Московский авиационный институт (национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Курсовой проект по курсу «Искусственный интеллект и глубокое обучение»

Студент: К. М. Воронов
Преподаватель: Б.В. Вишняков
Группа: М8О-407Б-19

Дата:
Оценка:
Подпись:

Москва, 2023

Задача

Выбрать задачу (классификации или регрессии), датасет и метрики качества. Выбранные данные необходимо проанализировать и визуализировать. Реализовать алгоритм (несколько алгоритмов), сравнить результаты работы с аналогом из sklearn.

1 Описание данных

1. Pregnancies - Number of times pregnant (Количество беременностей)

2. Glucose - Plasma glucose concentration a 2 hours in an oral glucose tolerance test (Концентрация глюкозы в плазме крови через 2 часа при пероральном тесте на толерантность к глюкозе)

3. BloodPressure - Diastolic blood pressure (mm Hg) (Диастолическое артериальное давление)

4. SkinThickness - Triceps skin fold thickness (mm) (Толщина кожной складки трицепса)

5. Insulin - 2-Hour serum insulin (mu U/ml) (2-Часовой сывороточный инсулин)

6. BMI - Body mass index (weight in kg/(height in m)$\hat{2}$) (Индекс массы тела)

7. DiabetesPedigree - Diabetes pedigree function (Функция родословной диабета)

8. Age (Возраст)

9. Outcome - Class variable (Наличие диабета)

2 Описание алгоритмов

В данной 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). В качестве расстояния используется евклидова метрика.

2) Naive Bayes Пусть у нас есть объект B, который описывается признаками b_i и классы A_k. Алгоритм основан на формуле Байеса:

$$P(A|B) = \frac{P(B|A)*P(A)}{P(B)},$$

Сделав предположение, что все признаки независимы между собой, мы можем разложить условную вероятность объекта при условии класса на произведение вероятностей наличия признаков при условии класса:

$$P(B|A_k) = \prod_{i=1}^{n} P(b_i|A_k)$$

$P(b_i|A)$ распределение каждого признака в зависимости от класса. Есть два варианта - предполагаем, что признаки по классу имеют нормальное распределение или строим гистограмму.

Так как все объекты разные, то $P(B)$ является одинаковой константой для всех B, поэтому ей можно пренебречь.

В результате берется класс, для которого $P(A|B)$ больше всего.

3 Предобработка данных

Для начала я проверил данные на присутствие в них дубликатов и пустых ячеек. Далее я проанализировал распределение данных. Оказалось, что в столбцах Glucose, BloodPressure, SkinThickness и BMI присутствуют выбросы в виде нулей, что невозможно для этих признаков. Из-за того, что данных не так много, я заполнил их средними значениями для каждого из исходов (есть диабет или нет). Также, так как здоровых было больше, чем больных, я сделал оверсепмлинг, путём копирования данных.

4 Реализация алгоритмов

Реализация KNN

```
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
```

Naive Bayes реализован 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
```

Вторая:

```
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 = []
        self.prob = []

        for j in zip(*np.unique(labels, return_counts=True)):
            self.prob.append(j[1] / len(labels))
            self.classes.append(j[0])
            self.mathexp.append(data[labels == j[0],].mean(axis = 0))
            self.variance.append(data[labels == j[0],].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)
                t[-1] *= self.prob[i]
                if t[-1] > maximum:
                    maximum = t[-1]
                    ans = self.classes[i]
            res[j] = ans
        return res
```

5 Метрики качества

В качестве метрик качества я использовал:

1. accuracy - доля правильно предсказанных объектов

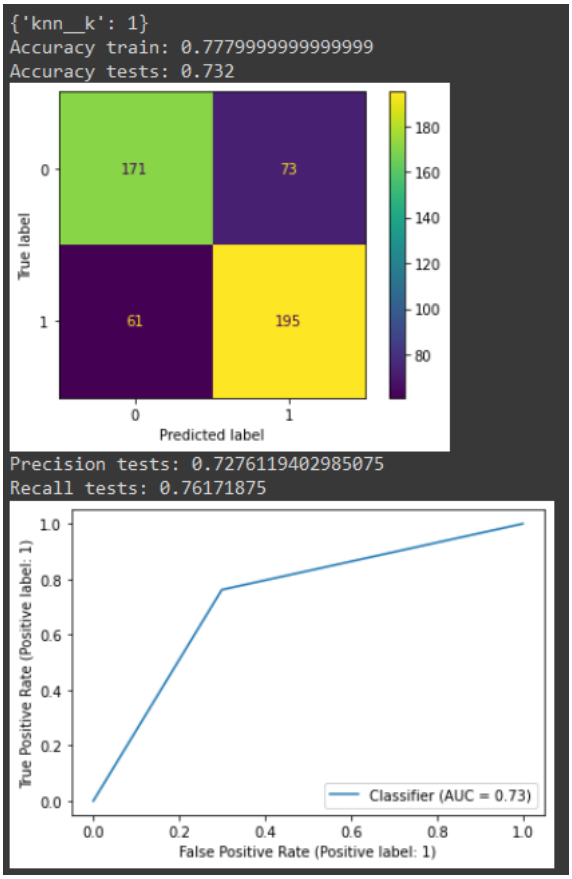
2. precision - доля правильно предсказанных положительных объектов среди всех объектов, предсказанных положительным классом

3. recall - доля правильно найденных положительных объектов среди всех объектов положительного класса.

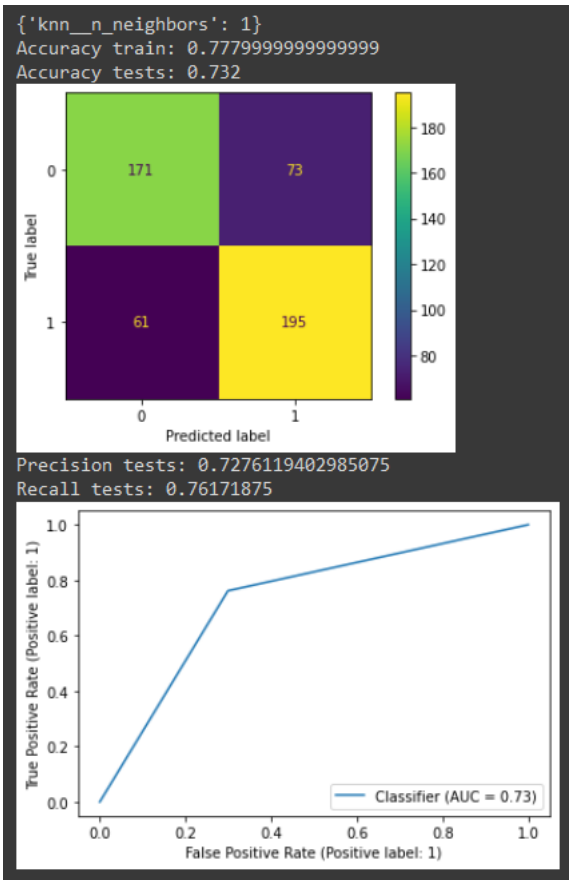
Использование всех этих метрик показывает более полную картину работы алгоритма.

6 Полученные результаты

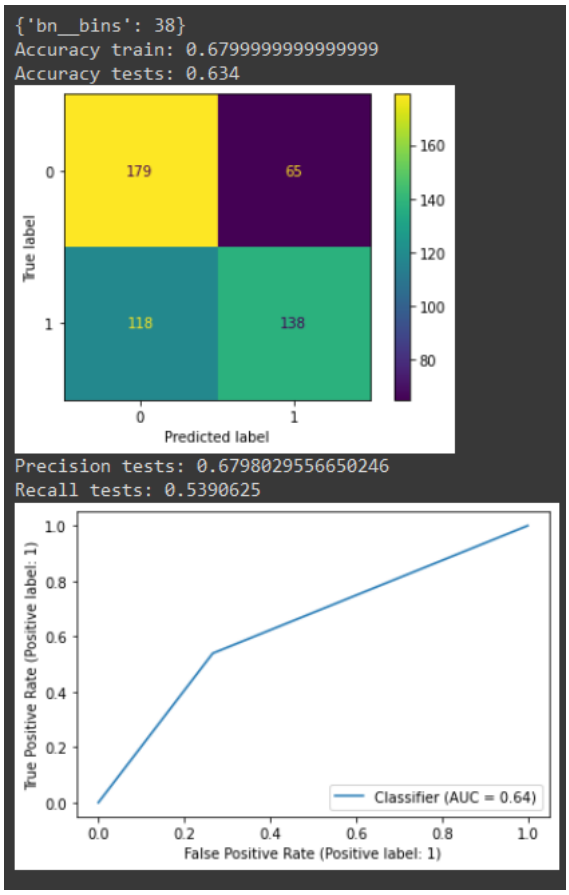
KNN показал себя неплохо, дав точность в районе 77 процентов. Моя реализация



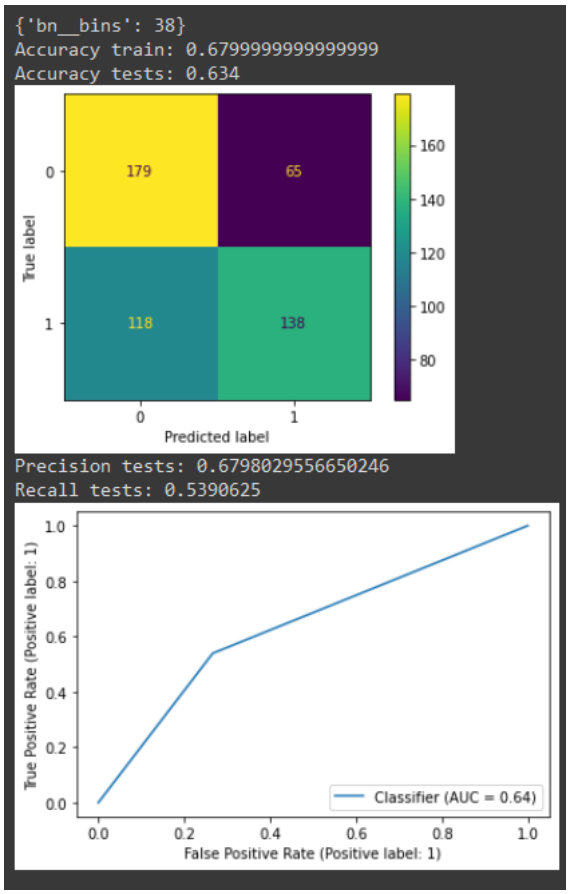
Реализация из sklearn дала точно такие же результаты



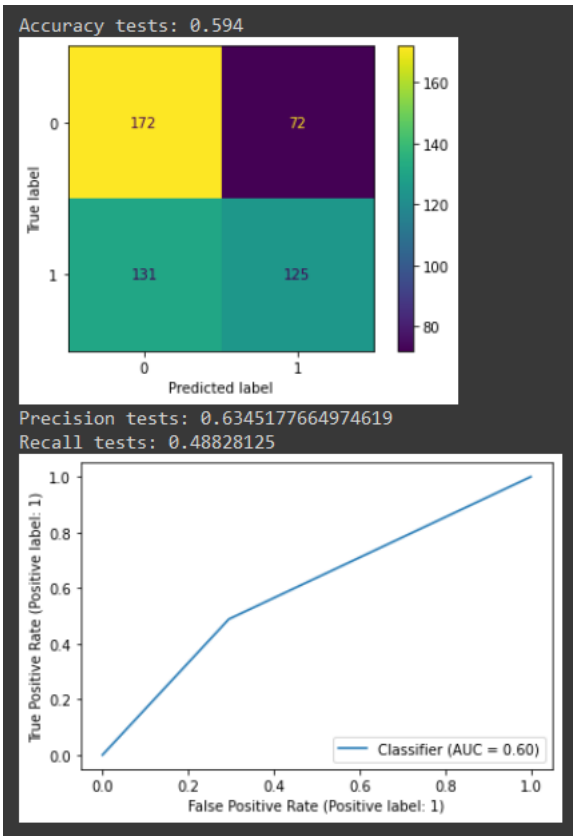
Байесовский классификатор показал себя хуже. Это связано с тем, что распределениях признаков по классам почти совпадают. Однако байесовский классификатор с использованием гистограмм показал себя лучше, дав точность в районе 68 процентов.



Байесовский классификатор с нормальным распределением, моя реализация



Байесовский классификатор с нормальным распределением, sklearn



7 Выводы

Сделав данный курсовой проект, я еще раз потренировался обрабатывать данные, реализовал несколько алгоритмов, а именно KNN и Naive Bayes. К сожалению, байесовский классификатор показал себя не очень хорошо на этих данных, однако он может себя проявить в другой раз. Я думаю, что на этих данных могут хорошо сработать дерево решений и случайный лес. Линейная модель здесь также не даст хороших результатов. Решение таких задач, как предсказывание болезней, может спасти чью-то жизнь, потому что очень часто на ранних этапах болезнь можно вылечить.