Машинное обучение. Лабораторная работа 1

Выполнил: *Коростелев Дмитрий Васильевич*

Группа: *М8О-308Б-18*

Настройка среды

In [1]:

```
%pip install numpy
%pip install pandas
%pip install sklearn
```

```
Requirement already satisfied: numpy in d:\programfiles\anaconda\lib\site-
packages (1.16.4)
Note: you may need to restart the kernel to use updated packages.
Requirement already satisfied: pandas in d:\programfiles\anaconda\lib\site
-packages (0.24.2)
Requirement already satisfied: pytz>=2011k in d:\programfiles\anaconda\lib
\site-packages (from pandas) (2019.1)
Requirement already satisfied: numpy>=1.12.0 in d:\programfiles\anaconda\l
ib\site-packages (from pandas) (1.16.4)
Requirement already satisfied: python-dateutil>=2.5.0 in d:\programfiles\a
naconda\lib\site-packages (from pandas) (2.8.0)
Requirement already satisfied: six>=1.5 in d:\programfiles\anaconda\lib\si
te-packages (from python-dateutil>=2.5.0->pandas) (1.12.0)
Note: you may need to restart the kernel to use updated packages.
Requirement already satisfied: sklearn in d:\programfiles\anaconda\lib\sit
e-packages (0.0)
Requirement already satisfied: scikit-learn in d:\programfiles\anaconda\li
b\site-packages (from sklearn) (0.21.2)
Requirement already satisfied: scipy>=0.17.0 in d:\programfiles\anaconda\l
ib\site-packages (from scikit-learn->sklearn) (1.2.1)
Requirement already satisfied: numpy>=1.11.0 in d:\programfiles\anaconda\l
ib\site-packages (from scikit-learn->sklearn) (1.16.4)
Requirement already satisfied: joblib>=0.11 in d:\programfiles\anaconda\li
b\site-packages (from scikit-learn->sklearn) (0.13.2)
Note: you may need to restart the kernel to use updated packages.
```

Задача и датасет

Датасет взят с сайта : http://archive.ics.uci.edu/ml/datasets/Mushroom
(http://archive.ics.uci.edu/ml/datasets/Mushroom)

В базе данных представлена информацию о видах грибов, их свойствах и особенностях. По описанию гриба при помощи искусственного интеллекта будем решать задачу классификации грибов на ядовитые и съедобные

Датасет

In [2]:

```
import pandas as pds

df = pds.read_csv('agaricus-lepiota.data')
```

In [3]:

```
df
```

Out[3]:

	edible	cap-shape	cap-surface	cap-color	bruises	odor	gill-attachment	gill-spacing	gill-size	gill-color	...	sur b
0	p	x	s	n	t	p	f	c	n	k	...	
1	e	x	s	y	t	a	f	c	b	k	...	
2	e	b	s	w	t	l	f	c	b	n	...	
3	p	x	y	w	t	p	f	c	n	n	...	
4	e	x	s	g	f	n	f	w	b	k	...	
5	e	x	y	y	t	a	f	c	b	n	...	
6	e	b	s	w	t	a	f	c	b	g	...	
7	e	b	y	w	t	l	f	c	b	n	...	
8	p	x	y	w	t	p	f	c	n	p	...	
9	e	b	s	y	t	a	f	c	b	g	...	
10	e	x	y	y	t	l	f	c	b	g	...	
11	e	x	y	y	t	a	f	c	b	n	...	
12	e	b	s	y	t	a	f	c	b	w	...	
13	p	x	y	w	t	p	f	c	n	k	...	
14	e	x	f	n	f	n	f	w	b	n	...	
15	e	s	f	g	f	n	f	c	n	k	...	
16	e	f	f	w	f	n	f	w	b	k	...	
17	p	x	s	n	t	p	f	c	n	n	...	
18	p	x	y	w	t	p	f	c	n	n	...	
19	p	x	s	n	t	p	f	c	n	k	...	
20	e	b	s	y	t	a	f	c	b	k	...	
21	p	x	y	n	t	p	f	c	n	n	...	
22	e	b	y	y	t	l	f	c	b	k	...	
23	e	b	y	w	t	a	f	c	b	w	...	
24	e	b	s	w	t	l	f	c	b	g	...	
25	p	f	s	w	t	p	f	c	n	n	...	
26	e	x	y	y	t	a	f	c	b	n	...	
27	e	x	y	w	t	l	f	c	b	w	...	
28	e	f	f	n	f	n	f	c	n	k	...	
29	e	x	s	y	t	a	f	w	n	n	...	
...	...	...	...	...	...	...	...	...	...	...	...	
8094	e	b	s	g	f	n	f	w	b	g	...	
8095	p	x	y	c	f	m	f	c	b	y	...	
8096	e	k	f	w	f	n	f	w	b	w	...	
8097	p	k	y	n	f	s	f	c	n	b	...	

	edible	cap-shape	cap-surface	cap-color	bruises	odor	gill-attachment	gill-spacing	gill-size	gill-color	...	sur b
8098	p	k	s	e	f	y	f	c	n	b	...	
8099	e	k	f	w	f	n	f	w	b	w	...	
8100	e	f	s	n	f	n	a	c	b	o	...	
8101	p	k	s	e	f	s	f	c	n	b	...	
8102	e	x	s	n	f	n	a	c	b	y	...	
8103	e	k	s	n	f	n	a	c	b	y	...	
8104	e	k	s	n	f	n	a	c	b	y	...	
8105	e	k	s	n	f	n	a	c	b	y	...	
8106	e	k	s	n	f	n	a	c	b	o	...	
8107	e	x	s	n	f	n	a	c	b	y	...	
8108	p	k	y	e	f	y	f	c	n	b	...	
8109	e	b	s	w	f	n	f	w	b	w	...	
8110	e	x	s	n	f	n	a	c	b	o	...	
8111	e	k	s	w	f	n	f	w	b	p	...	
8112	e	k	s	n	f	n	a	c	b	o	...	
8113	p	k	y	e	f	y	f	c	n	b	...	
8114	p	f	y	c	f	m	a	c	b	y	...	
8115	e	x	s	n	f	n	a	c	b	y	...	
8116	p	k	y	n	f	s	f	c	n	b	...	
8117	p	k	s	e	f	y	f	c	n	b	...	
8118	p	k	y	n	f	f	f	c	n	b	...	
8119	e	k	s	n	f	n	a	c	b	y	...	
8120	e	x	s	n	f	n	a	c	b	y	...	
8121	e	f	s	n	f	n	a	c	b	n	...	
8122	p	k	y	n	f	y	f	c	n	b	...	
8123	e	x	s	n	f	n	a	c	b	y	...	

8124 rows × 23 columns

Описание атрибутов и возможных значений:

1. edible : p=poisonous, e=edible
2. cap-shape: bell=b,conical=c,convex=x,flat=f, knobbed=k,sunken=s
3. cap-surface: fibrous=f,grooves=g,scaly=y,smooth=s
4. cap-color: brown=n,buff=b,cinnamon=c,gray=g,green=r, pink=p,purple=u,red=e,white=w,yellow=y
5. bruises?: bruises=t,no=f
6. odor: almond=a,anise=l,creosote=c,fishy=y,foul=f, musty=m,none=n,pungent=p,spicy=s
7. gill-attachment: attached=a,descending=d,free=f,notched=n
8. gill-spacing: close=c,crowded=w,distant=d
9. gill-size: broad=b,narrow=n
10. gill-color: black=k,brown=n,buff=b,chocolate=h,gray=g, green=r,orange=o,pink=p,purple=u,red=e, white=w,yellow=y
11. stalk-shape: enlarging=e,tapering=t
12. stalk-root: bulbous=b,club=c,cup=u,equal=e, rhizomorphs=z,rooted=r,missing=?
13. stalk-surface-above-ring: fibrous=f,scaly=y,silky=k,smooth=s
14. stalk-surface-below-ring: fibrous=f,scaly=y,silky=k,smooth=s
15. stalk-color-above-ring: brown=n,buff=b,cinnamon=c,gray=g,orange=o, pink=p,red=e,white=w,yellow=y
16. stalk-color-below-ring: brown=n,buff=b,cinnamon=c,gray=g,orange=o, pink=p,red=e,white=w,yellow=y
17. veil-type: partial=p,universal=u
18. veil-color: brown=n,orange=o,white=w,yellow=y
19. ring-number: none=n,one=o,two=t
20. ring-type: cobwebby=c,evanescent=e,flaring=f,large=l, none=n,pendant=p,sheathing=s,zone=z
21. spore-print-color: black=k,brown=n,buff=b,chocolate=h,green=r, orange=o,purple=u,white=w,yellow=y
22. population: abundant=a,clustered=c,numerous=n, scattered=s,several=v,solitary=y
23. habitat: grasses=g,leaves=l,meadows=m,paths=p, urban=u,waste=w,woods=d

Разделим датасет на обучающую и тестовую выборки. Так как записей в датасете - 8123, разделим датасет в соотношении 3:1

In [4]:

```
separate_index = int(3.0*len(df)/4.0)
print(separate_index)
```

```
6093
```

In [5]:

```
df_learn = df.iloc[0:separate_index]
df_test = df.iloc[separate_index:]
```

Алгоритм KNN

Подготовка датасета

Реализуем алгоритм KNN, сначала сопоставим всем качественным признакам числовые значения и будем работать с числовыми значениями.

In [6]:

```
edible_dict = { 'p' : 0, 'e' : 1 }
cap_shape_dict = { 'b' : 0, 'c' : 1, 'x' : 2, 'f' : 3, 'k' : 4, 's' : 5 }
cap_surface_dict = { 'f' : 0, 'g' : 1, 'y' : 2, 's' : 3 }
cap_color_dict = { 'n' : 0, 'b' : 1, 'c' : 2, 'g' : 3, 'r' : 4, 'p' : 5, 'u' : 6, 'e' :
7, 'w' : 8, 'y' : 9 }
bruises_dict = { 't' : 0, 'f' : 1 }
odor_dict = { 'a' : 0, 'l' : 1, 'c' : 2, 'y' : 3, 'f' : 4, 'm' : 5, 'n' : 6, 'p' : 7,
's' : 8 }
gill_attachment_dict = { 'a' : 0, 'd' : 1, 'f' : 2, 'n' : 3 }
gill_spacing_dict = { 'c' : 0, 'w' : 1, 'd' : 2 }
gill_size_dict = { 'b' : 0, 'n' : 1 }
gill_color_dict = { 'k' : 0, 'n' : 1, 'b' : 2, 'h' : 3, 'g' : 4, 'r' : 5, 'o' : 6, 'p'
: 7, 'u' : 8, 'e' : 9, 'w' : 10, 'y' : 11 }
stalk_shape_dict = { 'e' : 0, 't' : 1 }
stalk_root_dict = { 'b' : 0, 'c' : 1, 'u' : 2, 'e' : 3, 'z' : 4, 'r' : 5, '?' : 6 }
stalk_surface_above_ring_dict = { 'f' : 0, 'y' : 1, 'k' : 2, 's' : 3 }
stalk_surface_below_ring_dict = { 'f' : 0, 'y' : 1, 'k' : 2, 's' : 3 }
stalk_color_above_ring_dict = { 'n' : 0, 'b' : 1, 'c' : 2, 'g' : 3, 'o' : 4, 'p' : 5,
'e' : 6, 'w' : 7, 'y' : 8 }
stalk_color_below_ring_dict = { 'n' : 0, 'b' : 1, 'c' : 2, 'g' : 3, 'o' : 4, 'p' : 5,
'e' : 6, 'w' : 7, 'y' : 8 }
veil_type_dict = { 'p' : 0, 'u' : 1 }
veil_color_dict = { 'n' : 0, 'o' : 1, 'w' : 2, 'y' : 3 }
ring_number_dict = { 'n' : 0, 'o' : 1, 't' : 2 }
ring_type_dict = { 'c' : 0, 'e' : 1, 'f' : 2, 'l' : 3, 'n' : 4, 'p' : 5, 's' : 6, 'z' :
7 }
spore_print_color_dict = { 'k' : 0, 'n' : 1, 'b' : 2, 'h' : 3, 'r' : 4, 'o' : 5, 'u' :
6, 'w' : 7, 'y' : 8 }
population_dict = { 'a' : 0, 'c' : 1, 'n' : 2, 's' : 3, 'v' : 4, 'y' : 5 }
habitat_dict = { 'g' : 0, 'l' : 1, 'm' : 2, 'p' : 3, 'u' : 4, 'w' : 5, 'd' : 6 }

def replace_attribute_on_number(row):
    row['edible'] = edible_dict[row['edible']] # 1
    row['cap-shape'] = cap_shape_dict[row['cap-shape']] # 2
    row['cap-surface'] = cap_surface_dict[row['cap-surface']] # 3
    row['cap-color'] = cap_color_dict[row['cap-color']] # 4
    row['bruises'] = bruises_dict[row['bruises']] # 5
    row['odor'] = odor_dict[row['odor']] # 6
    row['gill-attachment'] = gill_attachment_dict[row['gill-attachment']] # 7
    row['gill-spacing'] = gill_spacing_dict[row['gill-spacing']] # 8
    row['gill-size'] = gill_size_dict[row['gill-size']] # 9
    row['gill-color'] = gill_color_dict[row['gill-color']] # 10
    row['stalk-shape'] = stalk_shape_dict[row['stalk-shape']] # 11
    row['stalk-root'] = stalk_root_dict[row['stalk-root']] # 12
    row['stalk-surface-above-ring'] = stalk_surface_above_ring_dict[row['stalk-surface-
above-ring']] # 13
    row['stalk-surface-below-ring'] = stalk_surface_below_ring_dict[row['stalk-surface-
below-ring']] # 14
    row['stalk-color-above-ring'] = stalk_color_above_ring_dict[row['stalk-color-above-
ring']] # 15
    row['stalk-color-below-ring'] = stalk_color_below_ring_dict[row['stalk-color-below-
ring']] # 16
    row['veil-type'] = veil_type_dict[row['veil-type']] # 17
    row['veil-color'] = veil_color_dict[row['veil-color']] # 18
    row['ring-number'] = ring_number_dict[row['ring-number']] # 19
    row['ring-type'] = ring_type_dict[row['ring-type']] # 20
    row['spore-print-color'] = spore_print_color_dict[row['spore-print-color']] # 21
    row['population'] = population_dict[row['population']] # 22
    row['habitat'] = habitat_dict[row['habitat']] # 23
```

```python
def change_atr_to_numbers_in_df(df):
    df_copy = df.copy()
    for index, row in df_copy.iterrows():
        replace_attribute_on_number(row)
    return df_copy

df_learn_numeric = change_atr_to_numbers_in_df(df_learn)
```

In [7]:

```
df_learn_numeric
```

Out[7]:

	edible	cap-shape	cap-surface	cap-color	bruises	odor	gill-attachment	gill-spacing	gill-size	gill-color	...	sur b
0	0	2	3	0	0	7	2	0	1	0	...	
1	1	2	3	9	0	0	2	0	0	0	...	
2	1	0	3	8	0	1	2	0	0	1	...	
3	0	2	2	8	0	7	2	0	1	1	...	
4	1	2	3	3	1	6	2	1	0	0	...	
5	1	2	2	9	0	0	2	0	0	1	...	
6	1	0	3	8	0	0	2	0	0	4	...	
7	1	0	2	8	0	1	2	0	0	1	...	
8	0	2	2	8	0	7	2	0	1	7	...	
9	1	0	3	9	0	0	2	0	0	4	...	
10	1	2	2	9	0	1	2	0	0	4	...	
11	1	2	2	9	0	0	2	0	0	1	...	
12	1	0	3	9	0	0	2	0	0	10	...	
13	0	2	2	8	0	7	2	0	1	0	...	
14	1	2	0	0	1	6	2	1	0	1	...	
15	1	5	0	3	1	6	2	0	1	0	...	
16	1	3	0	8	1	6	2	1	0	0	...	
17	0	2	3	0	0	7	2	0	1	1	...	
18	0	2	2	8	0	7	2	0	1	1	...	
19	0	2	3	0	0	7	2	0	1	0	...	
20	1	0	3	9	0	0	2	0	0	0	...	
21	0	2	2	0	0	7	2	0	1	1	...	
22	1	0	2	9	0	1	2	0	0	0	...	
23	1	0	2	8	0	0	2	0	0	10	...	
24	1	0	3	8	0	1	2	0	0	4	...	
25	0	3	3	8	0	7	2	0	1	1	...	
26	1	2	2	9	0	0	2	0	0	1	...	
27	1	2	2	8	0	1	2	0	0	10	...	
28	1	3	0	0	1	6	2	0	1	0	...	
29	1	2	3	9	0	0	2	1	1	1	...	
...	...	...	...	...	...	...	...	...	...	...	...	
6063	0	2	2	7	1	4	2	0	1	2	...	
6064	0	3	2	7	1	8	2	0	1	2	...	
6065	0	3	2	0	1	8	2	0	1	2	...	
6066	0	3	3	7	1	4	2	0	1	2	...	

	edible	cap-shape	cap-surface	cap-color	bruises	odor	gill-attachment	gill-spacing	gill-size	gill-color	...	sur b
6067	0	2	3	0	1	8	2	0	1	2	...	
6068	1	2	0	8	1	6	2	1	0	10	...	
6069	0	3	2	0	1	3	2	0	1	2	...	
6070	0	2	3	0	1	4	2	0	1	2	...	
6071	0	2	3	7	1	4	2	0	1	2	...	
6072	0	3	3	0	1	8	2	0	1	2	...	
6073	0	3	3	0	1	8	2	0	1	2	...	
6074	0	2	2	7	1	8	2	0	1	2	...	
6075	0	2	3	7	1	4	2	0	1	2	...	
6076	0	3	3	0	1	3	2	0	1	2	...	
6077	0	4	3	0	1	4	2	0	1	2	...	
6078	0	3	2	0	1	8	2	0	1	2	...	
6079	0	2	3	0	1	8	2	0	1	2	...	
6080	0	3	2	7	1	3	2	0	1	2	...	
6081	0	2	3	7	1	3	2	0	1	2	...	
6082	0	3	3	0	1	8	2	0	1	2	...	
6083	0	3	2	0	1	8	2	0	1	2	...	
6084	0	4	2	0	1	3	2	0	1	2	...	
6085	0	3	2	0	1	8	2	0	1	2	...	
6086	0	3	3	0	1	3	2	0	1	2	...	
6087	0	2	3	7	1	8	2	0	1	2	...	
6088	0	3	3	0	1	3	2	0	1	2	...	
6089	0	3	3	0	1	8	2	0	1	2	...	
6090	0	3	3	7	1	4	2	0	1	2	...	
6091	0	3	2	0	1	3	2	0	1	2	...	
6092	0	3	3	0	1	4	2	0	1	2	...	

6093 rows × 23 columns

Получим ядовитые и съедобные грибы

In [8]:

```
df_learn_poisonous = df_learn_numeric[df_learn_numeric['edible'] == 0]
df_learn_edible =    df_learn_numeric[df_learn_numeric['edible'] == 1]
```

Проверим, что разделение прошло успешно

In [9]:

```
print((len(df_learn_edible) + len(df_learn_poisonous) == len(df_learn_numeric)))
```

```
True
```

Функционал

Датасет подготовлен для обучения модели. Введем несколько дополнительных функций: эвклидово расстояние по которому будем рассчитывать расстояние и функцию взятия среднего арифметического.

In [10]:

```
from math import sqrt
def euclidean(v1, v2):
    return sqrt(sum([(x - y)**2 for x, y in zip(v1, v2)]))

def get_middle_point(df):
    result = []
    df_lists = df.values.tolist()
    for i in range(1, 23):
        sum = 0
        for llist in df_lists:
            sum += llist[i]
        result.append(sum/len(df))
    return result
```

Обучение модели

Найдем две 22-мерные точки соотвествующие вредным и съедобным грибам

In [11]:

```
poisonous_point = []
edible_point = []

def learn_process_knn():
    edible_point = get_middle_point(df_learn_edible)
    poisonous_point = get_middle_point(df_learn_poisonous)
    return edible_point, poisonous_point

edible_point, poisonous_point = learn_process_knn()
print("Центр группы съедобных грибов: {}".format(edible_point))
print()
print("Центр группы ядовитых грибов:  {}".format(poisonous_point))
```

```
Центр группы съедобных грибов: [2.3448275862068964, 1.4439315775183275, 4.
2682595710019005, 0.26147162639152866, 4.805321748574531, 1.99891392886234
05, 0.2478957371707847, 0.0781971219114852, 5.415965245723595, 0.703774097
2033668, 1.4993212055389629, 2.667390714091773, 2.524029323920717, 6.03393
972305186, 5.94270974748846, 0.0, 1.9989139288623405, 1.0524029323920716,
3.8661417322834644, 1.0363833831115938, 3.366549008960087, 3.6453977735541
68]

Центр группы ядовитых грибов:  [2.3991701244813277, 1.6157676348547718, 5.
003734439834025, 0.7410788381742739, 4.34149377593361, 2.0, 0.043153526970
954356, 0.3128630705394191, 4.75850622406639, 0.22987551867219916, 1.06058
09128630705, 2.211203319502075, 2.198755186721992, 4.197510373443984, 4.15
4356846473029, 0.0, 2.0, 1.029875518672199, 3.429875518672199, 3.073029045
6431537, 4.106224066390041, 3.0182572614107883]
```

Обучили модель, теперь проведем тестирование

Тестирование

In [12]:

```
def is_edible_knn(row):
    len_to_poisonous = euclidean(row[1:], poisonous_point)
    len_to_edible = euclidean(row[1:], edible_point)
    return len_to_edible < len_to_poisonous

def test_knn(df):
    df_numeric = change_atr_to_numbers_in_df(df)
    success_count = 0
    for index, row in df_numeric.iterrows():
        real_answer = (row['edible'] == 1)
        algo_answer = is_edible_knn(row)
        if real_answer == algo_answer:
            success_count += 1
    return success_count/len(df)

print("Точность алгоритма: {}%".format(test_knn(df_test)*100))
```

```
Точность алгоритма: 83.89955686853766%
```

Наивный байесовский классификатор

Реализуем простейший байесовский классификатор

Для начала посчитаем сколько в обучающей выборке встречаются ядовитые и съедобвные грибы

In [13]:

```
count_poisonous = len(df_learn_poisonous)
count_edible = len(df_learn_edible)
print("Ядовитых грибов: {}, съедобных: {}".format(count_poisonous,count_edible))
```

```
Ядовитых грибов: 2410, съедобных: 3683
```

Посчитаем вероятности встретить каждый признак в каждом классе

In [14]:

```
def get_dict_probabilities(df):
    res = list()
    df_lists = df.values.tolist()
    for i in range(1,23):
        atr_probabilities = dict()
        for llist in df_lists:
            if llist[i] in atr_probabilities:
                atr_probabilities[llist[i]] += 1
            else :
                atr_probabilities[llist[i]] = 1
        for item in atr_probabilities:
            atr_probabilities[item] /= len(df)
        res.append(atr_probabilities)
    return res

dict_poisonous = get_dict_probabilities(df_learn_poisonous)
dict_edible = get_dict_probabilities(df_learn_edible)
```

Тестирование

In [15]:

```
def is_edible_bayes(row):
    list_row = row.tolist()
    sum_poisonous = 0
    sum_edible = 0
    for i in range(1,23):
        if list_row[i] in dict_poisonous[i-1]:
            sum_poisonous += dict_poisonous[i-1][list_row[i]]
        if list_row[i] in dict_edible[i-1]:
            sum_edible += dict_edible[i-1][list_row[i]]
    return sum_edible > sum_poisonous

def test_bayes(df):
    df_numeric = change_atr_to_numbers_in_df(df)
    success_count = 0
    for index, row in df_numeric.iterrows():
        real_answer = (row['edible'] == 1)
        algo_answer = is_edible_bayes(row)
        if real_answer == algo_answer:
            success_count += 1
    return success_count/len(df)

print("Точность алгоритма: {}%".format(test_bayes(df_test)*100))
```

```
Точность алгоритма: 83.45642540620383%
```

Сравнение с реализацией sklearn

Алгоритм KNN

Подготовим данные для алгоритма

In [16]:

```
df_sklearn_train_data = list()
df_sklearn_train_ans = list()
for index, row in df_learn_poisonous.iterrows():
    df_sklearn_train_data.append(row.values[1:])
    df_sklearn_train_ans.append(row.values[0])
for index, row in df_learn_edible.iterrows():
    df_sklearn_train_data.append(row.values[1:])
    df_sklearn_train_ans.append(row.values[0])

df_sklearn_test_data = list()
df_sklearn_test_ans = list()
for index, row in change_atr_to_numbers_in_df(df_test).iterrows():
    df_sklearn_test_data.append(row.values[1:])
    df_sklearn_test_ans.append(row.values[0])
```

Проводим тестирование

In [17]:

```
from sklearn.neighbors import KNeighborsClassifier

nbrs = KNeighborsClassifier(n_neighbors=10, algorithm='auto').fit(df_sklearn_train_data
, df_sklearn_train_ans)

print("Точность алгоритма : {}%".format(nbrs.score(df_sklearn_test_data, df_sklearn_tes
t_ans)*100))
```

```
Точность алгоритма : 81.33924175283111%
```

Байесовский классификатор ¶

Проводим тестирование

In [18]:

```
from sklearn.naive_bayes import GaussianNB

gnb = GaussianNB().fit(df_sklearn_train_data, df_sklearn_train_ans)

print("Точность алгоритма : {}%".format(gnb.score(df_sklearn_test_data, df_sklearn_test
_ans)*100))
```

```
Точность алгоритма : 96.30723781388478%
```

Выводы

В ходе лабораторной работы были реализованы простейшие классификаторы KNN и байесовкий с приблизительной точностью для выбранного мной датасета в 83% каждый. Сравнил собственную реализацию с реализациями в библиотеке sklearn. Точность KNN собственной реализации оказалась примерно такой же как и в sklearn, алгоритм байесовского классификатора в sklearn имеет удивительно высокую точность в 96%, точность байесовского классификатора сильно выше моей собственной реализации.

В ходе выполнения работы смог убедится на практике в том, что при помощи математики, вычислительной техники и данных можно находить различные корреляции между метриками, которые не поддаются обычному разъяснению для единичных сущностей, иначе говоря, на основе статистики можно с большой долей вероятности предсказать некоторые свойства выбранного объекта.