Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ Информатика и системы управления

КАФЕДРА Системы обработки информации и управления

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К НАУЧНО-ИССЛЕДОВАТЕЛЬСКОЙ РАБОТЕ

НА ТЕМУ:

Модели машинного обучения

Студент РТ5-61Б _____ В. А. Андреев
(Группа) (Подпись, дата) (И.О.Фамилия)

Руководитель курсовой работы _____ Ю. Е. Гапанюк
(Подпись, дата) (И.О.Фамилия)

Консультант _____ _____
(Подпись, дата) (И.О.Фамилия)

2023 г.

Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

УТВЕРЖДАЮ

Заведующий кафедрой _____
(Индекс)

_____ _____
(И.О.Фамилия)

« _____ » _____ 20 ____ г.

З А Д А Н И Е
на выполнение научно-исследовательской работы

по теме Модели машинного обучения

Студент группы РТ5-61Б

Андреев Виктор Алексеевич
(Фамилия, имя, отчество)

Направленность НИР (учебная, исследовательская, практическая, производственная, др.)

учебная

Источник тематики (кафедра, предприятие, НИР) НИР

График выполнения НИР: 25% к 4 нед., 50% к 8 нед., 75% к 12 нед., 100% к 15 нед.

Техническое задание _____

Решение задачи машинного обучения на основе материалов дисциплины

Оформление научно-исследовательской работы:

Расчетно-пояснительная записка на 32 листах формата А4.
Перечень графического (иллюстративного) материала (чертежи, плакаты, слайды и т.п.)

Дата выдачи задания « 7 » февраля 2023 г.

Руководитель НИР _____ Ю. Е. Гапанюк
(Подпись, дата) (И.О.Фамилия)

Студент _____ В. А. Андреев
(Подпись, дата) (И.О.Фамилия)

Примечание: Задание оформляется в двух экземплярах: один выдается студенту, второй хранится на кафедре.

Содержание

Введение

В современном мире машинное обучение является одной из наиболее перспективных и актуальных технологий, которая находит свое применение в различных сферах деятельности, начиная от медицины и финансов и заканчивая производством и транспортом. Технологии машинного обучения позволяют компьютерам обучаться на основе большого количества данных и использовать полученные знания для решения сложных задач. В данной научно-исследовательской работе рассмотрены основные принципы и методы машинного обучения. Мы изучим различные алгоритмы обучения, задачи классификации. В результате выполнения данной работы получены необходимые знания и навыки для работы с технологиями машинного обучения, что позволяет успешно применять эти технологии в практической деятельности.

Основная часть

Цель научно-исследовательской работы – разработка эффективной модели машинного обучения для решения задачи классификации на выбранном наборе данных.

Последовательность действий:

1. Выбор набора данных для построения моделей машинного обучения.
2. Проведение разведочного анализа данных. Построение графиков, необходимых для понимания структуры данных. Анализ и заполнение пропусков в данных.
3. Выбор признаков, подходящих для построения моделей. Масштабирование данных.
4. Проведение корреляционного анализа данных. Формирование промежуточных выводов о возможности построения моделей.
5. Выбор метрик для последующей оценки качества моделей.
6. Выбор наиболее подходящих моделей для решения задачи классификации.
7. Формирование обучающей и тестовой выборок на основе исходных данных.
8. Построение базового решения для выбранных моделей без подбора гиперпараметров. Производится обучение моделей на основе обучающей выборки и оценка качества моделей на основе тестовой выборки.
9. Подбор гиперпараметров для выбранных моделей с помощью методов кросс-валидации.
10. Повторение пункта 8 для найденных оптимальных значений гиперпараметров. Сравнение качества полученных моделей.
11. Формирование выводов о качестве построенных моделей на основе выбранных метрик.
12. Создать веб-приложение для демонстрации хотя бы одной модели машинного обучения. У пользователя должна быть возможность изменения хотя бы одного гиперпараметра модели, при изменении гиперпараметра модель должна перестраиваться в веб-интерфейсе.

Заключение

В результате проведенной научно-исследовательской работы была разработана эффективная модель машинного обучения для решения задачи классификации на выбранном наборе данных. В ходе работы были выполнены все поставленные задачи.

Полученные результаты позволяют сделать вывод о том, что построенные модели машинного обучения имеют высокое качество и могут быть использованы для решения задачи классификации на данном наборе данных. Веб-приложение для демонстрации модели машинного обучения позволяет пользователю изменять гиперпараметры модели и наблюдать за изменением ее качества в режиме реального времени.

Таким образом, научно-исследовательская работа по технологиям машинного обучения позволила успешно решить задачу классификации на выбранном наборе данных и создать веб-приложение для демонстрации модели машинного обучения. Полученные результаты могут быть использованы в различных областях, где требуется решение задач классификации на основе данных.

Список использованных источников информации

1. Бурков, В.Н. Методы машинного обучения в задачах классификации / В.Н. Бурков. - М.: ФИЗМАТЛИТ, 2017. - 352 с.
2. Шестаков, А.В. Технологии машинного обучения: учебное пособие / А.В. Шестаков. - М.: Изд-во МГТУ им. Н.Э. Баумана, 2018. - 232 с.
3. Кузнецов, М.П. Машинное обучение и анализ данных: учебное пособие / М.П. Кузнецов, Е.В. Кузнецова. - М.: Изд-во МГУ, 2019. - 432 с.
4. Решетников, И.В. Методы машинного обучения и анализа данных: учебник для вузов / И.В. Решетников, В.К. Курганов, И.Б. Петров. - СПб.: БХВ-Петербург, 2018. - 480 с.
5. Карпов, О.В. Технологии машинного обучения: учебное пособие для студентов вузов / О.В. Карпов, М.В. Чернышев, А.В. Шестаков. - СПб.: Питер, 2019. - 288 с.

Приложение

Ход работы в Jupyter Notebook:

Поиск и выбор набора данных для построения моделей машинного обучения

В качестве набора данных используется набор данных химического анализа вин - https://archive.ics.uci.edu/ml/machine-learning-databases/wine/wine.data

Данные являются результатом химического анализа вин, выращенных в одном и том же регионе Италии тремя разными культиваторами. Существует тринадцать различных измерений различных компонентов, содержащихся в трех типах вина.

Набор данных содержит следующие параметры: Alcohol - Алкоголь; Acid - Яблочная кислота; Ash - Пепел; Alcalinity of Ash - Щелочность пепла; Magnesium - Магний; Total Phenols - Всего фенолов; Flavanoids - Флавоноиды; Nonflavanoid Phenols - Нефлаваноидные фенолы; Proanthocyanins - Проантоцианы; Colour Intensity - Интенсивность цвета; Hue - Оттенок; OD280/OD315 of diluted wines - OD280/OD315 разбавленных вин; Proline - Пролин.

Импорт библиотек и загрузка датасета

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.datasets import *
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
# from sklearn import svm, tree
from sklearn.tree import DecisionTreeClassifier
# from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import BaggingClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, balanced_accuracy_score
from sklearn.metrics import roc_curve, roc_auc_score
from sklearn.metrics import precision_score, recall_score, f1_score,
classification_report
# from sklearn.metrics import mean_absolute_error
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
from operator import itemgetter

def make_dataframe(ds_function):
    ds = ds_function()
    df = pd.DataFrame(data= np.c_[ds['data'], ds['target']],
                      columns= list(ds['feature_names']) + ['target'])
    return df

wine = load_wine()

df = make_dataframe(load_wine)
```

Проведение разведочного анализа данных. Построение графиков, необходимых для понимания структуры данных. Анализ и заполнение пропусков в данных.

Основные характеристики датасета

```
# Первые 5 строк датасета
df.head()
```

```
   alcohol  malic_acid   ash  alcalinity_of_ash  magnesium  total_phenols  \
0    14.23        1.71  2.43               15.6      127.0           2.80
1    13.20        1.78  2.14               11.2      100.0           2.65
2    13.16        2.36  2.67               18.6      101.0           2.80
3    14.37        1.95  2.50               16.8      113.0           3.85
4    13.24        2.59  2.87               21.0      118.0           2.80

   flavanoids  nonflavanoid_phenols  proanthocyanins  color_intensity   hue  \
0        3.06                  0.28             2.29             5.64  1.04
1        2.76                  0.26             1.28             4.38  1.05
2        3.24                  0.30             2.81             5.68  1.03
3        3.49                  0.24             2.18             7.80  0.86
4        2.69                  0.39             1.82             4.32  1.04

   od280/od315_of_diluted_wines  proline  target
0                          3.92   1065.0     0.0
1                          3.40   1050.0     0.0
2                          3.17   1185.0     0.0
3                          3.45   1480.0     0.0
4                          2.93    735.0     0.0
```

```
# Размер датасета - 178 строк, 14 колонок
df.shape
```

```
(178, 14)
```

```
# Список колонок
df.columns
```

```
Index(['alcohol', 'malic_acid', 'ash', 'alcalinity_of_ash', 'magnesium',
       'total_phenols', 'flavanoids', 'nonflavanoid_phenols',
       'proanthocyanins', 'color_intensity', 'hue',
       'od280/od315_of_diluted_wines', 'proline', 'target'],
      dtype='object')
```

```
# Список колонок с типами данных
df.dtypes
```

```
alcohol                         float64
malic_acid                      float64
ash                             float64
alcalinity_of_ash               float64
magnesium                       float64
total_phenols                   float64
flavanoids                      float64
nonflavanoid_phenols            float64
proanthocyanins                 float64
color_intensity                 float64
hue                             float64
od280/od315_of_diluted_wines    float64
proline                         float64
target                          float64
dtype: object
```

```python
# Проверим наличие пустых значений
# Цикл по колонкам датасета
for col in df.columns:
    # Количество пустых значений - все значения заполнены
    temp_null_count = df[df[col].isnull()].shape[0]
    print('{} - {}'.format(col, temp_null_count))
```

```
alcohol - 0
malic_acid - 0
ash - 0
alcalinity_of_ash - 0
magnesium - 0
total_phenols - 0
flavanoids - 0
nonflavanoid_phenols - 0
proanthocyanins - 0
color_intensity - 0
hue - 0
od280/od315_of_diluted_wines - 0
proline - 0
target - 0
```

```python
# Основные статистические характеристки набора данных
df.describe()
```

```
          alcohol  malic_acid         ash  alcalinity_of_ash   magnesium  \
count  178.000000  178.000000  178.000000         178.000000  178.000000
mean    13.000618    2.336348    2.366517          19.494944   99.741573
std      0.811827    1.117146    0.274344           3.339564   14.282484
min     11.030000    0.740000    1.360000          10.600000   70.000000
25%     12.362500    1.602500    2.210000          17.200000   88.000000
50%     13.050000    1.865000    2.360000          19.500000   98.000000
75%     13.677500    3.082500    2.557500          21.500000  107.000000
max     14.830000    5.800000    3.230000          30.000000  162.000000

       total_phenols  flavanoids  nonflavanoid_phenols  proanthocyanins  \
count     178.000000  178.000000            178.000000       178.000000
mean        2.295112    2.029270              0.361854         1.590899
std         0.625851    0.998859              0.124453         0.572359
min         0.980000    0.340000              0.130000         0.410000
25%         1.742500    1.205000              0.270000         1.250000
50%         2.355000    2.135000              0.340000         1.555000
75%         2.800000    2.875000              0.437500         1.950000
max         3.880000    5.080000              0.660000         3.580000

       color_intensity         hue  od280/od315_of_diluted_wines      proline
\
count       178.000000  178.000000                    178.000000   178.000000
mean          5.058090    0.957449                      2.611685   746.893258
std           2.318286    0.228572                      0.709990   314.907474
min           1.280000    0.480000                      1.270000   278.000000
25%           3.220000    0.782500                      1.937500   500.500000
50%           4.690000    0.965000                      2.780000   673.500000
75%           6.200000    1.120000                      3.170000   985.000000
max          13.000000    1.710000                      4.000000  1680.000000

           target
count  178.000000
mean     0.938202
std      0.775035
```

```
min      0.000000
25%      0.000000
50%      1.000000
75%      2.000000
max      2.000000
```

Выводы:

1. Все значения в датасете являются числовыми.
2. Представленный набор данных не содержит пропусков.

Построение графиков для понимания структуры данных

```
sns.pairplot(df)
```

```
<seaborn.axisgrid.PairGrid at 0x7f9100859fd0>
```

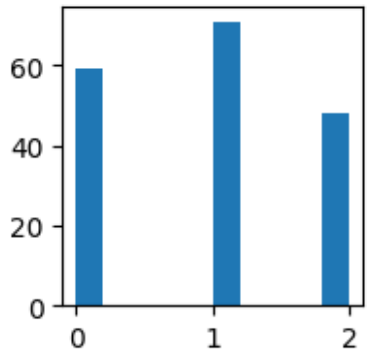
```
# Группировка по целевому признаку
sns.pairplot(df, hue="target")
```

```
<seaborn.axisgrid.PairGrid at 0x7f9123550dc0>
```

```
# Убедимся, что целевой признак подходит для задачи классификации
df['target'].unique()
```

```
array([0., 1., 2.])
```

```
# Оценим дисбаланс классов для target
fig, ax = plt.subplots(figsize=(2,2))
plt.hist(df['target'])
plt.show()
```




```
df['target'].value_counts()
```

```
1.0    71
0.0    59
2.0    48
Name: target, dtype: int64
```

```
# посчитаем дисбаланс классов
total = df.shape[0]
class_0, class_1, class_2 = df['target'].value_counts()
print('Класс 0 составляет {}%, класс 1 составляет {}%, а класс 2 составляет
{}%'
      .format(round(class_0 / total, 4)*100, round(class_1 / total, 4)*100,
round(class_2 / total, 4)*100))
```

```
Класс 0 составляет 39.89%, класс 1 составляет 33.15%, а класс 2 составляет
26.97%
```

Вывод. Дисбаланс классов присутствует, но является приемлемым.

Выбор признаков, подходящих для построения моделей. Кодирование категориальных признаков. Масштабирование данных. Формирование вспомогательных признаков, улучшающих качество моделей.

```
df.dtypes
```

```
alcohol                         float64
malic_acid                      float64
ash                             float64
alcalinity_of_ash               float64
magnesium                       float64
total_phenols                   float64
flavanoids                      float64
nonflavanoid_phenols            float64
proanthocyanins                 float64
color_intensity                 float64
hue                             float64
od280/od315_of_diluted_wines    float64
proline                         float64
target                          float64
dtype: object
```

Для построения моделей будем использовать все признаки.

Категориальные признаки отсутствуют, их кодирования не требуется.

Вспомогательные признаки для улучшения качества моделей в данном примере мы строить не будем.

Выполним масштабирование данных.

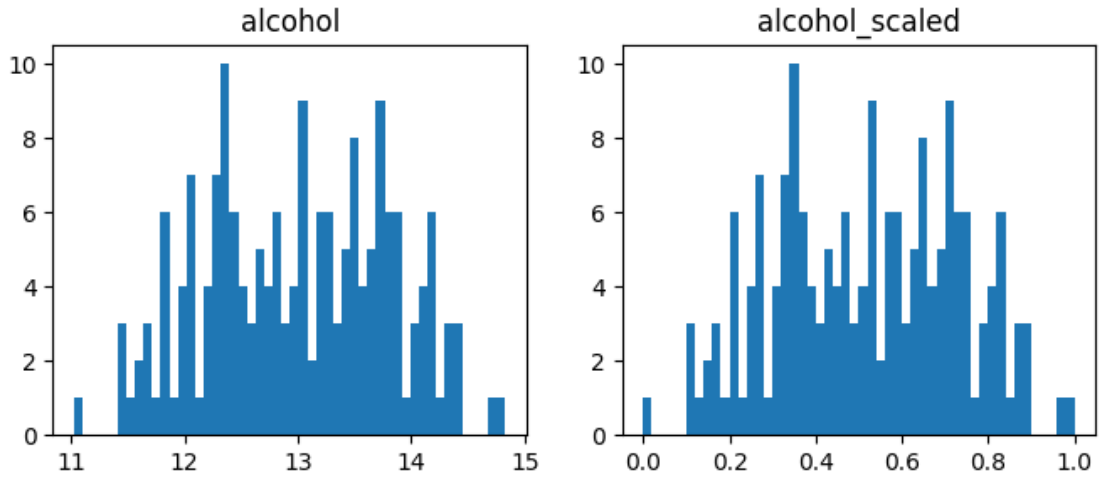
```
df.columns
```

```
Index(['alcohol', 'malic_acid', 'ash', 'alcalinity_of_ash', 'magnesium',
       'total_phenols', 'flavanoids', 'nonflavanoid_phenols',
       'proanthocyanins', 'color_intensity', 'hue',
       'od280/od315_of_diluted_wines', 'proline', 'target'],
      dtype='object')
```

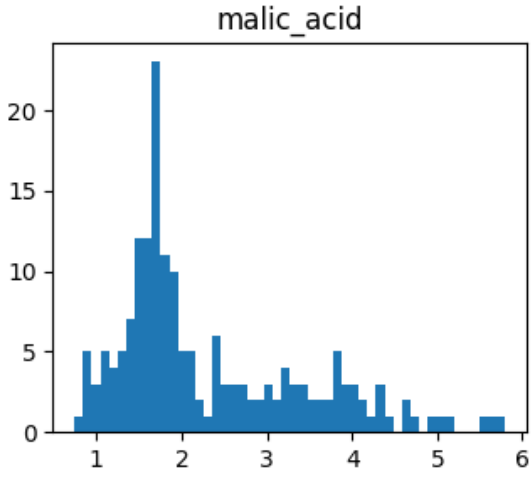
```
data_all=df

# колонки для масштабирования
scale_cols = ['alcohol', 'malic_acid', 'ash', 'alcalinity_of_ash', 'magnesium',
       'total_phenols', 'flavanoids', 'nonflavanoid_phenols',
       'proanthocyanins', 'color_intensity', 'hue',
       'od280/od315_of_diluted_wines', 'proline']

sc1 = MinMaxScaler()
sc1_data = sc1.fit_transform(data_all[scale_cols])

# Добавим масштабированные данные в набор данных
for i in range(len(scale_cols)):
    col = scale_cols[i]
    new_col_name = col + '_scaled'
    data_all[new_col_name] = sc1_data[:,i]

data_all.head()
```

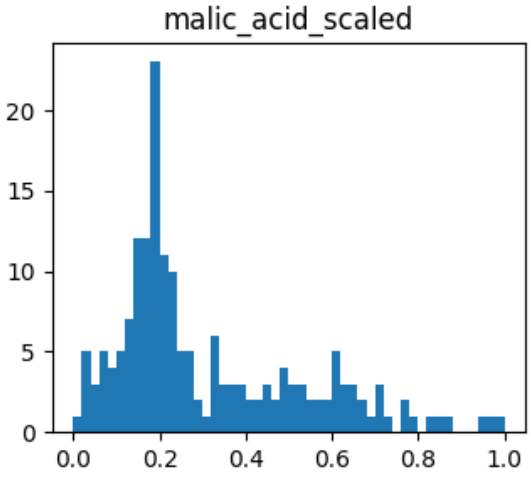
```
   alcohol  malic_acid   ash  alcalinity_of_ash  magnesium  total_phenols  \
0    14.23        1.71  2.43               15.6      127.0           2.80
1    13.20        1.78  2.14               11.2      100.0           2.65
2    13.16        2.36  2.67               18.6      101.0           2.80
3    14.37        1.95  2.50               16.8      113.0           3.85
4    13.24        2.59  2.87               21.0      118.0           2.80

   flavanoids  nonflavanoid_phenols  proanthocyanins  color_intensity  ...  \
0        3.06                  0.28             2.29             5.64  ...
1        2.76                  0.26             1.28             4.38  ...
2        3.24                  0.30             2.81             5.68  ...
3        3.49                  0.24             2.18             7.80  ...
4        2.69                  0.39             1.82             4.32  ...

   alcalinity_of_ash_scaled  magnesium_scaled  total_phenols_scaled  \
0                  0.257732          0.619565              0.627586
1                  0.030928          0.326087              0.575862
2                  0.412371          0.336957              0.627586
3                  0.319588          0.467391              0.989655
4                  0.536082          0.521739              0.627586

   flavanoids_scaled  nonflavanoid_phenols_scaled  proanthocyanins_scaled  \
0           0.573840                     0.283019                0.593060
1           0.510549                     0.245283                0.274448
2           0.611814                     0.320755                0.757098
3           0.664557                     0.207547                0.558360
4           0.495781                     0.490566                0.444795

   color_intensity_scaled  hue_scaled  od280/od315_of_diluted_wines_scaled  \
0                0.372014    0.455285                              0.970696
1                0.264505    0.463415                              0.780220
2                0.375427    0.447154                              0.695971
3                0.556314    0.308943                              0.798535
4                0.259386    0.455285                              0.608059

   proline_scaled
0        0.561341
1        0.550642
2        0.646933
3        0.857347
4        0.325963

[5 rows x 27 columns]
```

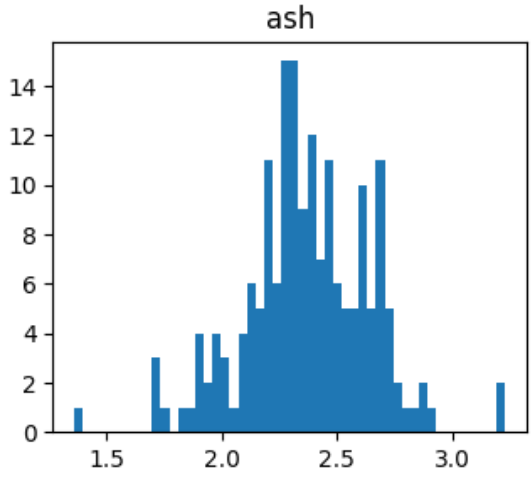
```python
# Проверим, что масштабирование не повлияло на распределение данных
for col in scale_cols:
    col_scaled = col + '_scaled'

    fig, ax = plt.subplots(1, 2, figsize=(8,3))
    ax[0].hist(data_all[col], 50)
    ax[1].hist(data_all[col_scaled], 50)
    ax[0].title.set_text(col)
    ax[1].title.set_text(col_scaled)
    plt.show()
```

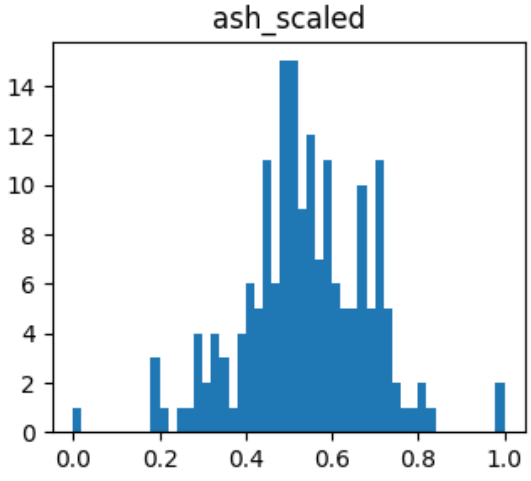

alcohol

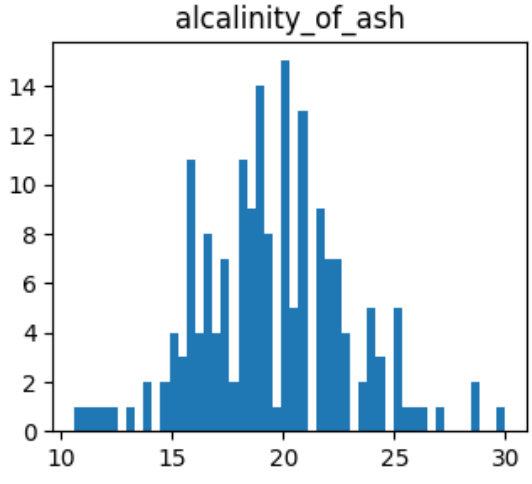
alcohol_scaled

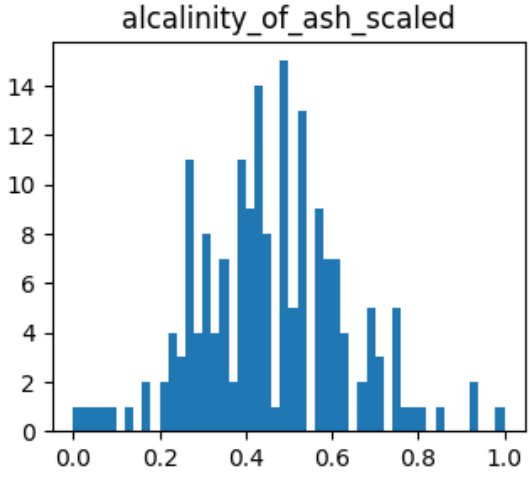

malic_acid

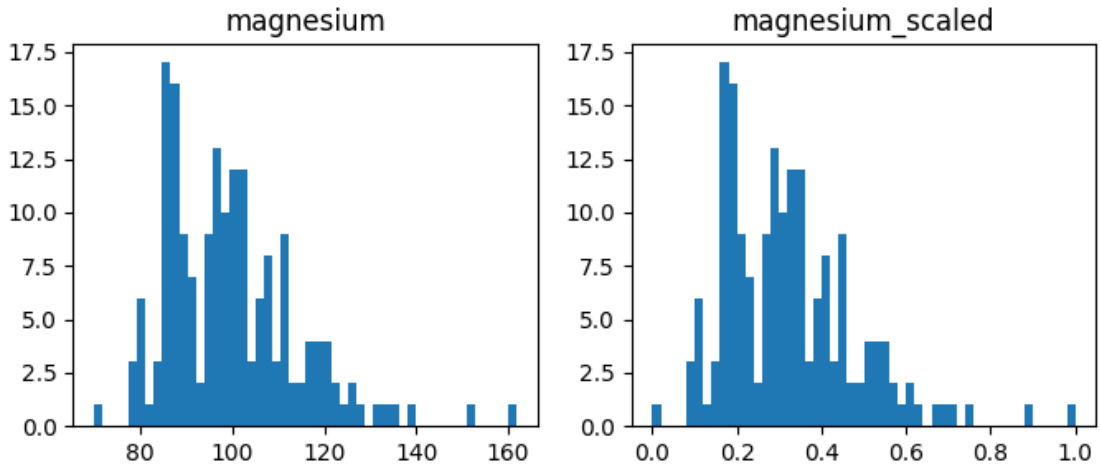

malic_acid_scaled


ash

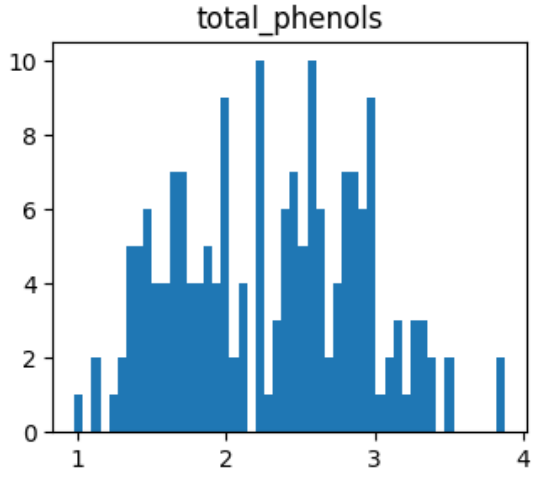

ash_scaled

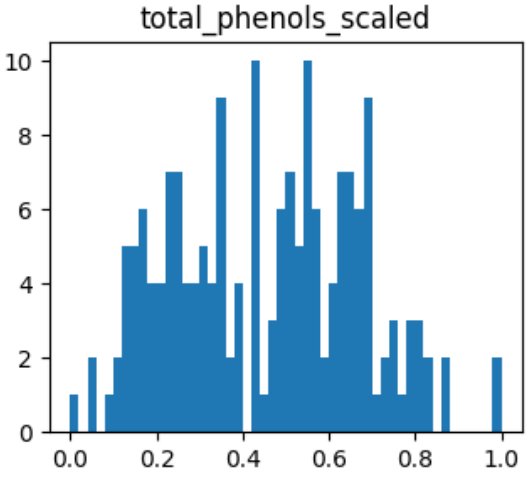

alcalinity_of_ash

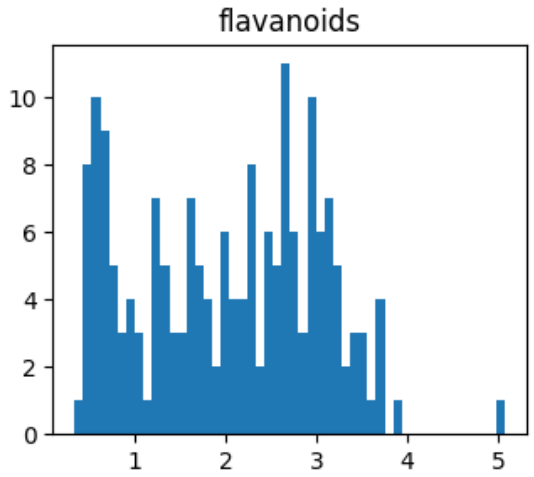

alcalinity_of_ash_scaled

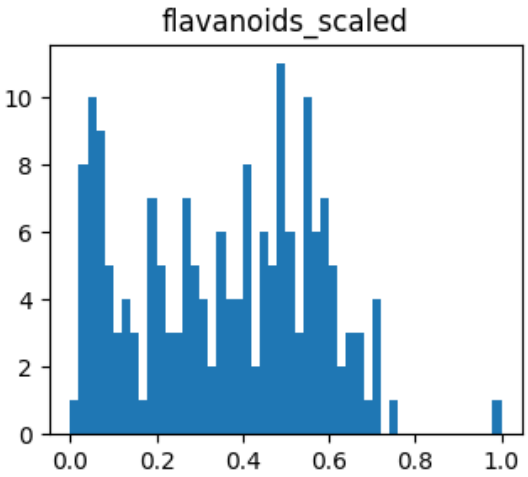

magnesium

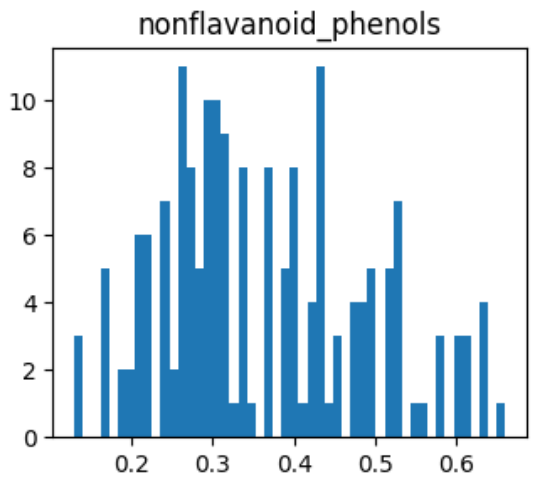
magnesium_scaled

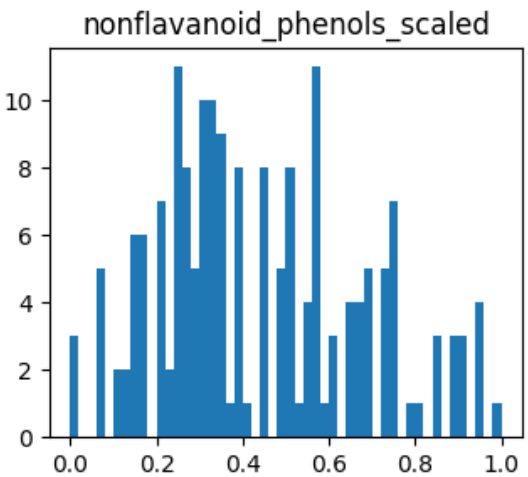

total_phenols

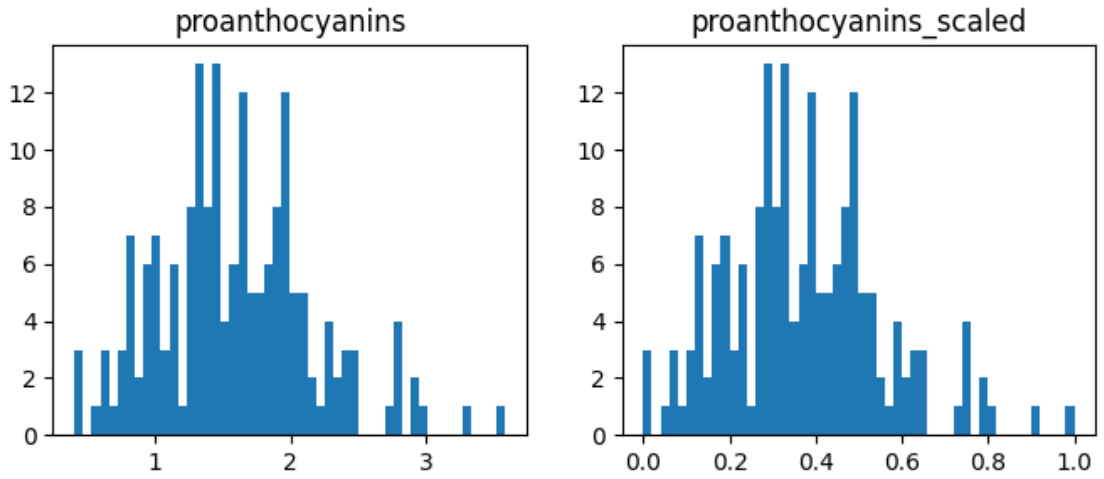

total_phenols_scaled


flavanoids

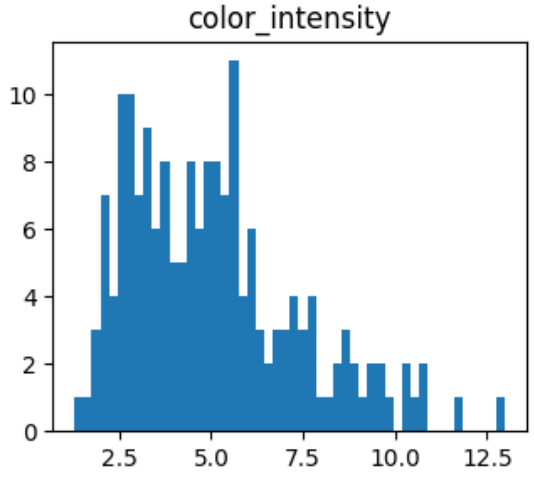

flavanoids_scaled

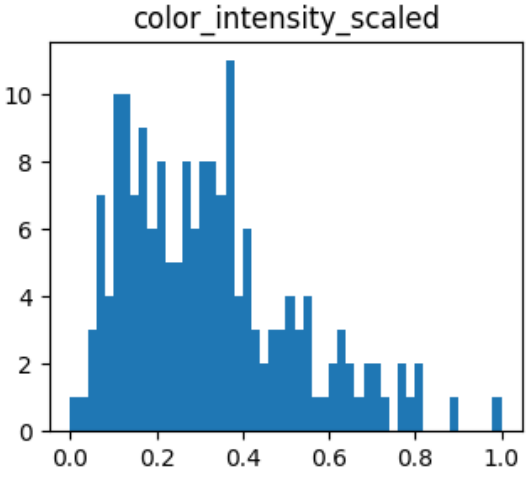

nonflavanoid_phenols

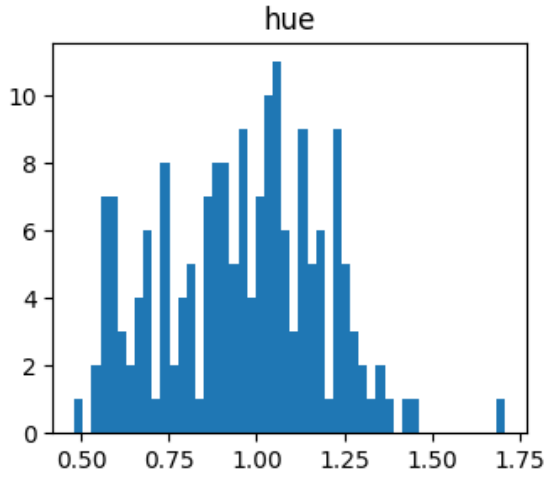

nonflavanoid_phenols_scaled

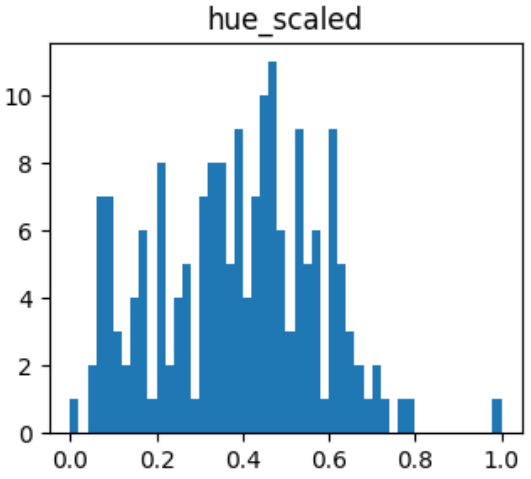

proanthocyanins
0
2
4
6
8
10
12
1
2
3

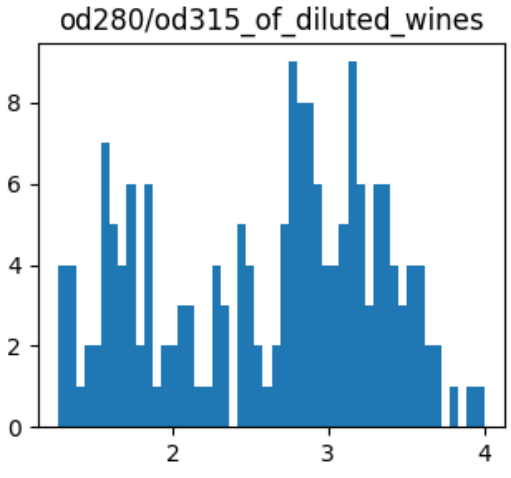
proanthocyanins_scaled
0
2
4
6
8
10
12
0.0
0.2
0.4
0.6
0.8
1.0

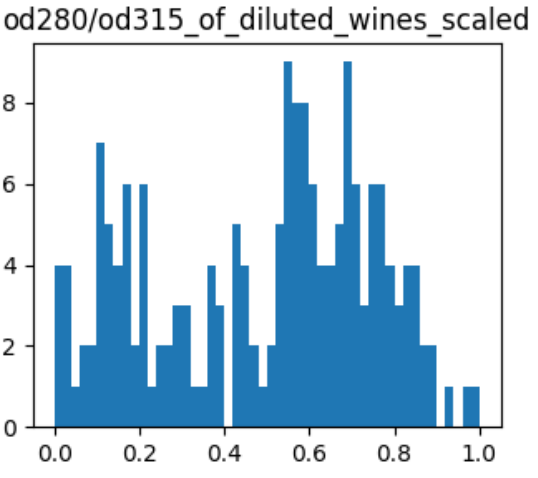

color_intensity
0
2
4
6
8
10
2.5
5.0
7.5
10.0
12.5

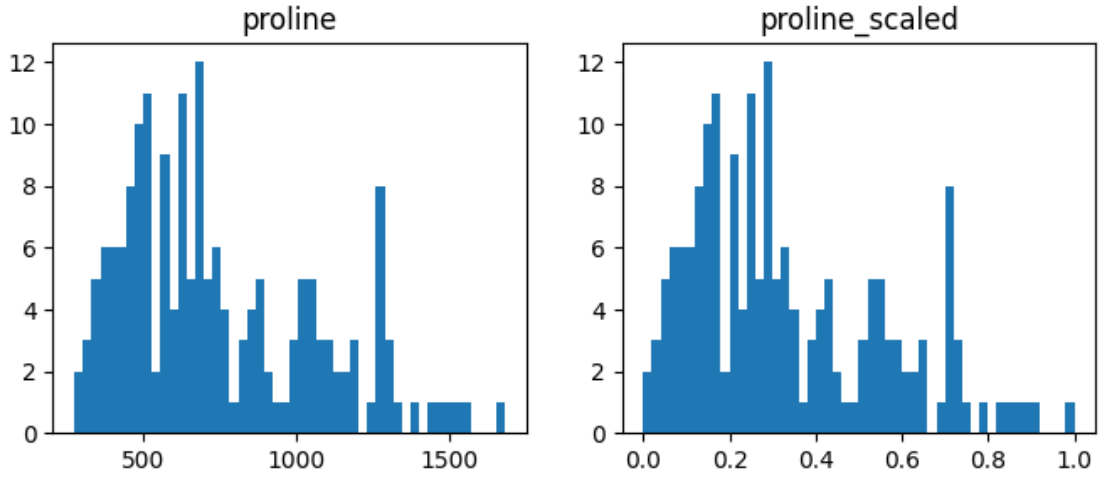

color_intensity_scaled
0
2
4
6
8
10
0.0
0.2
0.4
0.6
0.8
1.0


hue
0
2
4
6
8
10
0.50
0.75
1.00
1.25
1.50
1.75


hue_scaled
0
2
4
6
8
10
0.0
0.2
0.4
0.6
0.8
1.0


od280/od315_of_diluted_wines
0
2
4
6
8
2
3
4


od280/od315_of_diluted_wines_scaled
0
2
4
6
8
0.0
0.2
0.4
0.6
0.8
1.0



Проведение корреляционного анализа данных. Формирование промежуточных выводов о возможности построения моделей машинного обучения.

```
corr_cols_1 = scale_cols + ['target']
corr_cols_1
```

```
['alcohol',
 'malic_acid',
 'ash',
 'alcalinity_of_ash',
 'magnesium',
 'total_phenols',
 'flavanoids',
 'nonflavanoid_phenols',
 'proanthocyanins',
 'color_intensity',
 'hue',
 'od280/od315_of_diluted_wines',
 'proline',
 'target']
```

```
df_not_scaled = data_all[corr_cols_1]
```

```
df_not_scaled.head()
```

```
   alcohol  malic_acid   ash  alcalinity_of_ash  magnesium  total_phenols  \
0    14.23        1.71  2.43               15.6      127.0           2.80
1    13.20        1.78  2.14               11.2      100.0           2.65
2    13.16        2.36  2.67               18.6      101.0           2.80
3    14.37        1.95  2.50               16.8      113.0           3.85
4    13.24        2.59  2.87               21.0      118.0           2.80

   flavanoids  nonflavanoid_phenols  proanthocyanins  color_intensity   hue  \
0        3.06                  0.28             2.29             5.64  1.04
1        2.76                  0.26             1.28             4.38  1.05
2        3.24                  0.30             2.81             5.68  1.03
3        3.49                  0.24             2.18             7.80  0.86
4        2.69                  0.39             1.82             4.32  1.04

   od280/od315_of_diluted_wines  proline  target
0                          3.92   1065.0     0.0
1                          3.40   1050.0     0.0
2                          3.17   1185.0     0.0
```

```
3                         3.45   1480.0     0.0
4                         2.93    735.0     0.0
```

```
scale_cols_postfix = [x+'_scaled' for x in scale_cols]
corr_cols_2 = scale_cols_postfix + ['target']
corr_cols_2
```

```
['alcohol_scaled',
 'malic_acid_scaled',
 'ash_scaled',
 'alcalinity_of_ash_scaled',
 'magnesium_scaled',
 'total_phenols_scaled',
 'flavanoids_scaled',
 'nonflavanoid_phenols_scaled',
 'proanthocyanins_scaled',
 'color_intensity_scaled',
 'hue_scaled',
 'od280/od315_of_diluted_wines_scaled',
 'proline_scaled',
 'target']
```

```
df_scaled = data_all[corr_cols_2]
```

```
df_scaled.head()
```

```
   alcohol_scaled  malic_acid_scaled  ash_scaled  alcalinity_of_ash_scaled  \
0        0.842105           0.191700    0.572193                  0.257732
1        0.571053           0.205534    0.417112                  0.030928
2        0.560526           0.320158    0.700535                  0.412371
3        0.878947           0.239130    0.609626                  0.319588
4        0.581579           0.365613    0.807487                  0.536082

   magnesium_scaled  total_phenols_scaled  flavanoids_scaled  \
0          0.619565              0.627586           0.573840
1          0.326087              0.575862           0.510549
2          0.336957              0.627586           0.611814
3          0.467391              0.989655           0.664557
4          0.521739              0.627586           0.495781

   nonflavanoid_phenols_scaled  proanthocyanins_scaled  \
0                     0.283019                0.593060
1                     0.245283                0.274448
2                     0.320755                0.757098
3                     0.207547                0.558360
4                     0.490566                0.444795

   color_intensity_scaled  hue_scaled  od280/od315_of_diluted_wines_scaled  \
0                0.372014    0.455285                             0.970696
1                0.264505    0.463415                             0.780220
2                0.375427    0.447154                             0.695971
3                0.556314    0.308943                             0.798535
4                0.259386    0.455285                             0.608059

   proline_scaled  target
0        0.561341     0.0
1        0.550642     0.0
2        0.646933     0.0
3        0.857347     0.0
4        0.325963     0.0
```

```
fig, ax = plt.subplots(1, 1, sharex='col', sharey='row', figsize=(15,5))
fig.suptitle('Корреляционная матрица (до масштабирования)')
sns.heatmap(df_not_scaled.corr(), ax=ax, annot=True, fmt='.3f')
```

```
<Axes: >
```

```
fig, ax = plt.subplots(1, 1, sharex='col', sharey='row', figsize=(15,5))
fig.suptitle('Корреляционная матрица (после масштабирования)')
sns.heatmap(df_scaled.corr(), ax=ax, annot=True, fmt='.3f')
```

```
<Axes: >
```

Корреляционная матрица содержит коэффициенты корреляции между всеми парами признаков.

Корреляционная матрица симметрична относительно главной диагонали. На главной диагонали расположены единицы (корреляция признака самого с собой).

На основе корреляционной матрицы можно сделать следующие выводы:

1. Корреляционные матрицы для исходных и масштабированных данных совпадают.
2. Целевой признак наиболее сильно коррелирует с щелочностью пепла (0.52) и отрицательно коррелирует с флаваноидами (-0.85). Эти признаки обязательно следует оставить в модели.
3. Целевой признак слабо коррелирует с пеплом (-0.05). Скорее всего, этот признак стоит исключить из модели, возможно, он только ухудшит качество модели.
4. Целевой признак отчасти коррелирует с температурой (0.54). Этот признак стоит также оставить в модели.
5. Остальные признаки отчасти коррелируют как между собой, так и с целевым признаком. Их стои оставить в модели.
6. Большие по модулю значения коэффициентов корреляции свидетельствуют о значимой корреляции между исходными признаками и целевым признаком. На основании корреляционной матрицы можно сделать вывод о том, что данные позволяют построить модель машинного обучения.

Выбор метрик для последующей оценки качества моделей

Для задачи классификации будем использовать следующие модели:

1. Логистическая регрессия
2. Метод ближайших соседей
3. Машина опорных векторов
4. Решающее дерево
5. Бэггинг
6. Градиентный бустинг

Формирование обучающей и тестовой выборок на основе исходного набора данных

```
# На основе масштабированных данных выделим обучающую и тестовую выборки

y = df_scaled['target']
x = df_scaled.drop('target', axis = 1).drop('ash_scaled', axis = 1)

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.4,
random_state = 7)
```

```
x_train.shape, x_test.shape
```

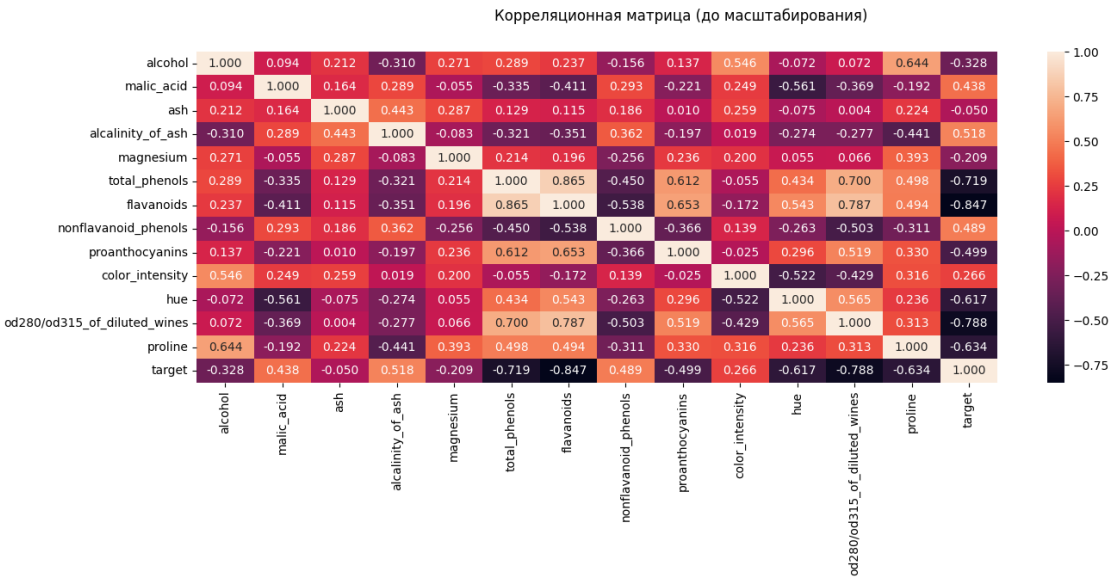
```
((106, 12), (72, 12))
```

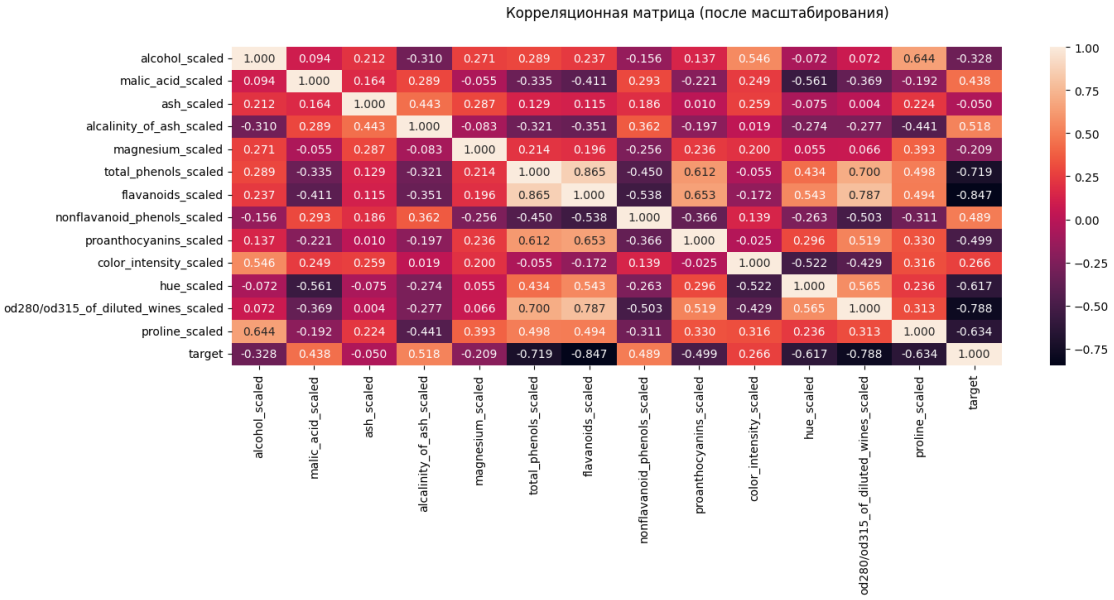
Построение базового решения (baseline) для выбранных моделей без подбора гиперпараметров. Производится обучение моделей на основе обучающей выборки и оценка качества моделей на основе тестовой выборки.

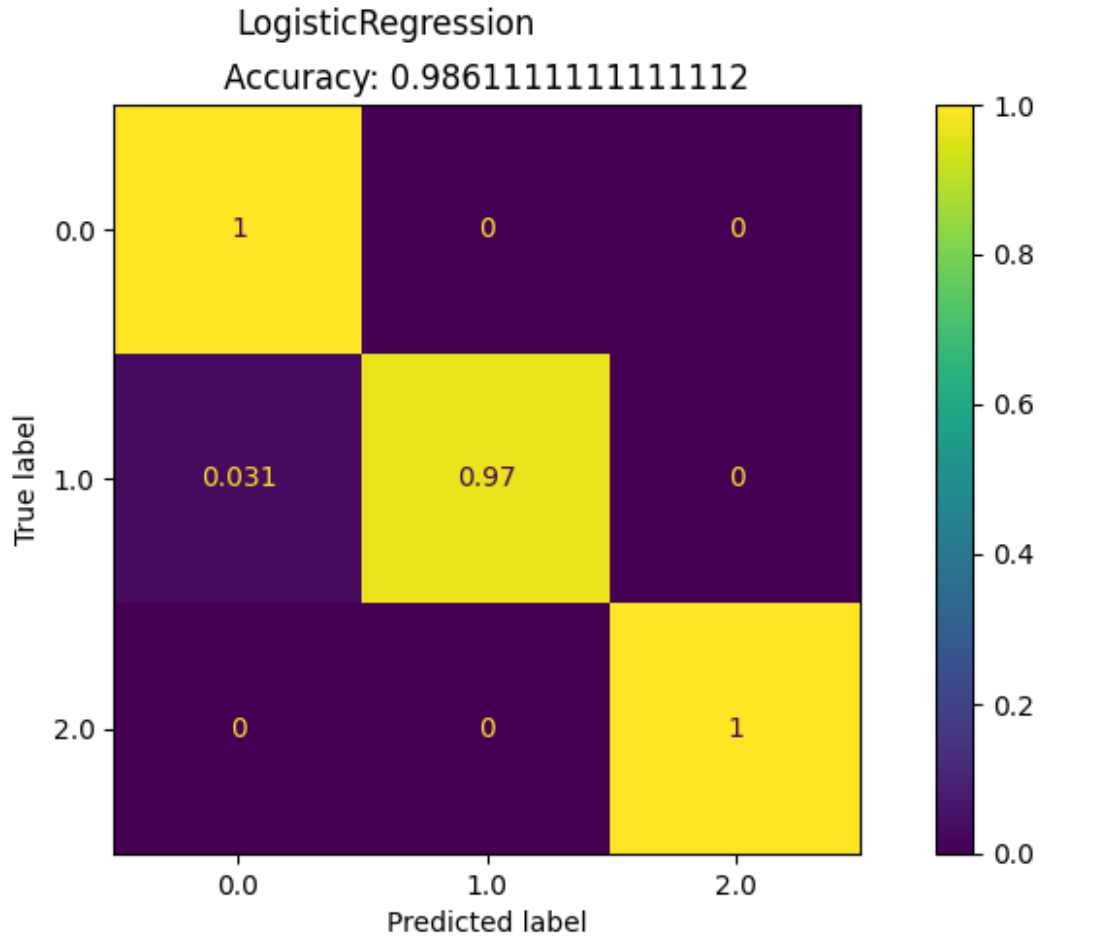
```
# Модели
clas_models = {'LogisticRegression': LogisticRegression(),
               'KNN_10':KNeighborsClassifier(n_neighbors=10),
               'SVC':SVC(probability=True),
               'DecisionTree':DecisionTreeClassifier(),
               'Bagging':BaggingClassifier(),
               'GradientBoosting':GradientBoostingClassifier()}

class MetricLogger:

    def __init__(self):
        self.df = pd.DataFrame(
            {'metric': pd.Series([], dtype='str'),
            'alg': pd.Series([], dtype='str'),
            'value': pd.Series([], dtype='float')})

    def add(self, metric, alg, value):
        """
        Добавление значения
        """
        # Удаление значения если оно уже было ранее добавлено

self.df.drop(self.df[(self.df['metric']==metric)&(self.df['alg']==alg)].index,
inplace = True)
        # Добавление нового значения
        temp = [{'metric':metric, 'alg':alg, 'value':value}]
        self.df = self.df.append(temp, ignore_index=True)

    def get_data_for_metric(self, metric, ascending=True):
        """
        Формирование данных с фильтром по метрике
        """
        temp_data = self.df[self.df['metric']==metric]
        temp_data_2 = temp_data.sort_values(by='value', ascending=ascending)
        return temp_data_2['alg'].values, temp_data_2['value'].values

    def plot(self, str_header, metric, ascending=True, figsize=(5, 5)):
        """
        Вывод графика
        """
        array_labels, array_metric = self.get_data_for_metric(metric,
ascending)
        fig, ax1 = plt.subplots(figsize=figsize)
        pos = np.arange(len(array_metric))
        rects = ax1.barh(pos, array_metric,
                         align='center',
                         height=0.5,
                         tick_label=array_labels)
        ax1.set_title(str_header)
        for a,b in zip(pos, array_metric):
```

```
            plt.text(0.5, a-0.05, str(round(b,3)), color='white')
        plt.show()

# Сохранение метрик
clasMetricLogger = MetricLogger()

def clas_train_model(model_name, model, clasMetricLogger):
    model.fit(x_train, y_train)
    # Предсказание значений
    Y_pred = model.predict(x_test)

    accuracy = accuracy_score(y_test.values, Y_pred)

    clasMetricLogger.add('accuracy', model_name, accuracy)

    fig, ax = plt.subplots(nrows=1, figsize=(10,5))

    cm = confusion_matrix(y_test, Y_pred, labels=np.unique(df_scaled.target),
normalize='true')
    disp = ConfusionMatrixDisplay(confusion_matrix=cm,
display_labels=np.unique(df_scaled.target))
    disp.plot(ax=ax)
    ax.set_title("Accuracy: {}".format(accuracy_score(y_test.values, Y_pred)))

    fig.suptitle(model_name)
    plt.show()


for model_name, model in clas_models.items():
    clas_train_model(model_name, model, clasMetricLogger)
```

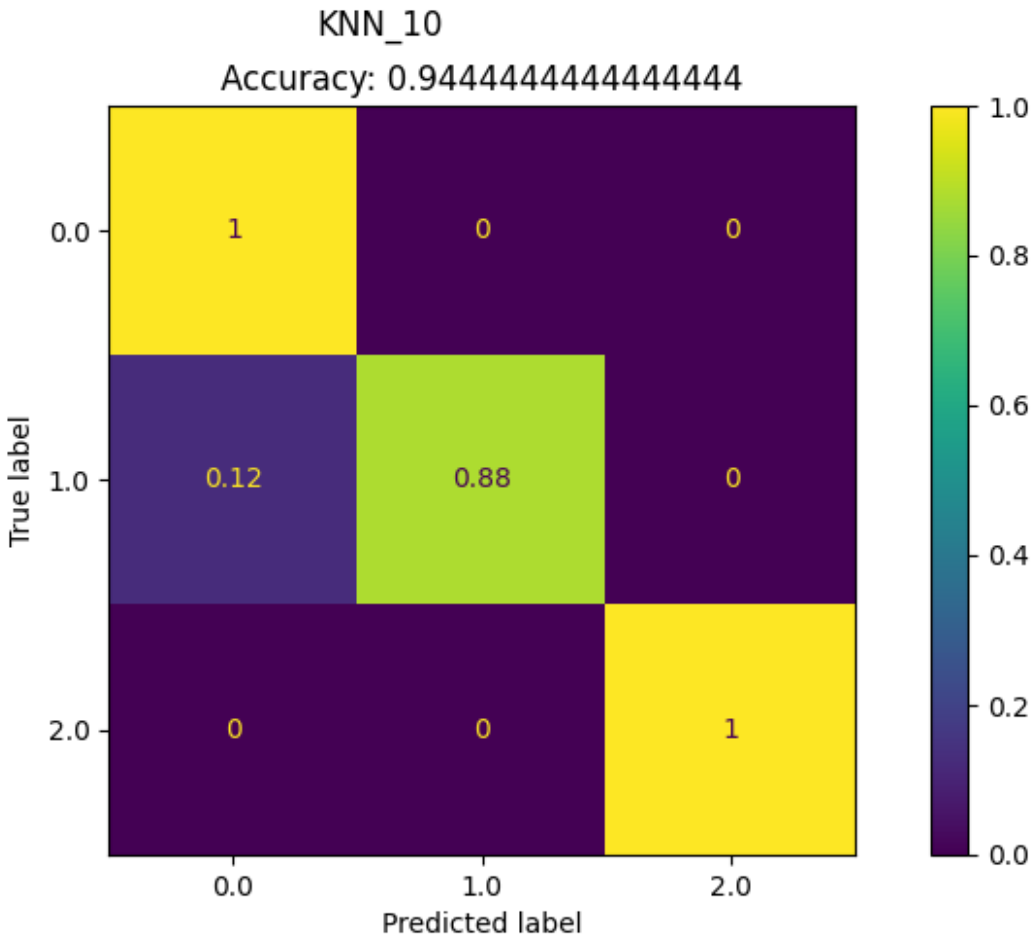
```
/var/folders/jr/9bbh58ys2356w7ntm3zfqcqw0000gn/T/ipykernel_61763/368643467.py:1
7: FutureWarning: The frame.append method is deprecated and will be removed
from pandas in a future version. Use pandas.concat instead.
  self.df = self.df.append(temp, ignore_index=True)
```



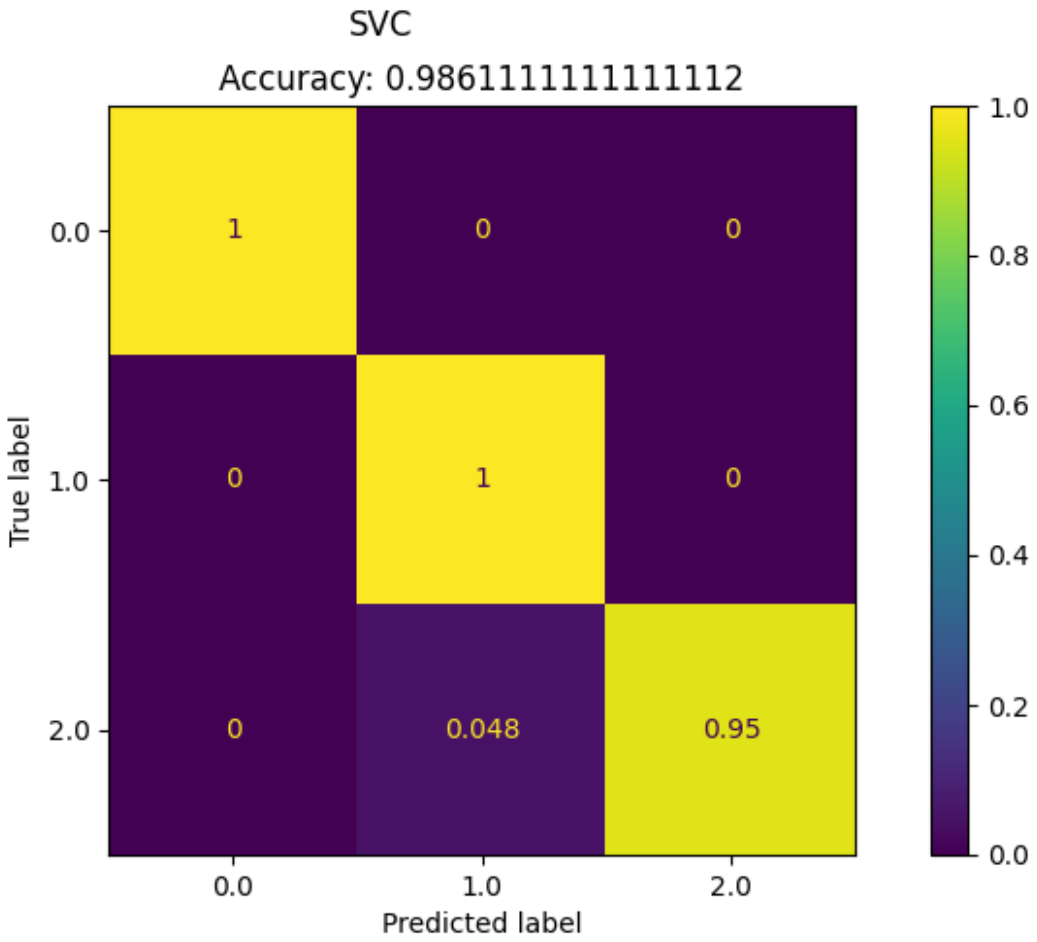




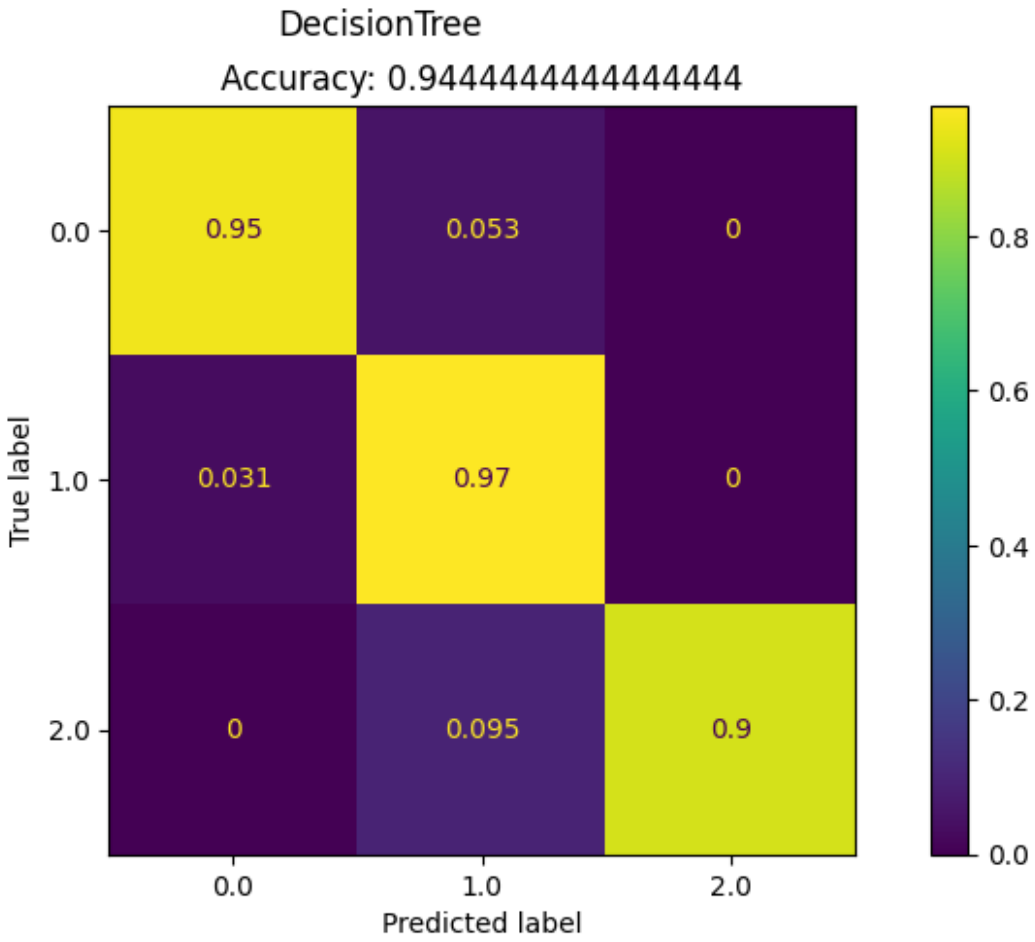

```
/var/folders/jr/9bbh58ys2356w7ntm3zfqcqw0000gn/T/ipykernel_61763/368643467.py:1
7: FutureWarning: The frame.append method is deprecated and will be removed
from pandas in a future version. Use pandas.concat instead.
  self.df = self.df.append(temp, ignore_index=True)
```



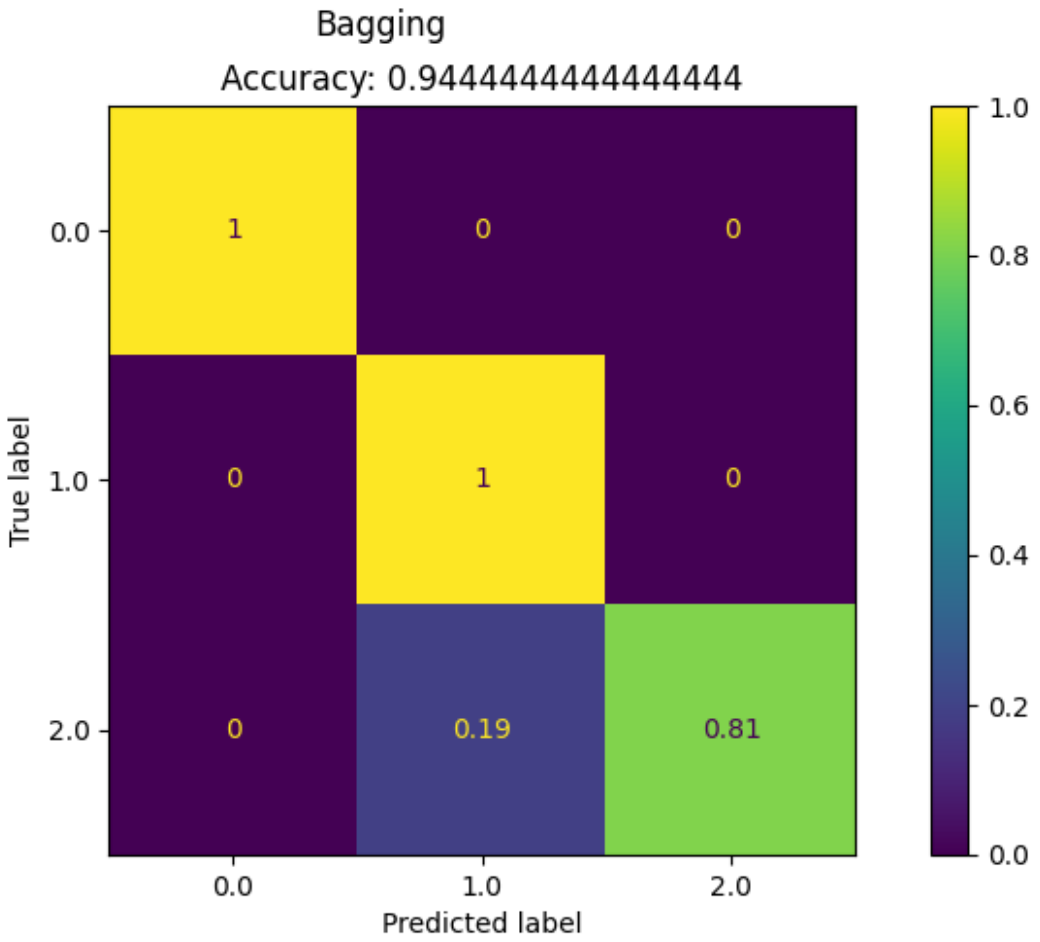

```
/var/folders/jr/9bbh58ys2356w7ntm3zfqcqw0000gn/T/ipykernel_61763/368643467.py:1
7: FutureWarning: The frame.append method is deprecated and will be removed
from pandas in a future version. Use pandas.concat instead.
  self.df = self.df.append(temp, ignore_index=True)
```



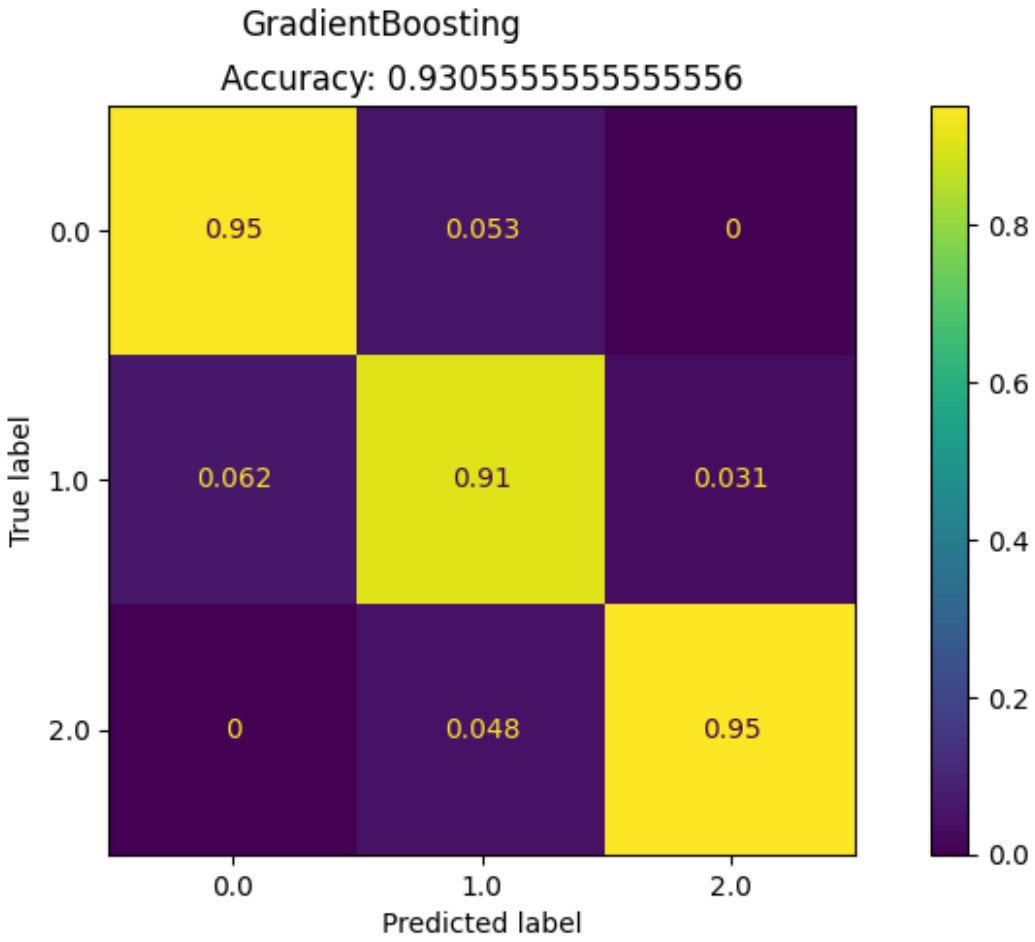

```
/var/folders/jr/9bbh58ys2356w7ntm3zfqcqw0000gn/T/ipykernel_61763/368643467.py:1
7: FutureWarning: The frame.append method is deprecated and will be removed
from pandas in a future version. Use pandas.concat instead.
  self.df = self.df.append(temp, ignore_index=True)
```




```
/var/folders/jr/9bbh58ys2356w7ntm3zfqcqw0000gn/T/ipykernel_61763/368643467.py:1
7: FutureWarning: The frame.append method is deprecated and will be removed
from pandas in a future version. Use pandas.concat instead.
  self.df = self.df.append(temp, ignore_index=True)
```




```
/var/folders/jr/9bbh58ys2356w7ntm3zfqcqw0000gn/T/ipykernel_61763/368643467.py:1
7: FutureWarning: The frame.append method is deprecated and will be removed
from pandas in a future version. Use pandas.concat instead.
  self.df = self.df.append(temp, ignore_index=True)
```



Подбор гиперпараметров для выбранных моделей. Рекомендуется использовать методы кросс-валидации. В зависимости от используемой библиотеки можно применять функцию GridSearchCV, использовать перебор параметров в цикле, или использовать другие методы.

```
x_train.shape
```

```
(106, 12)
```

```
n_range = np.array(range(2,31,1))
tuned_parameters = [{'n_neighbors': n_range}]
tuned_parameters
```

```
[{'n_neighbors': array([ 2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15,
16, 17, 18,
        19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30])}]
```

```
%%time
clf_gs = GridSearchCV(KNeighborsClassifier(), tuned_parameters, cv=5,
scoring='accuracy')
clf_gs.fit(x_train, y_train)
```

```
CPU times: user 641 ms, sys: 62.8 ms, total: 704 ms
Wall time: 388 ms
```

```
GridSearchCV(cv=5, estimator=KNeighborsClassifier(),
             param_grid=[{'n_neighbors': array([ 2,  3,  4,  5,  6,  7,  8,  9,
10, 11, 12, 13, 14, 15, 16, 17, 18,
       19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30])}],
             scoring='accuracy')
```

```
# Лучшая модель
clf_gs.best_estimator_
```

```
KNeighborsClassifier(n_neighbors=9)
```

```
clf_gs_best_params_txt = str(clf_gs.best_params_['n_neighbors'])
clf_gs_best_params_txt
```

```
'9'
```

```
# Изменение качества на тестовой выборке в зависимости от K-соседей
plt.plot(n_range, clf_gs.cv_results_['mean_test_score'])
```

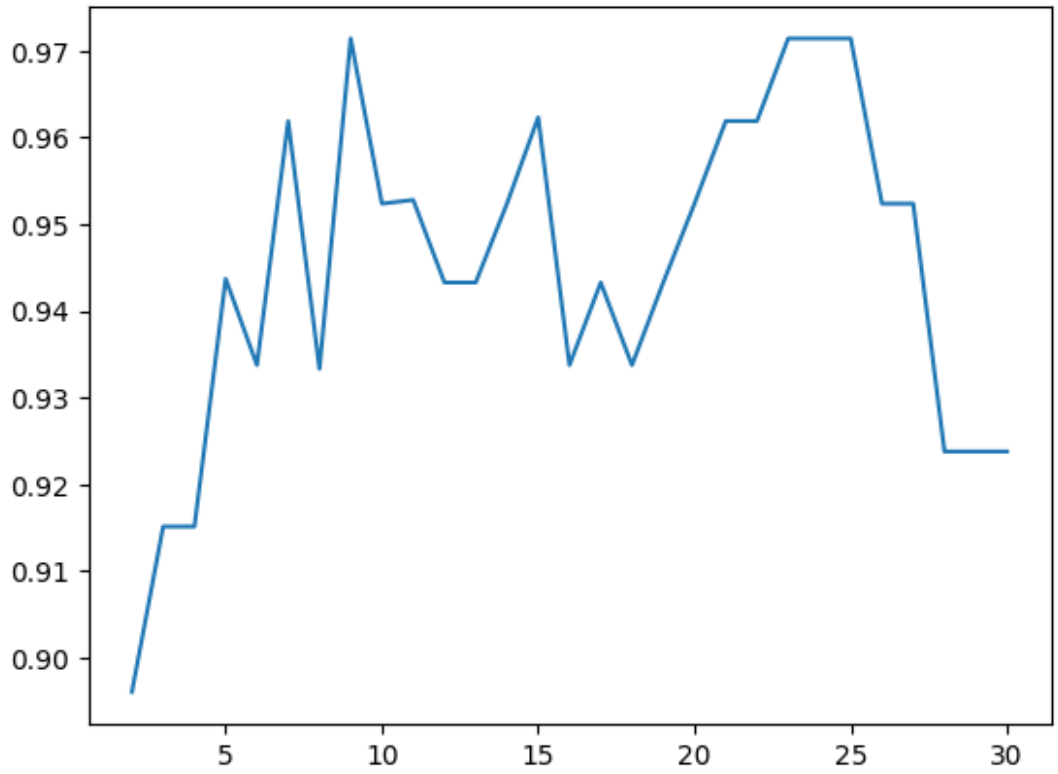
```
[<matplotlib.lines.Line2D at 0x7f91257a0b20>]
```

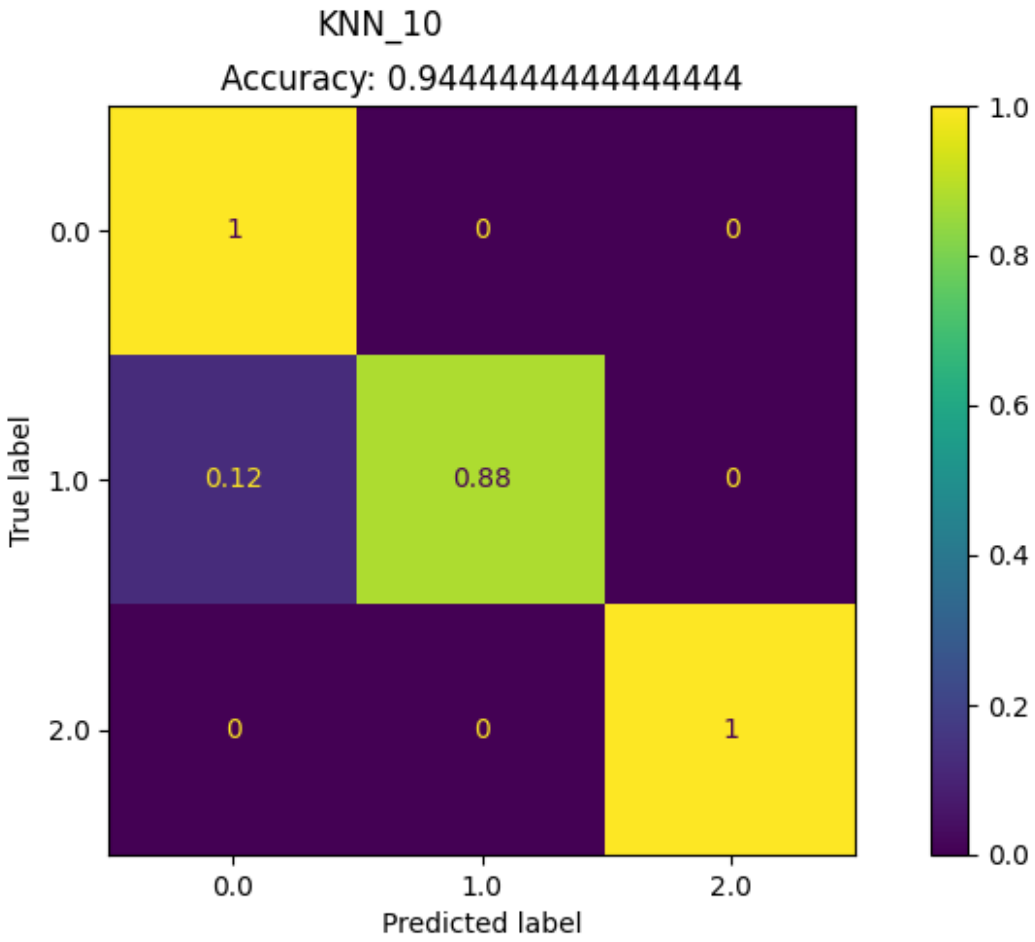
Повторение для найденных оптимальных значений гиперпараметров. Сравнение качества полученных моделей с качеством baseline-моделей.

```
clas_models_grid = {'KNN_10':KNeighborsClassifier(n_neighbors=10),
                    str('KNN_' +
clf_gs_best_params_txt):clf_gs.best_estimator_}
```

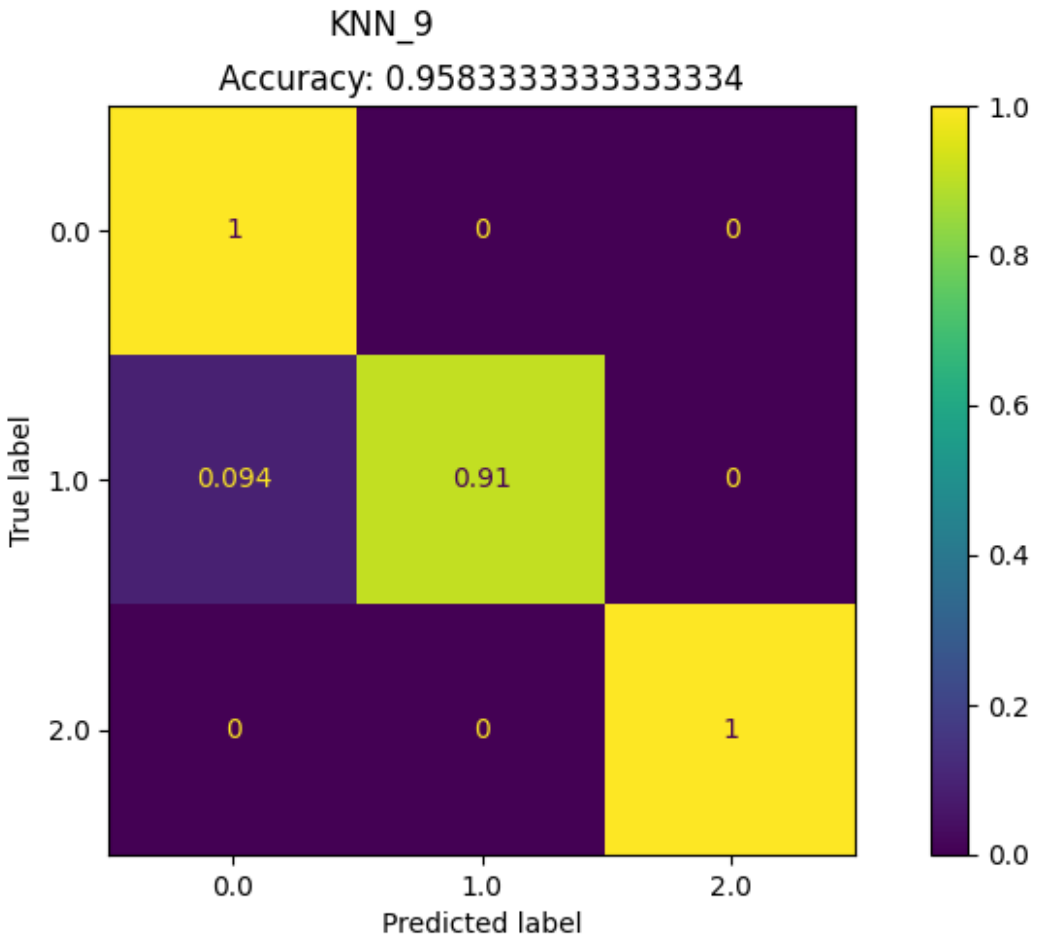
```
for model_name, model in clas_models_grid.items():
    clas_train_model(model_name, model, clasMetricLogger)
```

```
/var/folders/jr/9bbh58ys2356w7ntm3zfqcqw0000gn/T/ipykernel_61763/368643467.py:1
7: FutureWarning: The frame.append method is deprecated and will be removed
from pandas in a future version. Use pandas.concat instead.
  self.df = self.df.append(temp, ignore_index=True)
```






```
/var/folders/jr/9bbh58ys2356w7ntm3zfqcqw0000gn/T/ipykernel_61763/368643467.py:1
7: FutureWarning: The frame.append method is deprecated and will be removed
from pandas in a future version. Use pandas.concat instead.
  self.df = self.df.append(temp, ignore_index=True)
```

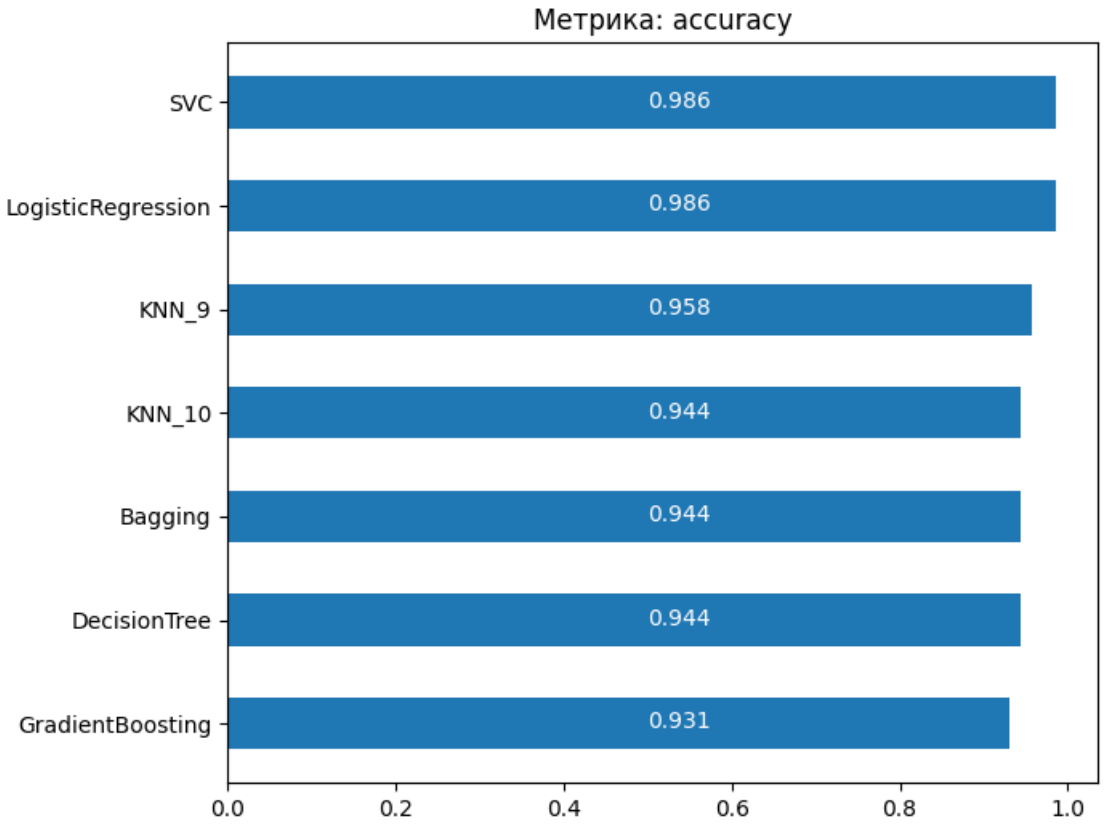


Формирование выводов о качестве построенных моделей на основе выбранных метрик. Результаты сравнения качества рекомендуется отобразить в виде графиков и сделать выводы в форме текстового описания.

```
# Метрики качества модели
clas_metrics = clasMetricLogger.df['metric'].unique()
clas_metrics
```

```
array(['accuracy'], dtype=object)
```

```
# Построим графики метрик качества модели
for metric in clas_metrics:
    clasMetricLogger.plot('Метрика: ' + metric, metric, figsize=(7, 6))
```



Вывод: лучшими оказались модели на основе метода опорных векторов и логистической регрессии.

Веб-приложение для демонстрации метода K-ближайших соседей:

```
import streamlit as st
import seaborn as sns
import pandas as pd
import numpy as np
from sklearn.datasets import *
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score
from sklearn.neighbors import KNeighborsClassifier
import matplotlib.pyplot as plt


@st.cache
def preprocess_data(data_in):
    '''
    Масштабирование признаков, функция возвращает X и y для кросс-валидации
    '''
    data_out = data_in.copy()
    # Числовые колонки для масштабирования
    scale_cols = ['alcohol', 'malic_acid', 'ash', 'alcalinity_of_ash',
'magnesium',
       'total_phenols', 'flavanoids', 'nonflavanoid_phenols',
       'proanthocyanins', 'color_intensity', 'hue',
       'od280/od315_of_diluted_wines', 'proline']
    new_cols = []
    sc1 = MinMaxScaler()
    sc1_data = sc1.fit_transform(data_out[scale_cols])
    for i in range(len(scale_cols)):
        col = scale_cols[i]
        new_col_name = col + '_scaled'
        new_cols.append(new_col_name)
        data_out[new_col_name] = sc1_data[:,i]
    return data_out[new_cols], data_out['target']

st.sidebar.header('Метод ближайших соседей')

def make_dataframe(ds_function):
    ds = ds_function()
    df = pd.DataFrame(data= np.c_[ds['data'], ds['target']],
                      columns= list(ds['feature_names']) + ['target'])
    return df

wine = load_wine()

data = make_dataframe(load_wine)

cv_slider = st.sidebar.slider('Количество фолдов:', min_value=3, max_value=10,
value=3, step=1)
step_slider = st.sidebar.slider('Шаг для соседей:', min_value=1, max_value=50,
value=10, step=1)

if st.checkbox('Показать корреляционную матрицу'):
    fig1, ax = plt.subplots(figsize=(10,5))
    sns.heatmap(data.corr(), annot=True, fmt='.2f')
    st.pyplot(fig1)
```

```
#Количество записей
data_len = data.shape[0]
#Вычислим количество возможных ближайших соседей
rows_in_one_fold = int(data_len / cv_slider)
allowed_knn = int(rows_in_one_fold * (cv_slider-1))
st.write('Количество строк в наборе данных - {}'.format(data_len))
st.write('Максимальное допустимое количество ближайших соседей с учетом
выбранного количества фолдов - {}'.format(allowed_knn))

# Подбор гиперпараметра
n_range_list = list(range(1,allowed_knn,step_slider))
n_range = np.array(n_range_list)
st.write('Возможные значения соседей - {}'.format(n_range))
tuned_parameters = [{'n_neighbors': n_range}]

data_X, data_y = preprocess_data(data)
clf_gs = GridSearchCV(KNeighborsClassifier(), tuned_parameters, cv=cv_slider,
scoring='accuracy')
clf_gs.fit(data_X, data_y)

st.subheader('Оценка качества модели')

st.write('Лучшее значение параметров - {}'.format(clf_gs.best_params_))

# Изменение качества на тестовой выборке в зависимости от К-соседей
fig1 = plt.figure(figsize=(7,5))
ax = plt.plot(n_range, clf_gs.cv_results_['mean_test_score'])
st.pyplot(fig1)
```