Случайность. Наивный байесовский классификатор

Цель занятия

В результате обучения на этой неделе вы:

- повторите основные понятия теории вероятностей;
- вспомните понятие условной вероятности, дискретных и непрерывных случайных величин;
- познакомитесь с центральной предельной теоремой и теоремой Байеса;
- узнаете, в каких задачах можно применить наивный байесовский классификатор;
- поймете назначение и принцип работы библиотеки sklearn.

План занятия

Конспект занятия

1. Вероятность. Свойства вероятности

Биномиальные коэффициенты

Пусть есть выборка из n объектов.

Вопрос 1. Как оценить количество подвыборок размера k? Сколькими способами мы можем выбрать выборку размера k из n объектов?

n^k способов выбора k объектов из n объектов с перестановкой (порядок имеет значение).

Вопрос 2. Каким образом можно упорядочить выборку из n объектов? Считаем, что все объекты разные.

$n! = n \cdot (n-1) \ldots 2 \cdot 1$ — способы упорядочивания (перестановки) n объектов = выбор n объектов из n объектов без перестановок (порядок имеет значение).

Первый объект можно поставить на любое из n мест. Второй — на любое, кроме уже занятого и т. д.

Вопрос 3. Сколько существует способов выбрать k объектов из n объектов без замены, если порядок не имеет значения?

Биномиальный коэффициент C_k^n (читается как «*C* из *n* по k»), или просто $\binom{n}{k}$ («*n* по k»):

$$C_k^n = \frac{n!}{k!(n-k)!} \qquad (1)$$

Почему формула (1) имеет такой вид? У нас всего всевозможных выборок размера *k* из *n*. *n* можно упорядочить *n!* раз. Порядок для нас не имеет значения, поэтому делим на *k!* (сколько раз можно упорядочить подвыборку из *k* элементов). А также делим на $(n-k)!$ — это то, что осталось.

Биномиальная теорема объясняет, как посчитать *n*-ю степень от суммы:

$$(a+b)^n = \sum_{k=0}^{n} \binom{n}{k} a^k b^{n-k} = \underbrace{\binom{n}{0}}_{=1} a^0 b^n + \underbrace{\binom{n}{1}}_{=n} a^1 b^{n-1} + \underbrace{\binom{n}{2}}_{=\frac{n(n-1)}{2}} a^2 b^{n-2} + \cdots + \underbrace{\binom{n}{n}}_{=1} a^n b^0$$

Мы можем говорить про вероятность с точки зрения дискретного пространства, где у нас есть конечное множество возможных исходов. Тогда у нас каждый исход — это какой-то выбор из этого пространства.

Биномиальные коэффициенты очень неплохо помогают при подсчете вероятностей в дискретном случае.

Статистика Бозе-Эйнштейна

Вопрос 4. Как оценить количество вариантов, если есть n объектов на выбор, и мы делаем k выборов, а порядок выбора все еще не имеет значения?

Эта проблема рассматривалась физиками в 1920-х годах и привела к открытию так называемой статистики Бозе-Эйнштейна.

Кстати, это эквивалентно нахождению количества решений $(x_1, \ldots, x_n)$ для уравнения $x_1 + x_2 + \ldots + x_n = k$, где x_i — неотрицательные целые числа.

Общее определение вероятности

Вероятностное пространство состоит из выборочного пространства *S* и функции вероятности *P*, которая принимает событие $A \subset S$ в качестве входных данных и возвращает *P(A)* — действительное число из отрезка [0,1]. *P* отвечает следующим аксиомам:

$$P(\emptyset) = 0, \qquad P(S) = 1$$

Если $A_1, A_2, \ldots$ являются непересекающимися (= взаимоисключающими, $A_i \cap A_j = \emptyset,\ i \neq j$) событиями, то $P\left(\bigcup_{j=1}^{\infty} A_j\right) = \sum_{j=1}^{\infty} P(A_j)$.

Пример 1. В наивной интерпретации рассматриваются камешки (элементарные события) одинаковой массы. События представляли собой груды камешков (возможно, перекрывающиеся).

В общем случае могут быть камешки различной массы: как счетное множество камешков, так и несчетное, если их масса в сумме равна 1.

Пример 2. Количество секунд дискретное, значит, счетное.

Может быть даже несчетное количество выборочных пространств — областей на прямой *R* или на плоскости R^2 и т. д.

Рассмотрим подходы к статистике и к вероятности.

Частотный (фишеровский) подход предполагает, что вероятность есть долгосрочная частота при большом количестве повторений эксперимента.

Например, какова вероятность выпадения «орла» при подбрасывании монетки при большом количестве подходов? Это частотная вероятность. Если в какой-то момент от монетки отпилим кусок, статистика поменяется.

Байесовский подход предполагает, что вероятность есть степень уверенности в рассматриваемом событии.

Оба этих подхода дополняют друг друга, и оба будут полезны. В обоих случаях приведенные выше аксиомы приводят к одним и тем же свойствам вероятности.

Свойства вероятности

Вероятность обладает следующими свойствами для любых событий A и B:

1. $P\left(A^c\right) = 1 - P(A)$, где A^c — отрицание события A.
2. Если $A \subset B$, то $P(A) \leq P(B)$
3. $P(A \cup B) = P(A) + P(B) - P(A \cap B)$

Доказательство:

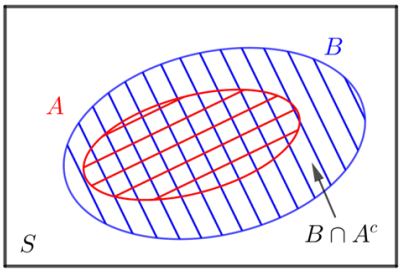
1. $P\left(A^c\right) = 1 - P(A)$, где A^c — отрицание события A.

Так как A и A^c не пересекаются и их объединение — S, из второй аксиомы следует:

$$P(S) = P(A \cup A^c) = P(A) + P(A^c)$$

Согласно первой аксиоме, $P(S) = 1$, то $P(A) + P(A^c) = 1$.

2. Если $A \subset B$, то $P(A) \leq P(B)$.



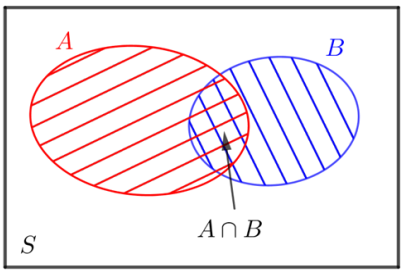
Если $A \subset B$, то $B = A \cup (B \cap A^c)$.

Так как A и $B \cap A^c$ не пересекаются, из второй аксиомы следует:

$$P(B) = P(A \cup (B \cap A^c)) = P(A) + P(B \cap A^c).$$

Если $P \geq 0$, $P(B \cap A^c) \geq 0$, то $P(B) \geq P(A)$.

3. $P(A \cup B) = P(A) + P(B) - P(A \cap B)$.



Если $A \cup B = A \cup (B \cap A^c)$, то $P(A \cup B) = P(A) + P(B \cap A^c)$.

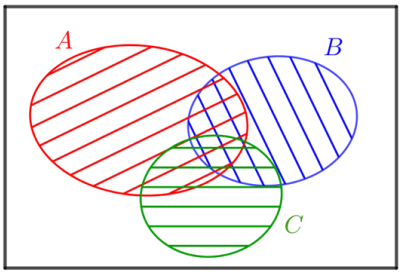
Так как $B \cap A$ и $B \cap A^c$ не пересекаются, то:

$$P(B \cap A) + P(B \cap A^c) = P(B).$$

Так как $P(B \cap A^c) = P(B) - P(A \cap B)$, то:

$$P(A \cup B) = P(A) + P(B) - P(A \cap B).$$

Рассмотрим 3 события. Интуиция подсказывает



(парные пересечения считаются дважды, тройные — три раза):

$$P(A \cup B \cup C) = P(A) + P(B) + P(C) - P(A \cap B) - P(A \cap C) - P(B \cap C) + P(A \cap B \cap C)$$

Формула включений-исключений

Для любых событий $A_1, \ldots, A_n$:

$$P\left(\bigcup_{i=1}^{n} A_i\right) = \sum_i P(A_i) - \sum_{i<j} P(A_i \cap A_j) + \sum_{i<j<k} P(A_i \cap A_j \cap A_k) - \ldots + (-1)^{n+1} P(A_1 \cap \ldots \cap A_n).$$

Эту формулу хорошо использовать, если есть некоторая симметрия событий A_i. Если симметрии нет, то сначала лучше попробовать другие инструменты.

Вероятность для независимых событий есть вероятность одного события умножить на вероятность другого.

Например, проблема соответствия де Монмора. Рассмотрим хорошо перетасованную колоду из n карт, пронумерованных от 1 до n. Вы переворачиваете их одну за другой, произнося при этом числа от 1 до n. Вы выигрываете, если в какой-то момент число, которое вы называете, совпадает с числом на перевернутой карте. Какова вероятность выигрыша?

Решение. Пусть A_i — это такое событие, что *i*-тая карта имеет номер *i*, написанный на ней. Нас интересует $P(A_1 \cup \ldots \cup A_n)$, потому что нам достаточно один раз угадать.

1. $P(A_i) = \frac{1}{n}$.
2. $P(A_i \cap A_j) = \frac{(n-2)!}{n!} = \frac{1}{n(n-1)}$. У нас сразу две карты лежат на своих местах. Сколькими способами мы можем упорядочить нашу выборку? — $n!$ Сколькими способами мы можем упорядочить нашу выборку, кроме этих двух карт? — $(n-2)!$
3. $P(A_i \cap A_j \cap A_k) = \frac{1}{n(n-1)(n-2)}$ и т. д.

В формуле включения-исключения будет n раз, включающих 1 событие; C_2^n раз, включающих 2 события; C_3^n для 3 событий и т. д. Таким образом,

$$P\left(\bigcup_{i=1}^{n} A_i\right) = \frac{n}{n} - \frac{C_2^n}{n(n-1)} + \frac{C_3^n}{n(n-1)(n-2)} - \ \dots \ + (-1)^{n+1}\frac{1}{n!} = 1 - \frac{1}{2!} + \frac{1}{3!} - \ \dots \ + (-1)^{n+1}\frac{1}{n!}$$

Это похоже на ряд Тейлора, где $e^{-1} = 1 - \frac{1}{1!} + \frac{1}{2!} - \frac{1}{3!} + \dots$, только имеется еще одна единица и знаки расположены в другом порядке. Тогда (для большого n):

P («выигрыша») $= 1 - 1/e \approx 0.63.$

Немного похоже на «парадокс дней рождения». При определенном наборе множества людей вероятность того, что у двух человек дни рождения в один и тот же день растет при небольших объемах выборки.

2. Условная вероятность. Теорема Байеса

Условная вероятность

Условная вероятность — вероятность некоторого события при условии наступления других событий.

Пример 1. Что, если события вообще друг от друга не зависят? Например, событие А — мы взяли с собой зонт, событие С — кролик съел морковку.

$P(A|C) = P(A|C^C) = P(A)$ — события друг с другом никак не связаны.

Пример 2. Событие А — мы взяли с собой зонт. Событие В — на улице по прогнозу сегодня идет дождь. Вероятность А при условии В:

$$P(A|B) = \frac{P(AB)}{P(B)} \qquad (2)$$

А и В **независимы**, если вероятность их совместного наступления:

$P(AB) = P(A \cap B) = P(A)P(B).$

В примере 1 события независимы, поэтому:

$$P(A|C) = P(A|C^C) = P(A) = \frac{P(AC)}{P(C)} = \frac{P(A)P(C)}{P(C)}.$$

Введем формулу **полной вероятности**. Пусть у нас есть дискретная случайная величина, которая принимает конечное число исходов. По сути, это задача классификации. Там у нас метка класса, это некоторая дискретная случайная величина.

Пример 3. Пусть у нас есть три вуза: МФТИ, МГУ, МГТУ им. Баумана: C_1, C_2, C_3 соответственно. Мы хотим посчитать вероятность того, что студент сдавал физику независимо от вуза:

$$P(A|y = C_1), P(A|y = C_2), P(A|y = C_3).$$

Просто сложить эти вероятности между собой кажется некорректным, так как вероятности условные. Мы не знаем, с какой вероятностью каждый человек поступал в каждый из этих вузов. Нам поможет формула полной вероятности:

$$P(A) = \sum_{k=1}^{3} P(A|y = C_k) \cdot P(y = C_k).$$

$P(y = C_k)$ — вероятность того, что студент поступал в данный вуз.

При этом $y = C_i \cap C_j = \varnothing, \forall\, i \neq j$.

$$P(A) = \sum_{k=1}^{3} P(A|y = C_k) \cdot P(y = C_k) = \sum_{C_k} \frac{P(AC_k)}{P(C_k)}.$$

Формула полной вероятности:

$$P(A) = \sum_{C_k} \frac{P(AC_k)}{P(C_k)} \qquad (3)$$

Теорема Байеса

Из формулы (2):

$P(A|B) = \frac{P(AB)}{P(B)} = \frac{P(BA)}{P(B)}$,

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

Тогда:

$P(A|B) \cdot P(B) = P(B|A) \cdot P(A),\ \forall A,\ B.$

Тогда **формула Байеса** или **теорема Байеса**:

$$P(A|B) = \frac{P(B|A) \cdot P(A)}{P(B)} \quad (4)$$

Формула (4) работает как для дискретных случайных величин, так и для непрерывных. В случае непрерывных от вероятности перейдем к плотности случайной величины.

Свойства формулы Байеса

Формула (4) позволяет пересчитывать какую-либо статистику при учете новых данных.

$P(A) = Prior$ — априорное распределение, наши априорные предположения.

$P(A|B) = Posterior$ — то, что мы получили, когда узнали что-то новое, апостериорное распределение.

$P(B|A) = Likelihood$ — те данные, которые пронаблюдали, правдоподобие.

Мы воспользуемся этим при построении наивного байесовского классификатора.

3. Наивный байесовский классификатор

Модель наивного байесовского классификатора

Модель наивного байесовского классификатора стоит особняком от простейших моделей, но является очень полезной и необходимой для освоения.

Пусть есть некоторое пространство признаков $L = \{x_i, y_i\}_{i=1}^{n}$, в котором находится наш объект $x_i \in R^p$, и для каждого объекта существует некоторая метка класса $y_i \in \{C_1, \dots, C_k\}$.

Будем сразу говорить о многоклассовой классификации, так как наивный байесовский классификатор, как и kNN, работает и в бинарном случае, и в мультиклассовом случае абсолютно одинаково.

$x_i \in R^p$, $y_i \in \{C_1, \dots, C_k\}$ — наша обучающая выборка.

Все, что нам нужно, чтобы построить качественный и эффективный классификатор, это вспомнить формулу Байеса (4).

Что мы хотим получить в задаче классификации? Мы хотим оценить вероятность того, что метка класса, допустим, равна 1 при условии того, что мы наблюдали некоторый x.

Событие A в данном случае $y_i = C_k$ — k-й класс.

B — это то, что мы пронаблюдали конкретный объект с некоторыми значениями признаков. Переформулируем:

$P(y_i = C_k|x_i) = \frac{P(x_i|y_i=C_k)P(y_i=C_k)}{P(x_i)}$ — формула Байеса в нашей постановке (5)

Мы можем оценить вероятность для каждого из классов и выбрать наиболее вероятную метку класса.

Как это считать? Воспользуемся предположением, что все признаки независимы. Это и есть наивное предположение, которое делает наивный байесовский классификатор наивным.

Зачем нам независимость признаков? Мы можем факторизовать вероятность и получить произведение вероятностей для каждой из величин отдельно. То же самое можем делать с признаками. То есть мы наблюдаем множество реализаций различных случайных величин:

$$P(x_i|y_i = C_k) = \prod_{l=1}^{p} P(x_i^l|y_i = C_k),$$

где *l* — номер признака.

Пример. Пусть объект описан тремя признаками — рост, вес и пол.

Какова вероятность пронаблюдать третьеклассника с ростом 170 см? Низкая. Какова вероятность пронаблюдать третьеклассника с весом 70 кг? Тоже низкая. Какова вероятность пронаблюдать третьеклассника мужского пола? Примерно 50%.

Получается, что общая вероятность пронаблюдать мужчину ростом 170 см и весом 70 кг в третьем классе — это $1\% \cdot 1\% \cdot 50\% = 5 \cdot 10^{-5}$.

Если взять логарифм от произведения, то получим сумму логарифмов, что гораздо удобнее.

Оптимальная метка класса

$$C^* = \arg\max_k P(y_i = C_k|\mathbf{x_i}) \quad (6)$$

Что делать с $P(y_i = C_k)$ и со знаменателем в формуле (5)?

$P(y_i = C_k)$ — какова вероятность пронаблюдать объект k-го класса из нашей выборки.

Например, частота встречаемости объектов в тех или иных классах. Это будет нормированная частота встречаемости объекта данного класса выборки.

Что делать со знаменателем $P(x_i)$? Посмотрим на формулу (6). Оптимальная метка класса — это та, которая наиболее вероятна. Наша задача максимизации (6) не зависит от знаменателя, поэтому от него можно просто избавиться.

Где используется наивный байесовский классификатор

Фильтрация спама в электронной почте. Для векторизации слов используется «мешок слов» (bag of words). Считается количество всех слов. То есть каждое слово — отдельный счетчик; и мы считаем, какие слова сколько раз попали в текст.

Получаем большой вектор размера всех слов в языке, в котором практически везде стоят нули, а в некоторых местах стоят счетчики. На этом признаковом описании наивный байесовский классификатор неплохо работал.

Гипотеза о независимости признаков (в данном случае это счетчики) выполнялась довольно неплохо. То есть счетчики слабо зависели друг от друга.

В наше время для фильтрации спама используют модели типа BERT.

Как наивный байесовский классификатор работает на практике

У нас есть формула (5) и есть выборка. Мы можем попробовать построить эмпирическую функцию распределения. Например, гистограмму, и использовать ее как оценку функции распределения.

Мы можем использовать некоторую гипотезу. Например, предположить, что наше распределение, из которого пришли все x, все признаки по отдельности, нормальное. Это работает, если все значения континуальные. И мы для каждого из признаков по отдельности будем искать оптимальные значения параметров распределения, из

которых они могли прийти. Например, посчитать среднее и дисперсию. Нам этого достаточно, чтобы полностью описать нормальное распределение.

Стоит обратить внимание, что для разных признаков можно использовать различные распределения. Для континуального признака можно использовать нормальное распределение, для бинарного — распределение Бернулли. Для третьего признака — распределение Пуассона. Это зависит от того, какой природы признаки и данные мы используем.

Наивный байесовский классификатор тем не менее неплохо восстанавливает вероятности для каждой из меток класса.

4. Реализация байесовского классификатора

Для реализации байесовского классификатора подключим библиотеки:

```
import numpy as np
import pandas as pd
import matplotlib
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import datasets
```

В sklearn будем пользоваться классическим датасетом Ирисов Фишера.

Sklearn

Из sklearn можно загружать различные датасеты. Не только искусственно сгенерированные, но и вполне реальные. Например, load_iris возвращает датасет Ирисы Фишера. Это словарь, в котором есть несколько слов:

```
dataset = datasets.load_iris()
```

```
print(dataset.DESCR)
```

В словаре 4 признака, 150 объектов, сбалансированные классы — 3 класса, все числовые.

Посмотрим на dataframe с помощью библиотеки pandas:

```
# for now you don't need to understand what happens in this code – just look at the table

ext_target = dataset.target[:, None]

pd.DataFrame(

      np.concatenate((dataset.data, ext_target,
      dataset.target_names[ext_target]), axis=1),

      columns=dataset.feature_names + ['target label', 'target name'],

)
```

```
features = dataset.data

target = dataset.target


features.shape, target.shape
```

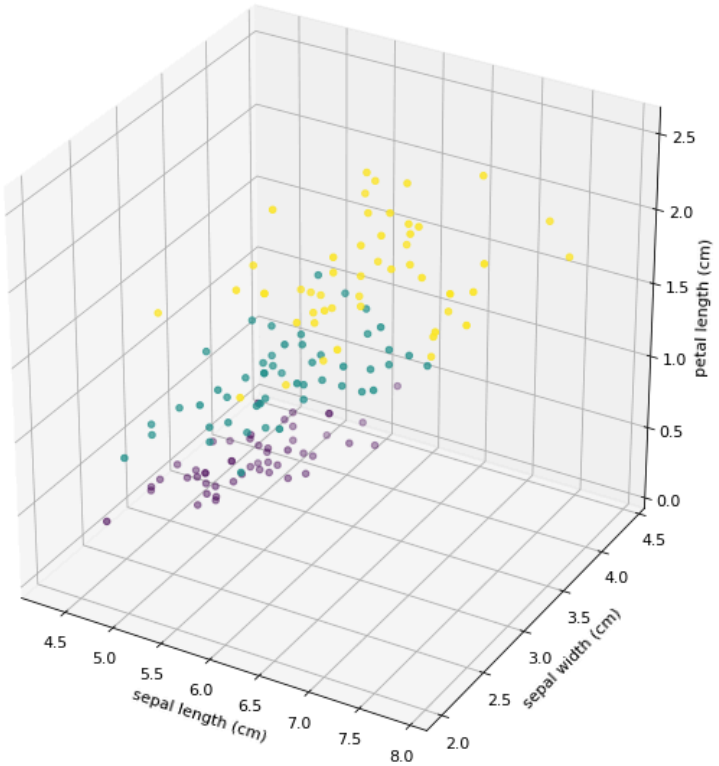
Как правило, размерность матрицы плана или матрицы объект/признак, design matrix — число объектов по вертикали на число признаков по горизонтали, если не проговаривать отдельно что-то другое. Нулевая размерность — это объект.

Нарисуем наши данные:

```
fig = plt.figure(figsize=(8, 8))
```

```
ax = Axes3D(fig)


ax.scatter(features[:, 0], features[:, 1], features[:, 3], c=target,
marker='o')

ax.set_xlabel(dataset.feature_names[0])

ax.set_ylabel(dataset.feature_names[1])

ax.set_zlabel(dataset.feature_names[2])


plt.show()
```

Нарисуем три измерения, четвертый признак отбросим:

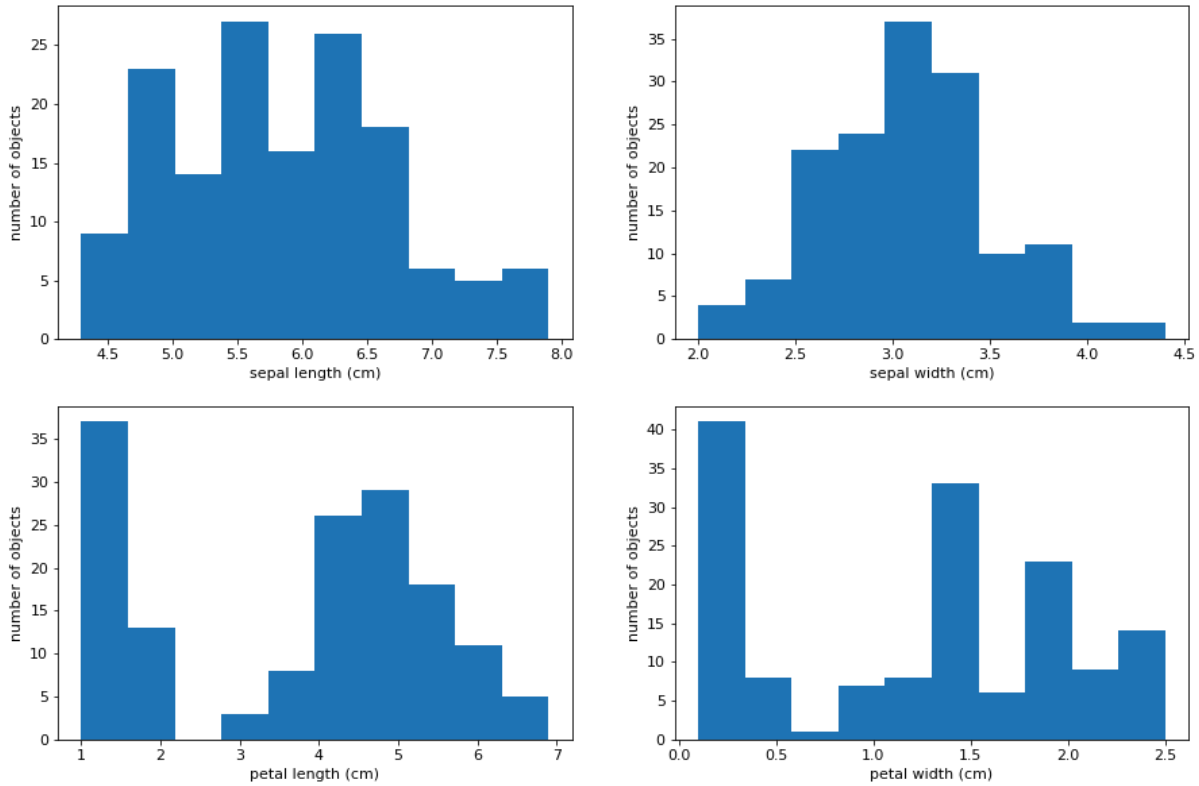


Посмотрим на распределение каждого признака по отдельности:

```
# remember this way to make subplots! It could be useful for you later in
```

```
your work
fig, axes = plt.subplots(2, 2, figsize=(15, 10))
for i, axis in enumerate(axes, flat):
      axis.hist(features[:, i]
      axis.set_xlabel(dataset.feature_names[i])
      axis.set_ylabel('number of objects')
```

По сути, мы строим гистограммы — функции плотности на основании наших данных.



В нашем случае это не совсем плотность, а частота. Мы нормируем на общее количество наблюдений, получаем аппроксимацию плотности в каждой точке.

Унимодальное распределение — распределение с одной модой, одной наивысшей точкой, наивысшим значением плотности, наиболее вероятная точка.

Так можно оценить эмпирическую функцию распределения с помощью гистограммы, ядерной оценки плотности.

Реализация байесовского классификатора

Посмотрим на непрерывные и дискретные случайные величины.

Начнем с чуть более простого решения, чем построение целого классификатора. Напишем класс для нормального распределения или гауссовского распределения:

$$f(x|\mu, \sigma^2) = \frac{1}{\sigma\sqrt{2\pi}} exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right) \quad (7)$$

Это распределение уже реализовано в библиотеках, но написать его полезно.

Так как в аналитическом виде выразить функцию вероятности сложно для гауссовского распределения, мы будем реализовывать PDF (Probability Density Function) — функцию плотности (7):

```
class GaussianDistribution:
      def __init__(self, feature):
            '''
            Args:
                  feature: column of design matrix, represents all
                  available values of features to model.
                  axis=0 stays for samples.
            '''
            self.mean = np.mean(feature, axis=0)
            self.std = np.std(feature, axis=0)
      def logpdf(self, value):
            '''Logarithm of probability density at value'''
            return - 0.5 * np.log(2. * np.pi * self.std**2) - (value -
            self.mean)**2 / (2 * self.std**2) # <YOUR CODE HERE>
      def pdf(self, value):
```

```
        return np.exp(self.logpdf(value)) # <YOUR CODE HERE>
```

Как по выборке оценить свойства распределения?

При увеличении размера выборки среднее по выборке стремится к мат. ожиданию, выборочная дисперсия стремится к дисперсии.

С точки зрения вычислений и устойчивости гораздо привычнее считать логарифм PDF.

Теперь посмотрим, правильно ли у нас все реализовано:

```
import scipy

_test = scipy.stats.norm(loc=features[:, :2].mean(axis=0),
scale=features[:, :2].std(axis=0))

assert np.allclose(

      GaussianDistribution(features[:, :2]).logpdf(features[:5, :2]),

      _test.logpdf(features[:5, :2])

)

print('Seems fine!')
```

Логарифм функции плотности и сама функция плотности не являются вероятностью.

Теперь посмотрим, как построить классификатор.

Мы решаем задачу классификации, то есть хотим оценить, какой меткой класса надо разметить каждый из объектов. Иначе, какая метка класса наиболее вероятна. То есть оценим вероятность определенной метки класса при условии x.

Прологарифмируем формулу Байеса:

$$log\ P(y_i = C_k|x_i) = logP(x_i|y_i = C_k) + logP(y_i = C_k) - logP(x_i) \quad (8)$$

$P(y_i = C_k)$ – вероятность пронаблюдать объект определенного класса. Если есть выборка, можем посчитать частоты всех классов и поделить на общее количество

объектов. Нормализованные частоты и будут нашей вероятностью. В нашем случае классы сбалансированы, их 3, поэтому вероятность всегда ⅓ для всех объектов.

Как оценить $P(x_i|y_i = C_k)$? Матрица ковариации — долго, дорого оценивать. Поэтому в наивном байесовском классификаторе предполагаем, что все признаки независимы.

Для $P(x_i)$ вспомним формулу полной вероятности:

$$P(x_i) = \sum_{k=1}^{K} P(x_i|y_i = C_k)P(y_i = C_k).$$

```
from sklearn.base import BaseEstimator, ClassifierMixin
from scipy.special import logsumexp


class NaiveBayes(BaseEstimator, ClassifierMixin):
      '''
      Please note, using ‘X’ and ‘y’ for design matrix and labels in
      general is not a good choice, better stick to more informative
      naming conventions.
      However, to make the code consistent with sklearn implementation,
      we use ‘X’ and ‘y’ variables here.
      '''
      def fit(self, X, y, sample_weight=None, distributions=None):
      '''
      sample_weight
            The argument is ignored. For compatibility only.
      '''
      self.unique_labels = np.unique(y)
      #If distributions of features are not specified, they a treated
      Gaussian
      if distributions is None:
            distributions = [GaussianDistribution] * X.shape[1]
```

```
        else:
                #Check whether distributions are passed for all features
                assert len(distributions) == X.shape[1]
        # Here we find distribution parameters for every features in every
        class subset
        # so P(x^i|y=C_k) will be estimated only using information from i-th
        feature of C_k class values
        self.conditional_feature_distributions = {} #Label: [distribution
        for feature 1, …]
        for label in self.unique_labels:
                feature_distribution = []
                for column_index in range(X.shape[1]):
                        # column_index feature values for objects from 'label'
                        class
                        feature_column = X[y == label, column_index]
                        fitted_distr =
                        distributions[column_index](feature_column)
                        feature_distribution.append(fitted_distr)
                self.conditional_feature_distributions[label] =
                feater_distribution


        # Prior label distributions (unconditional probability of each
        class)
        self.prior_label_distribution = {
                # <YOUR CODE HERE>
                label: sum((y==label).astype(float)) / len(y)
                for label in self.unique_labels
        }
def predict_log_proba(self, X):
        # Matrix of shape (n_objects : n_classes)
```

```
    class_log_probas = np.zeros((X.shape[0], len(self.unique_labels)).
    dtype=float)


    # Here we compute the class log probabilities for each class
    sequentially b

    for label_idx, label in enumerate(self.unique)labels):

        for idx in range(X.shape[1]):

            # All loglikelihood for every feature w.r.t. fixed label

            class_log_probas[:, label_idx] +=
            self.conditional_feature_distributions[label][idx].logp
            df(X[:, idx] # <YOUR CODE HERE>

        # Add log proba of label prior

        class_log_probas[:, label_idx] +=
        np.log(self.prior_label_distribution[label]) # <YOUR CODE
        HERE>


    for idx in range(X.shape[1]):

    # If you want to get probabilities, you need to subtract the log
    proba for every feature

        class_log_probas -= logsumexp(class_log_probas, axis=1) [:,
        None] # <YOUR CODE HERE>

    return class_log_probas

def predict_proba(self, X):

    return np.exp(self.predict_log_proba(X))


def predict(self, X):

    log_probas = self.predict_log_proba(X)

    # We need to cast labels to their original form (they may start from
    number other than 0)

    return np.array([self.unique_labels[idx] for idx in
    log_probas.argmax(axis=1)])
```

Мы нигде не посчитали знаменатель $P(x_i)$. Чтобы найти наиболее вероятный класс, его не надо считать, знаменатель от у не зависит.

Проверяем, что наш классификатор работает:

```
nb = NaiveBayes()

nb.fit(features, target)

print('log probas:\n{}'.format(nb.predict_log_proba(features[:2])))

print('predicted labels:\n{}'.format(nb.predict(features[:2])))

print('\nIt's alive! More tests coming.')
```

Здесь обучение является именно обучением. Вместо kNN, где мы запоминали всю обучающую выборку, здесь мы вытаскиваем статистики из выборки.

Воспользуемся sklearn, чтобы разбить данные на обучающую и валидационную выборки:

```
from sklearn.model_selection import train_test_split

features_train, features_test, target_train, target_test =
train_test_split(features, target, test_size=0.25)


print(features_train.shape, features_test.shape)
```

Обучаем наш наивный байесовский классификатор:

```
nb = NaiveBayes()

nb.fit(features_train, target_train,
distributions=[GaussianDistribution]*4)

nb_test_log_proba = nb.predict_log_proba(features_test)
```

Получаем точность 95% на обучающей выборке:

```
print('Naive Bayes classifier accuracy on the train set:
{}'.format(nb.score(features_train, target_train)))
```

На валидации 97% — даже выше, чем на обучении. Такое бывает, когда выборки маленькие.

```
print('Naive Bayes classifier accuracy on the train set:
{}'.format(nb.score(features_test, target_test)))
```

Проверим, все ли хорошо у нас работает:

```
from sklearn import naive_bayes

sklearn_nb = naive_bayes.GaussianNB()

sklearn_nb.fit(features_train, target_train)

sklearn_nb_test_log_proba = sklearn_nb.predict_log_proba(features_test)
```

Получаем 95%:

```
print('sklearn implementation accuracy on the train set:
{}'.format(sklearn_nb.score(features_train, target_train)))
```

На валидации то же самое, 97%:

```
print('sklearn implementation accuracy on the test set:
{}'.format(sklearn_nb.score(features_test, target_test)))
```

Мы можем посмотреть, что у нас все логарифмы вероятностей совпадают:

```
assert np.allclose(nb_test_log_proba, sklearn_nb_test_log_proba), 'log
probabilities do not match'

print('Seems alright!')
```

kNN

Сравним теперь с kNN:

```
from sklearn.neighbors import KNeighborsClassifier
```

Возьмем kNN с двумя соседями:

```
knn = KNeighborsClassifier(n_neighbors=2)
```

На тех же самых данных:

```
knn.fit(features_train, target_train)
```

Получаем 98% точность:

```
knn.score(feature_train, target_train)
```

На валидации 92%:

```
knn.score(feature_test, target_test)
```

Получаем, что kNN немного больше склонен к переобучению. Но на таких выборках судить не очень корректно.

5. Эмпирические функции распределения

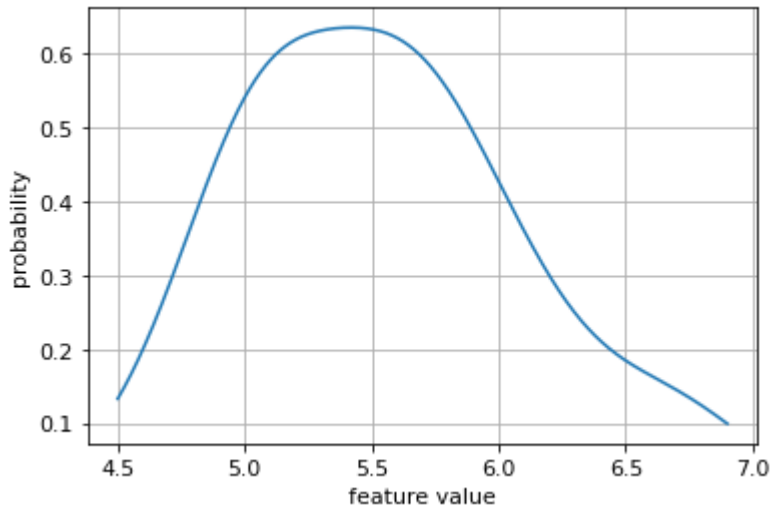
Что делать, если нам неизвестно, какое априорное распределение мы могли бы использовать. Для этого мы можем построить эмпирическую функцию распределения. Например, построить гистограмму или какую-нибудь ядерную оценку плотности:

```
plt.figure(figsize=(15, 15))
plt.violinplot(features, showmedians=True)
```

С ядерной оценкой плотности:

```
from sklearn.neighbors import KernelDensity
```

```
kde = KernelDensity(bandwidth=0.28, kernel='gaussian')
feature_col = features[target==2, 2]
kde.fit(feature_col.reshape((-1, 1)))
linspace = np.linspace(feature_col.min(), feature_col.max(), 1000)
plt.plot(linspace, np.exp(kde.score_samples(linspace.reshape((-1, 1)))))
plt.grid()
plt.xlabel('feature value')
plt.ylabel('probability')
```



Реализуем распределение на основе ядерной оценки плотности:

```
class GaussianKDE:
      def _init_(self, feature):
            self.kde = KernelDensity(bandwidth=1.)
            self.kde.fit(feature.reshape((-1, 1)))

      def logpdf(self, value):
            return self.kde.score_samples(value.reshape((-1, 1)))
```

```
    def pdf(self, value):
        return np.exp(self.log_proba(value))
```

Теперь наивный байесовский классификатор зададим гауссовским распределением:

```
nb_kde = NaiveBayes()
nb_kde.fit(features, target, distributions=[GaussianKDE]*4)
print('log probas:\n{}'.format(nb_kde.predict_log_proba(features[:2])))
print('predicted labels:\n{}'.format(nb_kde.predict(features[:2])))
print('\nIt's alive!')
```

Мы используем гауссовское ядро, чтобы все сгладить. Можно использовать и другие ядра.

На обучении получаем качество 95%:

```
print('KDE Naive Bayes classifier accuracy on the train set:
{}'.format(nb_kde.score(features_train, target_train)))
```

На валидации 86%:

```
print('KDE Naive Bayes classifier accuracy on the test set:
{}'.format(nb_kde.score(features_test, target_test)))
```

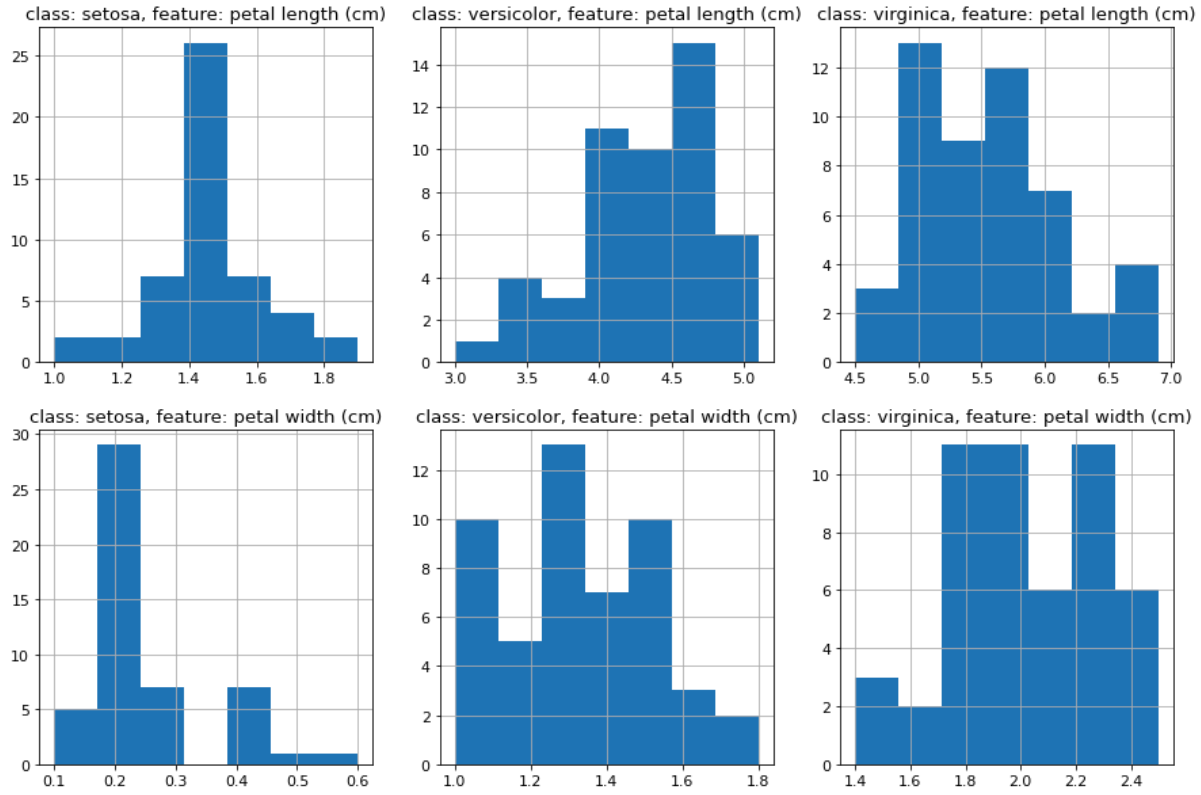
Посмотрим, почему при использовании ядерной оценки мы получили результат хуже, чем при использовании нормального prior.

Некоторые распределения у нас мультимодальные:

```
fig, axes = plt.subplots(2, 3, figsize=(15, 10))
for ax_idx, feature_idx in enumerate([2, 3]):
    for label in range(3):
```

```
            ax = axes[ax_idx, label]
            feature_col = features[target==label, feature_idx]
            ax.hist(feature_col, bins=7)
            ax.grid()
            ax.set_title('class: {}, feature: {}'.format(
                  dataset.target_names[label],
                  dataset.feature_names[feature_idx]
            ))
```

Получаем гистограммы для каждого класса:



С PDF:

```
fig, axes = plt.subplots(2, 3, figsize=(15, 10))
kde = KernelDensity(bandwidth=1., kernel='gaussian')
```

```
for ax_idx, feature_idx in enumerate([2, 3]):
      for label in range(3):
            ax = axes[ax_idx, label]
            feature_col = features[target==label, feature_idx]
            kde.fit(feature_col.reshape((-1, 1)))
            linspace = np.linspace(
                  0.8*feature_col.min(),
                  1.2*feature_col.max(),
                  1000
            )
            ax.plot(linspace,
            np.exp(kde.score_samples(linspace.reshape((-1, 1)))))
            ax.grid()
            ax.set_title('class: {}, feature: {}'.format(
                  dataset.target_names[label],
                  dataset.feature_names[feature_idx]
            ))
```

С гауссовским распределением:

```
fig, axes = plt.subplots(2, 3, figsize=(15, 10))

for ax_idx, feature_idx in enumerate([2, 3]):
      for label in range(3):
            ax = axes[ax_idx, label]
            feature_col = features[target==label, feature_idx]
            gaussian_distr = GaussianDistribution(feature_col)
```

```
        linspace = np.linspace(
              0.8*feature_col.min(),
              1.2*feature_col.max(),
              1000
        )
        ax.plot(linspace,
        np.exp(kde.score_samples(linspace.reshape((-1, 1)))))
        ax.grid()
        ax.set_title('class: {}, feature: {}'.format(
              dataset.target_names[label],
              dataset.feature_names[feature_idx]
        ))
```

У нас гипотеза о нормальном распределении наших данных каждого признака оказалась правильной. Поэтому качество классификации с гауссовским prior выше, чем с ядерной оценкой плотности.

Правильное априорное предположение при решении задачи позволяет не только значительно повысить качество решения, но и понизить сложность и время нахождения решение.

Дополнительное задание: посмотреть задачу с распределением Лапласа.