Практическая работа: "Бинарная логистическая регрессия"

Задача бинарной классификации

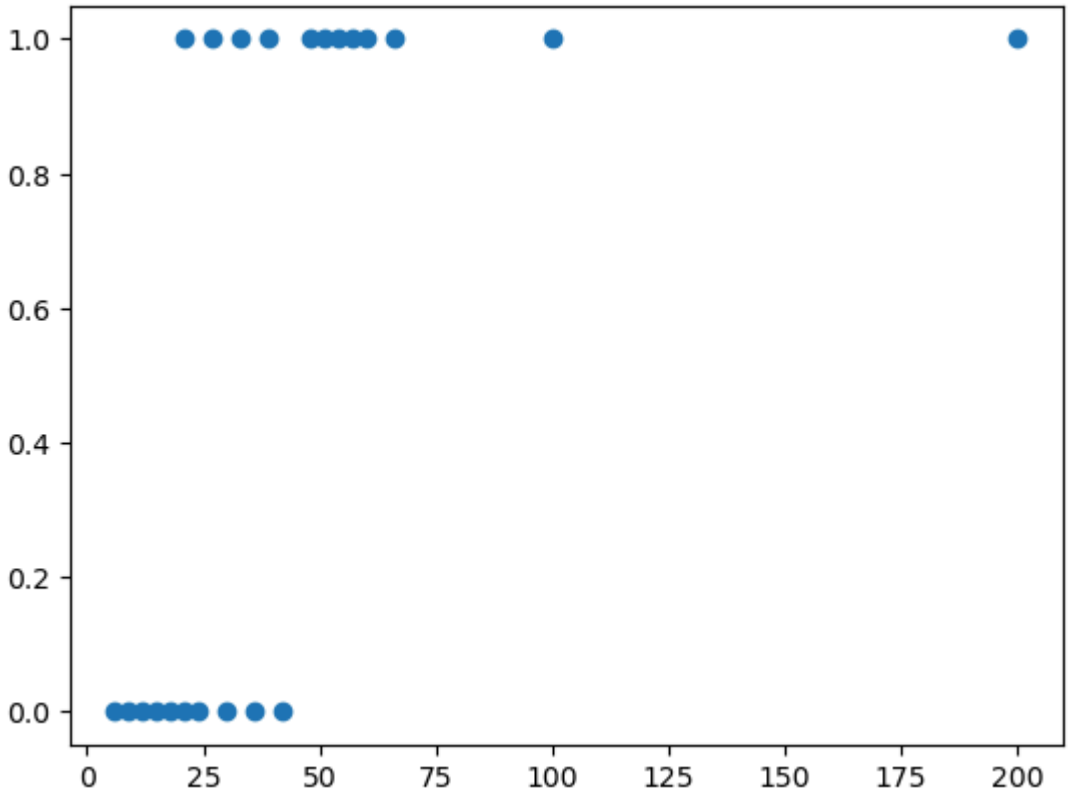
Некая фирма провела анализ зависимости расхода денежных средств на рекламу и эффекта от нее. Если эффект присутствовал, то в датасете ставилась 1, в противном случае - 0. Результаты были представоены в виде двух DataFrame x и y.
Произвести бинарную классификацию с использованием логистической регрессии.

Ввод [2]:
```
#Загрузим необходимые библиотеки
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

Ввод [30]:
```
#Исходные данные: x - расходы на рекламу, Y - объемы продаж
x = pd.DataFrame(np.array([39, 36, 60, 21, 27, 12, 51, 57, 54, 42, 18, 9, 48
Y = pd.DataFrame(np.array([1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0
#print(np.zeros(x.shape[1]).reshape(x.shape[1],1))
#print(np.zeros(x.shape[0]))
#a = np.zeros (x.shape[1]).reshape(x.shape[1],1)
#print(a)
#-x@a
#display(x, Y)
```

Ввод [31]:
```
plt.scatter(x,Y)
```

Out[31]:
```
<matplotlib.collections.PathCollection at 0x1aaf8dd2250>
```



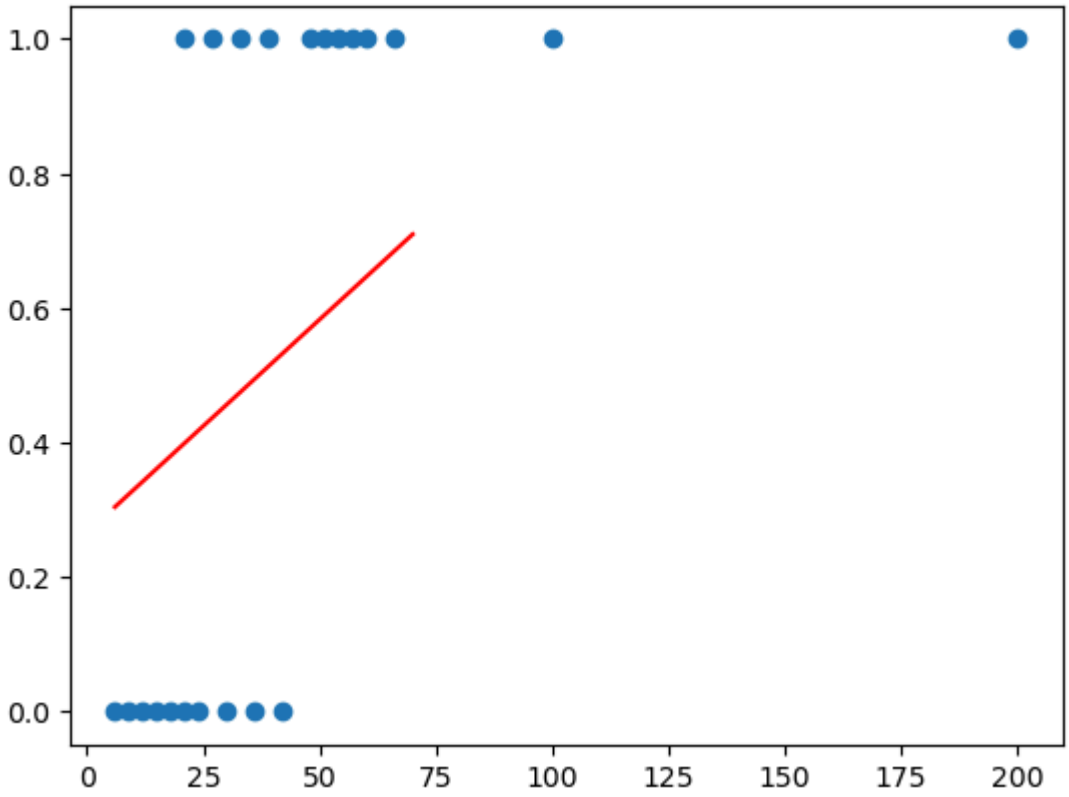
Попробуем разделить эти точки прямой линией, отделяющей один класс от другого. Модель начинает плохо работать в случае добавления новых точек на больших значениях x:

Ввод [32]:
```
a1 = ((x-x.mean())*(Y-Y.mean())).mean()/((x-x.mean())**2).mean()
a0 = Y.mean() - a1*x.mean()
print("Y = ", a0, " + ", a1, " * x")
x_space = np.linspace(6,70,20)
Y_pred = a0.item() + a1.item()*x_space
```

```
Y =  0    0.266493
dtype: float64  +  0    0.006333
dtype: float64  * x
```

Ввод [33]:
```
plt.plot(x_space, Y_pred, 'r')
plt.scatter(x, Y)
```

Out[33]:
```
<matplotlib.collections.PathCollection at 0x1aa800a43d0>
```

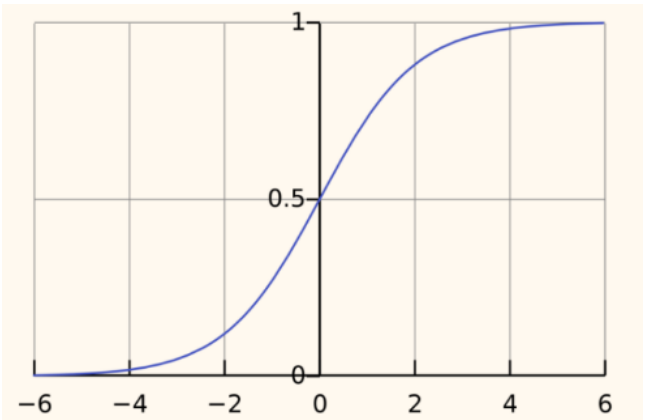


Функция логистической регрессии

Сигмоида

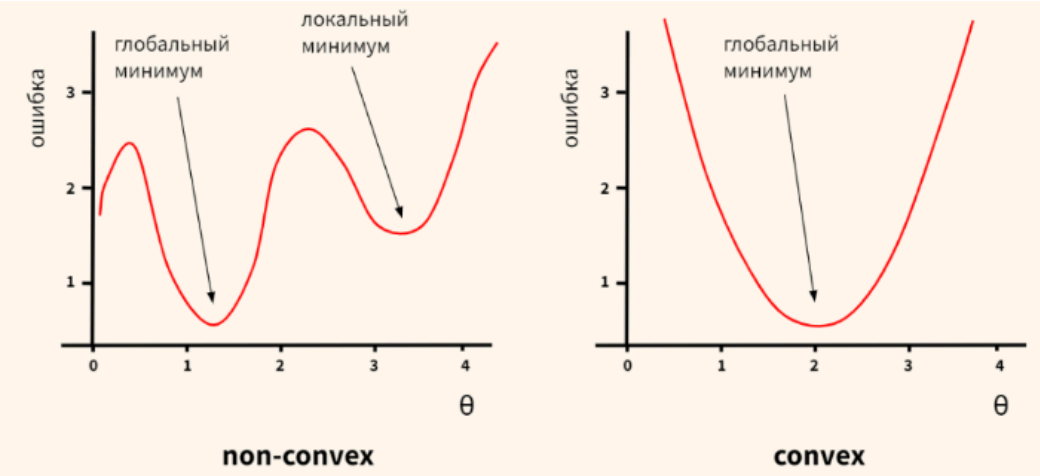
Возможное решение упомянутых выше сложностей — пропустить значение линейной регрессии через сигмоиду (sigmoid function), которая при любом значении x не выйдет из необходимого нам диапазона от 0 до 1.

$$g(x) = \frac{1}{1+e^{-x}}$$



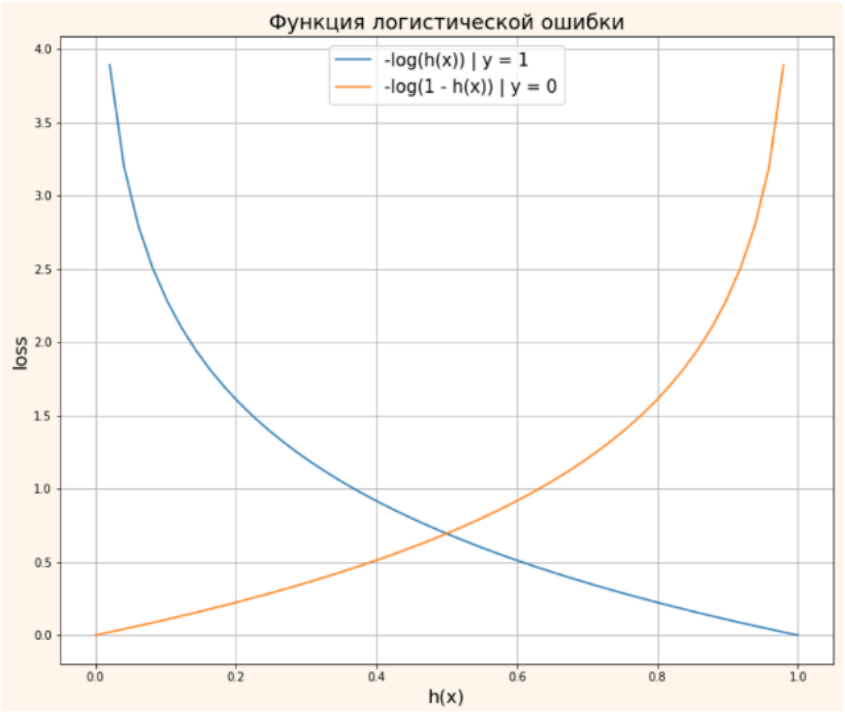
Logistic loss или функция кросс-энтропии

В модели логистической регрессии мы не можем использовать MSE. Дело в том, что если мы поместим результат сигмоиды (представляющей собою нелинейную функцию) в MSE, то на выходе получим невыпуклую функцию (non-convex), глобальный минимум которой довольно сложно найти.



Вместо MSE мы будем использовать функцию логистической ошибки, которую еще называют функцией бинарной кросс-энтропии (log loss, binary cross-entropy loss).

График и формула функции логистической ошибки



$$J(\theta) = -\frac{1}{n} \sum y * log(h_\theta(x)) + (1 - y) * log(1 - h_\theta(x))$$

Оценка качества модели

Матрица ошибок (confusion matrix)

	Прогноз: отрицательный (predicted negative)	**Прогноз: положительный** (predicted positive)
Факт: отрицательный (actual negative)	Истинно отрицательный (true negative, TN)	Ложноположительный (false positive, FP)
Факт: положительный (actual positive)	Ложноотрицательный (false negative, FN)	Истинно положительный (true positive, TP)

Доля правильно предсказанных значений называется accuracy. Чтобы ее посчитать, мы берем те значения, которые предсказаны верно (TP + TN) и делим на общее количество прогнозов.

Конкретизация задачи:

Реализовать класс модели логистической регрессии с использованием:

а) максимизации логарифмического правдоподобия методом градиентного подъема

б) минимизации функции потерь (кросс-энтропии) методом градиентного спуска

Сравнить результаты (время исполнения и достигнутые значения accuracy и f1) в случаях а и б при одном и том же числе шагов и скорости обучения.

Ввод [34]:
```
#Реализуем цикл градиентного спуска
class LogisticRegressionGD(object):
    def __init__(self):
        self.a = np.zeros(2).reshape(1, 2)

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

    def predict(self, x):
        return self.sigmoid(x)

    def coefs(self):
        return self.a

    def LogLikelihood(self, x, Y):
        return (Y * np.log(self.predict(x)) + (1 - Y) * np.log(1 - self.pred

    def CrossEntropy(self, x, Y):
        return (-Y * np.log(self.predict(x)) - (1 - Y) * np.log(1 - self.pre

    def accuracy(self, x, Y):
        return ((self.predict(x) > 0.5)==Y).mean()

    def fit(self, x, Y, alpha = 0.01, epsylon = 0.01, max_steps = 10000, Rty
        self.a = np.zeros (x.shape[1]).reshape(x.shape[1],1)
        steps, errors = [], []
        step = 0
        for _ in range(max_steps):
            if Rtype == "LL":
                new_error = self.LogLikelihood(x, Y)
                dT_a = x.T @(Y - self.predict(x)) / x.shape[0]
                self.a += alpha*dT_a

            elif Rtype == "CE":
                new_error = self.CrossEntropy(x, Y)
                #display(new_error)
                dT_a = -x.T @(Y - self.predict(x)) / x.shape[0]
                self.a -= alpha*dT_a
                print(self.a)
            step += 1
            print(step)
            steps.append(step)
            errors.append(new_error)
            #if new_error < epsylon:
            #    break
        return steps, errors
```

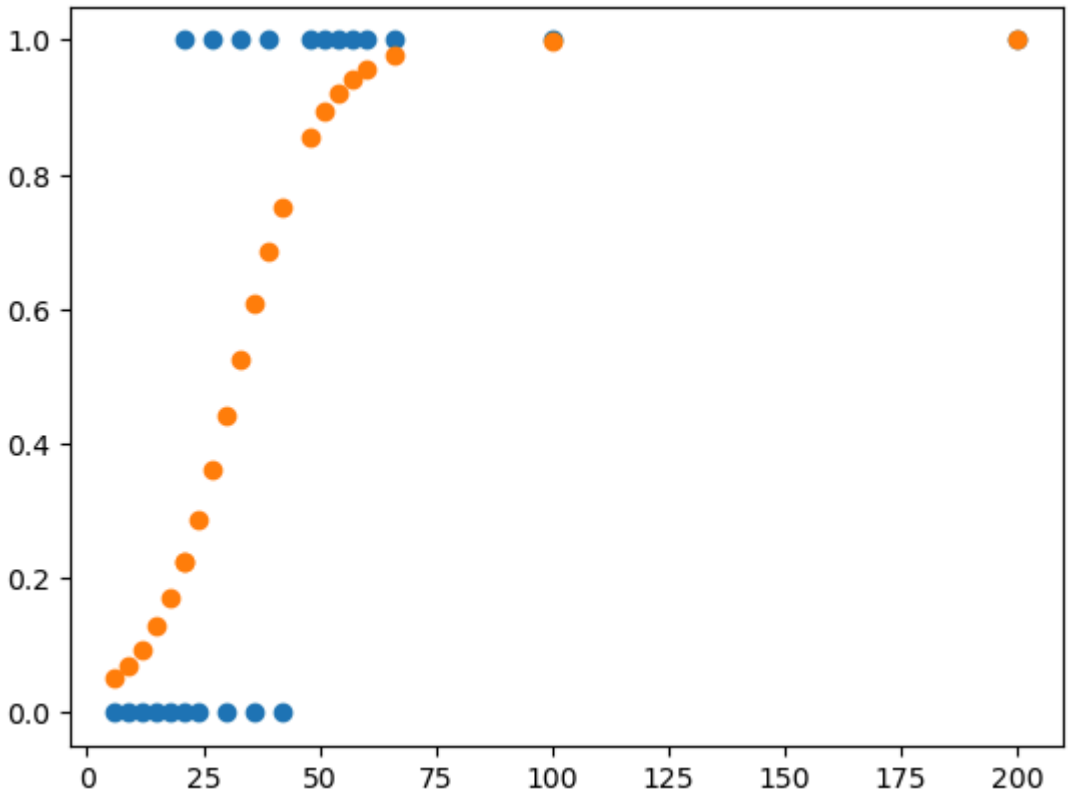
Ввод [35]:
```
x_ = x.copy()

import time

intercept = np.ones((x.shape[0], 1))
x_ = pd.DataFrame(np.concatenate((intercept, x), axis = 1))
```

Ввод [36]:
```
start_time_CE = time.time()
regr_CE = LogisticRegressionGD()
steps_CE, errors_CE =  regr_CE.fit(x_, Y, alpha = 0.01, epsylon = 0.01, max_

display(f"Время обучения: {time.time() - start_time_CE} сек.")

start_time_LL = time.time()
regr_LL = LogisticRegressionGD()
steps_LL, errors_LL =  regr_LL.fit(x_, Y, alpha = 0.01, epsylon = 0.01, max_

display(f"Время обучения: {time.time() - start_time_LL} сек.")
```

```
22
          0
0 -0.042427
1  0.075153
23
          0
0 -0.045805
1  0.000218
24
         0
0 -0.04526
1  0.12672
25
          0
0 -0.049222
1  0.037887
26
          0
0 -0.051500
1 -0.000345
```

Ввод [37]:
```
Y_Pred = regr_CE.predict(x_)
plt.scatter(x, Y)
plt.scatter(x, Y_Pred)
```

Out[37]: <matplotlib.collections.PathCollection at 0x1aa83cb1850>

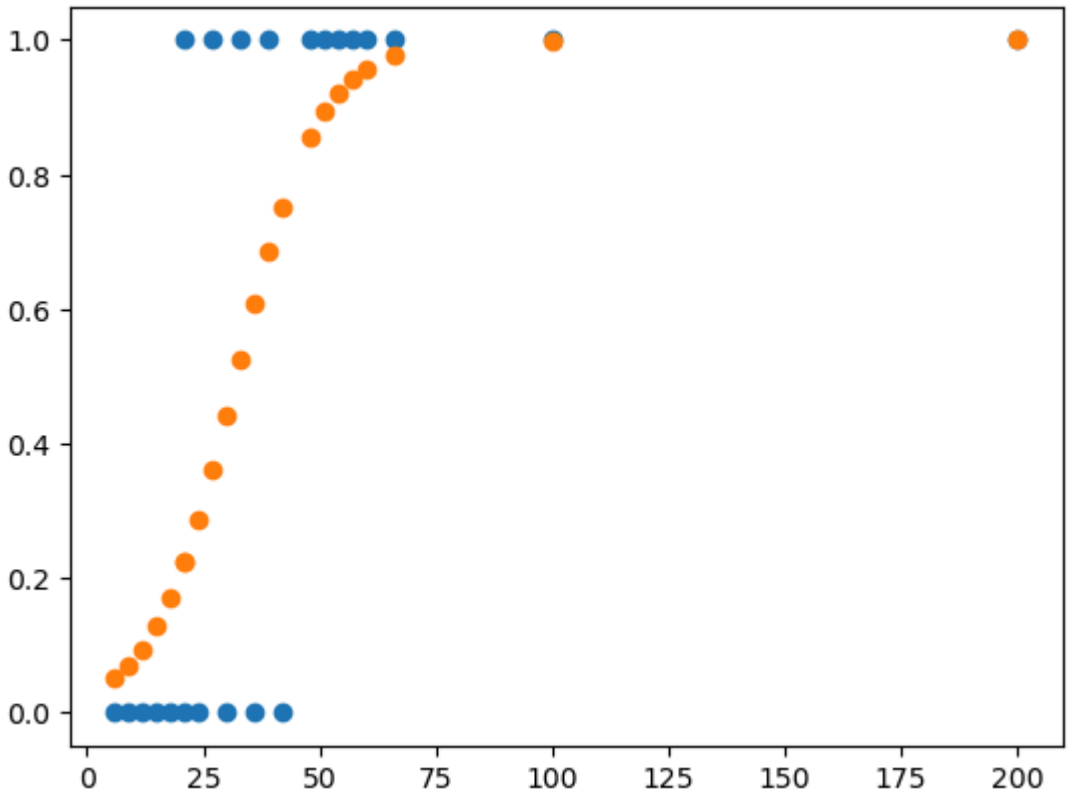


Ввод [38]:
```
display(regr_CE.coefs())
#display(x_)
```

	0
0	-3.598810
1	0.112152

Ввод [39]:
```
Y_Pred = regr_LL.predict(x_)
plt.scatter(x, Y)
plt.scatter(x, Y_Pred)
```

Out[39]:
```
<matplotlib.collections.PathCollection at 0x1aa8432d990>
```

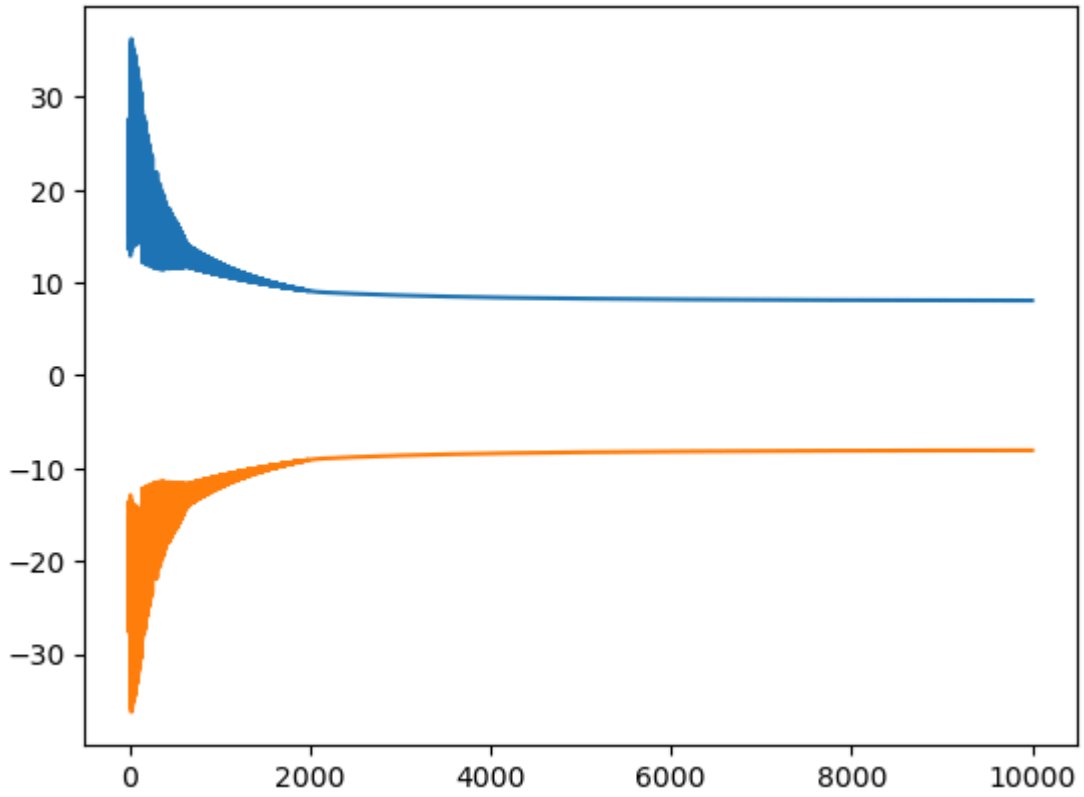


Ввод [40]:
```
display(regr_LL.coefs())
#display(x_)
```

	0
0	-3.598810
1	0.112152

Ввод [41]:
```
plt.plot(steps_CE, errors_CE)
plt.plot(steps_LL, errors_LL)
```

Out[41]:
```
[<matplotlib.lines.Line2D at 0x1aa8479af10>]
```



Ввод [42]:
```
print(regr_CE.accuracy(x_, Y))
print(regr_LL.accuracy(x_, Y))
```

```
0    0.818182
dtype: float64
0    0.818182
dtype: float64
```

Реализовать расчеты способами а и б для набора данных из файла insclass_train.csv/ Для 151 406 договоров страхования транспортных средств известны значения ряда признаков, в том числе пол, возраст, стаж вождения и коэффициент бонус-малус водителя, тип, марка, модель, год выпуска, страна – производитель, мощность и объем двигателя, а также признак target, равный 1, если заключение договора с клиентом является рисковым, и 0 в противном случае (файл insclass_train.csv).

Требуется построить модель, предсказывающую значение признака target для 22 624 договоров из тестового набора данных (файл insclass_test.csv).

В обучающем наборе данных для каждого договора известны следующие поля:

variable_1 - агрегированный коэффициент бонус-малус (повышающий или понижающий стоимость полиса в зависимости от аварийности в предыдущие периоды); variable_2 - индикатор расторжения договора по инициативе страхователя (клиента); variable_3 - индикатор расторжения договора по инициативе страховщика (страховой компании); variable_4 - идентификатор года выпуска транспортного средства; variable_5 - идентификатор страны - производителя транспортного средства; variable_6 - мощность двигателя в лошадиных силах; variable_7 - объем двигателя в

куб. см; variable_8 - идентификатор стороны расположения руля (левый или правый); variable_9 - пробег транспортного средства, покрываемый гарантией производителя; variable_10 - индикатор действия гарантии на транспортное средство; variable_11 - "мультидрайв" - индикатор допуска к управлению транспортным средством более одного водителя; variable_12 - возраст транспортного средства (в мес.); variable_13 - возраст водителя с максимальным стажем; variable_14 - коэффициент возраст-стаж; variable_15 - коэффициент краткосрочности; variable_16 - коэффициент мощности; variable_17 - коэффициент "мультидрайв"; variable_18 - территориальный коэффициент; variable_19 - коэффициент "КНДР"; variable_20 - идентификатор канала продаж; variable_21 - марка транспортного средства; variable_22 - модель транспортного средства; variable_23 - индикатор отечественных транспортных средств; variable_24 - пол водителя с максимальным коэффициентом "возраст-стаж"; variable_25 - индикатор пролонгации; variable_26 - индикатор совпадения собственника транспортного средства и водителя; variable_27 - стаж водителя с максимальным коэффициентом "возраст-стаж"; variable_28 - тип транспортного средства; target - класс риска, равный 1, если заключение договора с клиентом является рисковым, и 0 в противном случае.

Ввод [ ]:

Ввод [43]:
```
#Читаем файл, проверяем на пустые строки
df = pd.read_csv('insclass_train.csv',
        header=0,
        usecols=[ "variable_4","variable_6","variable_12","variable_13","var
#Проверить наличие пустых ячеек
df.isna().sum()
```

Out[43]:
```
variable_4        0
variable_6      111
variable_12    1528
variable_13    1657
variable_14    2927
variable_16      12
variable_17      12
variable_18      12
variable_19       0
variable_27    2067
target            0
dtype: int64
```

Ввод [44]:
```
#убрать пустые ячейки
df_cleaned = df.dropna()
df_cleaned.isna().sum()
```

Out[44]:
```
variable_4     0
variable_6     0
variable_12    0
variable_13    0
variable_14    0
variable_16    0
variable_17    0
variable_18    0
variable_19    0
variable_27    0
target         0
dtype: int64
```

Ввод [45]:
```
df_cleaned.head()
```

Out[45]:

	variable_4	variable_6	variable_12	variable_13	variable_14	variable_16	variable_17	varial
0	14	98.0	166.266987	49.041674	80.985224	98.648082	80.985224	80.9
1	7	106.0	80.338555	82.756867	80.985224	118.116608	80.985224	38.7
2	4	123.0	38.519899	35.842308	80.985224	162.514016	80.985224	273.4
3	9	102.0	109.845800	70.549602	80.985224	118.116608	80.985224	80.9
4	18	117.0	224.168209	42.499789	80.985224	118.116608	80.985224	118.1

Ввод [46]:
```
#данные по датасету
df_cleaned.describe()
```

Out[46]:

	variable_4	variable_6	variable_12	variable_13	variable_14	variabl
count	146529.000000	146529.000000	146529.000000	146529.000000	146529.000000	146529.00
mean	11.389220	117.574273	133.202504	49.988979	87.348600	132.11
std	6.912729	49.803903	87.917177	15.489248	31.932087	42.37
min	2.000000	1.000000	0.000000	20.731525	80.985224	28.13
25%	6.000000	86.000000	66.827923	38.519899	80.985224	98.64
50%	10.000000	107.000000	122.392869	46.438089	80.985224	118.11
75%	14.000000	136.000000	175.818287	59.293870	80.985224	162.51
max	80.000000	1975.000000	842.774591	115.578552	273.413449	214.25

Ввод [47]:
```
#Типы столбцов
df_cleaned.dtypes
```

Out[47]:
```
variable_4       int64
variable_6     float64
variable_12    float64
variable_13    float64
variable_14    float64
variable_16    float64
variable_17    float64
variable_18    float64
variable_19    float64
variable_27    float64
target           int64
dtype: object
```

Ввод [48]:
```
#разбиение на тренировочную, валидационную и тестовую выборки
train, validate, test = np.split(df_cleaned, [int(.7*len(df)), int(.85*len(d
```

Ввод [ ]:
```

```

Ввод [143]:
```
train
```

Out[143]:

	variable_4	variable_6	variable_12	variable_13	variable_14	variable_16	variable_17
0	14	98.0	166.266987	49.041674	80.985224	98.648082	80.985224
1	7	106.0	80.338555	82.756867	80.985224	118.116608	80.985224
2	4	123.0	38.519899	35.842308	80.985224	162.514016	80.985224
3	9	102.0	109.845800	70.549602	80.985224	118.116608	80.985224
4	18	117.0	224.168209	42.499789	80.985224	118.116608	80.985224
...	...	...	...	...	...	...	...
109489	6	123.0	65.580824	45.129796	80.985224	162.514016	80.985224
109490	6	102.0	55.478435	30.425246	80.985224	118.116608	80.985224
109491	3	125.0	19.323463	37.183600	80.985224	162.514016	80.985224
109492	21	71.0	257.798740	100.563999	80.985224	98.648082	80.985224
109493	13	140.0	164.130010	43.817058	80.985224	162.514016	80.985224

105984 rows × 11 columns

Ввод [144]:
```
X_t = train.iloc[:, train.columns != 'target'].values
y_t = train.iloc[:, train.columns == 'target'].values.reshape(-1)
#y_t = train.iloc[:, train.columns == 'target'].values
X_t = np.array(X_t, dtype=np.float64)
y_t = np.array(y_t, dtype=np.int64)
```

Ввод [ ]:
```

```

Ввод [145]:
```
X_t1000 = X_t[:1000,3]
y_t1000 = y_t[:1000]
x = pd.DataFrame(X_t1000)
Y = pd.DataFrame(y_t1000)
#plt.scatter(x,Y)
```

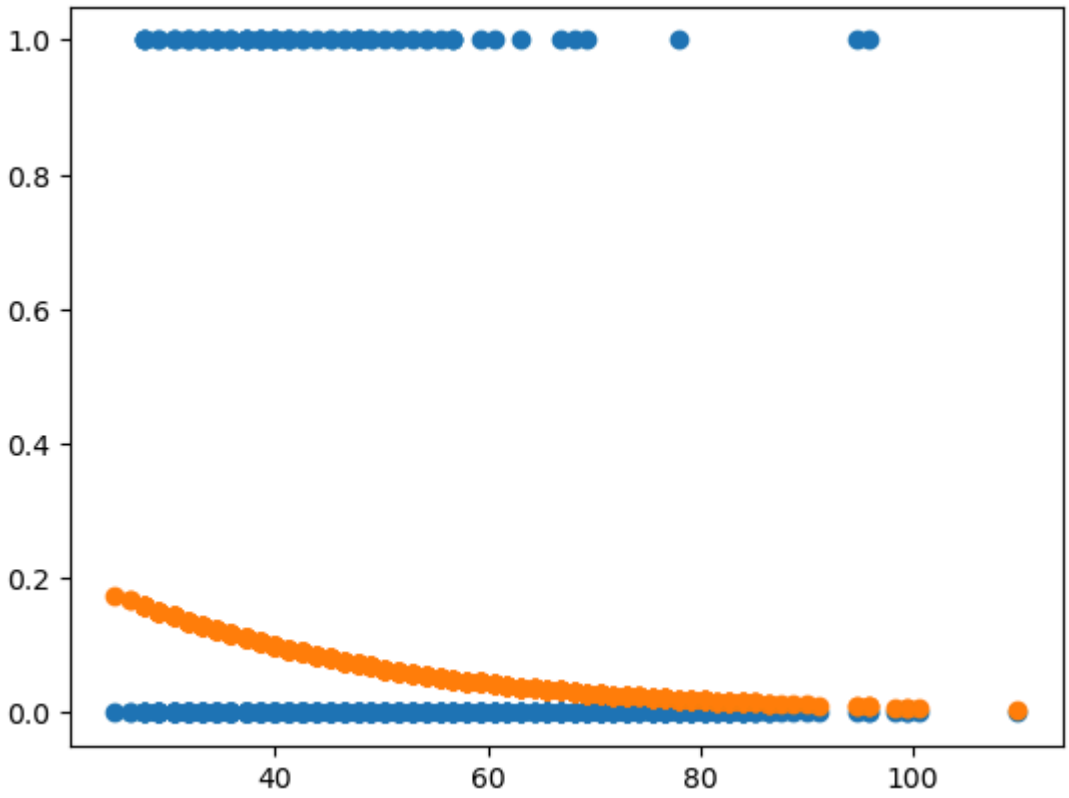
Ввод [146]:
```
x_ = x.copy()

import time

intercept = np.ones((x.shape[0], 1))
x_ = pd.DataFrame(np.concatenate((intercept, x), axis = 1))
```

Ввод [147]:
```
start_time_CE = time.time()
regr_CE = LogisticRegressionGD()
steps_CE, errors_CE =  regr_CE.fit(x_, Y, alpha = 0.01, epsylon = 0.01, max_

display(f"Время обучения: {time.time() - start_time_CE} сек.")

start_time_LL = time.time()
regr_LL = LogisticRegressionGD()
steps_LL, errors_LL =  regr_LL.fit(x_, Y, alpha = 0.01, epsylon = 0.01, max_

display(f"Время обучения: {time.time() - start_time_LL} сек.")
```

```
          0
0 -0.00425
1 -0.21435
1
          0
0 -0.003503
1 -0.181219
2
          0
0 -0.002761
1 -0.148286
3
          0
0 -0.002038
1 -0.115989
4
          0
0 -0.001371
1 -0.085778
```

Ввод [148]:
```
Y_Pred = regr_CE.predict(x_)
plt.scatter(x, Y)
plt.scatter(x, Y_Pred)
```

Out[148]:
```
<matplotlib.collections.PathCollection at 0x1aa90f6f350>
```



Ввод [149]:
```
errors_CE
```

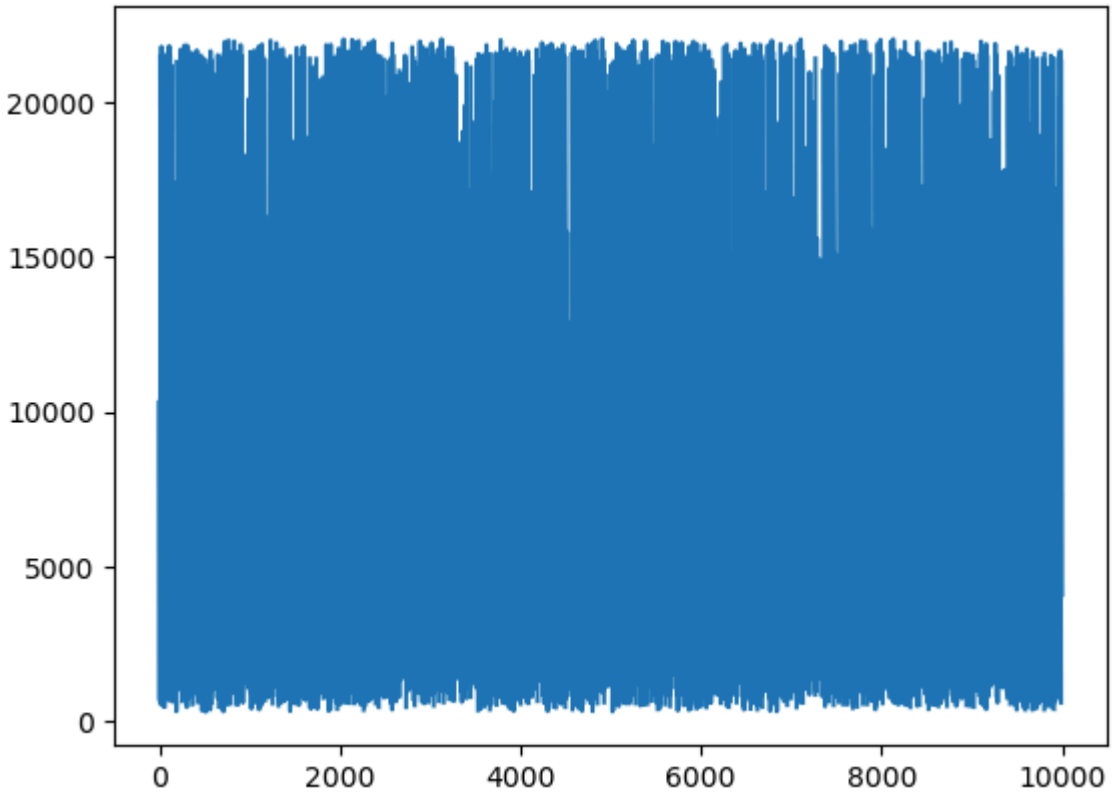
```
dtype: float64,
0    264.520064
dtype: float64,
0    264.519751
dtype: float64,
0    264.519439
dtype: float64,
0    264.519127
dtype: float64,
0    264.518815
dtype: float64,
0    264.518503
dtype: float64,
0    264.518191
dtype: float64,
0    264.517878
dtype: float64,
0    264.517566
dtype: float64,
0    264.517254
```

Ввод [150]:
```
display(regr_CE.coefs())
```

	0
0	-0.458247
1	-0.043767

Ввод [138]:
```
plt.plot(steps_CE, errors_CE)
```

Out[138]:
```
[<matplotlib.lines.Line2D at 0x1aa8353fa10>]
```



Ввод [ ]: