Лабораторная работа 1
Классификация

(http://www.machinelearning.ru/wiki/index.php?title=Классификация)

Классификация в машинном обучении – это причисление объекта к какой-либо категории на основании его признаков.

Подвиды задачи классификации по типу классов:

- бинарная классификация. Существут лишь два класса, простейший случай, служащий основой для многих более сложных задач.
- многоклассовая классификация. Число классов может достигать тысяч, требуются особые подходы.
- с непересекающимися классами. Объект может быть причислен только к одному классу.
- с пересекающимися классами. Объект может одновременно соответствовать нескольким классам.
- с нечеткими классами. Определяются степени принадлежности объекта ко всем имеющимся класссам, обычно в виде вещественного числа от 0 до 1.

В данном курсе лабораторных работ рассматривается только классификация с четкими непересекающимися классами.

Популярные методы классификации:

- метод ближайших соседей,
- деревья решений,
- наивный байесовский классификатор,
- машина опорных векторов.

Метод ближайших соседей (k Nearest Neighbors, или kNN)

Суть метода – находятся k ближайших соседей к рассматриваемому объекту, и объекту присваивается тот класс, который присвоен большинству его соседей. Какие соседи, такой и объект.

Близость соседей определяется расстоянием в системе координат, образованной признаками объектов.

Существует и взвешенный вариант метода – когда при голосовании учитывается не только класс соседа, но и его расстояние до объекта. Таким образом, более близкие соседи имеют больший вес, что может помочь в ситуациях, когда число соседей разных классов [примерно] одинаково.

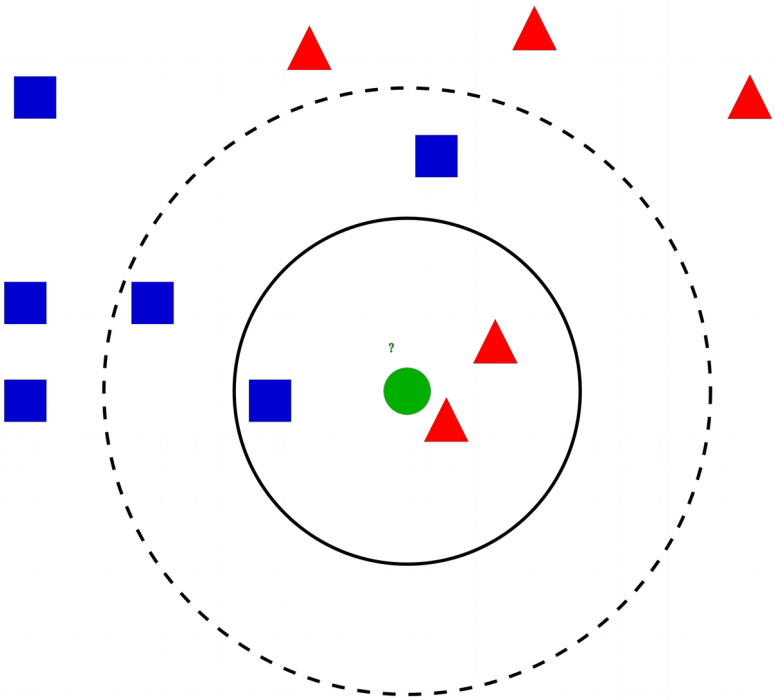
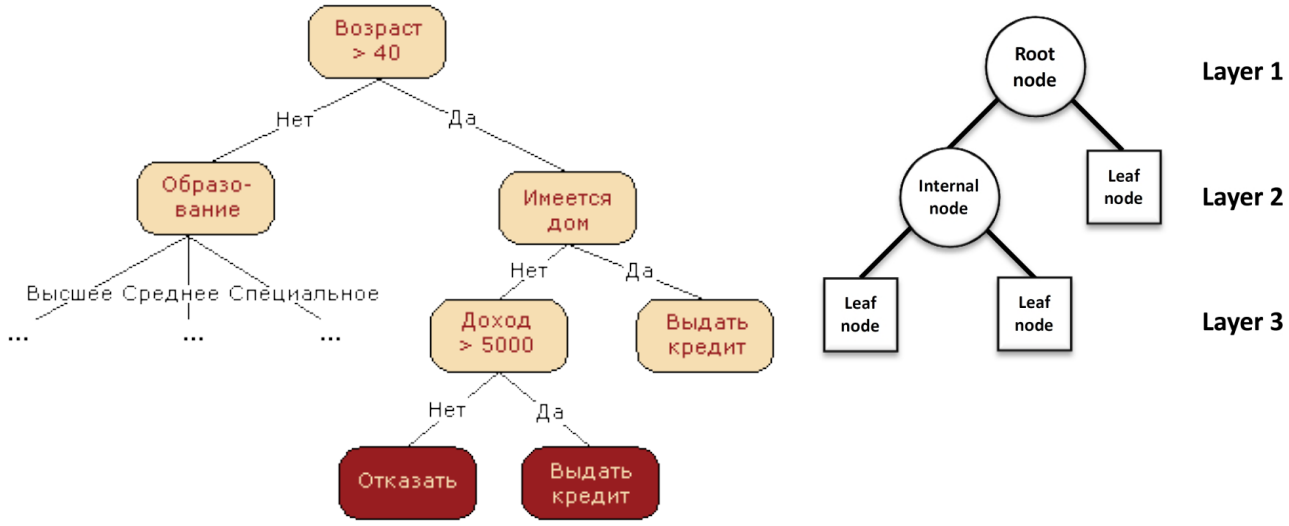


Рисунок демонстрирует следующую ситуацию: имеется два класса, обозначенных квадратами и треугольниками, и необходимо классифицировать объект, обозначенный кругом. При задании числа соседей k=3 получаем ответ, что объект-круг относится к классу треугольников, что выглядит верным решением. Однако при задании числа соседей k=5 получаем, что из пяти соседей трое – квадраты, и лишь двое – треугольники. Если использовать взвешенный метод, результатом окажется класс треугольников, потому что они гораздо ближе.

При использовании взвешенного метода, где в качестве весов используются расстояния di, значение веса может рассчитываться как wi = 1/di. Логично, ведь чем ближе объект, тем меньше расстояние и тем больше должен быть его вклад в голосование, т.е. вес.

K=3 без весов: треуг = 2, квадраты = 1. круг = max(2, 1) = 2 → класс “треугольники”	K=3 с весами: треуг = 1/0.1 + 1/0.16, квадраты = 1/0.23 круг = max(16.25, 4.3) = 16.25 →“треугольники”
K=5 без весов: треуг = 2, квадраты = 3. круг = max(2, 3) = 3 → класс “квадраты”	K=5 с весами: треуг = 1/0.1 + 1/0.16, квадраты = 1/0.23 + 1/0.44 + 1/0.38 круг = max(16.25, 9.25) = 16.26 →“треугольники

Метод решающих деревьев (Decision Trees)
(https://habr.com/en/company/ods/blog/322534/)
Решающие деревья применяются также в анализе данных и статистике. Состоят из листьев, веток (ребер) и узлов. В листьях (leaf node) записаны значения целевой функции, т.е. результаты (красненькие блоки на картинке слева), на ребрах указываются атрибуты, от которых зависит целевая функция, а в остальных узлах – атрибуты, по которым различаются случаи.
На рисунке слева показан пример инструкции по выдаче кредита в виде решающего дерева, справа – обозначены основные элементы и показано, как определяется количество слоёв.



Преимущество решающих деревьев в том, что их легко понять. В случае с инструкцией по выдаче кредита заемщику можно объяснить отказ – у него нет недвижимости и доход менее 5000. Если более сложные модели машинного обучения, способные достичь большей точности, зачастую являются “черным ящиком”, то в случае с деревьями можно получить побочный результат – полезные закономерности и взаимосвязи, что может быть очень полезно для больших объемов данных.

При построении дерева решений необходимо каким-то образом выстроить цепочку условий, приводящих к решению. Здесь можно вспомнить игру “20 вопросов”, в которой необходимо угадать загаданного знаменитого человека или персонажа, задавая вопросы, на которые можно ответить только “да” и “нет”. Очевидно, что начинать отгадывание стоит с наиболее общих вопросов, позволяющих отсечь как можно больше лишних вариантов - “Это мужчина?”, “Это вымышленный персонаж?” и т.д. Не стоит сразу спрашивать “Это француз по национальности?”, “Это Илон Маск?”.
Но как определить значимость, информативность условия в случае произвольной задачи? Описанный выше подход отгадывания интуитивно соответствует понятию прироста информации, основанного на энтропии.

Энтропия – это мера неопределенности, неупорядоченности системы, “мера хаоса”. Чем выше энтропия, тем менее упорядочена система. Обычно в информатике под энтропией понимается энтропия Шеннона, которая вычисляется следующим образом:

$$S = -\sum_{i=1}^{N} p_i \log_2(p_i), \; p_i - \textit{вероятность нахождения системы в } i - \textit{ом состоянии}.$$

Далее будем разбираться на примере нескольких записей из базы данных вышивших после крушения титаника. За признаки возьмем возраст и класс билета.

Возраст	Класс	Выжил
3	2	да
12	3	да
27	1	да
14	2	да
35	1	да
35	3	нет
19	1	нет
65	1	нет
29	2	да
31	3	да
49	3	нет
56	1	да
71	2	нет
20	3	нет

1. Посчитаем энтропию в начальном состоянии. В таблице классу “выжил” принадлежит 8 из 14 записей, классу “не выжил” - 6.

$$p_{выжил}=8/14\approx 0.5714,\ p_{не\ выжил}=6/14\approx 0.4286$$

$$S=-p_{выжил}\log_2(p_{выжил})-p_{не\ выжил}\log_2(p_{не\ выжил})$$

$$S=0.4613+0.5239=0.9852$$

Начальное значение энтропии близко к единице. Само это значение пока ни о чем не говорит.

2. Выполним разбиение на две группы по условию “возраст <= 16.5” и посчитаем энтропию для каждой группы.

2.1 Возраст <= 16.5: 3 записи.

$$p_{выжил}=3/3=1,\ p_{не\ выжил}=0$$

$$S=-1*\log_2(1)-0*\log_2(0)=0$$

Энтропия стала нулевой, что вполне логично, ведь никакой неопределенности нет: все записи относятся к классу “выжил”.

2.2 Возраст > 16.5: 11 записей.

$$p_{выжил}=5/11\approx 0.4545,\ p_{не\ выжил}=6/11\approx 0.5455$$

$$S=-0.4545*(-1.1376)-0.5455*(-0.8743)=0.9940$$

В этой группе значение энтропии стало даже больше первоначального.

Но посчитаем ***прирост информации при разбиении выборки по признаку***:

$$IG(Q)=S_0-\sum_{i=1}^{q}\frac{N_i}{N}S_i,$$

$q-$*число групп после разбиения*, $N-$*общее число элементов*,

N_i-*число элементов выборки, которые соответствуют группе* i.

$$IG(возраст\le 16.5)=S_0-\frac{3}{14}S_1-\frac{11}{14}S_2=0.9852-0-0.7810=0.2042$$

Получается, случился прирост информации. Ура!

3. Выполним разбиение группы 2.2 (возраст >= 16.5) на две подгруппы:

3.1. Возраст <= 23.5: 2 записи, обе с классом “не выжил”. S = 0.

3.2. Возраст > 23.5: 9 записей.

$$p_{выжил}=5/9\approx 0.5556,\ p_{не\ выжил}=4/9\approx 0.4444$$

$$S=-0.5556*(-0.8479)-0.4444*(-1.1701)=0.9911$$

Прирост информации:

$$IG(возраст\le 23.5)=S(возраст\ge 19)-\frac{2}{11}S_1-\frac{9}{11}S_2=0.9940-0-0.8109=0.1831$$

4. Разбиение группы 3.2 на две подгруппы по условию “возраст <= 33”

4.1 Возраст <= 33: 3/9, все выжили, S = 0.

4.2 Возраст > 33: 6/9, 2 выжили (0.3333), 4 нет (0.6667). S = 0.9183.

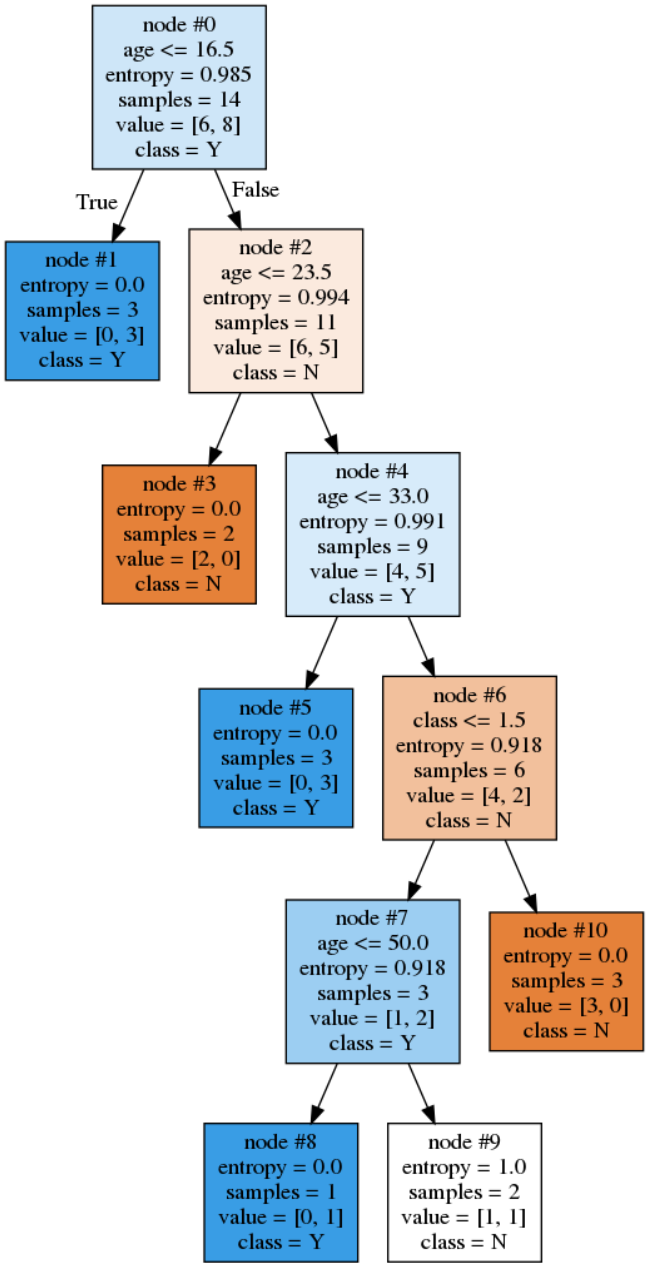
IG(age <= 33) = 0.9911 – 0 – (6/9)*0.9183 = 0.3789

5. Затем разбиение по условию “класс <= 1.5”.

5.1 Класс <= 1.5: 3/6, 2 выжили (0.6667), 1 нет (0.3333). S = 0.9183.
5.2 Класс > 1: 3/6, никто не выжил, S = 0.
IG(class <= 1.5) = 0.9183 – (3/6)*0.9183 – 0 = 0.4592

6. Затем разбиение по условию “возраст <= 50”
6.1 Возраст <= 50: 1/3, выжил, S = 0.
6.2 Возраст > 50: 2/3, 1 выжил, 1 нет. S = 1.
IG(age <= 50) = 0.9183 – 0 – (2/3)*1 = 0.2516
...и ваши мучения закончились. Почти.

А теперь запустим то же самое, но на питоне с sklearn. ***Код есть в папке “Классификация” на диске курса, файл tree_example.py.***



Возникает два логичных вопроса:

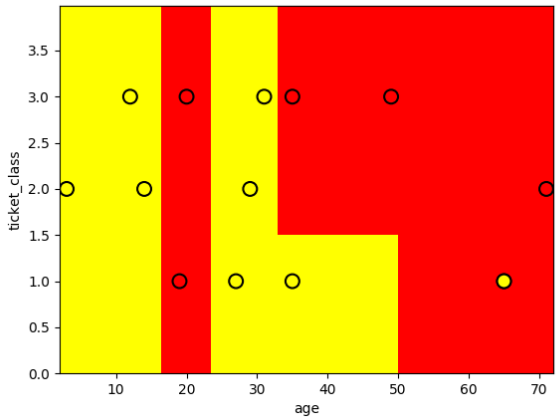
1. ~~Почему выбраны именно такие условия?~~ Что будет, если мы попробуем разбивать выборку на две группы иначе?
2. ~~Почему выбраны именно такие условия??~~ Откуда взялись эти непонятные дробные числа в условиях?

Итак, что же будет, если мы попробуем разбить выборку на две группы иначе? Допустим, мы начнем не с условия “возраст < 16.5”, а с условия “возраст < 23”:
Возраст < 23: 5/14 записей,
2 выжили (p=0.4), 3 нет (p=0.6).
S = 0.9710.
Возраст >= 23: 9/14 записей,
6 выжили (p=0.6667), 3 нет (0.3333).
S = 0.9183.
IG(age<23)=0.9852–(5/14)*0.9710–(9/14)*0.9183 = 0.0481.
Итого, прирост составил 0.0481.

Как-то мало, да? Похоже, стратегия с разделением на две группы так, чтобы хотя бы в одной из них у всех элементов был один класс, действительно лучше. А вообще, ***алгоритм на основе решающих деревьев на каждом шаге ищет такое условие для разделения на две группы, чтобы прирост информации был максимален.*** *Ну, наш алгоритм. Есть и другие способы разбиения.*
Найдите, какой ещё способ можно задавать в sklearn и какой параметр его задает?

Но почему в условиях дробные числа? Всё очень просто. Определившись с разбиением, алгоритм находит крайние значения для каждой из двух групп, а потом просто берет среднее между ними.

В первом разбиении (“возраст <= 16.5”) элементы разбились на группы по возрасту так:
группа 1: 3, 12, **14**.
группа 2: 27, 35, 35, **19**, 65, 29, 31, 49, 56, 71, 20.
условие: (14+19) / 2 = 16.5
Определив таким образом конкретные значения для условий, алгоритм разбивает всё пространство признаков на области, соответствующие классам:



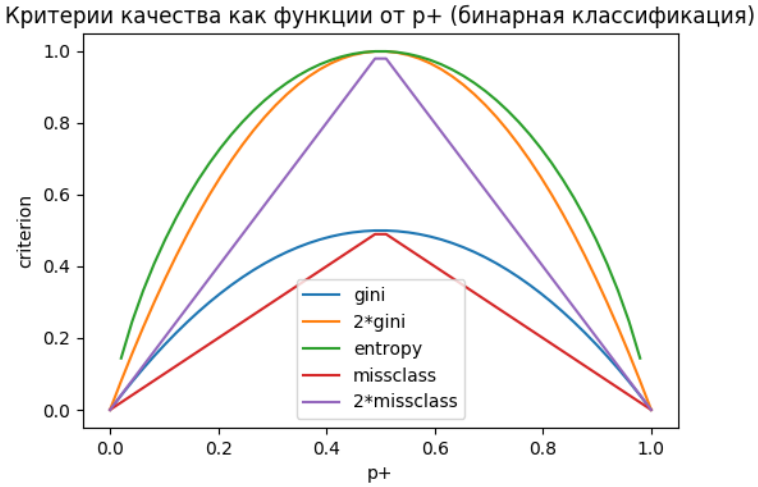
Теперь мы можем использовать обученное дерево для классификации новых элементов. Ура!

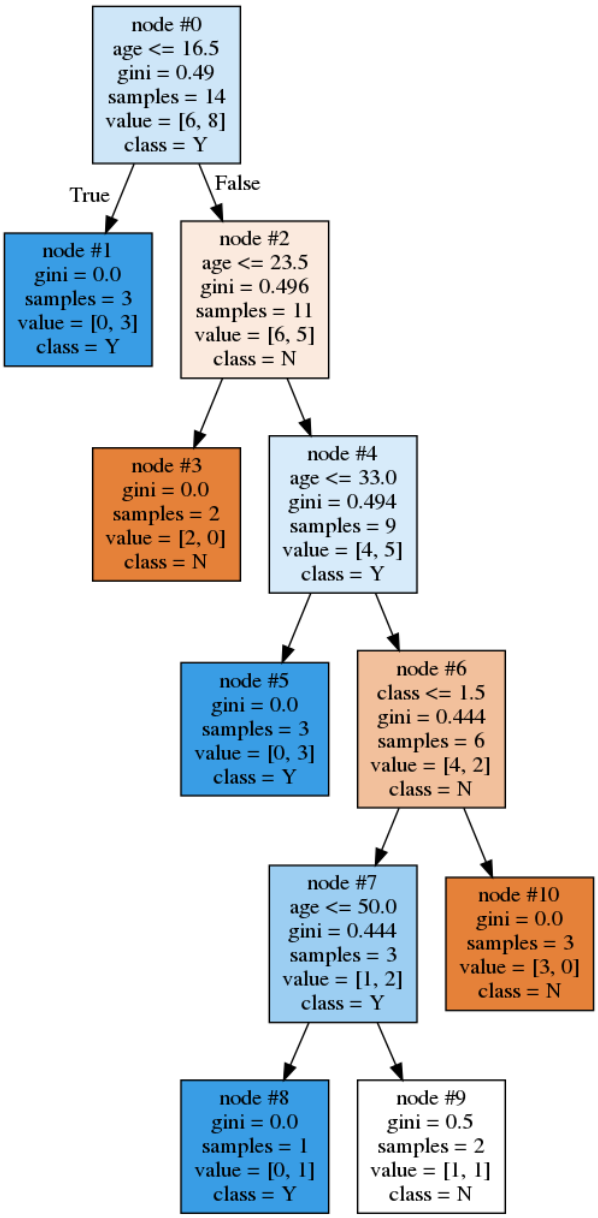
Когда данных много, нет смысла строить дерево до нулевой энтропии. Оно будет либо переобученное, либо будет долго обучаться. Обычно в таких случаях задают какое-то условие для остановки обучения: максимально допустимая глубина дерева, максимально допустимое значение энтропии, максимально допустимое число листьев и т.д.

Энтропия	$S = -\sum_{i=1}^{N} p_i \log_2(p_i)$
Неопределенность Джини	$G = 1 - \sum_{i=1}^{N} p_i^2$
Ошибка классификации	$E = 1 - \max_i p_i$

В нашем примере мы использовали понятие энтропии. Однако существуют и другие **критерии качества разбиения**. Самые популярные – энтропия и неопределенность Джини (Gini impurity). Кстати, индекс/коэффициент Джини – это совсем другое, так что не путайте.
Ещё есть ошибка классификации (misclassification error), но она используется редко. В таблице указаны их формулы.

В случае бинарной классификации, когда N=2, а p(+) - вероятность элемента иметь метку “+”, p(-) = 1 – p(+). Можно построить график значения функции критерия от p(+). ***Код на гугл-диске, Классификация → criteria_chart.py***.
Для удобства сравнения неопределенность Джини и ошибка классификации умножены на два. Видно, что неопределнность Джини и энтропия очень похожи. На практике они тоже дают практически одинаковый результат.





Вот как выглядит наше дерево, если использовать критерий Джини. Вообще-то, так же, как и с энтропией, только значения критерия другие.

Посчитаем исходное значение критерия и для первого разбиения:

$G_0 = 1 - (6/14)^2 - (8/14)^2 = 0.4898$

Возраст <= 16.5: 3 объекта, все выжили.

$G_{11} = 0$

Возраст > 16.5: 11 объектов, 5 выжило, 6 – нет.

$G_{12} = 1 - (6/11)^2 - (5/11)^2 = 0.4959$

Прирост информации ситается так же, как и раньше:

$$IG(age \le 16.5) = G_0 - \frac{3}{14} G_{11} - \frac{11}{14} G_{12} = 0.1002$$

Если удвоить, то становится очень похоже на прирост, основанный на энтропии, который мы ранее рассчитали. Он был равен 0.2042.

Наивный байесовский классификатор (Naive Bayes Classifier)

Байесовские классификаторы - это семейство вероятностных классификаторов, “основанных на принципе максимума апостериорной вероятности” (http://www.machinelearning.ru/wiki/index.php?title=Байесовский_классификатор). В них входят следующие методы:

- Наивный байесовский классификатор
- Линейный дискриминант Фишера
- Квадратичный дискриминант
- Метод парзеновского окна
- Метод радиальных базисных функций (RBF) ← *будет упомянуто в следующих лабах*
- Логистическая регрессия ← *а вот это тоже будет*

Нас интересует наивный байесовский классификатор, который, несмотря на свою простоту, до сих пор отлично работает во многих задачах, а в некоторых может конкурировать с нейронными сетями.

Классификатор основан на ***теореме (формуле) Байеса***. Вспомним её.

$$P(A|B)=\frac{P(B|A)*P(A)}{P(B)}$$

$P(A|B)$ - апостериорная вероятность гипотезы A при наступлении события B,

$P(B|A)$ - условная вероятность события B при истинности гипотезы A,

$P(A)$ - априорная вероятность события (гипотезы) A,

$P(B)$ - полная вероятность события B.

То есть, зная вероятность события-следствия B и гипотезы-причины A, а также вероятность, что за причиной A последует B, можно рассчитать вероятность того, что событие B было вызванно причиной A.

Пример. Энтомолог, изучая незнакомого жука, предполагает, что он может относиться к редкому подиду жуков, потому что у него на корпусе есть узор. В редком подвиде узор имеют 98% жуков, а среди обычных жуков – только 5%. Доля редких жуков среди всех – 0.1%.
Какова вероятность, что рассматриваемый жук относится к редкому подвиду?

Пусть B – это наличие узора, A – гипотеза, что жук редкий, C – гипотеза, что жук обычный.

$P(Редкий)=0.001$ $P(Обычный)=0.999$

$P(Узор|Редкий)=0.98$ $P(Узор|Обычный)=0.05$

$$P(Узор)=P(Узор|Редкий)*P(Редкий)+(Узор|Обычный)*P(Обычный)=0.05093$$

$$P(Редкий|Узор)=\frac{P(Узор|Редкий)*P(Редкий)}{P(Узор)}=\frac{0.98*0.001}{0.05093}=0.019\approx 2\%$$

Таким образом, вероятность того, что жук с узором – редкий, всего 2%.

“Психологические эксперименты показали, что люди часто неверно оценивают вероятность события на основе полученного опыта (апостериорная вероятность), поскольку игнорируют саму вероятность предположения (априорная вероятность). Поэтому правильный результат по формуле Байеса может сильно отличаться от интуитивно ожидаемого.”

Получается, что имея набор записей (обучающую базу) о том, какие причины (признаки) какое следствие (класс) дали, можно посчитать все необходимые значения вероятностей.
При классифицировании нового примера (нового набора признаков) необходимо для каждого класса посчитать по формуле Байеса ***P(Признаки|Класс)*** – получится что-то вроде вектора принадлежности к классам. В качестве результата берется класс с самой большой степенью принадлежности.

В частотном случае, когда проведено какое-то количество опытов, вместо вероятностей берётся частота появления события или гипотезы в виде доли от общего количества. Примерчик в частотной форме для случая бинарной классификации можно посмотреть тут: http://datareview.info/article/6-prostyih-shagov-dlya-osvoeniya-naivnogo-bayesovskogo-algoritma-s-primerom-koda-na-python/)

Для категориальны данных и таблиц частоты используется ***мультиномиальный тип*** байесовского классификатора, для численных данных – ***гауссовский тип***.
(https://scikit-learn.org/stable/modules/naive_bayes.html)

В случае гауссовского типа расчет условной вероятности происходит немного иначе.
Сначала рассчитываются мат.ожидание и дисперсия mu, var (от variance).
Затем условная вероятность того, что признак X равен значению x при том, что класс равен c, рассчитывается вот так:

$$P(X=x|Y=c)=\frac{1}{\sqrt{2\cdot Pi\cdot var}}*e^{\frac{-(x-mu)}{2\cdot var}}$$

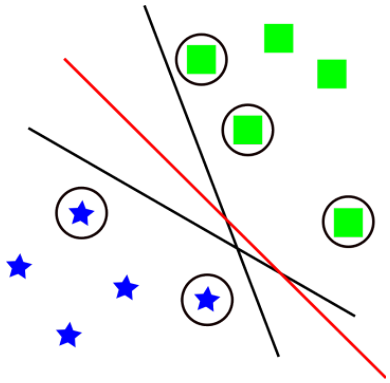
Когда понятен общий принцип, стоит указать ограничения метода.
Наивный байесовский классификатор основан на допущении, что ***признаки независимы***. То есть, все используемые признаки объекта должны вносить независимый вклад в вероятность того, что этот объект принадлежит одному из классов. А если используются не категориальные признаки (например, возраст), то алгоритм предполагает допущение, что они имеют **нормальное распределение**, что в реальных данных встречается ещё реже, чем независимость признаков. Наивно, не правда ли?
Однако, когда набор данных соответствует этим допущениям, наивный байесовский классификатор дает результат лучше и требует обучающих данных меньше, чем большинство других классических методов машинного обучения. Да и когда данные допущениям не соответствуют, справляется тоже неплохо.
В целом, это хороший стартовый алгоритм, который полезно попоробовать, прежде чем переходить к более сложным моделям.

Машина опорных векторов (Support Vector Machine)

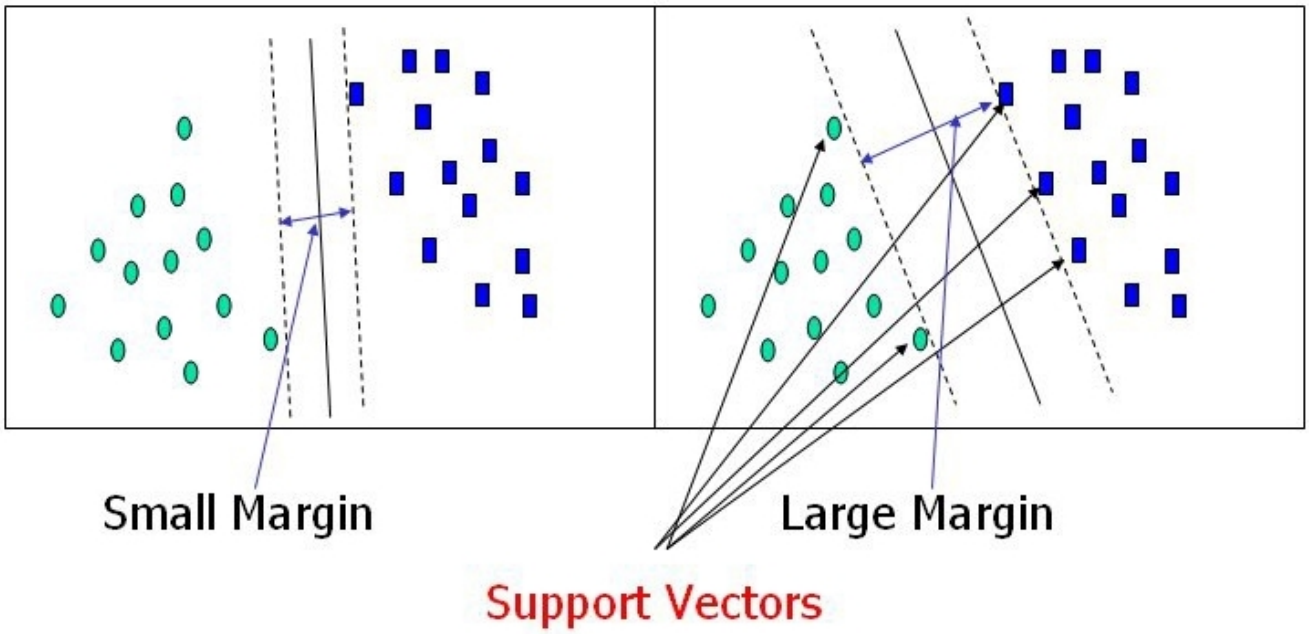
(http://web.mit.edu/6.034/wwwbob/svm-notes-long-08.pdf
https://jeremykun.com/2017/06/05/formulating-the-support-vector-machine-optimization-problem/)

Принято также название “метод опорных векторов”. Популярный алгоритм для задач классификации и регрессии.

Суть алгоритма заключается в поиске такой гиперплоскости, которая бы разделила классы наилучшим образом. На рисунке ниже показано, что между двумя классами можно провести множество разделительных линий, но оптимальной будет красная, потому что с ней достигается наибольший зазор (отступ) между самой лининей и крайними элементами классов. Такие элементы названются “опорные вектора” (***“support vectors”***), и на рисунке обведены кружочками. Зазор между ними и разделительной линией (на самом деле гиперплоскостью) называется ***“margin”***.



Очевидно, что ширина зазора зависит от выбора опорных векторов:



SVM относится к классу линейных классификаторов. Если объекты определяются двумя признаками, например, *{рост, вес}*, то есть, представляются в двумерном пространстве, то гиперплоскость будет линией. Если объекты определяются тремя признаками, то гиперплоскость будет двумерной плоскостью. Если классов больше двух, то разделяющая поверхность кусочно-линейна.
То есть, основная идея SVM – это поиск разделяющей гиперплоскости, которая бы давала максимальный зазор. Однако это не один алгоритм, а ***целое множество алгоритмов*** обучения с учителем для классификации и регрессии, которые используют описанную идею. Для нашего случая классификации мы можем пользоваться названием ***SVC – support vector classifier***.

Для желающих – немного истории:

- В ***1963*** Владимир Наумович Вапник и Алексей Яковлевич Червоненкис создали самый первый алгоритм SVM.
- В 1990 году Вапник перехал в США и в ***1992*** году совместно с Bernhard E. Boser и Isabelle M. Guyon предложил нелинейный вариант, применив ***“ядерный трюк”*** (***kernel trick***).
- Вариант алгоритма для случая линейной неразделимости (*soft margin*) был предложен в ***1993*** году Corinna Cortes и угадайте кем? Конечно же, В.Н. Вапником. Работа была опубликована в 1995 году (http://image.diku.dk/imagecanon/material/cortes_vapnik95.pdf)

SVC – бинарный классификатор, но его можно применить для многоклассовой классификации, используя вспомогательные методы. Например, можно создать набор бинарных классификаторов “один против всех”, где один класс – это i-й класс, а второй класс - все остальные классы, объединённые вместе. Или можно составить из классов все возможные пары и натренировать бинарные классификаторы на таких парах “один на один”. Кстати, попробуйте выяснить сами, какой вариант используется в sklearn – *one-versus-all* или *one-versus-one*?

Но как же происходит обучение, то есть, каким образом выбираются самые лучшие опорные вектора и строится гиперплоскость?

Начнем со случая ***линейно разделимой выборки***. Это такая выборка размерности *p*, которую можно разделить гиперплоскостью размерности *(p-1)*. Имеется в виду, без ошибок, т.е. каждая точка окажется в полуплоскости класса, к которому она относится.

Уравнение гиперплоскости имеет следующий вид:

$$w \cdot x - b = 0$$

w - вектор, перпендикулярный гиперплоскости, *b* – скаляр, характеризующий смещение (bias).
Вспомним, что произведение *w* и *x* – скалярное (dot product):

$$w \cdot x = \sum_{i=1}^{p} (w_i x_i), p - размерность$$

Также нам понадобится **||w||** - норма вектора, его магнитуда или, по-простому, длина.

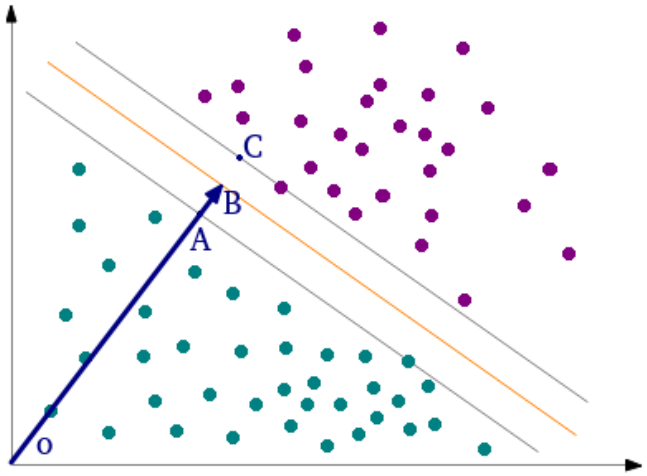
Две параллельные гиперплоскости, образующие зазор между классами, имеют уравнения:

$$w \cdot x - b = 1$$
$$w \cdot x - b = -1$$

Так как мы рассматриваем линейно разделимую выборку, то мы хотим выбрать гиперплоскости так, чтобы между ними не лежала ни одна точка обучающей выборки, т.е. для каждого обучающего примера *i*, относящегося к классу *ci* будет верно:

$$\begin{cases} w \cdot x_i - b \geq 1, ci = 1 \\ w \cdot x_i - b \leq -1, ci = -1 \end{cases} \Rightarrow \quad ci(w \cdot x_i - b) \geq 1$$

Давайте посмотрим на картинку. Видим два класса – фиолетовый и зелененький. Разделение классов определяется желтой гиперплоскостью, к которой проведен перпендикуляр OB.



Как посчитать расстояние между параллельными плоскостями? Приводить здесь всё не буду, но вот пример: https://tutors.com/math-tutors/geometry-help/distance-between-two-planes
Здесь стоит заметить, что наш случай ещё проще, так как коэффициенты при значениях координат одинаковы – это же вектор *w*. Кто не понял – попробуйте посчитать по примеру, и всё станет понятно.

Тогда длину OB можно рассчитать как $\frac{b}{||w||}$.

Ширина зазора равна $\frac{2}{||w||}$, а поскольку нам необходимо максимизировать зазор, становится очевидно, что нужно минимизировать ||w||. Минимизировать мы будем квадрат ||w||, ведь у него точно есть глобальный минимум.

Таким образом, получаем следующую задачу оптимизации:

$$\begin{cases} ||w||^2 \rightarrow min \\ ci(w \cdot x_i - b) \geq 1 \end{cases}$$

Это уже задача квадратичного программирования.

Квадратичное программирование (англ. *quadratic programming*, *QP*) — это процесс решения задачи оптимизации специального типа, а именно — задачи оптимизации (минимизации или максимизации) квадратичной функции нескольких переменных при линейных ограничениях на эти переменные. Квадратичное программирование является частным случаем нелинейного программирования.

То есть, получается, что классификация методом SVC сводится к решению популярной задачи оптимизации, для которой уже разработано множество способов решения. Однако такие общие методы часто имеют проблемы с вычислительной устойчивостью (например, ошибки округления) и слишком низкую скорость работы. В нашем конкретном случае, когда необходимо минимизировать квадратичную функцию, а ограничения линейны, задачу оптимизации можно решить с помощью теоремы Куна-Таккера.

Теорема ***(Karush 1939, Kuhn-Tucker 1951)***.
Предположим, мы имеем задачу оптимизации в $\mathbb{R}^n$ (n-мерном евклидовом пространстве) в следующей форме:

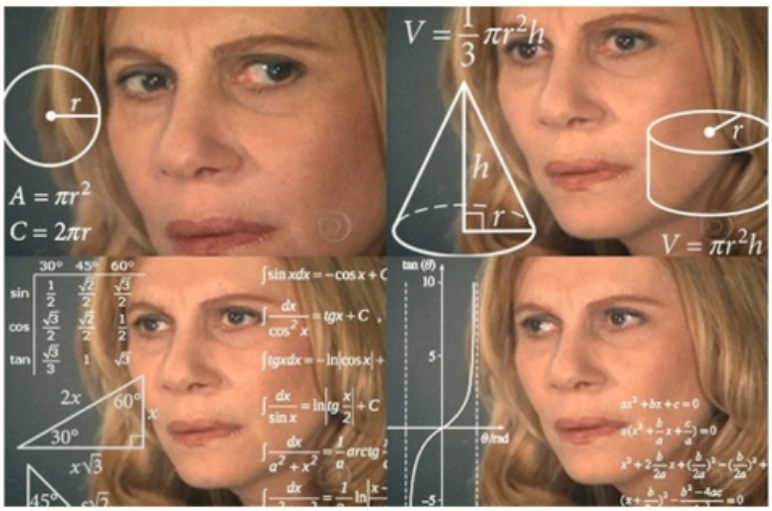
$$\begin{cases} \min f(x) \\ g_i(x) \le 0, \quad i = 1, \dots, m \end{cases}$$

где *f(x)* – дифференцируемая функция со входной переменной *x*, а g_1, …, g_m – это полином степени 1 (то есть, линейная функция) от того же самого *x*.

Пусть *z* – это локальный минимум *f*. Тогда существуют константы, называемые множителями Лагранжа (Lagrange multiplier) α_1, …, α_m такие, что следующее верно:

1. $-\nabla f(z) = \sum_{i=1}^{m} \alpha_i \nabla g_i(z)$ (Градиент в минимуме равен нулю)
2. $g_i(z) \le 0 \quad for\ all \quad i = 1, \dots, m$ (Ограничения прямой задачи удовлетворены)
3. $\alpha_i \ge 0 \quad for\ all \quad i = 1, \dots, m$ (Ограничения двойственной задачи удовлетворены)
4. $\alpha_i g_i(z) = 0 \quad for\ all \quad i = 1, \dots, m$ (Условие дополняющей нежёсткости)

Вообще на этом моменте многие источники говорят “По теореме Куна-Таккера эта задача эквивалентна двойственной задаче поиска седловой точки функции Лагранжа”.

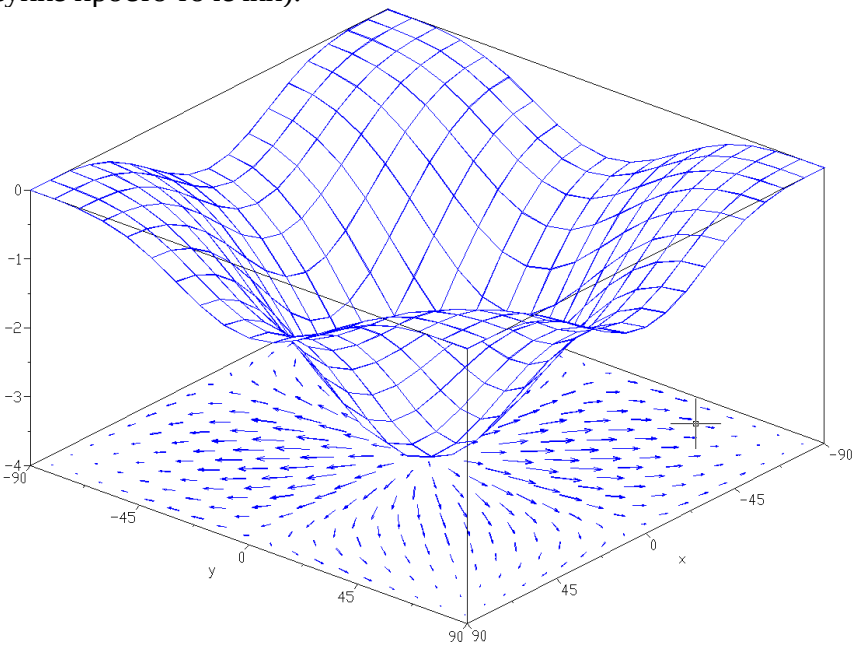


Выглядит страшновато, но наберемся решимости и попробуем разобраться, что за функция Лагранжа, зачем нам искать седловую точку и почему эта задача двойственная. И что делать с теоремой Куна-Таккера.

Итак, на всякий случай вспомним, что такое ***градиент***. Если говорить по-простому, то это вектор, который направлен по направлению наибольшего возрастания некой функции и имеет величину, равную скорости роста.
Вообще этот вектор получается из частных производных функции по каждой её переменной:

$$\nabla f(x_{1,}\ldots,x_n)=\left(\frac{\partial f}{\partial x_1},\ldots,\frac{\partial f}{\partial x_n}\right)$$

На рисунке ниже показан кусок поверхности, а в плоскости внизу показаны градиенты для функции, задающей эту поверхность. Видно, что в местах наибольшей крутизны градиенты-векторы имеют наибольшую длину, а ***в локальных минимумах/максимумах становятся нулевыми*** (на рисунке просто точечки).



Получается, что с помощью градиента можно найти минимум или максимум функции. И часто в таких задачах есть дополнительные ограничения, в самом простом случае, в виде равенств.

Например, мы хотим найти максимум функции $f(x,y,z) = xyz$, но с условием, что искомая точка $x^2+y^2+z^2=1$ лежит на окружности единичного радиуса. Это можно записать вот так:

$$\begin{cases} f(x,y,z) = xyz \\ x^2+y^2+z^2=1 \end{cases}$$

Давным-давно Ферма придумал технику для решения таких задач, которая затем была обобщена Лагранжем. Идея состояла в том, чтобы скомбинировать ограничения и функцию в единую новую функцию, градиент которой предоставлял бы достаточно информации, чтобы найти максимум. Как получить такую удобную функцию?

Сначала перепишем ограничение как $g(x,y,z) = x^2+y^2+z^2-1 = 0$, а теперь уже можем записать ***Лагранжиан*** *(он же – функция Лагранжа)* для задачи:

$$L(x, y, z, \lambda) = xyz + \lambda*(x^2+y^2+z^2-1)$$

Мы домножили ограничение на новую переменную λ и сложили с исходной функцией. Посмотрим, что будет, когда мы возьмем градиент:

$$(1)\ \frac{\partial L}{\partial x} = yz + \lambda 2x$$

$$(2)\ \frac{\partial L}{\partial y} = xz + \lambda 2y$$

$$(3)\ \frac{\partial L}{\partial z} = xy + \lambda 2z$$

$$(4)\ \frac{\partial L}{\partial \lambda} = x^2+y^2+z^2-1$$

Последнее уравнение (частная производная по λ) равно нулю, так как это исходное ограничение. А так как нам нужно, чтобы градиент был равен нулю, получается, что все остальные частные производные тоже должны быть равны нулю.

Решая это как систему уравнений, получим из (1), что $2\lambda = \frac{-yz}{x}$, поставив 2λ в (2), получим $x^2=y^2$, и подставив 2λ в (3), получим $x^2=z^2$. То есть, $x^2=y^2=z^2$, и из (4) мы знаем, что $x^2+y^2+z^2=1$, т.е. $x^2=y^2=z^2=1/3$. Максимум найден.

Вот так с помощью лагранжиана мы нашли максимум. В общем случае у нас не одно, а k ограничений в виде уравнений $g_i(x)=0$, и функция Лагранжа записывается вот так:

$$L(x, \lambda_1, \ldots, \lambda_k) = f(x) + \sum_{i=1}^{k} \lambda_i g_i(x)$$

Значит, для минимума z градиент лагранжиана будет равен нулю:

$$\nabla L(x, \lambda_1, \ldots, \lambda_k) = \nabla f(x) + \sum_{i=1}^{k} \lambda_i \nabla g_i(x) = 0$$

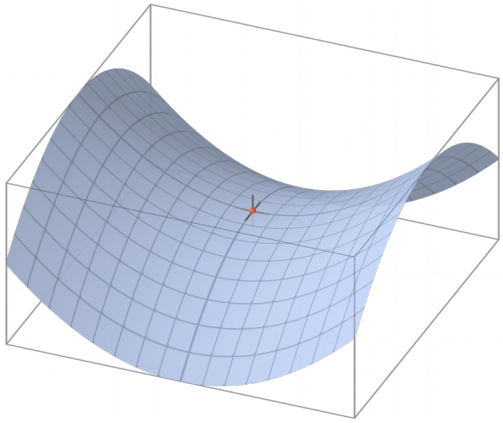
И тут посмотрим на условия 1 и 2 из теоремы Куна-Таккера:

1. $-\nabla f(z)=\sum_{i=1}^{m} \alpha_i \nabla g_i(z)$ (Градиент в минимуме равен нулю)
2. $g_i(z)\le 0 \ \ for\ all \ \ i=1,\ldots,m$ (Ограничения прямой задачи удовлетворены)

Учитывая, что α и λ – одно и то же, просто обозначены по-разному, теперь всё выглядит очень даже понятно. Но что за двойственная задача и к чему оставшиеся два условия из теоремы?

Дело в том, что для рассмотренной задачи минимизации (прямой задачи) может быть не одно, а множество решений. Как же нам выбрать оптимальное? Для этого как раз можно решить двойственную задачу максимизации. Решение двойственной задачи максимизации гарантирует, что найденный максимум – это нижняя граница решения прямой задачи, т.е. минимума.

Чтобы интуиция такого подхода стала понятнее, стоит вспомнить седло.



Такая поверхность выпуклая в одном направлении и вогнутая в другом. Градиент в седловой точке равен нулю. То есть, в одной плоскости седловая точка будет как бы “минимумом”, а в другой плоскости – как бы “максимумом”.

Условие 3 ?

Условие 4 требует, чтобы константа α_i или ограничительная функция g_i были равны нулю. Или даже обе:

4. $\alpha_i g_i(z) = 0 \quad for\ all \quad i = 1, \ldots, m$ (Условие дополняющей нежёсткости)

Продолжение...

ЗАДАНИЕ

Наконец, задание!

Так как данная работа – первая, то помимо заданий, непосредственно связанных с классификацией, будет много заданий, связанных с изучением базы данных.

(!!!) При построении графиков обязательно подписывайте оси

(!!!) Можно взять другую базу данных, об этом в самом конце

База данных:

New York City Airbnb Open Data

https://www.kaggle.com/dgomonov/new-york-city-airbnb-open-data

На гугл-диске курса в папке *Datasets → AB_NYC_2019.csv*

Описание столбцов-признаков:

id	ID жилья (объявления)
name	название
host_id	ID владельца
host_name	имя владельца
neighbourhood_group	район
neighbourhood	соседство, одно из многочисленных для каждого района
latitude	координаты широты
longitude	координаты долготы
room_type	тип жилья
price	цена
minimum_nights	минимальное кол-во ночей
number_of_reviews	число отзывов
last_review	последний отзыв
reviews_per_month	среднее число отзывов за месяц
calculated_host_listings_count	число единиц жилья (объявлений) на владельца
availability_365	количество дней, когда жилье доступно для бронирования (за год)

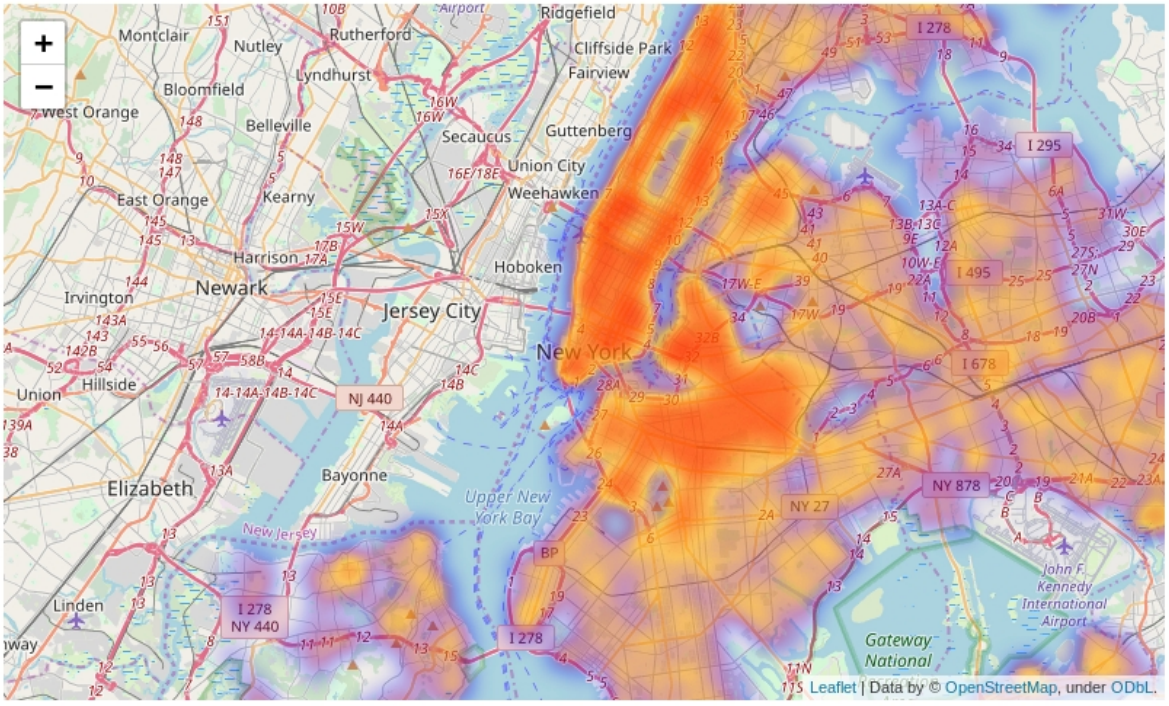
Задачи:

Часть 1 – работа с данными:

(библиотеки matplotlib или seaborn, если хотите что-то другое – пожалуйста)

1. Сколько записей в базе?
2. Постройте гистограммы всех признаков.
3. Есть ли значения, которые есть смысл отбросить? Есть ли некорректные значения? Если да – укажите, какие, сколько, и что вы с ними собираетесь делать.
4. Постройте график, где по осям x и y будут широта и долгота, а цветом помечены районы. *Это задание дано, чтобы научиться строить такие графики, ну и красиво получается :) Особые энтузиасты могут нарисовать маску плотности предложений на карте:*

```
import folium
from folium.plugins import HeatMap
m=folium.Map([40.7128,-74.0060],zoom_start=11)
HeatMap(data[['latitude','longitude']].dropna(),radius=8,gradient={0.2:'blue',0.4:'purple',0.6:'ora
nge',1.0:'red'}).add_to(m)
display(m)
```



5. Теперь подумайте, могут ли какие-то признаки быть взаимозависимы? Если да, то какие? А после этого постройте матрицу корреляции, но с цветовой индикацией, пожалуйста. Совпало с вашими догадками?
6. Возможно, многие уже задумались о насущном.. Постройте график, где по осям x и y будут широта и долгота, а цветом помечена цена. Те, кто в пункте 3 делал это с seaborn – есть смысл попробовать метод *pandas.DataFrame.plot*, потому что там цветовая индикация непрерывная, а seaborn разбивает на несколько жалких диапазонов, и при неравномерном распределении ничего не понятно. *Подсказка: если какое-то относительно небольшое количество записей сильно выбивается из общей кучи, можно их не рассматривать, нас пока интересуют общие закономерности, а не исключения.*
7. Составьте рейтинг слов из названий по популярности (частоте появления) и укажите 25 самых популярных с числом их появлений. *Не уверена, что вам пригодится эта информация, но особые энтузиасты могут сделать из полученных слов красивый вдохновляющий плакат с помощью библиотеки wordcloud*.

Всё! С данной частью закончено. Теперь вы умеете визуализировать данные и выполнять небольшой предварительный их анализ. Если вы проявили инициативу и нашли в данных что-то интересное – не стесняйтесь, прикладывайте к отчету.

Часть 2 – классификация:

Эта часть задания будет дана в более свободной форме, но она самая важная. Кому-то уже, возможно, хочется попредсказывать цену, и когда мы дойдем до регрессии, сделаем это, однако данная работа – по классификации.

Поэтому предлагаю предсказывать соседство по заданным параметрам – цена, тип жилья и т.д.

Как вариант, можно разбить цену на несколько диапазонов (не менее 10), получив тем самым несколько классов, и предсказывать диапазон цен.

Цель задания – добиться как можно большей точности по каждому из четырех алгоритмов классификации. В отчете приведите описание того, как вы в итоге пришли к полученной точности, что вы делали для того, чтобы её повысить, и почему вы считаете, что сделали всё, что могли. Работать необходимо как с базой, так и с параметрами алгоритма.

(!!!) Для определения точности используйте кросс-валидацию. Подсказка: признаки можно использовать не все, и можно даже их менять.

Итак, в части 2 вам необходимо:

1. Реализовать решение задачи классификации (просто как-то) каждым из рассмотренных алгоритмов.
2. Для каждого алгоритма составить список/таблицу настроечных параметров, описать их смысл, в каких случаях что используется и как это влияет (пусть предположительно) на результат.
3. С помощью предобработки данных и настройки параметров добиться максимально точного результата для каждого алгоритма, при этом сравнивать ещё время обучения и время работы моделей. Описать логику поиска лучших моделей, привести рассуждения, таблицы сравнения моделей, по которым вы определили лучший вариант, и т.д.
4. Сформулировать рекомендации по решению задачи классификации рассмотренными алгоритмами – в каких случаях какие алгоритмы и настроечные параметры нужно использовать.

Заметка по базам данных

Если у вас есть собственные данные, по которым нужно что-то классифицировать, либо вы нашли интересную базу данных – напишите мне на почту (rtc.machinelearning@gmail.com) письмо с темой “Подтверждение своей базы” и описанием базы. Я посмотрю и сообщу, подойдет она или нет.

Есть ещё вот такие базы:

- Титаник

Если возьмете эту базу, я ожидаю очень хорошего и подробного отчета, потому что по ней много материала.

https://www.kaggle.com/c/titanic

Данные можете скачать там же, либо взять с диска курса

- Распознавание вида активности человека по данным с датчиков смартфона.

Не факт, что удастся хорошо выявить какие-то зависимости и классифицировать с высокой точностью, поэтому это только для особых энтузиастов. Если не получится, придется пределывать с другой базой.

http://archive.ics.uci.edu/ml/datasets/Human+Activity+Recognition+Using+Smartphones