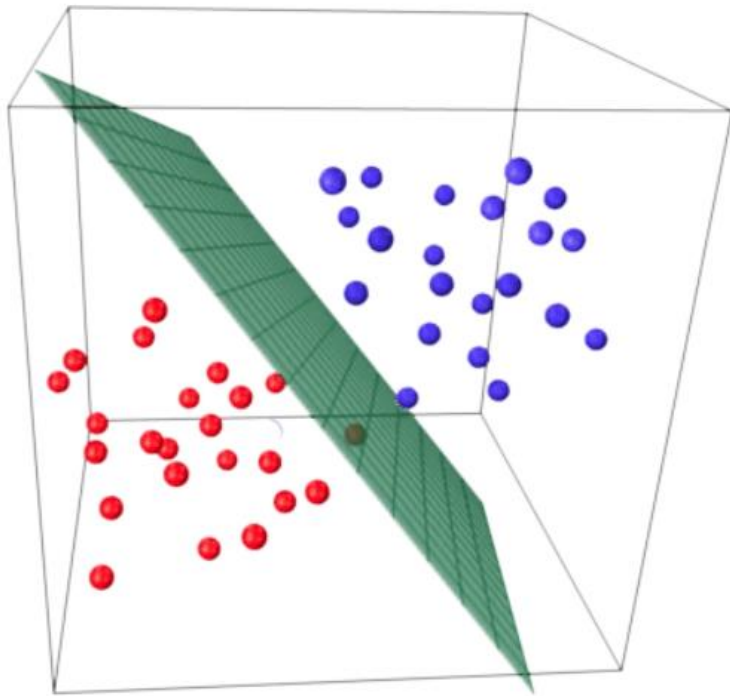


Линейные модели классификации

Линейный классификатор

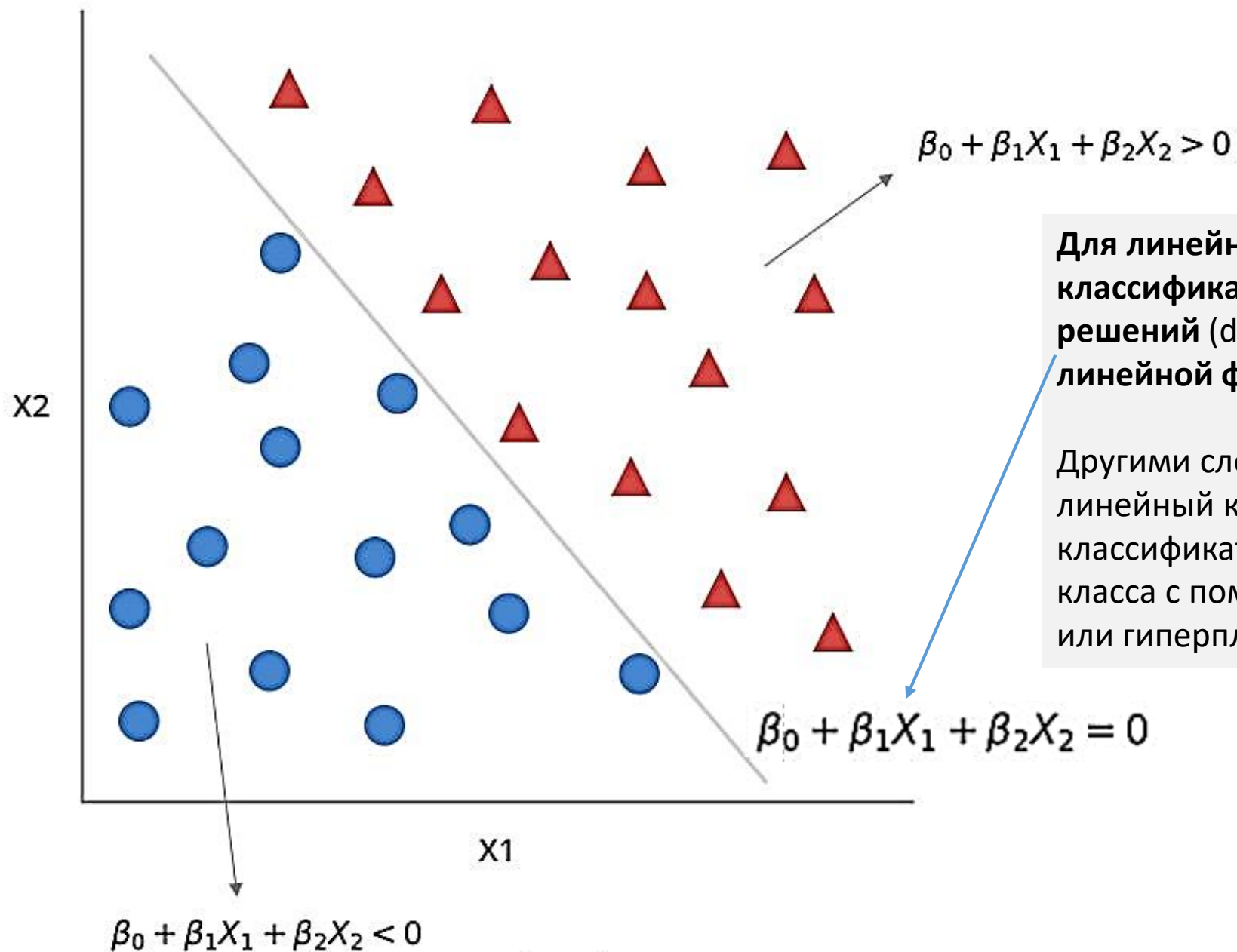
Основная идея линейного классификатора заключается в том, что признаковое пространство может быть разделено гиперплоскостью на два полупространства, в каждом из которых прогнозируется одно из двух значений целевого класса.



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

Используется линейная функция входных признаков:

$$\hat{y} = w[0] * x[0] + w[1] * x[1] + \dots + w[p] * x[p] + b > 0$$



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), реализованная в классе `linear_model.LogisticRegression` библиотеки `scikit-learn`;
- ✓ и **линейный метод опорных векторов** (linear support vector machines) или линейный SVM, реализованный в классе `svm.LinearSVC` (SVC расшифровывается как support vector classifier – классификатор опорных векторов).

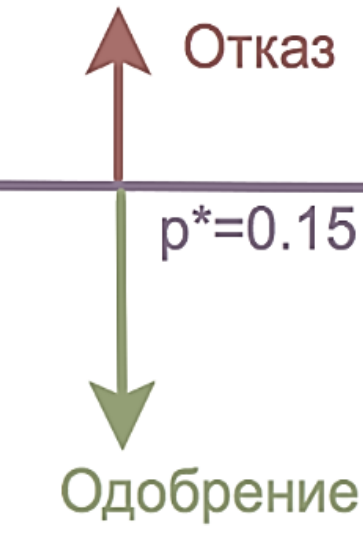
*Несмотря на свое название, логистическая регрессия является алгоритмом классификации, а не алгоритмом регрессии, и его не следует путать с линейной регрессией.

Логистическая регрессия

Логистическая регрессия является частным случаем линейного классификатора, но она обладает хорошим "умением" – прогнозировать вероятность p_+ отнесения примера X_i к классу "+":

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

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

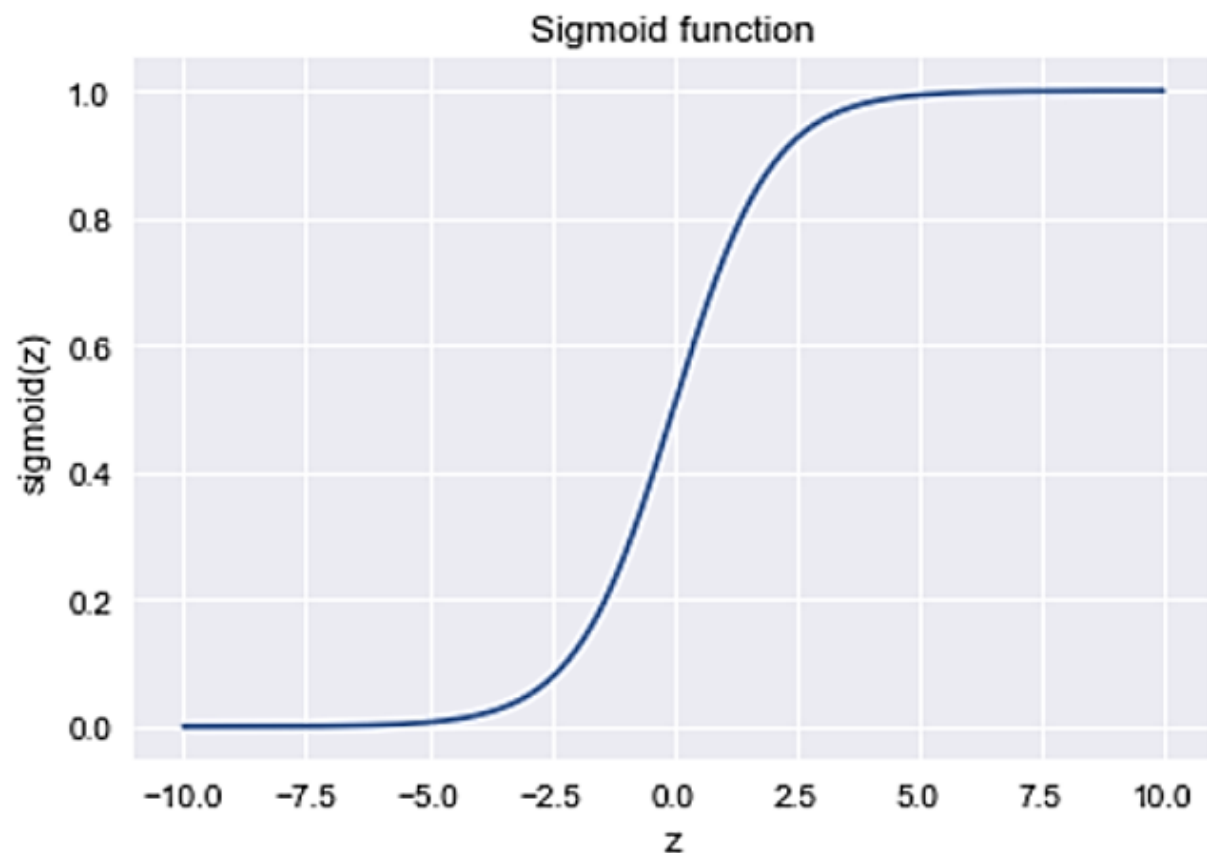


Прогнозирование не просто ответа ("+1" или "-1"), а именно вероятности отнесения к классу "+1" во многих задачах является очень важным бизнес-требованием. Например, в задаче **кредитного скоринга**, где традиционно применяется логистическая регрессия, часто **прогнозируют вероятность невозврата кредита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

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):

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



Реализация логистической регрессии в scikit-learn

```
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='deprecated', verbose=0,  
warm_start=False, n_jobs=None, l1_ratio=None) # \[source\]
```

Регуляризация

Регуляризация — метод добавления ограничений к условию с целью 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.

$$\hat{y} = w[0] * x[0] + w[1] * x[1] + ... + w[p] * x[p] + b > 0$$

Соответственно, необходимо добавить в целевую функцию штраф за слишком большие коэффициенты.

Существует два типа регуляризации:

- ✓ **L1-регуляризация** упрощает модель (и тем самым уменьшает переобучение) путём отбора наиболее важных факторов, которые сильнее всего влияют на результат, наименее важные - обнуляются.
- ✓ **L2-регуляризация** предотвращает переобучения модели путём запрета на непропорционально большие весовые коэффициенты.

По умолчанию обе модели используют L2 регуляризацию.

- **L₁-регуляризация** (англ. *lasso regression*), или **регуляризация Лассо**

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

- **L₂-регуляризация**, или **регуляризация Тихонова**
уравнений позволяет балансировать между соответствием модели данным и

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

L2-регуляризация способствует появлению малых весовых коэффициентов модели, но не способствует их точному равенству нулю.

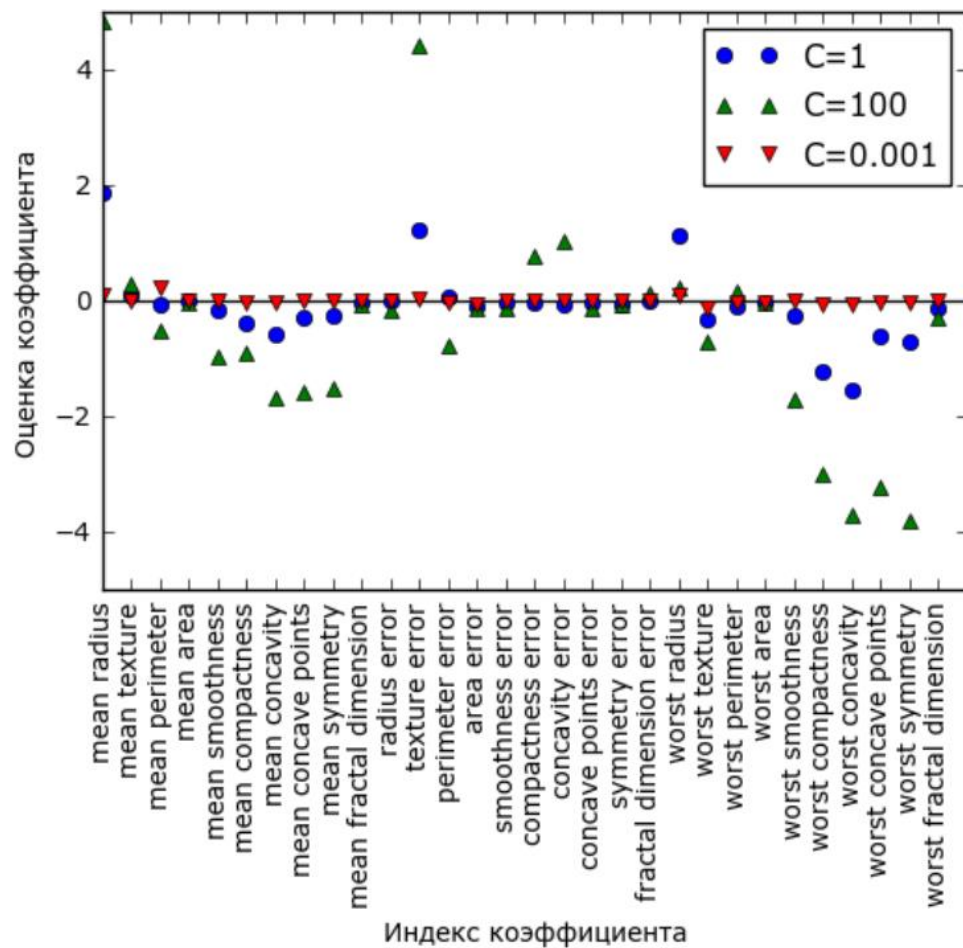


Рис. 2.17 Коэффициенты, полученные с помощью логистической регрессии с разными значениями C для набора данных Breast Cancer

При L1-регуляризации наименее важные параметры обнуляются

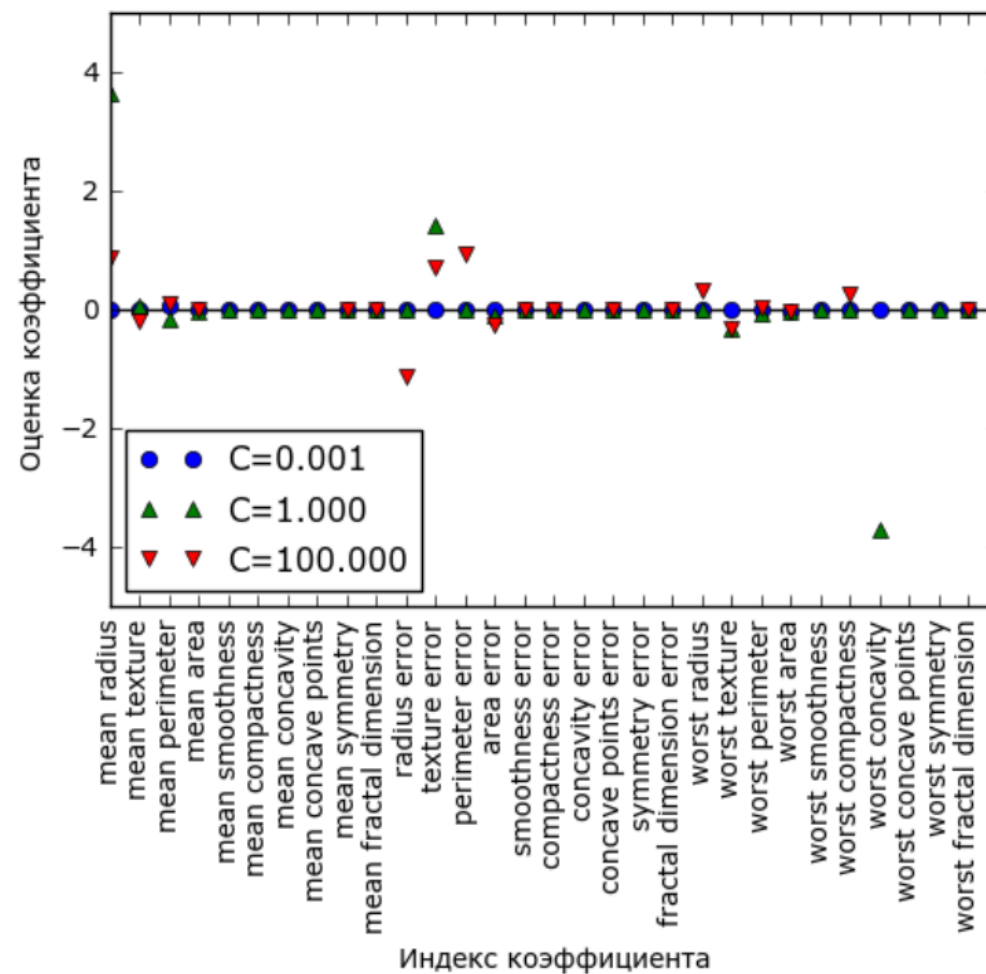
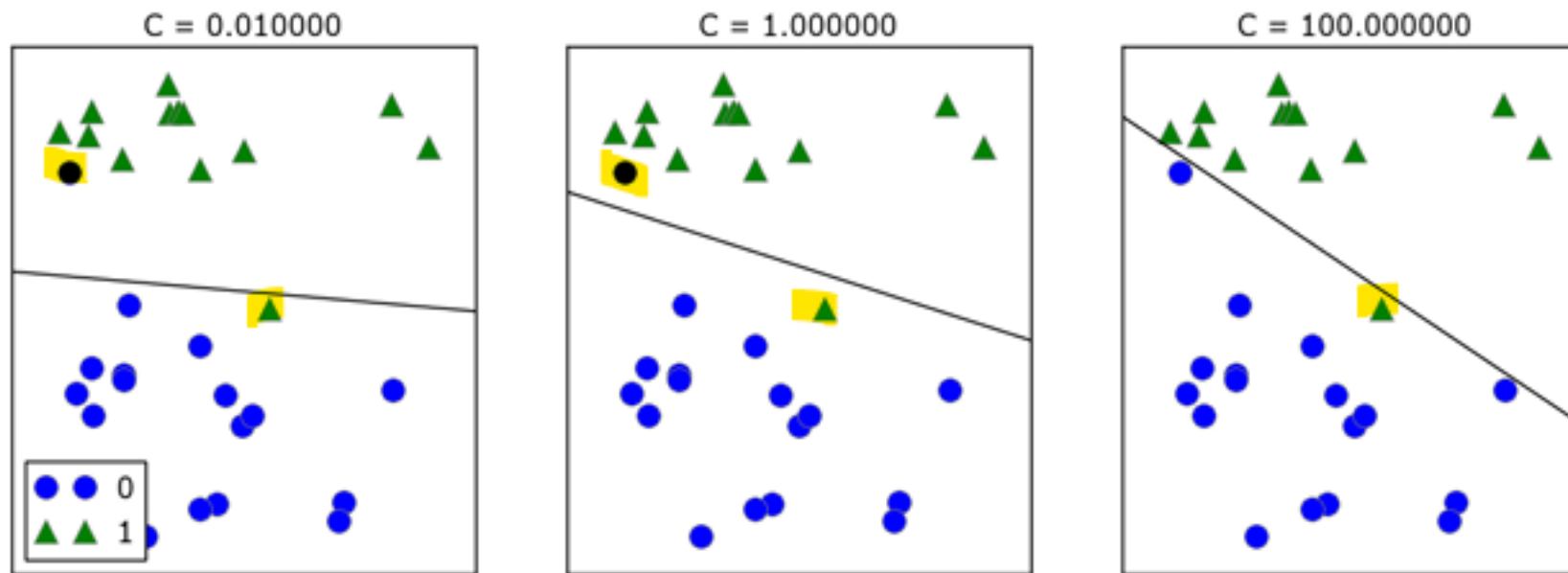


Рис. 2.18 Коэффициенты логистической регрессии с L1 штрафом для набора данных Breast Cancer (использовались различные значения C)

Параметр C определяет степень регуляризации. Использование низких значений C приводит к тому, что алгоритмы пытаются подстроиться под «большинство» точек данных, тем самым имеют хорошую обобщающую способность, тогда как использование более высоких значений C подчеркивает важность того, чтобы каждая отдельная точка данных была классифицирована правильно, но это ведет к переобучению модели.



Модель на графике справа старается изо всех сил правильно классифицировать все точки, но не может дать хорошего обобщения сразу для обоих классов. Другими словами, эта модель скорее всего переобучена.

LogisticRegression

Параметры отвечающие за регуляризацию: *penalty* и *C*

```
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='deprecated', verbose=0, warm_start=False,  
n_jobs=None, l1_ratio=None)
```

[\[source\]](#)

penalty : {'l1', 'l2', 'elasticnet', None}, default='l2'

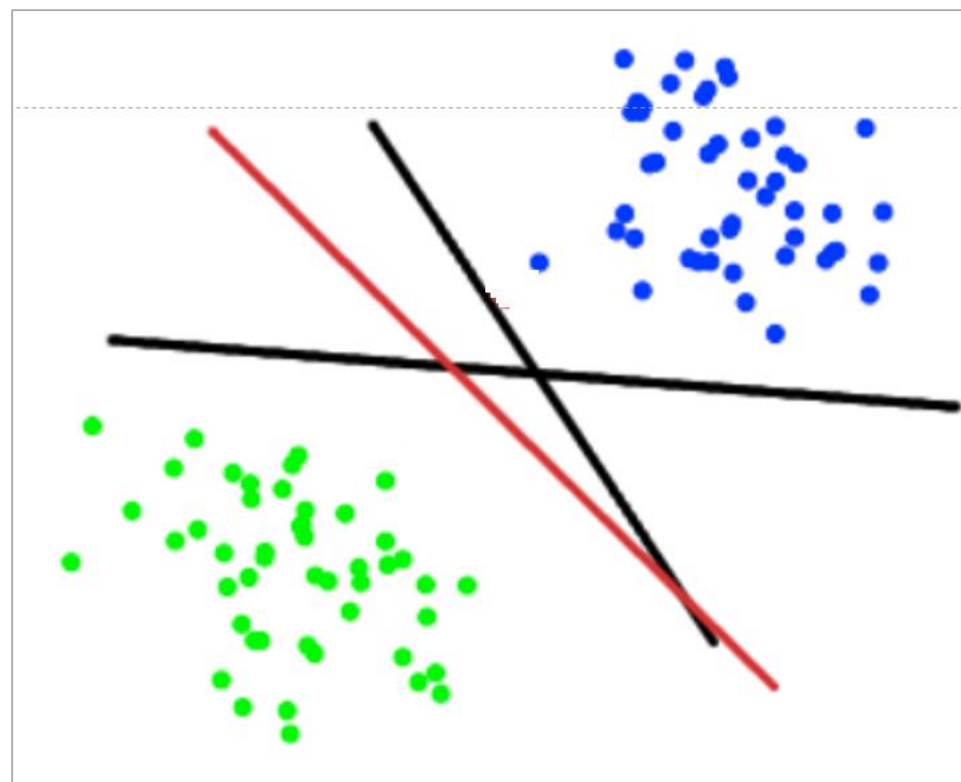
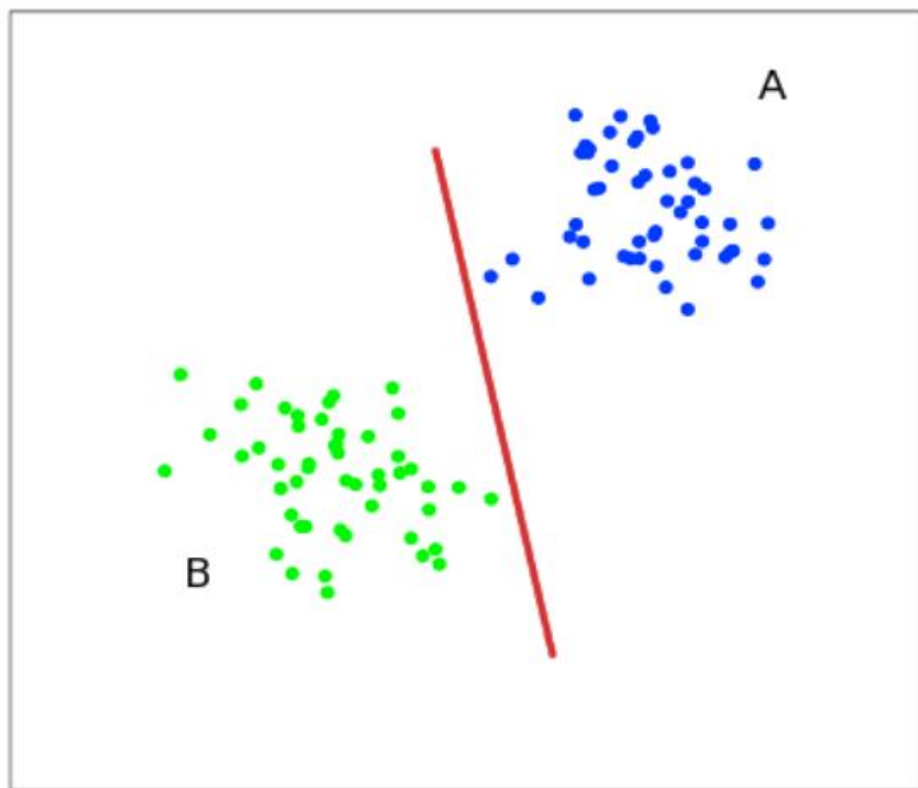
C : float, default=1.0

Specify the norm of the penalty:

- *None* : no penalty is added;
- *'l2'* : add a L2 penalty term and it is the default choice;
- *'l1'* : add a L1 penalty term;
- *'elasticnet'* : both L1 and L2 penalty terms are added.

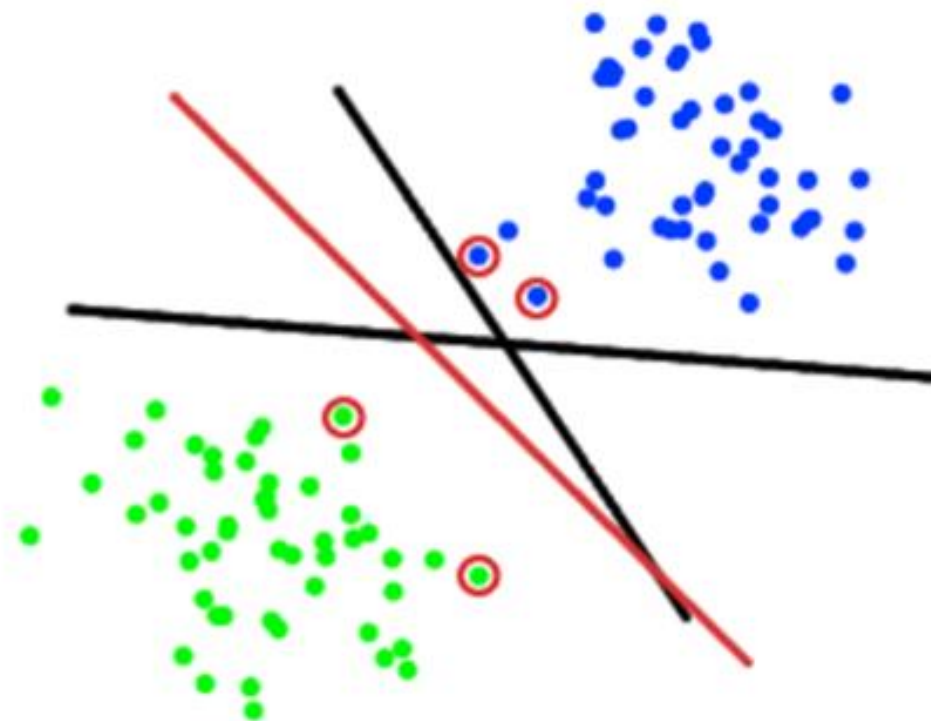
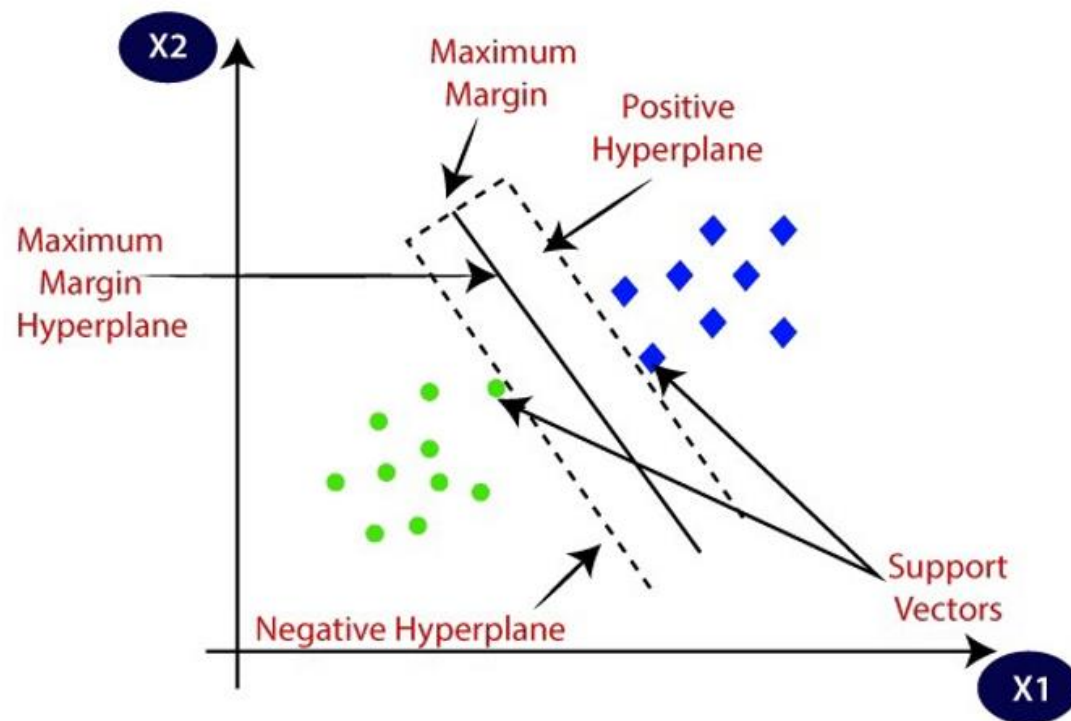
Метод Опорных Векторов (SVM)

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 алгоритм имеет широкое применение на практике. ***Суть работы Метода Опорных Векторов: алгоритм создает линию или гиперплоскость, которая разделяет данные на классы.***

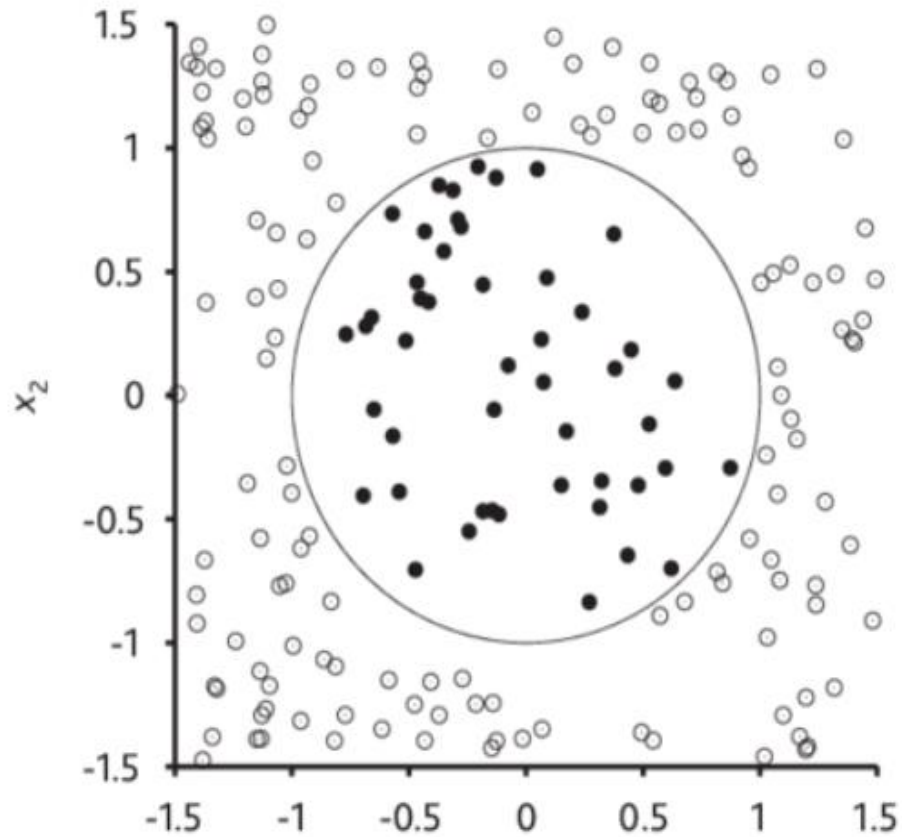


Как SVM находит лучшую линию

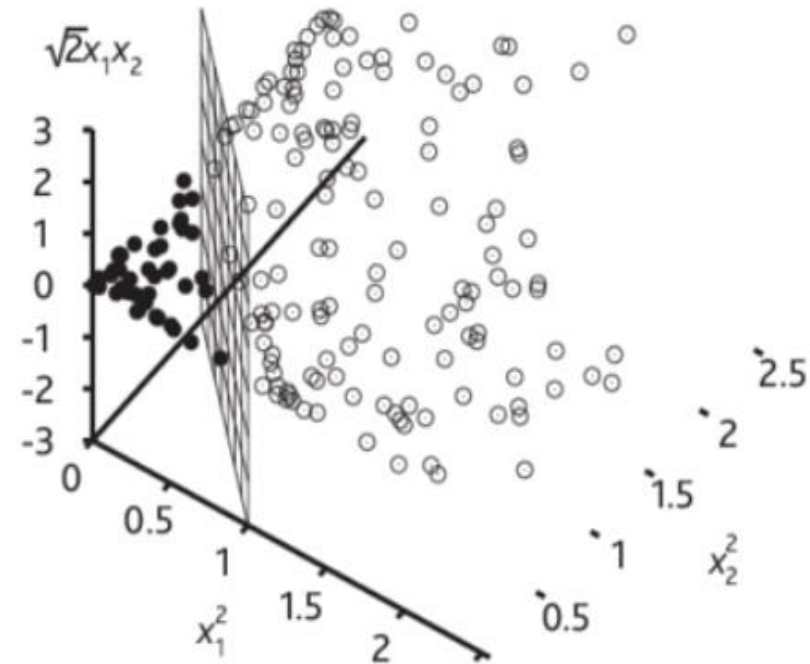
Алгоритм SVM устроен таким образом, что он ищет точки, которые расположены ближе всех к разделяющей гиперплоскости. Эти точки называются **опорными векторами (support vectors)**. Затем, алгоритм вычисляет расстояние между опорными векторами и разделяющей плоскостью. Это расстояние называется **зазором**. **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**.



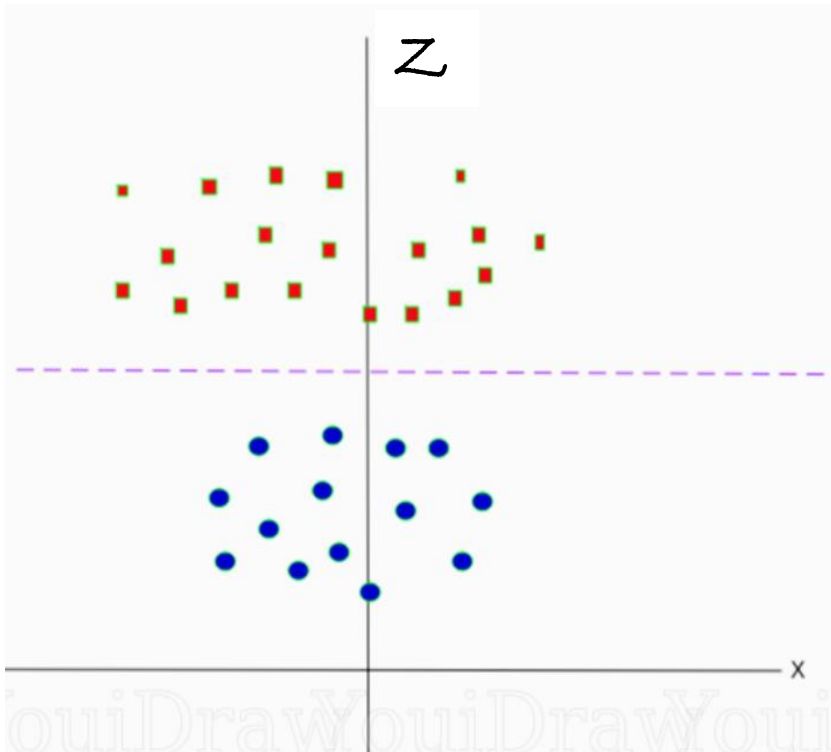
Но, например, невозможно начертить прямую линию, которая бы классифицировала эти данные.



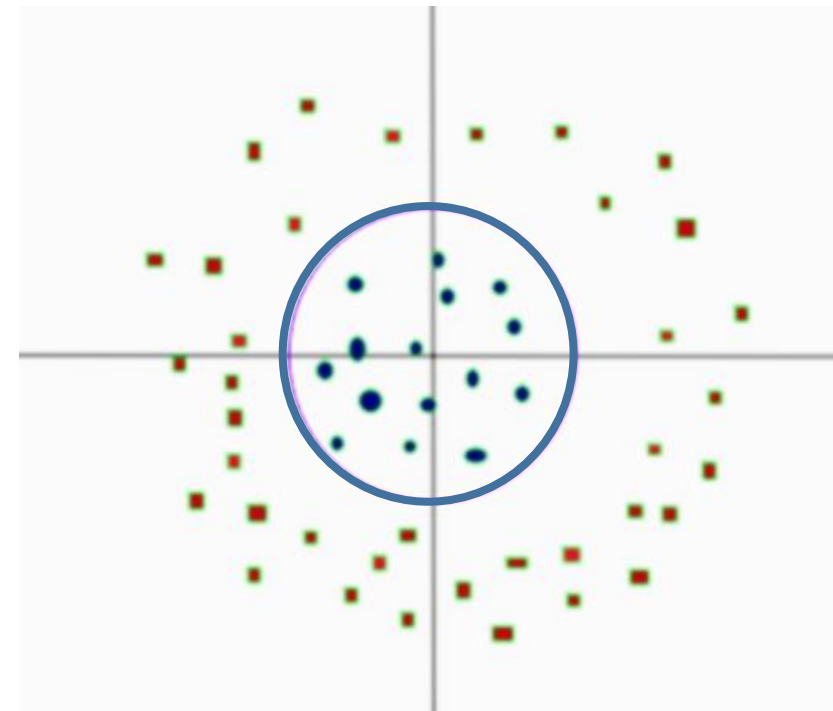
Решение: этот датасет можно разделить линейно, добавив дополнительное измерение, которое мы назовем осью Z.



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.



Теперь данные можно разделить линейно. Допустим линия разделяющая данные $z=k$.
Если $z = x^2 + y^2$, то, и $k = x^2 + y^2$ — 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.

Таким образом 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**.

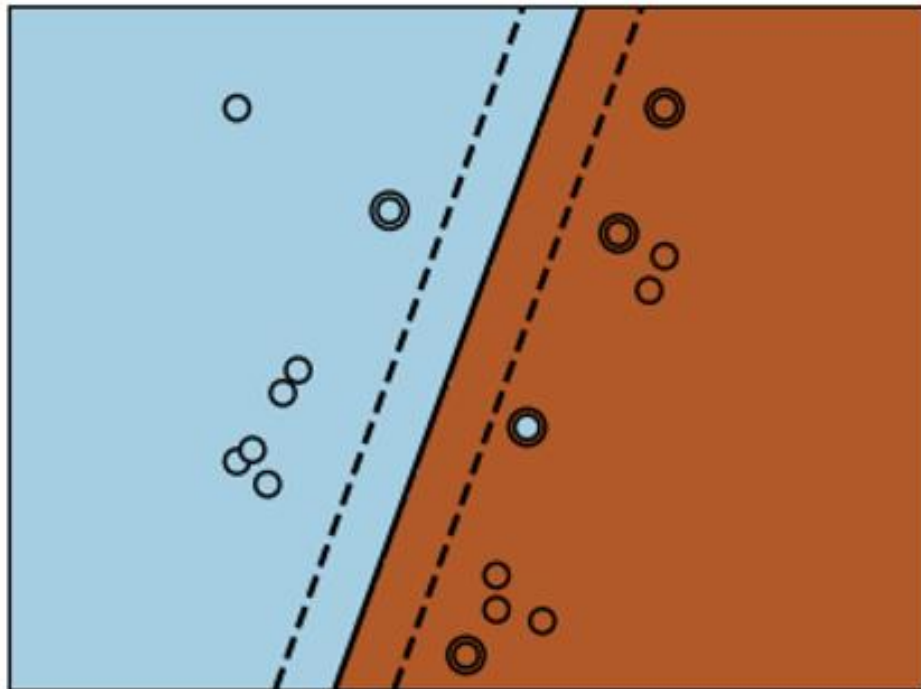
Результаты оценки моделей SVC на датасете «диабет» с разными ядрами, (score и confusion matrix) :

Линейное ядро	Полиномиальное степень 3	RBFядро
0.765625	0.7743	0.7638
<div>[117, 13] [25, 37]</div>	<div>[120, 10] [33, 29]</div>	<div>[119, 11] [32, 30]</div>

Примеры влияния разных ядер на вид оптимальной разделяющей гиперплоскости:

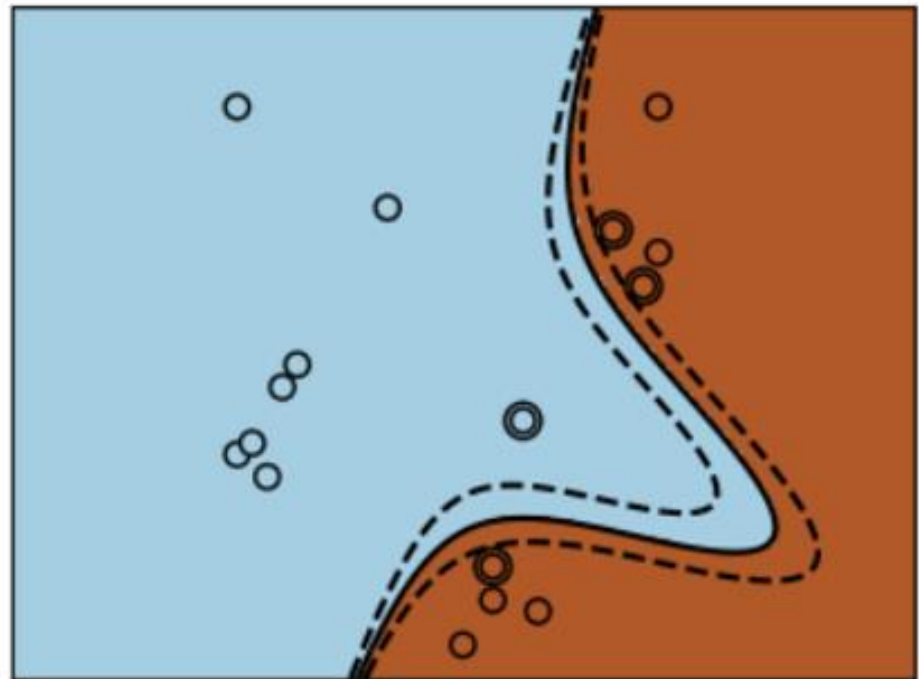
Linear kernel

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



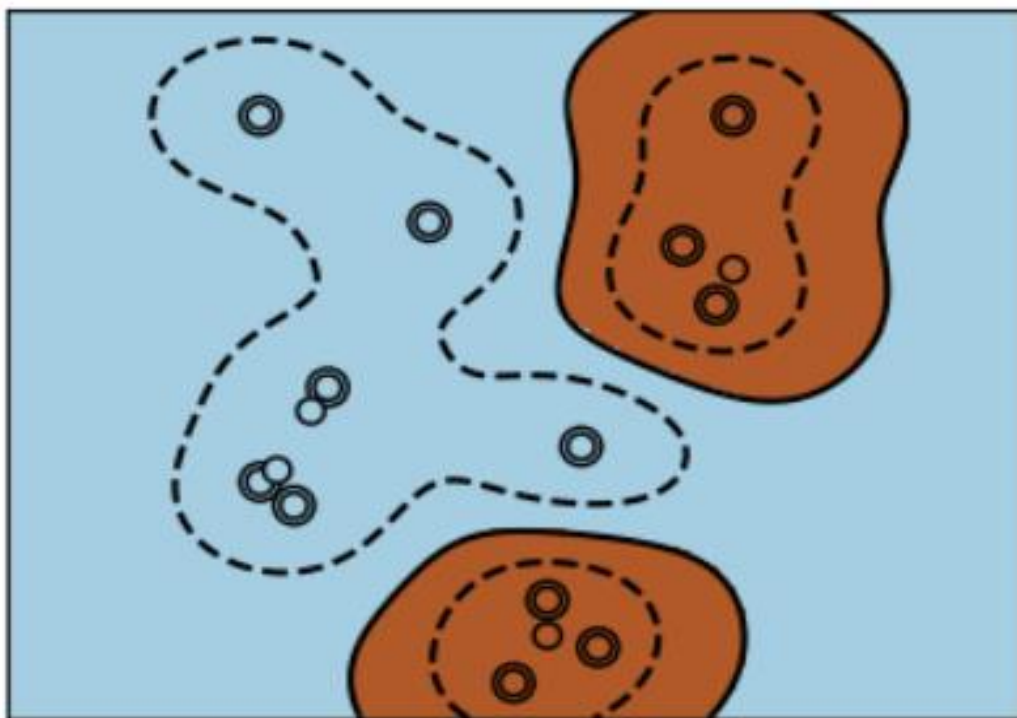
Полиномиальное ядро

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



Ядро RBF (радиальная базисная функция)

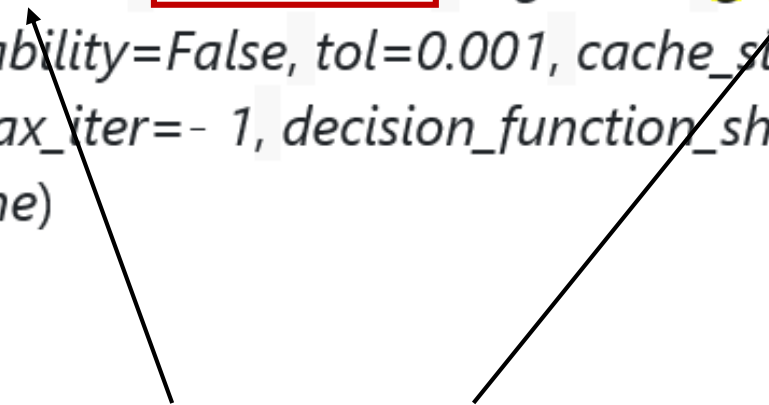
```
>>> svc = svm.SVC(kernel='rbf')  
>>> # gamma: inverse of size of  
>>> # radial kernel
```



SVC (Support Vector Classification).

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', bre  
ak_ties=False, random_state=None)
```



В методе **SVC** есть два параметра *регуляризации*:

- ✓ параметр **C** – задает степень регуляризации
- ✓ и параметр **gamma**

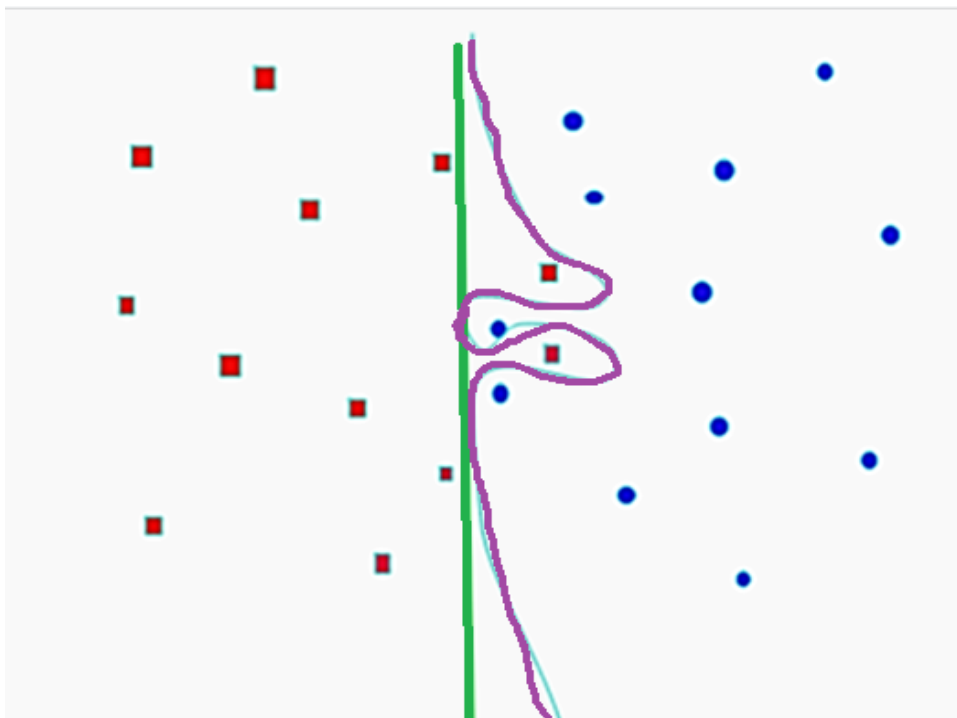
При правильной настройке **C** и **gamma** можно добиться оптимального результата, который построит более линейную гиперплоскость, игнорирующую выбросы, и, следовательно, более обобщающую.

Параметр C

C : float, default=1.0

Если в качестве порога решений использовать прямую (зеленая линия), 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 (фиолетовая линия), то модель явно получается переученной.



Чем выше параметр **C** тем более запутанная гиперплоскость будет в вашей модели, но и выше число верно-классифицированных объектов обучающей выборки.

Высокое значение C – меньше регуляризация модели и наоборот

Параметр Гамма

`gamma : {'scale', 'auto'} or float, default='scale'`
Kernel coefficient for 'rbf', 'poly' and 'sigmoid'.

Чем ниже гамма, тем больше элементов, даже тех, которые достаточно далеки от разделяющей гиперплоскости, принимают участие в процессе выбора идеальной разделяющей 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 **большой вес** при принятии решения.

Для выбора наилучших значений параметров C и γ рекомендуется использовать поиск по сетке **GridSearchCV**.

GridSearchCV генерирует кандидатов из сетки значений параметров, указанных с помощью `param_grid` параметра.

Например, следующее `param_grid` указывает, что необходимо исследовать две сетки: одну с линейным ядром и значениями C в $[1, 10, 100, 1000]$, а вторую с ядром RBF и перекрестным произведением значений C в диапазоне $[1, 10]$, $100, 1000$ и значения γ в $[0,001, 0,0001]$.

```
param_grid = [  
    {'C': [1, 10, 100, 1000], 'kernel': ['linear']},  
    {'C': [1, 10, 100, 1000], 'gamma': [0.001, 0.0001], 'kernel': ['rbf']},  
]
```

Пример вывода
наилучших
подобранных
параметров

```
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})
```

Метрики модели: чувствительность и специфичность

TP	FP
FN	TN

матрица ошибок

✚ 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:

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

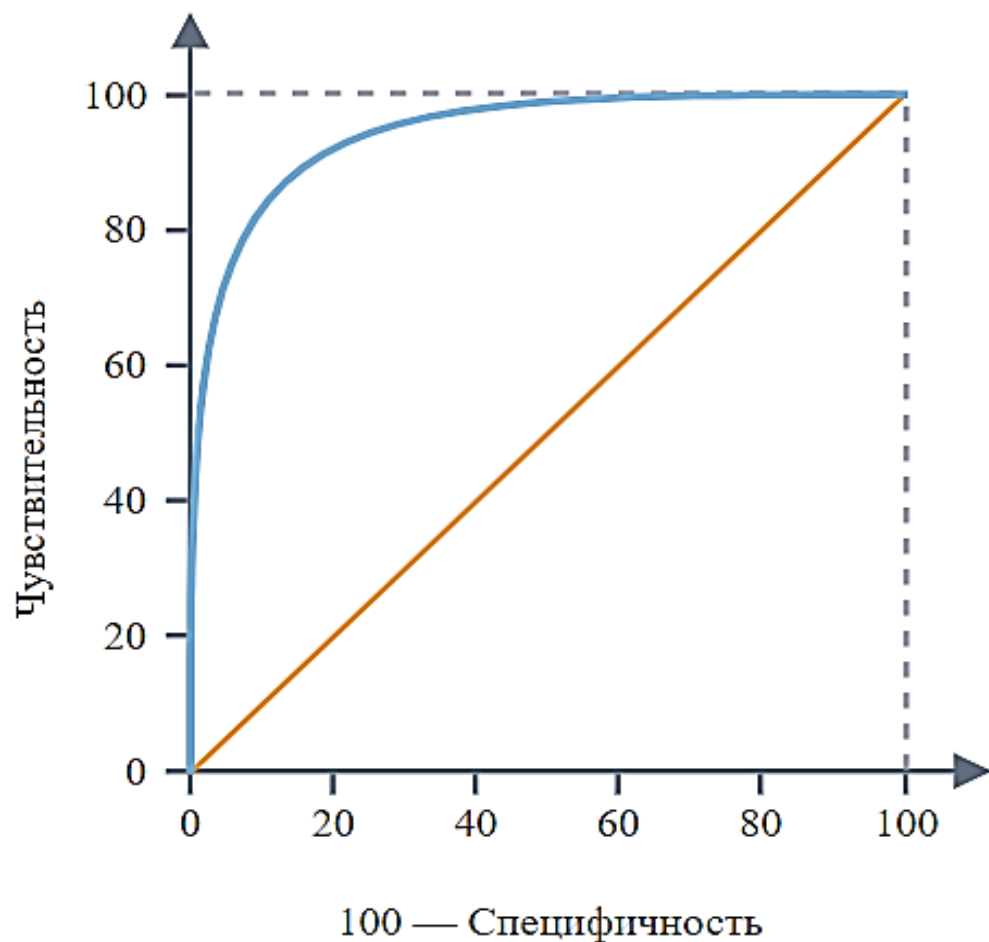
Модель с высокой чувствительностью часто дает истинный результат при наличии положительного исхода (обнаруживает положительные примеры).

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-кривая показывает зависимость количества верно классифицированных положительных примеров (чувствительность) от количества неверно классифицированных отрицательных примеров (100-специфичность).

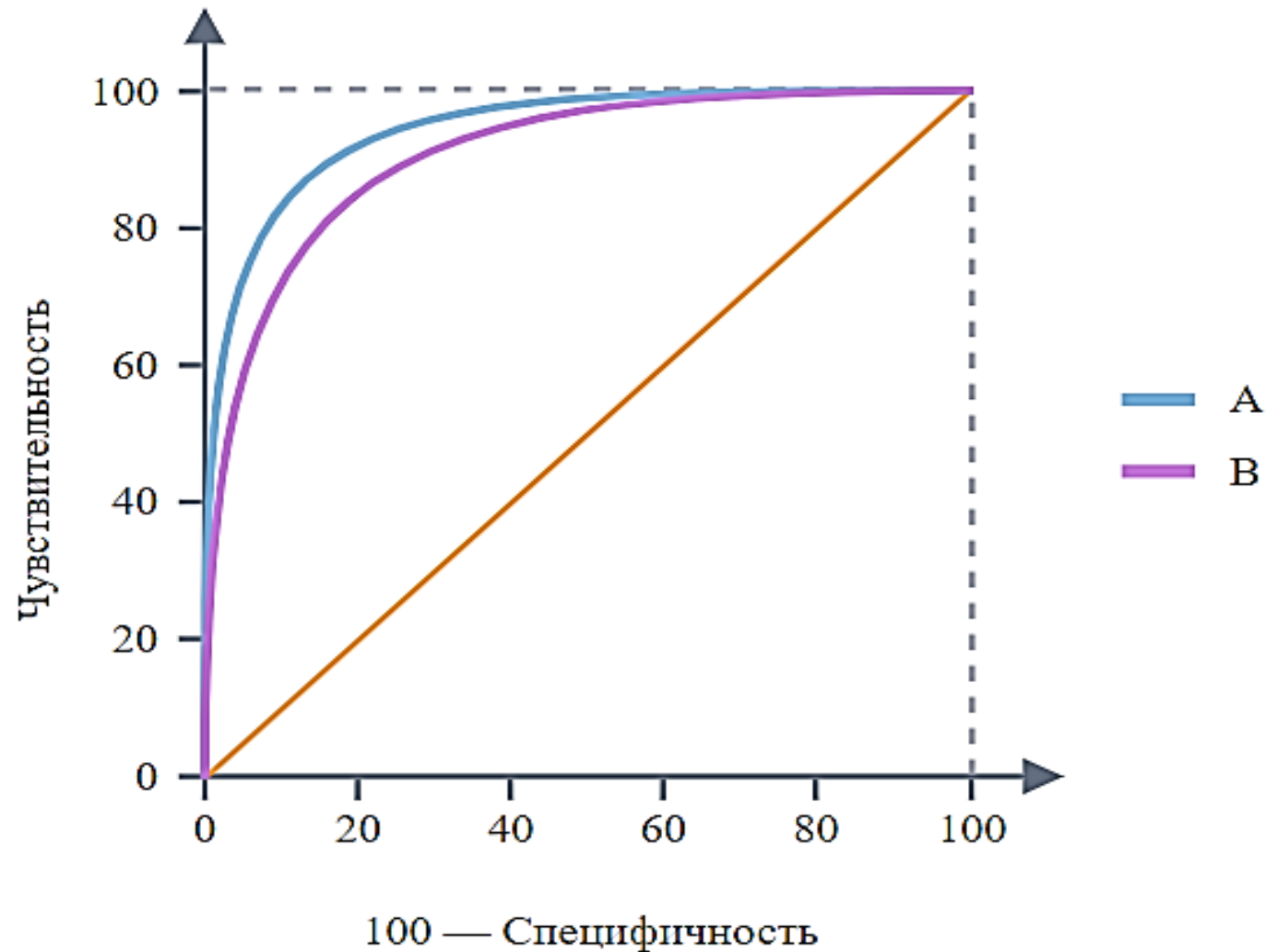


Для идеального классификатора график ROC-кривой проходит через верхний левый угол, где доля истинно положительных случаев составляет 100% или 1,0 (идеальная чувствительность), а доля ложно положительных примеров равна нулю.

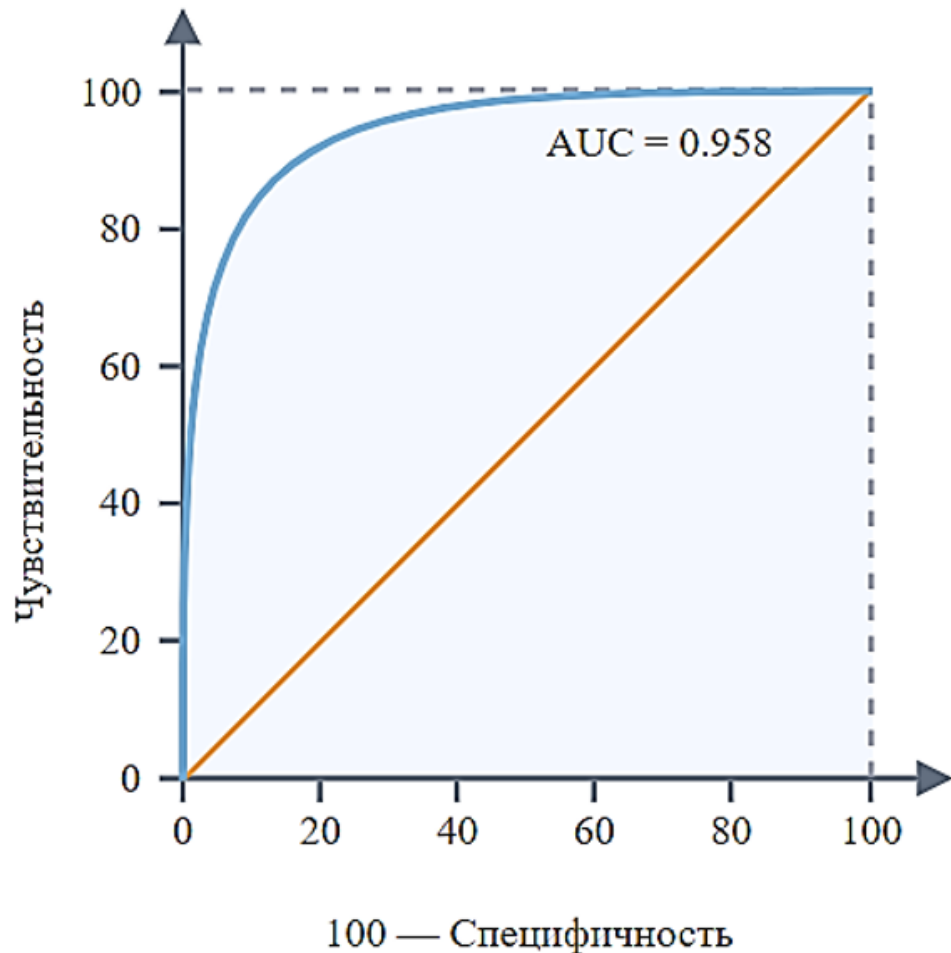
Поэтому чем ближе кривая к верхнему левому углу, тем выше предсказательная способность модели. Наоборот, чем меньше изгиб кривой и чем ближе она расположена к диагональной прямой, тем менее эффективна модель.

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.



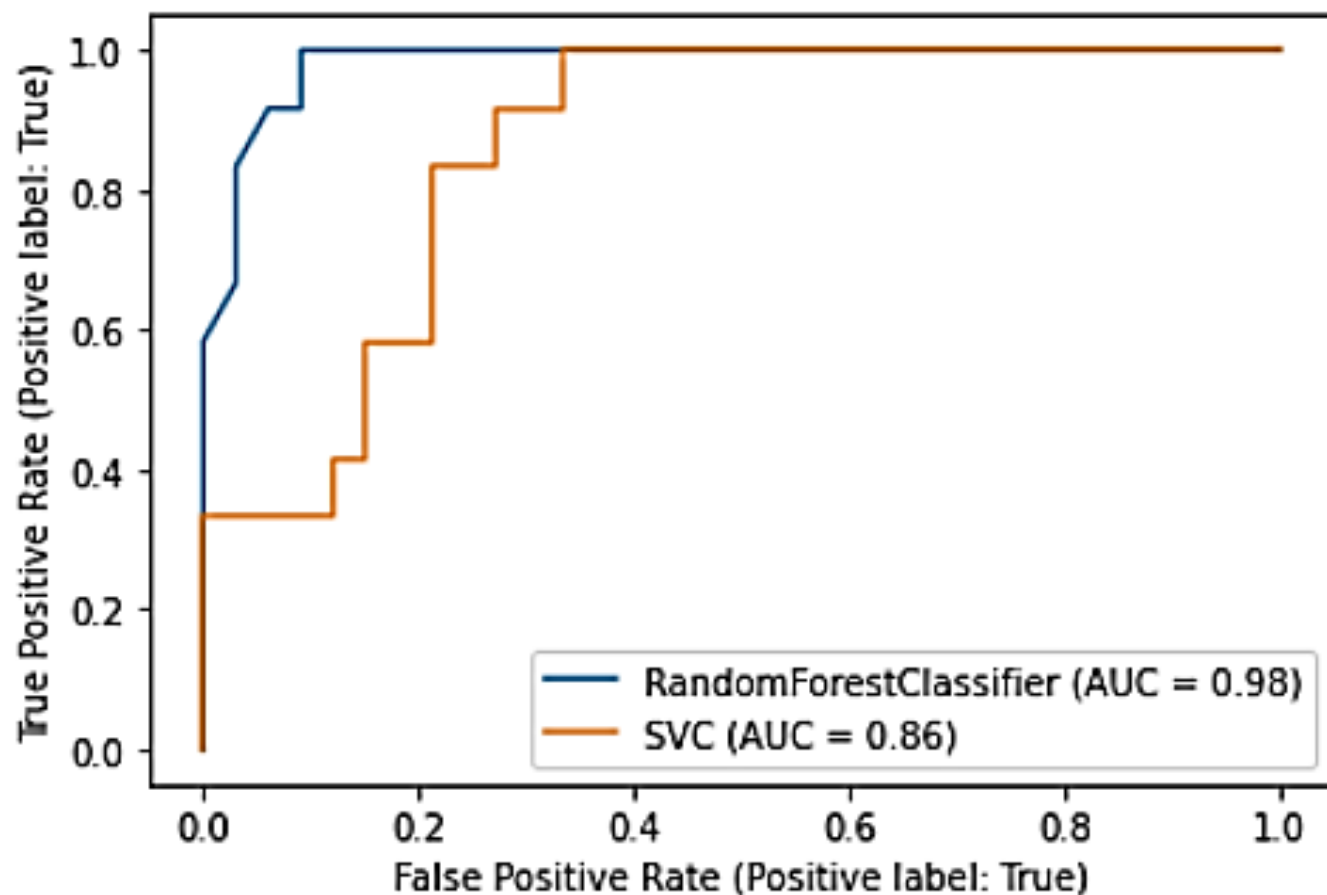
Визуальное сравнение кривых ROC не всегда позволяет выявить наиболее эффективную модель. Своеобразным методом сравнения ROC-кривых является **оценка площади под кривыми**. Теоретически она изменяется от 0 до 1,0, но, поскольку модель всегда характеризуется кривой, расположенной выше положительной диагонали, то обычно говорят об изменениях от 0,5 («бесполезный» классификатор) до 1,0 («идеальная» модель).



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

```
rfc = RandomForestClassifier(n_estimators=10, random_state=42)
rfc.fit(X_train, y_train)
ax = plt.gca()
rfc_disp = RocCurveDisplay.from_estimator(rfc, X_test, y_test, ax=ax)
svc_disp.plot(ax=ax)
plt.show()
```



Идеальная модель обладает 100% чувствительностью и специфичностью. Однако на практике невозможно одновременно повысить и чувствительность, и специфичность модели. Компромисс находится с помощью порога отсечения (cut-off value).

Для определения оптимального порога нужно задать критерий его определения, т.к. в разных задачах присутствует своя оптимальная стратегия. Критериями выбора порога отсечения могут выступать:

- ✓ Требование минимальной величины чувствительности (специфичности) модели.
- ✓ Требование максимальной суммарной чувствительности и специфичности модели
- ✓ Требование баланса между чувствительностью и специфичностью. В этом случае порог есть точка пересечения двух кривых, когда по оси X откладывается порог отсечения, а по оси Y — чувствительность или специфичность модели

