

## Лекция 3

### Линейные модели классификации. ROC-анализ

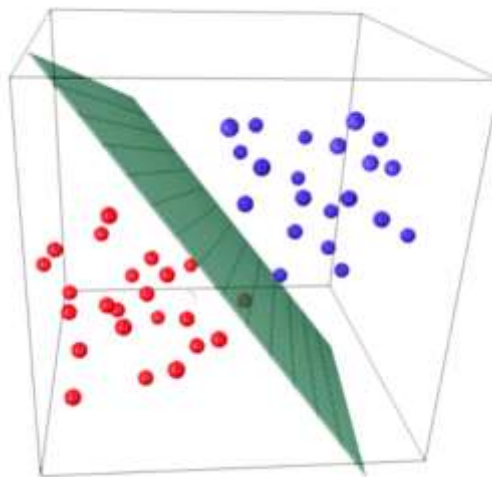
- Линейный классификатор
- Логистическая регрессия
- Регуляризация
- Метод Опорных Векторов
- ROC-анализ

#### Линейный классификатор

<https://habr.com/ru/company/ods/blog/323890/>

Основная идея линейного классификатора заключается в том, что признаковое пространство может быть разделено гиперплоскостью на два полупространства, в каждом из которых прогнозируется одно из двух значений целевого класса.

Если это можно сделать без ошибок, то обучающая выборка называется линейно разделимой.



**Рассмотрим задачу бинарной классификации**, причем метки целевого класса обозначим "+1" (положительные примеры) и "-1" (отрицательные примеры).

Один из самых простых линейных классификаторов получается на основе регрессии вот таким образом:

$$a(\vec{x}) = \text{sign}(\vec{w}^T x),$$

где

- $\vec{x}$  – вектор признаков примера (вместе с единицей);
- $\vec{w}$  – веса в линейной модели (вместе со смещением  $w_0$ );
- $\text{sign}(\bullet)$  – функция "сигнум", возвращающая знак своего аргумента;
- $a(\vec{x})$  – ответ классификатора на примере  $\vec{x}$ .

Для линейных моделей классификации граница принятия решений (decision boundary) является линейной функцией аргумента. Другими словами, (бинарный) линейный классификатор – это классификатор, который разделяет два класса с помощью линии, плоскости или гиперплоскости.

Существует масса алгоритмов обучения линейных моделей.

Двумя наиболее распространенными алгоритмами линейной классификации являются логистическая регрессия (logistic regression) и линейный метод опорных векторов (linear support vector machines) или линейный SVM.

## Логистическая регрессия

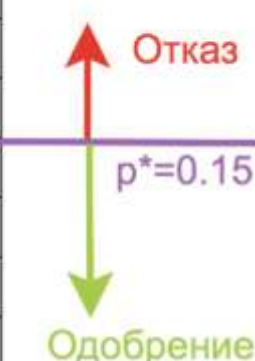
Несмотря на свое название, логистическая регрессия является алгоритмом классификации, а не алгоритмом регрессии, и его не следует путать с линейной регрессией.

Логистическая регрессия является частным случаем линейного классификатора, но она обладает хорошим "умением" – **прогнозировать вероятность**  $p_+$  отнесения примера  $\vec{x}_i$  к классу "+":

$$p_+ = P(y_i = 1 \mid \vec{x}_i, \vec{w})$$

Прогнозирование не просто ответа ("1" или "-1"), а именно вероятности отнесения к классу "+1" во многих задачах является очень важным бизнес-требованием. Например, в задаче *кредитного скоринга*, где традиционно применяется логистическая регрессия, часто прогнозируют вероятность невозврата кредита ( $p_+$ ). Клиентов, обратившихся за кредитом, сортируют по этой предсказанной вероятности (по убыванию), и получается скоркарта — по сути, рейтинг клиентов от плохих к хорошим.

Клиент	Вероятность невозврата
Mike	0.78
Jack	0.45
Larry	0.13
Kate	0.06
William	0.03
Jessica	0.02

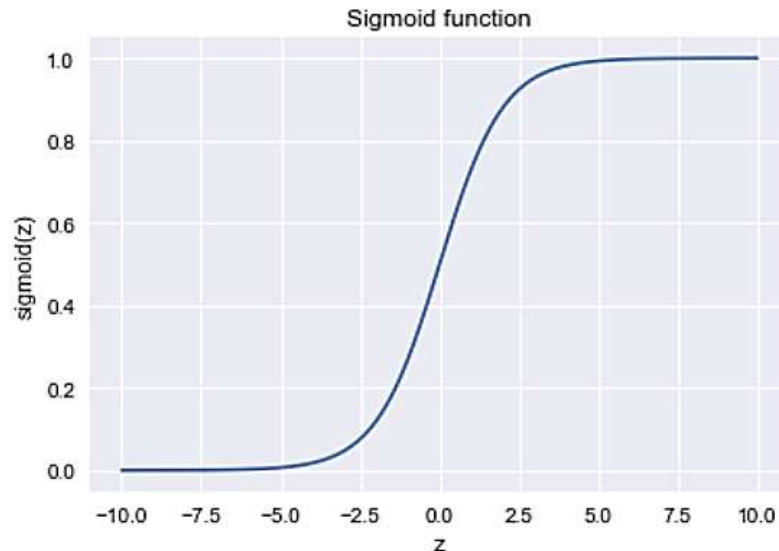


Пример скоркарты: банк выбирает для себя порог  $p^*$  предсказанной вероятности невозврата кредита (в примере – 0.15) и начиная с этого значения кредит не выдает.

Итак, есть задача - прогнозировать вероятность  $p_+ \in [0,1]$ .

Очевидно, для этого нужна некоторая функция  $f: \mathbb{R} \rightarrow [0,1]$ . В модели логистической регрессии для этого берется конкретная функция (сигмоида):

$$\sigma(z) = \frac{1}{1 + \exp^{-z}}$$



Обозначим  $P(X)$  вероятностью происходящего события  $X$ . Тогда отношение вероятностей  $OR(X)$  определяется из  $\frac{P(X)}{1 - P(X)}$ , а это — отношение вероятностей того, произойдет ли событие или не произойдет. Очевидно, что вероятность и отношение шансов содержат одинаковую информацию. Но в то время как  $P(X)$  находится в пределах от 0 до 1,  $OR(X)$  находится в пределах от 0 до  $\infty$ .

Если вычислить логарифм  $OR(X)$  (логарифм шансов, или логарифм отношения вероятностей), то легко заметить, что  $\log OR(X)$  находится в пределах от  $-\infty$  до  $\infty$ . Его-то мы и будем прогнозировать.

Рассмотрим, как логистическая регрессия будет делать прогноз

$$p_+ = P(y_i = 1 \mid \vec{x}_i, \vec{w})$$

, предположим, что веса  $w_i$  известны:

**Шаг 1.** Вычислить значение  $w_0 + w_1x_1 + w_2x_2 + \dots = \vec{w}^T \vec{x}$ . (уравнение  $\vec{w}^T \vec{x} = 0$  задает гиперплоскость, разделяющую примеры на 2 класса);

**Шаг 2.** Вычислить логарифм отношения шансов:  $\log(OR_+) = \vec{w}^T \vec{x}$ .

**Шаг 3.** Имея прогноз шансов на отнесение к классу "+" —  $OR_+$ , вычислить  $p_+$  с помощью простой зависимости:

$$p_+ = \frac{OR_+}{1 + OR_+} = \frac{\exp^{\vec{w}^T \vec{x}}}{1 + \exp^{\vec{w}^T \vec{x}}} = \frac{1}{1 + \exp^{-\vec{w}^T \vec{x}}} = \sigma(\vec{w}^T \vec{x})$$

Итак, логистическая регрессия прогнозирует вероятность отнесения примера к классу "+" (при условии, что мы знаем его признаки и веса модели)

как сигмоид-преобразование линейной комбинации вектора весов модели и вектора признаков примера:

$$p_+(x_i) = P(y_i = 1 \mid \vec{x}_i, \vec{w}) = \sigma(\vec{w}^T \vec{x}_i)$$

Обучить модель, это подобрать такие коэффициенты, при которых логистическая функция потерь будет минимальной.

$$\mathcal{L}_{\text{log}}(X, \vec{y}, \vec{w}) = \sum_{i=1}^{\ell} \log(1 + \exp^{-y_i \vec{w}^T \vec{x}_i})$$

$\mathcal{L}$  - Это логистическая функция потерь, просуммированная по всем объектам обучающей выборки.

Решение: МНК или метод градиентного спуска

## Реализация логистической регрессии в scikit-learn

`sklearn.linear_model.LogisticRegression`

`class sklearn.linear_model.LogisticRegression(penalty='l2', *, dual=False, tol=0.0001, C=1.0, fit_intercept=True, intercept_scaling=1, class_weight=None, random_state=None, solver='lbfgs', max_iter=100, multi_class='auto', verbose=0, warm_start=False, n_jobs=None, l1_ratio=None)`

По умолчанию применяется **регуляризация**.

## Регуляризация

Регуляризация в статистике, машинном обучении— метод добавления некоторых дополнительных ограничений к условию с целью решить некорректно поставленную задачу или предотвратить переобучение. **Эта информация часто имеет вид штрафа за сложность модели.**

Переобучение в большинстве случаев проявляется в том, что в получающихся многочленах слишком большие коэффициенты. Соответственно, необходимо добавить в целевую функцию штраф за слишком большие коэффициенты.

Некоторые виды регуляризации:

- $L_1$ -регуляризация (англ. *lasso regression*), или регуляризация

$$L_1 = \sum_i (y_i - y(t_i))^2 + \lambda \sum_i |a_i|.$$

- $L_2$ -регуляризация, или регуляризация Тихонова (в англоязычных уравнениях позволяет балансировать между соответствием дан

$$L_2 = \sum_i (y_i - y(t_i))^2 + \lambda \sum_i a_i^2.$$

# Метод Опорных Векторов

<https://habr.com/ru/post/428503/>

**Метод Опорных Векторов** или **SVM** (от англ. Support Vector Machines) — это линейный алгоритм, используемый в задачах классификации и регрессии. **Метод Опорных Векторов относится к методам обучения с учителем.** Данный метод изначально относится к бинарным классификаторам, хотя существуют способы заставить его работать и для задач мультиклассификации. Данный алгоритм имеет широкое применение на практике. **Суть работы Метода Опорных Векторов: алгоритм создает линию или гиперплоскость, которая разделяет данные на классы.**

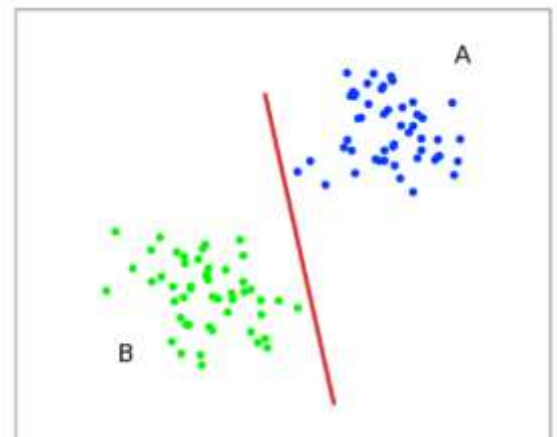
Рассмотрим решение задачи классификации, математическая формулировка которой такова: пусть  $X$  — пространство объектов (например,  $R^n$ ),  $Y$  — наши классы (например,  $Y = \{-1, 1\}$ ). При обучении алгоритм должен построить функцию  $F(x)=y$ , которая принимает в себя аргумент  $x$  — объект из пространства  $R^n$  и выдает метку класса  $y$ .

Идею метода удобно проиллюстрировать на следующем простом примере:

Даны точки на плоскости, разбитые на два класса.

Проведем линию, разделяющую эти два класса (красная линия). Далее, все новые точки (не из обучающей выборки) автоматически классифицируются следующим образом:

точка правее прямой попадает в класс А,  
точка левее прямой — в класс В.



Такую прямую назовем разделяющей прямой. Однако, в пространствах высоких размерностей прямая уже не будет разделять наши классы, так как понятие «правее прямой» или «левее прямой» теряет всякий смысл. Поэтому вместо прямых необходимо рассматривать гиперплоскости — пространства, размерность которых на единицу меньше, чем размерность исходного пространства. Например, в трехмерном пространстве гиперплоскость — это обычная двумерная плоскость.

## Формальное определение гиперплоскости

Гиперплоскость — это  $n-1$  мерная подплоскость в  $n$ -мерном Евклидовом пространстве, которая разделяет пространство на две отдельные части. Например, представим, что наша линия представлена в виде одномерного Евклидова пространства (т.е. наш набор данных лежит на прямой). Выберите точку на этой линии. Эта точка разделит набор данных, в нашем случае линию, на две части. У линии есть одна мера, а у точки 0 мер. Следовательно, точка — это гиперплоскость линии.

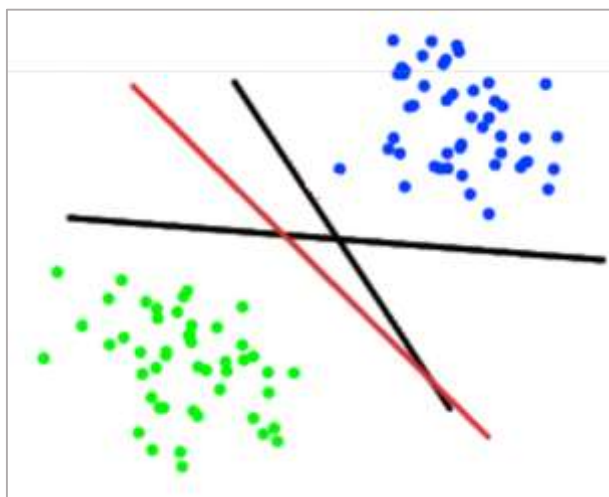
Для двумерного датасета разделяющей гиперплоскостью является прямая. Проще говоря, для  $n$ -мерного пространства существует  $n-1$  мерная гиперплоскость, разделяющая это пространство на две части.



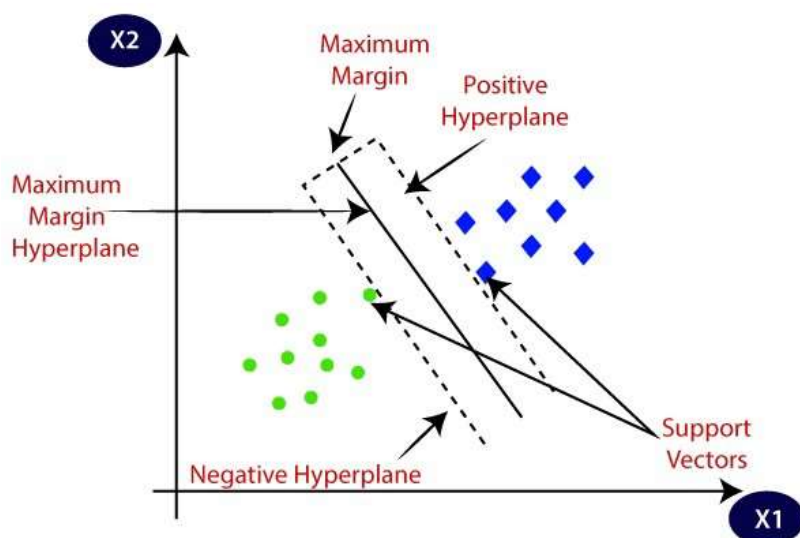
Однако в нашем примере существует несколько прямых, разделяющих два класса:

### Как SVM находит лучшую линию

Алгоритм SVM устроен таким образом, что он ищет точки, которые расположены ближе всех к разделяющей гиперплоскости. Эти точки называются **опорными векторами (support vectors)** (на рисунке обведены красным).



Затем, алгоритм вычисляет расстояние между опорными векторами и разделяющей плоскостью. Это расстояние называется **зазором**. Основная цель алгоритма — **максимизировать расстояние зазора**. Лучшей гиперплоскостью считается такая гиперплоскость, для которой этот зазор является максимально большим, поскольку, как правило, чем больше зазор, тем ниже ошибка обобщения классификатора. Такая гиперплоскость, называется **оптимальной разделяющей гиперплоскостью**.

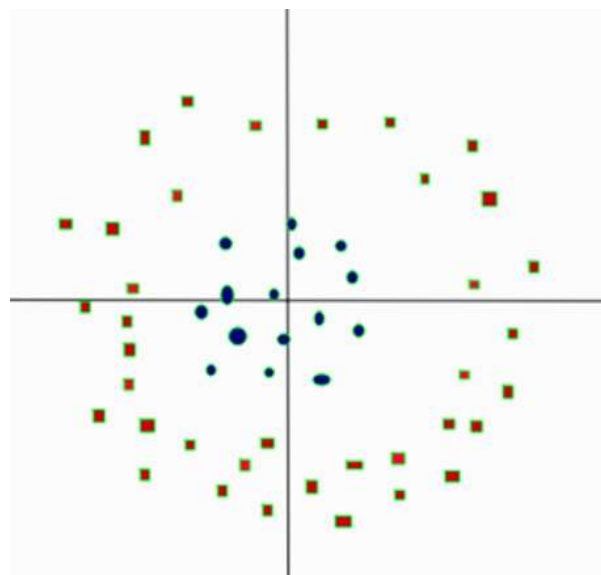


Рассмотрим пример, с более сложным датасетом, который нельзя разделить линейно.

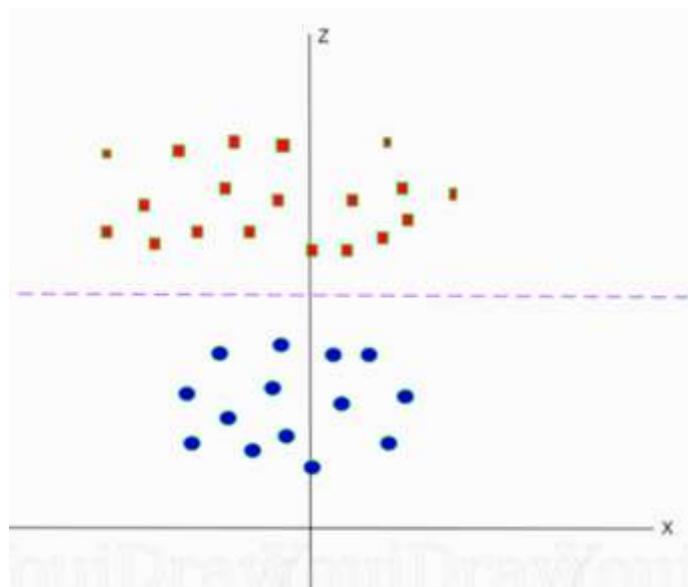
Здесь невозможно начертить прямую линию, которая бы классифицировала эти данные. Но этот датасет можно разделить линейно, добавив дополнительное измерение, которое мы назовем осью Z. Представим, что координаты на оси Z регулируются следующим ограничением:

$$z = x^2 + y^2$$

Таким образом, ордината Z представлена из квадрата расстояния точки до начала оси.



Ниже приведена визуализация того же набора данных, на оси Z.



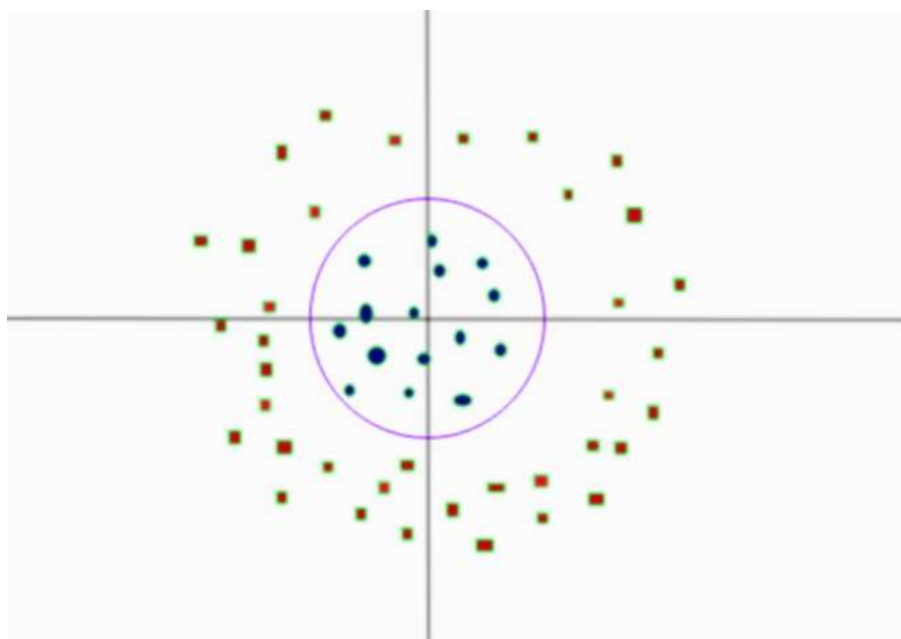
Теперь данные можно разделить линейно. Допустим линия разделяющая данные  $z=k$ , где  $k$  константа. Если

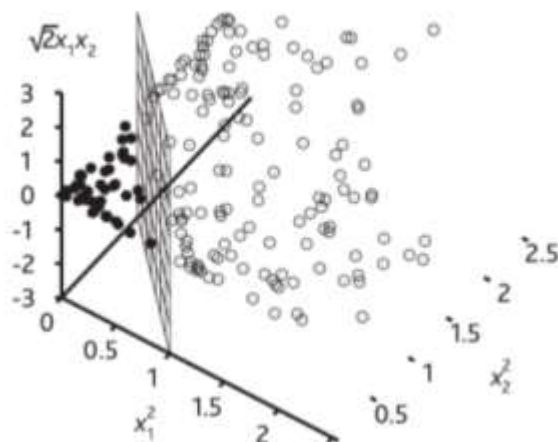
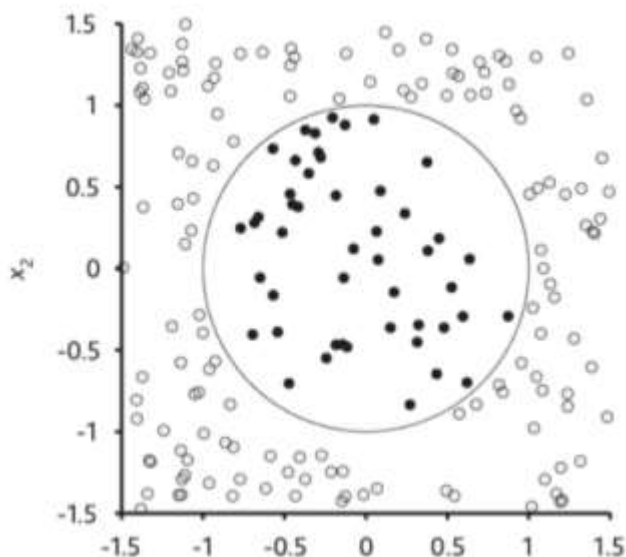
$$z = x^2 + y^2,$$

то, следовательно, и

$$k = x^2 + y^2 \text{ — формула окружности.}$$

Таким образом, можно спроецировать линейный разделитель, обратно к исходному количеству измерений выборки, используя эту трансформацию. В итоге, **можно классифицировать нелинейный набор данных добавив к нему дополнительное измерение**, а затем, привести обратно к исходному виду используя математическую трансформацию.





Итак, если выборка объектов с признаковым описанием из  $X=\mathbb{R}^n$  не является линейно разделимой, можно предположить, что существует некоторое пространство  $H$ , вероятно большей размерности, при переходе в которое выборка станет линейно разделимой. Пространство  $H$  называют **спрямляющим**.

Подобные подходы к изменению признакового пространства называют **ядровыми методами**, в которых используют повышение размерность пространства при помощи ядер.

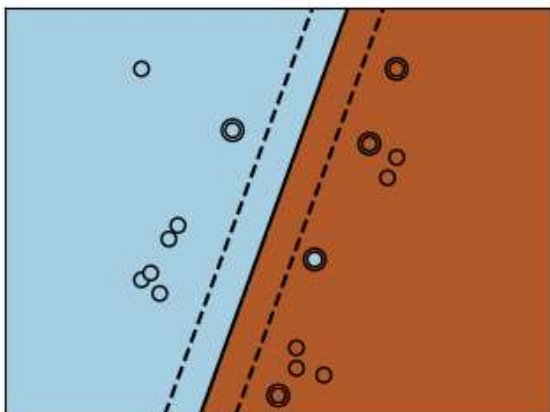
**Ядром** называют функцию  $K(x, z)$ , представленную в виде скалярного произведения в некотором пространстве:  $K(x, z) = \langle \varphi(x), \varphi(z) \rangle$ ,

где где  $\varphi: X \rightarrow H$  — отображение из исходного признакового пространства в некоторое спрямляющее пространство.

**Примеры влияния разных ядер на вид оптимальной разделяющей гиперплоскости:**

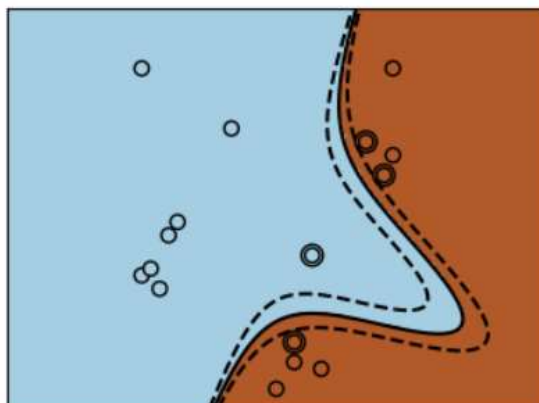
#### Linear kernel

```
>>> svc = svm.SVC(kernel='linear')
```



#### Полиномиальное ядро

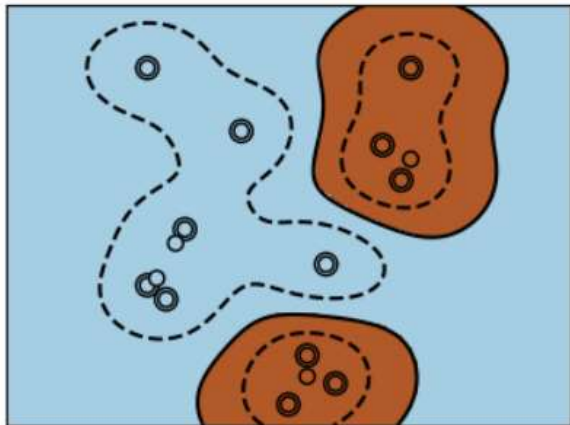
```
>>> svc = svm.SVC(kernel='poly',  
...               degree=3)  
>>> # degree: polynomial degree
```





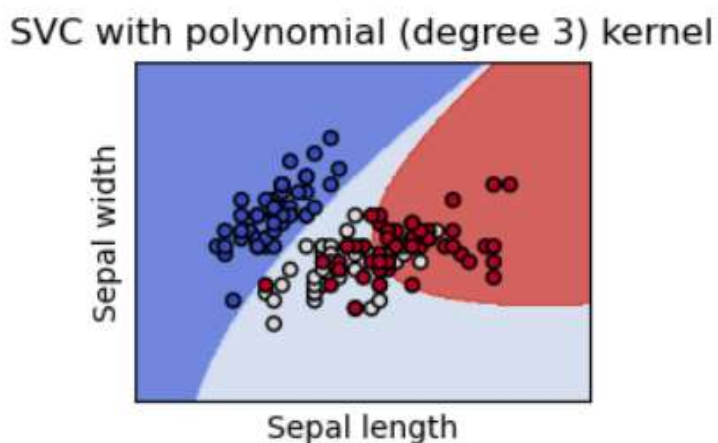
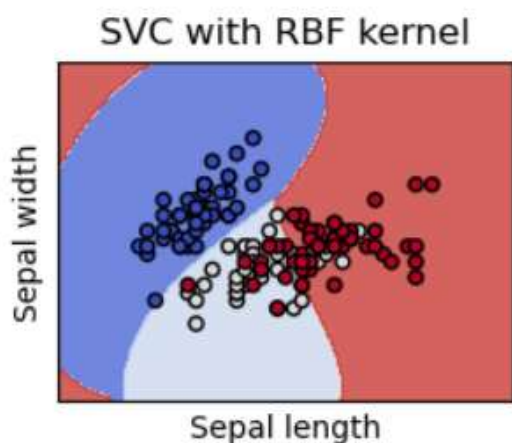
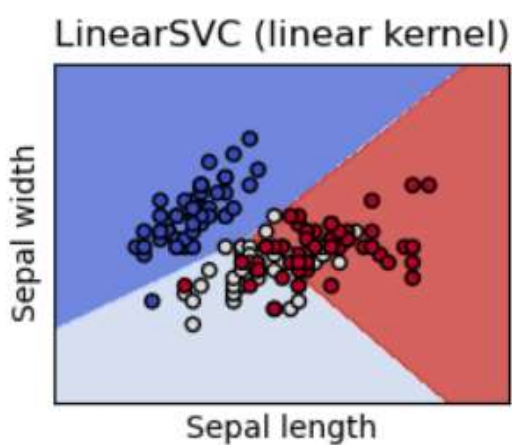
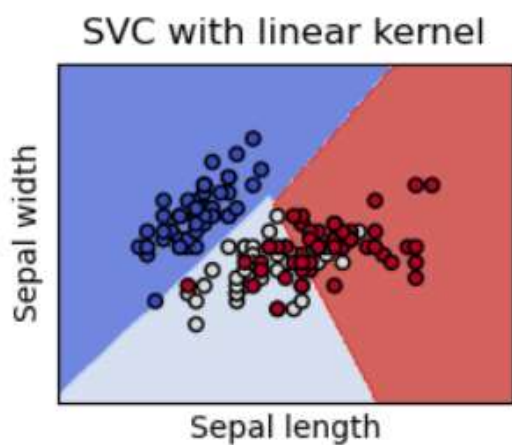
## Ядро RBF (радиальная базисная функция)

```
>>> svc = svm.SVC(kernel='rbf')  
>>> # gamma: inverse of size of  
>>> # radial kernel
```



[https://scikit-learn.org/stable/auto\\_examples/svm/plot\\_svm\\_kernels.html#sphx-glr-auto-examples-svm-plot-svm-kernels-py](https://scikit-learn.org/stable/auto_examples/svm/plot_svm_kernels.html#sphx-glr-auto-examples-svm-plot-svm-kernels-py)

Для многоклассовой классификации результат будет таким:



<https://scikit-learn.org/stable/modules/svm.html#tips-on-practical-use>

Класс **LinearSVC** лучше работает с большими выборками в отличие от класса **SVC**, использует линейное ядро по умолчанию.

**Пример:** Рассчитаем модели SVC с разными ядрами на датасете «диабет», результаты неоднозначные:

```
# instantiate the model (using the default parameters)
model_SVC = SVC(kernel='poly', degree=3)

# fit the model with data
model_SVC.fit(X_train, y_train)

model_SVC.score(X_train, y_train)
```

0.7743055555555556



```
# calculate the predicted values
y_pred = model_SVC.predict(X_test)
```

```
# import the metrics class
from sklearn import metrics
```

**Вычислим confusion matrix (матрицу ошибок)**

```
cnf_matrix = metrics.confusion_matrix(y_test, y_pred)
cnf_matrix
```

```
array([[120, 10],
       [ 33, 29]], dtype=int64)
```



В таблице представлены результаты оценки моделей SVC с разными ядрами, (score и confusion matrix) :

Линейное ядро	Полиномиальное степень 3	RBФядро
0.765625	0.7743	0.7638
[117, 13] [ 25, 37]	[120, 10] [ 33, 29]	[119, 11] [ 32, 30]

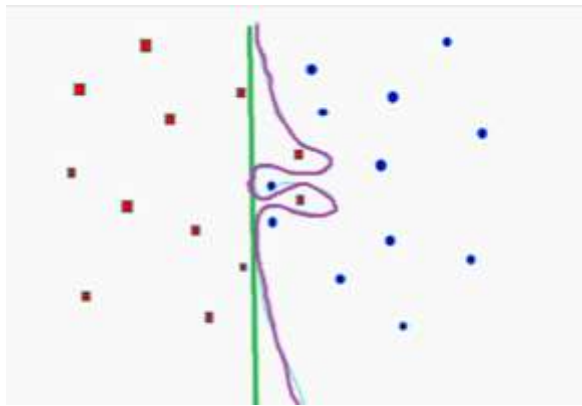
## Реализация в sklearn

```
class sklearn.svm.SVC(*, C=1.0, kernel='rbf', degree=3, gamma='scale', coef0=0.0, shrinking=True, probability=False, tol=0.001, cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape='ovr', break_ties=False, random_state=None)
```

### Настройка отдельных параметров

В данном примере есть несколько порогов принятия решений, которые можно определить для этой конкретной выборки. Например, если в качестве порога решений использовать прямую, то несколько объектов классифицируются неверно. Эти неверно классифицированные точки называются выбросами данных.

Если настроить параметры таким образом, что в конечном итоге получим более изогнутую линию, то модель явно получается переученной (точно классифицирует все данные обучающей выборки, однако не сможет показать столь же хорошие результаты на новых данных).



### Параметр C

**C : float, default=1.0**

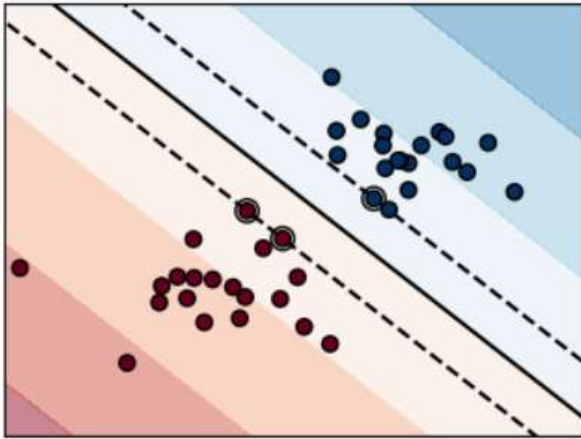
Regularization parameter. The strength of the regularization is inversely proportional to C. Must be strictly positive. The penalty is a squared l2 penalty.

**Параметр C помогает отрегулировать ту тонкую грань между гладкостью и точностью классификации объектов обучающей выборки.** Чем больше значение C, тем больше объектов обучающей выборки будут правильно классифицированы.

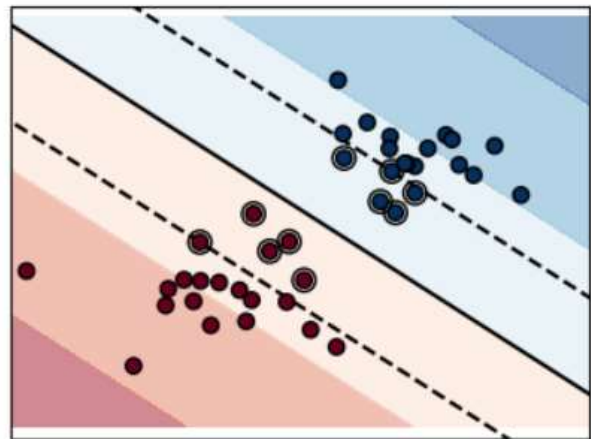
Чем выше число **C** тем более запутанная гиперплоскость будет в вашей модели, но и выше число верно-классифицированных объектов обучающей выборки. Поэтому, важно “подкручивать” параметры модели под конкретный набор данных, чтобы избежать переобучения но, в то же время достигнуть высокой точности.

Иными словами, параметр C задает степень регуляризации: небольшое значение C означает, что зазор рассчитывается с использованием многих или всех наблюдений вокруг разделительной линии (большая регуляризация); большое значение для C означает, что запас рассчитывается для наблюдений, близких к разделительной линии (меньше регуляризации).

Нерегулярная SVM



Регуляризованная SVM (по умолчанию)



## Параметр Гамма

**gamma** : {'scale', 'auto'} or float, default='scale'

Kernel coefficient for 'rbf', 'poly' and 'sigmoid'.

- if `gamma='scale'` (default) is passed then it uses  $1 / (n\_features * X.var())$  as value of gamma,
- if 'auto', uses  $1 / n\_features$ .

Чем ниже гамма, тем больше элементов, даже тех, которые достаточно далеки от разделяющей гиперплоскости, принимают участие в процессе выбора идеальной разделяющей гиперплоскости. Если же, гамма высокая, тогда алгоритм будет “опираться” только на те элементы, которые наиболее близки к самой гиперплоскости.

Если задать уровень гаммы слишком высоким, тогда в процессе принятия решения о расположении линии будут участвовать только самые близкие к линии элементы. Это поможет игнорировать выбросы в данных. Алгоритм SVM устроен таким образом, что точки расположенные наиболее близко относительно друг друга имеют больший вес при принятии решения.

Однако при правильной настройке **C** и **gamma** можно добиться оптимального результата, который построит более линейную гиперплоскость, игнорирующую выбросы, и, следовательно, более обобщающую.

Рекомендуется использовать **GridSearchCV** с параметрами C и gamma для выбора хороших значений.

```
from sklearn.model_selection import GridSearchCV
SVC_params = {"C": [0.5, 1], "gamma": [0.2, 0.6, 1]}
SVC_grid = GridSearchCV(model_SVC, SVC_params, cv=5, n_jobs=-1)
SVC_grid.fit(X_train, y_train);
SVC_grid.best_score_, SVC_grid.best_params_
```

(0.762143928035982, {'C': 0.5, 'gamma': 0.2})

# Чувствительность и специфичность модели

Вспомним таблицу ошибок (confusion matrix), чтобы ввести понятия чувствительности и специфичности модели.

Модель	Фактически положительно	Фактически отрицательно
Положительно	TP	FP
Отрицательно	FN	TN

- TP (True Positives) — верно классифицированные положительные примеры (так называемые истинно положительные случаи).
- TN (True Negatives) — верно классифицированные отрицательные примеры (истинно отрицательные случаи).
- FN (False Negatives) — положительные примеры, классифицированные как отрицательные (ошибка I рода). Это так называемый «ложный пропуск» — когда интересующее нас событие ошибочно не обнаруживается (ложно отрицательные примеры).
- FP (False Positives) — отрицательные примеры, классифицированные как положительные (ошибка II рода). Это ложное обнаружение, т.к. при отсутствии события ошибочно выносится решение о его присутствии (ложно положительные случаи).

Ошибка I-го рода (FP): чужого приняли за своего

Ошибка II-го рода (FN): своего приняли за чужого

Что является положительным событием, а что — отрицательным, зависит от конкретной задачи. Например, если мы прогнозируем вероятность наличия заболевания, то положительным исходом будет класс «Больной пациент», отрицательным — «Здоровый пациент». И наоборот, если мы хотим определить вероятность того, что человек здоров, то положительным исходом будет класс «Здоровый пациент», и так далее.

При анализе чаще оперируют не абсолютными показателями, а относительными — долями (rates), выраженными в процентах:

- ✚ Доля истинно положительных примеров (True Positives Rate):

$$TPR = \frac{TP}{TP + FN} \cdot 100 \%$$

- ✚ Доля ложно положительных примеров (False Positives Rate):

$$FPR = \frac{FP}{TN + FP} \cdot 100 \%$$



Введем еще два определения: **чувствительность** и **специфичность модели**. Ими определяется объективная ценность любого бинарного классификатора.

**Чувствительность (Sensitivity)** — доля истинно положительных случаев:

$$S_e = TPR = \frac{TP}{TP + FN} \cdot 100\%$$

**Специфичность (Specificity)** — доля истинно отрицательных случаев, которые были правильно идентифицированы моделью:

$$S_p = \frac{TN}{TN + FP} \cdot 100\%$$

**Модель с высокой чувствительностью часто дает истинный результат при наличии положительного исхода** (обнаруживает положительные примеры). Наоборот, **модель с высокой специфичностью чаще дает истинный результат при наличии отрицательного исхода** (обнаруживает отрицательные примеры). Если рассуждать в терминах медицины — задачи диагностики заболевания, где модель классификации пациентов на больных и здоровых называется диагностическим тестом, то получится следующее:

- ✓ **Чувствительный диагностический тест проявляется в гипердиагностике** — максимальном предотвращении пропуска больных.
- ✓ **Специфичный диагностический тест диагностирует только доподлинно больных.** Это важно в случае, когда, например, лечение больного связано с серьезными побочными эффектами и гипердиагностика пациентов не желательна.

## ROC-анализ

ROC-кривая (Receiver Operator Characteristic) часто используется для представления результатов бинарной классификации в машинном обучении.

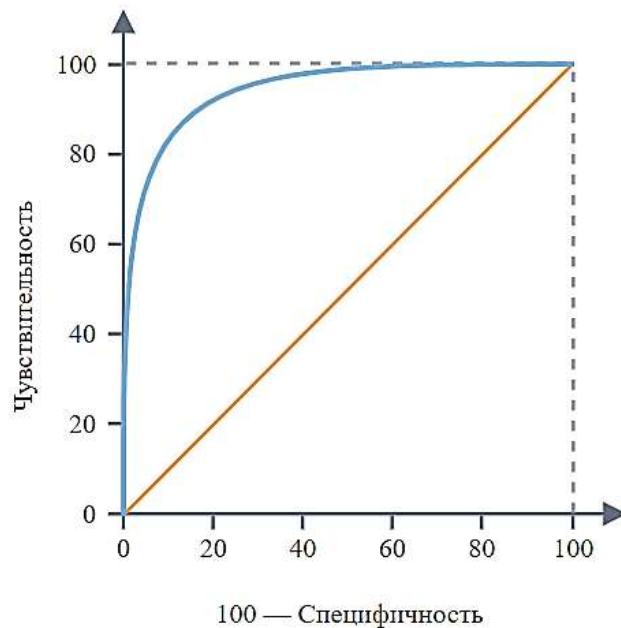
**ROC-кривая показывает зависимость количества верно классифицированных положительных примеров от количества неверно классифицированных отрицательных примеров.**

При этом предполагается, что **у классификатора имеется некоторый параметр, варьируя который, мы будем получать то или иное разбиение на два класса. Этот параметр часто называют порогом, или точкой отсечения (cut-off value).** В зависимости от него будут получаться различные величины ошибок I и II рода.

**ROC-кривая** получается следующим образом:

- ✓ Для каждого значения порога отсечения, которое меняется от 0 до 1 с шагом  $d_x$  (например, 0.01) рассчитываются значения чувствительности  $S_e$  и специфичности  $S_p$ . В качестве альтернативы порогом может являться каждое последующее значение примера в выборке.

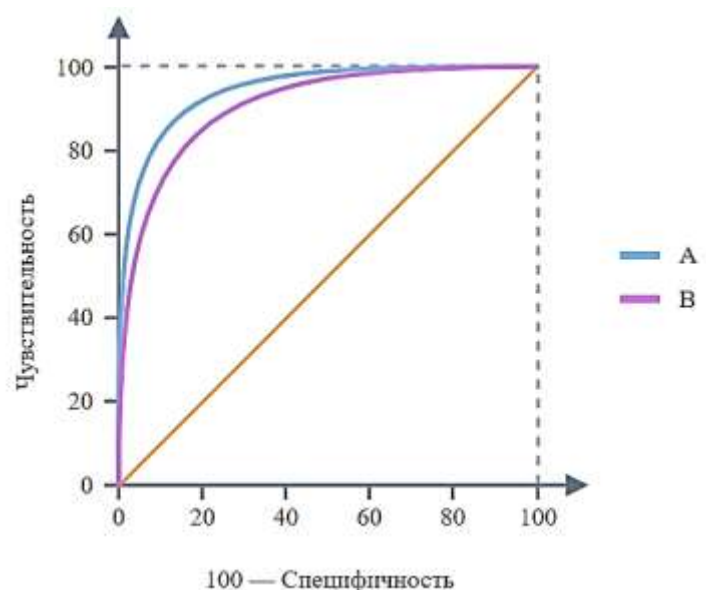
- ✓ Строится график зависимости: по оси Y откладывается чувствительность  $Se$ , по оси X —  $FPR = 100 - Sp$  — доля ложно положительных случаев.



ROC-кривая

Для идеального классификатора график ROC-кривой проходит через верхний левый угол, где доля истинно положительных случаев составляет 100% или 1,0 (идеальная чувствительность), а доля ложно положительных примеров равна нулю. Поэтому чем ближе кривая к верхнему левому углу, тем выше предсказательная способность модели. Наоборот, чем меньше изгиб кривой и чем ближе она расположена к диагональной прямой, тем менее эффективна модель. Диагональная линия ( $y=x$ ) соответствует «бесполезному» классификатору, т.е. полной неразличимости двух классов.

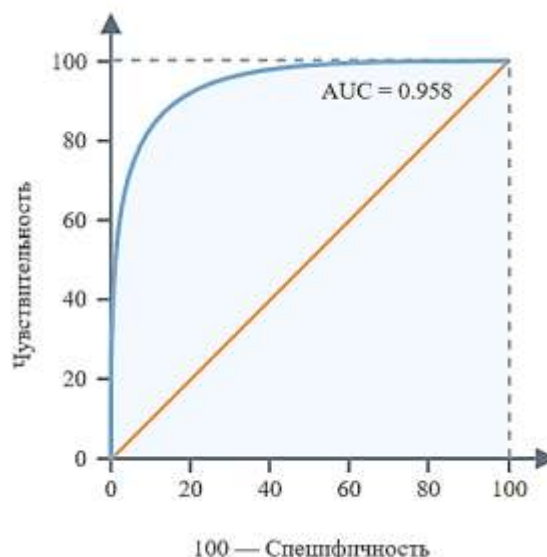
При визуальной оценке ROC-кривых расположение их относительно друг друга указывает на их сравнительную эффективность. Кривая, расположенная выше и левее, свидетельствует о большей предсказательной способности модели. Так, на рисунке видно, что модель «А» лучше.



Визуальное сравнение кривых ROC не всегда позволяет выявить наиболее эффективную модель. Своеобразным методом сравнения ROC-кривых является **оценка площади под кривыми**. Теоретически она изменяется от 0 до 1,0, но, поскольку модель всегда характеризуется кривой, расположенной выше

положительной диагонали, то обычно говорят об изменениях от 0,5 («бесполезный» классификатор) до 1,0 («идеальная» модель).

Эта оценка может быть получена непосредственно вычислением площади под многогранником, ограниченным справа и снизу осями координат и слева вверху — экспериментально полученными точками. Численный показатель площади под кривой называется **AUC** (Area Under Curve) см. рисунок.



С большими допущениями можно считать, что чем больше показатель AUC, тем лучшей прогностической силой обладает модель. Однако следует знать, что:

- показатель AUC предназначен скорее для сравнительного анализа нескольких моделей;
- AUC не содержит никакой информации о чувствительности и специфичности модели.

В литературе иногда приводится следующая экспертная шкала для значений AUCAUC, по которой можно судить о качестве модели:

Интервал AUC	Качество модели
0,9-1,0	Отличное
0,8-0,9	Очень хорошее
0,7-0,8	Хорошее
0,6-0,7	Среднее
0,5-0,6	Неудовлетворительное

**Идеальная модель обладает 100% чувствительностью и специфичностью.** Однако на практике добиться этого невозможно, более того, невозможно одновременно повысить и чувствительность, и специфичность модели. Компромисс находится с помощью порога отсека, т.к. пороговое значение влияет на соотношение Se и Sp. Можно говорить о **задаче нахождения оптимального порога отсека** (optimal cut-off value).

Порог отсека нужен для того, чтобы применять модель на практике: относить новые примеры к одному из двух классов. **Для определения оптимального порога нужно задать критерий его определения**, т.к. в разных задачах присутствует своя оптимальная стратегия. Критериями выбора порога отсека могут выступать:

- ✓ Требование минимальной величины чувствительности (специфичности) модели. Например, нужно обеспечить чувствительность теста не менее 80%.

В этом случае оптимальным порогом будет максимальная специфичность (чувствительность), которая достигается при 80%.

- ✓ Требование максимальной суммарной чувствительности и специфичности модели, т.е.

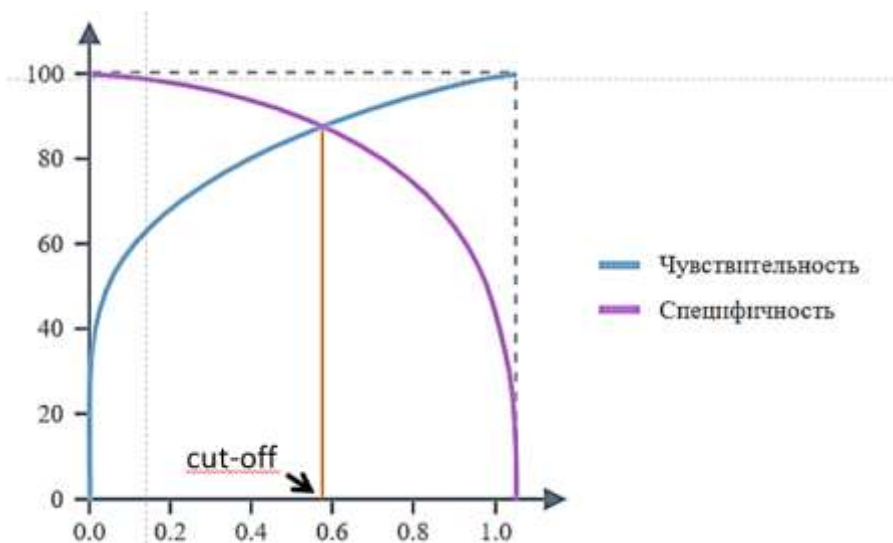
$$Cutt\_off_o = \max_k (Se_k + Sp_k)$$

В этом случае значение порога обычно предлагается пользователю по умолчанию.

- ✓ Требование баланса между чувствительностью и специфичностью, т.е. когда  $Se \approx Sp$

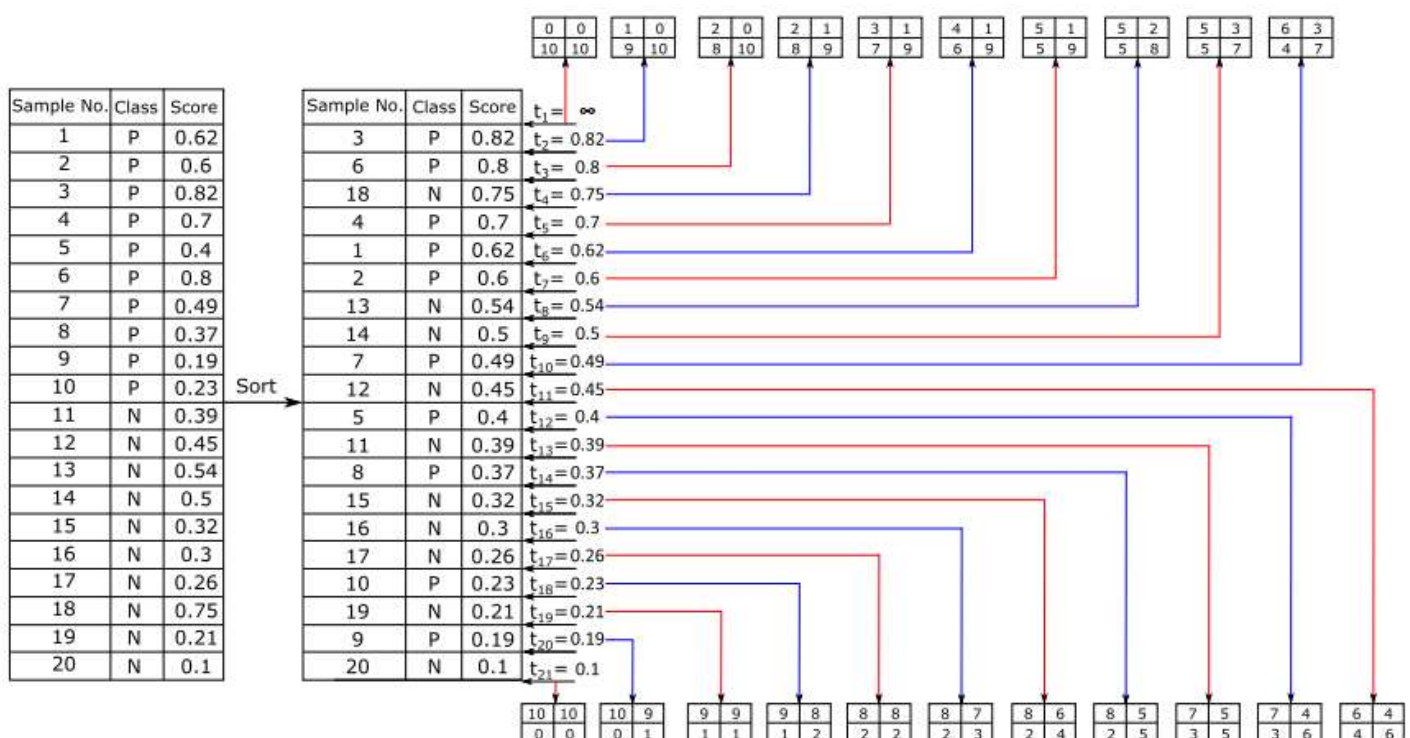
$$Cutt\_off_o = \min_k |Se_k - Sp_k|$$

В этом случае порог есть точка пересечения двух кривых, когда по оси X откладывается порог отсечения, а по оси Y — чувствительность или специфичность модели.

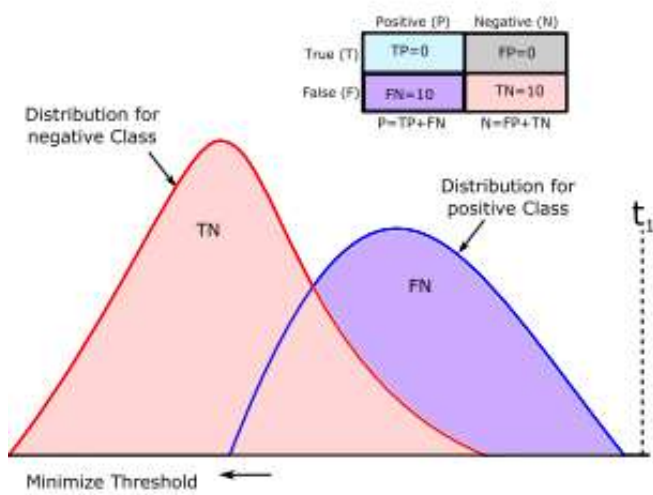


«Точка баланса» между чувствительностью и специфичностью

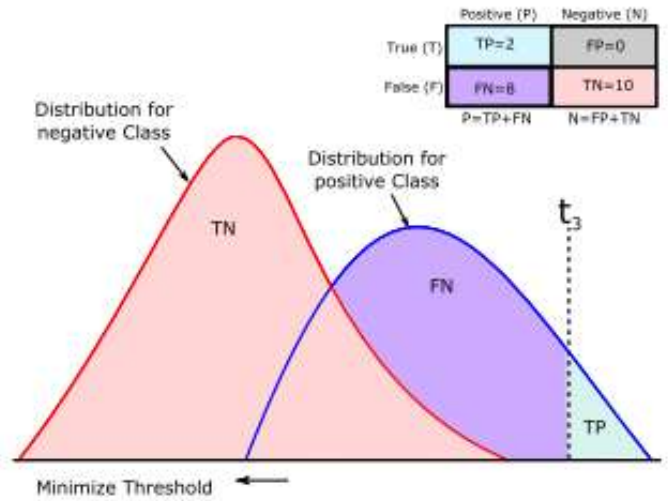
Наглядный пример расчета TPR и FPR при изменении порогового значения.



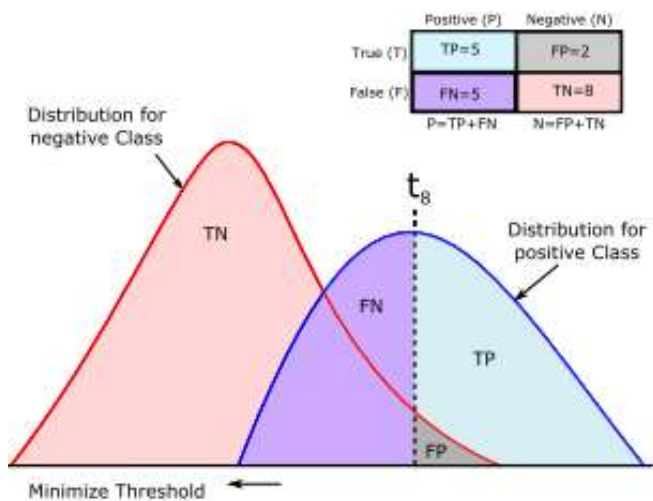




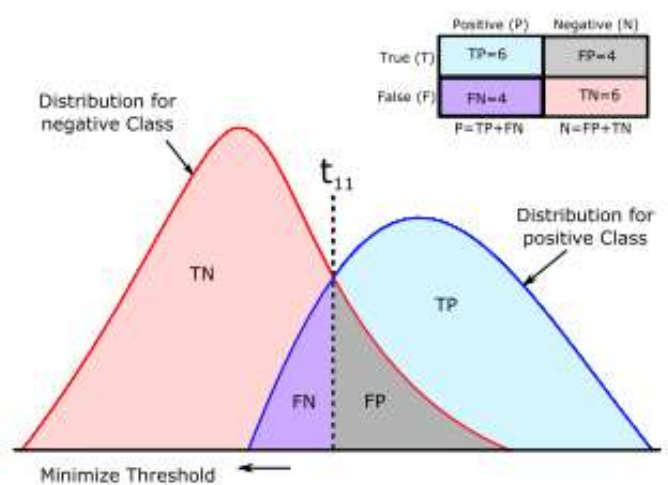
(a)



(b)



(c)



(d)

