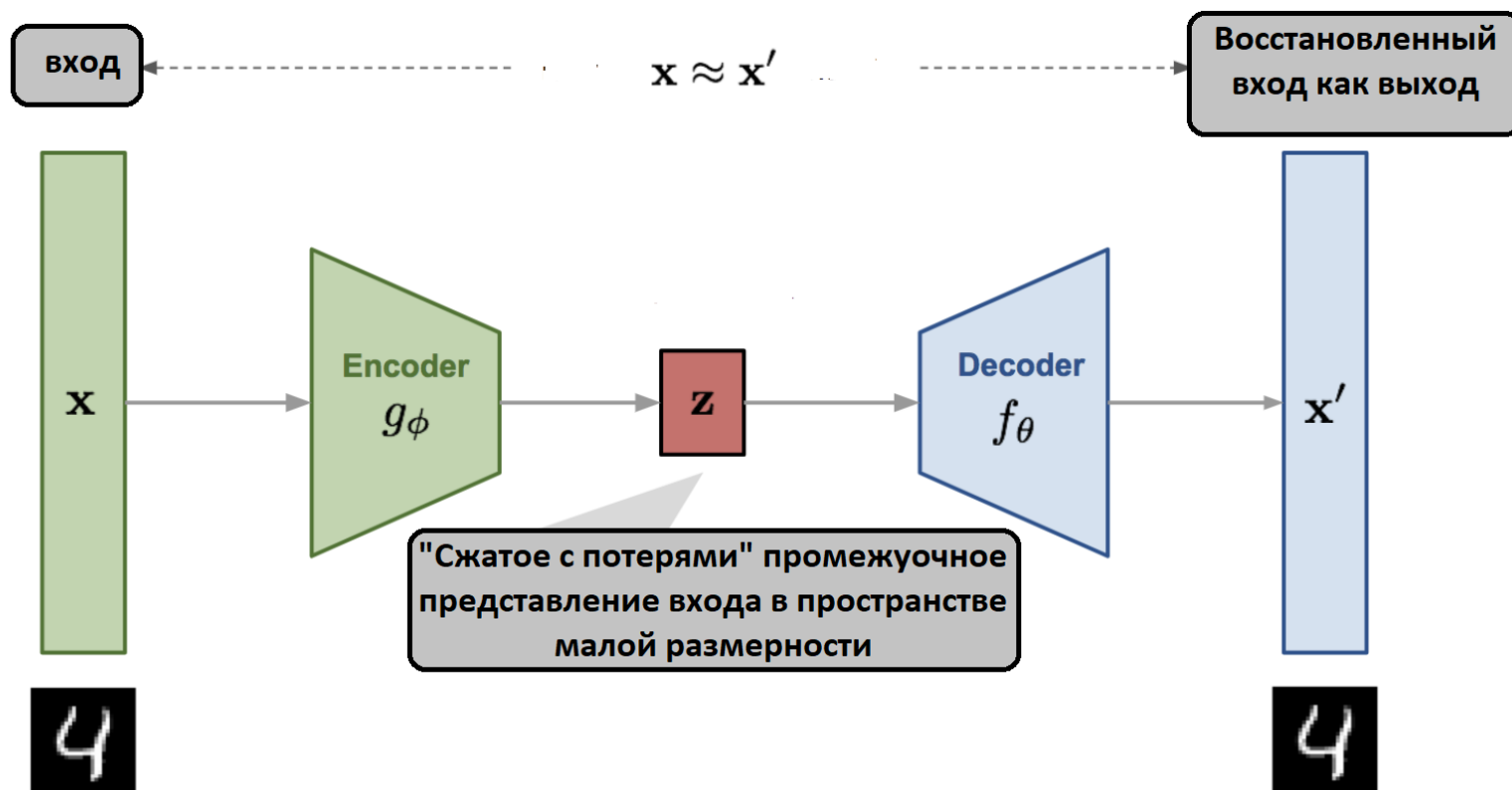
ET=np.apply_along_axis(np.linalg.norm, 1, E)
X_anom=X_one[ET>pd.DataFrame(ET).quantile(q=0.95)[0]]
X_anom_pca=X_one_pca[ET>pd.DataFrame(ET).quantile(q=0.9)[0]]

axes[0].scatter(X_one[:, 0], X_one[:, 1], c=ET)
axes[0].scatter(X_anom[:, 0], X_anom[:, 1], c="red")
axes[0].set_xlabel("X1")
axes[0].set_ylabel("X2")
axes[0].set_title("Source data")

axes[1].scatter(X_one_pca[:, 0], X_one_pca[:, 1], c=ET)
axes[1].scatter(X_anom_pca[:, 0], X_anom_pca[:, 1], c="red")
axes[1].set_xlabel("PC1")
axes[1].set_ylabel("PC2")
axes[1].set_title("Projection using KernelPCA")
```



# Автоэнкодер



- Архитектура типа «песочных часов» (иногда «вывернутые», тогда embedding)
- Обучается минимизируя ошибку восстановления на тренировочном наборе
- Оценка аномальности – ошибка восстановления  $\|x - x'\|$

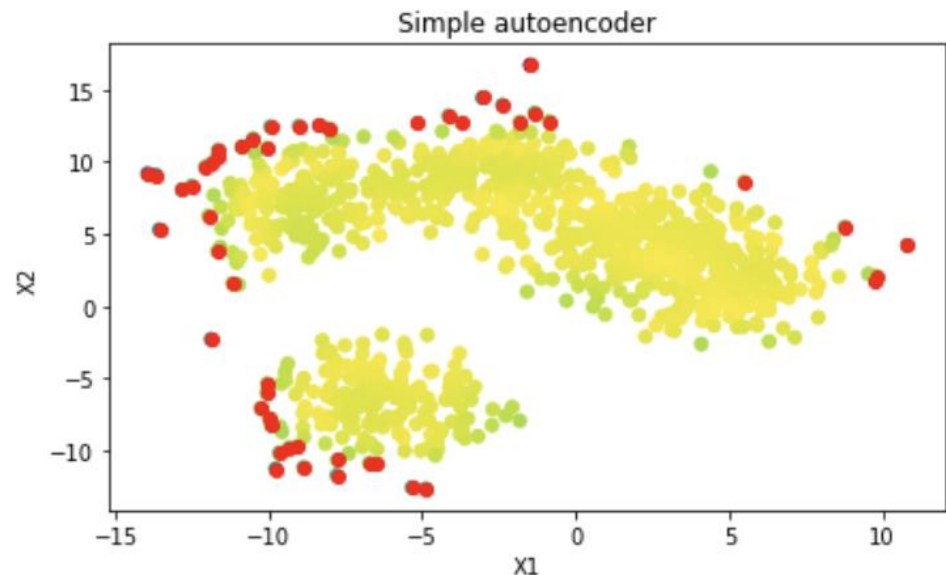
# Демо пример

```
from sklearn.neural_network import MLPRegressor
model = MLPRegressor(hidden_layer_sizes=(30,), max_iter=1000, alpha=0.01, activation="tanh")
model.fit(X_one, X_one)

E=abs(X_one-model.predict(X_one))

ET=np.apply_along_axis(np.linalg.norm, 1, E)
X_anom=X_one[ET>pd.DataFrame(ET).quantile(q=0.95)[0]]

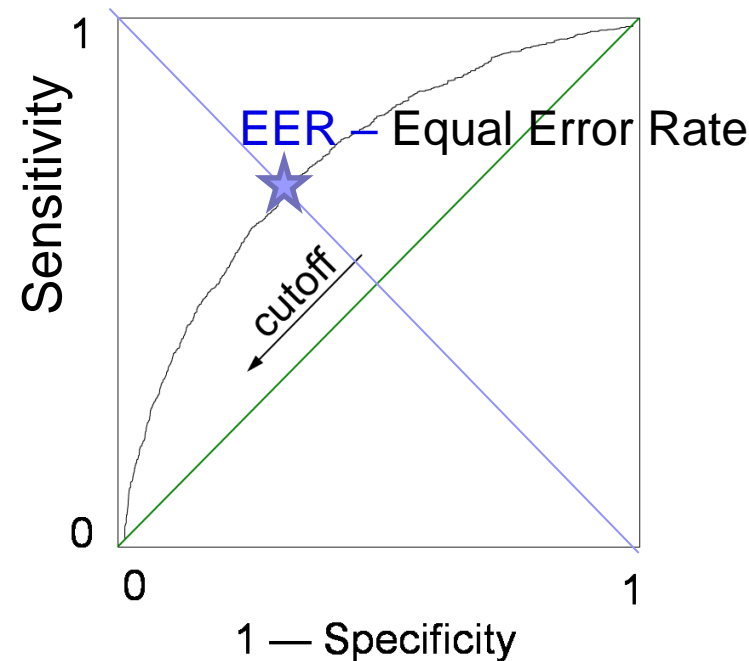
plt.figure(figsize = (7, 4))
plt.scatter(X_one[:, 0], X_one[:, 1], c=-ET)
plt.scatter(X_anom[:, 0], X_anom[:,1], c='red')
plt.xlabel('X1')
plt.ylabel('X2')
plt.title('Simple autoencoder')
```





# Оценка модели

		Predicted Class		
		0	1	
Actual Class	0	<b>True Negative</b>	<b>False Positive</b>	Actual Negative
	1	<b>False Negative</b>	<b>True Positive</b>	Actual Positive
		Predicted Negative	Predicted Positive	



**SENSITIVITY** (true positive rate (TPR),  
hit rate, recall)  
$$TPR = TP / (TP + FN)$$

**SPECIFICITY** (SPC) (true negative  
rate (TNR))  
$$SPC = TN / (FP + TN)$$

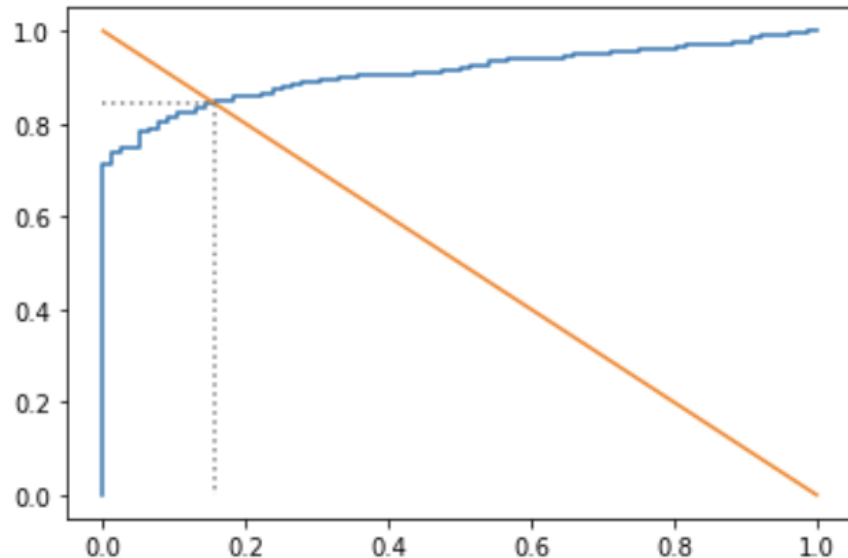
# Пример

```
from sklearn.ensemble import IsolationForest
from sklearn.metrics import roc_curve

isof = IsolationForest(contamination=0.01, random_state=0)
isof.fit(df_e.drop(columns="label"))
X_Y = isof.score_samples(df_e.drop(columns="label"))

rc=roc_curve(df_e["label"], X_Y, pos_label=6)

fpr, tpr, threshold = rc
fnr = 1 - tpr
eer_threshold = threshold[np.nanargmin(np.absolute((fnr - fpr)))]
EER = fpr[np.nanargmin(np.absolute((fnr - fpr)))]
```



```
plt.plot(tpr, fpr)
plt.plot([0,1],[1,0])
plt.plot([1-EER,1-EER],[EER,0],linestyle='dotted',color="grey")
plt.plot([0,1-EER],[EER,EER],linestyle='dotted',color="grey")
```