Метрики качества в задачах классификации ч.2

In [1]:

```
import numpy as np

from sklearn.datasets import make_blobs
from sklearn.linear_model import LogisticRegression

from sklearn.metrics import fbeta_score
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve
from sklearn.metrics import precision_recall_curve
from sklearn.metrics import auc
from sklearn.metrics import average_precision_score
from sklearn.metrics import f1_score
from sklearn.model_selection import train_test_split

import seaborn as sns
import matplotlib.pyplot as plt

sns.set(style='dark', font_scale=1.7)
```

Для дальнейшего изучения метрик качества возьмем сгенерируем выборку при помощи `make_blobs` из `sklearn` и обучим логистическую регрессию:

In [2]:

```
X, y = make_blobs(n_samples=(500, 50), centers=[[2, 2], [-2, -2]],
                  cluster_std=2.5, random_state=42)
```

In [3]:

```
plt.figure(figsize=(8, 5))
plt.title('Сгенерированная выборка')
plt.scatter(X[:, 0], X[:, 1], c=y, alpha=0.8, cmap='Accent')
plt.grid()
plt.xlabel('Признак 1'), plt.ylabel('Признак 2')
plt.show()
```

In [4]:

```
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42
)
```

In [5]:

```
clf = LogisticRegression(random_state=42)
clf.fit(X_train, y_train)
```

Out[5]:

```
LogisticRegression(random_state=42)
```

Обобщенная F-мера (F_β-мера)

F_β-мера в `sklearn` (https://scikit-learn.org/stable/modules/generated/sklearn.metrics.fbeta_score.html#sklearn.metrics.fbeta_score)

Обобщением F_1-меры является F_β-мера, которая равна взвешенному гармоническому precision и recall с коэффициентом β:

$$F_\beta = (1 + \beta^2)\frac{precision \cdot recall}{(\beta^2 \cdot precision) + recall}$$

В данном случае β определяет вес точности в метрике.

Эта метрика хороша в тех задачах, где от нас требуется обращать больше внимания на один из параметров precision или recall.

$0 < \beta < 1 \Rightarrow$ важнее precision

$\beta = 1 \Rightarrow$ получаем среднее гармоническое, то есть F_1-меру

$1 < \beta < +\infty \Rightarrow$ важнее recall

Одним из примеров задач, в которых полезно вычислять именно F_β с $\beta \neq 1$, является задача предсказания болезни. В такой задаче гораздо важнее recall, нежели precision: лучше здоровому человеку сказать, что он болен и проверить его более тщательно, чем не определить больного.

Посчитаем F_β-мера для нашей задачи:

In [6]:

```
fbeta = fbeta_score(y_test, clf.predict(X_test), beta=2)
print(f'FBeta-score (beta=2): {fbeta:.3f}')
```

```
FBeta-score (beta=2): 0.526
```

Площадь под ROC-кривой (Area Under ROC Curve, AUC-ROC)

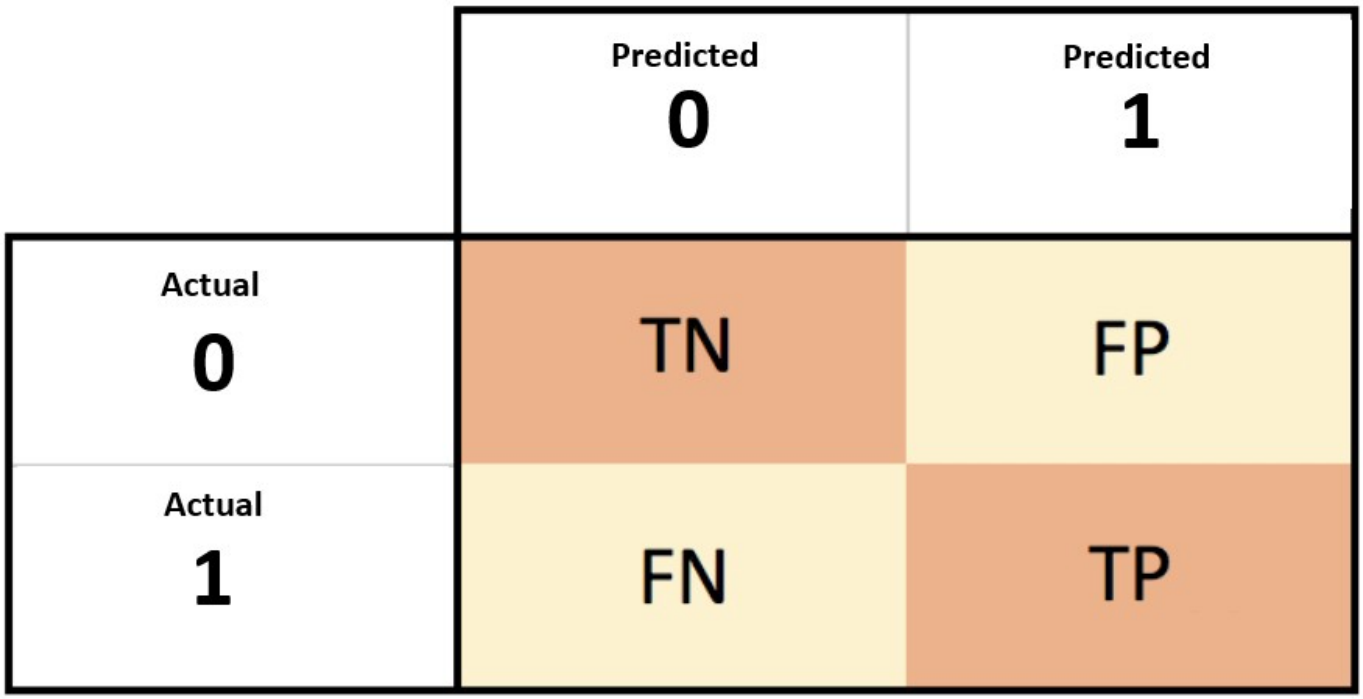
AUC-ROC в `sklearn` (https://scikit-learn.org/stable/modules/generated/sklearn.metrics.auc.html#sklearn.metrics.auc)

При переводе вещественнозначного ответа модели (например, в логистической регрессии, вероятности принадлежности к классу 1) в бинарную метку $\{0, 1\}$, мы должны выбрать порог перехода. Обычно, таким порогом является 0.5, но такой выбор не всегда является оптимальным, например, при отсутствии балансов классов.

Ранее изученные метрики (precision, recall, accuracy) характеризуют точность работы модели при конкретно выбранном пороге t бинарной классификации.

Одним из способов оценить модель в целом, не привязываясь к конкретному порогу, является **AUC-ROC**. Данная кривая представляет из себя кривую от $(0, 0)$ до $(1, 1)$ в координатах **True Positive Rate (TPR)** и **False Positive Rate (FPR)**.

Для понимания того, что такое TPR и FPR вспомним матрицу ошибок:



В приведенной ниже терминологии мы считаем класс с меткой 1 положительным, а с меткой 0 — негативным

TPR (True Positive Rate, полнота) показывает какую долю объектов положительного класса модель классифицировала правильно:

$$TPR = \frac{TP}{TP + FN}$$

FPR (False Positive Rate) показывает, какую долю из объектов негативного класса модель предсказала неверно:

$$FPR = \frac{FP}{FP + TN}$$

Каждая точка на графике соответствует выбору некоторого порога бинарной классификации t.

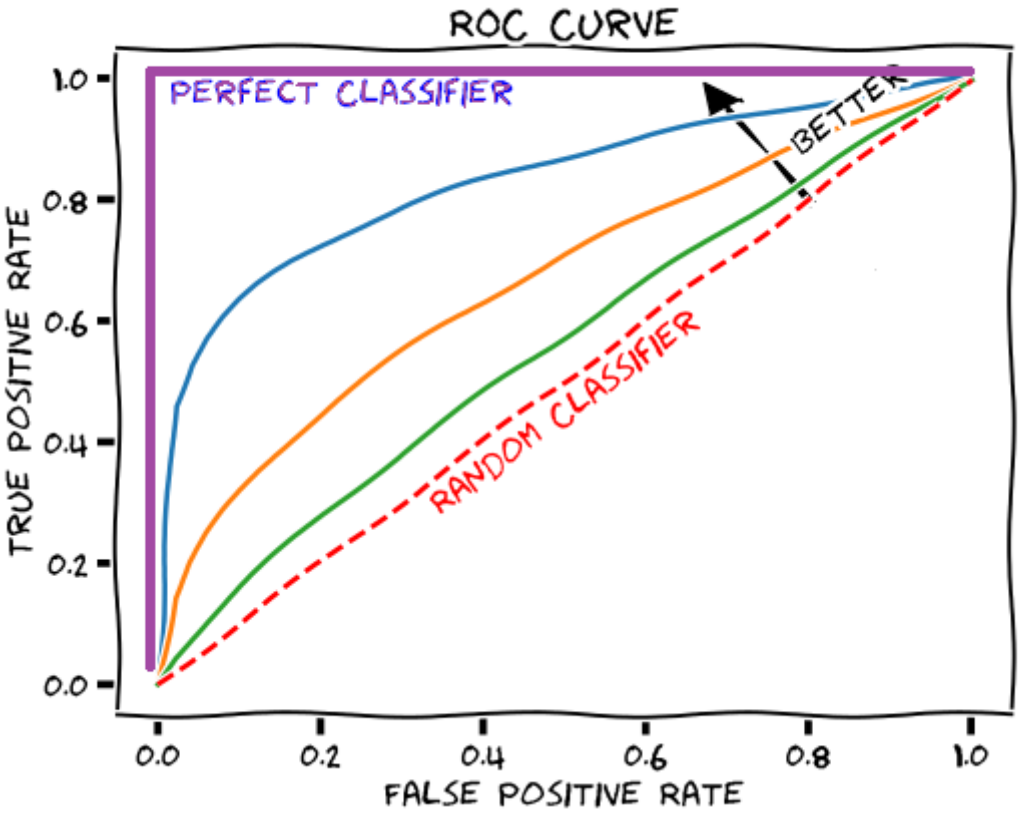
Кривая рисуется следующим образом:

- мы перебираем пороги, начиная с 1 и до 0.
- При $t = 1$ всем объектам присваивается класс 0, значит мы находимся в точке $(0, 0)$
- уменьшаем t до тех пор пока TPR и FPR не изменяются. Когда изменились, отмечаем точку (FPR , TPR) на графике
- продолжаем алгоритм до тех пор, пока все точки не будут рассмотрены

Площадь под кривой в данном случае показывает качество модели: чем она больше, тем классификатор лучше.

Также важным является крутизна самой кривой — мы хотим максимизировать TPR, минимизируя FPR, а значит, наша кривая в идеале должна стремиться к точке $(0, 1)$.

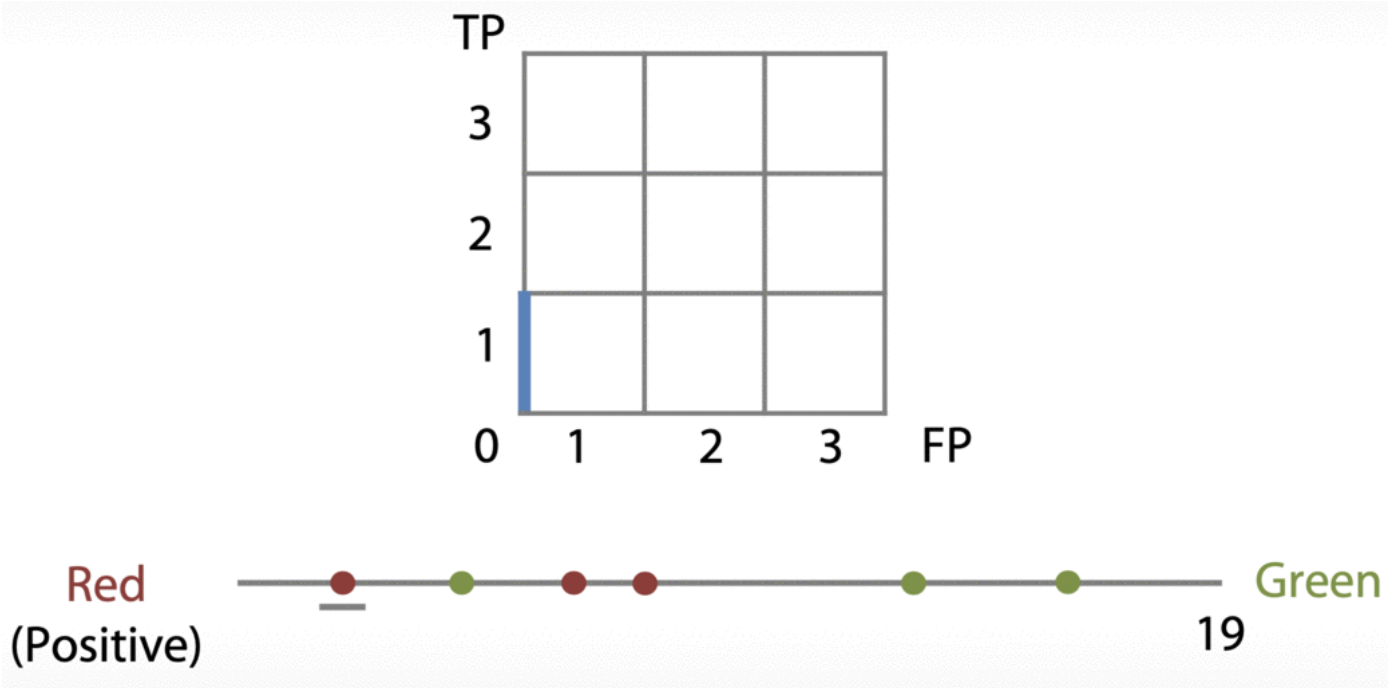
В идеальном случае, когда классификатор не делает ошибок (FPR = 0, TPR = 1) мы получим площадь под кривой, равную единице:



Заметим также, что FPR и TPR нормируются на размеры классов, поэтому AUC-ROC не поменяется при изменении баланса классов.

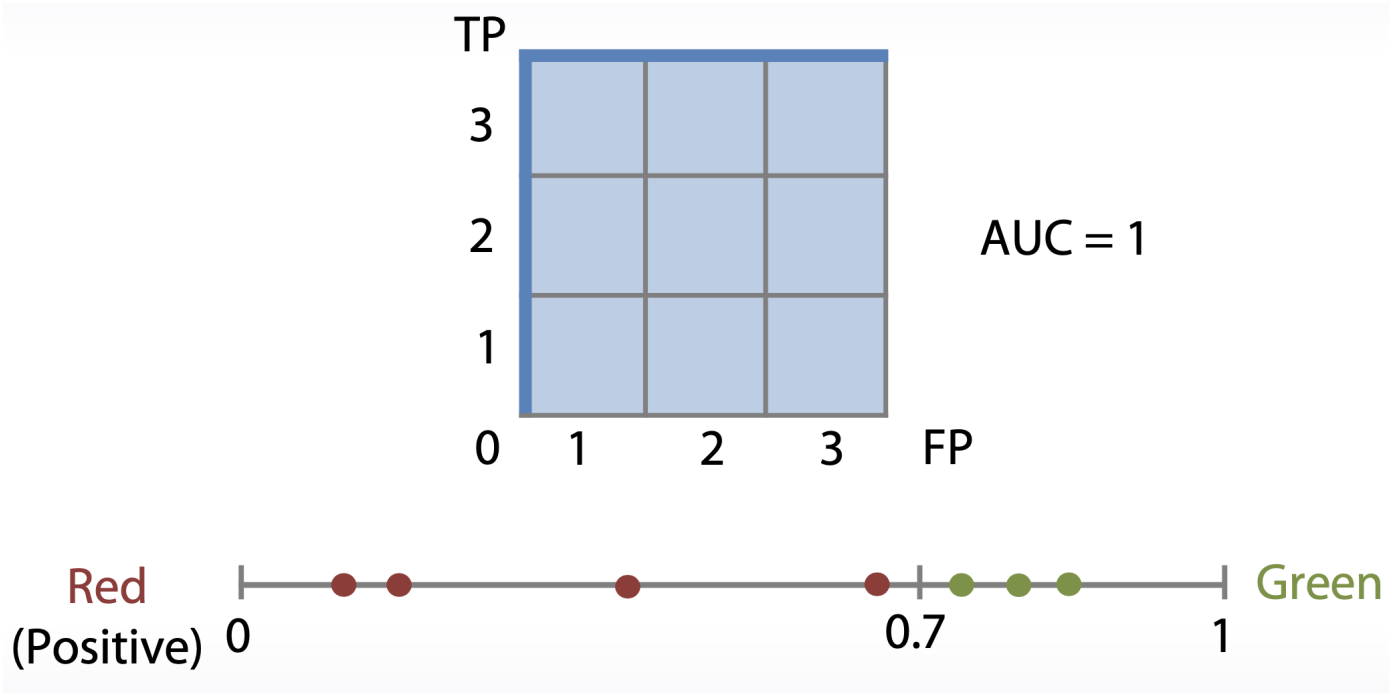
AUC-ROC через упорядоченные/неупорядоченные пары

Пусть в нашей тестовой выборке n объектов класса 0 и m объектов класса 1. Тогда представим квадрат $[0, 1] \times [0, 1]$ в виде прямоугольной сетки из $m \times n$ блоков.



Тогда каждый блок ниже синей линии соответствует паре (объект класса 1, объект класса 0), для которой наш алгоритм правильно предсказал порядок (объект класса 1 получил оценку выше, чем объект класса 0), блок выше синей линии — паре, на которой ошибся.

В случае правильного порядка всех пар AUC-ROC будет равен 1.



AUC-ROC может быть записан через количество правильно упорядоченных пар:

$$\text{AUC-ROC} = \frac{\text{\# correctly ordered pairs}}{\text{total number of pairs}} = 1 - \frac{\text{\# incorrectly ordered pairs}}{\text{total number of pairs}}$$

Эта формула игнорирует случаи, когда объекты разных классов находятся в одной точке. В такой ситуации считаем, что пара дает вклад $1/2$.

Такое альтернативое определение позволяет нам взглянуть на AUC-ROC, как на метрику ранжирования.

Посчитаем AUC-ROC для нашей задачи:

In [7]:

```
roc_auc = roc_auc_score(y_test, clf.predict_proba(X_test)[:, 1])
print(f'AUC-ROC: {roc_auc:.3f}')
```

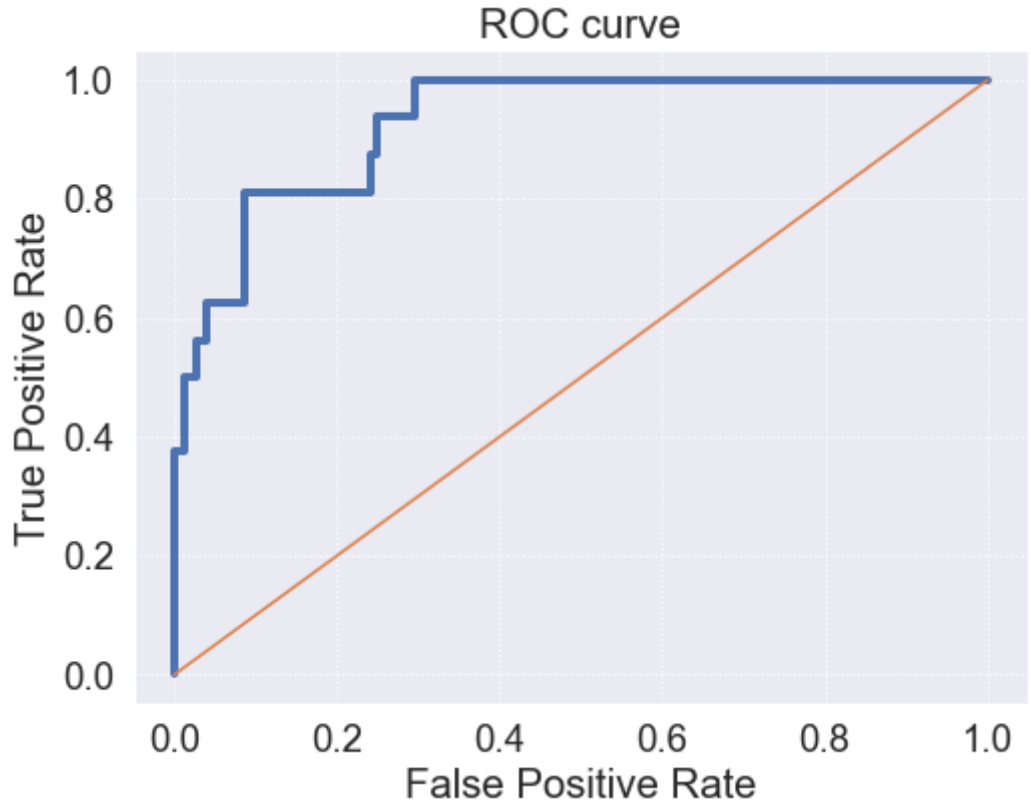
```
AUC-ROC: 0.929
```

Отрисуем ROC-кривую:

In [8]:

```
fpr, tpr, thresholds = roc_curve(y_test, clf.predict_proba(X_test)[:,1])

plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, lw=4, label='ROC curve')
plt.plot([0, 1], [0, 1])
plt.xlim([-0.05, 1.05])
plt.ylim([-0.05, 1.05])
plt.grid(ls=":")
plt.title('ROC curve')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.show()
```



Пример:

Попробуем проиллюстрировать почему подбор метрики под вашу задачу — это важно и брать первую попавшуюся может привести к неудовлетворительным результатам.

Рассмотрим задачу предсказания реакции клиента банка на звонок с предложением кредита. Пусть в действительности (в нашей обучающей выборке) 10 клиентов из 100000 приняли предложение о кредите после звонка.

Рассмотрим два классификатора:

1. Дает самую высокую вероятность 10 релеватным клиентам. Для такой модели:

$$AUCROC = 1$$

2. Располагает релеватных клиентов на позициях 10, 20, 30, 40, 50, 60, 70, 80, 90, 100.

   Получим:

In [9]:

```
y_true = np.zeros(100_000)
y_score = np.arange(100_000)[::-1]
y_true[np.arange(10, 101, 10)] = 1
roc_auc_value = roc_auc_score(y_true=y_true, y_score=y_score)
print(f"AUC-ROC: {roc_auc_value:.5f}")
```

```
AUC-ROC: 0.99949
```

Видим, что показатель очень высокий. В самом деле, из всех возможных 100 тыс. позиций, модель поместила все релевантные в топ 100, весьма хороший результат. Вот только если в реальности каждый звонок стоит денег и несет свои риски, то нам хочется как можно чаще получать конверсию звонка в кредит.

Имеем для первой модели:

$$precision \leq 1$$

Для второй:

$$precision \leq 0.1$$

И тут разница между моделями кратная.

Алгоритм подсчета

Рассмотрим алгоритм подсчета AUC-ROC за $O(n \log n)$.

Имеем:

- ответы: $y_1, \dots, y_n$;
- выходы модели: $p_1, \dots, p_n$.

Введем:

- $m_+ = TP + FN$;
- $m_- = FP + TN$.

Упорядочим вероятности $O(n \log n)$:

$$p_1, \dots, p_n \to p_{(n)}, \dots, p_{(1)}.$$

Применим эту же перестановку к $y_1, \dots, y_n$.

Теперь опишем алгоритм $O(n)$.

```
FPR = [0]*(n+1)
TRP = [0]*(n+1)
AUC = 0
for i in range(1, n+1):
    if y[i-1] == 1:
        FPR[i] = FPR[i-1]
        TPR[i] = TPR[i-1] + 1/m_plus
    else:
        FPR[i] = FPR[i-1] + 1/m_minus
        TPR[i] = TPR[i-1]
        AUC += 1/m_minus * TPR[i]
```

Стоит сразу отметить, что это упрощенная версия, которая не умеет обрабатывать корректно случаи, когда одному значению p соответствует сразу множество точек. В таком ситуации нам следует шагать "по диагонали" как будто мы сразу обработали все эти точки за один раз.

Площадь под PR-кривой (Area Under PR Curve, AUC-PR)

AUC-PR в `sklearn` (https://scikit-learn.org/stable/modules/generated/sklearn.metrics.precision_recall_curve.html#sklearn.metrics.precision_recall_

Аналогично ROC-кривой, определим PR-кривую как кривую в координатах recall и precision, начинающуюся в точке $(0, 0)$.

Каждая точка на графике соответствует выбору некоторого порога бинарной классификации t. Площадь под кривой в данном случае показывает качество модели: чем она больше, тем классификатор лучше.

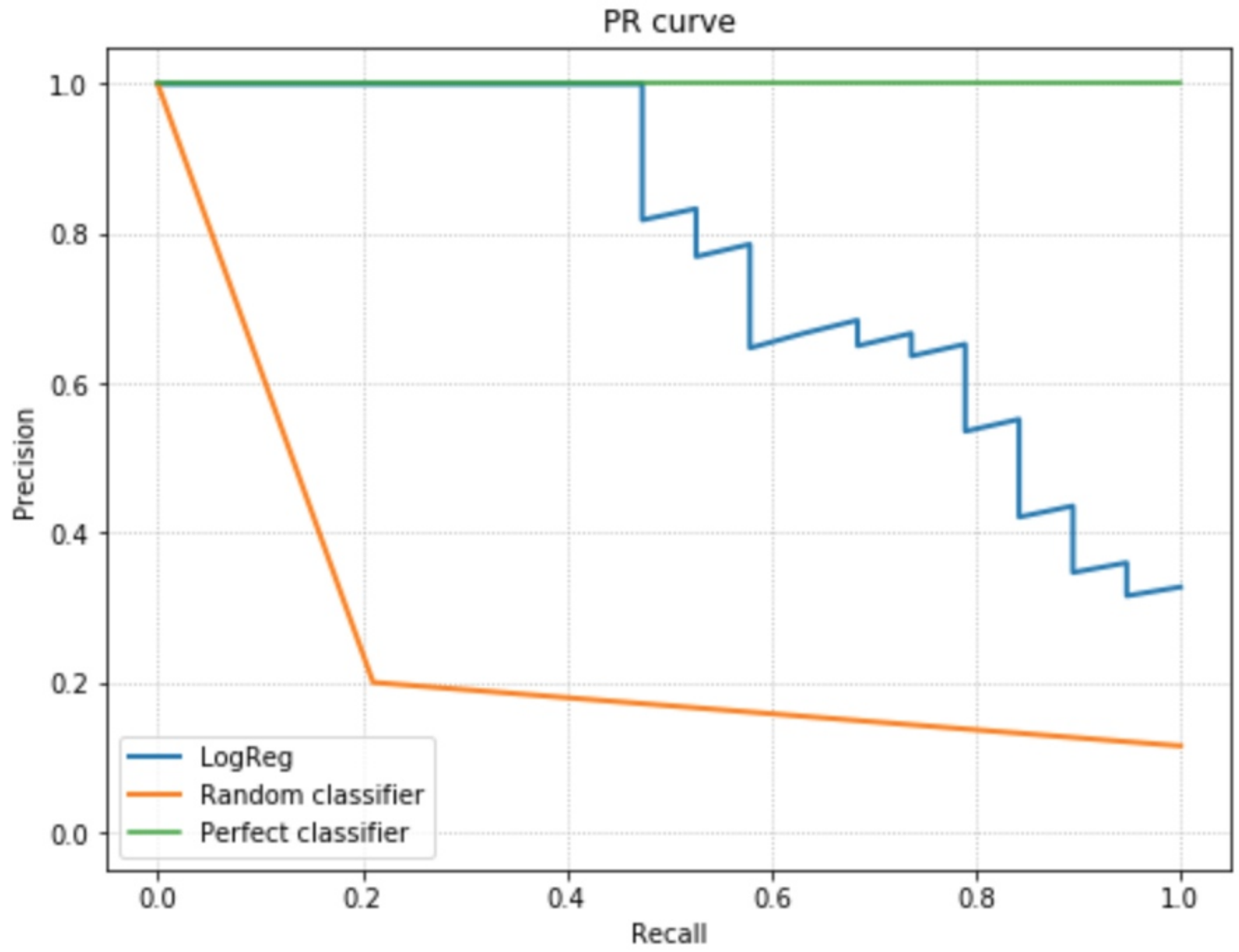
При построении кривой перебираем порог от 1 до 0. Как только precision или recall меняется, наносим точку на график.
При пороге $t = 1$: recall = 0, precision = 0
При пороге $t = 0$: recall = 1, precision = $TP/(TP + FP)$ (может как быть 1, так и быть меньше)

Замечание.

На графиках кажется, что кривая начинается из точки $(0, 1)$. Это связано с тем, что если t находится в окрестности 1, но при этом не равно 1, то precision скорей всего будет большим и близким к единице. Более того, многие стандартные методы отрисовки PR-кривой пропускают точку $t = 1$ и начинают отрисовку со следующего значения t.

Также важным является "пологость" самой кривой — мы хотим максимизировать и precision и recall, а значит, наша кривая в идеале должна стремиться к точке $(1, 1)$.

В идеальном случае, когда классификатор не делает ошибок (precision=1, recall=1) мы получим площадь под кривой, равную единице:

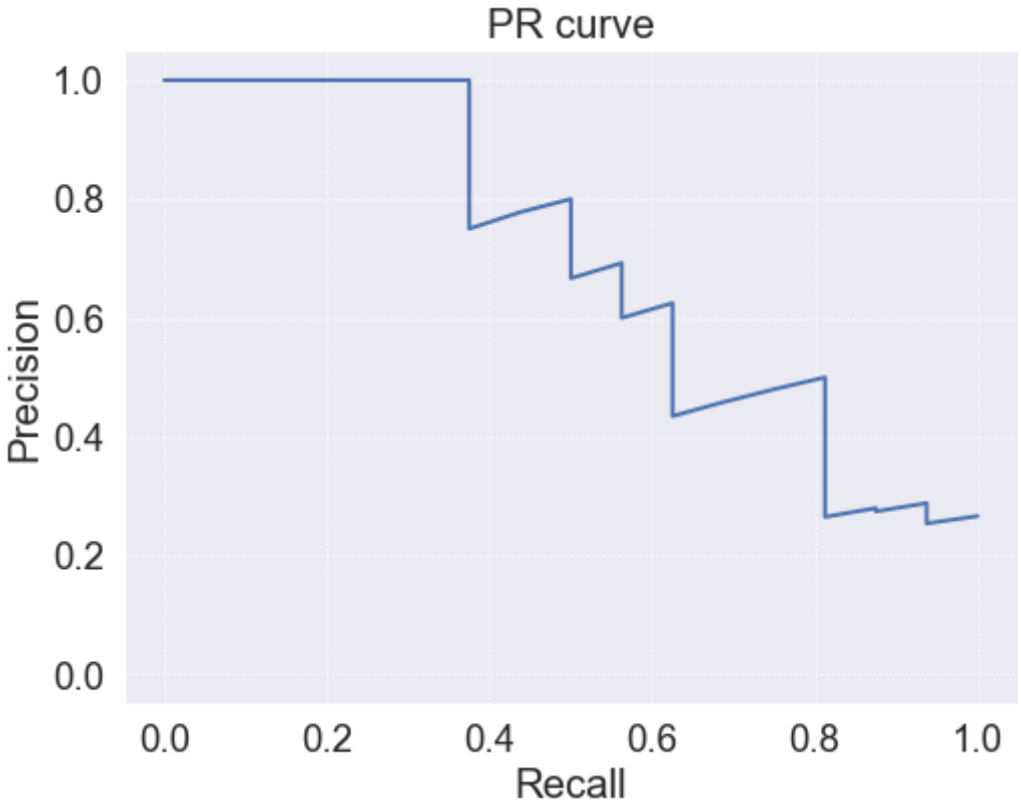


Отрисум AUC-PR для нашей задачи:

In [10]:

```
precisions, recalls, thresholds = precision_recall_curve(
    y_test, clf.predict_proba(X_test)[:,1]
)

plt.figure(figsize=(8, 6))
plt.plot(recalls, precisions, lw=2, label='PR curve')
plt.xlim([-0.05, 1.05])
plt.ylim([-0.05, 1.05])
plt.grid(ls=":")
plt.title('PR curve')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.show()
```



А также посчитаем площадь под данной PR-кривой, воспользовавшись функцией `auc`, которая считает площадь под кривыми с помощью метода трапеций:

In [11]:

```
auc_pr = auc(recalls, precisions)
print(f'AUC-PR: {auc_pr:.3f}')
```

```
AUC-PR: 0.692
```

Или же используя формулу для **Average precision**:

$$AP = \sum_n (R_n - R_{n-1}) P_n,$$

где P_n и R_n — precision и recall соответственно для n-ого порога классификации

In [12]:

```
average_precision = average_precision_score(
    y_test, clf.predict_proba(X_test)[:, 1]
)
print(f'Average precision: {average_precision:.3f}')
```

```
Average precision: 0.698
```

Обобщение метрик классификации на многоклассовый случай.

Как правило, это усреднение метрик для бинарной классификации.

Бывает два вида усреднения: микро-усреднение и макро-усреднение.

1. **Микро-усреднение:** считаем характеристики (например: TP, FP, TN, FN) для бинарной классификации (один класс против всех) и усредняем их по всем классам. После этого считаем итоговую метрику по усредненным характеристикам.
   Например, точность будет вычисляться по формуле :
   $$precision = \frac{\overline{TP}}{\overline{TP} + \overline{FP}},$$
   где $\overline{TP} = \frac{1}{K}\sum_{k=1}^{K} TP_k$
2. **Макро-усреднение:** для каждого класса считаем итоговую метрику как для бинарной классификации (один класс против всех). После этого устредняем итоговую метрику по всем классам.
   Например, точность будет вычислена как
   $$precision = \frac{1}{K}\sum_{k=1}^{K} precision_k,$$
   где $precision_k = \frac{TP_k}{TR_k + FP_k}$

Заметим, что при микро-усреднении вклад каждого класса зависит от его размера. Действительно, размер класса сильно влияет на значения матрицы ошибок: TP, FP, TN, FN.

При макро-усреднении такого эффекта не наблюдается: каждый класс вносит равный вклад в итоговую метрику, так как итоговая метрика уже не зависит от размера классов.

Поэтому если модель плохо работает с маленькими классами, то метрика, полученная при макро-усреднении будет хуже (меньше, если мы максимизируем метрику), чем при микро-усреднении (ведь при макро-усреднении маленькие классы внесли такой же вклад, что и большие).

Посчитаем микро-усреднение и макро-усреднение для F_1 -меры в задаче трехклассовой классификации

In [13]:

```
X, y = make_blobs(
    n_samples=(500, 250, 50), centers=[[2, 2], [-2, -2], [2, -2]],
    cluster_std=2.5, random_state=42
)
```

In [14]:

```
plt.figure(figsize=(8, 5))
plt.title('Сгенерированная выборка')
plt.scatter(X[:, 0], X[:, 1], c=y, alpha=0.8, cmap='Accent')
plt.grid()
plt.xlabel('Признак 1'), plt.ylabel('Признак 2')
plt.show()
```

In [15]:

```
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, stratify=y, random_state=42
)
```

In [16]:

```
clf = LogisticRegression(random_state=42)
clf.fit(X_train, y_train)
```

Out[16]:

```
LogisticRegression(random_state=42)
```

Микро-усреднение для F_1-меры:

In [17]:

```
f1_micro = f1_score(y_test, clf.predict(X_test), average='micro')
print(f'F_1 (micro): {f1_micro:.3f}')
```

```
F_1 (micro): 0.821
```

Макро-усреднение для F_1-меры:

In [18]:

```
f1_macro = f1_score(y_test, clf.predict(X_test), average='macro')
print(f'F_1 (macro): {f1_macro:.3f}')
```

```
F_1 (macro): 0.589
```

Полезные материалы:

Связь между AUC-ROC и AUC-PR (http://pages.cs.wisc.edu/~jdavis/davisgoadrichcamera2.pdf)

Визуализация AUC-ROC (http://www.navan.name/roc/)