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 ИУ_____

КАФЕДРА ИУ5_____

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К КУРСОВОМУ ПРОЕКТУ

НА ТЕМУ:

_____Решение задачи машинного обучения_____

Студент группы ИУ5-61Б___ _____ _____Кочетков М.Д.
(Группа) (Подпись, дата) (И.О.Фамилия)

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

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

2020 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 курсового проекта

по дисциплине «Технологии машинного обучения»_____

Студент группы ИУ5-61Б_____

_____Кочетков Михаил Дмитриевич_____
(Фамилия, имя, отчество)

Тема курсового проекта “Бинарная классификация”_____
_____
_____

Направленность КП (учебный, исследовательский, практический, производственный, др.)
_____
Источник тематики (кафедра, предприятие, НИР) _____

График выполнения проекта: 25% к ___ нед., 50% к ___ нед., 75% к __ нед., 100% к ___ нед.

Задание решение задачи машинного обучения на основе материалов дисциплины. Выполняется студентом единолично.
_____
_____
_____
_____

Оформление курсового проекта:

Расчетно-пояснительная записка на __32___ листах формата А4.
Перечень графического (иллюстративного) материала (чертежи, плакаты, слайды и т.п.)
_____
_____
_____
_____

Дата выдачи задания « 12 » февраля 2020 г.

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

Студент _____ _____Кочетков М.Д.
(Подпись, дата) (И.О.Фамилия)

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

Оглавление

1. Задание

В данной курсовой работе необходимо предпринять следующие шаги:

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 постройение графиков обучения и валидации, влияния значений гиперпарметров на качество моделей и т.д.

2. 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 данной курсовой работы необходимо применить навыки, полученные в течение курса «Технологии машинного обучения», и обосновать полученные результаты.

3. Постановка задачи

В данной курсовой работе ставится задача определения наличия у человека диабета с помощью методов машинного обучения: "Logistic Regression", "Support vector machine", "Decision tree", "Gradient boosting", "Random forest". С помощью различных метрик выбор метода, который наиболее эффективно и качественно определяет значение целевого признака.

4. Решение поставленной задачи

4.1. 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

- Описание выбранного датасета

Данный датасет состоит из данных, собранных по базе данных пациентов, которым пытались диагностировать диабет. В исследовании принимали участия только женщины старше 21 года. На основе данного датасета можно сделать предположение о наличии у пациента диабета.

- Информация об атрибутах:
  - ✓ Pregnancies – количество беременностей
  - ✓ Glucose - концентрация глюкозы в плазме крови через 2 часа при пероральном тесте на толерантность к глюкозе
  - ✓ BloodPressure - Диастолическое артериальное давление (мм рт. ст.)
  - ✓ SkinThickness - Толщина кожной складки трицепса (мм)
  - ✓ Insulin - 2-часовой сывороточный инсулин (му Ед / мл)
  - ✓ BMI - Индекс массы тела (вес в кг/(рост в m)^2)
  - ✓ DiabetesPedigreeFunction – Функция отражающая наследственность пациента по диабетическому признаку
  - ✓ Age – Возраст (в годах)
  - ✓ Outcome – Диагноз (0 – диабета нет, 1 – диабет есть)

В рассматриваемом примере будем решать ***задачу классификации***. Для этого в качестве целевого признака будем использовать атрибут " Outcome " (Диагноз). Поскольку признак содержит только два значения: отрицательный или положительный, то это задача бинарной классификации.

- Импорт библиотек

```
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_report
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_error, 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")
```

- Загрузка данных.

```
data = pd.read_csv('../data/diabetes.csv')
```

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

- Основные характеристики датасета

```
data.head()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

```
# Размер исходного датасета - 768 строк, 9 колонок
data.shape
```

```
(768, 9)
```

```
data.columns
targetColumn = "Outcome"
```

```
data.dtypes
```

```
Pregnancies                 int64
Glucose                     int64
BloodPressure               int64
SkinThickness               int64
Insulin                     int64
BMI                       float64
DiabetesPedigreeFunction  float64
Age                         int64
Outcome                     int64
dtype: object
```

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

```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

```
data.corr()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
Pregnancies	1.000000	0.129459	0.141282	-0.081672	-0.073535	0.017683	-0.033523	0.544341	0.221898
Glucose	0.129459	1.000000	0.152590	0.057328	0.331357	0.221071	0.137337	0.263514	0.466581
BloodPressure	0.141282	0.152590	1.000000	0.207371	0.088933	0.281805	0.041265	0.239528	0.065068
SkinThickness	-0.081672	0.057328	0.207371	1.000000	0.436783	0.392573	0.183928	-0.113970	0.074752
Insulin	-0.073535	0.331357	0.088933	0.436783	1.000000	0.197859	0.185071	-0.042163	0.130548
BMI	0.017683	0.221071	0.281805	0.392573	0.197859	1.000000	0.140647	0.036242	0.292695
DiabetesPedigreeFunction	-0.033523	0.137337	0.041265	0.183928	0.185071	0.140647	1.000000	0.033561	0.173844
Age	0.544341	0.263514	0.239528	-0.113970	-0.042163	0.036242	0.033561	1.000000	0.238356
Outcome	0.221898	0.466581	0.065068	0.074752	0.130548	0.292695	0.173844	0.238356	1.000000

Можно увидеть, что пропуски в данных отсутствуют.

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

Категориальных признаков в датасете нет, их кодирования не требуется.

Вспомогательные признаки для улучшения качества моделей строить не будем.

Построим графики для понимания структуры данных:

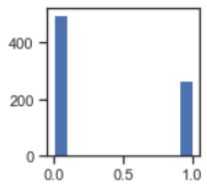
```
sns.pairplot(data, hue=targetColumn)
```

```
<seaborn.axisgrid.PairGrid at 0x1188950d0>
```


```
data[targetColumn].unique()
```

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

```
# Оценим дисбаланс классов для Occupancy
fig, ax = plt.subplots(figsize=(2,2))
plt.hist(data[targetColumn])
plt.show()
```




```
data[targetColumn].value_counts()
```

```
0    500
1    268
Name: Outcome, dtype: int64
```

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

```
Класс 0 составляет 65.10000000000001%, а класс 1 составляет 34.9%.
```

Можно заметить, что дисбаланс классов в данном случае присутствует, но является допустимым.

Выполним масштабирование данных.

```
# Числовые колонки для масштабирования
scale_cols = list(data.columns.drop(targetColumn))
```

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

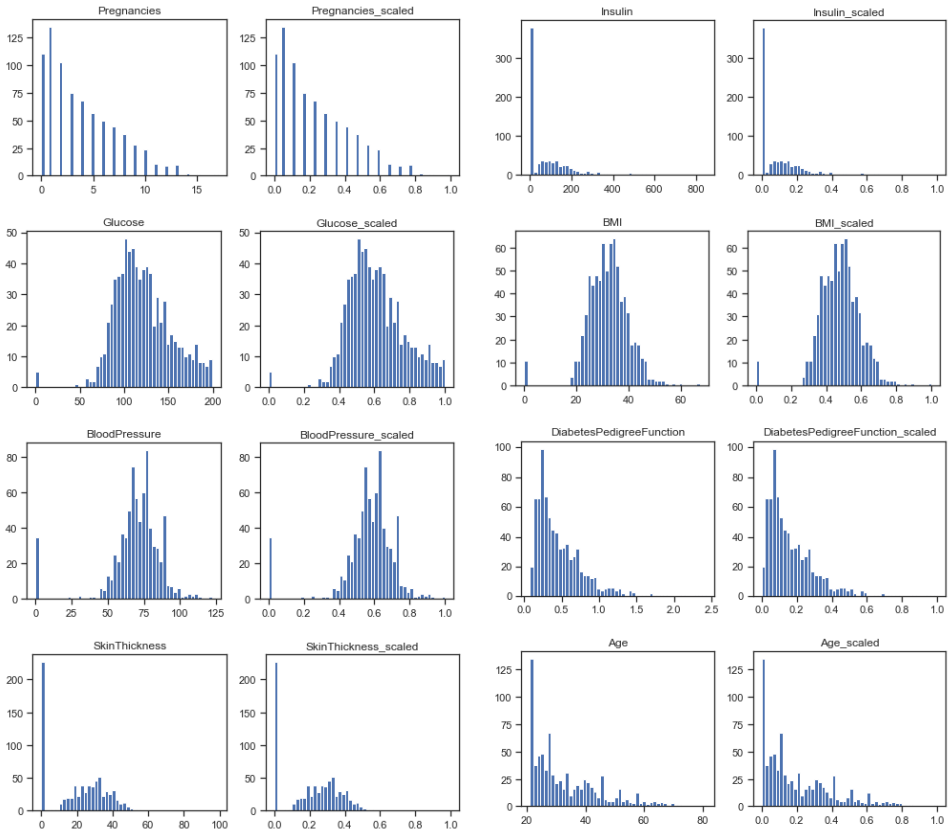
```
data_all = data.copy()
```

```
# 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()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome	Pregnancies_scaled	Glucose_scaled	BloodPr
0	6	148	72	35	0	33.6	0.627	50	1	0.352941	0.743719	
1	1	85	66	29	0	26.6	0.351	31	0	0.058824	0.427136	
2	8	183	64	0	0	23.3	0.672	32	1	0.470588	0.919598	
3	1	89	66	23	94	28.1	0.167	21	0	0.058824	0.447236	
4	0	137	40	35	168	43.1	2.288	33	1	0.000000	0.688442	

```
# 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()
```



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

```
corr_cols_1 = scale_cols + [targetColumn]
corr_cols_1
```

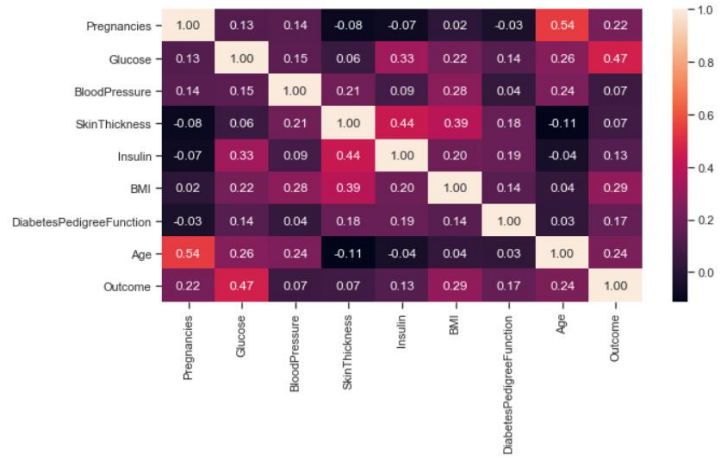
```
['Pregnancies',
 'Glucose',
 'BloodPressure',
 'SkinThickness',
 'Insulin',
 'BMI',
 'DiabetesPedigreeFunction',
 'Age',
 'Outcome']
```

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

```
['Pregnancies_scaled',
 'Glucose_scaled',
 'BloodPressure_scaled',
 'SkinThickness_scaled',
 'Insulin_scaled',
 'BMI_scaled',
 'DiabetesPedigreeFunction_scaled',
 'Age_scaled',
 'Outcome']
```

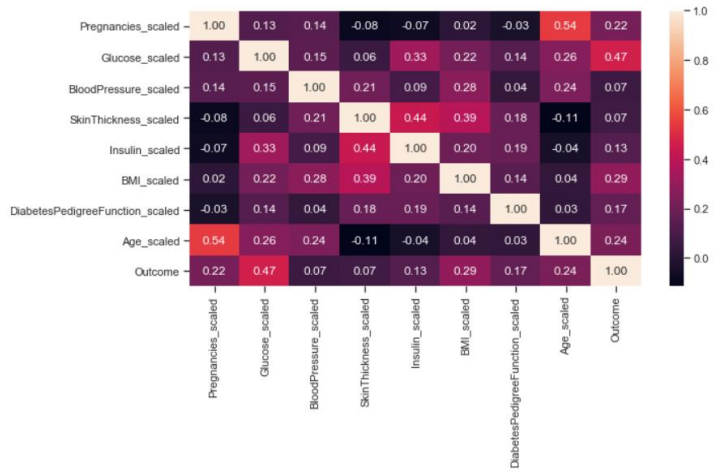
```
fig, ax = plt.subplots(figsize=(10,5))
sns.heatmap(data_all[corr_cols_1].corr(), annot=True, fmt='.2f')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x11b2b1cd0>
```




```
fig, ax = plt.subplots(figsize=(10,5))
sns.heatmap(data_all[corr_cols_2].corr(), annot=True, fmt='.2f')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x11cc8fb80>
```



На основе корреляционной матрицы можно сделать следующие выводы:

- ✓ Корреляционные матрицы для исходных и масштабированных данных совпадают.
- ✓ Целевой признак классификации "Outcome" наиболее сильно коррелирует с уровнем глюкозы (0.47), индексом массы тела (0.29), возрастом (0.24) и количества беременностей (0.22). Эти признаки обязательно следует оставить в модели классификации.

4.5. Выбор метрик для последующей оценки качества моделей.

В качестве метрик для решения задачи классификации будем использовать:

1. Метрика precision:

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

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

Используется функция *precision_score.*

2. Метрика recall (полнота):

$$recall = \frac{TP}{TP+FN}$$

Доля верно предсказанных классификатором положительных объектов, из всех действительно положительных объектов.

Используется функция *recall_score*.

3. Метрика F1-мера

Для того, чтобы объединить precision и recall в единую метрику используется F_β-мера, которая вычисляется как среднее гармоническое от precision и recall:

$$F_\beta = (1 + \beta^2) \cdot \frac{precision \cdot recall}{precision + recall}$$

где β определяет вес точности в метрике.

На практике чаще всего используют вариант F1-меры (которую часто называют F-мерой) при β=1:

$$F_1 = 2 \cdot \frac{precision \cdot recall}{precision + recall}$$

Для вычисления используется функция *f1_score*.

4. ***Метрика ROC AUC***

Используется для оценки качества бинарной классификации. Основана на вычислении следующих характеристик:

$$TPR = \frac{TP}{TP+FN}$$

$$FPR = \frac{FP}{FP+TN}$$

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.

Чем сильнее отклоняется кривая от верхнего левого угла графика, тем хуже качество классификации.

В качестве количественной метрики используется площадь под кривой - ROC AUC (Area Under the Receiver Operating Characteristic Curve). Чем ниже проходит кривая тем меньше ее площадь и тем хуже качество классификатора.

Для получения ROC AUC используется функция *roc_auc_score*.

Сохранение и визуализация метрик

Разработаем класс, который позволит сохранять метрики качества построенных моделей и реализует визуализацию метрик качества.

```
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()
```

4.6. *Выбор наиболее подходящих моделей для решения задачи классификации или регрессии.*

Для задачи классификации будем использовать следующие модели:

- Логистическая регрессия

Метод, используемый для решения задачи бинарной классификации.

Метод выдает вероятность принадлежности объекта к нулевому/единичному классам.

Используется класс *LogisticRegression*.

- Машина опорных векторов

Основная идея метода — перевод исходных векторов в пространство более высокой размерности и поиск разделяющей гиперплоскости с максимальным зазором в этом пространстве. Две параллельных гиперплоскости строятся по обеим сторонам

гиперплоскости, разделяющей классы. Разделяющей гиперплоскостью будет гиперплоскость, максимизирующая расстояние до двух параллельных гиперплоскостей. Алгоритм работает в предположении, что чем больше разница или расстояние между этими параллельными гиперплоскостями, тем меньше будет средняя ошибка классификатора.

Для решения задачи классификации используем класс:

SVC - основной классификатор на основе SVM. Поддерживает различные ядра.

- Решающее дерево

Для текущего выбранного признака (колонки) из N признаков построить все варианты ветвления по значениям (для категориальных признаков) или по диапазонам значений (для числовых признаков).

Если подвыборке соответствует единственное значение целевого признака, то в дерево добавляется терминальный лист, который соответствует предсказанному значению.

Если в подвыборке больше одного значения целевого признака, то предыдущие пункты выполняются рекурсивно для подвыборки.

Для решения задачи классификации используется класс *DecisionTreeClassifier*.

- Случайный лес (ансамблевая)

Случайный лес можно рассматривать как алгоритмом бэггинга над решающими деревьями.

Но при этом каждое решающее дерево строится на случайно выбранном подмножестве признаков. Эта особенность называется "feature bagging" и основана на методе случайных подпространств.

Случайный лес для задача классификации реализуется в scikit-learn с помощью класса *RandomForestClassifier*.

Задание параметра n_jobs=-1 распараллеливает алгоритм на максимально возможное количество процессоров.

- Градиентный бустинг (ансамблевая)

В отличие от методов бэггинга и случайного леса, которые ориентированы прежде всего на минимизизацию дисперсии (Variance), методы бустинга ориентированы прежде всего на минимизацию смещения (Bias) и, отчасти, на минимизизацию дисперсии.

Исторически первым полноценным алгоритмом бустинга считается алгоритм AdaBoost.

AdaBoost реализуется в scikit-learn с помощью класса *AdaBoostClassifier* для задач классификации.

4.7. Формирование обучающей и тестовой выборок на основе исходного набора данных.

Возьмем наши масштабированные данные и выделим обучающую и тестовую. Процент тестовой выборки составит 33% от общего объема данных.

```
task_clas_cols = ['Glucose_scaled', 'BMI_scaled', 'Age_scaled', 'Pregnancies_scaled']
```

```
clas_X_train, clas_X_test, clas_Y_train, clas_Y_test = train_test_split(data_all[task_clas_cols], data_all["Outcome"], test_size
=0.33, random_state=42)
clas_X_train.shape, clas_Y_train.shape, clas_X_test.shape, clas_Y_test.shape
```

```
((514, 4), (514,), (254, 4), (254,))
```

4.8. 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.

```
# 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='darkorange',
             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()
```

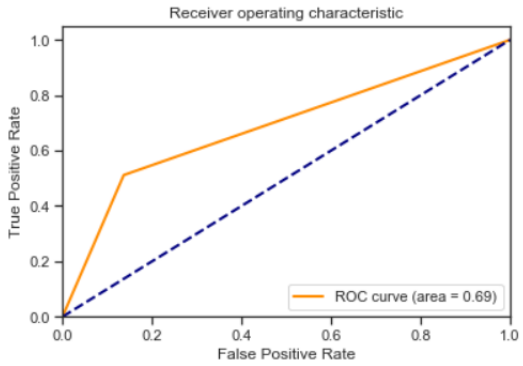
```
# Модели
clas_models = {'LogR': LogisticRegression(),
               'KNN_5':KNeighborsClassifier(n_neighbors=5),
               'SVC':SVC(),
               'Tree':DecisionTreeClassifier(),
               'RF':RandomForestClassifier(),
               'GB':GradientBoostingClassifier()}
```

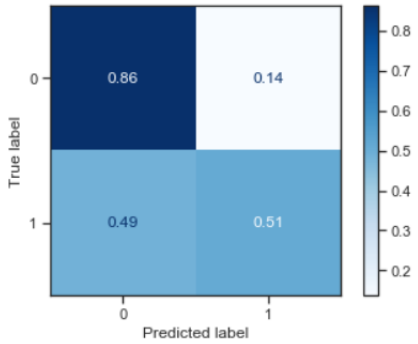
```
# Сохранение метрик
clasMetricLogger = MetricLogger()
```

```
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)
    recall = recall_score(clas_Y_test.values, Y_pred)
    f1 = f1_score(clas_Y_test.values, Y_pred)
    roc_auc = roc_auc_score(clas_Y_test.values, Y_pred)

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

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

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

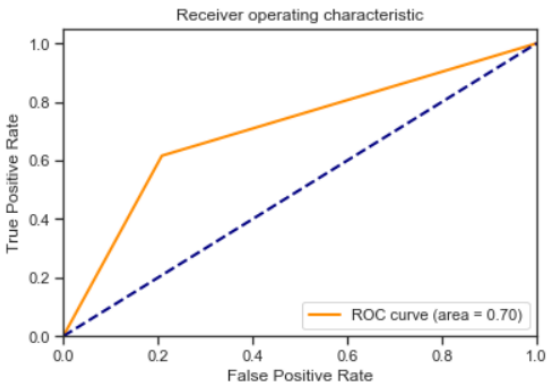
```
for model_name, model in clas_models.items():
    clas_train_model(model_name, model, clasMetricLogger)
```

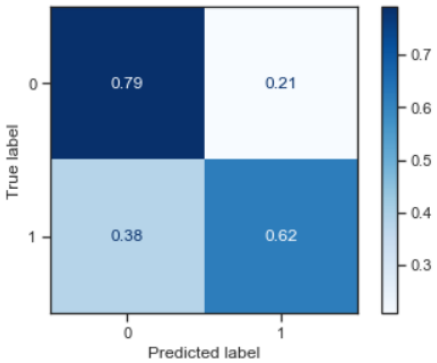
```
**************************************************
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
**************************************************
```



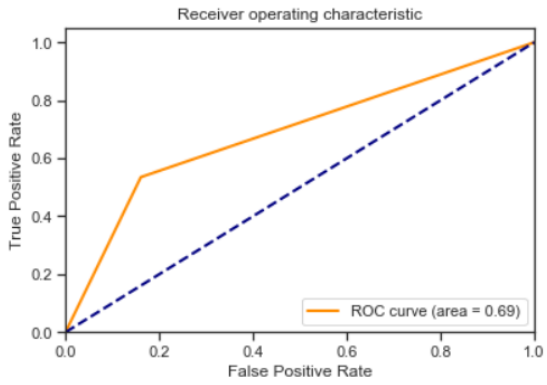


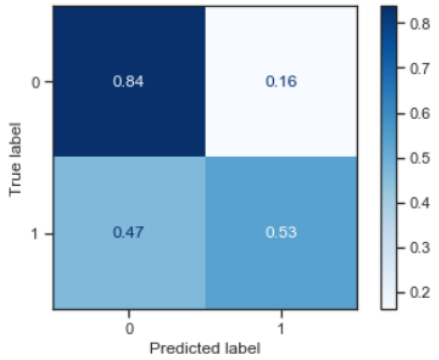

```
**************************************************
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                     weights='uniform')
**************************************************
```



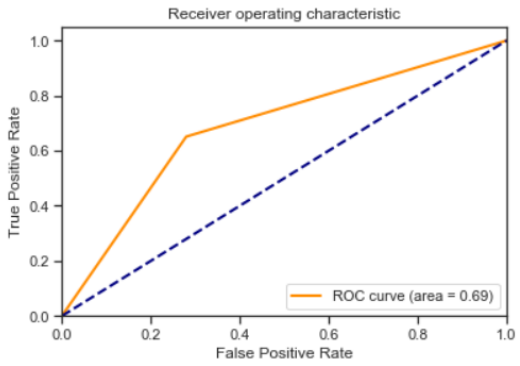


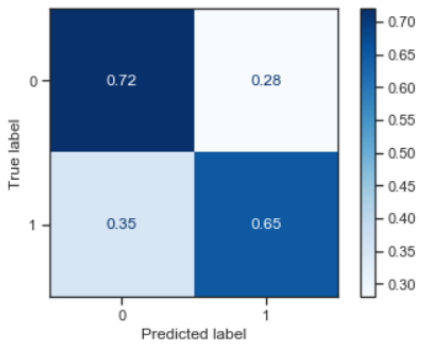
```
**************************************************
SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='scale', kernel='rbf',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
**************************************************
```



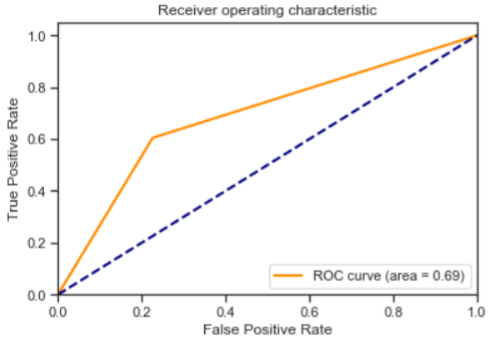


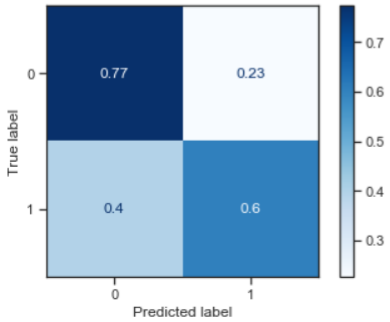

```
**************************************************
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=None, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=None, splitter='best')
**************************************************
```



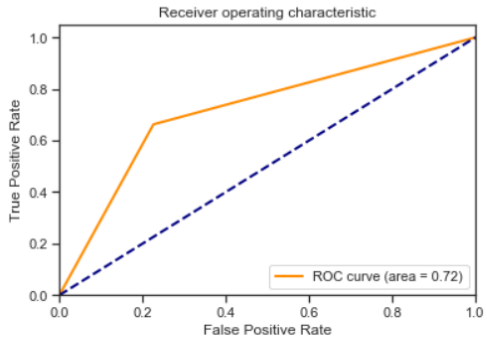


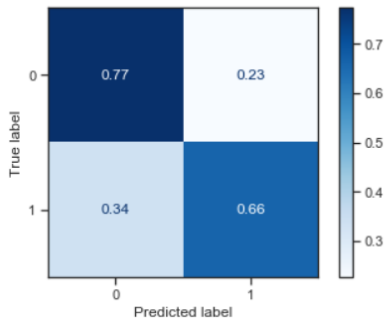
```
**************************************************
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=None, max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, 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)
**************************************************
```






```
**************************************************
GradientBoostingClassifier(ccp_alpha=0.0, criterion='friedman_mse', init=None,
                           learning_rate=0.1, loss='deviance', max_depth=3,
                           max_features=None, max_leaf_nodes=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_iter_no_change=None, presort='deprecated',
                           random_state=None, subsample=1.0, tol=0.0001,
                           validation_fraction=0.1, verbose=0,
                           warm_start=False)
**************************************************
```





4.9. *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.*

```
n_range = np.array(range(1,200,10))
tuned_parameters = [{'n_neighbors': n_range}]
tuned_parameters
```

```
[{'n_neighbors': array([  1,  11,  21,  31,  41,  51,  61,  71,  81,  91, 101, 111, 121,
        131, 141, 151, 161, 171, 181, 191])}]
```

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

```
CPU times: user 503 ms, sys: 5.9 ms, total: 509 ms
Wall time: 509 ms

GridSearchCV(cv=5, error_score=nan,
             estimator=KNeighborsClassifier(algorithm='auto', leaf_size=30,
                                            metric='minkowski',
                                            metric_params=None, n_jobs=None,
                                            n_neighbors=5, p=2,
                                            weights='uniform'),
             iid='deprecated', n_jobs=None,
             param_grid=[{'n_neighbors': array([  1,  11,  21,  31,  41,  51,  61,  71,  81,  91, 101, 111, 121,
       131, 141, 151, 161, 171, 181, 191])}],
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring='roc_auc', verbose=0)
```

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

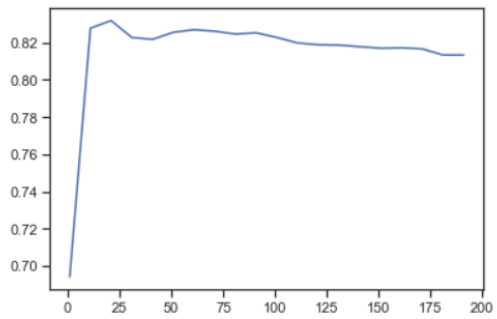
```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=21, p=2,
                     weights='uniform')
```

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

```
{'n_neighbors': 21}
```

```
# 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 0x11b8079d0>]
```

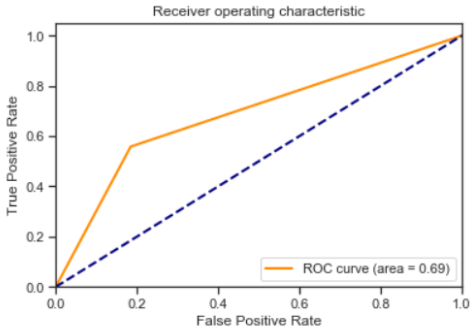


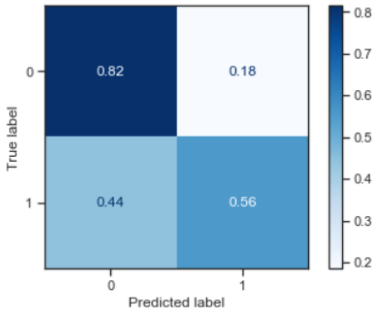
4.10. *Повторение пункта 8 для найденных оптимальных значений гиперпараметров. Сравнение качества полученных моделей с качеством baseline-моделей.*

```
clas_models_grid = {'KNN_21':clf_gs.best_estimator_}
```

```
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=21, p=2,
                     weights='uniform')
******************************************************
```



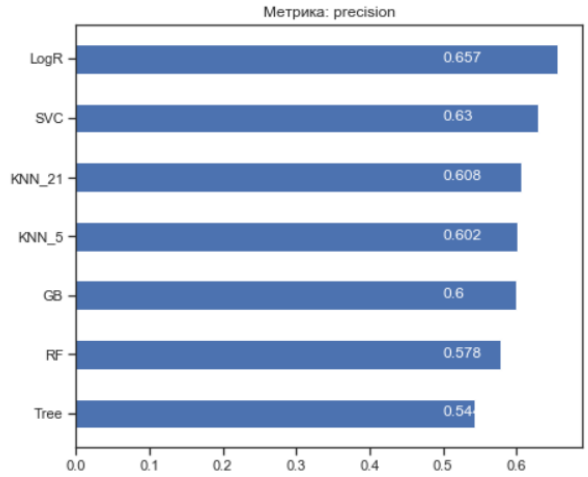


4.11. *Формирование выводов о качестве построенных моделей на основе выбранных метрик. Результаты сравнения качества рекомендуется отобразить в виде графиков и сделать выводы в форме текстового описания.*

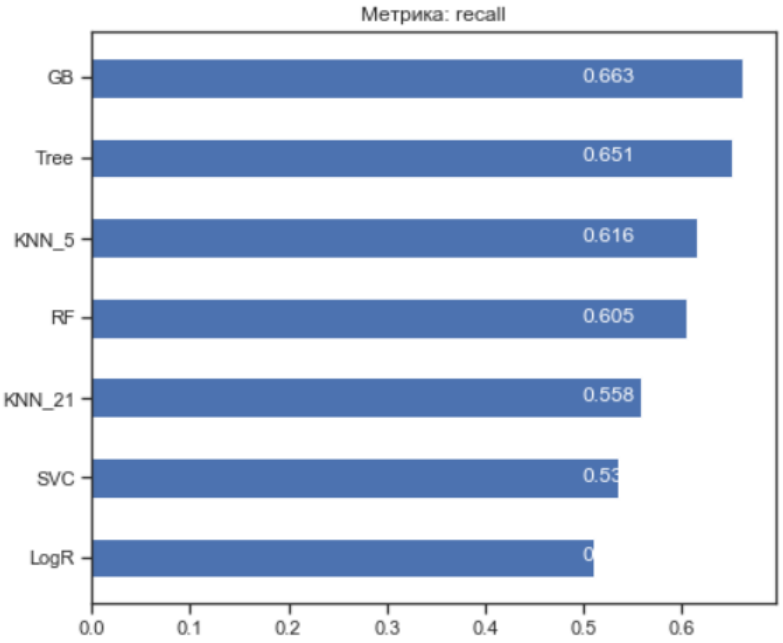
```
# Метрики качества модели
clas_metrics = clasMetricLogger.df['metric'].unique()
clas_metrics
```

```
array(['precision', 'recall', 'f1', 'roc_auc'], dtype=object)
```

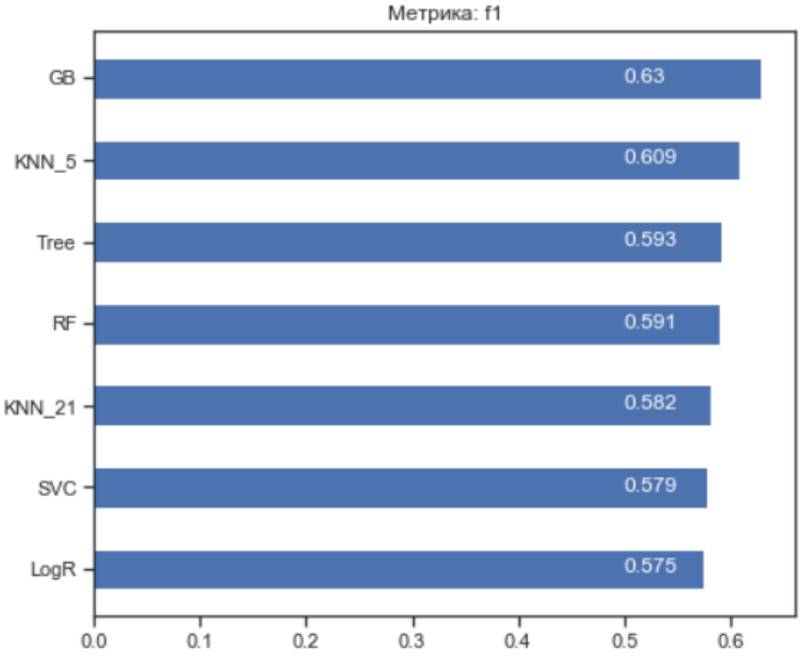
```
# Построим графики метрик качества модели
for metric in clas_metrics:
    clasMetricLogger.plot('Метрика: ' + metric, metric, figsize=(7, 6))
```



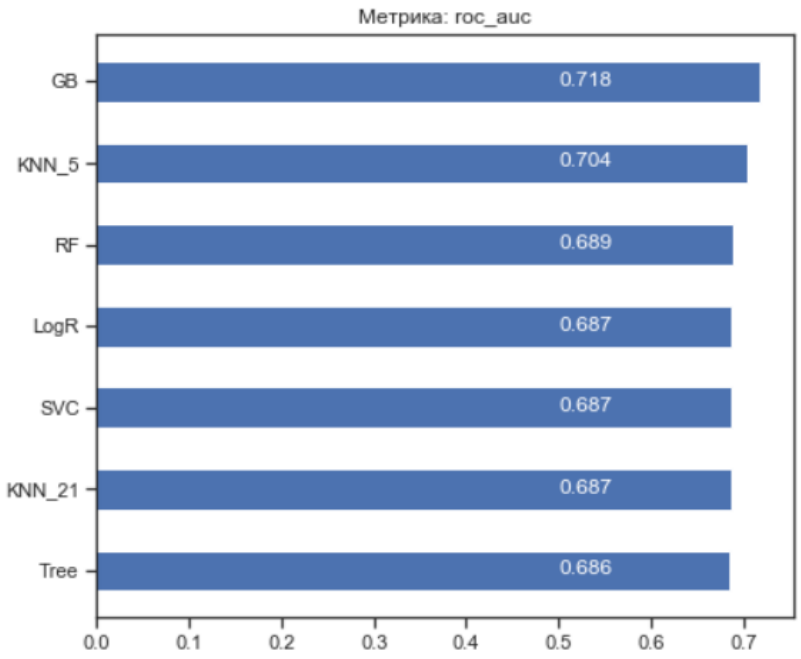
По метрике precision лучшими моделями являются: *Логистическая регрессия, Машина опорных векторов.*



По метрике recall лучшими моделями являются: Градиентный спуск *и случайный лес.*



По метрике f1 лучшими моделями являются: *Градиентный спуск и метод ближайших соседей.*



По метрике ROC AUC лучшими моделями также являются: *Градиентный спуск и метод ближайших соседей.*

Вывод: на основании трех метрик из четырех используемых, лучшими оказалась модель *Градиентный спуск.*

5. Заключение

Из всех рассмотренных алгоритмов: "Logistic Regression", "Support vector machine", "Decision tree", "Gradient boosting", "Random forest" для модели классификации пациентов по заболеванию диабет наиболее эффективным оказался алгоритм градиентного спуска, т.е. "Gradient boosting". Как известно Gradient boosting направлен на обнаружение сложных связей между атрибутами, с чем плохо справляются остальные методы.

6. Список литературы

1. Репозиторий курса "Технологии машинного обучения", бакалавриат, 6 семестр. Лекции по теории машинного обучения. Ю.Е. Гапанюк [Электронный ресурс]. – Электрон. дан. - URL: https://github.com/ugapanyuk/ml_course_2020/wiki/COURSE_TMO (дата обращения: 03.06.2020)
2. Pima Indians Diabetes Dataset [Электронный ресурс]. – Электрон. дан. - URL: https://www.kaggle.com/salihacur/diabetes (дата обращения: 03.05.2020)
3. Машинное обучение (часть 1). А.М.Миронов [Электронный ресурс]. – Электрон. дан. - URL: http://www.intsys.msu.ru/staff/mironov/machine_learning_vol1.pdf (дата обращения: 03.05.2020)
4. Scikit learn[Электронный ресурс]. – Электрон. дан. - URL: https://scikit-learn.org/stable/index.html (дата обращения: 03.05.2020)