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 КУРСОВОМУ ПРОЕКТУ

НА ТЕМУ:

«Решение задачи машинного обучения»

Студент ИУ5-65Б _____ **А.В. Герасименко**
(Группа) (Подпись, дата) (И.О.Фамилия)

Руководитель курсового проекта _____ **Ю.Е. Гапанюк**
(Подпись, дата) (И.О.Фамилия)

Консультант _____ **Ю.Е. Гапанюк**
(Подпись, дата) (И.О.Фамилия)

2021 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 ____ИУ5___
(Индекс)
_____ В.М. Черненький_
(И.О.Фамилия)
« _____ » _____ 20 ____ г.

З А Д А Н И Е
на выполнение курсового проекта

по дисциплине «Технологии машинного обучения»

Студент группы ИУ5-65Б

Герасименко Анастасия Витальевна
(Фамилия, имя, отчество)

Тема курсового проекта «Решение задачи машинного обучения»

Направленность КП (учебный, исследовательский, практический, производственный, др.)
учебный

Источник тематики (кафедра, предприятие, НИР) кафедра

График выполнения проекта: 25% к 3 нед., 50% к 9 нед., 75% к 12 нед., 100% к 16 нед.

Задание Решение задачи машинного обучения. Результатом курсового проекта является отчет, содержащий описания моделей, тексты программ и результаты экспериментов.

Оформление курсового проекта:

Расчетно-пояснительная записка на 43 листах формата А4.
Перечень графического (иллюстративного) материала (чертежи, плакаты, слайды и т.п.)

Дата выдачи задания « 7 » февраля 2021 г.

Руководитель курсового проекта _____ **Ю.Е. Гапанюк**
(Подпись, дата) (И.О.Фамилия)

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

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

Содержание

Введение

Курсовой проект – самостоятельная часть учебной дисциплины «Технологии машинного обучения» – учебная и практическая исследовательская студенческая работа, направленная на решение комплексной задачи машинного обучения. Результатом курсового проекта является отчет, содержащий описания моделей, тексты программ и результаты экспериментов.

Курсовой проект опирается на знания, умения и владения, полученные студентом в рамках лекций и лабораторных работ по дисциплине.

В рамках курсового проекта возможно проведение типового или нетипового исследования.

- Типовое исследование – решение задачи машинного обучения на основе материалов дисциплины. Выполняется студентом единолично.
- Нетиповое исследование – решение нестандартной задачи. Тема должна быть согласована с преподавателем. Как правило, такая работа выполняется группой студентов.

Основная часть

Схема типового исследования

Схема типового исследования, проводимого студентом в рамках курсовой работы, содержит выполнение следующих шагов:

1. Поиск и выбор набора данных для построения моделей машинного обучения. На основе выбранного набора данных студент должен построить модели машинного обучения для решения или задачи классификации, или задачи регрессии.
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. Кодирование категориальных признаков. Масштабирование данных. Формирование вспомогательных признаков, улучшающих качество моделей.
4. Проведение корреляционного анализа данных. Формирование промежуточных выводов о возможности построения моделей машинного обучения. В зависимости от набора данных, порядок выполнения пунктов 2, 3, 4 может быть изменен.
5. Выбор метрик для последующей оценки качества моделей. Необходимо выбрать не менее трех метрик и обосновать выбор.
6. Выбор наиболее подходящих моделей для решения задачи классификации или регрессии. Необходимо использовать не менее пяти моделей, две из которых должны быть ансамблевыми.
7. Формирование обучающей и тестовой выборок на основе исходного набора данных.
8. 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.
9. 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.
10. Повторение пункта 8 для найденных оптимальных значений гиперпараметров. Сравнение качества полученных моделей с качеством baseline-моделей.
11. 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. Рекомендуется построение графиков обучения и валидации, влияния значений гиперпарметров на качество моделей и т.д.

Приведенная схема исследования является рекомендуемой. В зависимости от решаемой задачи возможны модификации.

Оценка за курсовой проект

При проведении типового исследования оценка за курсовой проект выставляется на основе следующих критериев:

- Оценка "удовлетворительно" - выполнение пунктов типового исследования с упрощенными требованиями (достаточно обучить две модели, из них только для одной осуществить подбор гиперпараметров, использовать для проверки качества одну

метрику)
- Оценка "хорошо" - выполнение всех пунктов типового исследования с учетом заданных требований (количество моделей, количество метрик и т.д.)
- Оценка "отлично":
  - Реализация всех требований для оценки "хорошо".
  - Разработка макета веб-приложения, предназначенного для анализа данных.
    - Вариант 1. Макет должен быть реализован для одной модели машинного обучения. Макет должен позволять:
      - Задавать гиперпараметры алгоритма,
      - Производить обучение,
      - Осуществлять просмотр результатов обучения, в том числе в виде графиков.
    - Вариант 2. Макет должен быть реализован для нескольких моделей машинного обучения. Макет должен позволять:
      - Выбирать модели для обучения,
      - Производить обучение,
      - Осуществлять просмотр результатов обучения, в том числе в виде графиков.
    - Для разработки рекомендуется использовать следующие (или аналогичные) фреймворки:
      - Streamlit
      - Gradio
      - Dash
- "+1 балл" за курсовой проект - за применение к выбранному набору данных произвольной библиотеки AutoML и сравнение качества моделей, полученных вручную и с использованием AutoML.
- "+1 балл" за курсовой проект - за дополнительное использование в курсовом проекте методов кластеризации, понижения размерности, методов рекомендательных систем.
- В случае получения максимального балла за курсовую работу без "+1 балла", "+1 балл" переносится на экзамен по дисциплине.

Последовательность действий

1) Поиск и выбор набора данных для построения моделей машинного обучения.

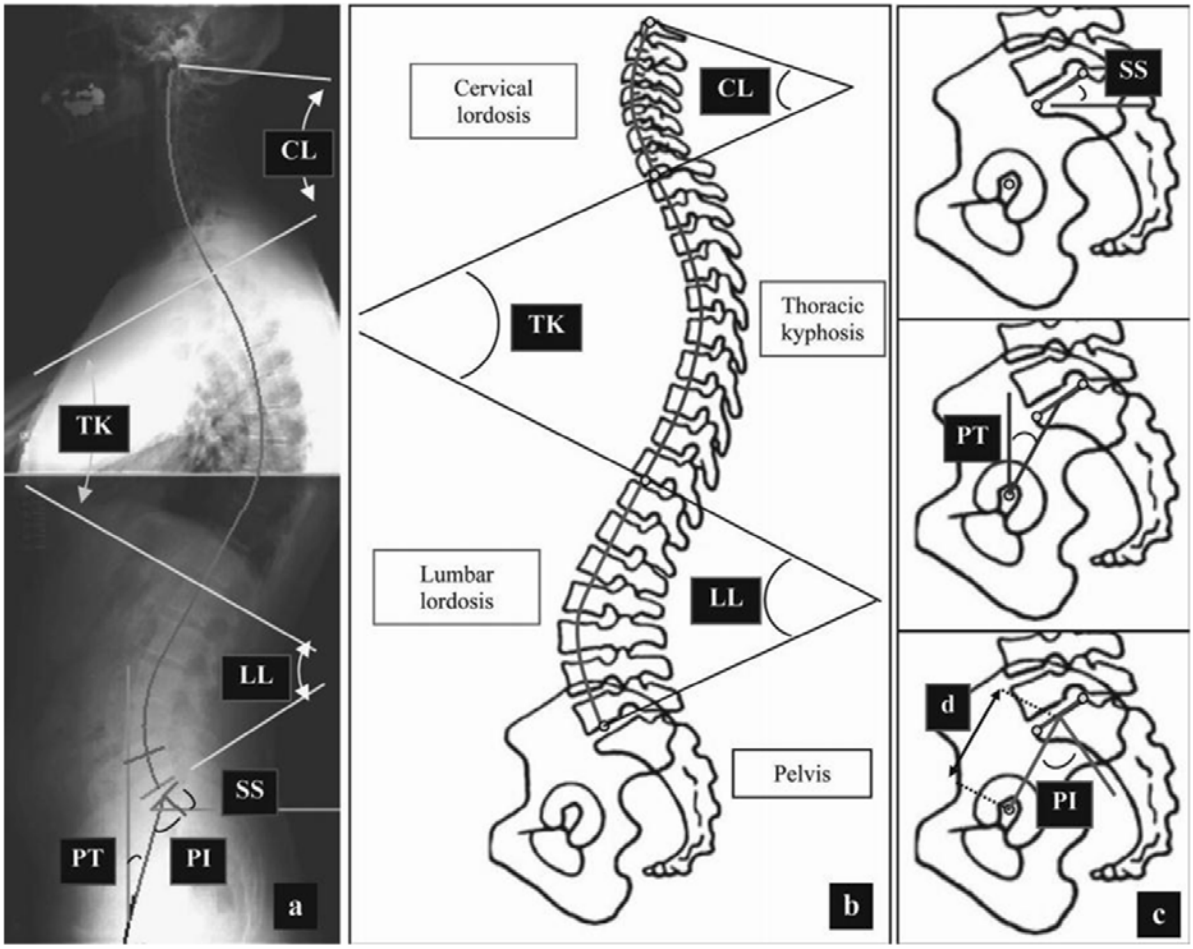
В работе используется набор данных, состоящий из 13 колонок и 310 наблюдений. 12 столбцов – это числовые атрибуты позвоночника/спины. Последняя колонка – это состояние пациента: аномальное указывает на наличие боли в спине, нормальное - на отсутствие боли в спине. Этот набор данных предназначен для определения состояния здоровья человека, с помощью собранных физических данных о позвоночнике. Набор данных содержит следующие колонки:

1. Угол падения таза (Pelvic_incidence)
2. Наклон таза (Pelvic_tilt)
3. Угол поясничного лордоза (Lumbar_lordosis_angle)
4. Наклон крестца (Sacral_slope)
5. Тазовый радиус (Pelvic_radius)
6. Степень спондилолистеза (Degree_spondylolisthesis)

7. Наклон таза (Pelvic_slope)
8. Прямой наклон (Direct_tilt)
9. Наклон грудной клетки (Thoracic_slope)
10. Наклон шейки матки (Cervical_tilt)
11. Угол крестца (Sacrum_angle)
12. Наклон сколиоза (Scoliosis_slope)
13. Класс (Class_att)

In [1]:
```
from IPython.display import Image
Image("example.png")
```

Out[1]:



- PI - Pelvic_incidence
- PT - Pelvic_tilt
- LL - Lumbar_lordosis_angle

In [2]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import streamlit as st
import matplotlib.pyplot as plt
from catboost import Pool, CatBoostClassifier
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, clas
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import train_test_split, learning_curve
```

```
from sklearn.metrics import plot_confusion_matrix
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import mean_absolute_error, mean_squared_error, mean
from sklearn.metrics import roc_curve, roc_auc_score
from sklearn.svm import SVC, NuSVC, LinearSVC, OneClassSVM, SVR, NuSVR, L
from sklearn.tree import DecisionTreeClassifier, DecisionTreeRegressor, e
from sklearn.ensemble import RandomForestClassifier, RandomForestRegresso
from sklearn.ensemble import ExtraTreesClassifier, ExtraTreesRegressor
from sklearn.ensemble import GradientBoostingClassifier, GradientBoosting
from sklearn.preprocessing import LabelEncoder
import warnings
warnings.filterwarnings('ignore')
%matplotlib inline
sns.set(style="ticks")
```

In [3]:
```
col_list = ['Pelvic_incidence',
                'Pelvic_tilt',
                'Lumbar_lordosis_angle',
                'Sacral_slope',
                'Pelvic_radius',
                'Degree_spondylolisthesis',
                'Pelvic_slope',
                'Direct_tilt',
                'Thoracic_slope',
                'Cervical_tilt',
                'Sacrum_angle',
                'Scoliosis_slope',
                'Class_att',
                'To_drop']
data = pd.read_csv('Dataset_spine.csv', names=col_list, header=1, sep=","
data.drop('To_drop', axis=1, inplace=True)
```

2) Проведение разведочного анализа данных. Построение графиков, необходимых для понимания структуры данных.

In [4]:
```
data.head()
```

Out[4]:

	Pelvic_incidence	Pelvic_tilt	Lumbar_lordosis_angle	Sacral_slope	Pelvic_radius	Degree_spon
0	39.056951	10.060991	25.015378	28.995960	114.405425	
1	68.832021	22.218482	50.092194	46.613539	105.985135	
2	69.297008	24.652878	44.311238	44.644130	101.868495	
3	49.712859	9.652075	28.317406	40.060784	108.168725	
4	40.250200	13.921907	25.124950	26.328293	130.327871	

In [5]:
```
data.shape
```

Out[5]: (309, 13)

In [6]:
```
data.columns
```

Out[6]:

```
Index(['Pelvic_incidence', 'Pelvic_tilt', 'Lumbar_lordosis_angle',
```

In [7]:
```
data.dtypes
```

Out[7]:
```
Pelvic_incidence            float64
Pelvic_tilt                 float64
Lumbar_lordosis_angle       float64
Sacral_slope                float64
Pelvic_radius               float64
Degree_spondylolisthesis    float64
Pelvic_slope                float64
Direct_tilt                 float64
Thoracic_slope              float64
Cervical_tilt               float64
Sacrum_angle                float64
Scoliosis_slope             float64
Class_att                    object
dtype: object
```

In [8]:
```
data.isnull().sum()
```

Out[8]:
```
Pelvic_incidence            0
Pelvic_tilt                 0
Lumbar_lordosis_angle       0
Sacral_slope                0
Pelvic_radius               0
Degree_spondylolisthesis    0
Pelvic_slope                0
Direct_tilt                 0
Thoracic_slope              0
Cervical_tilt               0
Sacrum_angle                0
Scoliosis_slope             0
Class_att                   0
dtype: int64
```

In [9]:
```
data['Class_att_le'] = data['Class_att'].map({'Abnormal': 1, 'Normal': 0})
```

In [10]:
```
print(data_log[['Class_att', 'Class_att_le']])
```

```
    Class_att  Class_att_le
0    Abnormal             1
1    Abnormal             1
2    Abnormal             1
3    Abnormal             1
4    Abnormal             1
..        ...           ...
304    Normal             0
305    Normal             0
306    Normal             0
307    Normal             0
308    Normal             0

[309 rows x 2 columns]
```

In [11]:
```
data['Class_att'].value_counts()
```

Out[11]:
```
Abnormal    209
Normal      100
Name: Class_att, dtype: int64
```

In [12]:
```
data.head()
```

Out[12]:

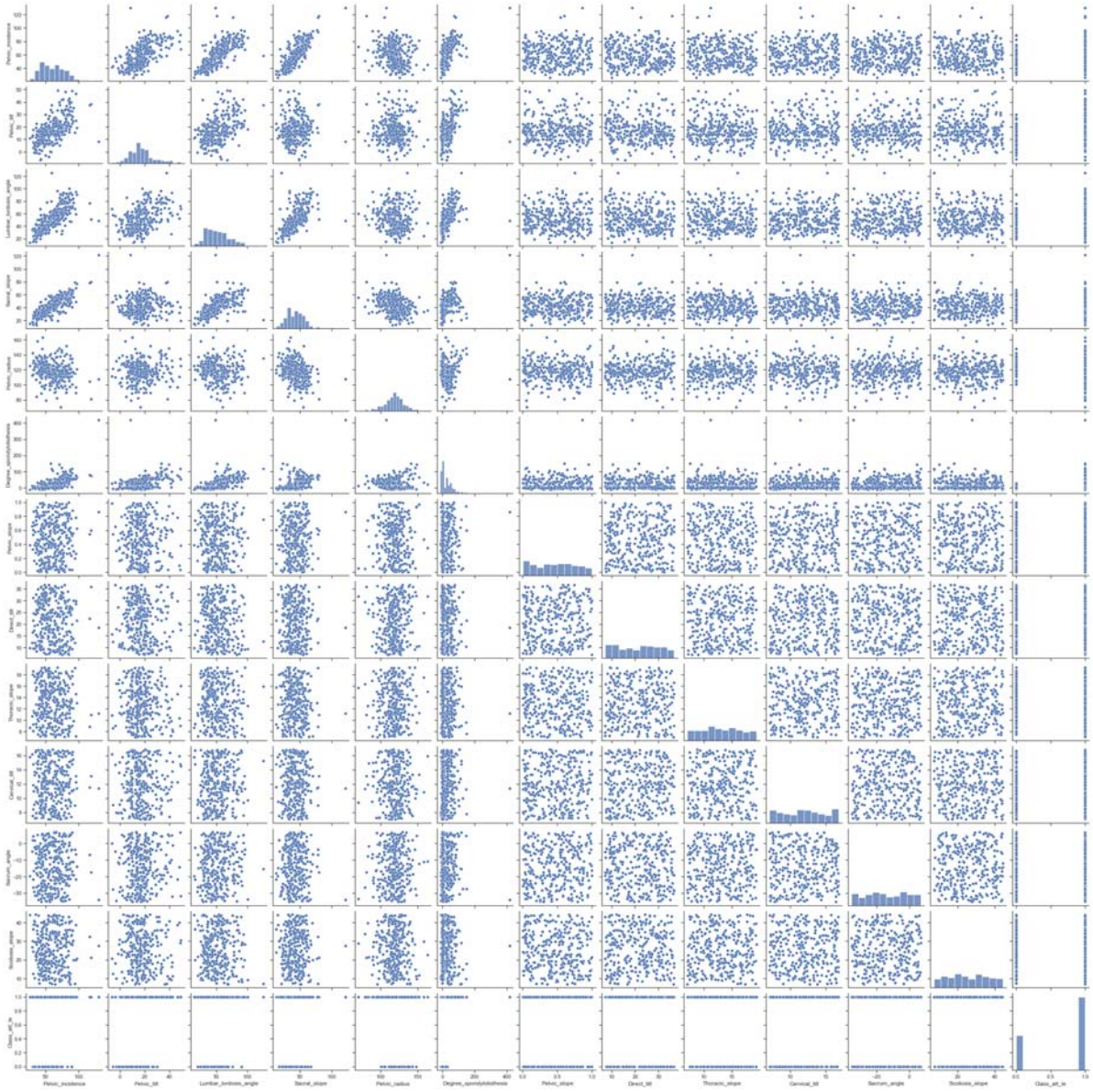
	Pelvic_incidence	Pelvic_tilt	Lumbar_lordosis_angle	Sacral_slope	Pelvic_radius	Degree_spon
0	39.056951	10.060991	25.015378	28.995960	114.405425	
1	68.832021	22.218482	50.092194	46.613539	105.985135	
2	69.297008	24.652878	44.311238	44.644130	101.868495	
3	49.712859	9.652075	28.317406	40.060784	108.168725	
4	40.250200	13.921907	25.124950	26.328293	130.327871	

Набор данных не содержит пропусков, категориальные пизнаки закодированы.

In [13]:
```
sns.pairplot(data)
```

```
2021-06-02 11:12:07.425 INFO    numexpr.utils: NumExpr defaulting to 4
threads.
```

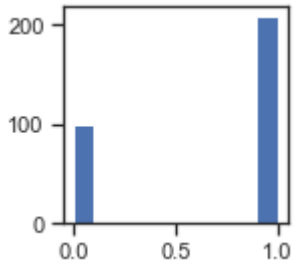
Out[13]:
```
<seaborn.axisgrid.PairGrid at 0x1cf415577c0>
```



In [14]:
```
sns.pairplot(data, hue="Class_att_le")
```

Out[14]: <seaborn.axisgrid.PairGrid at 0x1cf48198550>

In [15]:
```
# Оценим дисбаланс классов для Class_att_le
fig, ax = plt.subplots(figsize=(2,2))
plt.hist(data['Class_att_le'])
plt.show()
```

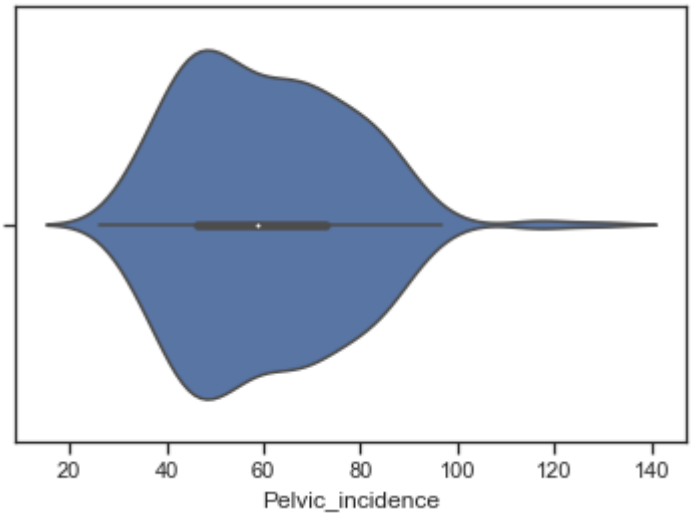


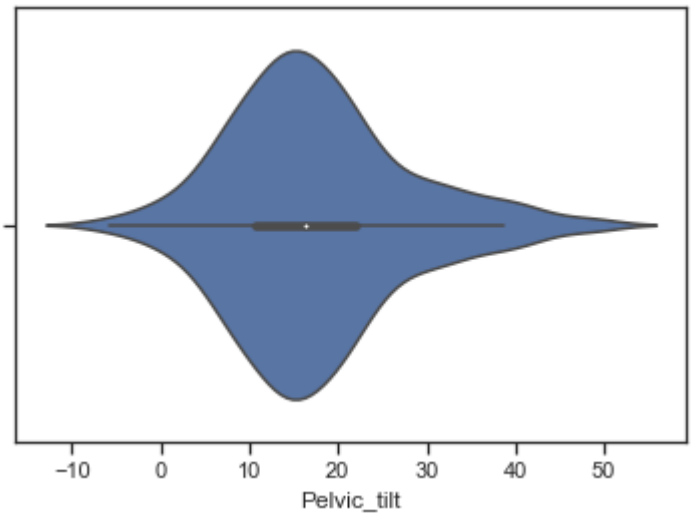
In [16]:
```
data['Class_att_le'].value_counts()
```

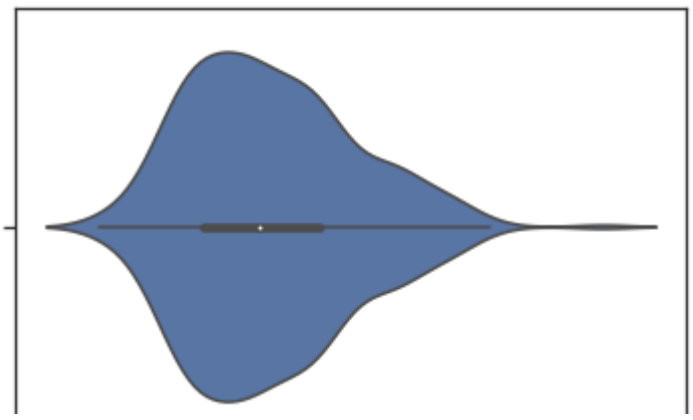
Out[16]:
```
1    209
0    100
Name: Class_att_le, dtype: int64
```

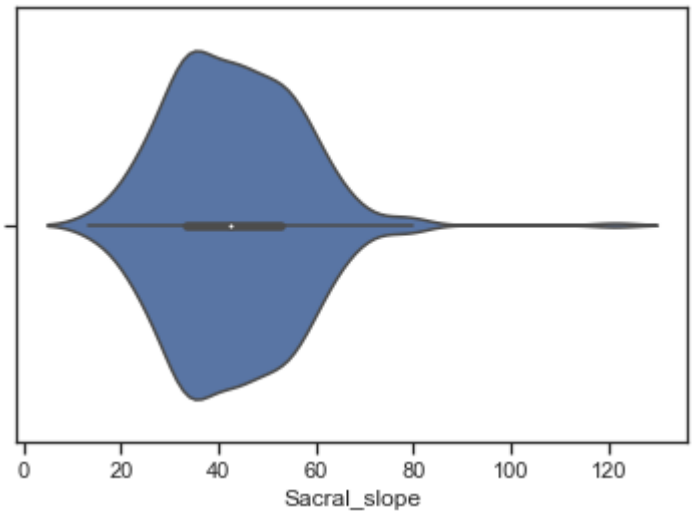
In [17]:
```
# посчитаем дисбаланс классов
```

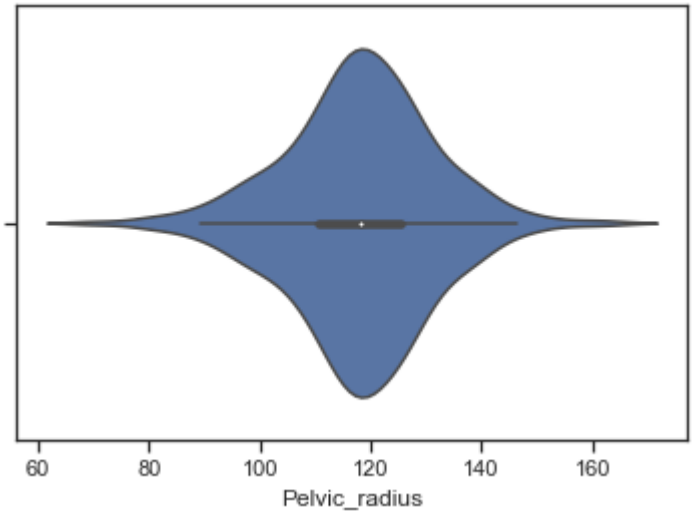
In [18]:
```
# Скрипичные диаграммы для числовых колонок
for col in ['Pelvic_incidence',
                'Pelvic_tilt',
                'Lumbar_lordosis_angle',
                'Sacral_slope',
                'Pelvic_radius',
                'Degree_spondylolisthesis',
                'Pelvic_slope',
                'Direct_tilt',
                'Thoracic_slope',
                'Cervical_tilt',
                'Sacrum_angle',
                'Scoliosis_slope']:
    sns.violinplot(x=data[col])
    plt.show()
```

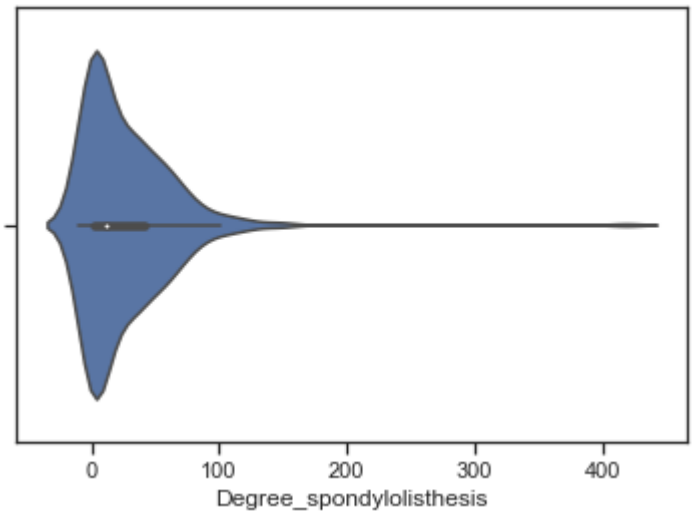


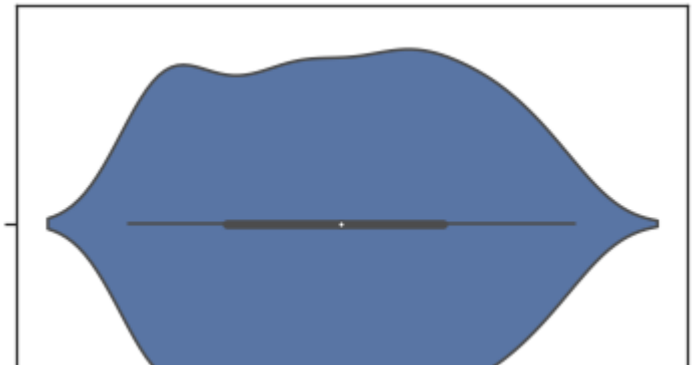


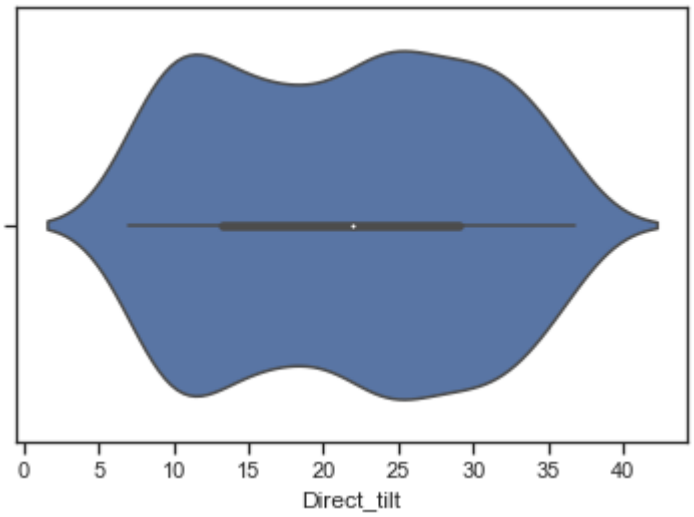


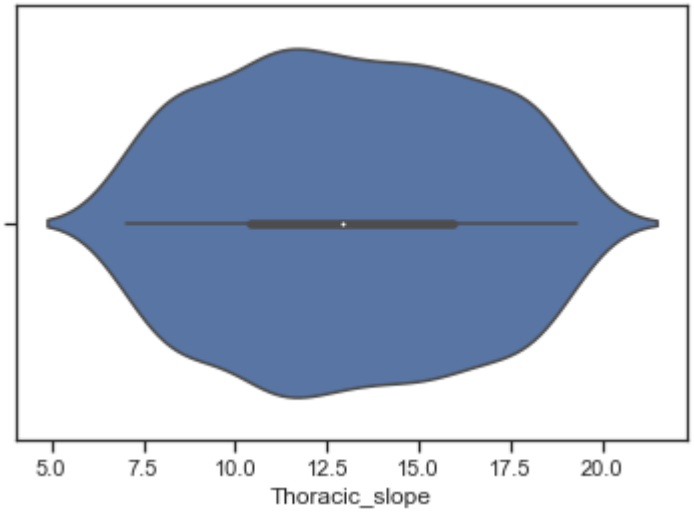

0
20
40
60
80
100
120
Sacral_slope

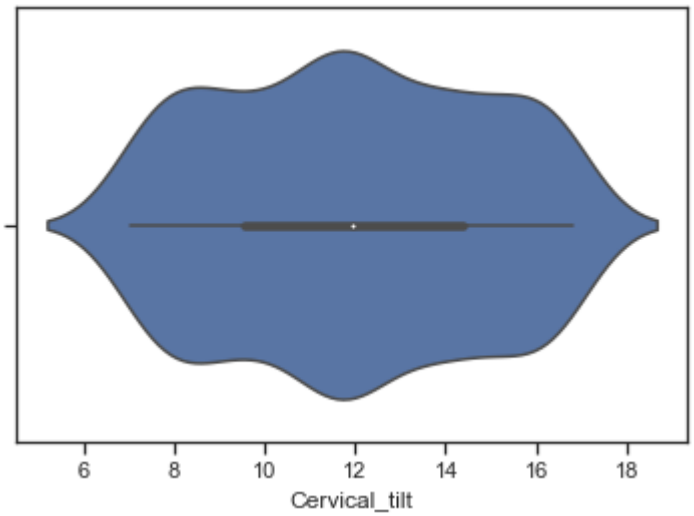

60
80
100
120
140
160
Pelvic_radius

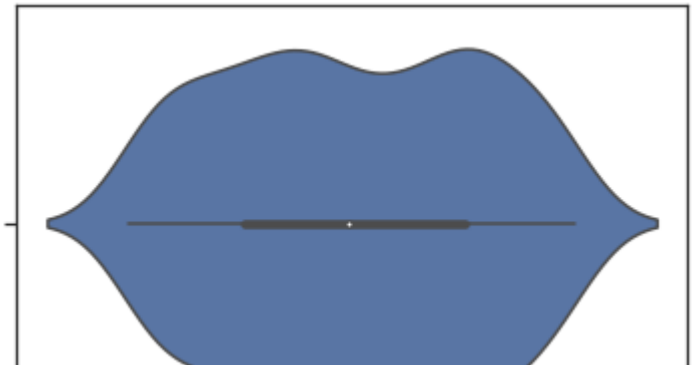

0
100
200
300
400
Degree_spondylolisthesis

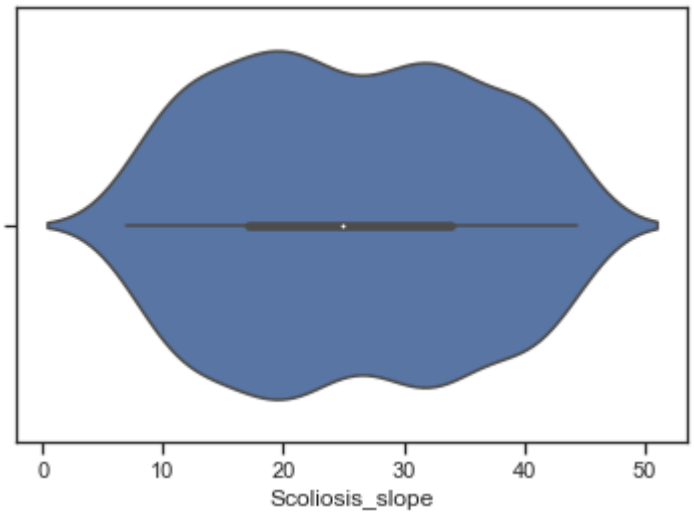



0
5
10
15
20
25
30
35
40
Direct_tilt


5.0
7.5
10.0
12.5
15.0
17.5
20.0
Thoracic_slope


6
8
10
12
14
16
18
Cervical_tilt





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

Для построения моделей будем использовать все признаки. Категориальные признаки закодированы.Выполним масштабирование данных.

In [19]:
```
# Числовые колонки для масштабирования
scale_cols = ['Pelvic_incidence',
              'Pelvic_tilt',
              'Lumbar_lordosis_angle',
              'Sacral_slope',
              'Pelvic_radius',
              'Degree_spondylolisthesis',
              'Pelvic_slope',
              'Direct_tilt',
              'Thoracic_slope',
              'Cervical_tilt',
              'Sacrum_angle',
              'Scoliosis_slope']
```

In [20]:
```
sc1 = MinMaxScaler()
sc1_data = sc1.fit_transform(data[scale_cols])
```

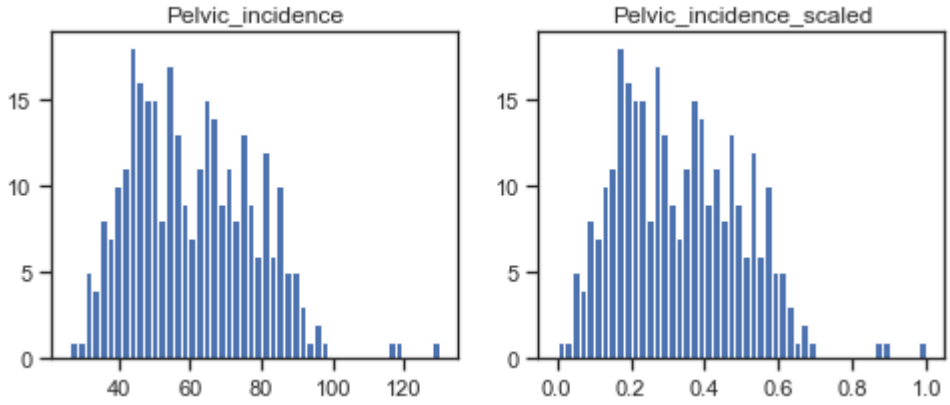
In [21]:
```
# Добавим масштабированные данные в набор данных
for i in range(len(scale_cols)):
    col = scale_cols[i]
    new_col_name = col + '_scaled'
    data[new_col_name] = sc1_data[:,i]
```

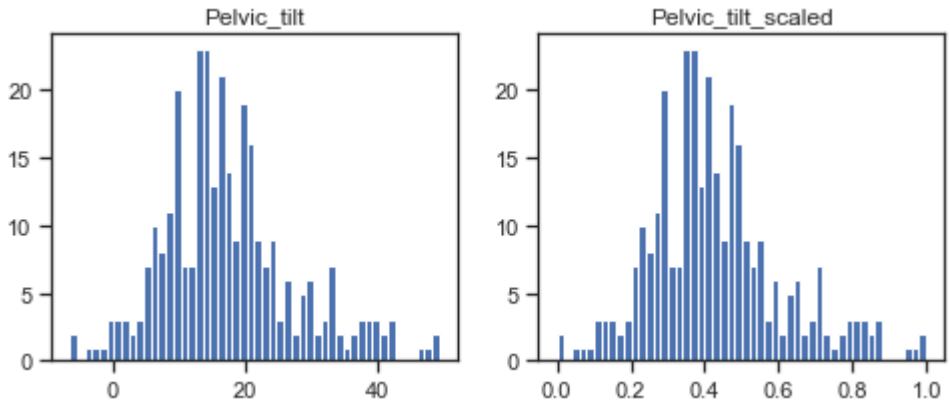
In [22]:
```
data.head()
```

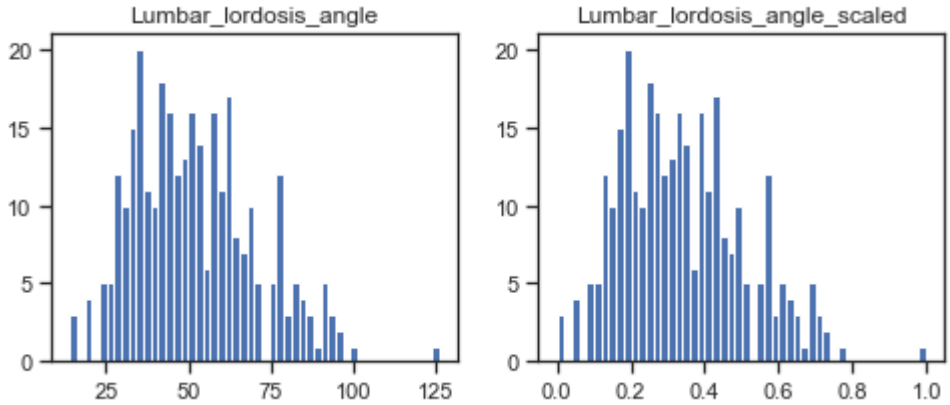
Out[22]:

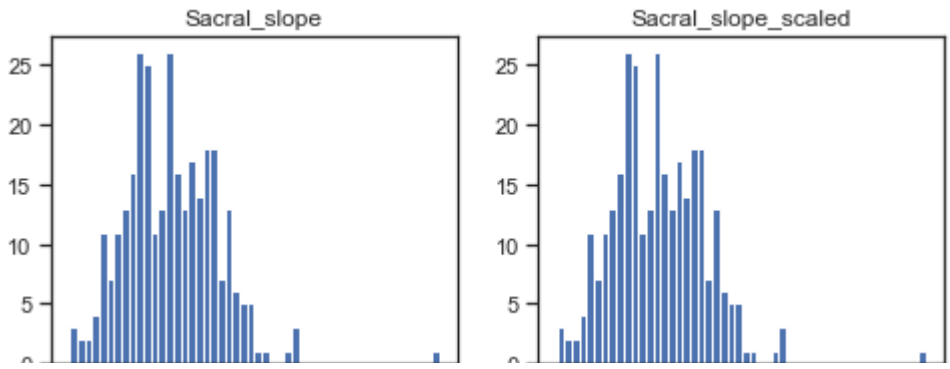
	Pelvic_incidence	Pelvic_tilt	Lumbar_lordosis_angle	Sacral_slope	Pelvic_radius	Degree_spon
0	39.056951	10.060991	25.015378	28.995960	114.405425	
1	68.832021	22.218482	50.092194	46.613539	105.985135	
2	69.297008	24.652878	44.311238	44.644130	101.868495	
3	49.712859	9.652075	28.317406	40.060784	108.168725	

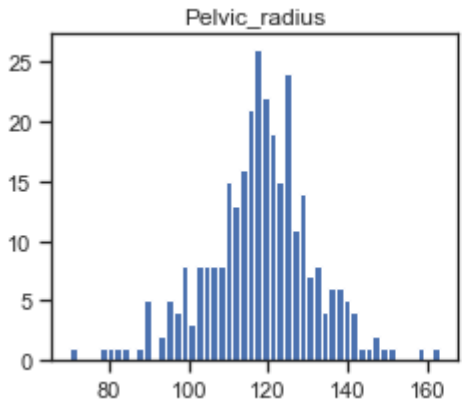
In [23]:
```
# Проверим, что масштабирование не повлияло на распределение данных
for col in scale_cols:
    col_scaled = col + '_scaled'

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

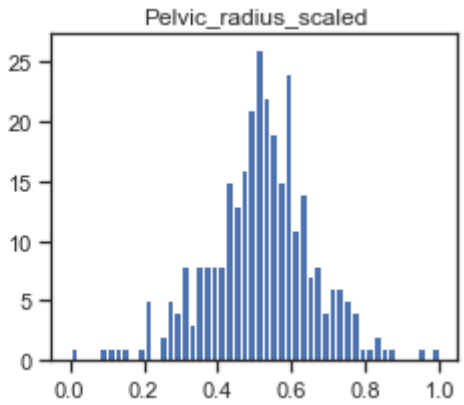


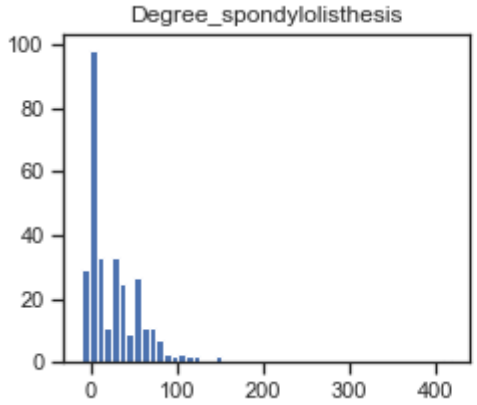


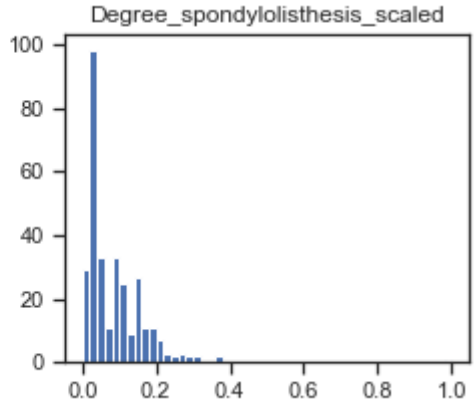


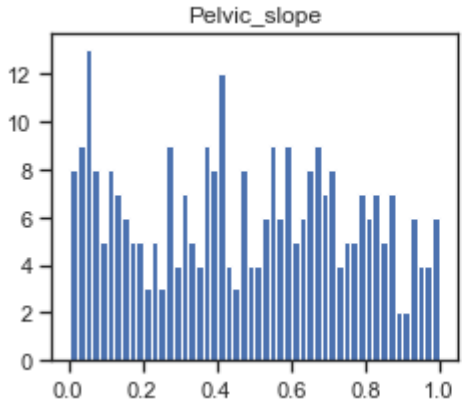


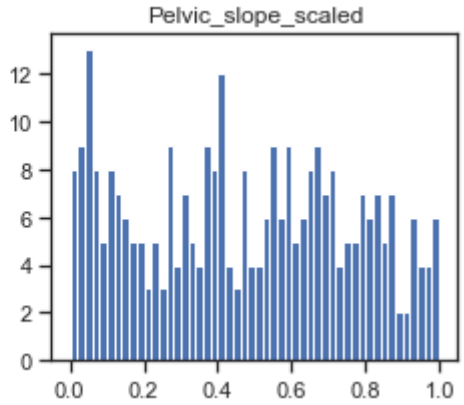

Pelvic_radius
25
20
15
10
5
0
80
100
120
140
160

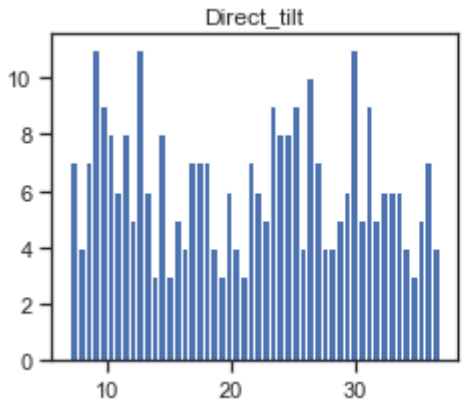

Pelvic_radius_scaled
25
20
15
10
5
0
0.0
0.2
0.4
0.6
0.8
1.0

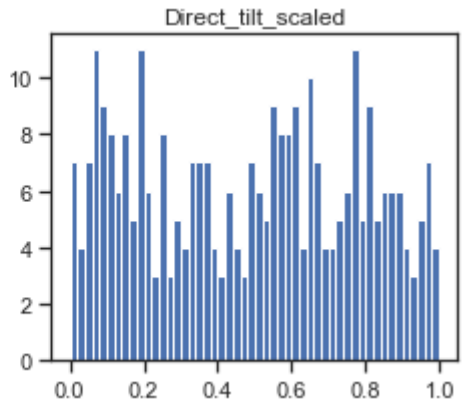

Degree_spondylolisthesis
100
80
60
40
20
0
0
100
200
300
400

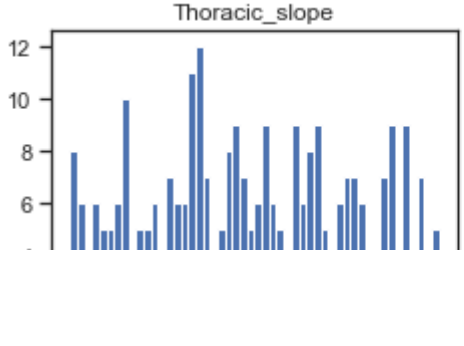

Degree_spondylolisthesis_scaled
100
80
60
40
20
0
0.0
0.2
0.4
0.6
0.8
1.0

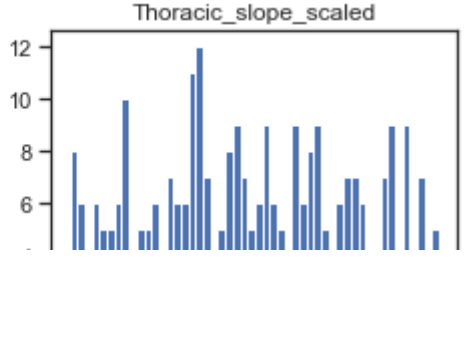

Pelvic_slope
12
10
8
6
4
2
0
0.0
0.2
0.4
0.6
0.8
1.0

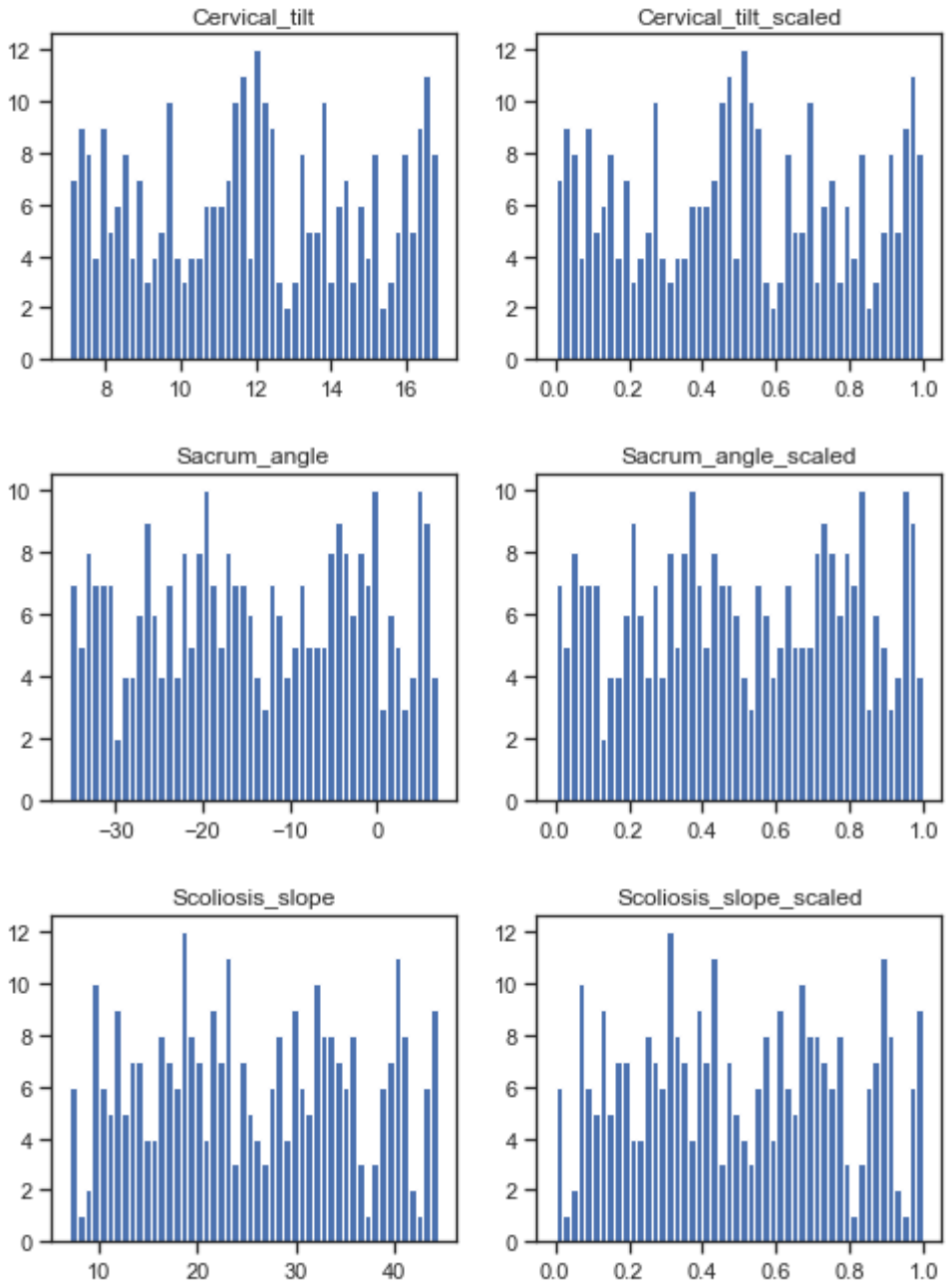

Pelvic_slope_scaled
12
10
8
6
4
2
0
0.0
0.2
0.4
0.6
0.8
1.0


Direct_tilt
10
8
6
4
2
0
10
20
30


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


Thoracic_slope
12
10
8
6


Thoracic_slope_scaled
12
10
8
6



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

In [24]:
```
corr_cols_1 = scale_cols + ['Class_att_le']
```

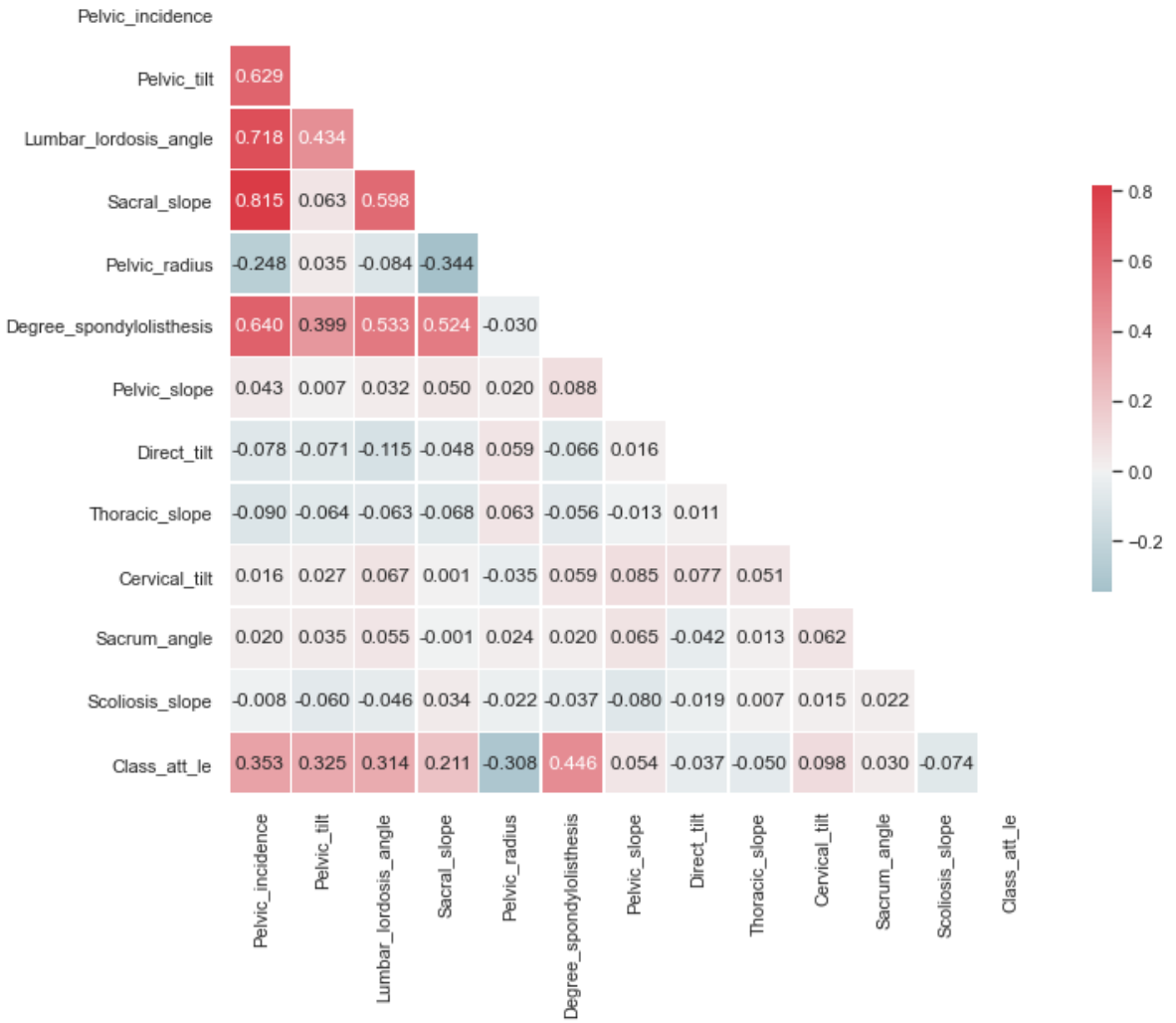
Out[24]:

```
['Pelvic incidence',
```

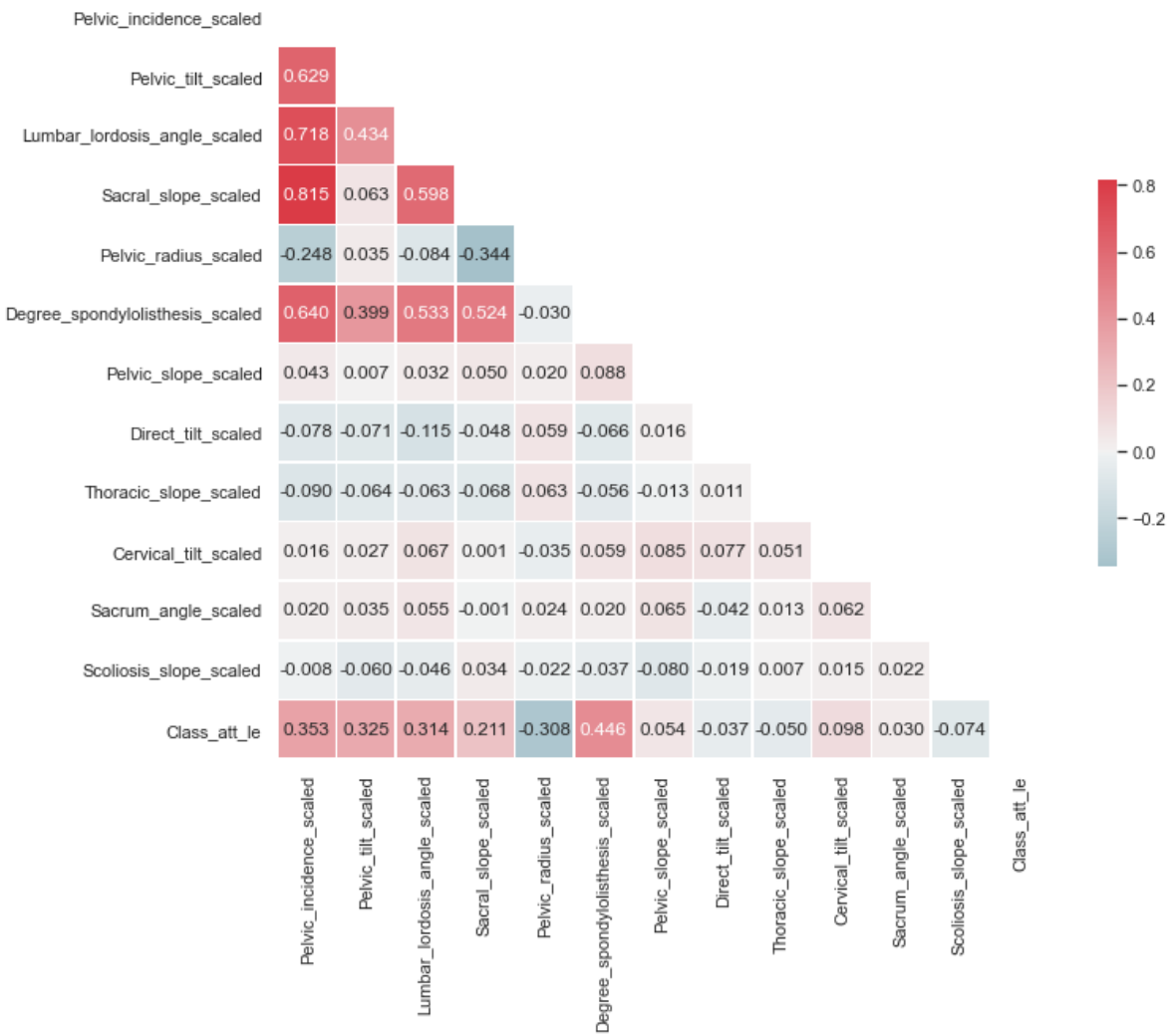
In [25]:
```
scale_cols_postfix = [x+'_scaled' for x in scale_cols]
corr_cols_2 = scale_cols_postfix + ['Class_att_le']
corr_cols_2
```

Out[25]:
```
['Pelvic_incidence_scaled',
 'Pelvic_tilt_scaled',
 'Lumbar_lordosis_angle_scaled',
 'Sacral_slope_scaled',
 'Pelvic_radius_scaled',
 'Degree_spondylolisthesis_scaled',
 'Pelvic_slope_scaled',
 'Direct_tilt_scaled',
 'Thoracic_slope_scaled',
 'Cervical_tilt_scaled',
 'Sacrum_angle_scaled',
 'Scoliosis_slope_scaled',
 'Class_att_le']
```

In [26]:
```
sns.set(style="white")
corr = data[corr_cols_1].corr()
mask = np.zeros_like(corr, dtype=np.bool)
mask[np.triu_indices_from(mask)] = True
f, ax = plt.subplots(figsize=(11, 9))
cmap = sns.diverging_palette(220, 10, as_cmap=True)
g=sns.heatmap(corr, mask=mask, cmap=cmap, center=0, annot=True, fmt='.3f'
              square=True, linewidths=.5, cbar_kws={"shrink": .5})
```



In [27]:
```
sns.set(style="white")
corr = data[corr_cols_2].corr()
mask = np.zeros_like(corr, dtype=np.bool)
mask[np.triu_indices_from(mask)] = True
f, ax = plt.subplots(figsize=(11, 9))
cmap = sns.diverging_palette(220, 10, as_cmap=True)
g=sns.heatmap(corr, mask=mask, cmap=cmap, center=0, annot=True, fmt='.3f'
              square=True, linewidths=.5, cbar_kws={"shrink": .5})
```



На основе корреляционной матрицы можно сделать следующие выводы:

- Корреляционные матрицы для исходных и масштабированных данных совпадают.
- Целевой признак классификации "Class_att_le" наиболее сильно коррелирует со следующими признаками:
  1. "Degree_spondylolisthesis" (0.446);
  2. "Pelvic_incidence" (0.353);
  3. "Pelvic_tilt" (0.325)
  4. "Lumbar_lordosis_angle" (0.314) Эти признаки следует оставить в модели классификации.
- Признаки "Pelvic_incidence" и "Sacral_slope" имеют большую корреляцию, поэтому оба признака не следует включать в модель. Будем использовать признак "Pelvic_incidence".
- На основании корреляционной матрицы можно сделать вывод о том, что данные позволяют построить модель машинного обучения.

5) Выбор метрик для последующей оценки качества моделей

В качестве метрик для решения задачи классификации будем использовать:

- Precision - доля верно предсказанных классификатором положительных объектов, из всех объектов, которые классификатор верно или неверно определил как положительные.
- Recall - доля верно предсказанных классификатором положительных объектов, из всех действительно положительных объектов.
- F_1-мера - для объединения precision и recall в единую метрику
- ROC AUC. Основана на вычислении следующих характеристик:
  - True Positive Rate, откладывается по оси ординат. Совпадает с recall.
  - False Positive Rate, откладывается по оси абсцисс. Показывает какую долю из объектов отрицательного класса алгоритм предсказал неверно.

In [28]:
```
# Отрисовка ROC-кривой
def draw_roc_curve(y_true, y_score, pos_label=1, average='micro'):
    fpr, tpr, thresholds = roc_curve(y_true, y_score,
                                     pos_label=pos_label)
    roc_auc_value = roc_auc_score(y_true, y_score, average=average)
    plt.figure()
    lw = 2
    plt.plot(fpr, tpr, color='darkturquoise',
             lw=lw, label='ROC curve (area = %0.2f)' % roc_auc_value)
    plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver operating characteristic')
    plt.legend(loc="lower right")
    plt.show()
```

In [29]:
```
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']=
        # 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=ascendi
        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, asc
        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()
```

6) Выбор наиболее подходящих моделей для решения задачи классификации

Для задачи классификации будем использовать следующие модели:

- Логистическая регрессия
- Метод ближайших соседей
- Машина опорных векторов
- Решающее дерево
- Случайный лес
- Градиентный бустинг

7) Формирование обучающей и тестовой выборок на основе исходного набора данных

In [30]:
```
# Признаки для задачи классификации
class_cols = ['Pelvic_incidence',
              'Pelvic_tilt',
              'Lumbar_lordosis_angle',
              'Degree_spondylolisthesis',
              ]
```

In [31]:
```
X = data[class_cols]
Y = data['Class_att_le']
X.shape
```

Out[31]: (309, 4)

In [32]:
```
# С использованием метода train_test_split разделим выборку на обучающую
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.1
```

In [33]:
```
X_train.shape, X_test.shape, Y_train.shape, Y_test.shape
```

Out[33]: ((278, 4), (31, 4), (278,), (31,))

8) 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

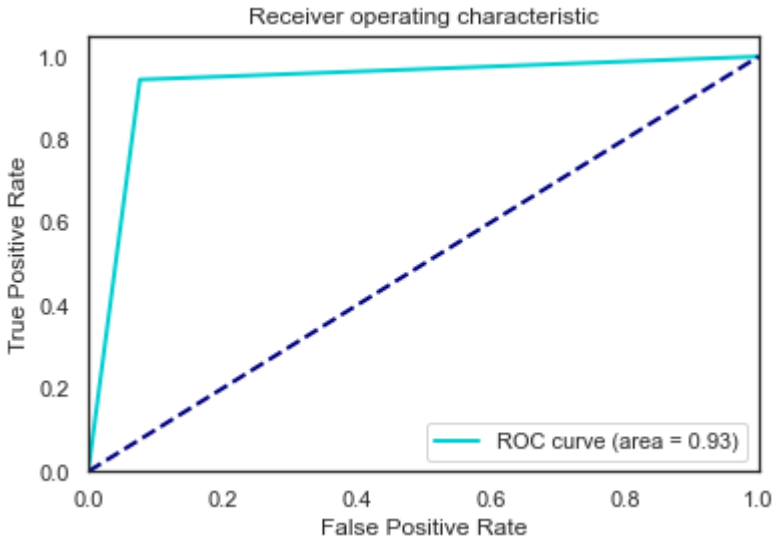
In [34]:
```
# Модели
clas_models = {'LogR': LogisticRegression(),
               'KNN_5':KNeighborsClassifier(n_neighbors=5),
               'SVC':SVC(),
               'Tree':DecisionTreeClassifier(),
               'RF':RandomForestClassifier(),
               'GB':GradientBoostingClassifier()}
```

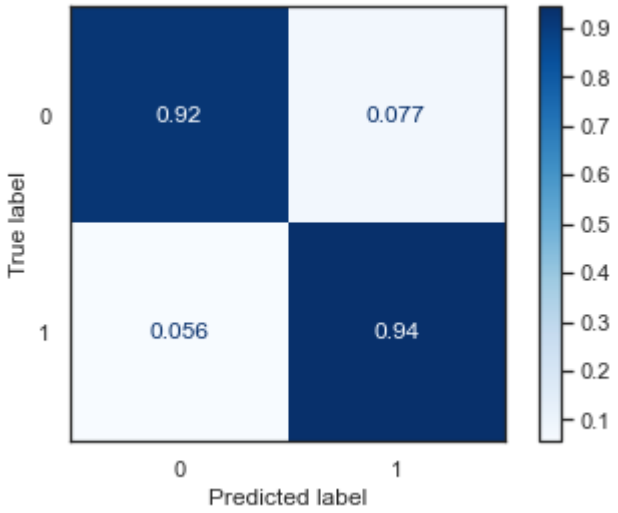
In [35]:
```
# Сохранение метрик
clasMetricLogger = MetricLogger()
```

In [36]:
```
def train_model(model_name, model, MetricLogger):
    model.fit(X_train, Y_train)
    Y_pred = model.predict(X_test)

    precision = precision_score(Y_test.values, Y_pred)
    recall = recall_score(Y_test.values, Y_pred)
    f1 = f1_score(Y_test.values, Y_pred)
    roc_auc = roc_auc_score(Y_test.values, Y_pred)

    MetricLogger.add('precision', model_name, precision)
    MetricLogger.add('recall', model_name, recall)
    MetricLogger.add('f1', model_name, f1)
    MetricLogger.add('roc_auc', model_name, roc_auc)

    print('*******************************************************')
    print(model)
    print('*******************************************************')
    draw_roc_curve(Y_test.values, Y_pred)

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

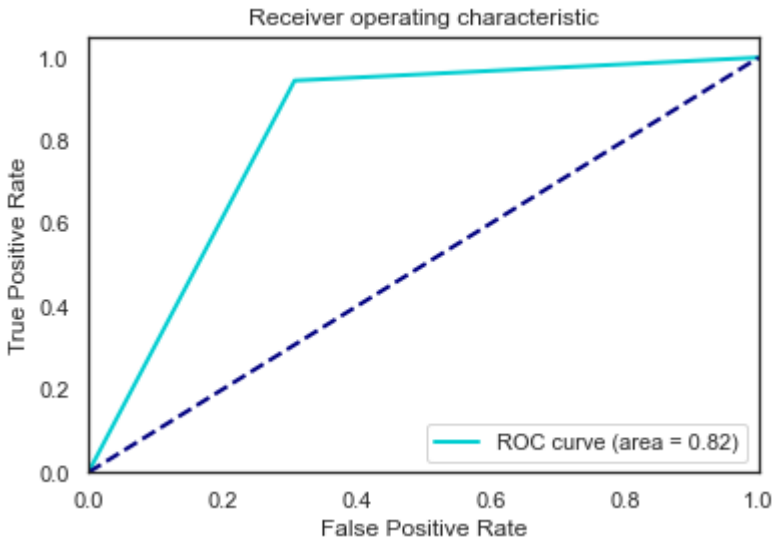
In [37]:
```
for model_name, model in clas_models.items():
    train_model(model_name, model, clasMetricLogger)
```

```
******************************************************
LogisticRegression()
******************************************************
```

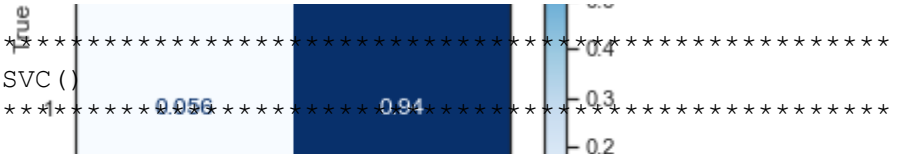


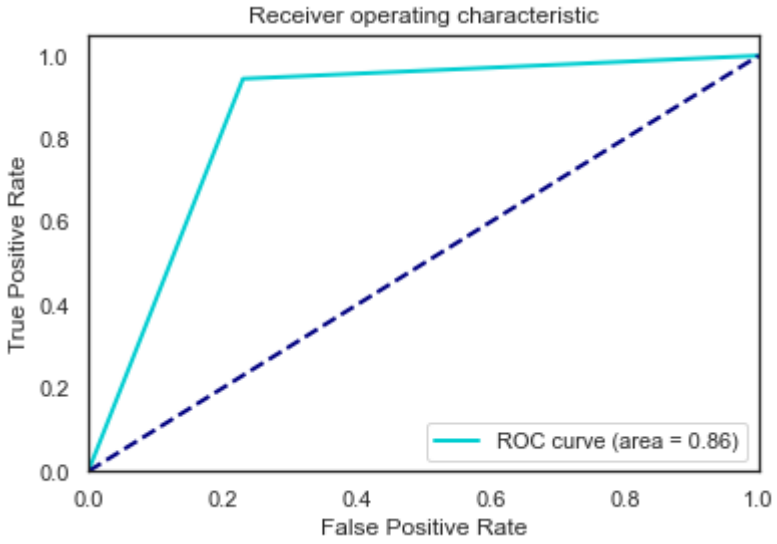


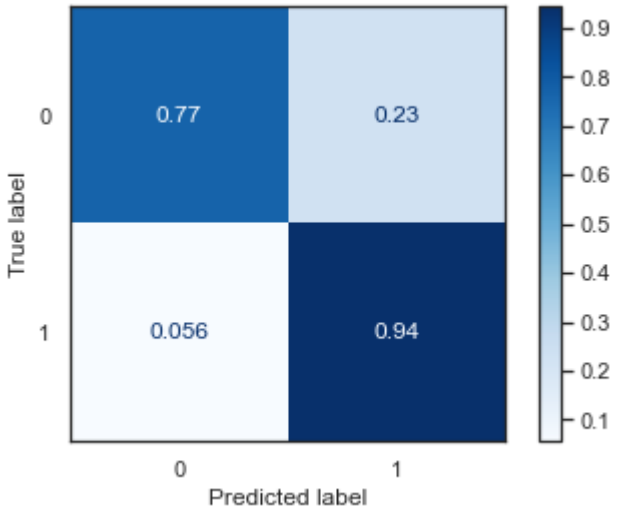

```
******************************************************
KNeighborsClassifier()
******************************************************
```



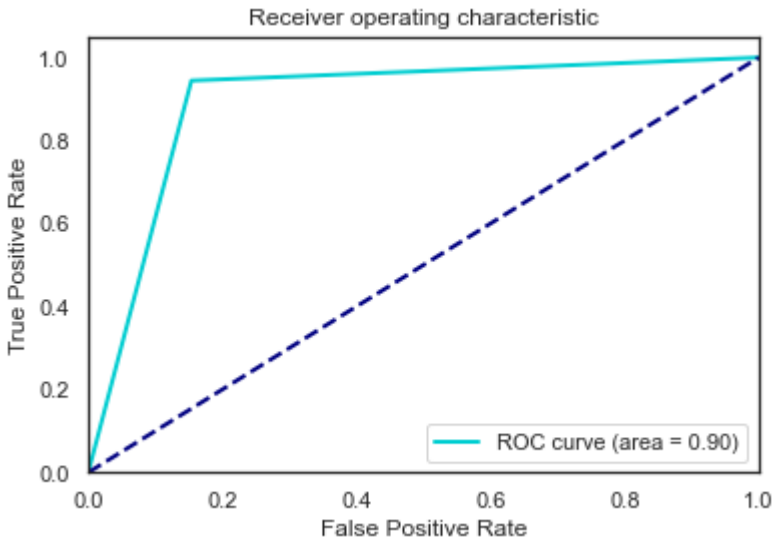
****
SVC()
****

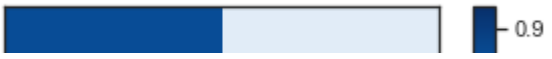




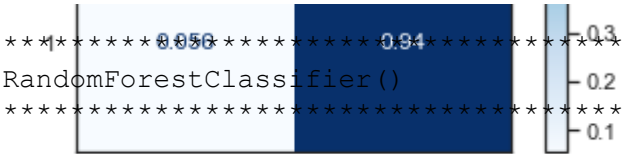


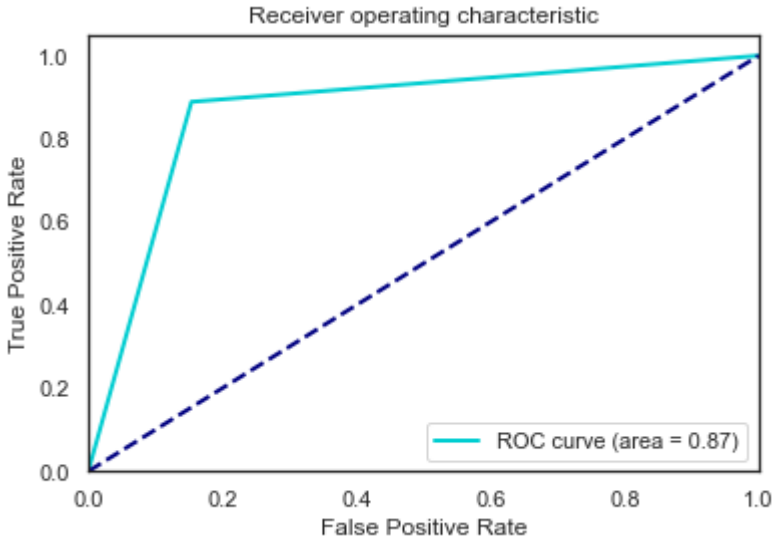
****
DecisionTreeClassifier()
****

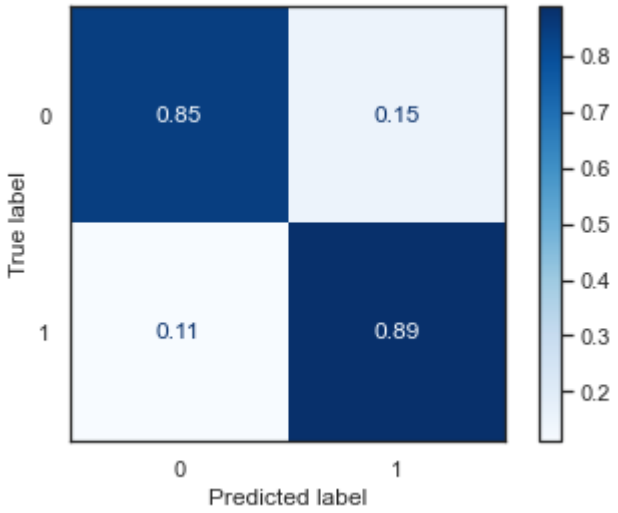




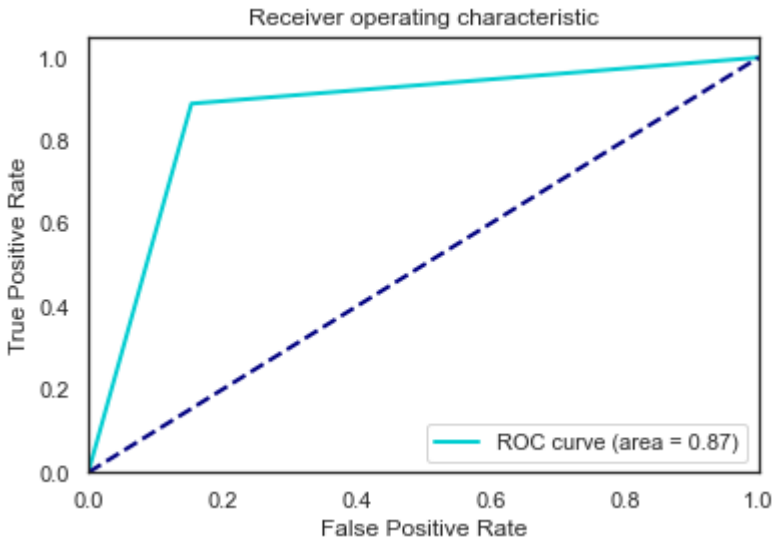
****
RandomForestClassifier()
****

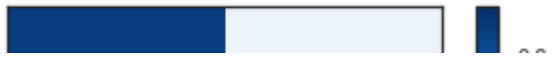


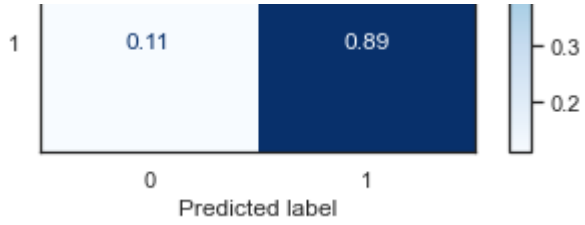




****
GradientBoostingClassifier()
****







ыбранных моделей.

In [38]:
```
n_range = np.array(range(1,100,1))
tuned_parameters = [{'n_neighbors': n_range}]
tuned_parameters
```

Out[38]:
```
[{'n_neighbors': array([ 1,  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, 31, 32, 33, 34,
        35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51,
        52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68,
        69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85,
        86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99])}]
```

In [39]:
```
gs_KNN = GridSearchCV(KNeighborsClassifier(), tuned_parameters, cv=5, sco
gs_KNN.fit(X_train, Y_train)
```

Out[39]:
```
GridSearchCV(cv=5, estimator=KNeighborsClassifier(),
             param_grid=[{'n_neighbors': array([ 1,  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, 31, 32, 33, 34,
       35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51,
       52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68,
       69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85,
       86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99])}],
             scoring='roc_auc')
```

In [40]:
```
# Лучшая модель
gs_KNN.best_estimator_
```

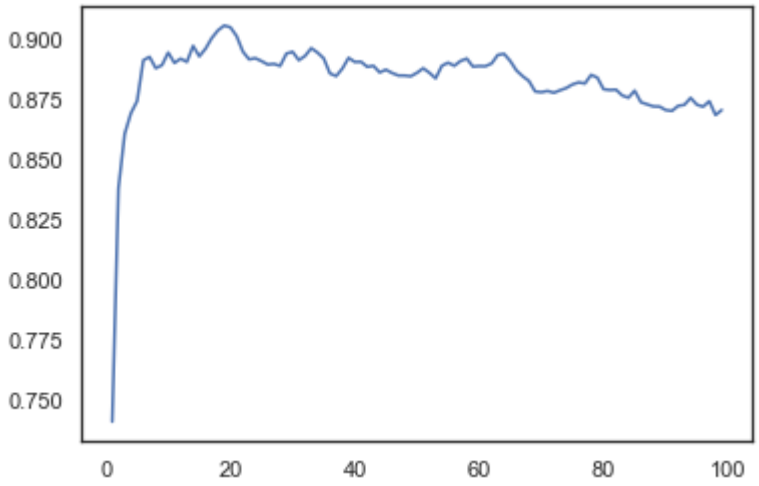
Out[40]:
```
KNeighborsClassifier(n_neighbors=19)
```

In [41]:
```
# Лучшее значение параметров
gs_KNN.best_params_
```

Out[41]:
```
{'n_neighbors': 19}
```

In [42]:
```
plt.plot(n_range, gs_KNN_cv_results['mean_test_score'])
```

Out[42]:
```
[<matplotlib.lines.Line2D at 0x1cf4fb366d0>]
```



Логистическая регрессия

In [43]:
```
grid={"C":np.logspace(-3,3,3)}
gs_LogR = GridSearchCV(LogisticRegression(), grid, cv=5, scoring='roc_auc
gs_LogR.fit(X_train, Y_train)
```

Out[43]:
```
GridSearchCV(cv=5, estimator=LogisticRegression(),
             param_grid={'C': array([1.e-03, 1.e+00, 1.e+03])},
             scoring='roc_auc')
```

In [44]:
```
# Лучшая модель
gs_LogR.best_estimator_
```

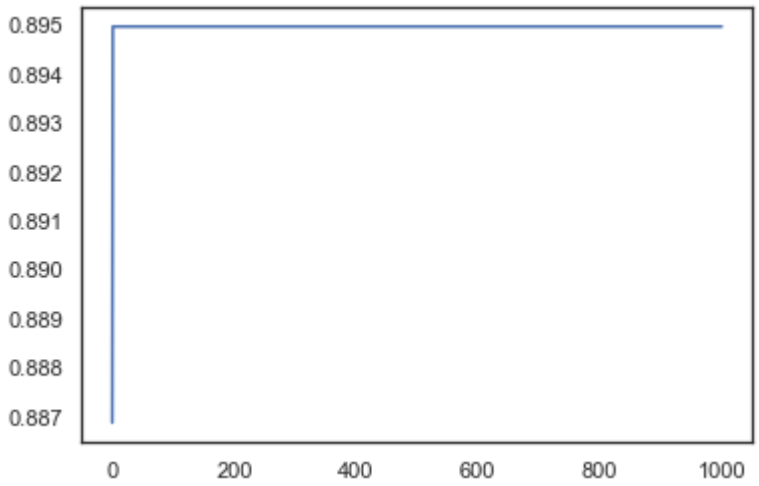
Out[44]:
```
LogisticRegression()
```

In [45]:
```
# Лучшее значение параметров
gs_LogR.best_params_
```

Out[45]:
```
{'C': 1.0}
```

In [46]:
```
# Изменение качества на тестовой выборке
plt.plot(np.logspace(-3,3,3), gs_LogR.cv_results_['mean_test_score'])
```

Out[46]:
```
[<matplotlib.lines.Line2D at 0x1cf4f35aaf0>]
```



Машина опорных векторов

In [47]:
```
SVC_grid={"C":np.logspace(-3,5,12)}
gs_SVC = GridSearchCV(SVC(), SVC_grid, cv=5, scoring='roc_auc')
gs_SVC.fit(X_train, Y_train)
```

Out[47]:
```
GridSearchCV(cv=5, estimator=SVC(),
             param_grid={'C': array([1.00000000e-03, 5.33669923e-03,
2.84803587e-02, 1.51991108e-01,
       8.11130831e-01, 4.32876128e+00, 2.31012970e+01, 1.23284674e+02,
       6.57933225e+02, 3.51119173e+03, 1.87381742e+04, 1.00000000e+0
5])},
             scoring='roc_auc')
```

In [48]:
```
# Лучшая модель
gs_SVC.best_estimator_
```

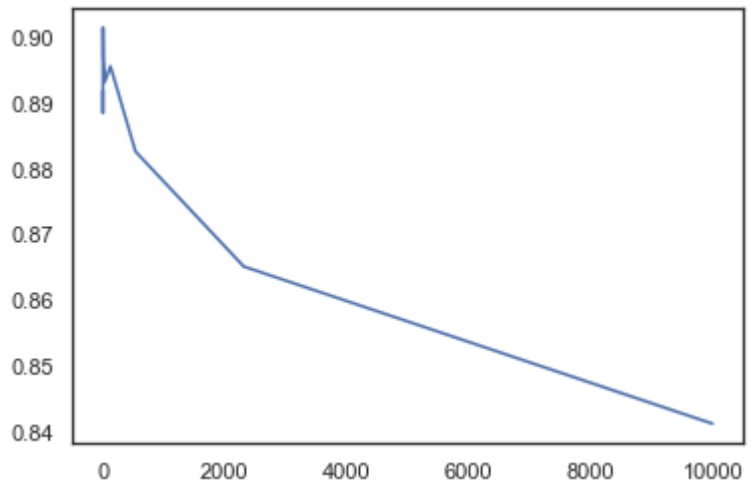
Out[48]:
```
SVC(C=4.328761281083062)
```

In [49]:
```
# Лучшее значение параметров
gs_SVC.best_params_
```

Out[49]:
```
{'C': 4.328761281083062}
```

In [50]:
```
# Изменение качества на тестовой выборке
plt.plot(np.logspace(-3,4,12), gs_SVC.cv_results_['mean_test_score'])
```

Out[50]:
```
[<matplotlib.lines.Line2D at 0x1cf4ed99940>]
```



Решающее дерево

In [51]:
```
tree_params={"max_depth":range(1,20), "max_features":range(1,5)}
gs_Tree = GridSearchCV(DecisionTreeClassifier(), tree_params, cv=5, scori
gs_Tree.fit(X_train, Y_train)
```

Out[51]:
```
GridSearchCV(cv=5, estimator=DecisionTreeClassifier(),
             param_grid={'max_depth': range(1, 20),
                         'max_features': range(1, 5)},
             scoring='precision')
```

In [52]:
```
# Лучшая модель
gs_Tree.best_estimator_
```

Out[52]:
```
DecisionTreeClassifier(max_depth=1, max_features=4)
```

In [53]:
```
# Лучшее значение параметров
gs_Tree.best_params_
```

Out[53]:
```
{'max_depth': 1, 'max_features': 4}
```

Случайный лес

In [54]:
```
RF_params={"max_leaf_nodes":range(2,12), "max_samples":range(2,22)}
gs_RF = GridSearchCV(RandomForestClassifier(), RF_params, cv=5, scoring='
gs_RF.fit(X_train, y_train)
```

Out[54]:
```
GridSearchCV(cv=5, estimator=RandomForestClassifier(),
             param_grid={'max_leaf_nodes': range(2, 12),
                         'max_samples': range(2, 22)},
             scoring='roc_auc')
```

In [55]:
```
# Лучшая модель
gs_RF.best_estimator_
```

Out[55]:
```
RandomForestClassifier(max_leaf_nodes=11, max_samples=17)
```

In [56]:
```
# Лучшее значение параметров
gs_RF.best_params_
```

Out[56]:
```
{'max_leaf_nodes': 11, 'max_samples': 17}
```

Градиентный бустинг

In [57]:
```
GB_params={"max_features":range(1,4), "max_leaf_nodes":range(2,22)}
gs_GB = GridSearchCV(GradientBoostingClassifier(), GB_params, cv=5, scori
gs_GB.fit(X_train, y_train)
```

Out[57]:
```
GridSearchCV(cv=5, estimator=GradientBoostingClassifier(),
             param_grid={'max_features': range(1, 4),
                         'max_leaf_nodes': range(2, 22)},
             scoring='f1')
```

In [58]:
```
# Лучшая модель
gs_GB.best_estimator_
```

Out[58]:
```
GradientBoostingClassifier(max_features=2, max_leaf_nodes=10)
```

In [59]:
```
# Лучшее значение параметров
gs_GB.best_params_
```

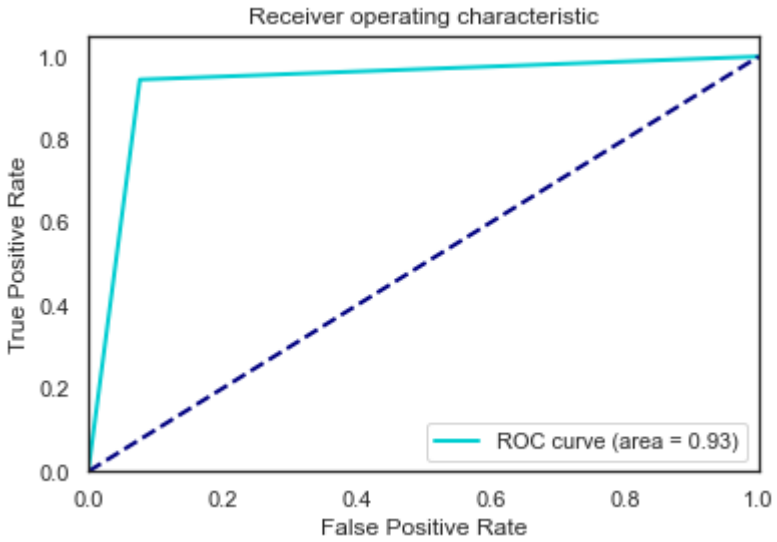
Out[59]:
```
{'max_features': 2, 'max_leaf_nodes': 10}
```

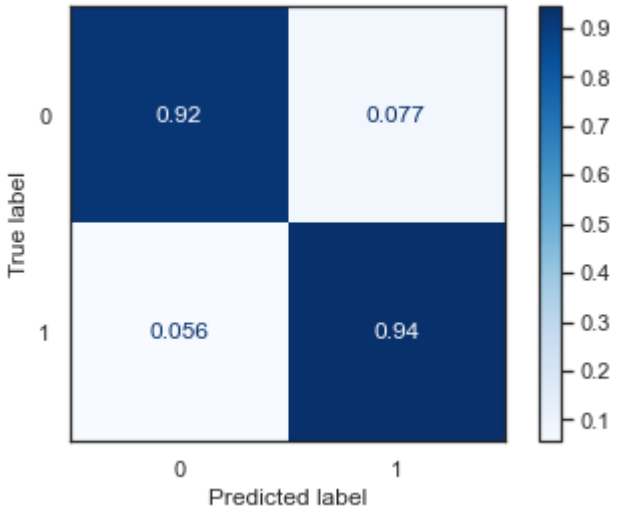
10) Повторение пункта 8 для найденных оптимальных значений гиперпараметров. Сравнение качества полученных моделей с качеством baseline-моделей

In [60]:
```
models_grid = { 'LogR_new':gs_LogR.best_estimator_,
                'KNN_new':gs_KNN.best_estimator_,
                 'SVC_new':gs_SVC.best_estimator_,
               'Tree_new':gs_Tree.best_estimator_,
                'RF_new':gs_RF.best_estimator_,
               'GB_new':gs_GB.best_estimator_
               }
```

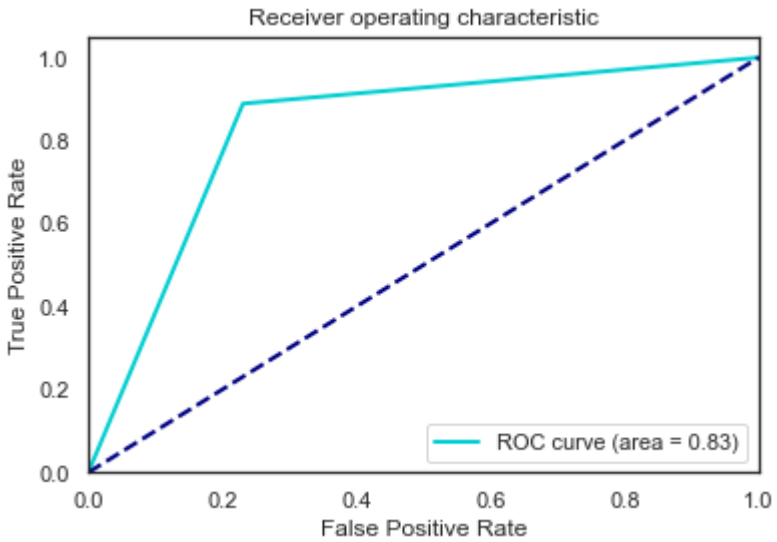
In [61]:
```
for model_name, model in models_grid.items():
    train_model(model_name, model, clasMetricLogger)
```

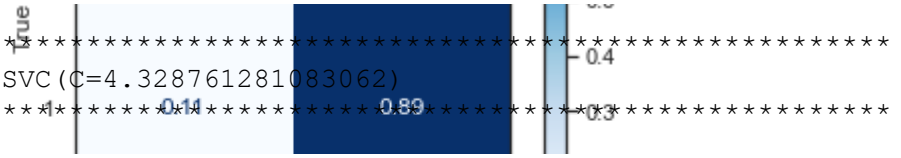
```
******************************************************
LogisticRegression()
******************************************************
```



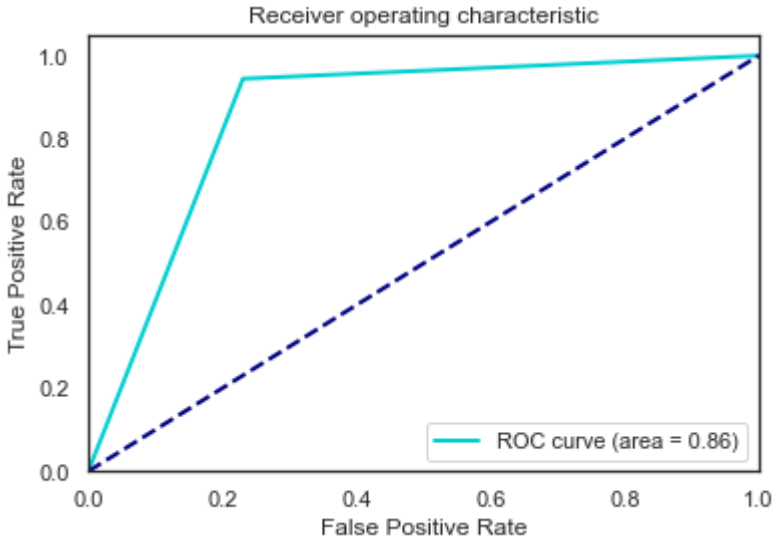


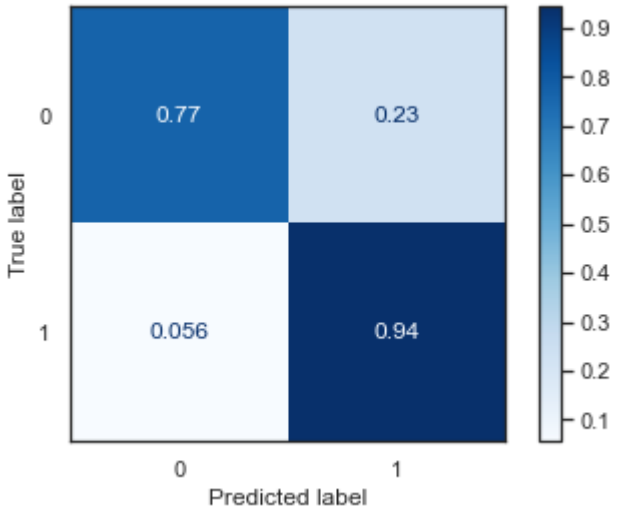

```
******************************************************
KNeighborsClassifier(n_neighbors=19)
******************************************************
```



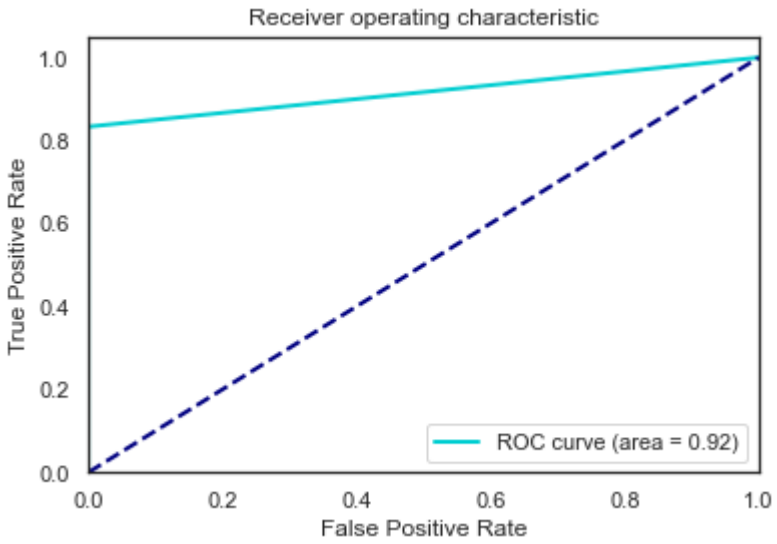


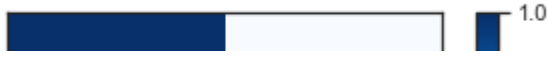
****
SVC(C=4.328761281083062)
****



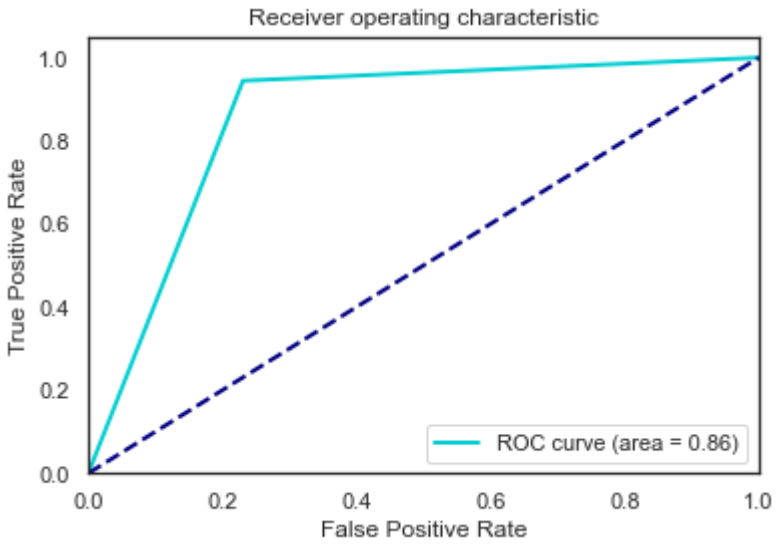


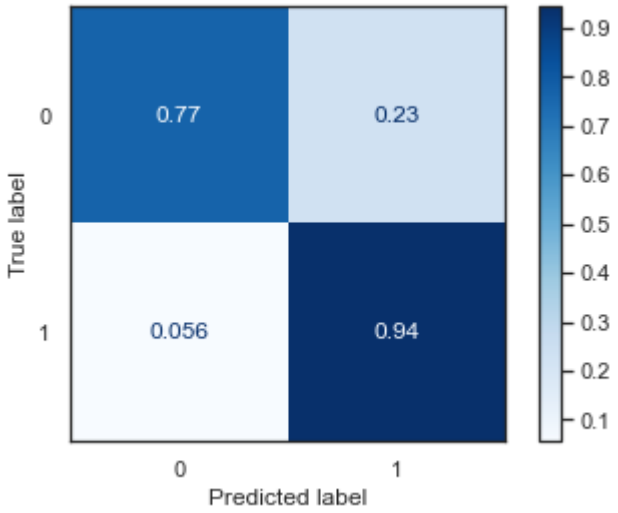
****
DecisionTreeClassifier(max_depth=1, max_features=4)
****



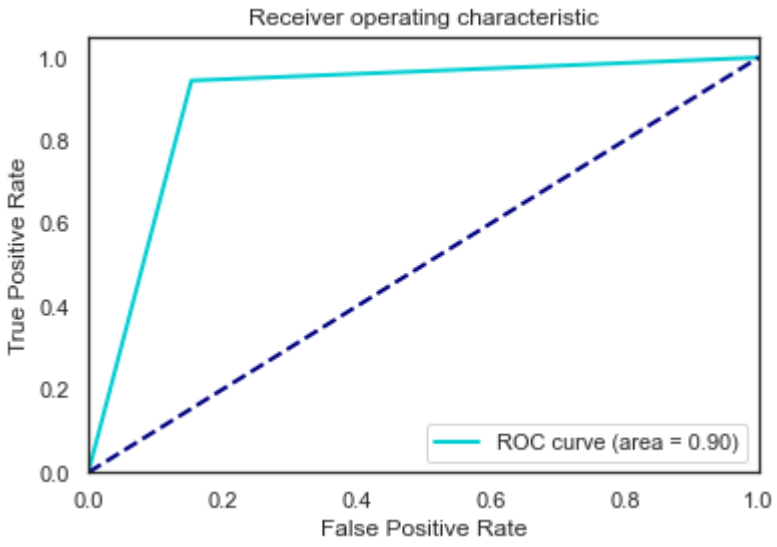


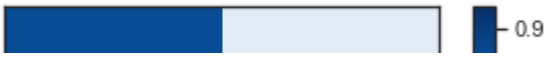
```
**************************************************
RandomForestClassifier(max_leaf_nodes=11, max_samples=17)
**************************************************
```



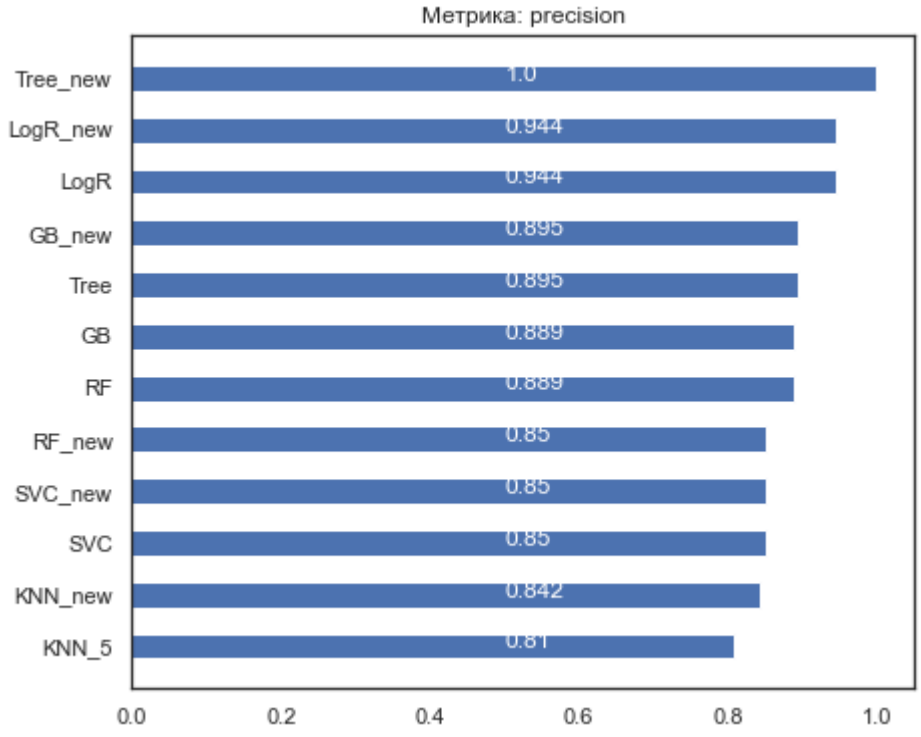


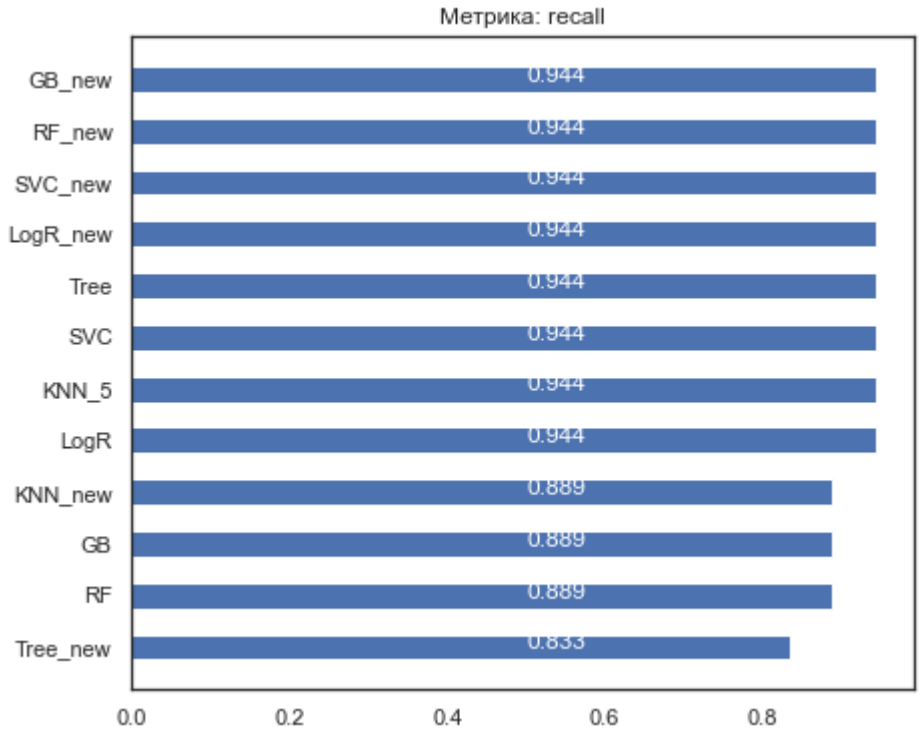

```
**************************************************
GradientBoostingClassifier(max_features=2, max_leaf_nodes=10)
**************************************************
```

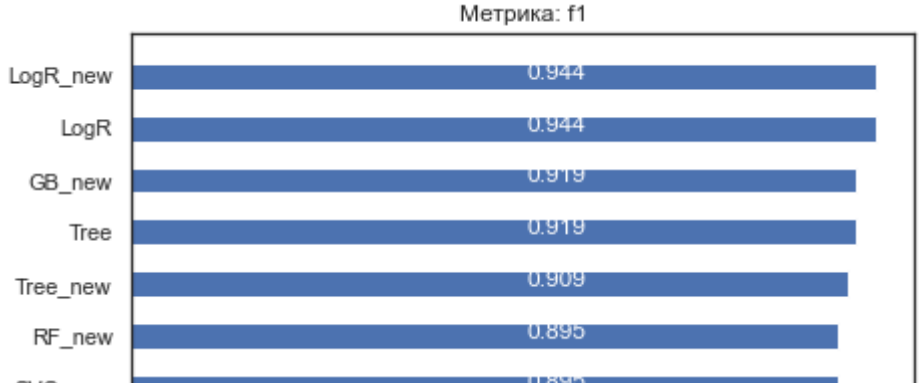


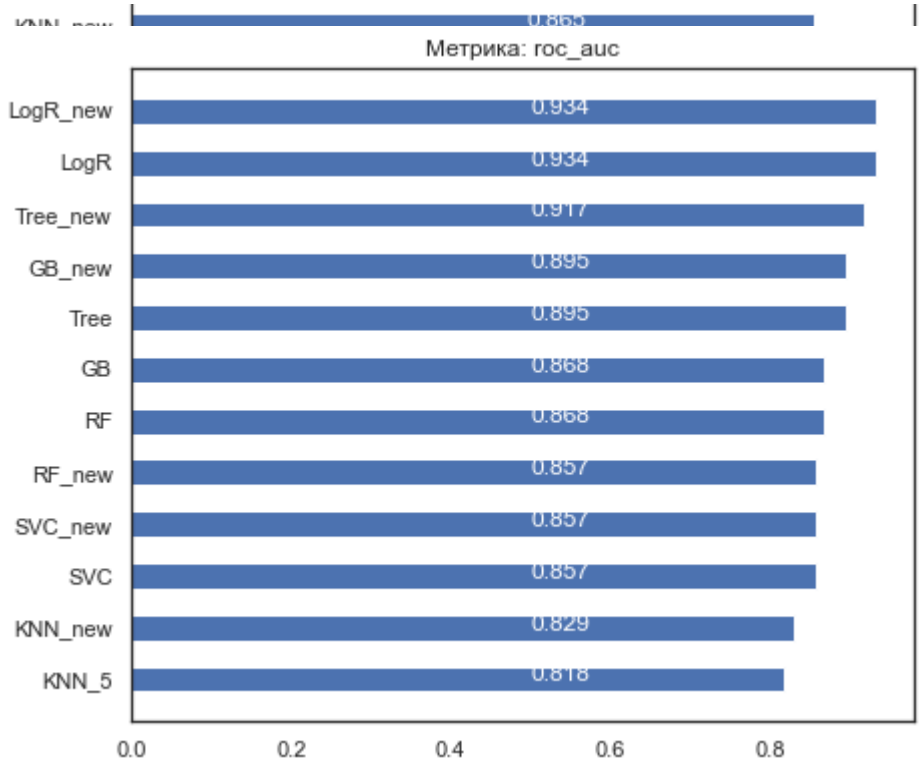


In [63]:
```
# Построим графики метрик качества модели
for metric in clas_metrics:
    clasMetricLogger.plot('Метрика: ' + metric, metric, figsize=(7, 6))
```









Вывод: на основании трех метрик из четырех, лучшими моделями оказались случайный лес и логистическая регрессия.

Реализация макета с помощью стримлит (на основе 6 лабораторной работы) приведена в файле Lab6.py

Применение к выбранному набору данных произвольной библиотеки AutoML и сравнение качества моделей, полученных вручную и с использованием AutoML

In [64]:
```
from tpot import TPOTClassifier
```

In [65]:
```
pipeline_optimizer = TPOTClassifier()
pipeline_optimizer = TPOTClassifier(generations=5, population_size=20, cv
pipeline_optimizer.fit(X_train, Y_train)
print(pipeline_optimizer.score(X_test, Y_test))
pipeline_optimizer.export('tpot_exported_pipeline.py')
```

```
Generation 1 - Current best internal CV score: 0.8238311688311688

Generation 2 - Current best internal CV score: 0.8238311688311688
```

Вывод: результаты AutoML схожи с результатами, найденными вручную.

Разработка макета веб-приложения, предназначенного для анализа данных.

Вариант 1. Макет должен быть реализован для одной модели машинного обучения. Макет должен позволять:

- Задавать гиперпараметры алгоритма,
- Производить обучение,
- Осуществлять просмотр результатов обучения, в том числе в виде графиков.

In [ ]:
```
import streamlit as st
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split, learning_curve
from sklearn.metrics import plot_confusion_matrix, accuracy_score, roc_cu
from sklearn.metrics import roc_auc_score, f1_score
from sklearn.preprocessing import MinMaxScaler
from catboost import Pool, CatBoostClassifier

# Запуск приложения streamlit run C:/Users/User/Desktop/TMO_NST/Lab6/Lab6

def load():
    col_list = ['Pelvic_incidence',
                'Pelvic_tilt',
                'Lumbar_lordosis_angle',
                'Sacral_slope',
                'Pelvic_radius',
                'Degree_spondylolisthesis',
                'Pelvic_slope',
                'Direct_tilt',
                'Thoracic_slope',
                'Cervical_tilt',
                'Sacrum_angle',
                'Scoliosis_slope',
                'Class_att',
                'To_drop']
    data = pd.read_csv('Dataset_spine.csv', names=col_list, header=1, sep
    data.drop('To_drop', axis=1, inplace=True)
    return data

#Готовим данные к ML
def preprocess_data(data):
    scale_cols = ['Pelvic_incidence',
                  'Pelvic_tilt',
                  'Lumbar_lordosis_angle',
                  'Sacral_slope',
                  'Pelvic_radius',
                  'Degree_spondylolisthesis',
                  'Pelvic_slope',
                  'Direct_tilt',
                  'Thoracic_slope',
                  'Cervical_tilt',
                  'Sacrum_angle',
                  'Scoliosis_slope']
```

```
    sc1 = MinMaxScaler()
    sc1_data = sc1.fit_transform(data[scale_cols])
    for i in range(len(scale_cols)):
        data[scale_cols[i]] = sc1_data[:, i]
    data['Class_att'] = data['Class_att'].map({'Abnormal': 1, 'Normal': 0
    # Разделим данные на целевой столбец и признаки
    X = data.drop("Class_att", axis=1)
    Y = data["Class_att"]
    # С использованием метода train_test_split разделим выборку на обучак
    X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0
    return X_train, X_test, Y_train, Y_test

#Отрисовка графика ROC_CURVE
def draw_roc_curve(y_true, y_score, ax, pos_label=1, average='micro'):
    fpr, tpr, thresholds = roc_curve(y_true, y_score,
                                     pos_label=pos_label)
    roc_auc_value = roc_auc_score(y_true, y_score, average=average)
    # plt.figure()
    lw = 2
    ax.plot(fpr, tpr, color='darkorange',
            lw=lw, label='ROC curve (area = %0.2f)' % roc_auc_value)
    ax.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
    ax.set_xlim([0.0, 1.0])
    ax.set_xlim([0.0, 1.05])
    ax.set_xlabel('False Positive Rate')
    ax.set_ylabel('True Positive Rate')
    ax.set_title('Receiver operating characteristic')
    ax.legend(loc="lower right")

#вывод метрик ML
def print_metrics(X_train, Y_train, X_test, Y_test, clf):
    clf.fit(X_train, Y_train)
    target = clf.predict(X_test)
    test_score = accuracy_score(Y_test, target)
    roc_res = clf.predict_proba(X_test)
    roc_auc = roc_auc_score(Y_test, roc_res[:, 1])
    f1_test_score = f1_score(Y_test, target)
    st.write(f"accuracy (точность): {test_score}")
    st.write(f"f1 метрика: {f1_test_score}")
    st.write(f"ROC AUC: {roc_auc}")
    fig1, ax1 = plt.subplots()
    draw_roc_curve(Y_test, roc_res[:, 1], ax1)
    st.pyplot(fig1)
    fig2, ax2 = plt.subplots(figsize=(10, 5))
    plot_confusion_matrix(clf, X_test, Y_test, ax=ax2, display_labels=['1
                          cmap = 'Purples', normalize='true')
    ax2.set(title="Confusion matrix")
    st.pyplot(fig2)
    return test_score

#Вывод кривой обучения
def plot_learning_curve(data_X, data_y, clf, name='accuracy', scoring='ac
    train_sizes, train_scores, test_scores = learning_curve(estimator=clf
            scoring=scoring, X=data_X, y=data_y, train_sizes=np.linspace(
    train_mean = np.mean(train_scores, axis=1)
    train_std = np.std(train_scores, axis=1)
    test_mean = np.mean(test_scores, axis=1)
    test_std = np.std(test_scores, axis=1)
    fig = plt.figure(figsize=(7, 5))
    plt.plot(train_sizes, train_mean, color='blue', marker='o',
```

```
            markersize=5, label=f'тренировочная {name}-мера')
    plt.fill_between(train_sizes, train_mean + train_std,
                     train_mean - train_std, alpha=0.15, color='blue')
    plt.plot(train_sizes, test_mean, color='green',
             linestyle='--', marker='s', markersize=5,
             label=f'проверочная {name}-мера')
    plt.fill_between(train_sizes, test_mean + test_std,
                     test_mean - test_std, alpha=0.15, color='green')
    plt.grid()
    plt.legend(loc='lower right')
    plt.xlabel('Число тренировочных образцов')
    plt.ylabel(f'{name}-мера')
    st.pyplot(fig)


if __name__ == '__main__':
    st.title('Метод градиентного бустинга')
    data = load()
    data_X_train, data_X_test, data_y_train, data_y_test = preprocess_dat

    # Будем показывать матрицу только по запросу, чттобы не тормозить про
    if st.checkbox('Показать корреляционную матрицу'):
        fig_corr, ax = plt.subplots(figsize=(20, 20))
        sns.heatmap(data.corr(), annot=True, cmap = 'Purples', fmt='.3f')
        st.pyplot(fig_corr)

    # выбор гиперпараметров в сайдбаре
    st.sidebar.subheader('Гиперпараметры :')
    estimators = st.sidebar.slider('Количество деревьев: ',
                                   min_value=1, max_value=100, value=5, s
    max_depth = st.sidebar.slider('Максимальная глубина',
                                  min_value=1, max_value=10, value=4, ste
    eval_metric = st.sidebar.selectbox('Оптимизируемая метрика:', ('Accur

    # Вывод результатов
    translation_dict = {'Accuracy': 'accuracy', 'F1': 'f1', 'AUC': 'roc_a
    gd = CatBoostClassifier(n_estimators=estimators, max_depth=max_depth,
                            eval_metric=eval_metric, random_state=1)
    result = print_metrics(data_X_train, data_y_train, data_X_test, data_
    data_X = pd.concat([data_X_train, data_X_test])
    data_y = pd.concat([data_y_train, data_y_test])
    plot_learning_curve(data_X, data_y, gd, name=translation_dict.get(eva
                        scoring=translation_dict.get(eval_metric))
    #  показать данные
    if st.checkbox('Показать первые 10 строк датасета "Dataset_spine"'):
        st.write(data.head(10))
```

Гиперпараметры :

Количество деревьев:

5

1 100

Максимальная глубина

4

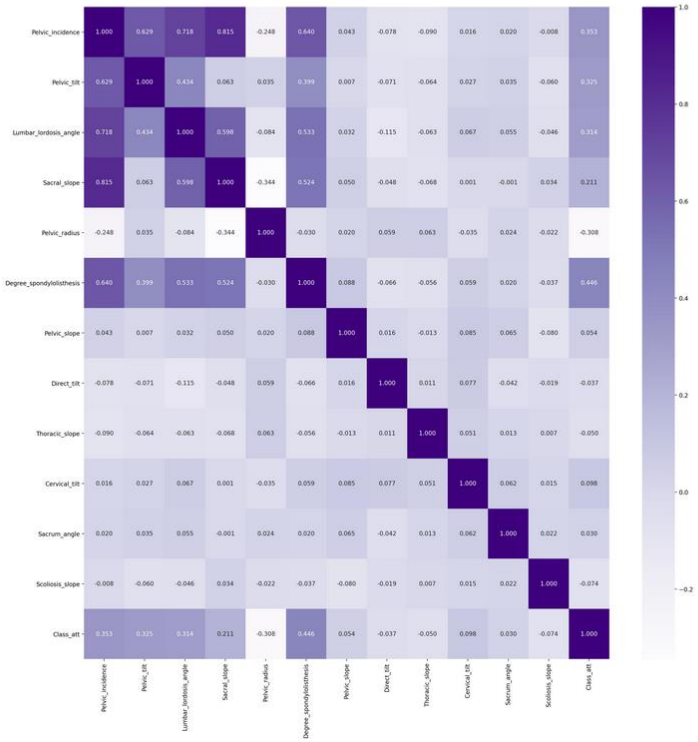
1 10

Оптимизируемая метрика:

Accuracy

Метод градиентного бустинга

Показать корреляционную матрицу



Гиперпараметры :

Количество деревьев:

5

1 100

Максимальная глубина

4

1 10

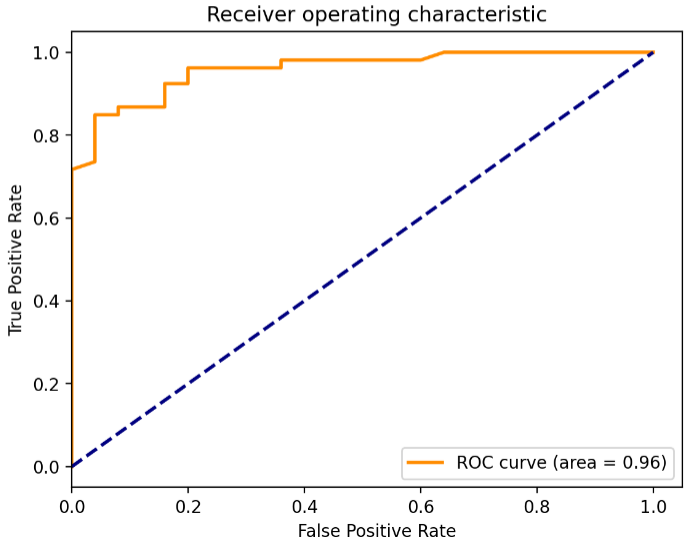
Оптимизируемая метрика:

Accuracy

accuracy (точность): 0.8846153846153846

f1 метрика: 0.9158878504672898

ROC AUC: 0.9584905660377359



Гиперпараметры :

Количество деревьев:

5

1 100

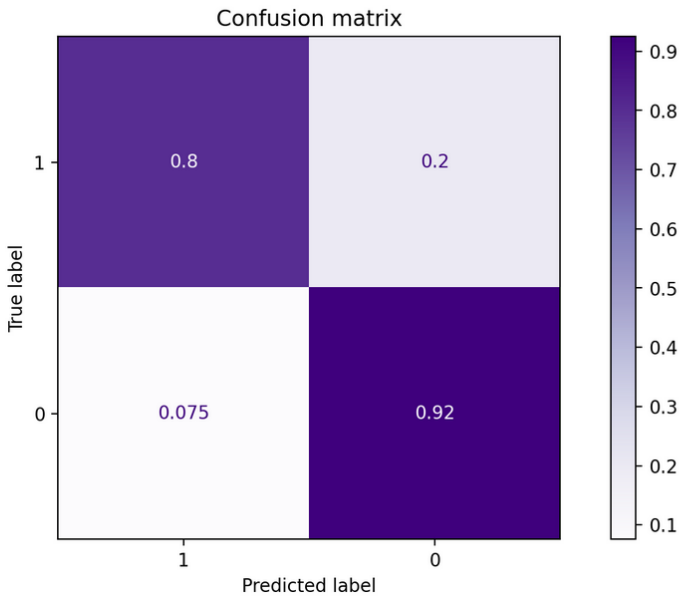
Максимальная глубина

4

1 10

Оптимизируемая метрика:

Accuracy



Гиперпараметры :

Количество деревьев:

5

1 100

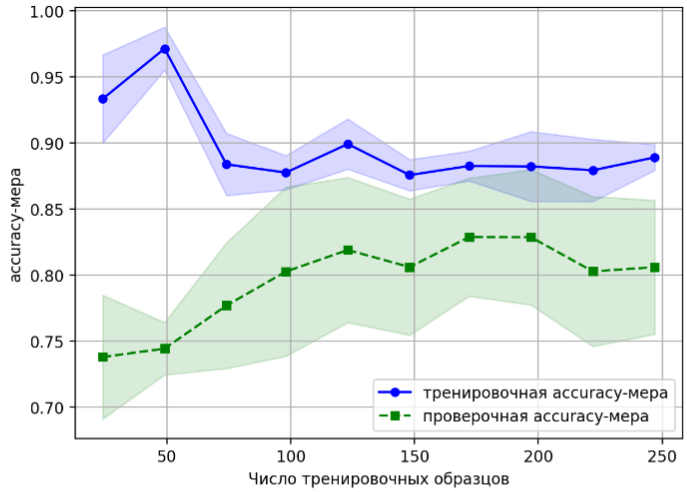
Максимальная глубина

4

1 10

Оптимизируемая метрика:

Accuracy



Показать первые 10 строк датасета "Dataset_spine"

	Pelvic_incidence	Pelvic_tilt	Lumbar_lordosis_angle	Sacral_slope	Pelvic_
0	0.1245	0.2968	0.0986	0.1446	
1	0.4117	0.5139	0.3230	0.3077	
2	0.4162	0.5574	0.2713	0.2894	
3	0.2273	0.2895	0.1281	0.2470	
4	0.1360	0.3657	0.0996	0.1199	
5	0.2632	0.4004	0.2073	0.2240	
6	0.1854	0.3092	0.1346	0.1966	
7	0.1702	0.3588	0.2568	0.1563	
8	0.1016	0.2066	0.2501	0.1694	
9	0.2272	0.3500	0.1551	0.2156	

Заключение

В данном курсовом проекте была решена типовая задача машинного обучения. Был выбран набор данных для построения моделей машинного обучения, проведен разведочный анализ данных и построены графики, необходимые для понимания структуры данных. Были выбраны признаки, подходящие для построения моделей, масштабированы данные и проведен корреляционный анализ данных. Это позволило сформировать промежуточные выводы о возможности построения моделей машинного обучения.

На следующем этапе были выбраны метрики для последующей оценки качества моделей и наиболее подходящие модели для решения задачи классификации. Затем были сформированы обучающая и тестовая выборки на основе исходного набора данных и построено базовое решение для выбранных моделей без подбора гиперпараметров.

Следующим шагом был подбор гиперпараметров для выбранных моделей, после чего мы смогли сравнить качество полученных моделей с качеством baseline-моделей. Большинство моделей, для которых были подобраны оптимальные значения гиперпараметров, показали лучший результат.

В заключение, были сформированы выводы о качестве построенных моделей на основе выбранных метрик. Для наглядности результаты сравнения качества были отображены в виде графиков, а также сделаны выводы в форме текстового описания. Четыре метрики показали, что для выбранного набора данных лучшей моделью оказалась «машина опорных векторов».

Список использованных источников информации

1. Ю.Е. Гапанюк, Лекции по курсу «Технологии машинного обучения» 2020-2021 учебный год.

2. Scikit-learn Machine Learning in Python [Электронный ресурс]. URL: https://scikit- learn.org/stable/ (дата обращения: 30.05.2021)

3. Lower Back Pain Symptoms Dataset [Электронный ресурс]. URL: https://www.kaggle.com/sammy123/lower-back-pain-symptoms-dataset (дата обращения: 30.05.2021)

4. Хирургическая анатомия пояснично-крестцового сочленения позвоночника [Электронный ресурс]. URL: https://meduniver.com/Medical/neiroxirurgia/anatomia_poiasnichno-krestcovogo-sochlenenia.html (дата обращения: 30.05.2021)