

Урок 3. Логистическая регрессия. Log Loss

План урока

- [Логистическая регрессия](#)
- [Логистическая функция потерь](#)
- [Пример](#)
- [Домашние задания](#)

In [1]:

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 import seaborn as sns
4 sns.set(style="whitegrid")
5 sns.set_context("paper", font_scale=2)
```

C:\ProgramData\Anaconda3\lib\site-packages\statsmodels\tools_testing.py:19:
FutureWarning: pandas.util.testing is deprecated. Use the functions in the public API at pandas.testing instead.
import pandas.util.testing as tm

In [2]:

```
1 %matplotlib inline
2 plt.style.use('seaborn-ticks')
3 plt.rcParams.update({'font.size': 14})
```

Логистическая регрессия



$$\vec{y} = X\vec{w}$$

Если предсказывать с помощью линейной регрессии, то могут появляться такие числа:

-1, -0.2, 1.3, 0.05 ...

$p = [0, 1]$ - вероятность

$$odds = \frac{p}{1-p} = \frac{p_1}{p_0} - \text{шансы}$$

In [3]:

```
1 0.5 / (1 - 0.5)
```

Out[3]:

1.0

In [4]:

```
1 0.01 / (1 - 0.01)
```

Out[4]:

0.010101010101010102

In [5]:

```
1 0.99 / (1 - 0.99)
```

Out[5]:

98.999999999999991

In [6]:

```
1 p = 1e-10
2 np.log(p / (1 - p))
```

Out[6]:

-23.025850929840455

In [7]:

```
1 p = 0.99
2 np.log(p / (1 - p))
```

Out[7]:

4.595119850134589

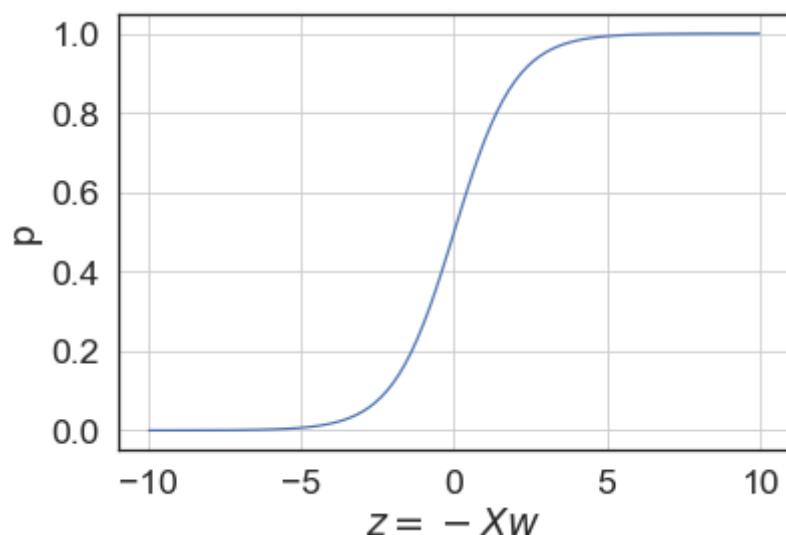
$$\begin{aligned} X\vec{w} &= \ln\left(\frac{\vec{p}}{1-\vec{p}}\right) \\ \exp(X\vec{w}) &= \frac{\vec{p}}{1-\vec{p}} \\ (1-\vec{p})\exp(X\vec{w}) &= \vec{p} \\ \exp(X\vec{w}) - \vec{p}\exp(X\vec{w}) &= \vec{p} \\ \exp(X\vec{w}) &= \vec{p}(1 + \exp(X\vec{w})) \\ \vec{p} &= \frac{\exp(X\vec{w})}{1 + \exp(X\vec{w})} \\ \vec{p} &= \frac{1}{1 + \exp(-X\vec{w})} \end{aligned}$$

In [8]:

```
1 def sigmoid(z):
2     res = 1 / (1 + np.exp(-z))
3     return res
```

In [9]:

```
1 # сигмоида
2 z = np.linspace(-10, 10, 101)
3 probabilities = sigmoid(z)
4 plt.plot(z, probabilities)
5 plt.xlabel('$z=-Xw$')
6 plt.ylabel('p')
7 plt.grid();
```



Логистическая функция потерь

$$\text{Logloss} = -y \ln(p) - (1 - y) \ln(1 - p)$$

$y = 0/1$ - истинный класс

$p = [0, 1]$ - ответ алгоритма

In [10]:

```
1 def calc_logloss(y, y_pred):
2     err = np.mean(- y * np.log(y_pred) - (1.0 - y) * np.log(1.0 - y_pred))
3     return err
```

In [11]:

```
1 # Пример применения
2 y_true = 1
3 y_pred = 0.99
4 calc_logloss(y_true, y_pred)
```

Out[11]:

0.01005033585350145

Пример

In [12]:

```
1 X = np.array([[1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
2               [1, 1, 2, 1, 3, 0, 5, 10, 1, 2], # стаж репетитора
3               [500, 700, 750, 600, 1450,      # средняя стоимость занятия
4               800, 1500, 2000, 450, 1000],
5               [1, 1, 2, 1, 2, 1, 3, 3, 1, 2]], # квалификация репетитора
6               dtype = np.float64).T
7
8 y = np.array([0, 0, 1, 0, 1, 0, 1, 0, 1, 1]) # подходит или нет репетитор
9 X
```

Out[12]:

```
array([[1.00e+00, 1.00e+00, 5.00e+02, 1.00e+00],
       [1.00e+00, 1.00e+00, 7.00e+02, 1.00e+00],
       [1.00e+00, 2.00e+00, 7.50e+02, 2.00e+00],
       [1.00e+00, 1.00e+00, 6.00e+02, 1.00e+00],
       [1.00e+00, 3.00e+00, 1.45e+03, 2.00e+00],
       [1.00e+00, 0.00e+00, 8.00e+02, 1.00e+00],
       [1.00e+00, 5.00e+00, 1.50e+03, 3.00e+00],
       [1.00e+00, 1.00e+01, 2.00e+03, 3.00e+00],
       [1.00e+00, 1.00e+00, 4.50e+02, 1.00e+00],
       [1.00e+00, 2.00e+00, 1.00e+03, 2.00e+00]])
```

In [13]:

```
1 def standard_scaler(x):
2     res = (x - x.mean()) / x.std()
3     return res
4
5 X_st = X.copy()
6 X_st[:, 2] = standard_scaler(X[:, 2])
7 X_st[:, 2]
```

Out[13]:

```
array([-0.97958969, -0.56713087, -0.46401617, -0.77336028,  0.97958969,
        -0.36090146,  1.08270439,  2.11385144, -1.08270439,  0.05155735])
```

In [14]:

```
1 def eval_LR_model(X, y, iterations, alpha=1e-4):
2     np.random.seed(42)
3     w = np.random.randn(X.shape[1])
4     n = X.shape[0]
5     for i in range(1, iterations + 1):
6         z = np.dot(X, w)
7         y_pred = sigmoid(z)
8         err = calc_logloss(y, y_pred)
9         w -= alpha * (1/n * np.dot(X.T, (y_pred - y)))
10        if i % (iterations / 10) == 0:
11            print(i, w, err)
12    return w
```

In [15]:

```
1 w = eval_LR_model(X_st, y, iterations=1000, alpha=1e-2)

100 [ 0.25769171 -0.68297243  0.68841418  1.2405903 ] 0.5908596023453087
200 [ 0.14805533 -0.69317972  0.77965488  1.21287192] 0.5687278067099053
300 [ 0.06257746 -0.68605867  0.85181735  1.21283095] 0.5560473747533587
400 [-0.00887935 -0.68415387  0.90713736  1.22443854] 0.5476800278385013
500 [-0.0701621  -0.68661758  0.94984062  1.24404686] 0.541676599615156
600 [-0.1239972  -0.69202069  0.98318257  1.26924759] 0.536984186497779
700 [-0.17239277 -0.69933957  1.00953865  1.29824779] 0.5330427663756571
800 [-0.21681092 -0.7078635  1.03062761  1.32973537] 0.52955515035808
900 [-0.25830834 -0.71709934  1.04769231  1.36276535] 0.5263621224044184
1000 [-0.29764618 -0.72670545  1.061634  1.39666497] 0.5233765331724747
```

Домашние задания

In [16]:

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 import seaborn as sns
4 sns.set(style="whitegrid")
5 sns.set_context("paper", font_scale=2)
```

In [17]:

```
1 %matplotlib inline
2 plt.style.use('seaborn-ticks')
3 plt.rcParams.update({'font.size': 14})
```

1. *Измените функцию `calc_logloss` так, чтобы нули по возможности не попадали в `np.log` (как вариант - `np.clip`).

In [18]:

```
1 def calc_logloss(y, y_pred):
2     err = np.mean(- y * np.log(np.clip(y_pred, 1e-50, 1 - (1e-10)))) - (1.0 - y) * np.log
3     return err
```

In [19]:

```
1 # Пример применения
2 y_true = 1
3 y_pred = 1
4 calc_logloss(y_true, y_pred)
```

Out[19]:

1.000000082790371e-10

In [20]:

```
1 # Пример применения
2 y_true = 1
3 y_pred = 0
4 calc_logloss(y_true, y_pred)
```

Out[20]:

115.12925464970229

Сделал так, чтобы в pr.log не попадали не 0 не 1

2. Подберите аргументы функции eval_LR_model для логистической регрессии таким образом, чтобы log loss был минимальным.

In [21]:

```
1 X = np.array([[1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
2               [1, 1, 2, 1, 3, 0, 5, 10, 1, 2], # стаж репетитора
3               [500, 700, 750, 600, 1450,
4               800, 1500, 2000, 450, 1000], # средняя стоимость занятия
5               [1, 1, 2, 1, 2, 1, 3, 3, 1, 2]], # квалификация репетитора
6               dtype = np.float64).T
7
8 y = np.array([0, 0, 1, 0, 1, 0, 1, 0, 1, 1]) # подходит или нет репетитор
9 X
```

Out[21]:

```
array([[1.00e+00, 1.00e+00, 5.00e+02, 1.00e+00],
       [1.00e+00, 1.00e+00, 7.00e+02, 1.00e+00],
       [1.00e+00, 2.00e+00, 7.50e+02, 2.00e+00],
       [1.00e+00, 1.00e+00, 6.00e+02, 1.00e+00],
       [1.00e+00, 3.00e+00, 1.45e+03, 2.00e+00],
       [1.00e+00, 0.00e+00, 8.00e+02, 1.00e+00],
       [1.00e+00, 5.00e+00, 1.50e+03, 3.00e+00],
       [1.00e+00, 1.00e+01, 2.00e+03, 3.00e+00],
       [1.00e+00, 1.00e+00, 4.50e+02, 1.00e+00],
       [1.00e+00, 2.00e+00, 1.00e+03, 2.00e+00]])
```

In [22]:

```
1 def standard_scaler(x):
2     res = (x - x.mean()) / x.std()
3     return res
4
5 X = X.copy()
6 X[:, 2] = standard_scaler(X[:, 2])
7 X[:, 2]
```

Out[22]:

```
array([-0.97958969, -0.56713087, -0.46401617, -0.77336028,  0.97958969,
        -0.36090146,  1.08270439,  2.11385144, -1.08270439,  0.05155735])
```

In [23]:

```
1 def eval_LR_model(X, y, iterations, alpha=1e-4):
2     np.random.seed(42)
3     w = np.random.randn(X.shape[1])
4     n = X.shape[0]
5     for i in range(1, iterations + 1):
6         z = np.dot(X, w)
7         y_pred = sigmoid(z)
8         err = calc_logloss(y, y_pred)
9         w -= alpha * (1/n * np.dot(X.T, (y_pred - y)))
10        if i % (iterations / 10) == 0:
11            print(i, w, err)
12    return w
```

In [24]:

```
1 def sigmoid(x):
2     return 1 / (1 + np.exp(-x))
```

In [25]:

```
1 eval_LR_model(X, y, iterations=10000, alpha=300)
```

```
1000 [-1867.42989341 -279.99246887 -417.4995521 1706.33264797] 4.75730916
2495939e-06
2000 [-1867.67501322 -279.51790769 -417.75362765 1706.08752816] 3.37111738
96614227e-06
3000 [-1867.83960475 -279.15616123 -417.95533598 1705.92293662] 2.61433535
8705259e-06
4000 [-1867.97002745 -278.86763556 -418.11652491 1705.79251393] 2.13505791
24414726e-06
5000 [-1868.07837678 -278.62778208 -418.25054843 1705.6841646 ] 1.80429059
26388712e-06
6000 [-1868.17108179 -278.42253903 -418.36523608 1705.59145958] 1.56226561
80626814e-06
7000 [-1868.25209919 -278.24316767 -418.46546756 1705.51044219] 1.37749336
23284842e-06
8000 [-1868.32404947 -278.08386985 -418.55448224 1705.43849191] 1.23180700
63030248e-06
9000 [-1868.38875959 -277.94060148 -418.63453979 1705.37378179] 1.11399060
30068374e-06
10000 [-1868.44755442 -277.81042948 -418.70727918 1705.31498696] 1.0167449
136605736e-06
```

Out[25]:

```
array([-1868.44755442, -277.81042948, -418.70727918, 1705.31498696])
```

Опыт показывает, что при постоянном увеличении количества итераций и значения α \log loss постоянно уменьшается

3. Создайте функцию `calc_pred_proba`, возвращающую предсказанную вероятность класса 1 (на вход подаются значения признаков X и веса, которые уже посчитаны функцией `eval_LR_model`, на выходе - массив `y_pred_proba`).

In [26]:

```
1 X = np.array([[1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
2               [1, 1, 2, 1, 3, 0, 5, 10, 1, 2], # стаж репетитора
3               [500, 700, 750, 600, 1450,
4               800, 1500, 2000, 450, 1000], # средняя стоимость занятия
5               [1, 1, 2, 1, 2, 1, 3, 3, 1, 2]], # квалификация репетитора
6               dtype = np.float64).T
7
8 y = np.array([0, 0, 1, 0, 1, 0, 1, 0, 1, 1]) # подходит или нет репетитор
9 x
```

Out[26]:

```
array([[1.00e+00, 1.00e+00, 5.00e+02, 1.00e+00],
       [1.00e+00, 1.00e+00, 7.00e+02, 1.00e+00],
       [1.00e+00, 2.00e+00, 7.50e+02, 2.00e+00],
       [1.00e+00, 1.00e+00, 6.00e+02, 1.00e+00],
       [1.00e+00, 3.00e+00, 1.45e+03, 2.00e+00],
       [1.00e+00, 0.00e+00, 8.00e+02, 1.00e+00],
       [1.00e+00, 5.00e+00, 1.50e+03, 3.00e+00],
       [1.00e+00, 1.00e+01, 2.00e+03, 3.00e+00],
       [1.00e+00, 1.00e+00, 4.50e+02, 1.00e+00],
       [1.00e+00, 2.00e+00, 1.00e+03, 2.00e+00]])
```

In [27]:

```
1 def eval_LR_model(X, y, iterations, alpha=1e-4):
2     np.random.seed(42)
3     w = np.random.randn(X.shape[1])
4     n = X.shape[0]
5     for i in range(1, iterations + 1):
6         z = np.dot(X, w)
7         y_pred = sigmoid(z)
8         err = calc_logloss(y, y_pred)
9         w -= alpha * (1/n * np.dot(X.T, (y_pred - y)))
10    return w
```

In [28]:

```
1 def standard_scaler(x):
2     res = (x - x.mean()) / x.std()
3     return res
4
5 X = X.copy()
6 X[:, 2] = standard_scaler(X[:, 2])
7 X[:, 2]
```

Out[28]:

```
array([-0.97958969, -0.56713087, -0.46401617, -0.77336028,  0.97958969,
       -0.36090146,  1.08270439,  2.11385144, -1.08270439,  0.05155735])
```

Далее транспонируем матрицы, так как нам удобнее работать со строками

In [29]:

```
1 X_tr = X.transpose()
2 y_tr = y.reshape(1, y.shape[0])
```

Реализуем функцию, возвращающую предсказанную вероятность класса 1

In [30]:

```
1 def calc_pred_proba(w, X):
2
3     m = X.shape[1]
4
5     y_predicted = np.zeros((1, m))
6     w = w.reshape(X.shape[0], 1)
7
8     A = sigmoid(np.dot(w.T, X))
9
10    return A
```

In [31]:

```
1 w = eval_LR_model(X, y, iterations=1000, alpha=0.5)
2
3 pred_proba = calc_pred_proba(w, X_tr)
4 print(f"Предсказанные вероятности класса 1: {pred_proba[0]}")
```

Предсказанные вероятности класса 1: [0.27142194 0.17963641 0.98122786 0.22216
112 0.71033953 0.35778609
0.99436302 0.10401932 0.29847465 0.96415991]

4. Создайте функцию calc_pred, возвращающую предсказанный класс (на вход подаются значения признаков X и веса, которые уже посчитаны функцией eval_LR_model, на выходе - массив y_pred).

In [32]:

```
1 def calc_pred(w, X):
2
3     m = X.shape[1]
4
5     y_predicted = np.zeros((1, m))
6     w = w.reshape(X.shape[0], 1)
7
8     A = sigmoid(np.dot(w.T, X))
9
10    for i in range(A.shape[1]):
11        if (A[:,i] > 0.5):
12            y_predicted[:, i] = 1
13        elif (A[:,i] <= 0.5):
14            y_predicted[:, i] = 0
15
16    return y_predicted
```

In [33]:

```
1 w = eval_LR_model(X, y, iterations=1000, alpha=0.5)
2
3 y_predicted = calc_pred(w, X_tr)
4 print(f"Предсказанные вероятности класса 1: {y_predicted[0]}")
```

Предсказанные вероятности класса 1: [0. 0. 1. 0. 1. 0. 1. 0. 0. 1.]

5. Посчитайте accuracy, матрицу ошибок, precision и recall, а также F1-score.

accuracy

In [34]:

```
1 y_predicted = y_predicted[0].copy()
2 y_predicted
```

Out[34]:

array([0., 0., 1., 0., 1., 0., 1., 0., 0., 1.])

In [35]:

```
1 y_tr = y_tr[0].copy()
2 y_tr
```

Out[35]:

array([0, 0, 1, 0, 1, 0, 1, 0, 1, 1])

In [36]:

```
1 accuracy = 100.0 - np.mean(np.abs(y_predicted - y_tr)*100.0)
2 accuracy
```

Out[36]:

90.0

confusion_matrix

Напишем код для построения confusion_matrix по следующему принципу:

	$y = 1$	$y = 0$
$a_1(x) = 1$	0	0
$a_1(x) = 0$	0	0

In [37]:

```
1 confusion_matrix_ = np.array([[0, 0], [0, 0]])
2 for i in range(len(y_tr)):
3     if (y_tr[i] == 1) & (y_predicted[i] == 1):
4         confusion_matrix_[0][0] += 1
5
6     if (y_tr[i] == 0) & (y_predicted[i] == 0):
7         confusion_matrix_[1][1] += 1
8
9     if (y_tr[i] == 1) & (y_predicted[i] == 0):
10        confusion_matrix_[1][0] += 1
11
12    if (y_tr[i] == 0) & (y_predicted[i] == 1):
13        confusion_matrix_[0][1] += 1
```

In [38]:

```
1 confusion_matrix_
```

Out[38]:

```
array([[4, 0],
       [1, 5]])
```

Проверим

In [39]:

```
1 from sklearn.metrics import confusion_matrix
2 confusion_matrix(y_tr, y_predicted)
```

Out[39]:

```
array([[5, 0],
       [1, 4]], dtype=int64)
```

Если помнить, что ось истинных значений вертикальная, а предсказаний - горизонтальная, то видно, что наши расчеты confusion_matrix - верные.

precision

$$precision(a, X) = \frac{TP}{TP + FP}.$$

In [40]:

```
1 precision = confusion_matrix_[0][0] / (confusion_matrix_[0][0] + confusion_matrix_[0][1])
2 precision
```

Out[40]:

```
1.0
```

Проверим

In [41]:

```
1 from sklearn.metrics import precision_score
2 precision_score(y_tr, y_predicted)
```

Out[41]:

1.0

recall

$$recall(a, X) = \frac{TP}{TP + FN},$$

In [42]:

```
1 recall = confusion_matrix_[0][0] / (confusion_matrix_[0][0] + confusion_matrix_[1][0])
2 recall
```

Out[42]:

0.8

Проверим

In [43]:

```
1 from sklearn.metrics import recall_score
2 recall_score(y_tr, y_predicted)
```

Out[43]:

0.8

F1-score

$$F1 - score = \frac{2 \cdot precision \cdot recall}{precision + recall}.$$

In [44]:

```
1 F_score = 2 * precision * recall / (precision + recall)
2 F_score
```

Out[44]:

0.8888888888888889

Проверим

In [45]:

```
1 from sklearn.metrics import f1_score
2 f1_score(y_tr, y_predicted)
```

Out[45]:

0.8888888888888889

6. Могла ли модель переобучиться? Почему?

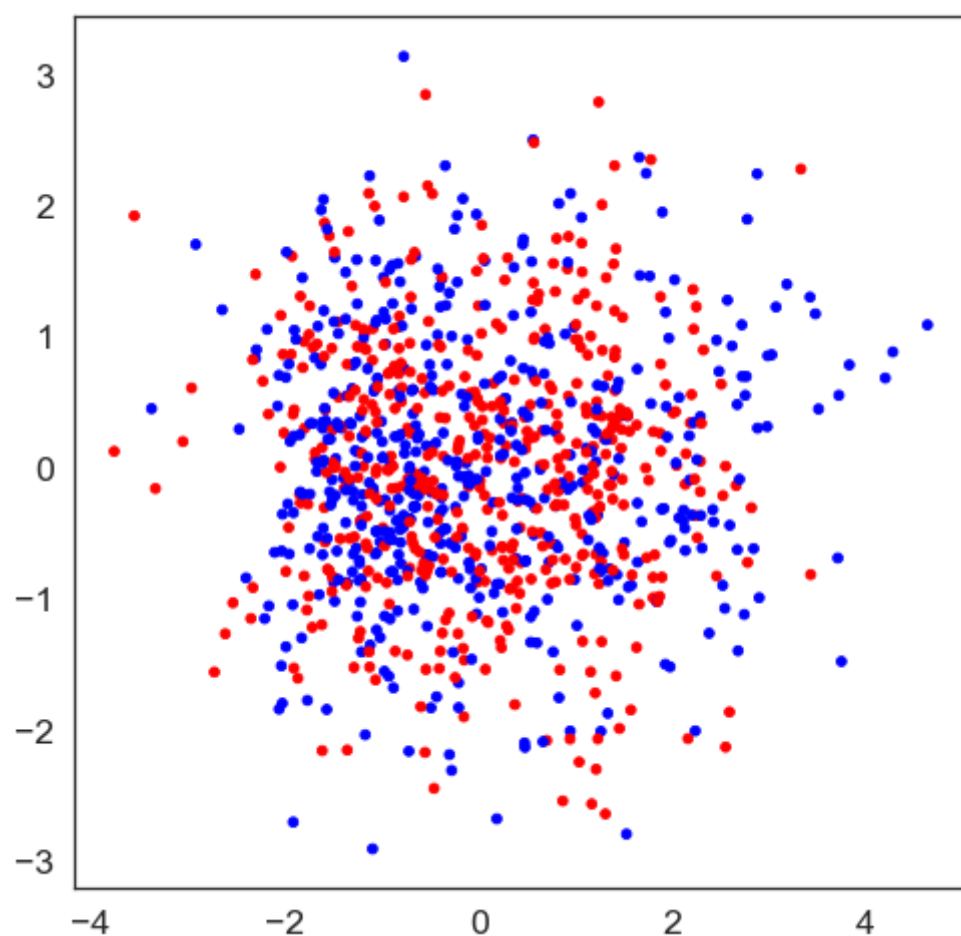
Да, модель может переобучиться также как и при обучении линейной регрессии. Это происходит потому, что вектор коэффициентов подбирается под конкретный набор данных (некоторые веса становятся очень большими) и при постоянном увеличении количества итераций и значения α метрики метрики могут быть хуже. Проверим это на примере. Для это нам понадобится большая выборка, что бы можно было разделить ее на обучающую и тестовую. Для большей наглядности я не буду масштабировать признаки

In [46]:

```
1 from sklearn import datasets
2 from matplotlib.colors import ListedColormap
3
4 # сгенерируем данные с помощью sklearn.datasets
5 classes = datasets.make_classification(n_samples=1000, n_features=5, n_informative=2,
6                                     n_redundant=0, n_classes=2, random_state=1)
7 # datasets.make_blobs(centers = 5, cluster_std = 1, random_state=1)
8
9 # и изобразим их на графике
10 colors = ListedColormap(['red', 'blue'])
11
12 plt.figure(figsize=(8, 8))
13 plt.scatter([x[0] for x in classes[0]], [x[1] for x in classes[0]], c=classes[1], cmap=
```

Out[46]:

<matplotlib.collections.PathCollection at 0x1236f5c8>



In [47]:

```
1 # перемешивание датасета
2 np.random.seed(41)
3 shuffle_index = np.random.permutation(classes[0].shape[0])
4 X_shuffled, y_shuffled = classes[0][shuffle_index], classes[1][shuffle_index]
```

Далее разделим выборку на обучающую и тестовую.

In [48]:

```
1 # разбишка на обучающую и тестовую выборки
2 train_proportion = 0.5
3 train_test_cut = int(len(classes[0]) * train_proportion)
4
5 X_train, X_test, y_train, y_test = \
6     X_shuffled[:train_test_cut], \
7     X_shuffled[train_test_cut:], \
8     y_shuffled[:train_test_cut], \
9     y_shuffled[train_test_cut:]
10
11 print("Размер массива признаков обучающей выборки", X_train.shape)
12 print("Размер массива признаков тестовой выборки", X_test.shape)
13 print("Размер массива ответов для обучающей выборки", y_train.shape)
14 print("Размер массива ответов для тестовой выборки", y_test.shape)
```

Размер массива признаков обучающей выборки (500, 5)

Размер массива признаков тестовой выборки (500, 5)

Размер массива ответов для обучающей выборки (500,)

Размер массива ответов для тестовой выборки (500,)

In [49]:

```
1 # # Масштабируем признаки
2 # def standard_scaler(x):
3 #     res = (x - x.mean()) / x.std()
4 #     return res
5
6 # X_train = X_train.copy()
7 # X_train = standard_scaler(X_train)
8
9 # X_test = X_test.copy()
10 # X_test = standard_scaler(X_test)
```

In [50]:

```
1 # Транспонируем признаки для удобства работы
2 X_train_tr = X_train.transpose()
3 y_train_tr = y_train.reshape(1, y_train.shape[0])
4 X_test_tr = X_test.transpose()
5 y_test_tr = y_test.reshape(1, y_test.shape[0])
```

In [51]:

```
1 def optimize(w, X, y, n_iterations, eta):
2     # потери будем записывать в список для отображения в виде графика
3     losses = []
4
5     for i in range(n_iterations):
6         loss, grad = log_loss(w, X, y)
7         w = w - eta * grad
8
9         losses.append(loss)
10
11     return w, losses
```

In [52]:

```
1 # Напишем функцию для выполнения предсказаний
2 def predict(w, X):
3
4     m = X.shape[1]
5
6     y_predicted = np.zeros((1, m))
7     w = w.reshape(X.shape[0], 1)
8
9     A = sigmoid(np.dot(w.T, X))
10
11 #     За порог отнесения к тому или иному классу примем вероятность 0.5
12 for i in range(A.shape[1]):
13     if (A[:,i] > 0.5):
14         y_predicted[:, i] = 1
15     elif (A[:,i] <= 0.5):
16         y_predicted[:, i] = 0
17 #     print(A)
18 return y_predicted
```

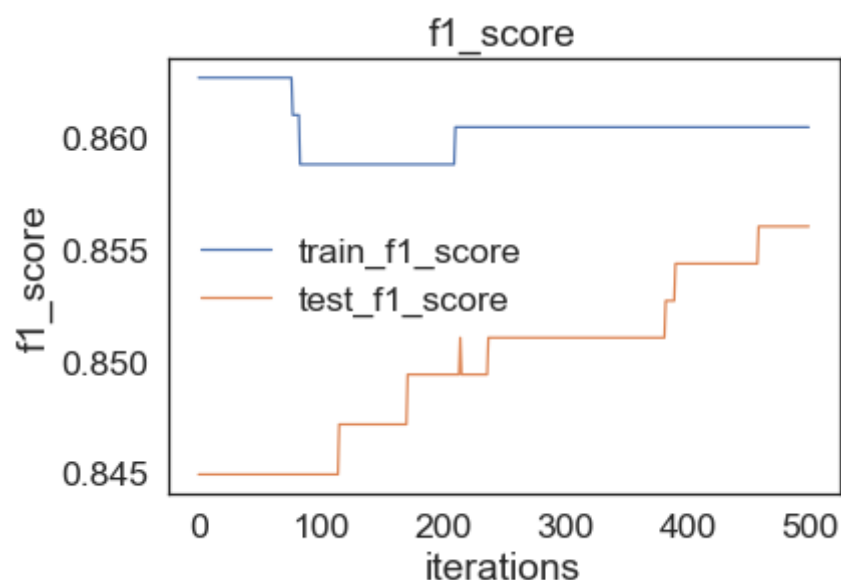
In [53]:

```
1 def log_loss(w, X, y):
2     m = X.shape[1]
3
4     # используем функцию сигмоиды, написанную ранее
5     A = sigmoid(np.dot(w.T, X)) # вероятность отнесения объекта к классу "+1"
6
7     loss = -1.0 / m * np.sum(y * np.log(A) + (1 - y) * np.log(1 - A))
8     # loss = -1.0 / m * np.log(1 + np.exp(np.dot(w.T, X)))
9     grad = 1.0 / m * np.dot(X, (A - y).T)
10
11 return loss, grad
```

Посмотрим как меняется f1 при изменении количества итеаций с 1 до 300

In [54]:

```
1 train_f1_score = []
2 test_f1_score = []
3
4 for i in range(1, 501):
5     # инициализируем начальный вектор весов
6     w0 = np.zeros((X_train_tr.shape[0], 1))
7
8     n_iterations = i
9     eta = 0.01
10
11     w, losses = optimize(w0, X_train_tr, y_train_tr, n_iterations, eta)
12
13     y_predicted_train = predict(w, X_train_tr)
14     y_predicted_test = predict(w, X_test_tr)
15
16
17     # В качестве меры точности возьмем f1_score
18
19     f1_score_train = f1_score(y_train_tr[0], y_predicted_train[0])
20     f1_score_test = f1_score(y_test_tr[0], y_predicted_test[0])
21
22     train_f1_score.append(f1_score_train)
23     test_f1_score.append(f1_score_test)
24
25 plt.title('f1_score')
26 plt.xlabel('iterations')
27 plt.ylabel('f1_score')
28 plt.plot(range(len(train_f1_score)), train_f1_score, label='train_f1_score');
29 plt.plot(range(len(test_f1_score)), test_f1_score, label='test_f1_score');
30 plt.legend();
31
```



In [55]:

```
1 w_1 = w
```

Видим, что в обоих случаях модель переобучается

7. *Создайте функции `eval_LR_model_I1` и `eval_LR_model_I2` с применением L1 и L2 регуляризации соответственно.

L2 регуляризация

In [56]:

```
1 # перемешивание датасета
2 np.random.seed(41)
3 shuffle_index = np.random.permutation(classes[0].shape[0])
4 X_shuffled, y_shuffled = classes[0][shuffle_index], classes[1][shuffle_index]
```

Далее разделим выборку на обучающую и тестовую.

In [57]:

```
1 # разбивка на обучающую и тестовую выборки
2 train_proportion = 0.5
3 train_test_cut = int(len(classes[0]) * train_proportion)
4
5 X_train, X_test, y_train, y_test = \
6     X_shuffled[:train_test_cut], \
7     X_shuffled[train_test_cut:], \
8     y_shuffled[:train_test_cut], \
9     y_shuffled[train_test_cut:]
10
11 print("Размер массива признаков обучающей выборки", X_train.shape)
12 print("Размер массива признаков тестовой выборки", X_test.shape)
13 print("Размер массива ответов для обучающей выборки", y_train.shape)
14 print("Размер массива ответов для тестовой выборки", y_test.shape)
```

Размер массива признаков обучающей выборки (500, 5)

Размер массива признаков тестовой выборки (500, 5)

Размер массива ответов для обучающей выборки (500,)

Размер массива ответов для тестовой выборки (500,)

In [58]:

```
1 # Масштабируем признаки
2 # def standard_scaler(x):
3 #     res = (x - x.mean()) / x.std()
4 #     return res
5
6 # X_train = X_train.copy()
7 # X_train = standard_scaler(X_train)
8
9 # X_test = X_test.copy()
10 # X_test = standard_scaler(X_test)
```

In [59]:

```
1 # Транспонируем признаки для удобства работы
2 X_train_tr = X_train.transpose()
3 y_train_tr = y_train.reshape(1, y_train.shape[0])
4 X_test_tr = X_test.transpose()
5 y_test_tr = y_test.reshape(1, y_test.shape[0])
```

In [60]:

```
1 def optimize(w, X, y, n_iterations, eta, alpha):
2     #   потери будем записывать в список для отображения в виде графика
3     losses = []
4     for i in range(n_iterations):
5         loss, grad = log_loss(w, X, y)
6         w = w - eta * (grad + 2 * alpha * w)
7
8         losses.append(loss)
9
10    return w, losses
```

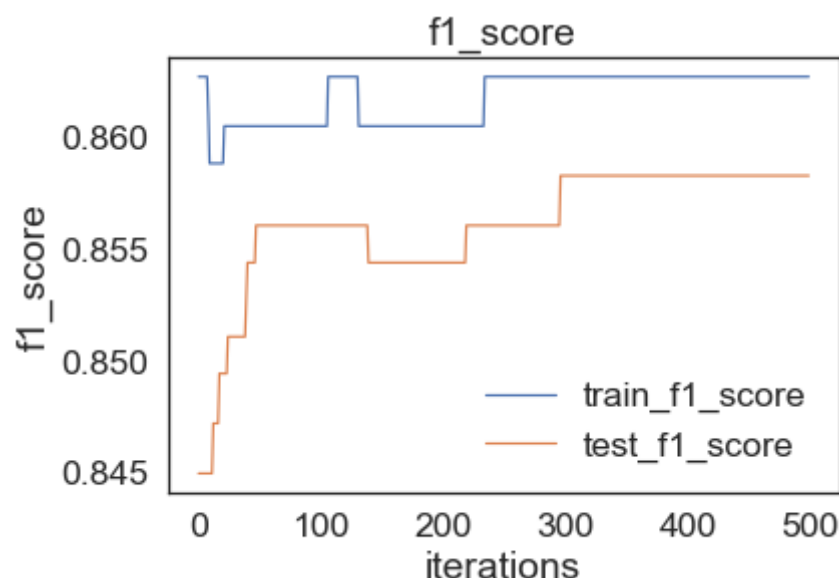
In [61]:

```
1 # Напишем функцию для выполнения предсказаний
2 def predict(w, X):
3
4     m = X.shape[1]
5
6     y_predicted = np.zeros((1, m))
7     w = w.reshape(X.shape[0], 1)
8
9     A = sigmoid(np.dot(w.T, X))
10
11    #   За порог отнесения к тому или иному классу примем вероятность 0.5
12    for i in range(A.shape[1]):
13        if (A[:,i] > 0.5):
14            y_predicted[:, i] = 1
15        elif (A[:,i] <= 0.5):
16            y_predicted[:, i] = 0
17    #   print(A)
18    return y_predicted
```

Посмотрим как меняется f1 при изменении количества итераций с 1 до 300

In [62]:

```
1 train_f1_score = []
2 test_f1_score = []
3
4 for i in range(1, 501):
5     # инициализируем начальный вектор весов
6     w0 = np.zeros((X_train_tr.shape[0], 1))
7
8     n_iterations = i
9     eta = 0.01
10    alpha = 0.1
11    w, losses = optimize(w0, X_train_tr, y_train_tr, n_iterations, alpha, eta)
12
13    y_predicted_train = predict(w, X_train_tr)
14    y_predicted_test = predict(w, X_test_tr)
15
16
17    # В качестве меры точности возьмем f1_score
18
19    f1_score_train = f1_score(y_train_tr[0], y_predicted_train[0])
20    f1_score_test = f1_score(y_test_tr[0], y_predicted_test[0])
21
22    train_f1_score.append(f1_score_train)
23    test_f1_score.append(f1_score_test)
24
25 plt.title('f1_score')
26 plt.xlabel('iterations')
27 plt.ylabel('f1_score')
28 plt.plot(range(len(train_f1_score)), train_f1_score, label='train_f1_score');
29 plt.plot(range(len(test_f1_score)), test_f1_score, label='test_f1_score');
30 plt.legend();
31
```



In [63]:

```
1 w_1_l_2 = w
```

Посмотрим как меняется f1 при изменении значений скорости обучения с 0.001 до 1

In [64]:

```
1 w_1
```

Out[64]:

```
array([[ -0.04907834],
       [  0.03825192],
       [  0.05707463],
       [  1.11011662],
       [  0.01072386]])
```

In [65]:

```
1 w_1_1_2
```

Out[65]:

```
array([[ -0.00681178],
       [  0.12299019],
       [  0.12603075],
       [  1.64404056],
       [  0.04889121]])
```

По графикам видно, что переобучение снизилось и веса стали меньше

L1 регуляризация

In [66]:

```
1 # перемешивание датасета
2 np.random.seed(41)
3 shuffle_index = np.random.permutation(classes[0].shape[0])
4 X_shuffled, y_shuffled = classes[0][shuffle_index], classes[1][shuffle_index]
```

Далее разделим выборку на обучающую и тестовую.

In [67]:

```
1 # разбивка на обучающую и тестовую выборки
2 train_proportion = 0.5
3 train_test_cut = int(len(classes[0]) * train_proportion)
4
5 X_train, X_test, y_train, y_test = \
6     X_shuffled[:train_test_cut], \
7     X_shuffled[train_test_cut:], \
8     y_shuffled[:train_test_cut], \
9     y_shuffled[train_test_cut:]
10
11 print("Размер массива признаков обучающей выборки", X_train.shape)
12 print("Размер массива признаков тестовой выборки", X_test.shape)
13 print("Размер массива ответов для обучающей выборки", y_train.shape)
14 print("Размер массива ответов для тестовой выборки", y_test.shape)
```

Размер массива признаков обучающей выборки (500, 5)
Размер массива признаков тестовой выборки (500, 5)
Размер массива ответов для обучающей выборки (500,)
Размер массива ответов для тестовой выборки (500,)

In [68]:

```
1 # Масштабируем признаки
2 # def standard_scaler(x):
3 #     res = (x - x.mean()) / x.std()
4 #     return res
5
6 # X_train = X_train.copy()
7 # X_train = standard_scaler(X_train)
8
9 # X_test = X_test.copy()
10 # X_test = standard_scaler(X_test)
```

In [69]:

```
1 # Транспонируем признаки для удобства работы
2 X_train_tr = X_train.transpose()
3 y_train_tr = y_train.reshape(1, y_train.shape[0])
4 X_test_tr = X_test.transpose()
5 y_test_tr = y_test.reshape(1, y_test.shape[0])
```

In [70]:

```
1 def optimize(w, X, y, n_iterations, eta, alpha):
2     # потери будем записывать в список для отображения в виде графика
3     losses = []
4     for i in range(n_iterations):
5         loss, grad = log_loss(w, X, y)
6         w = w - eta * (grad + alpha * np.sign(w))
7
8         losses.append(loss)
9
10    return w, losses
```

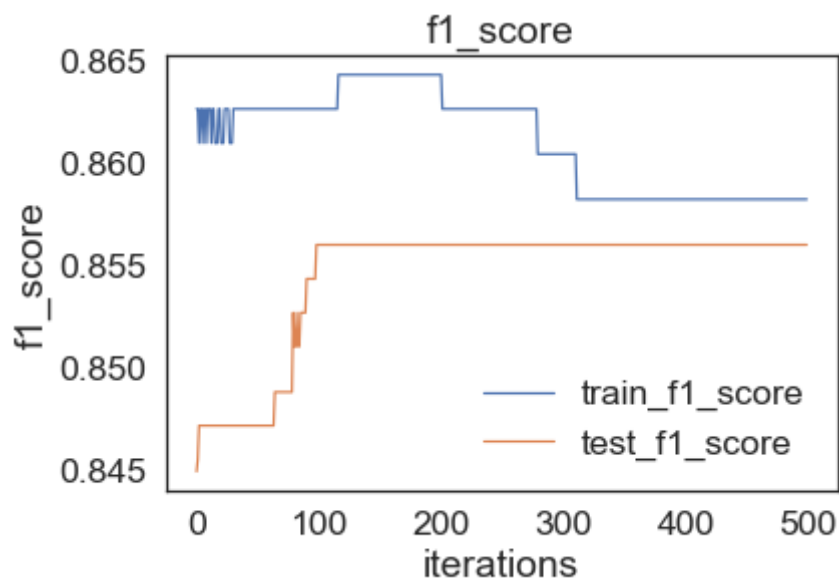
In [71]:

```
1  # Напишем функцию для выполнения предсказаний
2  def predict(w, X):
3
4      m = X.shape[1]
5
6      y_predicted = np.zeros((1, m))
7      w = w.reshape(X.shape[0], 1)
8
9      A = sigmoid(np.dot(w.T, X))
10
11  # За порог отнесения к тому или иному классу примем вероятность 0.5
12  for i in range(A.shape[1]):
13      if (A[:,i] > 0.5):
14          y_predicted[:, i] = 1
15      elif (A[:,i] <= 0.5):
16          y_predicted[:, i] = 0
17  # print(A)
18  return y_predicted
```

Посмотрим как меняется f1 при изменении количества итераций с 1 до 300

In [72]:

```
1 train_f1_score = []
2 test_f1_score = []
3
4 for i in range(1, 501):
5     # инициализируем начальный вектор весов
6     w0 = np.zeros((X_train_tr.shape[0], 1))
7
8     n_iterations = i
9     eta = 0.01
10    alpha = 0.03
11    w, losses = optimize(w0, X_train_tr, y_train_tr, n_iterations, alpha, eta)
12
13    y_predicted_train = predict(w, X_train_tr)
14    y_predicted_test = predict(w, X_test_tr)
15
16
17    # В качестве меры точности возьмем f1_score
18
19    f1_score_train = f1_score(y_train_tr[0], y_predicted_train[0])
20    f1_score_test = f1_score(y_test_tr[0], y_predicted_test[0])
21
22    train_f1_score.append(f1_score_train)
23    test_f1_score.append(f1_score_test)
24
25 plt.title('f1_score')
26 plt.xlabel('iterations')
27 plt.ylabel('f1_score')
28 plt.plot(range(len(train_f1_score)), train_f1_score, label='train_f1_score');
29 plt.plot(range(len(test_f1_score)), test_f1_score, label='test_f1_score');
30 plt.legend();
31
```



In [73]:

```
1 w_1_1_1 = w
```


In [74]:

1	w_1
---	-----

Out[74]:

```
array([[ -0.04907834],
       [  0.03825192],
       [  0.05707463],
       [  1.11011662],
       [  0.01072386]])
```

In [75]:

1	w_1_l_1
---	---------

Out[75]:

```
array([[ -8.11923126e-05],
       [  3.88653450e-02],
       [  5.88497870e-02],
       [  1.60187041e+00],
       [  1.03949304e-04]])
```

По графикам видно, что переобучение после трехсотой итерации снизилось, а веса стали ОЧЕНЬ маленькими