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: К. М. Воронов
Преподаватель: Самир Ахмед
Группа: М8О-307Б-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 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 $\frac{P(A|B)=P(B|A)*P(A)}{P(B)}$

3)Linear/ Logistic Regression

Попытка провести разделяющую гиперплоскость между классами

4)SVM

Линейная с дополнительным условием: максимизируется расстояние от объектов до гиперплоскости

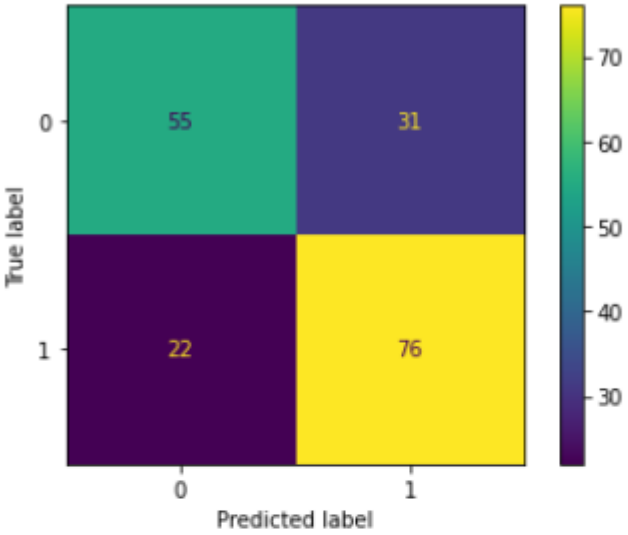
2 Ход выполнения

Я начал с реализации KNN. Выглядит это следующим образом:

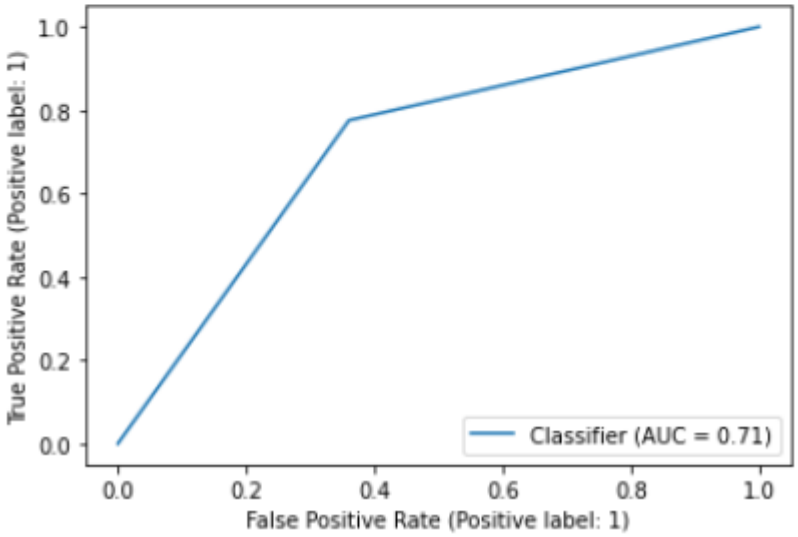
```
from sklearn.base import BaseEstimator, ClassifierMixin

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, row1, row2):
        distance = 0
        for i in range(len(row1)):
            distance += (row1[i] - row2[i]) ** 2
        return math.sqrt(distance)

    def predict(self, maindata):
        res = np.ndarray((maindata.shape[0],))
        for j, data in enumerate(maindata):
            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
```

Класс наследован от BaseEstimator и ClassifierMixin (как и все последующие). Соответственно реализовано две основные функции: fit, которая должна обучать модель на тренировочных данных (в этом алгоритме обучение не нужно, поэтому тут данные просто сохраняются), и predict, которая уже непосредственно выдает результат для тестовых данных. В качестве меры я использовал классическое расстояние Евклида. Результаты получились следующие:

```
{'knn__k': 20}
Accuracy train: 0.7111173236417855
Accuracy tests: 0.7119565217391305
```




```
Precision tests: 0.7102803738317757
Recall tests: 0.7755102040816326
```



Наилучшая точность достигается при k = 20 и составляет около 71% правильных ответов. (Для этого алгоритма позже была достигнута еще более высокая точность, но об этом позднее). При тестировании, как и везде далее, использовались pipeline и Кросс-валидация. KNN из sklearn дал практически такие же результаты при k = 22.

Далее я перешел к Naive Bayes и реализовал его двумя способами:

1)По гистограммам

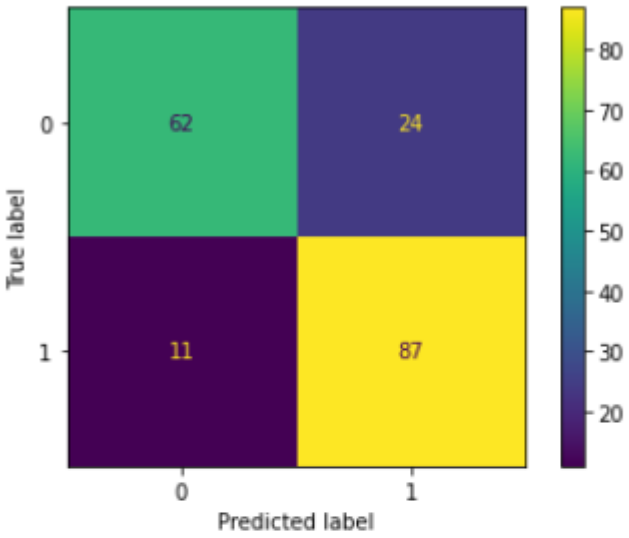
2)Делая предположение, что данные имеют нормальное распределение

Первая реализация выглядит следующим образом:

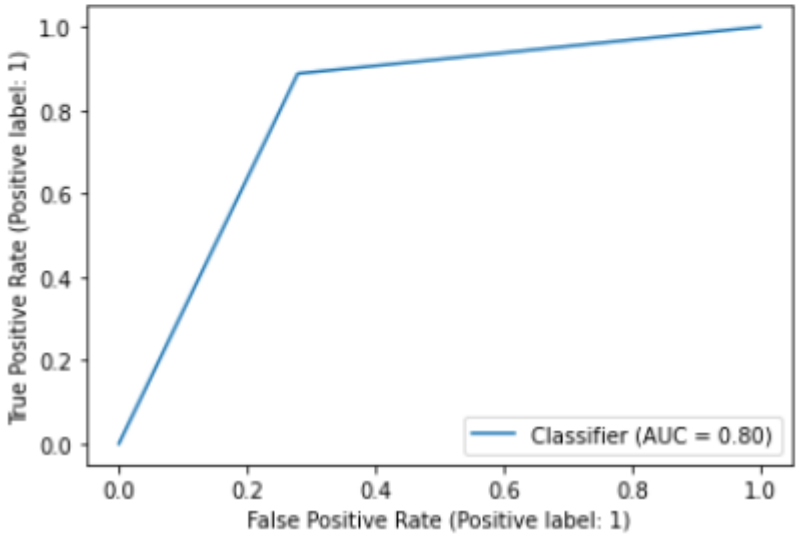
```
from sklearn.base import BaseEstimator, ClassifierMixin

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
```

Для каждого класса я делаю гистограммы по каждому признаку. Также строю гистограммы непосредственно для классов. В качестве параметра принимается количество разбиений для гистограмм. Результаты следующие:

```
{'bn__bins': 2}
Accuracy train: 0.8365203615692852
Accuracy tests: 0.8097826086956522
```




```
Precision tests: 0.7837837837837838
Recall tests: 0.8877551020408163
```



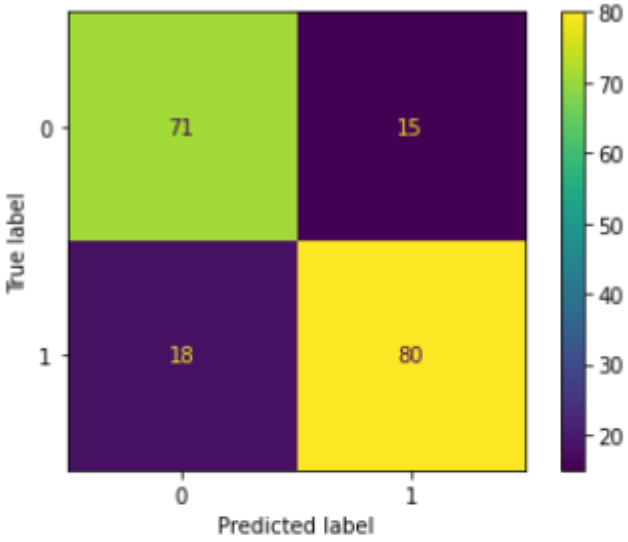
Лучшая точность для тренировочных данных достигается при 2-х интервалах. Вручную при 10 интервалах точность на тестовых была 83%.

Вторая реализация:

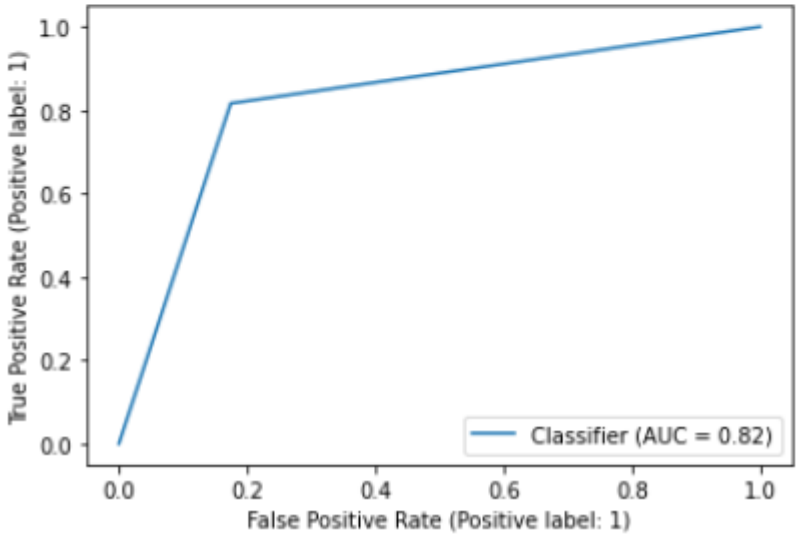
```
from sklearn.base import BaseEstimator, ClassifierMixin

class GaussianNaiveBayes(BaseEstimator, ClassifierMixin):
    def __init__(self):
        pass

    def fit(self, data, labels):
        self.data = data
        self.labels = labels
        self.mathexp = []
        self.variance = []
        self.classes = []

        for j in np.unique(labels):
            self.classes.append(j)
            self.mathexp.append(data[labels == j,].mean(axis = 0))
            self.variance.append(data[labels == j,].var(axis = 0))

    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)):
                t = np.exp((-1/2) * ((data - self.mathexp[i]) ** 2) / (2 * self.variance
                    [i])) / np.sqrt(2 * np.pi * self.variance[i])
                t = np.cumprod(t)
                if t[-1] > maximum:
                    maximum = t[-1]
                    ans = self.classes[i]
            res[j] = ans
        return res
```

Здесь я считаю матожидание и дисперсию, пользуясь потом плотностью вероятности нормального распределения.

```
Accuracy tests: 0.8206521739130435
```




```
Precision tests: 0.8421052631578947
Recall tests: 0.8163265306122449
```



Тут количество правильных ответов достигло отметки в 82%. Реализация из sklearn дала результат в 83%. Такие хорошие результаты обуславливаются тем, что распределение признаков немного похоже на нормальное.

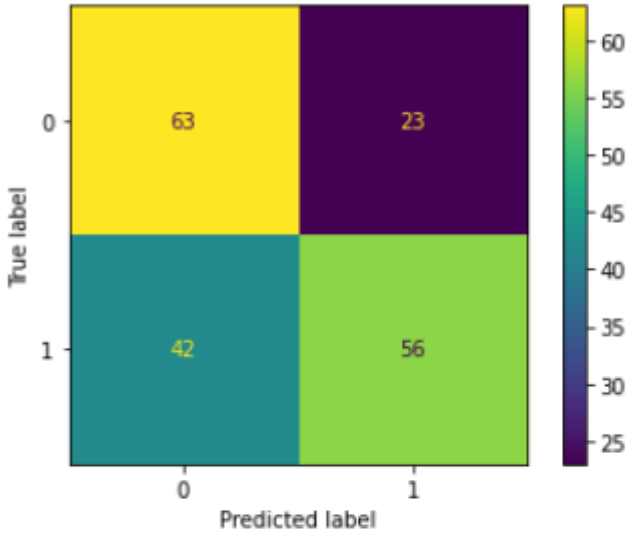
Настало время Linear/ Logistic Regression. Код выглядит следующим образом:

```
from sklearn.base import BaseEstimator, ClassifierMixin

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

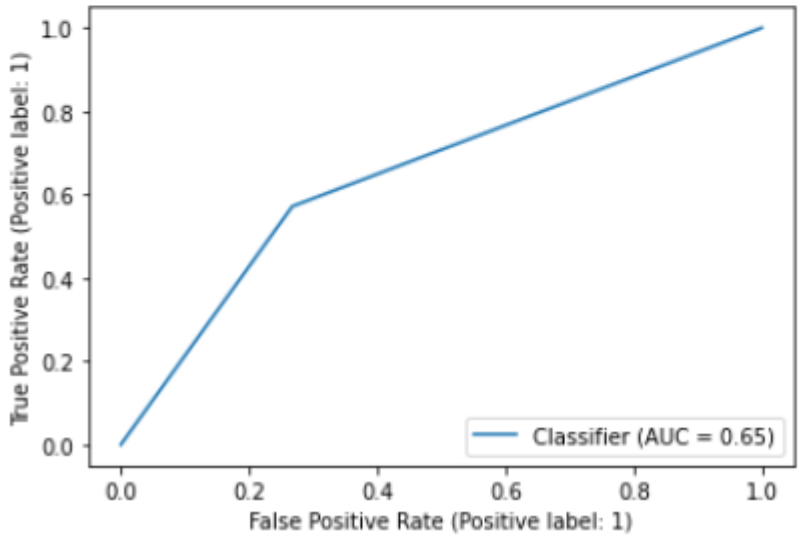
Тут на вход подается 3 параметра: размер батча(batch_size) - размер «пачки», которую мы будем загонять в модель, коэффициент обучения(lr) - скорость градиентного спуска и количество эпох(nepoch) - сколько раз мы будет прогонять модель по одним и тем же данным. Параметры модели - матрица W, на которую мы будем умножать. Так как функция должна выглядеть, как Wx + b, увеличим размерность W и x на 1, добавив туда тем самым b. В качестве функции ошибки используется log loss. Также используется функция sigmoid для интепретации вывода результатов модели.

Результаты следующие:

```
{'lin__batch_size': 10, 'lin__lr': 0.1, 'lin__nepoch': 20}
Accuracy train: 0.6742894418041189
Accuracy tests: 0.6467391304347826
```




```
Precision tests: 0.7088607594936709
Recall tests: 0.5714285714285714
```



Как видно, не сильно впечатляюще. При анализе было видно, что многие данные плохо разделяются линией.

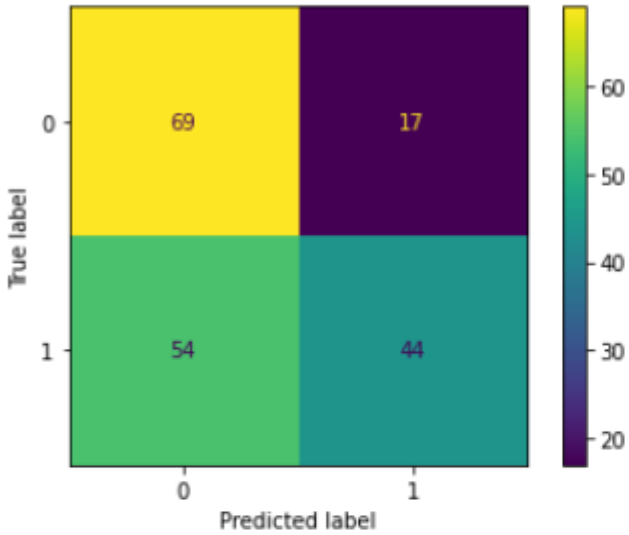
Напоследок остался SVM. Он практически не отличается от прошлого алгоритма. Различея лишь в функции ошибки, здесь она hinge loss. Также пришлось переименовать в ответах 0 на -1.

Код:

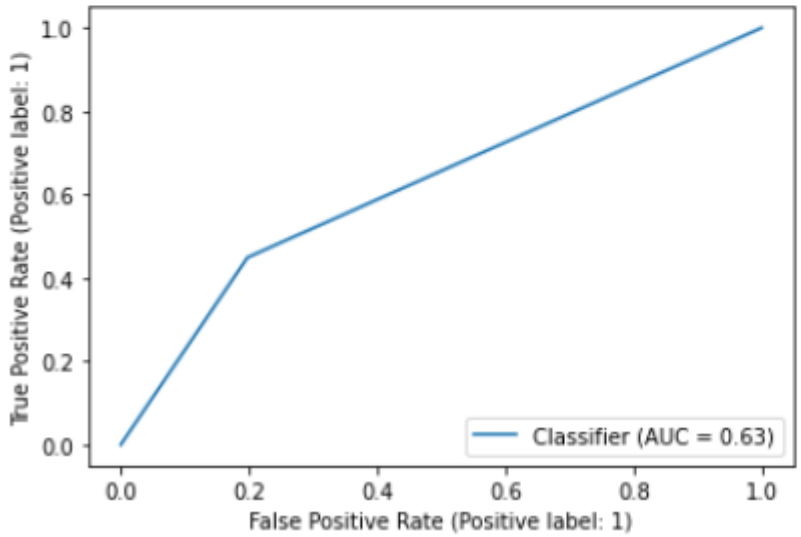
```
from sklearn.base import BaseEstimator, ClassifierMixin
class SVM(BaseEstimator, ClassifierMixin):
    def __init__(self, lr, lambd, batch_size, nepoch):
        self.nepoch = nepoch
        self.lr = lr
        self.lambd = lambd
        self.batch_size = batch_size

    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]

                dp = 2 * self.lambd * self.W + sums
                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)
        return np.sign(p)
```

Тут я добавил коэффициент регуляризации lambd. По результатам следующее:

```
{'lin__batch_size': 10, 'lin__lambd': 0, 'lin__lr': 0.1, 'lin__nepoch': 10}
Accuracy train: 0.6647283570962632
Accuracy tests: 0.6141304347826086
```




```
Precision tests: 0.7213114754098361
Recall tests: 0.4489795918367347
```

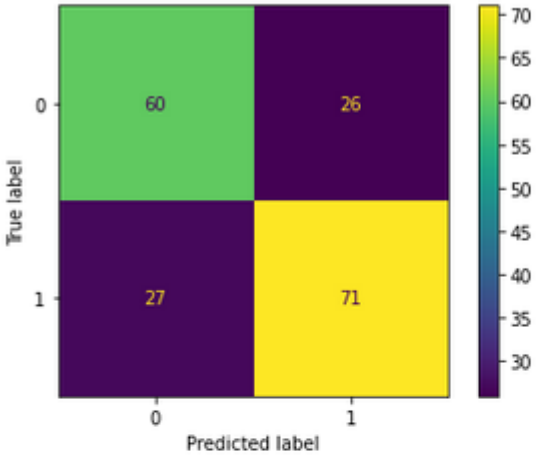


После таких плохих результатов работы линейных моделей я попытался преобразовать данные. Хорошо сказалось добавление произведения двух лучше всего разделяемых параметров: MaxHR и Oldpeak. Новый параметр тоже более-менее разделим.

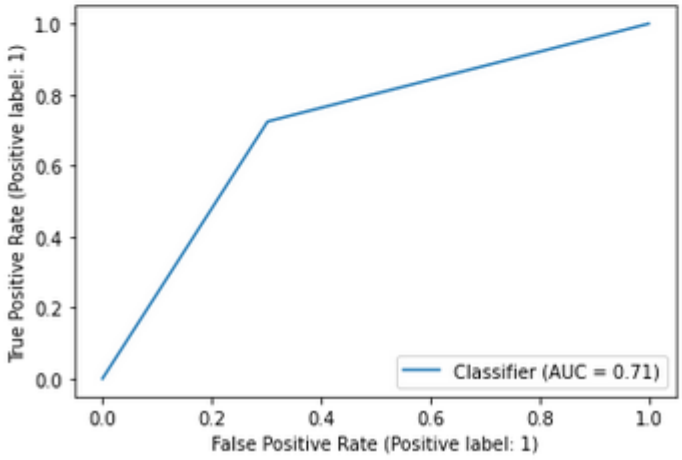
После этого преобразования линейные модели стали давать результаты получше.

Linear/ Logistic Regression

```
{'lin__batch_size': 10, 'lin__lr': 0.01, 'lin__nepoch': 20}
Accuracy train: 0.7180039138943248
Accuracy tests: 0.7119565217391305
```

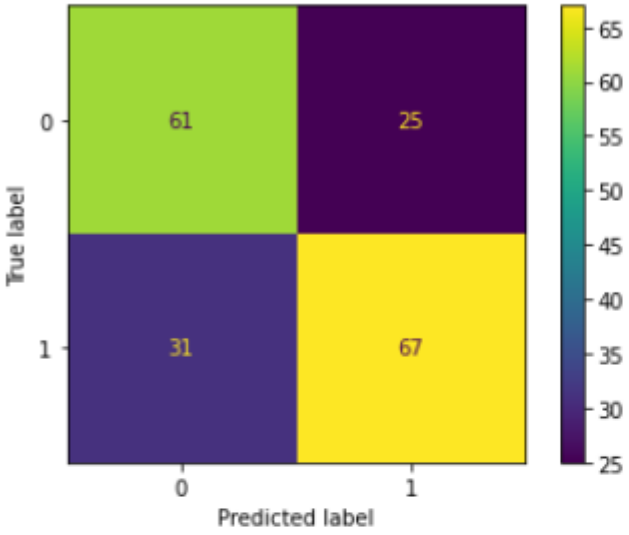



```
Precision tests: 0.7319587628865979
Recall tests: 0.7244897959183674
```

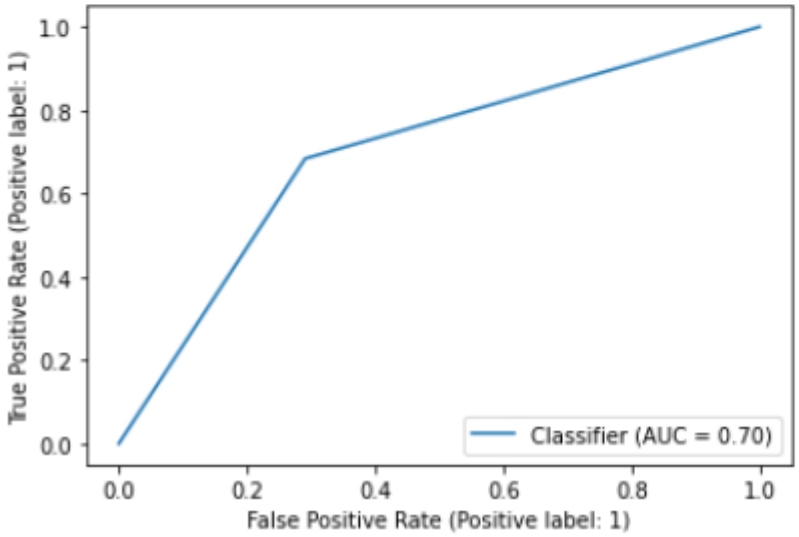


SVM

```
{'lin__batch_size': 10, 'lin__lambd': 0, 'lin__lr': 0.01, 'lin__nepoch': 20}
Accuracy train: 0.7261764979964588
Accuracy tests: 0.6956521739130435
```

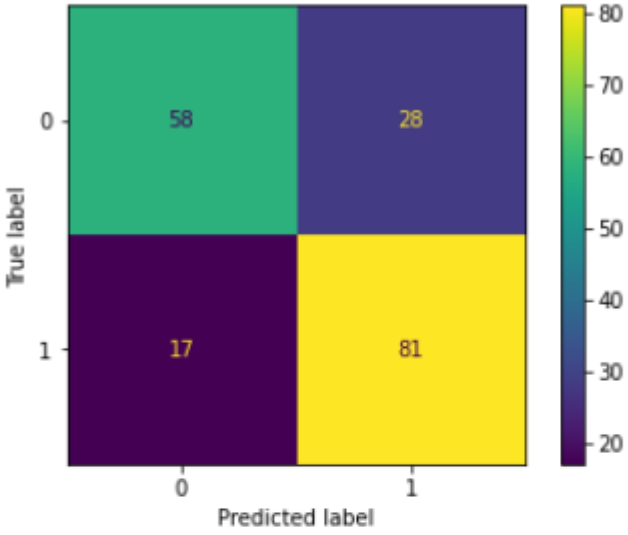



```
Precision tests: 0.7282608695652174
Recall tests: 0.6836734693877551
```

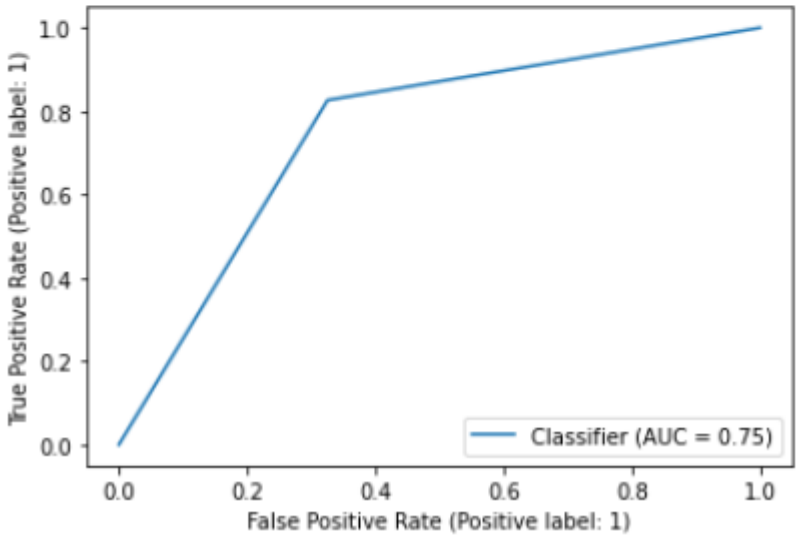


Также стал лучше работать KNN

```
{'knn__k': 17}
Accuracy train: 0.7615786040443575
Accuracy tests: 0.7554347826086957
```



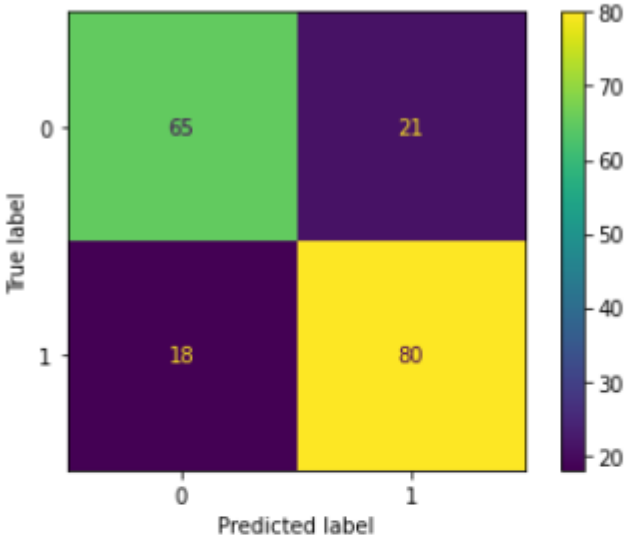

```
Precision tests: 0.7431192660550459
Recall tests: 0.826530612244898
```



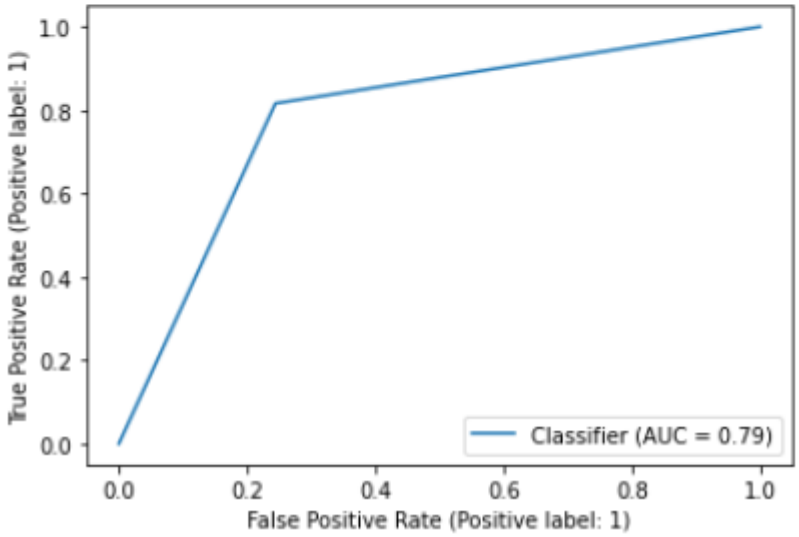
Однако Naive Bayes стал работать хуже.

С гистограммами:

```
{'bn__bins': 2}
Accuracy train: 0.8065045196160655
Accuracy tests: 0.7880434782608695
```

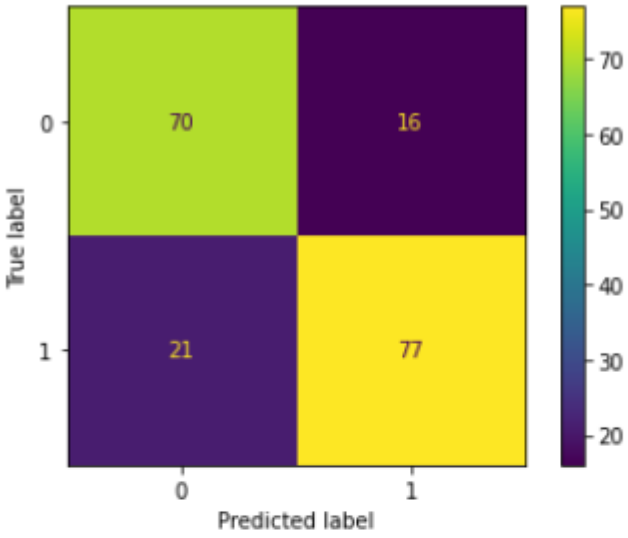



```
Precision tests: 0.7920792079207921
Recall tests: 0.8163265306122449
```

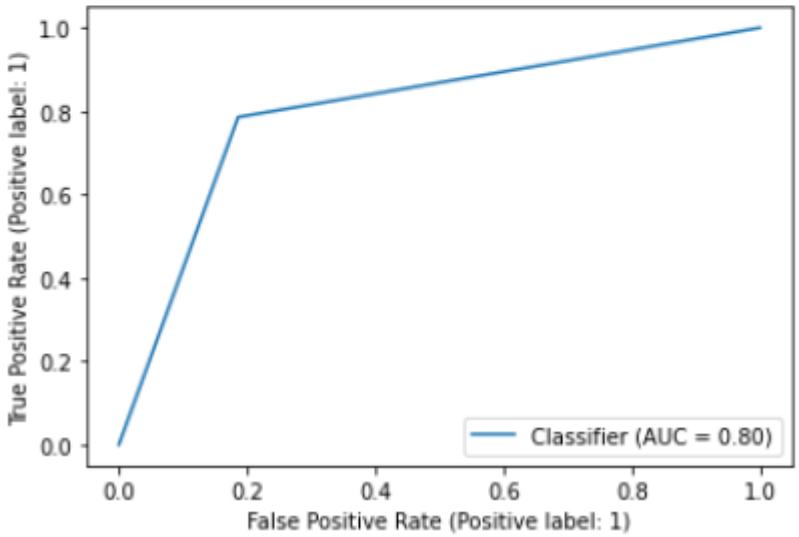


На основе нормального распределения:

```
Accuracy tests: 0.7989130434782609
```




```
Precision tests: 0.8279569892473119
Recall tests: 0.7857142857142857
```



Лучшие обученные модели с гиперпараметрами заархивированы с помощью pickle.

3 Выводы

Данная лабораторная работа дала мне интересный опыт в работе с настоящими данными. В качестве темы я взял медицину, так как эта область, как мне показалось, наиболее хороша с использованием искусственного интеллекта. Реализовав все эти алгоритмы, я получил точность 70-82%, что вполне неплохо. Конечно, доля ошибки немаленькая, но в такой теме хуже не будет, если лишний раз будешь следить за своим здоровьем. Думаю, что если доработать эту модель с использованием более подходящих алгоритмов и достичь точности около 90%, то ее можно использовать на практике.