Обработка больших данных и распределенные вычисления
Лабораторная работа 4. Применение методов и готовых алгоритмов многомерного анализа данных для решения предсказательных задач

Цель: изучить способы применения методов многомерного анализа данных из заданного набора, а также известных алгоритмов на их основе, для решения предсказательных задач, в том числе за счет построения и использования моделей.

Задачи:
1. Сформулировать задачу для анализа данных из данного набора и решить ее методом многомерной линейной регрессии.
2. Сформулировать задачу для анализа данных из данного набора и решить ее методом выявления аномалий.
3. Сформулировать задачу для анализа данных из данного набора и решить ее методом иерархической кластеризации.
4. Сформулировать задачу для анализа данных из данного набора и решить ее с помощью наивного байесовского классификатора.

Список рекомендуемых источников:

1. Нисчал Н. Python — это просто. Пошаговое руководство по программированию и анализу данных: Пер. с англ. — СПб.: БХВ-Петербург, 2022
2. Погружение в аналитику данных. Маунт Джордж: Пер. с англ. — СПб.: БХВ-Петербург, 2023
3. Пример решения задачи множественной регрессии с помощью Python (URL: https://habr.com/ru/articles/206306/)
4. Агломеративная кластеризация и дендрограмма в Python (URL: https://teletype.in/@dt_analytic/LtpSsL__xO2)
5. Истина где-то рядом — ищем аномалии с Python. Часть 1: теория (URL: https://www.reg.ru/blog/ishchem-anomalii-s-python-chast-1/)
6. Наивный байесовский классификатор. Основная идея, модификации и реализация с нуля на Python (URL: https://habr.com/ru/articles/802435/)

Методические указания:

Описание набора данных и постановка задачи

Дан набор данных, котором описаны следующие атрибуты помещения:

Поле	Описание	Тип
X1	Относительная компактность	FLOAT
X2	Площадь	FLOAT
X3	Площадь стены	FLOAT
X4	Площадь потолка	FLOAT
X5	Общая высота	FLOAT
X6	Ориентация	INT
X7	Площадь остекления	FLOAT
X8	Распределенная площадь остекления	INT
y1	Нагрузка при обогреве	FLOAT
y2	Нагрузка при охлаждении	FLOAT

В нем $X1 \ldots X8$ — характеристики помещения, на основании которых будет проводиться анализ, а $y1, y2$ — значения нагрузки, которые надо спрогнозировать.

Предварительный анализ данных

Для начала загрузим наши данные и посмотрим на них:

```
from pandas import read_csv, DataFrame
from sklearn.neighbors import KNeighborsRegressor
from sklearn.linear_model import LinearRegression, LogisticRegression
from sklearn.svm import SVR
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import r2_score
from sklearn.cross_validation import train_test_split

dataset = read_csv('EnergyEfficiency/ENB2012_data.csv',';')
dataset.head()
```

	X1	X2	X3	X4	X5	X6	X7	X8	Y1	Y2
0	0.98	514.5	294.0	110.25	7	2	0	0	15.55	21.33
1	0.98	514.5	294.0	110.25	7	3	0	0	15.55	21.33
2	0.98	514.5	294.0	110.25	7	4	0	0	15.55	21.33
3	0.98	514.5	294.0	110.25	7	5	0	0	15.55	21.33
4	0.90	563.5	318.5	122.50	7	2	0	0	20.84	28.28

Теперь давайте посмотрим не связаны ли между собой какие-либо атрибуты. Сделать это можно, рассчитав коэффициенты корреляции для всех столбцов.

```
dataset.corr()
```

	X1	X2	X3	X4	X5	X6	X7	X8	Y1	Y2
X1	1.000000e+00	-9.919015e-01	-2.037817e-01	-8.688234e-01	8.277473e-01	0.000000	1.283986e-17	1.764620e-17	0.622272	0.634339
X2	-9.919015e-01	1.000000e+00	1.955016e-01	8.807195e-01	-8.581477e-01	0.000000	1.318356e-16	-3.558613e-16	-0.658120	-0.672999
X3	-2.037817e-01	1.955016e-01	1.000000e+00	-2.923165e-01	2.809757e-01	0.000000	-7.969726e-19	0.000000e+00	0.455671	0.427117
X4	-8.688234e-01	8.807195e-01	-2.923165e-01	1.000000e+00	-9.725122e-01	0.000000	-1.381805e-16	-1.079129e-16	-0.861828	-0.862547
X5	8.277473e-01	-8.581477e-01	2.809757e-01	-9.725122e-01	1.000000e+00	0.000000	1.861418e-18	0.000000e+00	0.889431	0.895785
X6	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	1.000000	0.000000e+00	0.000000e+00	-0.002587	0.014290
X7	1.283986e-17	1.318356e-16	-7.969726e-19	-1.381805e-16	1.861418e-18	0.000000	1.000000e+00	2.129642e-01	0.269841	0.207505
X8	1.764620e-17	-3.558613e-16	0.000000e+00	-1.079129e-16	0.000000e+00	0.000000	2.129642e-01	1.000000e+00	0.087368	0.050525
Y1	6.222722e-01	-6.581202e-01	4.556712e-01	-8.618283e-01	8.894307e-01	-0.002587	2.698410e-01	8.736759e-02	1.000000	0.975862

Y2	6.343391e-01	-6.729989e-01	4.271170e-01	-8.625466e-01	8.957852e-01	0.014290	2.075050e-01	5.052512e-02	0.975862	1.000000

Как можно заметить из нашей матрицы, коррелируют между собой следующие столбы (Значение коэффициента корреляции больше 95%):

- y1 --> y2
- x1 --> x2
- x4 --> x5

Теперь давайте выберем, какие столбцы их наших пар мы можем убрать из нашей выборки. Для этого, в каждой паре, выберем столбцы, которые в большей степени оказывают влияние на прогнозные значения *Y1* и *Y2* и оставим их, а остальные удалим.

Как можно заметить и матрицы с коэффициентами корреляции на ***y1,y2*** больше значения оказывают ***X2*** и ***X5***, нежели X1 и X4, таким образом мы можем последние столбцы мы можем удалить.

```
dataset = dataset.drop(['X1','X4'], axis=1)
dataset.head()
```

Помимо этого, можно заметить, что поля ***Y1*** и ***Y2*** очень тесно коррелируют между собой. Но, т. к. нам надо спрогнозировать оба значения мы их оставляем «как есть».

Выбор модели

Отделим от нашей выборки прогнозные значения:

```
trg = dataset[['Y1','Y2']]
trn = dataset.drop(['Y1','Y2'], axis=1)
```

Давайте поместим все наши модели в один список для удобства дальнейшего анализа:

```
        models = [LinearRegression(), # метод наименьших квадратов
                  RandomForestRegressor(n_estimators=100, max_features ='sqrt'),
# случайный лес
                  KNeighborsRegressor(n_neighbors=6), # метод ближайших соседей
```

```
                    SVR(kernel='linear'), # метод опорных векторов с линейным ядром
                    LogisticRegression() # логистическая регрессия
                    ]
```

Итак модели готовы, теперь мы разобьем наши исходные данные на 2 подвыборки: *тестовую* и *обучающую*. Кто читал мои предыдущие статьи знает, что сделать это можно с помощью функции train_test_split() из пакета scikit-learn:

```
Xtrn, Xtest, Ytrn, Ytest = train_test_split(trn, trg, test_size=0.4)
```

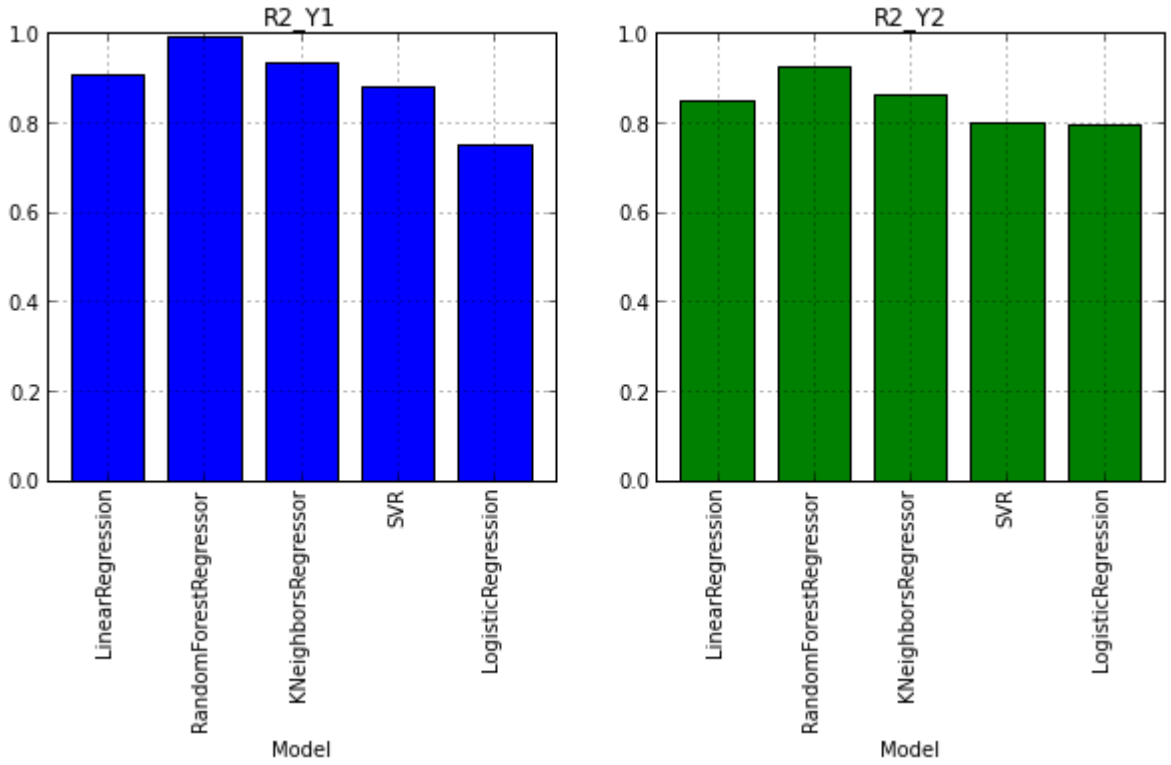
Теперь, т. к. нам надо спрогнозировать 2 параметра $y1,y2$, надо построить регрессию для каждого из них. Кроме этого, для дальнейшего анализа, можно записать полученные результаты во временный *DataFrame*. Сделать это можно так:

```
        #создаем временные структуры
        TestModels = DataFrame()
        tmp = {}
        #для каждой модели из списка
        for model in models:
            #получаем имя модели
            m = str(model)
            tmp['Model'] = m[:m.index('(')]
            #для каждого столбцам результирующего набора
            for i in  xrange(Ytrn.shape[1]):
                #обучаем модель
                model.fit(Xtrn, Ytrn[:,i])
                #вычисляем коэффициент детерминации
                tmp['R2_Y%s'%str(i+1)] = r2_score(Ytest[:,0],
model.predict(Xtest))
            #записываем данные и итоговый DataFrame
            TestModels = TestModels.append([tmp])
        #делаем индекс по названию модели
        TestModels.set_index('Model', inplace=True)
```

Как можно заметить из кода выше, для расчета коэффициента R^2 используется функция r2_score().

Итак, данные для анализа получены. Давайте теперь построим графики и посмотрим какая модель показала лучший результат:

```
fig, axes = plt.subplots(ncols=2, figsize=(10,4))
TestModels.R2_Y1.plot(ax=axes[0], kind='bar', title='R2_Y1')
TestModels.R2_Y2.plot(ax=axes[1], kind='bar', color='green', title='R2_Y2')
```



Анализ результатов и выводы

Из графиков, приведенных выше, можно сделать вывод, что лучше других с задачей справился метод *RandomForest* (случайный лес). Его коэффициенты детерминации выше остальных по обоим переменным: $R^2_{y1} \approx 99\,\%, R^2_{y2} \approx 90\,\%$ ля дальнейшего анализа давайте заново обучим нашу модель:

```
model = models[1]
model.fit(Xtrn, Ytrn)
```

При внимательном рассмотрении, может возникнуть вопрос, почему в предыдущий раз и делили зависимую выборку *Ytrn* на переменные(по столбцам), а теперь мы этого не делаем.

Дело в том, что некоторые методы, такие как *RandomForestRegressor*, может работать с несколькими прогнозируемыми переменными, а другие (например *SVR*) могут работать только с одной переменной. Поэтому на при предыдущем обучении мы использовали разбиение по столбцам, чтобы избежать ошибки в процессе построения некоторых моделей. Выбрать модель это, конечно же, хорошо, но еще неплохо бы обладать

информацией, как каждый фактор влиет на прогнозное значение. Для этого у модели есть свойство *feature_importances_*.

С помощью него, можно посмотреть вес каждого фактора в итоговой моделей:

```
model.feature_importances_
```

array([0.40717901, 0.11394948, 0.34984766, 0.00751686, 0.09158358,
0.02992342])

В нашем случае видно, что больше всего на нагрузку при обогреве и охлаждении влияют общая высота и площадь. Их общий вклад в прогнозной модели около 72%.

Также необходимо отметить, что по вышеуказанной схеме можно посмотреть влияние каждого фактора отдельно на обогрев и отдельно на охлаждение, но т. к. эти факторы у нас очень тесно коррелируют между собой ($r = 97\,\%$), мы сделали общий вывод по ним обоим который и был написан выше.

Наивный байесовский классификатор

Наивный байесовский классификатор (Naive Bayes classifier) — вероятностный классификатор на основе формулы Байеса со строгим (наивным) предположением о независимости признаков между собой при заданном классе, что сильно упрощает задачу классификации из-за оценки одномерных вероятностных плотностей вместо одной многомерной.

В данном случае, одномерная вероятностная плотность — это оценка вероятности каждого признака отдельно при условии их независимости, а многомерная — оценка вероятности комбинации всех признаков, что вытекает из случая их зависимости. Именно по этой причине данный классификатор называется наивным, поскольку позволяет сильно упростить вычисления и повысить эффективность алгоритма. Однако такое предположение не всегда является верным на практике и в ряде случаев может привести к значительному ухудшению качества прогнозов.

Наивный байесовский классификатор - специальный частный случай байесовского классификатора, основанный на дополнительном предположении, что объекты $x \in X$ описываются n статистически независимыми признаками:

$$x \equiv (\xi_1, \ldots, \xi_n) \equiv (f_1(x), \ldots, f_n(x)).$$

Предположение о независимости означает, что функции правдоподобия классов представимы в виде

$$p_y(x) = p_{y1}(\xi_1) \cdot \ldots \cdot p_{yn}(\xi_n),$$

где $p_{yj}(\xi_j)$ — плотность распределения значений j-го признака для класса y.

Предположение о независимости существенно упрощает задачу, так как оценить n одномерных плотностей гораздо легче, чем одну n -мерную плотность. К сожалению, оно крайне редко выполняется на практике, отсюда и название метода.

Наивный байесовский классификатор может быть как параметрическим, так и непараметрическим, в зависимости от того, каким методом восстанавливаются одномерные плотности.

Основные преимущества наивного байесовского классификатора — простота реализации и низкие вычислительные затраты при обучении и классификации. В тех редких случаях, когда признаки действительно независимы (или почти независимы), наивный байесовский классификатор (почти) оптимален.

Основной его недостаток — относительно низкое качество классификации в большинстве реальных задач.

Чаще всего он используется либо как примитивный эталон для сравнения различных моделей алгоритмов, либо как элементарный строительный блок в алгоритмических композициях.

Наивный Байес в задачах фильтрации спама

В контексте фильтрации спама наивный байесовский классификатор основан на частоте появления слов в сообщениях для спама и не спама, и максимизации произведения их вероятностей. Наивность в данном случае будет заключаться в предположении о независимости слов в сообщении от порядка и контекста. Тогда формула Байеса приобретает следующий вид:

$$P(C|M) \propto P(C)\prod_{i=1}^{n} P(w_i|C),\ \ w_i \in M$$

где:

- C — класс спам или не спам;
- M — сообщение;
- wi — i-е слово в сообщении M;
- $\propto$ — знак пропорциональности.

Для лучшего понимания рассмотрим следующий пример. Предположим, мы хотим классифицировать сообщение **"Hi, you won a discount and you can get the prize this evening."** и у нас есть обучающая выборка, состоящая из следующих сообщений:

Message	Class
Hi, how are you?	Not spam
Congratulations, you won a prize!	Spam
Buy the product now and get a discount!	Spam
Let's walk this evening	Not spam

Первым делом необходимо рассчитать частоту появления всех уникальных слов и их общее количество в сообщениях для спама и не спама. Затем производится расчёт вероятностей встретить каждое слово в спам и не спам сообщениях на основе этих частот. Когда в сообщении есть слова, которые раньше не встречались в обучающей выборке, используется сглаживание. Существует много различных видов сглаживаний, но суть самого простого из них заключается в добавлении **1** при подсчёте частот слов в сообщениях. Такой приём позволяет избежать проблему нулевой вероятности. Ниже приведена таблица с расчётом вероятностей для всех слов.

Word	Frequency in Not Spam	Frequency in Spam	Probability in Not Spam	Probability in Spam
Hi	1 + 1 = 2	0 + 1 = 1	2 / 28 = 0.0714	1 / 33 = 0.03
how	1 + 1 = 2	0 + 1 = 1	2 / 28 = 0.0714	1 / 33 = 0.03
are	1 + 1 = 2	0 + 1 = 1	2 / 28 = 0.0714	1 / 33 = 0.03
you	1 + 1 = 2	1 + 1 = 2	2 / 28 = 0.0714	2 / 33 = 0.06
Congratulations	0 + 1 = 1	1 + 1 = 2	1 / 28 = 0.0357	2 / 33 = 0.06
won	0 + 1 = 1	1 + 1 = 2	1 / 28 = 0.0357	2 / 33 = 0.06
a	0 + 1 = 1	2 + 1 = 3	1 / 28 = 0.0357	3 / 33 = 0.09
prize	0 + 1 = 1	1 + 1 = 2	1 / 28 = 0.0357	2 / 33 = 0.06
Buy	0 + 1 = 1	1 + 1 = 2	1 / 28 = 0.0357	2 / 33 = 0.06
the	0 + 1 = 1	1 + 1 = 2	1 / 28 = 0.0357	2 / 33 = 0.06
product	0 + 1 = 1	1 + 1 = 2	1 / 28 = 0.0357	2 / 33 = 0.06
now	0 + 1 = 1	1 + 1 = 2	1 / 28 = 0.0357	2 / 33 = 0.06
and	0 + 1 = 1	1 + 1 = 2	1 / 28 = 0.0357	2 / 33 = 0.06
get	0 + 1 = 1	1 + 1 = 2	1 / 28 = 0.0357	2 / 33 = 0.06
discount	0 + 1 = 1	1 + 1 = 2	1 / 28 = 0.0357	2 / 33 = 0.06
Let's	1 + 1 = 2	0 + 1 = 1	2 / 28 = 0.0714	1 / 33 = 0.03
walk	1 + 1 = 2	0 + 1 = 1	2 / 28 = 0.0714	1 / 33 = 0.03
this	1 + 1 = 2	0 + 1 = 1	2 / 28 = 0.0714	1 / 33 = 0.03
evening	1 + 1 = 2	0 + 1 = 1	2 / 28 = 0.0714	1 / 33 = 0.03
can	0 + 1 = 1	0 + 1 = 1	1 / 28 = 0.0357	1 / 33 = 0.03
Total amount of words	**28**	**33**		

В конце рассчитываются вероятности сообщения быть спамом или не спамом, а итоговым прогнозом будет класс с максимальной вероятностью.

$$P(C|M) = P(C) \cdot P('Hi'|C) \cdot P('you'|C) \cdot P('won'|C) \cdot P('a'|C) \cdot$$
$$\cdot P('discount'|C) \cdot P('and'|C) \cdot P('you'|C) \cdot P('can'|C) \cdot P('get'|C) \cdot$$
$$\cdot P('the'|C) \cdot P('prize'|C) \cdot P('this'|C) \cdot P('evening'|C)$$

Где:

- $C \in (Spam,\ \ Not\ \ Spam)$;
- $P(Spam) = P(Not\ \ Spam) = \frac{2}{4} = 0.5$

Вероятность сообщения быть спамом:

$$P(Spam|M) = 0.5 \cdot 0.03 \cdot 0.06 \cdot 0.06 \cdot 0.09 \cdot 0.06 \cdot 0.06 \cdot 0.06 \cdot 0.03 \cdot$$
$$\cdot 0.06 \cdot 0.06 \cdot 0.06 \cdot 0.03 \cdot 0.03 \approx 6.12 \cdot 10^{-18}$$

Вероятность того, что сообщение не является спамом:

$$P(Not\ \ Spam|M) = 0.5 \cdot 0.0714 \cdot 0.0714 \cdot 0.0357 \cdot 0.0357 \cdot 0.0357 \cdot 0.0357 \cdot$$
$$\cdot 0.0714 \cdot 0.0357 \cdot 0.0357 \cdot 0.0357 \cdot 0.0357 \cdot 0.0714 \cdot 0.0714 \approx 2.45 \cdot 10^{-18}$$

Поскольку $P(Spam|M) > P(Not\ \ Spam|M) \rightarrow$ **сообщение является спамом**. Стоит добавить, что на практике для удобства расчётов зачастую используется логарифм вероятности вместо самой вероятности.

Импорт необходимых библиотек

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from mlxtend.plotting import plot_decision_regions
```

Реализация на Python с нуля

```
class GaussianNaiveBayes:
    def fit(self, X, y):
        classes, cls_counts = np.unique(y, return_counts=True)
        n_classes = len(classes)
        self.priors = cls_counts / len(y)

        # calculate the mean and standard deviations of features by classes
        self.X_cls_mean = np.array([np.mean(X[y == c], axis=0) for c in
range(n_classes)])
        self.X_stds = np.array([np.std(X[y == c], axis=0) for c in
range(n_classes)])

    # calculate the probability density of the feature according to the Gaussian
distribution
    def pdf(self, x, mean, std):
        return (1 / (np.sqrt(2 * np.pi) * std)) * np.exp(-0.5 * ((x - mean) /
std) ** 2)

    def predict(self, X):
        pdfs = np.array([self.pdf(x, self.X_cls_mean, self.X_stds) for x in X])
        posteriors = self.priors * np.prod(pdfs, axis=2)    # shorten Bayes
formula

        return np.argmax(posteriors, axis=1)
```

Код для отрисовка графика

```
def decision_boundary_plot(X, y, X_train, y_train, clf, feature_indexes,
title=None):
    feature1_name, feature2_name = X.columns[feature_indexes]
    X_feature_columns = X.values[:, feature_indexes]
    X_train_feature_columns = X_train[:, feature_indexes]
    clf.fit(X_train_feature_columns, y_train)

    plot_decision_regions(X=X_feature_columns, y=y.values, clf=clf)
    plt.xlabel(feature1_name)
    plt.ylabel(feature2_name)
    plt.title(title)
```

Загрузка датасета

```
X1, y1 = load_iris(return_X_y=True, as_frame=True)
X1_train, X1_test, y1_train, y1_test = train_test_split(X1.values, y1.values,
random_state=0)
print(X1, y1, sep='\n')
```

```
     sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)
0                  5.1               3.5                1.4               0.2
1                  4.9               3.0                1.4               0.2
2                  4.7               3.2                1.3               0.2
3                  4.6               3.1                1.5               0.2
4                  5.0               3.6                1.4               0.2
..                 ...               ...                ...               ...
145                6.7               3.0                5.2               2.3
146                6.3               2.5                5.0               1.9
147                6.5               3.0                5.2               2.0
148                6.2               3.4                5.4               2.3
149                5.9               3.0                5.1               1.8

[150 rows x 4 columns]
0      0
1      0
2      0
3      0
4      0
      ..
145    2
146    2
147    2
148    2
149    2
Name: target, Length: 150, dtype: int64
```

Обучение моделей и оценка полученных результатов

Не смотря на свою простоту, в данном случае алгоритм показал отличный результат, классифицировав правильно абсолютно все образцы, что возможно

благодаря построению гибкой решающей границы с высокой обобщающей способностью. Из этого можно сделать интересный вывод, что в некоторых ситуациях более простые модели могут работать гораздо лучше, чем сложные, что можно будет заметить в дальнейшем на примере других алгоритмов.

Naive Bayes

```
nb_clf = GaussianNaiveBayes()
nb_clf.fit(X1_train, y1_train)
nb_clf_pred_res = nb_clf.predict(X1_test)
nb_clf_accuracy = accuracy_score(y1_test, nb_clf_pred_res)

print(f'Naive Bayes classifier accucacy: {nb_clf_accuracy}')
print(nb_clf_pred_res)


Naive Bayes classifier accucacy: 1.0
[2 1 0 2 0 2 0 1 1 1 2 1 1 1 1 0 1 1 0 0 2 1 0 0 2 0 0 1 1 0 2 1 0 2 2 1 0
 1]
```

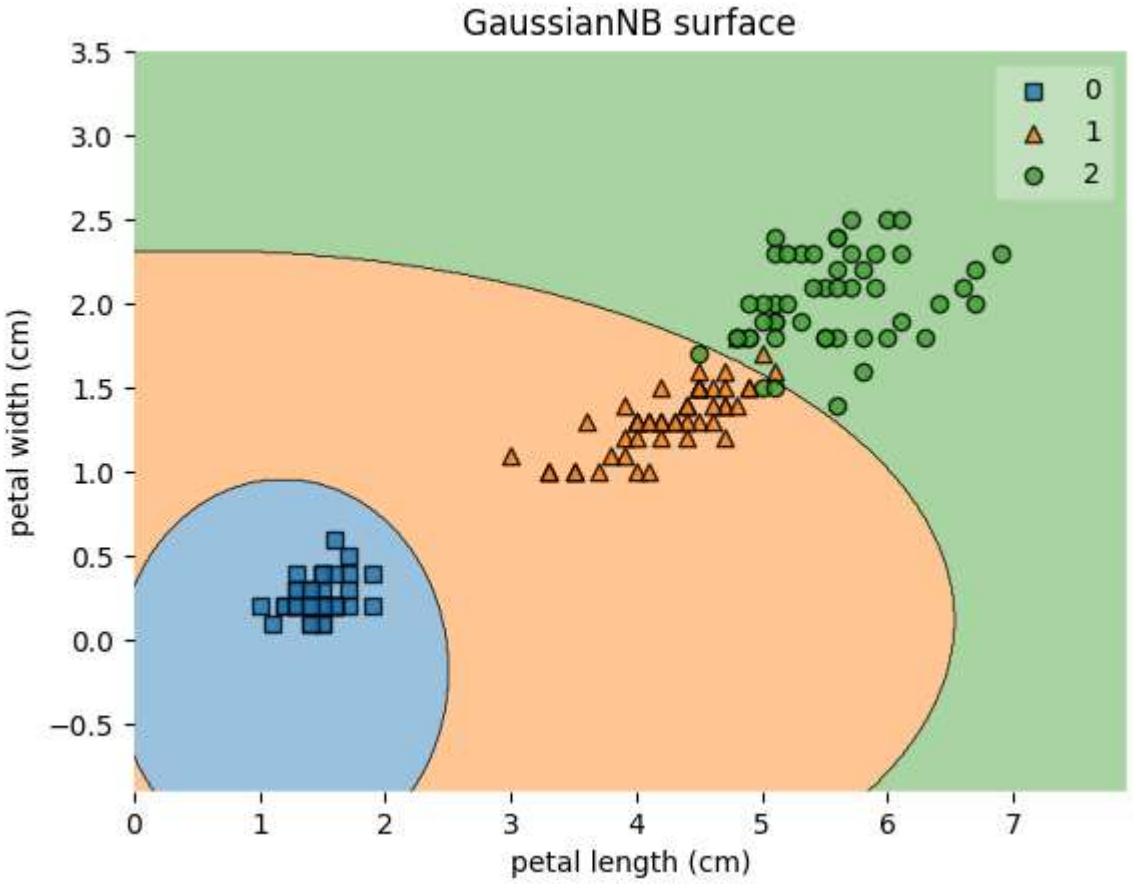
Naive Bayes (scikit-learn)

```
sk_nb_clf = GaussianNB()
sk_nb_clf.fit(X1_train, y1_train)
sk_nb_clf_pred_res = sk_nb_clf.predict(X1_test)
sk_nb_clf_accuracy = accuracy_score(y1_test, sk_nb_clf_pred_res)

print(f'sk Naive Bayes classifier accucacy: {sk_nb_clf_accuracy}')
print(sk_nb_clf_pred_res)

feature_indexes = [2, 3]
title1 = 'GaussianNB surface'
decision_boundary_plot(X1, y1, X1_train, y1_train, sk_nb_clf, feature_indexes,
title1)


sk Naive Bayes classifier accucacy: 1.0
[2 1 0 2 0 2 0 1 1 1 2 1 1 1 1 0 1 1 0 0 2 1 0 0 2 0 0 1 1 0 2 1 0 2 2 1 0
 1]
```



Преимущества и недостатки наивного байесовского классификатора

Преимущества:

- простота в реализации и интерпретации;
- практически не требуется настройка параметров;
- высокая скорость работы и точность прогнозов во многих ситуациях;
- имеет относительно хорошую устойчивость к шуму и выбросам, поскольку основан на вероятностных распределениях и наивном предположении о независимости признаков.

Недостатки:

- в случае нарушения предположения о независимости признаков, точность прогнозов может значительно снизиться;
- может отдавать предпочтение к классам с бОльшим количеством образцов в случае несбалансированных данных.