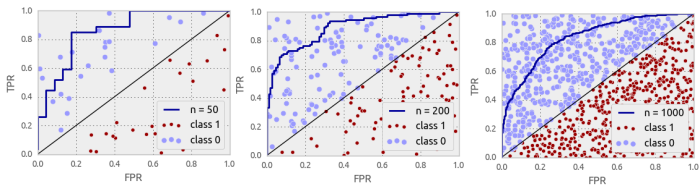
**AUC ROC**

[2017/07/28](https://alexanderdyakonov.wordpress.com/2017/07/28/auc-roc-%d0%bf%d0%bb%d0%be%d1%89%d0%b0%d0%b4%d1%8c-%d0%bf%d0%be%d0%b4-%d0%ba%d1%80%d0%b8%d0%b2%d0%be%d0%b9-%d0%be%d1%88%d0%b8%d0%b1%d0%be%d0%ba/) [alexanderdyakonov](https://alexanderdyakonov.wordpress.com/author/alexanderdyakonov/) [образование](https://alexanderdyakonov.wordpress.com/category/%d0%be%d0%b1%d1%80%d0%b0%d0%b7%d0%be%d0%b2%d0%b0%d0%bd%d0%b8%d0%b5/)[AUC](https://alexanderdyakonov.wordpress.com/tag/auc/), [качество](https://alexanderdyakonov.wordpress.com/tag/%d0%ba%d0%b0%d1%87%d0%b5%d1%81%d1%82%d0%b2%d0%be/), [тест](https://alexanderdyakonov.wordpress.com/tag/%d1%82%d0%b5%d1%81%d1%82/), [ROC](https://alexanderdyakonov.wordpress.com/tag/roc/)

Площадь под ROC-кривой – один из самых популярных функционалов качества в задачах бинарной классификации. На мой взгляд, простых и полных источников информации «что же это такое» нет. Как правило, объяснение начинают с введения разных терминов (FPR, TPR), которые нормальный человек тут же забывает. Также нет разборов каких-то конкретных задач по AUC ROC. В этом посте описано, как я объясняю эту тему студентам и своим сотрудникам…



Допустим, решается задача классификации с двумя классами {0, 1}. Алгоритм выдаёт некоторую оценку (может, но не обязательно, вероятность) принадлежности объекта к классу 1. Можно считать, что оценка принадлежит отрезку [0, 1].

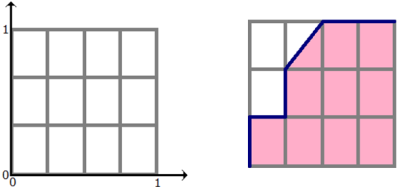
Часто результат работы алгоритма на фиксированной тестовой выборке визуализируют с помощью **ROC-кривой** (ROC = receiver operating characteristic, иногда говорят «кривая ошибок»), а качество оценивают как площадь под этой кривой – **AUC** (AUC = area under the curve). Покажем на конкретном примере, как строится кривая.

Пусть алгоритм выдал оценки, как показано в табл. 1. Упорядочим строки табл. 1 по убыванию ответов алгоритма – получим табл. 2. Ясно, что в идеале её столбец «класс» тоже станет упорядочен (сначала идут 1, потом 0); в самом худшем случае – порядок будет обратный (сначала 0, потом 1); в случае «слепого угадывания» будет случайное распределение 0 и 1.



Чтобы нарисовать ROC-кривую, надо взять единичный квадрат на координатной плоскости, см. рис. 1, разбить его на m равных частей горизонтальными линиями и на ***n*** – вертикальными, где ***m***– число 1 среди правильных меток теста (в нашем примере ***m***=3), ***n*** – число нулей (***n***=4). В результате квадрат разбивается сеткой на ***m***×***n*** блоков.

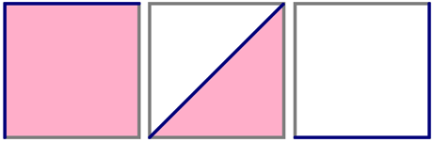
Теперь будем просматривать строки табл. 2 сверху вниз и прорисовывать на сетке линии, переходя их одного узла в другой. Стартуем из точки (0, 0). Если значение метки класса в просматриваемой строке 1, то делаем шаг вверх; если 0, то делаем шаг вправо. Ясно, что в итоге мы попадём в точку (1, 1), т.к. сделаем в сумме ***m*** шагов вверх и ***n*** шагов вправо.

Рис.1. Построение ROC-кривой.

На рис. 1 (справа) показан путь для нашего примера – это и является ROC-кривой. **Важный момент:** если у нескольких объектов значения оценок равны, то мы делаем шаг в точку, которая на ***a*** блоков выше и ***b*** блоков правее, где ***a*** – число единиц в группе объектов с одним значением метки,***b***– число нулей в ней. В частности, если все объекты имеют одинаковую метку, то мы сразу шагаем из точки (0, 0) в точку (1, 1).

**AUC ROC** – площадь под ROC-кривой – часто используют для оценивания качества упорядочивания алгоритмом объектов двух классов. Ясно, что это значение лежит на отрезке [0, 1]. В нашем примере AUC ROC = 9.5 / 12 ~ 0.79.

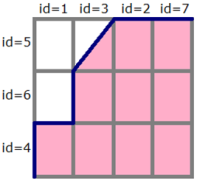
Выше мы описали случаи идеального, наихудшего и случайного следования меток в упорядоченной таблице. Идеальному соответствует ROC-кривая, проходящая через точку (0, 1), площадь под ней равна 1. Наихудшему – ROC-кривая, проходящая через точку (1, 0), площадь под ней – 0. Случайному – что-то похожее на диагональ квадрата, площадь примерно равна 0.5.

Рис. 2. ROC-кривые для наилучшего (AUC=1), случайного (AUC=0.5) и наихудшего (AUC=0) алгоритма.

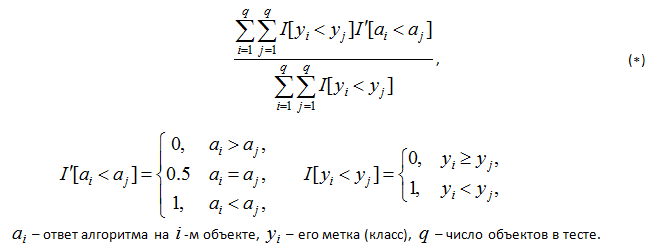
**Замечание.** ROC-кривая считается неопределённой для тестовой выборки целиком состоящей из объектов только одного класса. Большинство современных реализаций выдают ошибку при попытки построить её в этом случае

**Смысл AUC ROC**

Сетка на рис. 1 разбила квадрат на ***m***×***n*** блоков. Ровно столько же пар вида (объект класса 1, объект класса 0), составленных из объектов тестовой выборки. Каждый закрашенный блок на рис. 1 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), незакрашенный блок – паре, на которой ошибся.

Рис. 3. Каждый блок соответствует паре объектов.

Таким образом, **AUC ROC равен доле пар объектов вида (объект класса 1, объект класса 0), которые алгоритм верно упорядочил**, т.е. первый объект идёт в упорядоченном списке раньше. Численно это можно записать так:

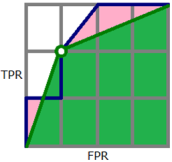


**Замечание.** В формуле (\*) все постоянно ошибаются, забывая случай равенства ответов алгоритма на нескольких объектах. Также эту формулу все постоянно переоткрывают. Она хороша тем, что легко обобщается и на другие задачи обучения с учителем.

**Принятие решений на основе ROC-кривой**

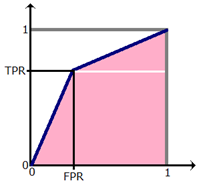
Пока наш алгоритм выдавал оценки принадлежности к классу 1. Ясно, что на практике нам часто надо будет решить: какие объекты отнести к классу 1, а какие к классу 0. Для этого нужно будет выбрать некоторый порог (объекты с оценками выше порога считаем принадлежащими классу 1, остальные – 0).

Выбору порога соответствует выбор точки на ROC-кривой. Например, для порога 0.25 и нашего примера – точка указана на рис. 4 (1/4, 2/3). см. табл. 3

Рис. 4. Выбор порога для бинаризации.

Заметим, что 1/4 – это процент точек класса 0, которые неверно классифицированы нашим алгоритмом (это называется **FPR = False Positive Rate**), 2/3 – процент точек класса 1, которые верно классифицированы нашим алгоритмом (это называется **TPR = True Positive Rate**). Именно в этих координатах (FPR, TPR) построена ROC-кривая. Часто в литературе её определяют как кривую зависимости TPR от FPR при варьировании порога для бинаризации.

Кстати, для бинарных ответов алгоритма тоже можно вычислить AUC ROC, правда это практически никогда не делают, поскольку ROC-кривая состоит из трёх точек, соединёнными линиями: (0,0), (FPR, TPR), (1, 1), где FPR и TPR соответствуют любому порогу из интервала (0, 1). На рис. 4 (зелёным) показана ROC-кривая бинаризованного решения, заметим, что AUC после бинаризации уменьшился и стал равным 8.5/12 ~ 0.71.

Рис. 5. Вычисление AUC ROC в случае бинарных ответов.

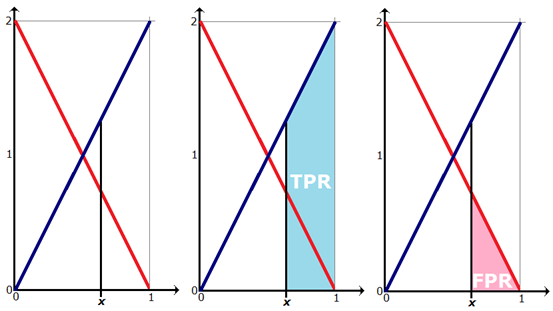
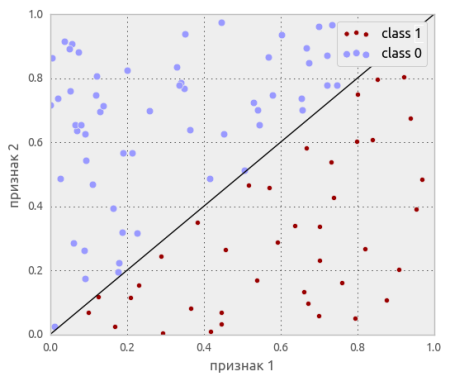
В общем случае, как видно из рис. 5 AUC ROC для бинарного решения равна

eq2.png

(как сумма площадей двух треугольников и квадрата). Это выражение имеет самостоятельную ценность и является «честной точностью» в задаче с дисбалансом классов (но об этом надо писать отдельный пост).

**Задача**

На ответах алгоритма ***a***(***x***) объекты класса 0 распределены с плотностью ***p***(***a***)=2-2***a***, а объекты класса 1 – с плотностью ***p***(***a***)=2***a***, см. рис. 6. Интуитивно понятно, что алгоритм обладает некоторой разделяющей способностью (большинство объектов класса 0 имеют оценку меньше 0.5, а большинство объектов класса 1 – больше). Попробуйте угадать, чему здесь равен AUC ROC, а мы покажем как построить ROC-кривую и вычислить площадь под ней. Отметим, что здесь мы не работаем с конкретной тестовой выборкой, а считаем, что знаем распределения объектов всех классов. Такое может быть, например, в модельной задаче, когда объекты лежат в единичном квадрате, объекты выше одной из диагоналей принадлежат классу 0, ниже – классу 1, для решения используется логистическая регрессия (см. рис. 7). В случае, когда решение зависит только от одного признака (при втором коэффициент равен нулю), получаем как раз ситуацию, описанную в нашей задаче.

Рис. 6. Распределения в модельной задаче.Рис. 7. Модельная задача (показана лишь подвыборка).

Значение TPR при выборе порога бинаризации равно площади, изображённой на рис. 6 (центр), а FPR – площади, изображённой на рис. 6 (справа), т.е.

eq3.png  
Параметрическое уравнение для ROC-кривой получено, можно уже сразу вычислить площадь под ней:  
eq4.png  
Но если Вы не любите параметрическую запись, легко получить:

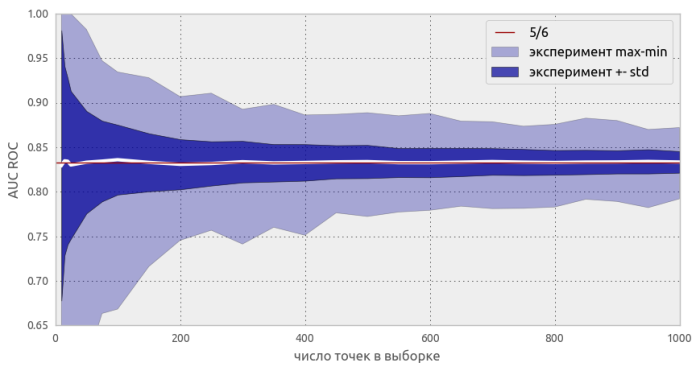
eq5.png  
Заметим, что максимальная точность достигается при пороге бинаризации 0.5 (почему?), и она равна 3/4 = 0.75 (что не кажется очень большой). Это частая ситуация: AUC ROC существенно выше максимальной достижимой точности (accuracy)!

Кстати, AUC ROC бинаризованного решения (при пороге бинаризации 0.5) равна 0.75! Подумайте, почему это значение совпало с точностью?

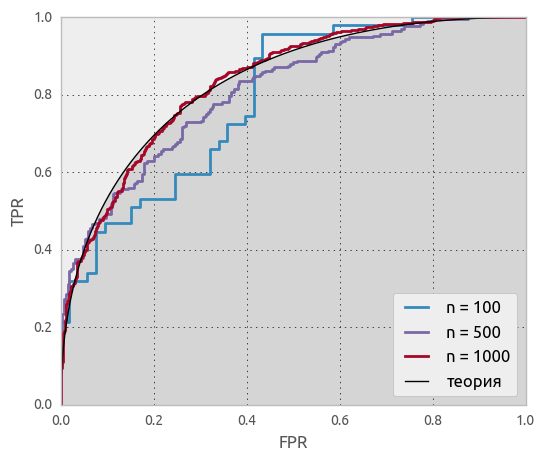
В такой «непрерывной» постановке задачи (когда объекты двух классов описываются плотностями) AUC ROC имеет вероятностный смысл: **это вероятность того, что случайно взятый объект класса 1 имеет оценку принадлежности к классу 1 выше, чем случайно взятый объект класса 0.**

eq6.png

Для нашей модельной задачи можно провести несколько экспериментов: взять конечные выборки разной мощности с указанными распределениями. На рис. 8 показаны значения AUC ROC в таких экспериментах: все они распределены около теоретического значения 5/6, но разброс достаточно велик для небольших выборок. Запомните: **для оценки AUC ROC выборка в несколько сотен объектов мала!**

Рис. 8. Варьирование AUC ROC в экспериментах.

Также полезно посмотреть, как выглядят ROC-кривые в наших экспериментах. Естественно, при увеличении объёма выборок ROC-кривые, построенные по выборкам, будут сходиться к теоретической кривой (построенной для распределений).

Рис. 9. ROC-кривые в экспериментах.

**Замечание**  
Интересно, что в рассмотренной задаче исходные данные линейны (например, плотности – линейные функции), а ответ (ROC-кривая) нелинейная (и даже неполиномиальная) функция!

**Замечание**  
Почему-то многие считают, что ROC-кривая всегда является ступенчатой функцией – лишь при построении её для конечной выборки, в которой нет объектов с одинаковыми оценками. Полезно посмотреть на интерактивную [визуализацию ROC-кривых](http://www.navan.name/roc/) для нормально распределённых классов.

**Максимизация AUC ROC на практике**

Оптимизировать AUC ROC напрямую затруднительно по нескольким причинам:

* эта функция недифференцируема по параметрам алгоритма,
* она в явном виде не разбивается на отдельные слагаемые, которые зависят от ответа только на одном объекте (как происходит в случае log\_loss).

Есть несколько подходов к оптимизации

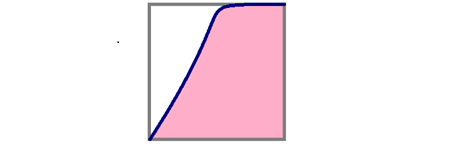
* замена в (\*) индикаторной функции на похожую дифференцируемую функцию,
* использование смысла функционала (если это вероятность верного упорядочивания пары объектов, то можно перейти к новой выборке, состоящей из пар),
* ансамблирование нескольких алгоритмов с преобразованием их оценок [в ранги](https://pandas.pydata.org/pandas-docs/version/0.18.1/generated/pandas.DataFrame.rank.html) (логика здесь простая: AUC ROC зависит только от порядка объектов, поэтому конкретные оценки не должны существенно влиять на ответ).

**Замечания**

* AUC ROC не зависит от строго возрастающего преобразования ответов алгоритма (например, возведения в квадрат), поскольку зависит не от самих ответов, а от меток классов объектов при упорядочивании по этим ответам.
* Часто используют критерий качества Gini, он принимает значение на отрезке [–1, +1] и линейно выражается через площадь под кривой ошибок:

Gini = 2 ×AUC\_ROC – 1

* AUC ROC можно использовать для оценки качества признаков. Считаем, что значения признака — это ответы нашего алгоритма (не обязательно они должны быть нормированы на отрезок [0, 1], ведь нам важен порядок). Тогда выражение 2×|AUC\_ROC — 0.5| вполне подойдёт для оценки качества признака: оно максимально, если по этому признаку 2 класса строго разделяются и минимально, если они «перемешаны».
* Практически во всех источниках приводится неверный алгоритм построения ROC-кривой и вычисления AUC ROC. По нашему описанию легко эти алгоритмы исправить…
* Часто утверждается, что AUC ROC не годится для задач с сильным дисбалансом классов. При этом приводятся совершенно некорректные обоснования этого. Рассмотрим одно из них. Пусть в задаче 1 000 000 объектов, при этом только 10 объектов из первого класса. Допустим, что объекты это сайты интернета, а первый класс – сайты, релевантные некоторому запросу. Рассмотрим алгоритм, ранжирующий все сайты в соответствии в эти запросом. Пусть он в начало списка поставил 100 объектов класса 0, потом 10 – класса 1, потом – все остальные класса 0. AUC ROC будет довольно высоким: 0.9999. При этом ответ алгоритма (если, например, это выдача поисковика) нельзя считать хорошей: в верхней части выдачи 100 нерелевантных сайтов. Разумеется, нам не хватит терпения пролистать выдачу и добраться до 10 тех самых релевантных.  
  В чём некорректность этого примера?! Главное: в том, что он никак не использует дисбаланс классов. С таким же успехом объектов класса 1 могло быть 500 000 – ровно половина, тогда AUC ROC чуть поменьше: 0.9998, но суть остаётся прежней. Таким образом, этот пример не показывает неприменимость AUC ROC в задачах с дисбалансом классов, а лишь в задачах поиска! Для таких задач есть другие функционалы качества, кроме того, есть специальные вариации AUC, например [AUC@k](http://ttic.uchicago.edu/~meshi/papers/structAUC_aistats14.pdf" \t "_blank).
* В банковском скоринге AUC\_ROC очень популярный функционал, хотя очевидно, что он также здесь не очень подходит. Банк может выдать ограниченное число кредитов, поэтому главное требование к алгоритму – чтобы среди объектов, которые получили наименьшие оценки были только представители класса 0 («вернёт кредит», если мы считаем, что класс 1 – «не вернёт» и алгоритм оценивает вероятность невозврата). Об этом можно судить по форме ROC-кривой (см. рис. 10).

Рис. 10. Форма ROC-кривой, которая соответствует случаю, когда группа объектов с самыми низкими оценками имеет метки 0.

**П.С.**

Если Вы дочитали до конца — можете попробовать пройти тест [по AUC ROC](https://docs.google.com/forms/d/e/1FAIpQLSfrZOU9TaDWIvxBabf8saK-unmijfOHwkANpARNCrVQ-g3KyQ/viewform)(авторский, публикуется впервые). Задачи теста можно обсуждать в комментариях. Любые замечание по тексту — смело пишите!