Московский авиационный институт (национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №1 по курсу «Искусственный интеллект»
Тема: Линейные модели

Студент: А. В. Тимофеев
Преподаватель: Самир Ахмед
Группа: М8О-407Б-19

Дата:
Оценка:
Подпись:

Москва, 2022

Задача

Вы собрали данные и их проанализировали, визуализировали и представили отчет своим партнерам и спонсорам. Они согласились, что ваша задача имеет перспективу и продемонстрировали заинтересованность в вашем проекте. Самое время реализовать прототип! Вы считаете, что нейронные сети переоценены (просто боитесь признаться, что у вас не хватает ресурсов и данных), и считаете что за машинным обучением классическим будущее и потому собираетесь использовать классические модели. Вашим первым предположением является предположение, что данные и все в этом мире имеет линейную зависимость, ведь не зря же в конце каждой нейронной сети есть линейный слой классификации. В качестве первых моделей вы выбрали, линейную / логистическую регрессию и SVM. Так как вы очень осторожны и боитесь ошибиться, вы хотите реализовать случай, когда все таки мы не делаем никаких предположений о данных, и взяли за основу идею "близкие объекты дают близкий ответ"и идею, что теорема Байеса имеет ранг королевской теоремы. Так как вы не доверяете другим людям, вы хотите реализовать алгоритмы сами с нуля без использования scikit-learn (почти). Вы хотите узнать насколько хорошо ваши модели работают на выбранных вам данных и хотите замерить метрики качества. Ведь вам нужно еще отчитаться спонсорам!

1 Описание

В качестве датасета я выбрал «Heart Attack Analysis & Prediction Dataset» с сайта kaggle.
Он находится по ссылке https://www.kaggle.com/datasets/rashikrahmanpritom/heart-attack-analysis-prediction-dataset?select=heart.csv.
В данном датасете собраны некоторые признаки, влияющие на возникновение сердечного приступ.

В этом наборе данных приведены признаки:

1. Age : Возраст пациента
2. Sex: Пол пациента
3. exang: стенокардия, вызванная физической нагрузкой (1 = да; 0 = нет)
4. cp: тип боли в груди
   - 4.1 Value 1: типичная стенокардия
   - 4.2 Value 2: атипичная стенокардия
   - 4.3 Value 3: неангинальная боль
   - 4.4 Value 4: бессимптомное течение
5. trtbps: артериальное давление в состоянии покоя (в мм рт. ст.)
6. chol: холесторал в мг / дл, определяемый с помощью датчика ИМТ
7. fbs: (уровень сахара в крови натощак> 120 мг / дл) (1 = истина; 0 = ложь)
8. rest_ecg : результаты электрокардиографии в состоянии покоя
   - 8.1 Value 0: нормальное
   - 8.2 Value 1: аномалия зубца ST-T (инверсия зубца T и / или повышение или понижение ST> 0,05 мВ)
   - 8.3 Value 2: отображение вероятной или определенной гипертрофии левого желудочка по критериям Эстеса
9. thalach: достигнутая максимальная частота сердечных сокращений
10. target : 0 = меньше шансов сердечного приступа 1 = больше шансов сердечного приступа

В данной лабораторной работе реализованы следующие алгоритмы обучения:

1) k-Nearest Neighbors (KNN) Идея заключается в определении класса объекта по классам k ближайших(каких больше - такой и класс).

2) Naive Bayes Построен на формуле Байеса $P(A|B) = \frac{P(B|A)P(A)}{P(B)}$

3)Linear / Logistic Regression Попытка провести разделяющую гиперплоскость между классами

4)SVM Линейная с дополнительным условием: максимизируется расстояние от объектов до гиперплоскости

2 Ход работы

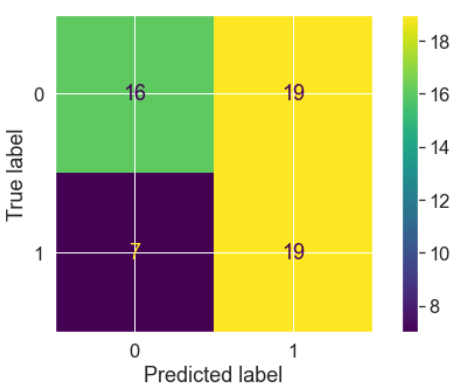
KNN

Проведем те же манипуляции с датасетом что и в лабораторной работе № 0. Далее начнем обучать модели.

Сначала обучим модель KNN. Её код приведен ниже.

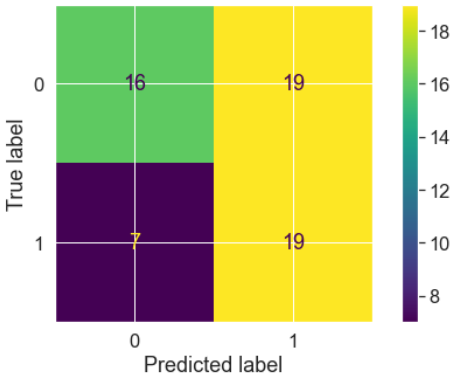
```
class KNN(BaseEstimator, ClassifierMixin):
    def __init__(self, k):
        self.k = k

    def fit(self, data, labels):
        self.data = data
        self.labels = labels

    def euclidean_distance(self, data, row):
        distance = 0
        for i in range(len(data)):
            distance += (data[i] - row[i]) ** 2
        return math.sqrt(distance)

    def predict(self, dataX):
        res = np.ndarray((dataX.shape[0],))
        for j, data in enumerate(dataX):
            distances = []
            for i, row in enumerate(self.data):
                distances.append((self.euclidean_distance(data, row), self.labels[i]))
            distances.sort(key = lambda tup: tup[0])
            dictionary = collections.defaultdict(int)
            for i in range(self.k):
                dictionary[distances[i][1]] += 1
            res[j] = max(dictionary.items(), key = lambda tup: tup[1])[0]
        return res
```

```
{'knn__k': 3}
Accuracy train: 0.6697278911564626
Accuracy: 0.5737704918032787
Recall: 0.7307692307692307
Precision: 0.5
```



Все классы наследуем от BaseEstimator и ClassifierMixin. Соответственно реализовано две основные функции: fit, обучающая модель на тренировочных данных, в этом алгоритме данная функция только сохроняет данные, и predict, которая уже непосредственно выдает результат для тестовых данных. В качестве меры используем классическое расстояние Евклида. Из результатов видно, что точность крайне низкая, вероятно, потому что точки на графике находятся одним облаком и их сложно отличить друг от друга данным алгоритмом.

```
{'knn__n_neighbors': 3}
Accuracy train: 0.6697278911564626
Accuracy: 0.5737704918032787
Recall: 0.7307692307692307
Precision: 0.5
```

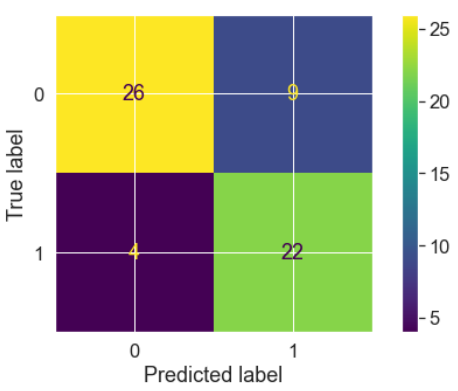


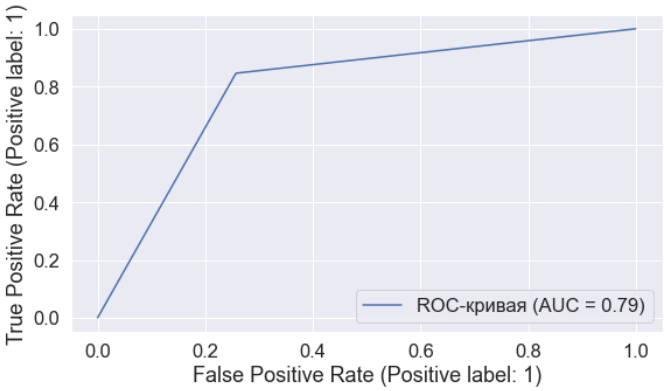
Точность, полученная с помощью коробочного решения, равна точности, полученной с помощью моего решения. Значит проблема действительно а датасете.

NaiveBayes

```
class NaiveBayes(BaseEstimator, ClassifierMixin):
    def __init__(self, bins):
        self.bins = bins
        pass

    def fit(self, data, labels):
        self.data = data
        self.labels = labels
        self.classes = []
        for j in np.unique(labels):

            self.classes.append([])
            for i in range (data.shape[1]):
                self.classes[j].append([*np.histogram(data[labels == j, i], bins = self.
                    bins)])
                self.classes[j][-1][0] = self.classes[j][-1][0].astype('float64') / len(
                    data[labels == j, i])

        self.prclasses = np.unique(labels, return_counts = True)[1] / len(labels)

    def predict(self, maindata):
        res = np.ndarray((maindata.shape[0],))
        for j, data in enumerate(maindata):
            maximum = 0
            ans = 0
            for i in range(len(self.classes)):
                p = self.prclasses[i]
                for k in range(len(self.classes[i])):
                    ind = np.digitize(data[k], self.classes[i][k][1])

                    if ind >= len(self.classes[i][k][1]) or ind <= 0:
                        p = 0
                    else:
                        p *= self.classes[i][k][0][ind - 1]

                if p > maximum:
                    maximum = p
                    ans = i
            res[j] = ans
        return res
```

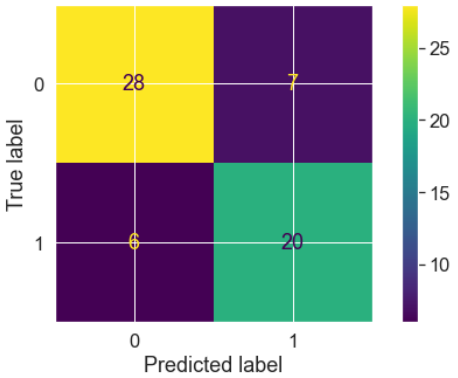
```
{'bn__bins': 8}
Accuracy train: 0.7812925170068027
Accuracy: 0.7868852459016393
Recall: 0.8461538461538461
Precision: 0.7096774193548387
```

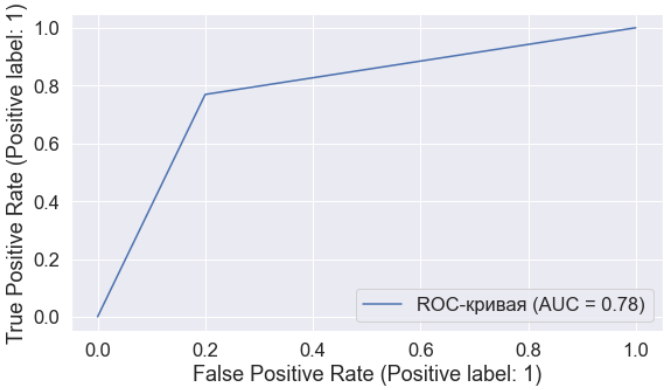




Приемлимую точность удалось получить при помощи алгоритма Байесовского классификатора. Из тепловой карты видно, что многие велечины почти не коррелируют друг с другом, поэтому наивное предположение об условной независимости между каждой парой характеристик при заданном значении переменной класса вместе с методом Байеса дает высокую точность. Также распределение признаков похоже на нормальное, это тоже дает плюс Байесовскому классификатору.

```
Accuracy: 0.7868852459016393
Recall: 0.7692307692307693
Precision: 0.7407407407407407
```





Примерно такой же точности достигает коробочное решение.

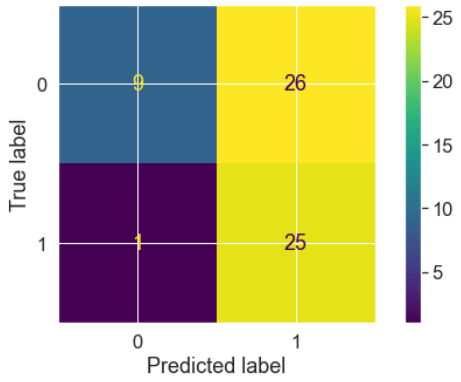
Logistic Regression

```
1 class Logistic(BaseEstimator, ClassifierMixin):
2     def __init__(self, lr, nepoch, batch_size):
3         self.lr = lr
4         self.nepoch = nepoch
```

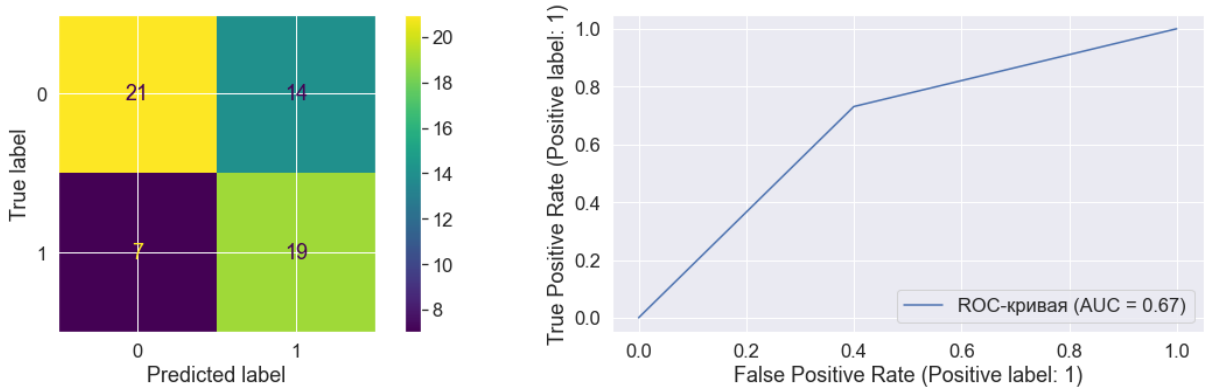
```
        self.batch_size = batch_size
        pass

    def sigmoid(self, x):
        self.l = 1 / (1 + np.exp(-x))
        return self.l

    def fit(self, data, labels):
        data = np.concatenate((data, np.ones((data.shape[0],1))), axis = 1)
        self.W = np.random.normal(0, 1, (len(data[0]),))

        for i in range(self.nepoch):
            for i in range(0, len(data), self.batch_size):
                xb = data[i:i + self.batch_size]
                yb = labels[i:i + self.batch_size]
                p = np.dot(self.W, xb.T)
                s = self.sigmoid(p)
                dp = np.dot(xb.T, (s - yb).T)
                self.W -= self.lr * dp

    def predict(self, maindata):
        maindata = np.concatenate((maindata, np.ones((maindata.shape[0],1))), axis = 1)
        p = np.dot(self.W, maindata.T)
        s = self.sigmoid(p)
        return (s > 0.5).astype('int64')
```

В данном методе на вход подается 3 параметра: размер батча(batch$_size$) – , , (lr) – ($nepoch$) – . – $W, ., Wx + b, Wx1, b.logloss.sigmoid$.

```
{'log__batch_size': 1,'log__lr': 0.1,'log__nepoch': 20}
Accuracy train: 0.6282312925170068
Accuracy: 0.5573770491803278
Recall: 0.9615384615384616
Precision: 0.49019607843137253
```



```
{'log__alpha': 0.001,'log__max_iter': 100}
Accuracy train: 0.6656462585034014
Accuracy: 0.6557377049180327
Recall: 0.7307692307692307
Precision: 0.5757575757575758
```



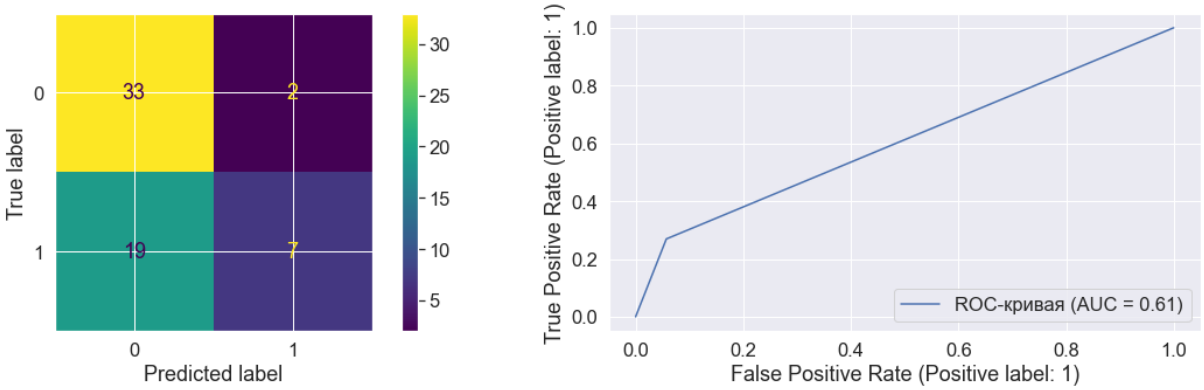
Точность снова низкая у обоих вариантов реализации. Это происходит из-за того что данные плохо разделяются линией.

SVM

```
class SVM(BaseEstimator, ClassifierMixin):
    def __init__(self, lr, lambd, batch_size, nepoch):
        self.nepoch = nepoch
        self.lr = lr
        self.lambd = lambd
        self.batch_size = batch_size

    def fit(self, data, labels):
        data = np.concatenate((data, np.ones((data.shape[0],1))), axis=1)
        self.W = np.random.normal(0, 1, (len(data[0]),))

        for i in range(self.nepoch):
            for i in range(0, len(data), self.batch_size):
                xb = data[i:i + self.batch_size]
                yb = labels[i:i + self.batch_size]

                p = np.dot(self.W, xb.T)

                sums = np.zeros_like(self.W)
                for i in range(len(p)):
                    if 1 - p[i] * yb[i] > 0:
                        sums -= xb[i] * yb[i]
```

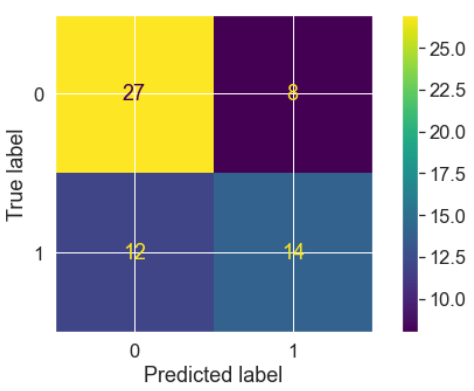
```
23
24                dp = 2 * self.lambd * self.W + sums
25                self.W -= self.lr * dp
26
27
28        def predict(self, maindata):
29            maindata = np.concatenate((maindata, np.ones((maindata.shape[0],1))), axis=1)
30            p = np.dot(self.W, maindata.T)
31            return np.sign(p)
```

```
{'lin__batch_size': 5,'lin__lambd': 0.001,'lin__lr': 0.5,'lin__nepoch': 10}
Accuracy train: 0.6568027210884353
Accuracy: 0.6557377049180327
Recall: 0.2692307692307692
Precision: 0.7777777777777778
```



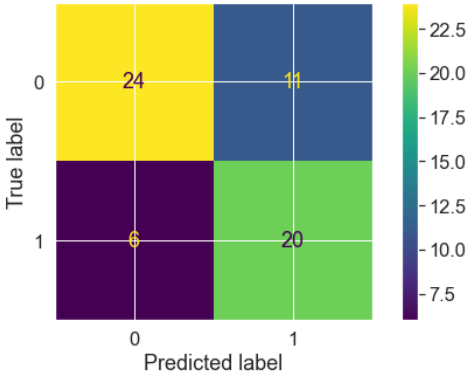
Та же регрессия только с добавлением параметра lambda.

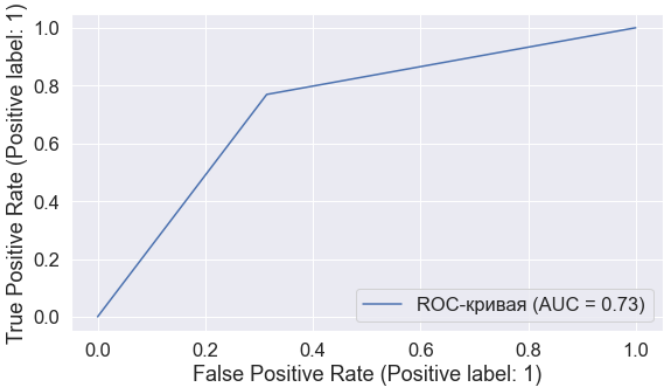
```
{'lin__alpha': 0.0001,'lin__max_iter': 1000}
Accuracy train: 0.702295918367347
Accuracy: 0.6721311475409836
Recall: 0.5384615384615384
Precision: 0.6363636363636364
```



После таких плохих результатов работы линейных моделей я попытался преобразовать данные. Точность увеличилась после добавления произведения двух лучше всего разделяемых параметров: thalachh и oldpeak и бинаризации категориальных параметров датасета.

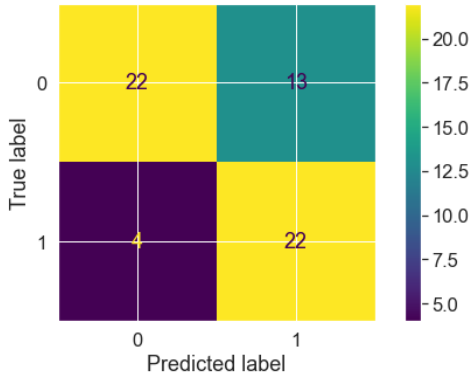
Logistic Regression





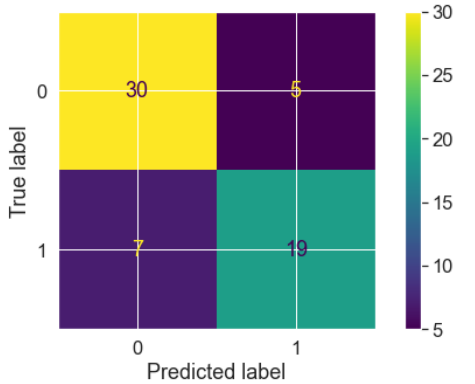

```
{'log__batch_size': 1,'log__lr': 0.1,'log__nepoch': 20}
Accuracy train: 0.7147959183673469
Accuracy: 0.7213114754098361
Recall: 0.7692307692307693
Precision: 0.6451612903225806
```

KNN



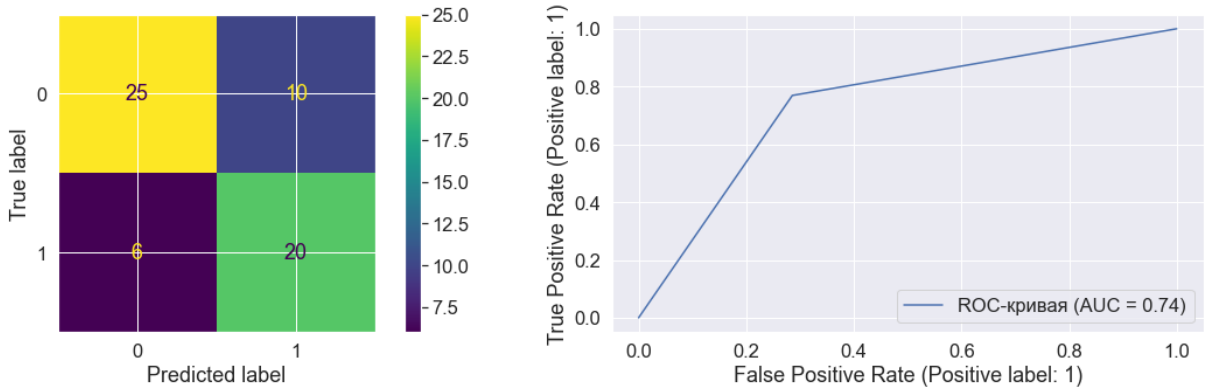

```
{'knn__k': 22}
Accuracy train: 0.7023809523809523
Accuracy: 0.7213114754098361
Recall: 0.8461538461538461
Precision: 0.6285714285714286
```

NaiveBayes




```
{'bn__bins': 2}
Accuracy train: 0.7894557823129251
Accuracy: 0.8032786885245902
Recall: 0.7307692307692307
Precision: 0.7916666666666666
```

SVM




```
{'lin__batch_size': 10,'lin__lambd': 0,'lin__lr': 0.1,'lin__nepoch': 20}
Accuracy train: 0.7272108843537415
Accuracy: 0.7377049180327869
Recall: 0.7692307692307693
Precision: 0.6666666666666666
```

3 Выводы

Данная лабораторная работа дала мне интересный опыт в работе с настоящими данными. В качестве темы я взял медицину, так как для этой область, часто испоьзуется искусственный интеллект. Реализовав все эти алгоритмы, я получил в лучшем случае точность примерно 80%, что вполне неплохо. Конечно, доля ошибки достаточно велика, но лучше лишний раз пройти обследование в поликлинике и не подтвердить результат выданный алгоритмом. Точность удалось увеличить во всех алгоритмах кроме Байесовский классификатор, когда я произвел дополнительные манипуляции над датасетом. Возможно есть еще методы при которых получилось бы увеличить точность, но мне они неизвестны.