

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: Лошманов Ю.А.
Преподаватель: Самир Ахмед
Группа: М8О-306Б-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:

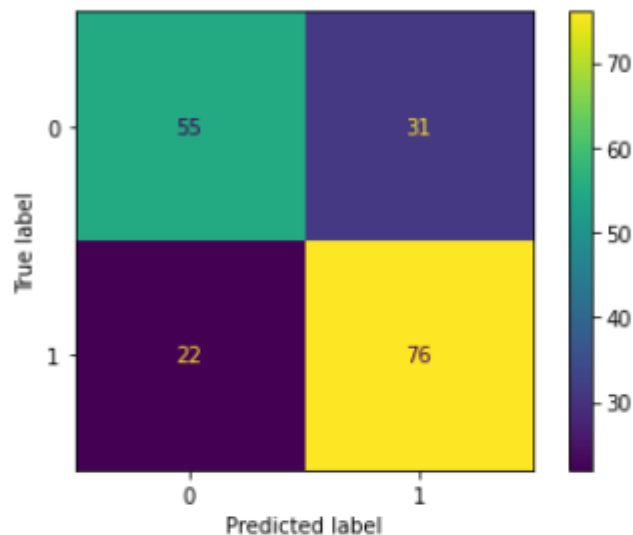
```
1 from sklearn.base import BaseEstimator, ClassifierMixin
2
3 class KNN(BaseEstimator, ClassifierMixin):
4     def __init__(self, k):
5         self.k = k
6
7     def fit(self, data, labels):
8         self.data = data
9         self.labels = labels
10
11    def euclidean_distance(self, row1, row2):
12        distance = 0
13        for i in range(len(row1)):
14            distance += (row1[i] - row2[i]) ** 2
15        return math.sqrt(distance)
16
17    def predict(self, maindata):
18        res = np.ndarray((maindata.shape[0],))
19        for j, data in enumerate(maindata):
20            distances = []
21            for i, row in enumerate(self.data):
22                distances.append((self.euclidean_distance(data, row), self.labels[i]))
23            distances.sort(key = lambda tup: tup[0])
24            dictionary = collections.defaultdict(int)
25            for i in range(self.k):
26                dictionary[distances[i][1]] += 1
27            res[j] = max(dictionary.items(), key = lambda tup: tup[1])[0]
28        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}
```

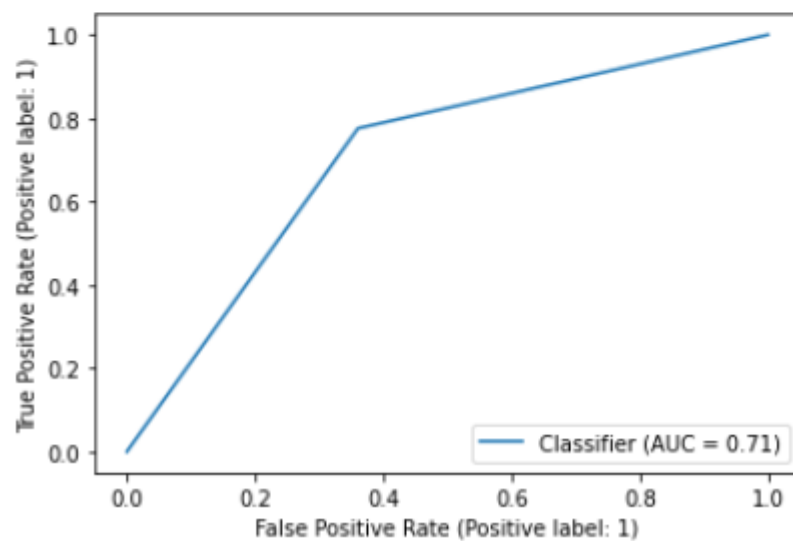
```
Accuracy train: 0.7111173236417855
```

```
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

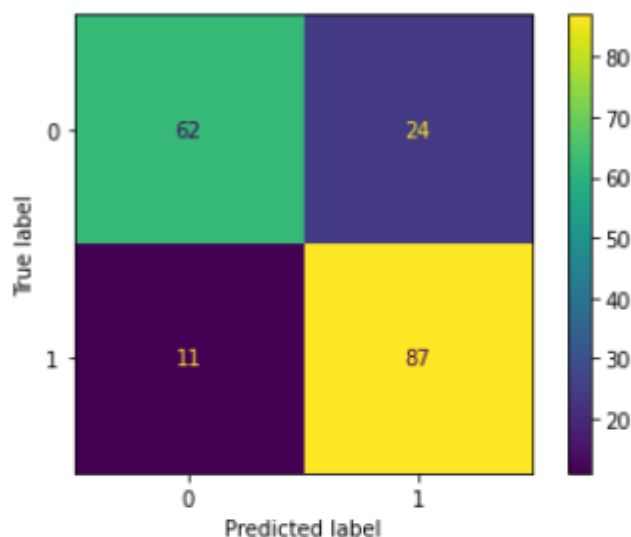
2) Делая предположение, что данные имеют нормальное распределение

Первая реализация выглядит следующим образом:

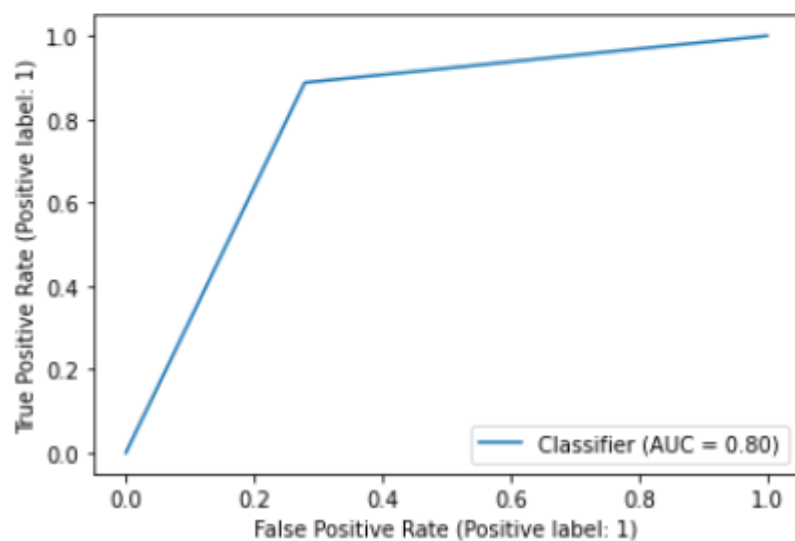
```
1 from sklearn.base import BaseEstimator, ClassifierMixin
2
3 class NaiveBayes(BaseEstimator, ClassifierMixin):
4     def __init__(self, bins):
5         self.bins = bins
6         pass
7
8     def fit(self, data, labels):
9         self.data = data
10        self.labels = labels
11        self.classes = []
12        for j in np.unique(labels):
13
14            self.classes.append([])
15            for i in range(data.shape[1]):
16                self.classes[j].append(*np.histogram(data[labels == j, i], bins = self.
17                    bins))
18                self.classes[j][-1][0] = self.classes[j][-1][0].astype('float64') / len(
19                    data[labels == j, i])
20
21        self.prclasses = np.unique(labels, return_counts = True)[1] / len(labels)
22
23    def predict(self, maindata):
24        res = np.ndarray((maindata.shape[0],))
25        for j, data in enumerate(maindata):
26            maximum = 0
27            ans = 0
28            for i in range(len(self.classes)):
29                p = self.prclasses[i]
30                for k in range(len(self.classes[i])):
31                    ind = np.digitize(data[k], self.classes[i][k][1])
32
33                    if ind >= len(self.classes[i][k][1]) or ind <= 0:
34                        p = 0
35                    else:
36                        p *= self.classes[i][k][0][ind - 1]
37
38                if p > maximum:
39                    maximum = p
40                    ans = i
41
42        res[j] = ans
43    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
```



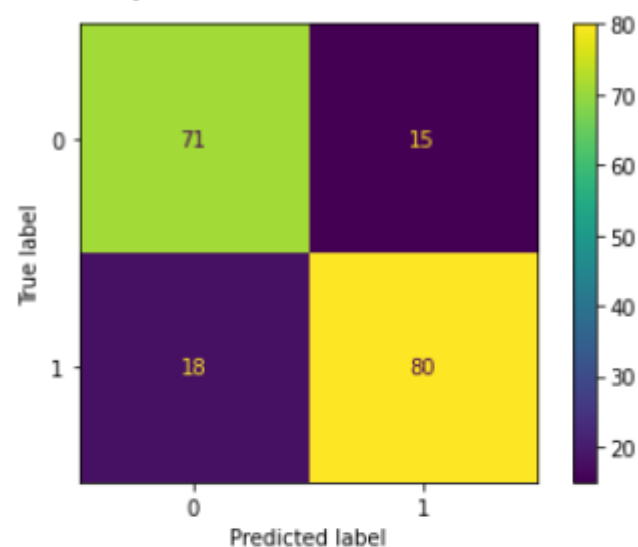
Лучшая точность для тренировочных данных достигается при 2-х интервалах. Вручную при 10 интервалах точность на тестовых была 83%.

Вторая реализация:

```
1 from sklearn.base import BaseEstimator, ClassifierMixin
2
3 class GaussianNaiveBayes(BaseEstimator, ClassifierMixin):
4     def __init__(self):
5         pass
6
7     def fit(self, data, labels):
8         self.data = data
9         self.labels = labels
10        self.mathexp = []
11        self.variance = []
12        self.classes = []
13
14        for j in np.unique(labels):
15            self.classes.append(j)
16            self.mathexp.append(data[labels == j,].mean(axis = 0))
17            self.variance.append(data[labels == j,].var(axis = 0))
18
19    def predict(self, maindata):
20        res = np.ndarray((maindata.shape[0],))
21        for j, data in enumerate(maindata):
22            maximum = 0
23            ans = 0
24            for i in range(len(self.classes)):
25                t = np.exp((-1/2) * ((data - self.mathexp[i]) ** 2) / (2 * self.variance
26                    [i])) / np.sqrt(2 * np.pi * self.variance[i])
27                t = np.cumprod(t)
28                if t[-1] > maximum:
29                    maximum = t[-1]
30                    ans = self.classes[i]
31            res[j] = ans
32        return res
```

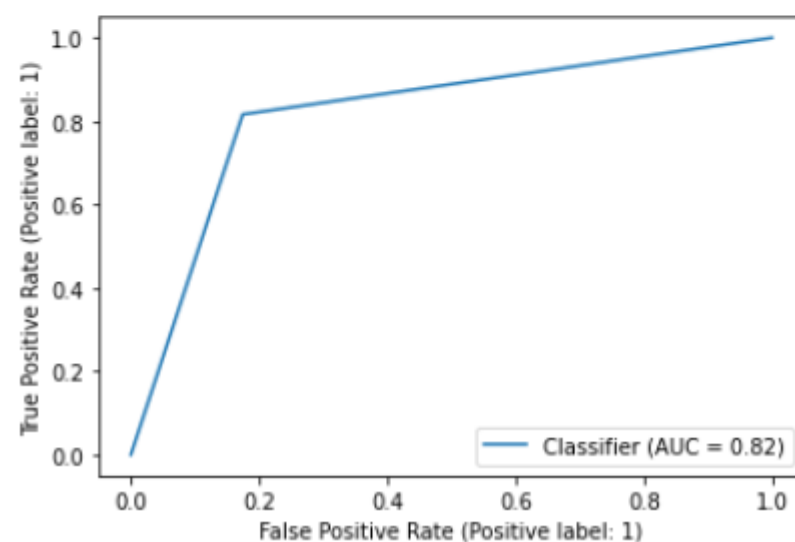
Здесь я считаю матожидание и дисперсию, пользуясь потом плотностью вероятности нормального распределения.

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.

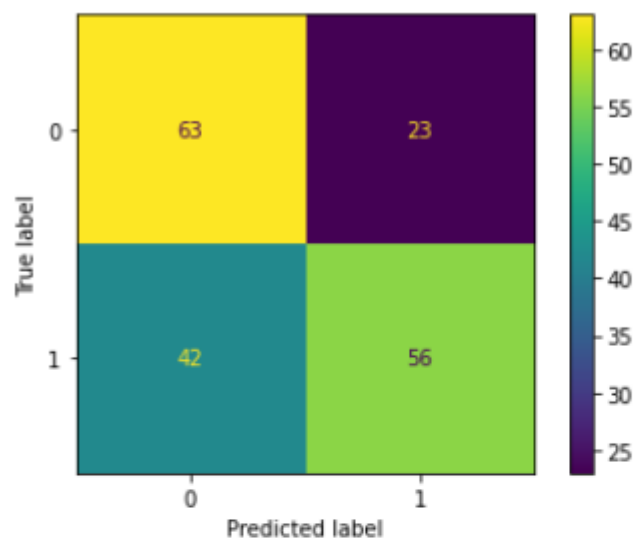
Настало время Linear/ Logistic Regression. Код выглядит следующим образом:

```
1 from sklearn.base import BaseEstimator, ClassifierMixin
2
3 class Linear(BaseEstimator, ClassifierMixin):
4     def __init__(self, lr, nepoch, batch_size):
5         self.lr = lr
6         self.nepoch = nepoch
7         self.batch_size = batch_size
8         pass
9
10    def sigmoid(self, x):
11        self.l = 1 / (1 + np.exp(-x))
12        return self.l
13
14    def fit(self, data, labels):
15        data = np.concatenate((data, np.ones((data.shape[0],1))), axis = 1)
16        self.W = np.random.normal(0, 1, (len(data[0]),))
17
18        for i in range(self.nepoch):
19            for i in range(0, len(data), self.batch_size):
20                xb = data[i:i + self.batch_size]
21                yb = labels[i:i + self.batch_size]
22                p = np.dot(self.W, xb.T)
23                s = self.sigmoid(p)
24                dp = np.dot(xb.T, (s - yb).T)
25                self.W -= self.lr * dp
26
27    def predict(self, maindata):
28        maindata = np.concatenate((maindata, np.ones((maindata.shape[0],1))), axis = 1)
29        p = np.dot(self.W, maindata.T)
30        s = self.sigmoid(p)
31        return (s > 0.5).astype('int64')
```

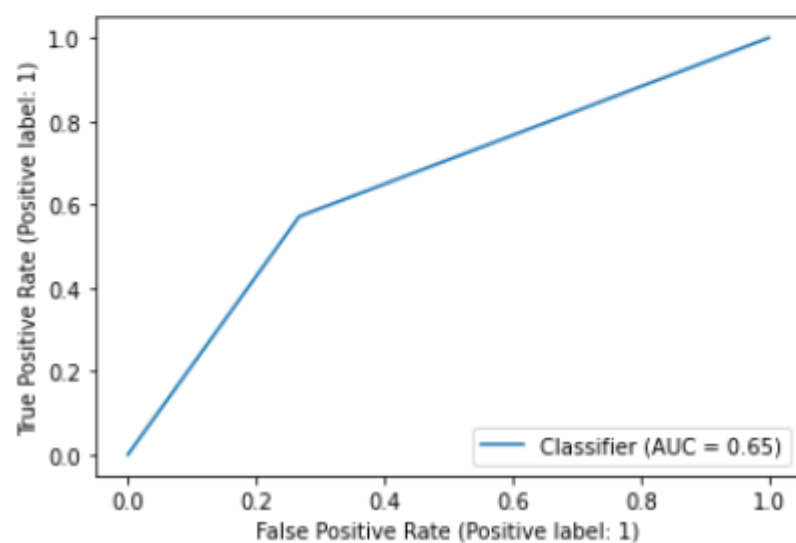
Тут на вход подается 3 параметра: размер батча(batch_size) - размер «пачки», которую мы будем загонять в модель, коэффициент обучения(lr) - скорость градиентного спуска и количество эпох(nepoch) - сколько раз мы будем прогонять модель по одним и тем же данным. Параметры модели - матрица W, на которую 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 интерпретации 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. Различия лишь в функции ошибки, здесь она hinge loss. Также пришлось переименовать в ответах 0 на -1.

Код:

```

1 from sklearn.base import BaseEstimator, ClassifierMixin
2 class SVM(BaseEstimator, ClassifierMixin):
3     def __init__(self, lr, lamdb, batch_size, nepoch):
4         self.nepoch = nepoch
5         self.lr = lr
6         self.lamdb = lamdb
7         self.batch_size = batch_size
8
9     def fit(self, data, labels):
10        data = np.concatenate((data, np.ones((data.shape[0],1))), axis=1)
11        self.W = np.random.normal(0, 1, (len(data[0]),))
12
13        for i in range(self.nepoch):
14            for i in range(0, len(data), self.batch_size):
15                xb = data[i:i + self.batch_size]
16                yb = labels[i:i + self.batch_size]
17
18                p = np.dot(self.W, xb.T)
19
20                sums = np.zeros_like(self.W)
21                for i in range(len(p)):
22                    if 1 - p[i] * yb[i] > 0:
23                        sums -= xb[i] * yb[i]
24
25                dp = 2 * self.lamdb * self.W + sums
26                self.W -= self.lr * dp
27
28
29    def predict(self, maindata):
30        maindata = np.concatenate((maindata, np.ones((maindata.shape[0],1))), axis=1)
31        p = np.dot(self.W, maindata.T)
32        return np.sign(p)

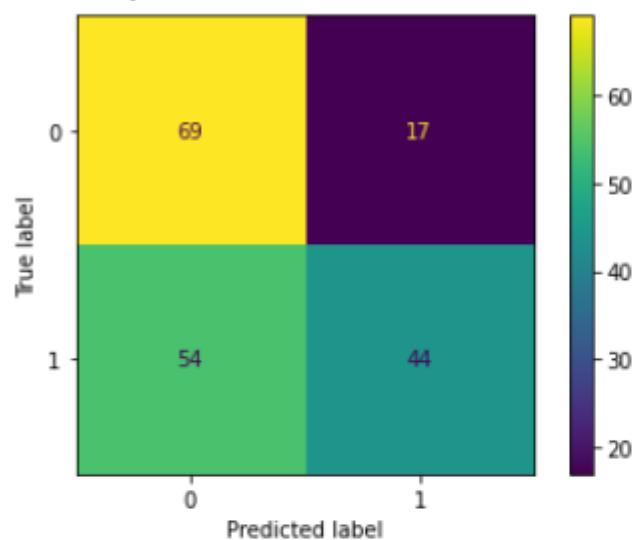
```

Тут я добавил коэффициент регуляризации lamdb. По результатам следующее:

```
{'lin_batch_size': 10, 'lin_lambda': 0, 'lin_lr': 0.1, 'lin_nepoch': 10}
```

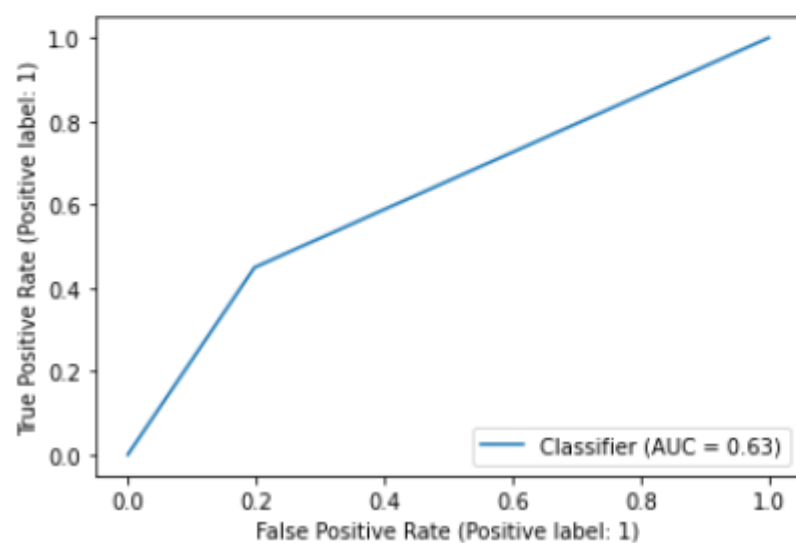
Accuracy train: 0.6647283570962632

Accuracy tests: 0.6141304347826086

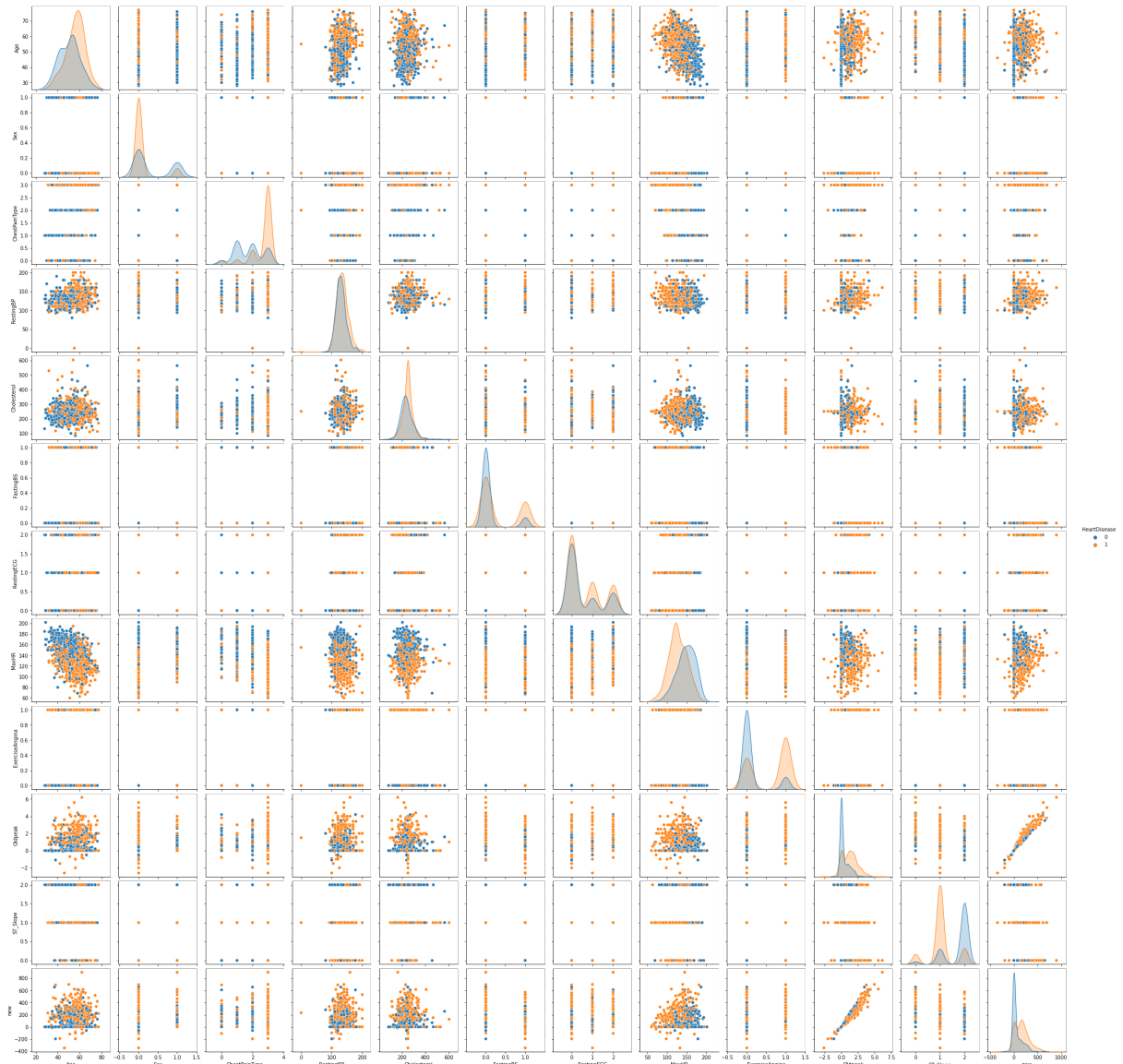


Precision tests: 0.7213114754098361

Recall tests: 0.4489795918367347



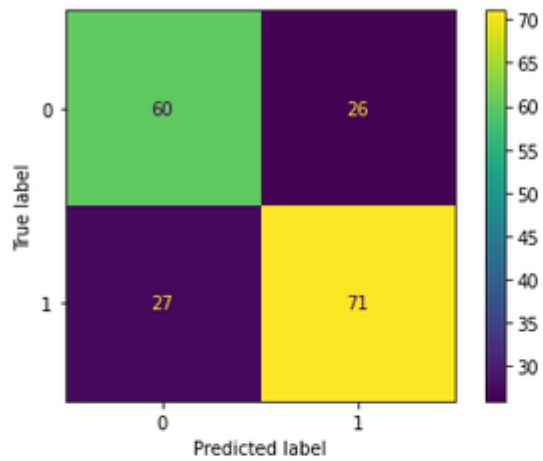
После таких плохих результатов работы линейных моделей я попытался преобразовать данные. Хорошо сказалось добавление произведения двух лучше всего разделяемых параметров: MaxHR и Oldpeak. Новый параметр тоже более-менее разделим.



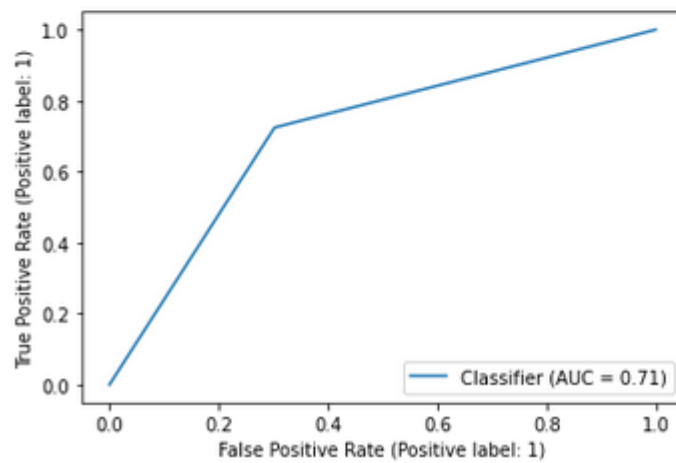
После этого преобразования линейные модели стали давать результаты лучше.

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

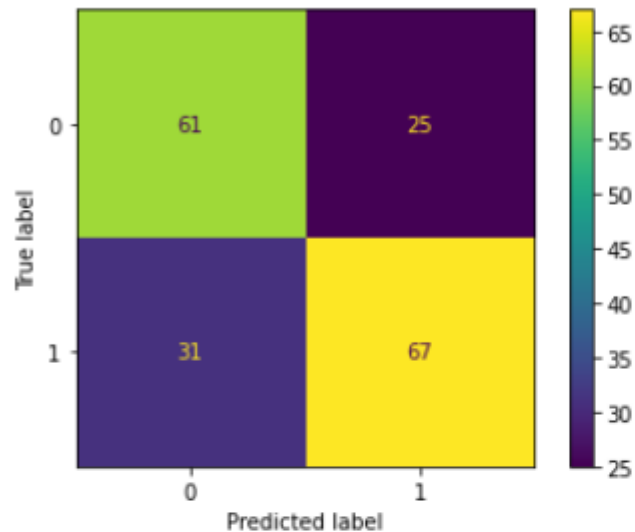


SVM

```
{'lin_batch_size': 10, 'lin_lambda': 0, 'lin_lr': 0.01, 'lin_nepoch': 20}
```

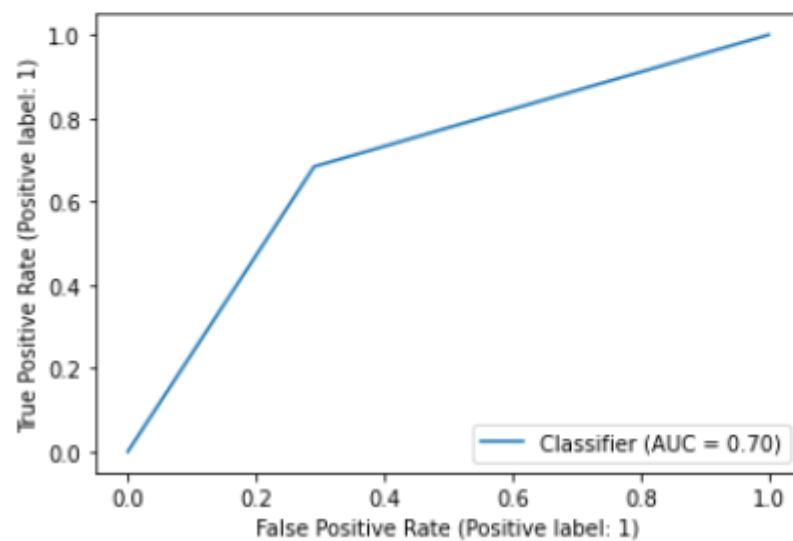
```
Accuracy train: 0.7261764979964588
```

```
Accuracy tests: 0.6956521739130435
```



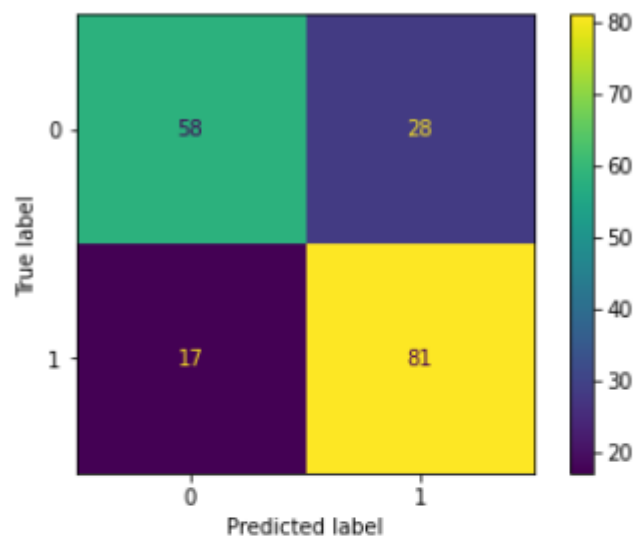
```
Precision tests: 0.7282608695652174
```

```
Recall tests: 0.6836734693877551
```

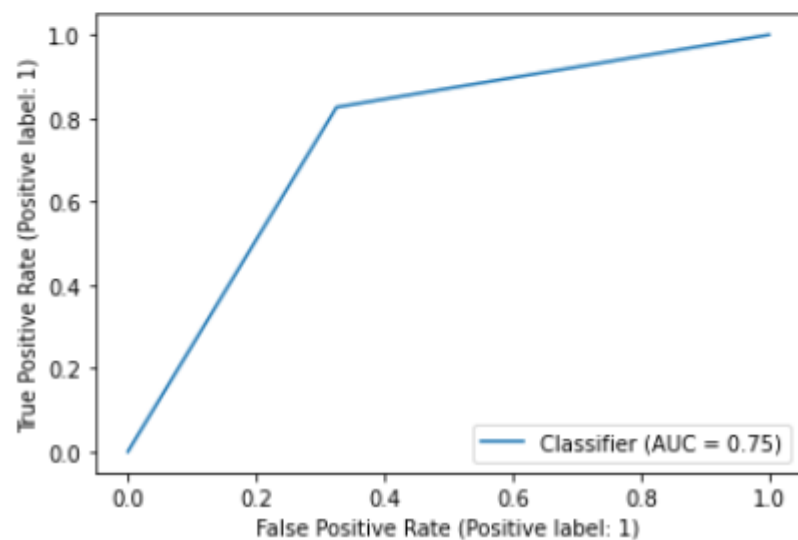


Также стал лучше работать KNN


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

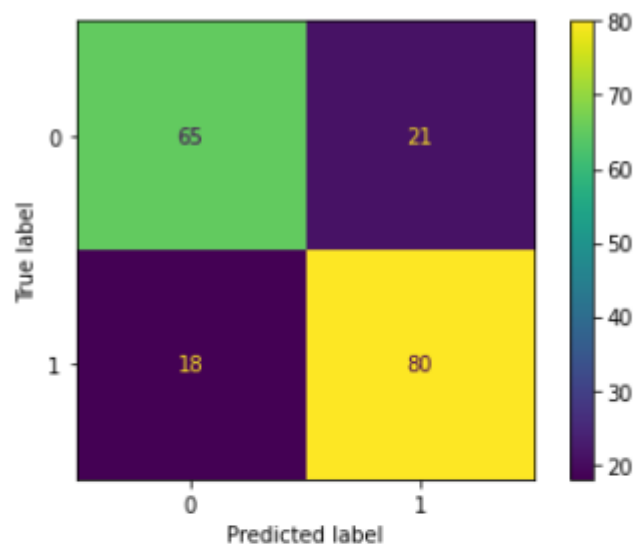


```
Precision tests: 0.7431192660550459  
Recall tests: 0.826530612244898
```

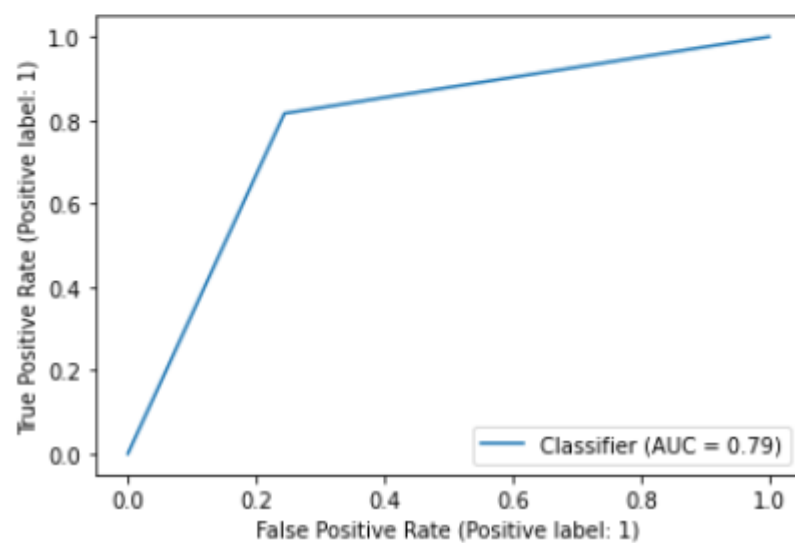


Однако Naïve Bayes стал работать хуже.
С гистограммами:

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

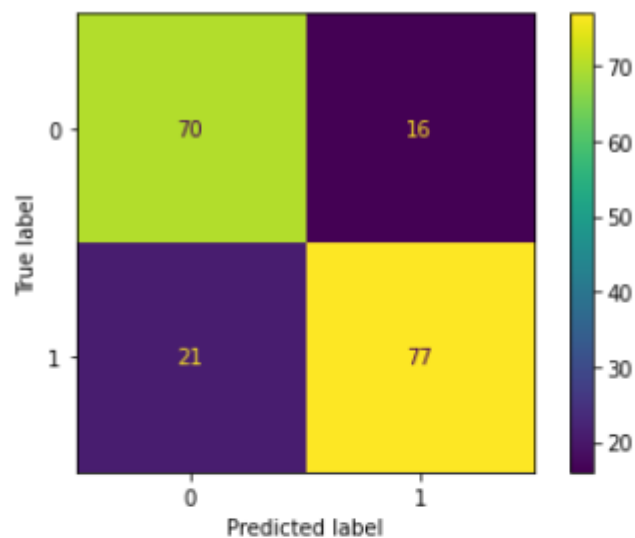


```
Precision tests: 0.7920792079207921  
Recall tests: 0.8163265306122449
```



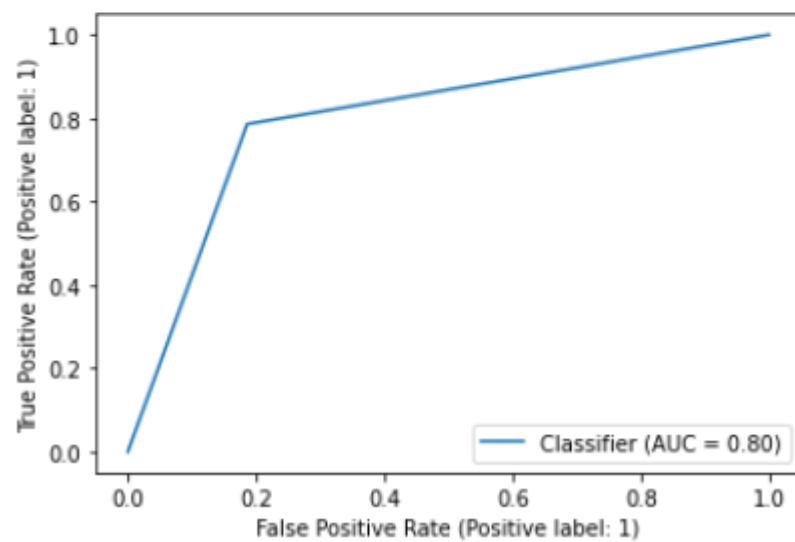
На основе нормального распределения:

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.