Московский государственный технический университет им. Н.Э. Баумана
Факультет «Информатика и системы управления»

Кафедра «Системы обработки информации и управления»

«Методы машинного обучения»

Отчет по Лабораторной работе №6

Ансамбли моделей машинного обучения.

Выполнила:

студентка группы ИУ5-22М

Петрова Ирина

Проверил: доцент, к.т.н.

Гапанюк Ю. Е.

Москва, 2020

Лабораторная работа №6. Ансамбли моделей машинного обучения.

Цель лабораторной работы: изучение ансамблей моделей машинного обучения.

Требования к отчету: отчет по лабораторной работе должен содержать:

- титульный лист; описание задания; текст программы;
- экранные формы с примерами выполнения
- программы.
-

Задание:

1. Выберите набор данных (датасет) для решения задачи классификации или регресии.
2. В случае необходимости проведите удаление или заполнение пропусков и кодирование категориальных признаков.
3. С использованием метода train_test_split разделите выборку на обучающую и тестовую.
4. Обучите две ансамблевые модели. Оцените качество моделей с помощью одной из подходящих для задачи метрик. Сравните качество полученных моделей.
5. Произведите для каждой модели подбор значений одного гиперпараметра. В зависимости от используемой библиотеки можно применять функцию GridSearchCV, использовать перебор параметров в цикле, или использовать другие методы.
6. Повторите пункт 4 для найденных оптимальных значений гиперпараметров. Сравните качество полученных моделей с качеством моделей, полученных в пункте 4.

Текстовое описание набора данных

Используется набор данных, использующий данные химического анализа для установления происхождения вина: https://archive.ics.uci.edu/ml/datasets/Wine (https://archive.ics.uci.edu/ml/datasets/Wine)

Эти данные являются результатами химического анализа вин, выращенных в одном регионе Италии, но полученных из трех различных сортов. В результате анализа было определено 13 компонентов, содержащихся в каждом из трех видов вин.

Датасет содержит следующие колонки:

- Алкоголь
- Яблочная кислота
- Зола
- Щелочность золы
- Магний
- Всего фенолов
- Флаваноиды
- Нефлаваноидные фенолы
- Проантоцианы
- Интенсивность цвета
- Оттенок
- OD280 / OD315 (разбавленность вина)
- Пролин

Импорт библиотек

In [1]:

```
import numpy as np import
pandas as pd import seaborn as
sns import matplotlib.pyplot as
plt
from sklearn.preprocessing import MinMaxScaler
from sklearn.linear_model import LinearRegression, LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsRegressor, KNeighborsClassifier
from sklearn.metrics import accuracy_score, balanced_accuracy_score
from sklearn.metrics import precision_score, recall_score, f1_score, classification_rep
ort
from sklearn.metrics import confusion_matrix from
sklearn.metrics import plot_confusion_matrix from
sklearn.model_selection import GridSearchCV
from sklearn.metrics import mean_absolute_error, mean_squared_error, mean_squared_log_e
rror, median_absolute_error, r2_score
from sklearn.metrics import roc_curve, roc_auc_score
from sklearn.svm import SVC, NuSVC, LinearSVC, OneClassSVM, SVR, NuSVR, LinearSVR from
sklearn.tree import DecisionTreeClassifier, DecisionTreeRegressor, export_graphviz from
sklearn.ensemble import RandomForestClassifier, RandomForestRegressor from
sklearn.ensemble import ExtraTreesClassifier, ExtraTreesRegressor from sklearn.ensemble
import GradientBoostingClassifier, GradientBoostingRegressor from gmdhpy import gmdh
%matplotlib inline  sns.set(style="ticks")
```

1. Выбор датасета

In [2]:

```
from sklearn.datasets import *
```

In [3]:

```
wine = load_wine()
```

In [4]:

```
train = pd.DataFrame(data= np.c_[wine['data'], wine['target']],

columns= wine['feature_names'] + ['target']) In [5]:
```

```
# Проверим наличие пустых значений
train.isnull().sum()
```

Out[5]:

```
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
dtype: int64
```

3. Разделение выборки на обучающую и тестовую

In [6]:

```
# 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']
```

In [7]:

```
# Воспользуемся наличием тестовых выборок,
# включив их в корреляционную матрицу
corr_cols_1 = scale_cols + ['target']
corr_cols_1
```

Out[7]:

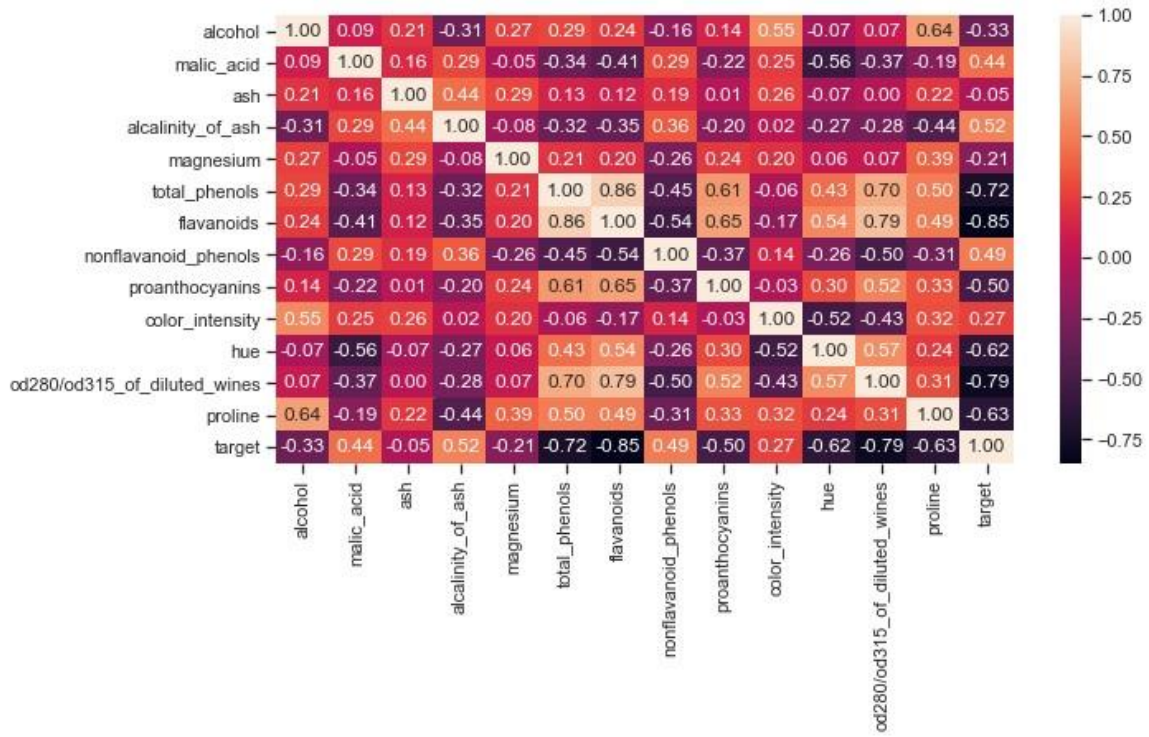
```
['alcohol',
 'malic_acid',
 'ash',
 'alcalinity_of_ash',
 'magnesium',
 'total_phenols',
 'flavanoids',
 'nonflavanoid_phenols',
 'proanthocyanins',
 'color_intensity',
 'hue',
 'od280/od315_of_diluted_wines',
 'proline',
 'target']
```

In [8]:

```
fig, ax = plt.subplots(figsize=(10,5))
sns.heatmap(train[corr_cols_1].corr(), annot=True, fmt='.2f')
```

Out[8]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x222263c67f0>
```



На основе корреляционной матрицы можно сделать следующие выводы:

- Целевой признак классификации "target" наиболее сильно коррелирует с щелочностью золы (0.52), нефлаваноидными фенолами (0.49) и яблочной кислотой (0.44). Эти признаки обязательно следует оставить в модели классификации.
- Целевой признак регрессии "flavanoids" наиболее сильно коррелирует с "total_phenols" (0.86) и OD280 / OD315 (разбавленностью вина) (0.79). Эти признаки обязательно следует оставить в модели регрессии.
- 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.

В качестве метрик для решения задачи классификации будем использовать:

Метрики, формируемые на основе матрицы ошибок:

1. Метрика precision: Можно переводить как точность, но такой перевод совпадает с переводом метрики "accuracy".

$$precision = \frac{TP}{TP+FP}$$

–

Доля верно предсказанных классификатором положительных объектов, из всех объектов, которые классификатор верно или неверно определил как положительные.

Используется функция precision_score.

В качестве метрик для решения задачи регрессии будем использовать:

1. Mean absolute error - средняя абсолютная ошибка

$$R^2(y, \hat{y}) = 1 - \frac{\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{N}(y_i - \overline{y_i})^2}$$

где:

- y - истинное значение целевого признака
- $\hat{y}$ - предсказанное значение целевого признака
- N - размер тестовой выборки
- $\overline{y_i} = \frac{1}{N} \cdot \sum_{i=1}^{N} y_i$

Вычисляется с помощью функции mean_absolute_error.

In [9]:

```
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()
```

Выбор наиболее подходящих моделей для решения задачи классификации или регрессии.

Для задачи классификации и регрессии будем использовать следующие модели:

- Случайный лес
- Градиентный бустинг

In [10]:

```
# Признаки для задачи классификации
task_clas_cols = ['alcalinity_of_ash', 'nonflavanoid_phenols',
                  'malic_acid', 'color_intensity']
```

In [11]:

```
# Разделение выборки на обучающую и тестовую
clas_X_train, clas_X_test, clas_Y_train, clas_Y_test = train_test_split(
    train[task_clas_cols], train['target'], test_size=0.5, random_state=1)
clas_X_train.shape, clas_X_test.shape, clas_Y_train.shape, clas_Y_test.shape
```

Out[11]: ((89, 4), (89, 4),

(89,), (89,))

In [12]:

```
# Признаки для задачи регресии
task_regr_cols = ['total_phenols', 'od280/od315_of_diluted_wines',
                  'proanthocyanins', 'proline']
```

In [13]:

```
# Разделение выборки на обучающую и тестовую
regr_X_train, regr_X_test, regr_Y_train, regr_Y_test = train_test_split(
    train[task_regr_cols], train['flavanoids'], test_size=0.5, random_state=1)
regr_X_train.shape, regr_X_test.shape, regr_Y_train.shape, regr_Y_test.shape
```

Out[13]:

((89, 4), (89, 4), (89,), (89,))

4. Обучение моделей

Решение задачи классификации

In [14]:

```
# Модели
clas_models = {'RF':RandomForestClassifier(),
               'GB':GradientBoostingClassifier()}
```

In [15]:

```
# Сохранение метрик
clasMetricLogger = MetricLogger()
```

In [16]:

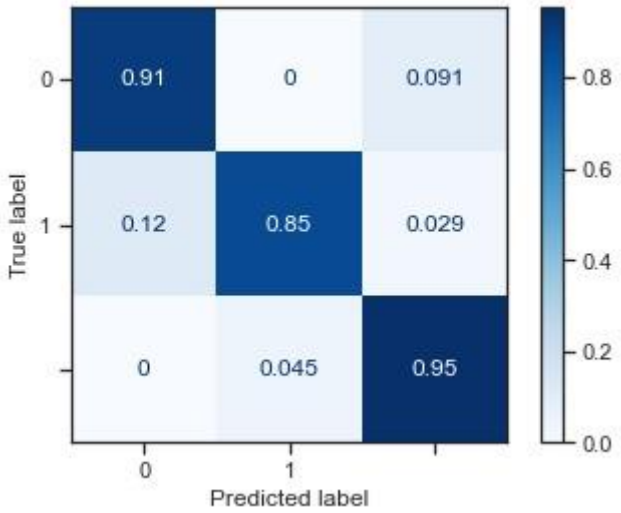
```
def clas_train_model(model_name, model, clasMetricLogger):
 model.fit(clas_X_train, clas_Y_train)    Y_pred =
 model.predict(clas_X_test)
    precision = precision_score(clas_Y_test.values, Y_pred, average = 'weighted')

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

    print('*******************************************************')
 print(model)
    print('*******************************************************')
```

```
   plot_confusion_matrix(model, clas_X_test, clas_Y_test.values,
display_labels=['0','1'],                        cmap=plt.cm.Blues,
normalize='true')    plt.show()
```

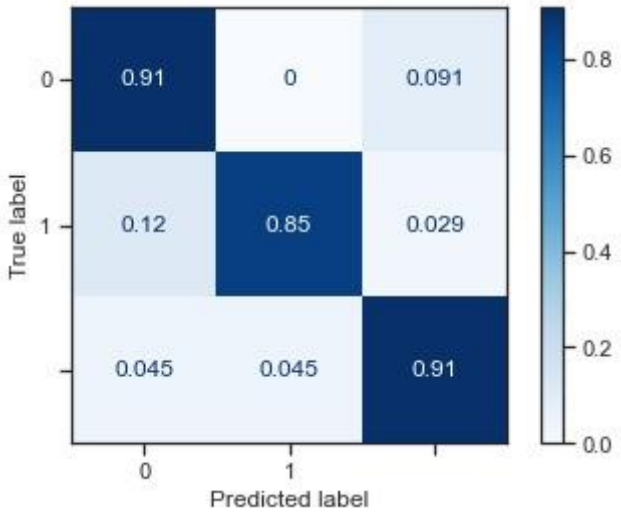
In [17]:

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

```
**************************************************
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
 criterion='gini', max_depth=None, max_features='aut
o',
                       max_leaf_nodes=None, max_samples=None,
 min_impurity_decrease=0.0, min_impurity_split=None,
 min_samples_leaf=1, min_samples_split=2,
 min_weight_fraction_leaf=0.0, n_estimators=100,
 n_jobs=None, oob_score=False, random_state=None,
 verbose=0, warm_start=False)
**************************************************
```




```
**************************************************
GradientBoostingClassifier(ccp_alpha=0.0, criterion='friedman_mse', init=N
one,                           learning_rate=0.1, loss='deviance',
max_depth=
3,                             max_features=None, max_leaf_nodes=None,
 min_impurity_decrease=0.0, min_impurity_split=N
one,
                           min_samples_leaf=1, min_samples_split=2,
 min_weight_fraction_leaf=0.0, n_estimators=100,
 n_iter_no_change=None, presort='deprecated',
 random_state=None, subsample=1.0, tol=0.0001,
 validation_fraction=0.1, verbose=0,
 warm_start=False)
**************************************************
```



Решение задачи регрессии

In [18]:

```
# Модели
regr_models = {'RF':RandomForestRegressor(),
               'GB':GradientBoostingRegressor()}
```

In [19]:

```
# Сохранение метрик
regrMetricLogger = MetricLogger()
```

In [20]:

```
def regr_train_model(model_name, model, regrMetricLogger):
    model.fit(regr_X_train, regr_Y_train)
    Y_pred = model.predict(regr_X_test)

    mae = mean_absolute_error(regr_Y_test, Y_pred)

    regrMetricLogger.add('MAE', model_name, mae)

    print('*******************************************************')
    print(model)
    print()
    print('MAE={}'.format(
        round(mae, 3)))
    print('*******************************************************')
```

In [21]:

```
for model_name, model in regr_models.items():
 regr_train_model(model_name, model, regrMetricLogger)
```

```
**************************************************
RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse',
max_depth=None, max_features='auto', max_leaf_nodes= None,
                      max_samples=None, min_impurity_decrease=0.0,
min_impurity_split=None, min_samples_leaf=1,
min_samples_split=2, min_weight_fraction_leaf=0.0,
n_estimators=100, n_jobs=None, oob_score=False,
random_state=None, verbose=0, warm_start=False)

MAE=0.262
**************************************************
**************************************************
GradientBoostingRegressor(alpha=0.9, ccp_alpha=0.0, criterion='friedman_ms
e',
                          init=None, learning_rate=0.1, loss='ls', max_dep
th=3,
                          max_features=None, max_leaf_nodes=None,
min_impurity_decrease=0.0, min_impurity_split=No ne,
                          min_samples_leaf=1, min_samples_split=2,
min_weight_fraction_leaf=0.0, n_estimators=100,
n_iter_no_change=None, presort='deprecated',
random_state=None, subsample=1.0, tol=0.0001,
validation_fraction=0.1, verbose=0, warm_start=F alse)
MAE=0.256
**************************************************
```

5. Подбор гиперпараметра K с использованием GridSearchCV

Для задачи классификации

In [22]:

```
clas_X_train.shape
```

Out[22]:

```
(89, 4)
```

In [23]:

```
n_range = np.array(range(1,70,10))
tuned_parameters = [{'n_neighbors': n_range}]
tuned_parameters
```

Out[23]:

```
[{'n_neighbors': array([ 1, 11, 21, 31, 41, 51, 61])}]
```

In [24]:

```
%%time
clf_gs = GridSearchCV(KNeighborsClassifier(), tuned_parameters, cv=5)
clf_gs.fit(clas_X_train, clas_Y_train)
```

```
Wall time: 277 ms
```

Out[24]:

```
GridSearchCV(cv=5, error_score=nan,
             estimator=KNeighborsClassifier(algorithm='auto', leaf_size=3
0,
                                            metric='minkowski',
metric_params=None, n_jobs=Non e,
                                            n_neighbors=5, p=2,
weights='uniform'),              iid='deprecated', n_jobs=None,
             param_grid=[{'n_neighbors': array([ 1, 11, 21, 31, 41, 51, 6
1])}],
             pre_dispatch='2*n_jobs', refit=True, return_train_score=Fals
e,               scoring=None, verbose=0) In [25]:
```

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

Out[25]:

```
KNeighborsClassifier(algorithm='auto',  leaf_size=30,  metric='minkowski',
metric_params=None,         n_jobs=None,         n_neighbors=1,         p=2,
weights='uniform') In [26]:
```

```
# Лучшее значение параметров
clf_gs.best_params_
```

Out[26]:

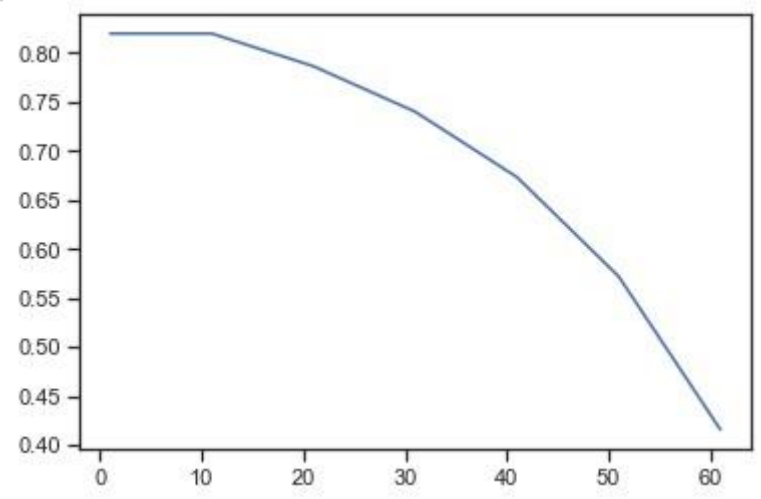
```
{'n_neighbors': 1}
```

In [27]:

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

Out[27]: [<matplotlib.lines.Line2D at

```
0x222289f77b8>]
```



Для задачи регрессии

In [28]:

```
n_range = np.array(range(1,70,10))
tuned_parameters = [{'n_neighbors': n_range}]
tuned_parameters
```

Out[28]:

```
[{'n_neighbors': array([ 1, 11, 21, 31, 41, 51, 61])}]
```

In [29]:

```
%%time
regr_gs = GridSearchCV(KNeighborsRegressor(), tuned_parameters, cv=5, scoring='neg_mean
_squared_error')
regr_gs.fit(regr_X_train, regr_Y_train)
```

```
Wall time: 204 ms
```

Out[29]:

```
GridSearchCV(cv=5, error_score=nan,
 estimator=KNeighborsRegressor(algorithm='auto', leaf_size=30,
 metric='minkowski',
 metric_params=None, n_jobs=Non
e,
                                             n_neighbors=5, p=2,
 weights='uniform'),              iid='deprecated', n_jobs=None,
             param_grid=[{'n_neighbors': array([ 1, 11, 21, 31, 41, 51, 6
```

```
1])}],              pre_dispatch='2*n_jobs', refit=True,
 return_train_score=Fals
e,              scoring='neg_mean_squared_error',
 verbose=0)
```

In [30]:

```
# Лучшая модель
regr_gs.best_estimator_
```

Out[30]:

```
KNeighborsRegressor(algorithm='auto',  leaf_size=30,  metric='minkowski',
metric_params=None,        n_jobs=None,        n_neighbors=11,        p=2,
weights='uniform') In [31]:
```

```
# Лучшее значение параметров
regr_gs.best_params_
```

Out[31]:

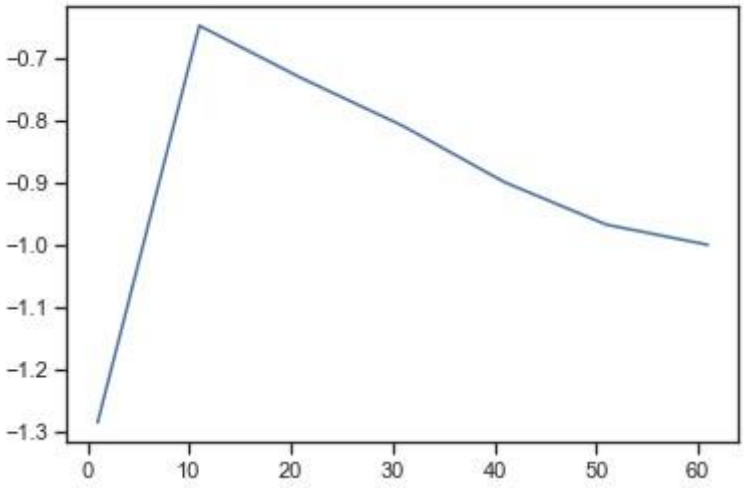
```
{'n_neighbors': 11}
```

In [32]:

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

Out[32]:

```
[<matplotlib.lines.Line2D at 0x22228a69e80>]
```



6. Повторение пункта 4 для найденных оптимальных значений гиперпараметров.

Решение задачи классификации

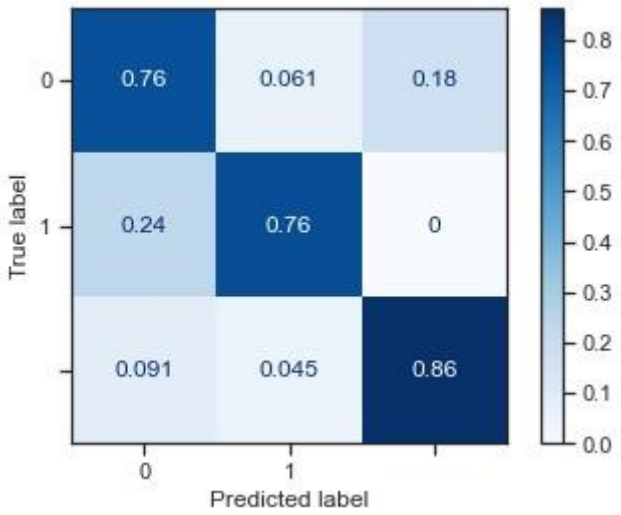
In [33]:

```
clas_models_grid = {'KNN_1':clf_gs.best_estimator_}
```

In [34]:

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

```
**************************************************
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
metric_params=None, n_jobs=None, n_neighbors=1, p=2,
weights='uniform')
**************************************************
```



Решение задачи регрессии

In [35]:

```
regr_models_grid = {'KNN_11':regr_gs.best_estimator_}
```

In [36]:

```
for model_name, model in regr_models_grid.items():
    regr_train_model(model_name, model, regrMetricLogger)
```

```
**************************************************
```

```
KNeighborsRegressor(algorithm='auto', leaf_size=30, metric='minkowski',
metric_params=None, n_jobs=None, n_neighbors=11, p=2,
weights='uniform')

MAE=0.634
**************************************************
```

Сравнение качества полученных моделей

Решение задачи классификации

In [37]:

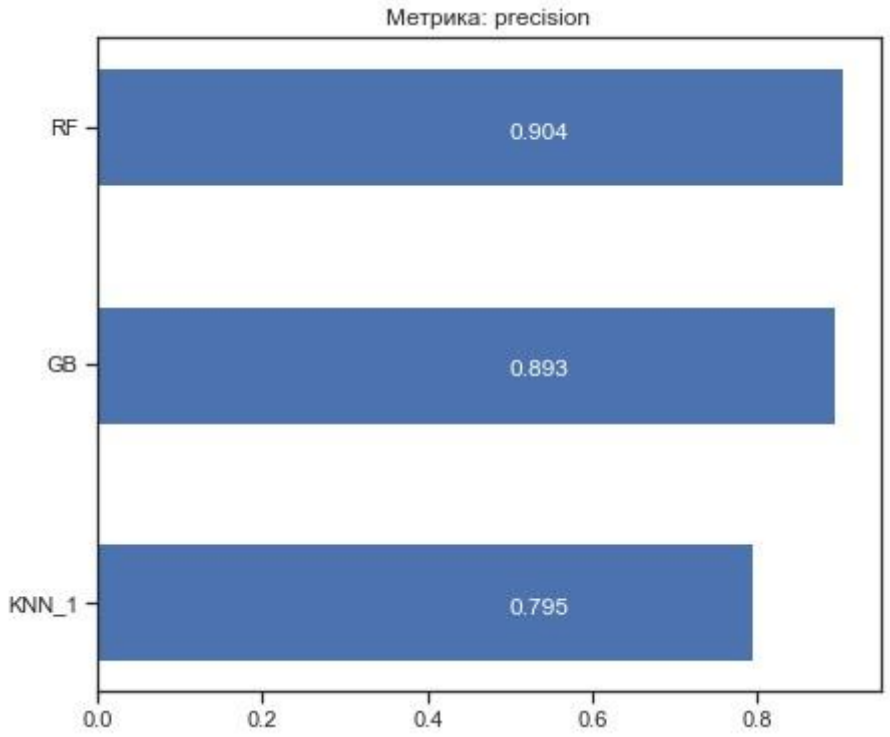
```
# Метрики качества модели
clas_metrics = clasMetricLogger.df['metric'].unique()
clas_metrics
```

Out[37]:

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

In [38]:

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



Вывод: на основании метрики Precision, лучшей оказалась модель "случайный лес".

Решение задачи классификации

In [39]:

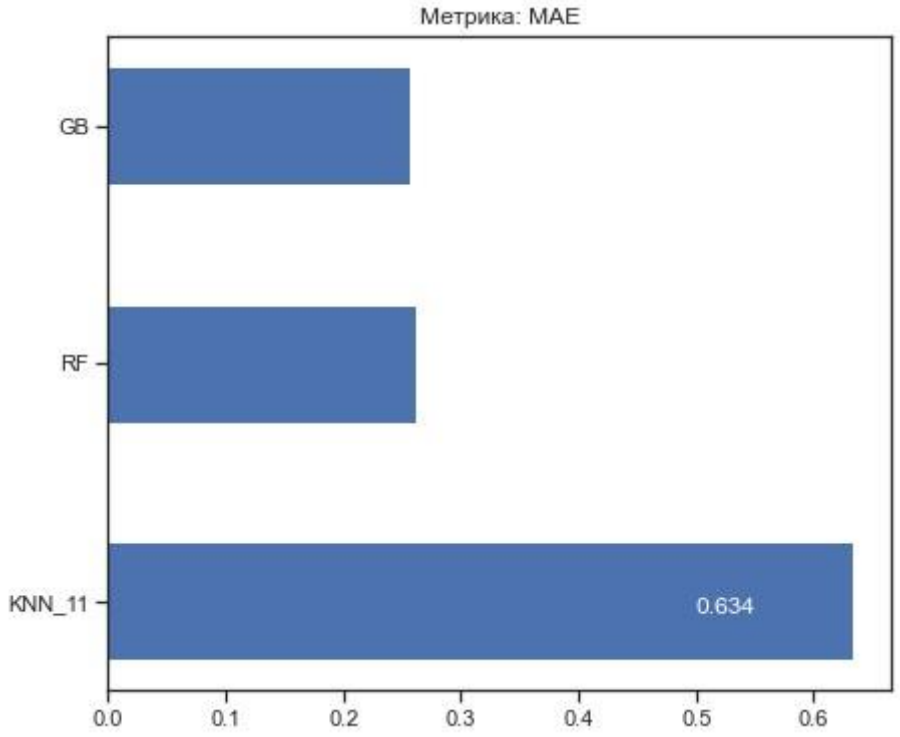
```
# Метрики качества модели
regr_metrics = regrMetricLogger.df['metric'].unique()
regr_metrics
```

Out[39]:

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

In [40]:

```
regrMetricLogger.plot('Метрика: ' + 'MAE', 'MAE', ascending=False, figsize=(7, 6))
```



Вывод: на основании метрики MAE, лучшей оказалась модель градиентного бустинга.