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. А. Чернобаев
Преподаватель: Самир Ахмед
Группа: М8О-308Б-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 Ход решения

Log regression

Статистическая модель, используемая для прогнозирования вероятности возникновения некоторого события путём его сравнения с логистической кривой. Эта регрессия выдаёт ответ в виде вероятности бинарного события (1 или 0).

```
class Logistic(BaseEstimator, ClassifierMixin):
  def __init__(self, lr, nepoch, batch_size):
    self.lr = lr
    self.nepoch = nepoch
    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)
        s = self.sigmoid(p)
        dp = np.dot(xb.T, (s - yb).T)
        self.W -= self.lr * dp

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

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

Результат:

```
{'log__batch_size': 1, 'log__lr': 0.01, 'log__nepoch': 20}
Accuracy train: 0.6825962769624742
Accuracy: 0.7150837988826816
Recall: 0.4383561643835616
Precision: 0.7619047619047619
```

Рис. 1:

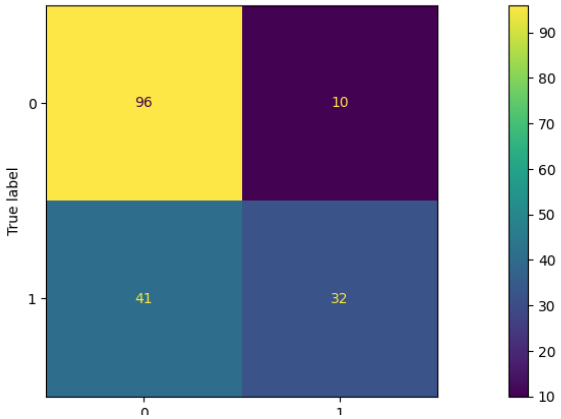

Рис. 2:

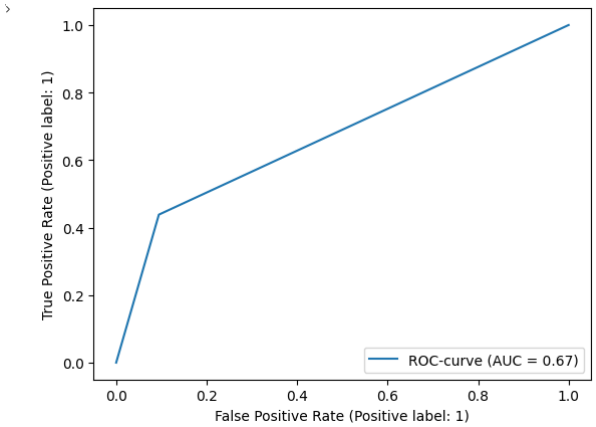

Рис. 3:

Коробочный резульат:

```
{'log__alpha': 0.0001, 'log__max_iter': 1000}
Accuracy train: 0.6980202895695854
Accuracy: 0.7597765363128491
Recall: 0.4657534246575342
Precision: 0.8947368421052632
```

Рис. 4:

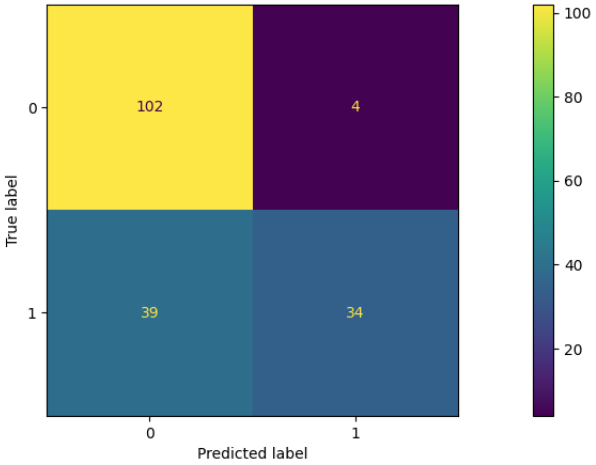

Рис. 5:

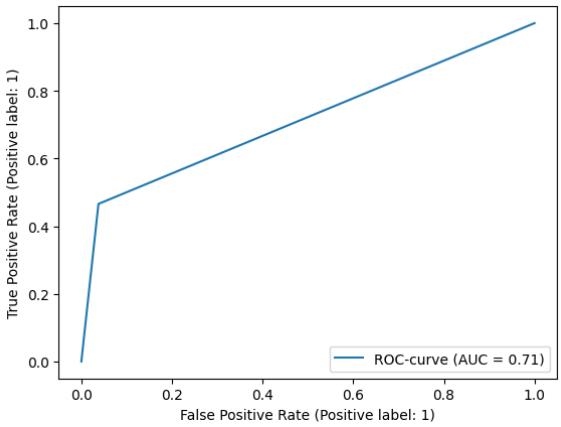

Рис. 6:

Точность коробочного решения немного выше самодельного варианта.

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

```

```
8   def fit(self, data, labels):
9     data = np.concatenate((data, np.ones((data.shape[0],1))), axis=1)
10    self.W = np.random.normal(0, 1, (len(data[0]),))
11
12    for i in range(self.nepoch):
13      for i in range(0, len(data), self.batch_size):
14        xb = data[i:i + self.batch_size]
15        yb = labels[i:i + self.batch_size]
16
17        p = np.dot(self.W, xb.T)
18
19        sums = np.zeros_like(self.W)
20        for i in range(len(p)):
21          if 1 - p[i] * yb[i] > 0:
22            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, main_data):
29    main_data = np.concatenate((main_data, np.ones((main_data.shape[0],1))), axis=1)
30    p = np.dot(self.W, main_data.T)
31    return np.sign(p)
```

Основная идея метода — перевод исходных векторов в пространство более высокой размерности и поиск разделяющей гиперплоскости с наибольшим зазором в этом пространстве. Две параллельных гиперплоскости строятся по обеим сторонам гиперплоскости, разделяющей классы. Разделяющей гиперплоскостью будет гиперплоскость, создающая наибольшее расстояние до двух параллельных гиперплоскостей. Алгоритм основан на допущении, что чем больше разница или расстояние между этими параллельными гиперплоскостями, тем меньше будет средняя ошибка классификатора.

Результат:

```
{'lin__batch_size': 1, 'lin__lambd': 0.001, 'lin__lr': 0.01, 'lin__nepoch': 10}
Accuracy train: 0.6882103811681276
Accuracy: 0.7150837988826816
Recall: 0.4657534246575342
Precision: 0.7391304347826086
```

Рис. 7:

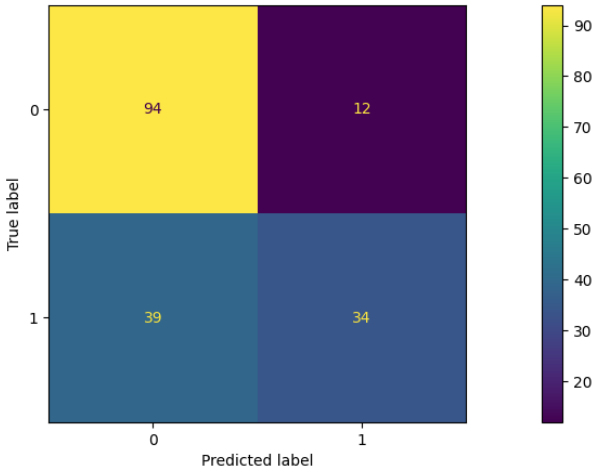

Рис. 8:

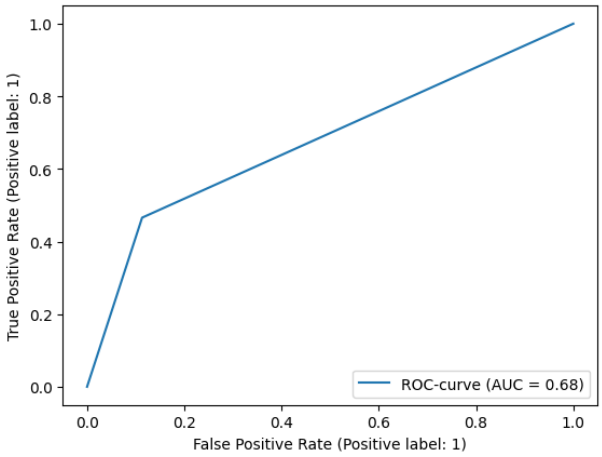

Рис. 9:

Коробочный результат:

```
{'lin__alpha': 0.0001, 'lin__max_iter': 100}
Accuracy train: 0.7022456416822613
Accuracy: 0.7262569832402235
Recall: 0.3424657534246575
Precision: 0.9615384615384616
```

Рис. 10:

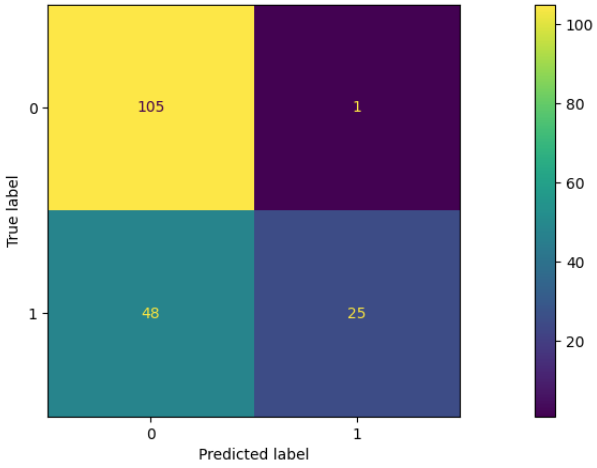

Рис. 11:

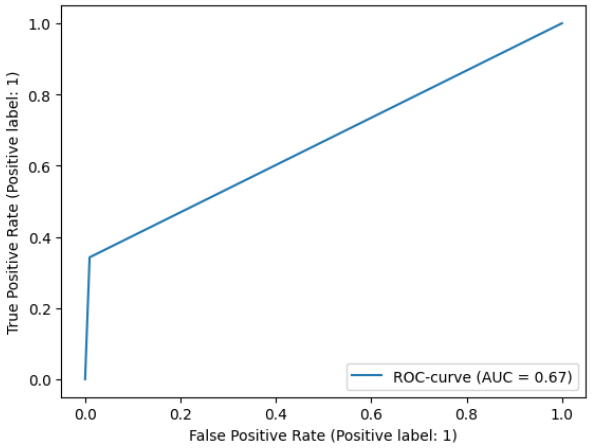

Рис. 12:

Точность решения примерно равна точности логистической регрессии.

KNN

```
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
```

В случае использования метода для классификации объект присваивается тому классу, который является наиболее распространённым среди k соседей данного элемента, классы которых уже известны. В случае использования метода для регрессии, объекту присваивается среднее значение по k ближайшим к нему объектам, значения которых уже известны.

Результат:

```
{'knn__k': 8}
Accuracy train: 0.7275189599133262
Accuracy: 0.776536312849162
Recall: 0.6575342465753424
Precision: 0.7619047619047619
```

Рис. 13:

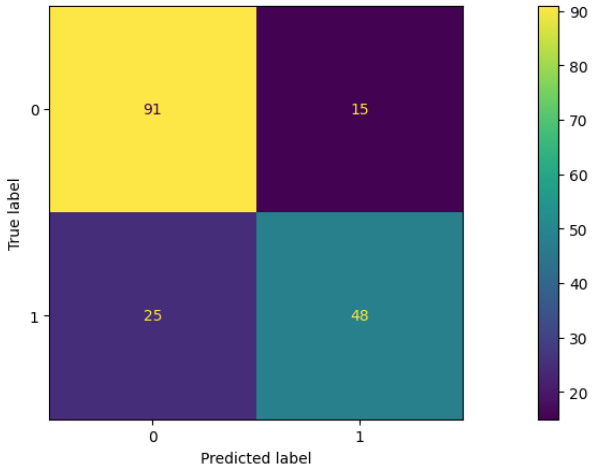

Рис. 14:

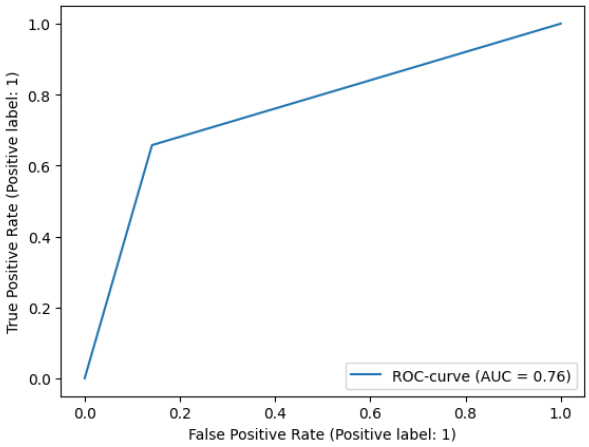

Рис. 15:

Коробочный результат:

```
{'knn__n_neighbors': 7}
Accuracy train: 0.716320299418891
Accuracy: 0.7541899441340782
Recall: 0.6438356164383562
Precision: 0.7230769230769231
```

Рис. 16:

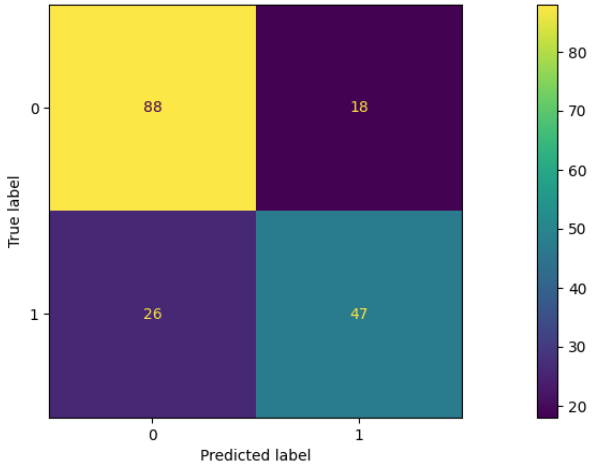

Рис. 17:

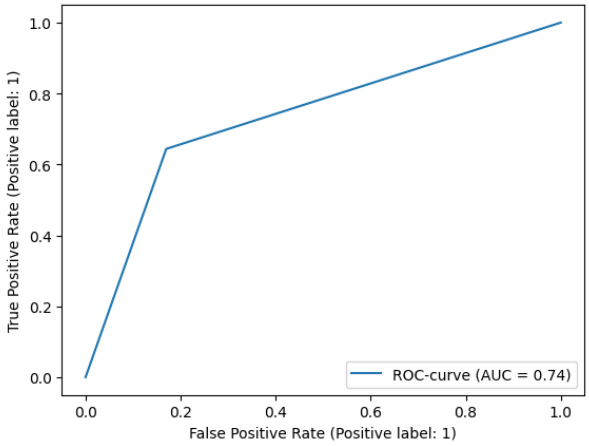

Рис. 18:

Точность данного решения больше, чем у двух предыдущих.

Naive Bayes

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

  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, main_data):
    res = np.ndarray((main_data.shape[0],))
    for j, data in enumerate(main_data):
      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
```

Простой вероятностный классификатор, основанный на применении теоремы Байеса со строгими (наивными) предположениями о независимости. В зависимости от точной природы вероятностной модели, наивные байесовские классификаторы могут обучаться очень эффективно. Во многих практических приложениях для оценки параметров для наивных байесовых моделей используют метод максимального правдоподобия; другими словами, можно работать с наивной байесовской моделью, не веря в байесовскую вероятность и не используя байесовские методы.

Результат:

```
{'bn__bins': 68}
Accuracy train: 0.743011917659805
Accuracy: 0.7318435754189944
Recall: 0.5616438356164384
Precision: 0.7192982456140351
```

Рис. 19:

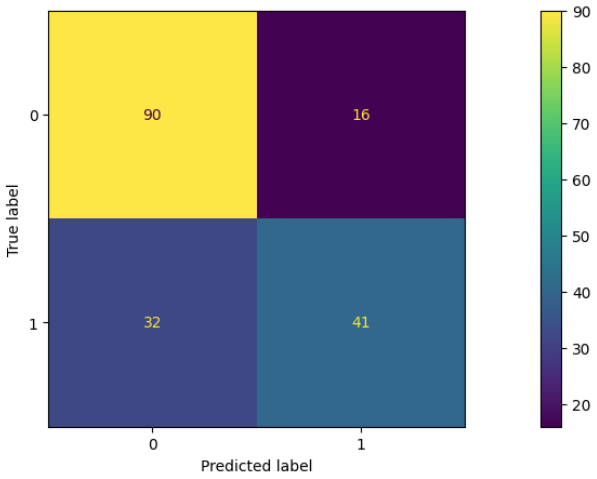

Рис. 20:

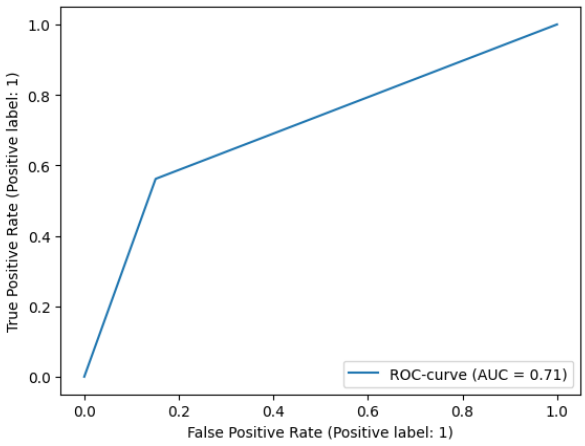

Рис. 21:

Коробочный результат:

```
Accuracy: 0.776536312849162
Recall: 0.6164383561643836
Precision: 0.7894736842105263
```

Рис. 22:

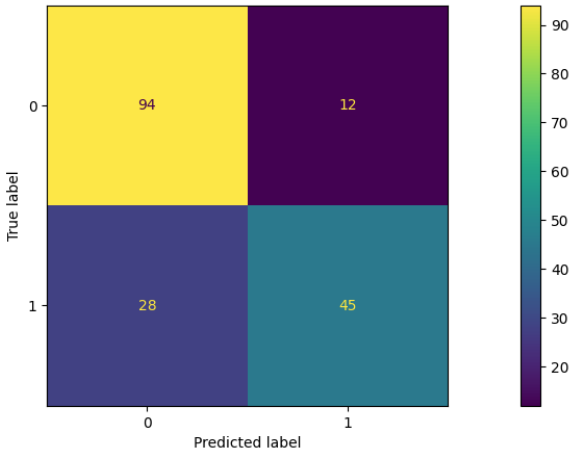

Рис. 23:

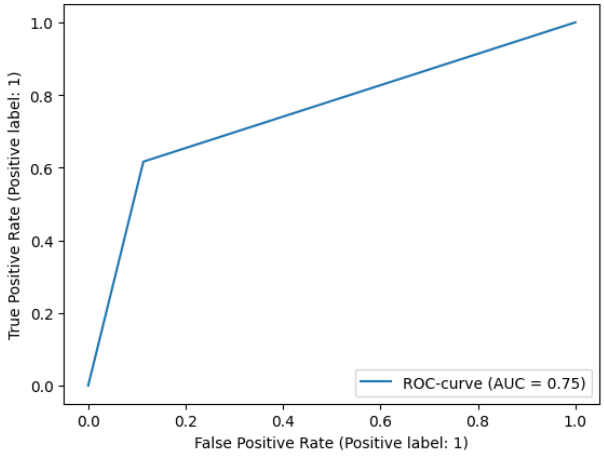

Рис. 24:

Точность данного решения примерно равна точности метода k ближайших соседей.

2 Выводы

В данной лабораторной работе я познакомился с классическими алгоритмами машинного обучения, а именно с: Логистической регрессией, методом опорных векторов, методом k-ближайших соседей и с наивным байесовским классификатором. Точность методов была в районе 68-77% (что меньше точности, используя нейронные сети, используя их я получил точность в районе 82%). Однако нейронные сети требуют вычисления на gpu и обучаются дольше. В целом, для определённый задач точность 68-77% может быть приемлемой.