Классификация грибов

0. Импортируем необходимые модули

In [ ]:
```
# Импортируем модуль для работы с данными
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sb
# Импортируем модули для работы с моделями
import numpy as np
from sklearn import linear_model, metrics, svm
import torch
from torch import nn
```

1. Загрузка датасета

In [ ]:
```
FILE_NAME = "mushrooms.csv"
df = pd.read_csv(FILE_NAME)
```

2. Описание набора данных и решаемой задачи

Набор данных загружен с сайта kaggle по ссылке: https://www.kaggle.com/uciml/mushroom-classification

В этом наборе данных включены описания гипотетических образцов, соответствующих 23 видам жаберных грибов семейства Agaricus и Lepiota, взятых из Полевого справочника Общества Одюбона по североамериканским грибам (1981). Каждый вид определяется как определенно съедобный, определенно ядовитый или съедобный неизвестного вида и не рекомендуется. Последний класс был объединен с ядовитым. Набор формируется из 22 атрибутов и целевой переменной, обозначающей класс съедобности:

1. class - класс (целевая переменная) съедобный=e, ядовитый=p;
2. cap-shape - форма шапочки: b=колокольчик, c=конический, x=выпуклый, f=плоский, k=выпяченный, s=утопленный;
3. cap-surface - поверхность шапочки: f=волокнистая, g=с бороздками, y=чешуйчатая, s=гладкая;
4. cap-color - цвет шапочки: n=коричневый, b=желтовато-коричневый, c=корица, g=серый, r=зелёный, p=розовый, u=фиолетовый, e=красный, w=белый, y=жёлтый;
5. bruises - синяки: t=Истина, f=Ложь;
6. odor - запах: миндальный=a, анис=l, креозот=c, рыбный=y, неприятный=f, затхлый=m, нет=n, острый=p, пряный=s;
7. gill-attachment - жаберное прикрепление: прикреплено=a, нисходящее=d, свободное=f, зубчатое=n;
8. gill-spacing - расстояние между жабрами: близко=c, тесно=w, далеко=d;
9. gill-size - размер жабр: широкий=b, узкий=n;
10. gill-color - цвет жабр: черный=k, коричневый=n, желтовато-коричневый=b, шоколадный=h, серый=g, зеленый=r, оранжевый=o, розовый=p, фиолетовый=u,

красный=e, белый=w, желтый=y;
11. stalk-shape - форма стебля: увеличение=e, сужение=t;
12. stalk-root - стебель-корень: луковичный=b, булавидный=c, чашка=u, равный=e, ризоморфы=z, укорененный=r, отсутствует=?;
13. stalk-surface-above-ring - поверхность стебля над кольцом: волокнистая=f, чешуйчатая=y, шелковистая=k, гладкая=s;
14. stalk-surface-below ring - поверхность стебля под кольцом: волокнистый=f, чешуйчатый=y, шелковистый=k, гладкий=s;
15. stakl-color-above-ring - цвет стебля над кольцом: коричневый=n, желтовато-коричневый=b, коричный=c, серый=g, оранжевый=o, розовый=p, красный=e, белый=w, желтый=y;
16. stakl-color-below-ring - цвет стебля под кольцом: коричневый=n, желтовато-коричневый=b, коричный=c, серый=g, оранжевый=o, розовый=p, красный=e, белый=w, желтый=y;
17. veil-type - тип покрова: частичная=p, полная=u;
18. veil-color - цвет покрова: коричневый=n, оранжевый=o, белый=w, желтый=y;
19. ring-number - количество колец: нет=n, один=o, два=t;
20. ring-type - тип кольца: паутина=c, исчезающий=e, вспыхивающий=f, большой=l, нет=n, подвесной=p, оболочка=s, зональный=z;
21. spore-print-color - цвет спорового отпечатка: черный=k, коричневый=n, желтовато-коричневый=b, шоколадный=h, зеленый=r, оранжевый=o, фиолетовый=u, белый=w, желтый=y;
22. population - популяция: изобилие=a, сгруппированная=c, многочисленная=n, рассеянная=s, несколько=v, одиночная=y;
23. habitat - среда обитания: трава=g, листья=l, луга=m, тропинки=p, город=u,отходы=w, леса=d.

Постановка задачи: Необходимо на основе описания характеристик гриба определить его класс как съедобного или ядовитого. Целевую переменную можно представить бинарными значенями. Таким образом, задача сводится к бинарной классификации. Решение задачи достигается путём решения задачи оптимизации параметров модели, при которых достигается оптимальная точность предсказания модели.

In [ ]:
```
display(df.head(5))      # Выведем данные (первые пять)
display(df.describe())  # Выведем описание данных
display(df.dtypes)     # Выведем типы данных
```

	class	cap-shape	cap-surface	cap-color	bruises	odor	gill-attachment	gill-spacing	gill-size	gill-color	...	stalk-surface-below-ring	stal colo abov rin
0	p	x	s	n	t	p	f	c	n	k	...	s	
1	e	x	s	y	t	a	f	c	b	k	...	s	
2	e	b	s	w	t	l	f	c	b	n	...	s	
3	p	x	y	w	t	p	f	c	n	n	...	s	
4	e	x	s	g	f	n	f	w	b	k	...	s	

5 rows × 23 columns

	class	cap-shape	cap-surface	cap-color	bruises	odor	gill-attachment	gill-spacing	gill-size	gill-color	...	stalk-surface-below-ring
count	8124	8124	8124	8124	8124	8124	8124	8124	8124	8124	...	8124
unique	2	6	4	10	2	9	2	2	2	12	...	4
top	e	x	y	n	f	n	f	c	b	b	...	s
freq	4208	3656	3244	2284	4748	3528	7914	6812	5612	1728	...	4936

4 rows × 23 columns

```
class                     object
cap-shape                 object
cap-surface               object
cap-color                 object
bruises                   object
odor                      object
gill-attachment           object
gill-spacing              object
gill-size                 object
gill-color                object
stalk-shape               object
stalk-root                object
stalk-surface-above-ring  object
stalk-surface-below-ring  object
stalk-color-above-ring    object
stalk-color-below-ring    object
veil-type                 object
veil-color                object
ring-number               object
ring-type                 object
spore-print-color         object
population                object
habitat                   object
dtype: object
```

3. Выделение целевой и факторных переменных

В качестве целевой переменной выступит столбец "class"

В качестве факторных переменных выступят все остальные столбцы

In [ ]:
```
target_var = df.iloc[:,0]       # iloc позволяет обращаться к данным по индексу
factor_vars = df.iloc[:,1:]     # делаем срезы по столбцам

print("Целевая переменная:", target_var.name)
print("Факторные переменные:", list(factor_vars.columns))
```

```
Целевая переменная: class
Факторные переменные: ['cap-shape', 'cap-surface', 'cap-color', 'bruises', 'odor',
'gill-attachment', 'gill-spacing', 'gill-size', 'gill-color', 'stalk-shape', 'stalk-
root', 'stalk-surface-above-ring', 'stalk-surface-below-ring', 'stalk-color-above-ri
ng', 'stalk-color-below-ring', 'veil-type', 'veil-color', 'ring-number', 'ring-typ
e', 'spore-print-color', 'population', 'habitat']
```

4. Преобразование набора данных

Заменим значения в столбце "class": "e" на 1, "p" на 0. В данных огромное количество категориальных переменных. Для работы с ними используем приём One-hot encoding. Выведем получившийся результат

In [ ]:
```
# Целевой переменной зададим бинарные значения
df.loc[df["class"]=="e", "class"] = 1
df.loc[df["class"]=="p", "class"] = 0

# Применяем One-hot encoding
temp_df = df["class"]
df = df.drop(axis=1, columns=["class"])
df = pd.get_dummies(df)
df.insert(loc=0, column="class", value=temp_df.values)

display(df)
```

	class	cap-shape_b	cap-shape_c	cap-shape_f	cap-shape_k	cap-shape_s	cap-shape_x	cap-surface_f	cap-surface_g	cap-surface_s
0	0	0	0	0	0	0	1	0	0	1
1	1	0	0	0	0	0	1	0	0	1
2	1	1	0	0	0	0	0	0	0	1
3	0	0	0	0	0	0	1	0	0	0
4	1	0	0	0	0	0	1	0	0	1
...	...	...	...	...	...	...	...	...	...	...
8119	1	0	0	0	1	0	0	0	0	1
8120	1	0	0	0	0	0	1	0	0	1
8121	1	0	0	1	0	0	0	0	0	1
8122	0	0	0	0	1	0	0	0	0	0
8123	1	0	0	0	0	0	1	0	0	1

8124 rows × 118 columns

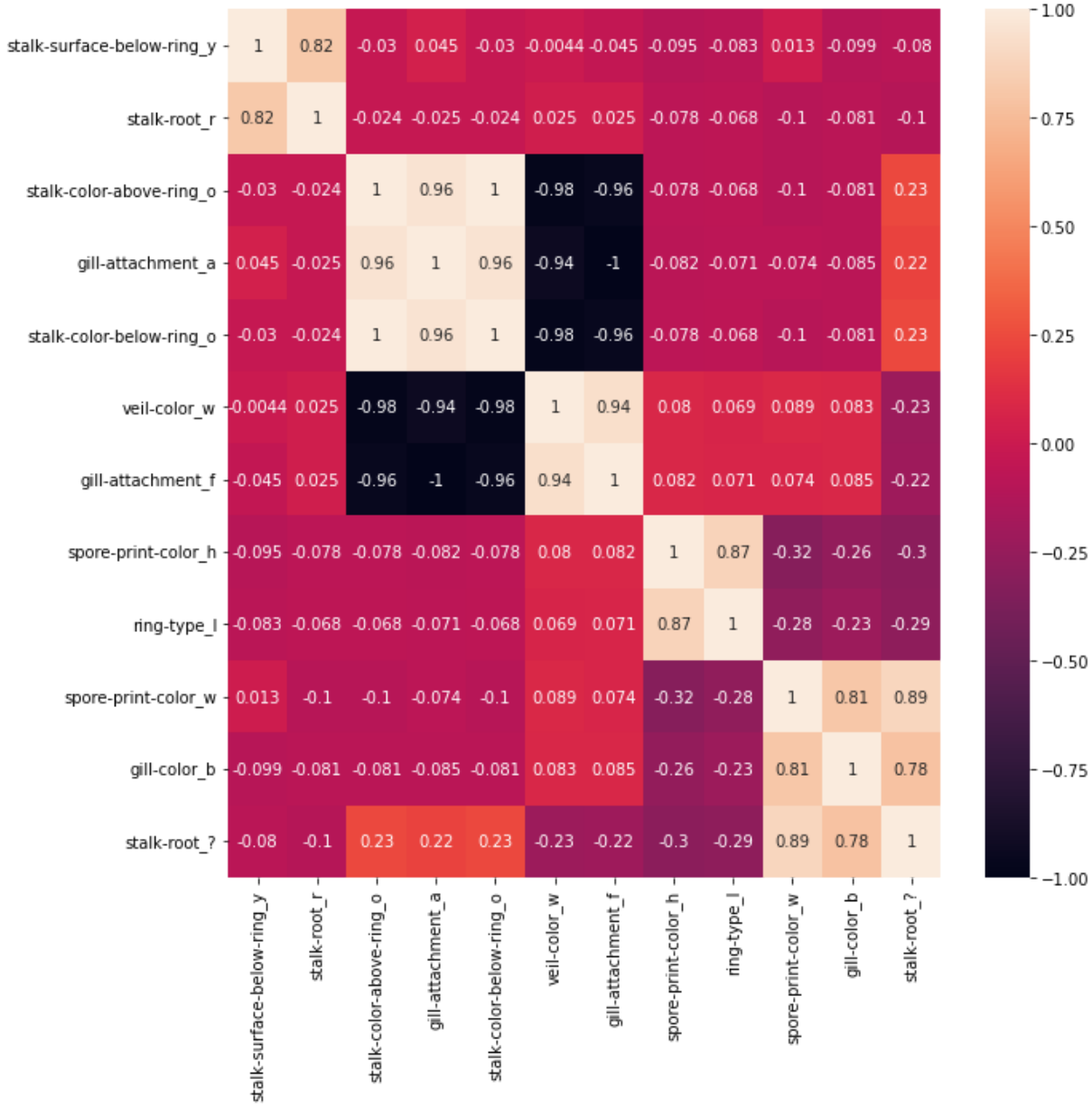
5. Удаление ненужных данных, анализ отсутствующих значений

В данных нет отсутствующих значений, приступим к выявлению ненужных данных. Для этого воспользуемся корреляционным анализом, чтобы выявить коррелирующие переменные и исключить одну из каждой пары. Как видно, корреляций между факторными переменными не наблюдается, следовательно необходимости в уменьшении размерности входных данных не требуется. Также отметим, что исходя из описания во втором пункте в наборе нет аномальных данных: отсутствуют значения вне разумного диапазона, ошибок типов данных.

In [ ]:
```
display(df.isna().sum())        # Проверяем наличие отсутствующих данных
```

```
# Приступим к выявлению корреляций между переменными
# Учтём, что столбец "class" не учитывается в корреляционной матрице
corr_mtx = df.corr().to_numpy()
corr_columns = []
for i in range(corr_mtx.shape[0]):
    for j in range(corr_mtx.shape[1]):
        if corr_mtx[i][j] >= 0.8 and corr_mtx[i][j] < 1 and i > j:
            print(f"Корреляционный коэффициент между столбцами {df.columns[i+1]} и {
            if df.columns[i+1] not in corr_columns: corr_columns.append(df.columns[i
            if df.columns[j+1] not in corr_columns: corr_columns.append(df.columns[j

plt.figure(figsize=(10, 10))
sb.heatmap(df[corr_columns].corr(), annot=True)
plt.show()

# Удалим 7 столбцов
columns_to_drop = ['stalk-surface-below-ring_y', 'stalk-color-above-ring_o',
                   'stalk-color-below-ring_o', 'veil-color_w', 'spore-print-color_h
                   'spore-print-color_w', 'stalk-root_?']
df = df.drop(axis=1, columns=columns_to_drop)
display(df)
```

```
class          0
cap-shape_b    0
cap-shape_c    0
cap-shape_f    0
cap-shape_k    0
              ..
habitat_l      0
habitat_m      0
habitat_p      0
habitat_u      0
habitat_w      0
Length: 118, dtype: int64
Корреляционный коэффициент между столбцами stalk-surface-below-ring_y и stalk-root_r
равен 0.8174442116132162
Корреляционный коэффициент между столбцами stalk-color-above-ring_o и gill-attachmen
t_a равен 0.9550973436312871
Корреляционный коэффициент между столбцами stalk-color-below-ring_o и gill-attachmen
t_a равен 0.9550973436312871
Корреляционный коэффициент между столбцами veil-color_w и gill-attachment_f равен 0.
9352375509363872
Корреляционный коэффициент между столбцами spore-print-color_h и ring-type_l равен
0.8689302627422747
Корреляционный коэффициент между столбцами spore-print-color_w и gill-color_b равен
0.8055732780196003
Корреляционный коэффициент между столбцами spore-print-color_w и stalk-root_? равен
0.8865409947282424
```



	class	cap-shape_b	cap-shape_c	cap-shape_f	cap-shape_k	cap-shape_s	cap-shape_x	cap-surface_f	cap-surface_g	cap-surface_s
0	0	0	0	0	0	0	1	0	0	1
1	1	0	0	0	0	0	1	0	0	1
2	1	1	0	0	0	0	0	0	0	1
3	0	0	0	0	0	0	1	0	0	0
4	1	0	0	0	0	0	1	0	0	1
...	...	...	...	...	...	...	...	...	...	...
8119	1	0	0	0	1	0	0	0	0	1
8120	1	0	0	0	0	0	1	0	0	1
8121	1	0	0	1	0	0	0	0	0	1
8122	0	0	0	0	1	0	0	0	0	0
8123	1	0	0	0	0	0	1	0	0	1

8124 rows × 111 columns

6. Просмотр статистических характеристик выборки

In [ ]:
```
display(df.describe()[:3])  # Выведем описание данных
```

	cap-shape_b	cap-shape_c	cap-shape_f	cap-shape_k	cap-shape_s	cap-shape_x	cap-surface_f
count	8124.000000	8124.000000	8124.000000	8124.000000	8124.000000	8124.000000	8124.000000
mean	0.055638	0.000492	0.387986	0.101920	0.003939	0.450025	0.285574
std	0.229235	0.022185	0.487321	0.302562	0.062641	0.497527	0.451715

3 rows × 110 columns

7. Разбиение выборки на обучающую и тестовую выборки

Разобьём выборку на обучающую и тестовую в пропорциях 75% и 25% соответственно, предварительно перемешав случайным образом данные.

Для этой цели перемешаем выборку с помощью метода sample. frac=1 означает, что мы возвращаем все строки, reset_index отвечает за составление новых индексов для строк.

In [ ]:
```
# Перемешаем датасет
df = df.sample(frac=1).reset_index(drop=True)
train_df = df.iloc[:6094, :].astype("float32")                          # Делаем сре
test_df = df.iloc[6094:, :].reset_index(drop=True).astype("float32")    # Проводим а

# Выведем результат
print("train_df:", train_df.shape)
display(train_df.head(5))
print("test_df:", test_df.shape)
display(test_df.head(5))
```

```
train_df: (6094, 111)
```

	class	cap-shape_b	cap-shape_c	cap-shape_f	cap-shape_k	cap-shape_s	cap-shape_x	cap-surface_f	cap-surface_g	cap-surface_s	...
0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	...
1	1.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0	...
2	0.0	0.0	0.0	0.0	0.0	0.0	1.0	1.0	0.0	0.0	...
3	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	...
4	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	...

5 rows × 111 columns

```
test_df: (2030, 111)
```

	class	cap-shape_b	cap-shape_c	cap-shape_f	cap-shape_k	cap-shape_s	cap-shape_x	cap-surface_f	cap-surface_g	cap-surface_s	...

	class	cap-shape_b	cap-shape_c	cap-shape_f	cap-shape_k	cap-shape_s	cap-shape_x	cap-surface_f	cap-surface_g	cap-surface_s	...
0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	1.0	...
1	0.0	0.0	0.0	0.0	0.0	0.0	1.0	1.0	0.0	0.0	...
2	1.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	1.0	...
3	1.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0	...
4	1.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0	...

5 rows × 111 columns

8. Обучение моделей.

Обучим несколько видов моделей обучения с учителем. Полученные выборки предполагают решение задачи бинарной классификации.

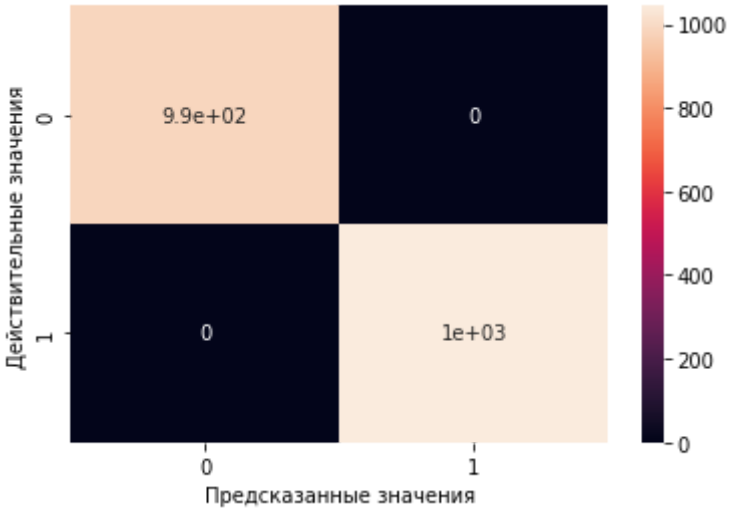
8.1 Обучение первой модели (множественная линейная регрессия)

In [ ]:

```
from warnings import filterwarnings
filterwarnings("ignore")

# Инициализируем модель
reg = linear_model.LinearRegression()

# Обучение модели
X_train = train_df.iloc[:,1:]
Y_train = train_df.iloc[:,0]
reg.fit(X_train, Y_train)

# Оценим модель на тестовой выборке
X_test = test_df.iloc[:,1:]
Y_test = test_df.iloc[:,0]
pred = reg.predict(X_test)
for i in range(len(pred)):
    pred[i] = 1 if pred[i] >= 0.5 else 0
print("Метрика достоверности предсказания:", metrics.accuracy_score(Y_test, pred))
print("Метрика точности:", metrics.precision_score(Y_test, pred))
print("Метрика полноты:", metrics.recall_score(Y_test, pred))
print("Матрица классификации:\n", metrics.confusion_matrix(Y_test, pred))

# Графическая форма
%matplotlib inline
class_names = [0, 1]
fig, ax = plt.subplots()
ticks = np.arange(len(class_names))
plt.xticks(ticks, class_names)
plt.yticks(ticks, class_names)
sb.heatmap(pd.DataFrame(
    metrics.confusion_matrix(Y_test, pred)),
    annot=True)
plt.ylabel('Действительные значения')
plt.xlabel('Предсказанные значения')

# Пример расчёта значения целевой переменной по входным значениям
reg_result = reg.predict([X_test.iloc[1,:]])
result = 1 if reg_result >= 0.5 else 0
```

```
print("------------------------------")
print("Пример расчёта значения целевой переменной по входным значениям")
print("Значение рассчитанной целевой переменной:", reg_result)
print("Интерпретация значения переменной:", result)
print("Истинное значение переменной:", Y_train[1])
```

```
Метрика достоверности предсказания: 1.0
Метрика точности: 1.0
Метрика полноты: 1.0
Матрица классификации:
 [[ 986    0]
 [   0 1044]]
------------------------------
Пример расчёта значения целевой переменной по входным значениям
Значение рассчитанной целевой переменной: [-1.1920929e-07]
Интерпретация значения переменной: 0
Истинное значение переменной: 1.0
```



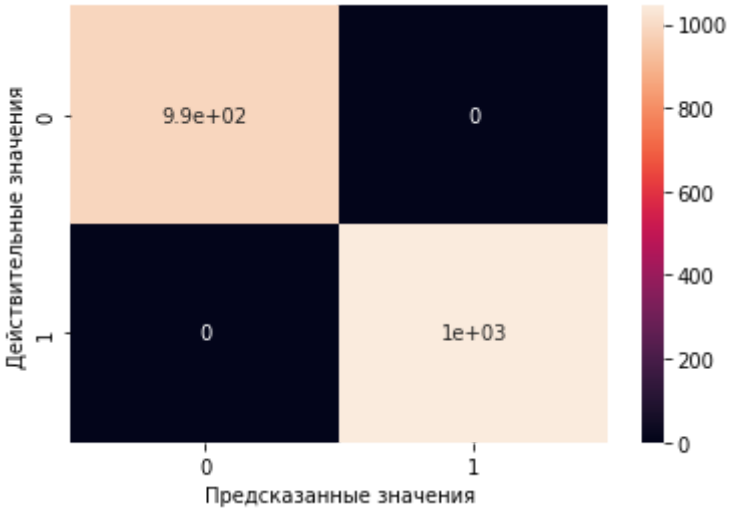
8.2 Обучение второй модели (логистическая регрессия)

In [ ]:

```
# Инициализируем модель
log_reg = linear_model.LogisticRegression()

# Обучение модели
X_train = train_df.iloc[:,1:]
Y_train = train_df.iloc[:,0]
log_reg.fit(X_train, Y_train)

# Оценим модель
X_test = test_df.iloc[:,1:]
Y_test = test_df.iloc[:,0]
print("Метрика достоверности предсказания:", metrics.accuracy_score(Y_test, log_reg.
print("Метрика точности:", metrics.precision_score(Y_test, log_reg.predict(X_test)))
print("Метрика полноты:", metrics.recall_score(Y_test, log_reg.predict(X_test)))
print("Матрица классификации:\n", metrics.confusion_matrix(Y_test, log_reg.predict(X

# Графическая форма
%matplotlib inline
class_names = [0, 1]
fig, ax = plt.subplots()
ticks = np.arange(len(class_names))
plt.xticks(ticks, class_names)
plt.yticks(ticks, class_names)
sb.heatmap(pd.DataFrame(
    metrics.confusion_matrix(Y_test, log_reg.predict(X_test))),
    annot=True)
plt.ylabel('Действительные значения')
```

```
plt.xlabel('Предсказанные значения')

# Пример расчёта значения целевой переменной по входным значениям
log_reg_result = log_reg.predict([X_test.iloc[1,:]])
log_result = 1 if log_reg_result >= 0.5 else 0
print("------------------------------")
print("Пример расчёта значения целевой переменной по входным значениям")
print("Значение рассчитанной целевой переменной:", log_reg_result)
print("Интерпретация значения переменной:", log_result)
print("Истинное значение переменной:", Y_train[1])
```

```
Метрика достоверности предсказания: 1.0
Метрика точности: 1.0
Метрика полноты: 1.0
Матрица классификации:
 [[ 986    0]
 [   0 1044]]
------------------------------
Пример расчёта значения целевой переменной по входным значениям
Значение рассчитанной целевой переменной: [0.]
Интерпретация значения переменной: 0
Истинное значение переменной: 1.0
```



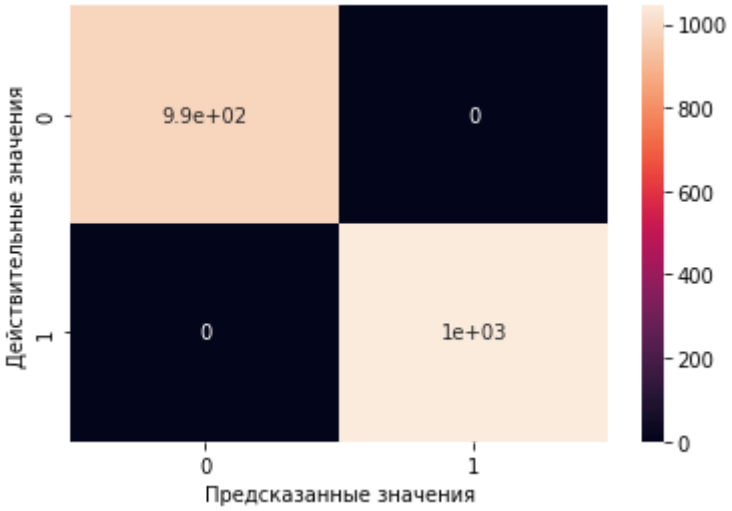
8.3 Обучение третьей модели (Метод опорных векторов)

In [ ]:

```
# Инициализируем модель
svm_model = svm.SVC()

# Обучение модели
X_train = train_df.iloc[:,1:]
Y_train = train_df.iloc[:,0]
svm_model.fit(X_train, Y_train)

# Оценим модель
X_test = test_df.iloc[:,1:]
Y_test = test_df.iloc[:,0]
print("Метрика достоверности предсказания:", metrics.accuracy_score(Y_test, svm_mode
print("Метрика точности:", metrics.precision_score(Y_test, svm_model.predict(X_test)
print("Метрика полноты:", metrics.recall_score(Y_test, svm_model.predict(X_test)))
print("Матрица классификации:\n", metrics.confusion_matrix(Y_test, svm_model.predict

# Графическая форма
%matplotlib inline
class_names = [0, 1]
fig, ax = plt.subplots()
ticks = np.arange(len(class_names))
plt.xticks(ticks, class_names)
```

```
plt.yticks(ticks, class_names)
sb.heatmap(pd.DataFrame(
    metrics.confusion_matrix(Y_test, svm_model.predict(X_test))),
    annot=True)
plt.ylabel('Действительные значения')
plt.xlabel('Предсказанные значения')

# Пример расчёта значения целевой переменной по входным значениям
svm_model_result = svm_model.predict([X_test.iloc[1,:]])
result = 1 if svm_model_result >= 0.5 else 0
print("------------------------------")
print("Пример расчёта значения целевой переменной по входным значениям")
print("Значение рассчитанной целевой переменной:", svm_model_result)
print("Интерпретация значения переменной:", result)
print("Истинное значение переменной:", Y_train[1])
```

```
Метрика достоверности предсказания: 1.0
Метрика точности: 1.0
Метрика полноты: 1.0
Матрица классификации:
 [[ 986    0]
 [   0 1044]]
------------------------------
Пример расчёта значения целевой переменной по входным значениям
Значение рассчитанной целевой переменной: [0.]
Интерпретация значения переменной: 0
Истинное значение переменной: 1.0
```



8.4 Обучение четвёртой модели (Нейронная сеть)

In [ ]:

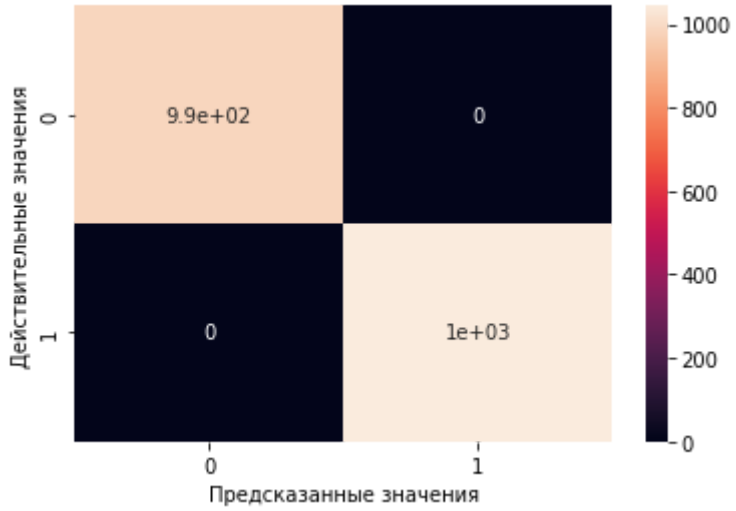
```
# Определяем устройство для вычислений
device = "cuda" if torch.cuda.is_available() else "cpu"
print(f"Используется устройство: {device}")

# Создаём класс нейронной сети
class NeuralNetwork(nn.Module):
    def __init__(self):
        super(NeuralNetwork, self).__init__()
        self.flatten = nn.Flatten()
        self.linear_relu_stack = nn.Sequential(
            nn.Linear(110, 60),
            nn.ReLU(),
            nn.Dropout(p=0.5),
            nn.Linear(60, 1),
            nn.Sigmoid()
        )
```

```
    def forward(self, x):
        x = self.flatten(x)
        logits = self.linear_relu_stack(x)
        return logits.squeeze()

model = NeuralNetwork().to(device)
#model = model.float()

# Создаём лосс-функцию и оптимизатор
loss_fn = nn.BCEWithLogitsLoss()    # Бинарная кросс-энтропия
optimizer = torch.optim.Adam(model.parameters())    # Метод оптимизации Adam

# Разделим выборку на пакеты
train_size = len(train_df)
BATCH_SIZE = 32
BATCH_NUMBER = 0
RESIDUAL_NUMBER = 0
X_batches = []
Y_batches = []
X_residuals = []
Y_residuals = []
for i, batch_df in train_df.groupby(np.arange(train_size) // BATCH_SIZE):
    if batch_df.iloc[:,1:].shape[0] == BATCH_SIZE:
        X_batches.append(batch_df.iloc[:,1:].values)
        Y_batches.append(batch_df.iloc[:,0].values)
        BATCH_NUMBER += 1
    elif batch_df.iloc[:,1:].shape[0] < BATCH_SIZE:
        X_residuals.append(batch_df.iloc[:,1:].values)
        Y_residuals.append(batch_df.iloc[:,0].values)
        RESIDUAL_NUMBER += 1
X_batches = torch.tensor(X_batches).float()
Y_batches = torch.tensor(Y_batches).float()
X_residuals = torch.tensor(X_residuals).float()
Y_residuals = torch.tensor(Y_residuals).float()

test_size = len(test_df)
X_test = torch.tensor(test_df.iloc[:,1:].values).float()
Y_test = torch.tensor(test_df.iloc[:,0].values).float()

# Обучение модели
def train(X_batches, Y_batches, BATCH_NUMBER, X_residuals, Y_residuals, RESIDUAL_NUM
    model.train()
    current = 0
    for i in range(BATCH_NUMBER):
        X, y = X_batches[i].to(device), Y_batches[i].to(device)

        # Рассчитываем ошибку прогноза
        pred = model(X)
        loss = loss_fn(pred, y)

        #Обратное распространение ошибки
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        if i % 30 == 0:
            loss, current = loss.item(), i * BATCH_SIZE
            print(f"Значение loss: {loss}; Прогресс: {current}/{train_size}")

    for i in range(RESIDUAL_NUMBER):
        X, y = X_residuals[i].to(device), Y_residuals[i].to(device)

        # Рассчитываем ошибку прогноза
        pred = model(X)
```

```
        loss = loss_fn(pred, y)

        #Обратное распространение ошибки
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        if i % 30 == 0:
            loss, current = loss.item(), current + i * BATCH_SIZE
            print(f"Значение loss: {loss}; Прогресс: {current}/{train_size}")

def test(X_test, Y_test):
    model.eval()
    with torch.no_grad():
        test_loss, correct = 0, 0
        X, y = X_test.to(device), Y_test.to(device)

        # Рассчитываем ошибку прогноза
        pred = model(X)
        test_loss = loss_fn(pred, y).item()
        for i in range (len(pred)):
            correct += 1 if (pred[i] >= 0.5 and y[i] == 1) or (pred[i] < 0.5 and y[i
        correct /= test_size
        print(f"Точность: {100*correct}%; Среднее значение loss-функции: {test_loss}

EPOCHS = 5
for ep in range(EPOCHS):
    print(f"Эпоха: {ep+1}\n-------------------------------")
    train(X_batches, Y_batches, BATCH_NUMBER, X_residuals, Y_residuals, RESIDUAL_NUM
    test(X_test, Y_test)
print("Обучение завершено!\n-------------------------------")

# Оценим модель
pred = model(X_test.to(device)).to("cpu").detach().numpy()
for i in range(len(pred)):
    pred[i] = 1 if pred[i] >= 0.5 else 0
print("Метрика достоверности предсказания:", metrics.accuracy_score(Y_test, pred))
print("Метрика точности:", metrics.precision_score(Y_test, pred))
print("Метрика полноты:", metrics.recall_score(Y_test, pred))
print("Матрица классификации:\n", metrics.confusion_matrix(Y_test, pred))

# Графическая форма
%matplotlib inline
class_names = [0, 1]
fig, ax = plt.subplots()
ticks = np.arange(len(class_names))
plt.xticks(ticks, class_names)
plt.yticks(ticks, class_names)
sb.heatmap(pd.DataFrame(
    metrics.confusion_matrix(Y_test, pred)),
    annot=True)
plt.ylabel('Действительные значения')
plt.xlabel('Предсказанные значения');
```

```
Используется устройство: cuda
Эпоха: 1
-------------------------------
Значение loss: 0.7524926662445068; Прогресс: 0/6094
Значение loss: 0.6043857336044312; Прогресс: 960/6094
Значение loss: 0.5784928202629089; Прогресс: 1920/6094
Значение loss: 0.5509383678436279; Прогресс: 2880/6094
Значение loss: 0.5794936418533325; Прогресс: 3840/6094
Значение loss: 0.48723217844963074; Прогресс: 4800/6094
Значение loss: 0.43144717812538147; Прогресс: 5760/6094
```

```
Значение loss: 0.5889532566070557; Прогресс: 5760/6094
Точность: 97.68472906403942%; Среднее значение loss-функции: 0.5171509385108948

Эпоха: 2
------------------------------
Значение loss: 0.5527323484420776; Прогресс: 0/6094
Значение loss: 0.4645736515522003; Прогресс: 960/6094
Значение loss: 0.4897664487361908; Прогресс: 1920/6094
Значение loss: 0.513242244720459; Прогресс: 2880/6094
Значение loss: 0.5351964235305786; Прогресс: 3840/6094
Значение loss: 0.47120970487594604; Прогресс: 4800/6094
Значение loss: 0.4218035340309143; Прогресс: 5760/6094
Значение loss: 0.5856904983520508; Прогресс: 5760/6094
Точность: 99.55665024630542%; Среднее значение loss-функции: 0.504031777381897

Эпоха: 3
------------------------------
Значение loss: 0.5335543155670166; Прогресс: 0/6094
Значение loss: 0.45052677392959595; Прогресс: 960/6094
Значение loss: 0.4809349775314331; Прогресс: 1920/6094
Значение loss: 0.5057548880577087; Прогресс: 2880/6094
Значение loss: 0.530800998210907; Прогресс: 3840/6094
Значение loss: 0.4687004089355469; Прогресс: 4800/6094
Значение loss: 0.4211575984954834; Прогресс: 5760/6094
Значение loss: 0.5847501754760742; Прогресс: 5760/6094
Точность: 99.90147783251231%; Среднее значение loss-функции: 0.5004458427429199

Эпоха: 4
------------------------------
Значение loss: 0.5318574905395508; Прогресс: 0/6094
Значение loss: 0.44854968786239624; Прогресс: 960/6094
Значение loss: 0.48032522201538086; Прогресс: 1920/6094
Значение loss: 0.5046207308769226; Прогресс: 2880/6094
Значение loss: 0.5288896560668945; Прогресс: 3840/6094
Значение loss: 0.4683581590652466; Прогресс: 4800/6094
Значение loss: 0.42088115215301514; Прогресс: 5760/6094
Значение loss: 0.584638774394989; Прогресс: 5760/6094
Точность: 100.0%; Среднее значение loss-функции: 0.49919894337654114

Эпоха: 5
------------------------------
Значение loss: 0.5291143655776978; Прогресс: 0/6094
Значение loss: 0.4465307891368866; Прогресс: 960/6094
Значение loss: 0.48027896881103516; Прогресс: 1920/6094
Значение loss: 0.5044705867767334; Прогресс: 2880/6094
Значение loss: 0.5282405614852905; Прогресс: 3840/6094
Значение loss: 0.46805453300476074; Прогресс: 4800/6094
Значение loss: 0.42157500982284546; Прогресс: 5760/6094
Значение loss: 0.5846499800682068; Прогресс: 5760/6094
Точность: 100.0%; Среднее значение loss-функции: 0.4986216425895691

Обучение завершено!
------------------------------
Метрика достоверности предсказания: 1.0
Метрика точности: 1.0
Метрика полноты: 1.0
Матрица классификации:
 [[ 986    0]
 [   0 1044]]
```



Заключение

Мы обучили ряд моделей решать задачу бинарной классификации на наборе данных о съедобности грибов по ряду категориальных признаков. Для обучения необходимо было провести предобработку данных, которая заключалась в перевод категориальных переменных в подходящую для машинного обучения форму. Была создана обучающая выборка, котора затем разбивалась на пакеты. Также была создана тестовая выборка, на которой происходила оценка качества работы моделей.

Обучено 4 модели, среди которых множественная линейная регрессия, логистическая регрессия, метод опорных векторов и полносвязная нейронная сеть на тестовой выборке дали 100% точности.