Урок 3. Классификация. Логистическая регрессия.

План занятия

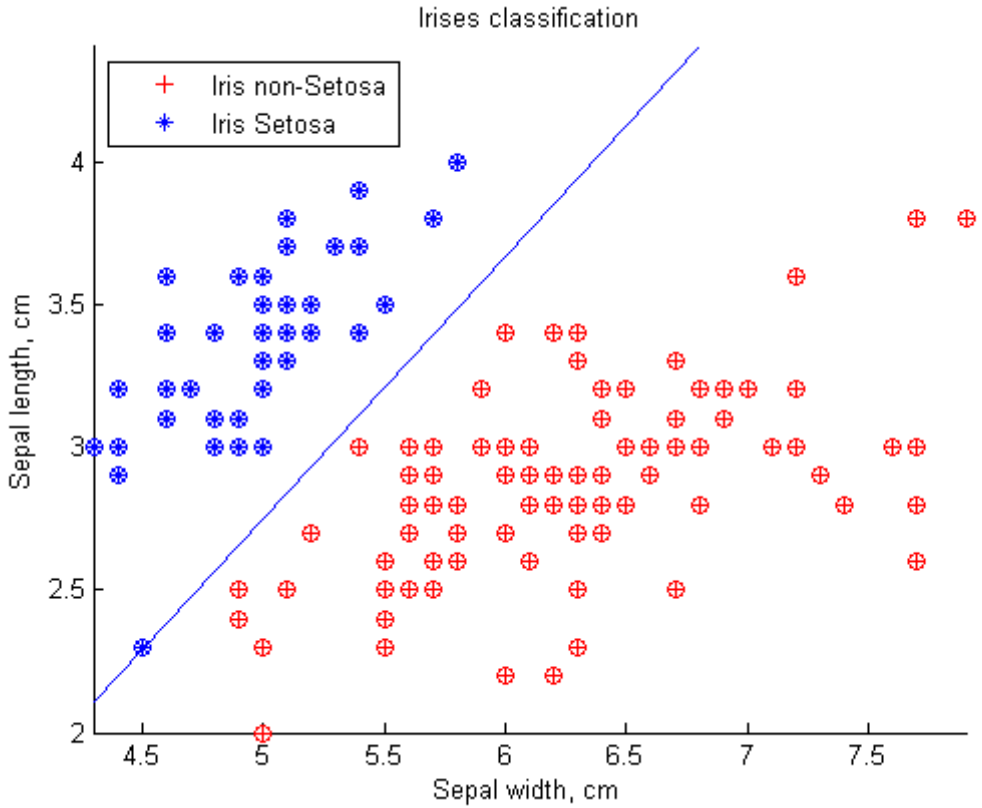
- Теоретическая часть
  - Линейная классификация
    - Функционал ошибки в линейной классификации
  - Логистическая регрессия
    - Метод максимального правдоподобия
    - Реализация логистической регрессии
  - Оценка качества классификации
- Практическая часть
  - Домашнее задание

Теоретическая часть

Линейная классификация

До этого мы разговаривали о задачах регрессии, то есть о восстановлении непрерывной зависимости по имеющимся данным. Однако, это не единственный тип задач в машинном обучении. В этом уроке речь пойдет о задачах *классификации*. Это такие задачи, в которых объекты делятся на конечное количество классов, и целью обучения является получение модели, способной соотносить объекты к тому или иному классу.

Простейшим случаем является *бинарная классификация*, то есть случай, когда у нас имеется два класса. Единственное отличие от линейной регрессии здесь в том, что пространство ответов состоит из двух элементов, в нашем случае возьмем $\mathbb{Y} = \{-1, 1\}$, где -1 и 1 означают принадлежность к первому или второму классу, соответственно. Пример такой задачи упоминался на первом уроке, когда говорилось о распознавании спам-писем. В этом случае, -1 означало, что письмо не является спамом, а 1 - что является.



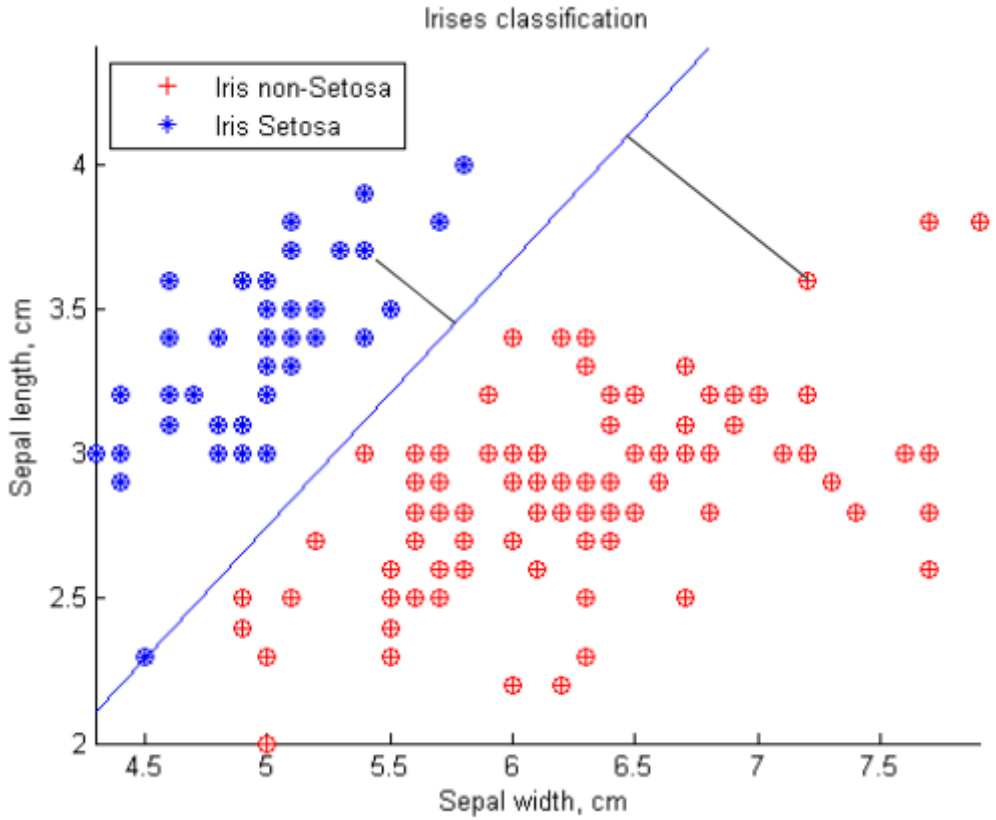
Как и в случае регрессии, в классификации можно использовать линейные модели. Это называется *линейной классификацией.* Линейные классификаторы устроены похожим на линейную регрессию образом, за одним лишь различием - для получения бинарных значений берется только знак от значения $a(x)$:

$$a(x) = \text{sign}\left(w_0 + \sum_{i=1}^{d} w_i x^i\right).$$

Аналогично линейной регрессии, после добавления константного признака формула имеет вид

$$\text{sign}\left(\sum_{i=1}^{d+1} w_i x^i\right) = \text{sign}\left(\langle w, x\rangle\right).$$

Множество точек $\langle w, x \rangle = 0$ образует *гиперплоскость* в пространстве признаков и делит его на две части. Объекты, расположенные по разные стороны от нее, относятся к разным классам.



Ввод [ ]:
```
+ + = +
- - = +
+ - = -
- + = -
```

$$M_i = y_i(\langle w, x \rangle)$$

$M_i > 0$ - классификатор дает верный ответ
$M_i < 0$ - классификатор ошибается

Стоит отметить, что для некоторого объекта x расстояние до этой гиперплоскости будет равняться $\frac{|\langle w,x\rangle|}{||w||}$, соответственно, при классификации нам важен не только знак скалярного произведения $\langle w, x\rangle$, но и его значение: чем выше оно, тем больше будет расстояние от объекта до разделяющей гиперплоскости, что будет означать, что алгоритм более уверен в отнесении объекта к данному классу. Это приводит нас к значению *отступа*, который равен скалярному произведению вектора весов w на вектор признаков x, умноженному на истинное значение ответа y, которое, как мы помним, принимает значения -1 и 1:

$$M_i = y_i \langle w, x_i \rangle .$$

Таким образом, если скалярное произведение отрицательно, и истинный ответ равен -1, отступ будет больше нуля. Если скалярное произведение положительно, и истинный ответ равен 1, отступ также будет положителен. То есть $M_i > 0$, когда классификатор дает верный ответ, и $M_i < 0$, когда классификатор ошибается. Отступ характеризует корректность ответа, а его абсолютное значение свидетельствует о расстоянии от разделяющей гиперплоскости, то есть о мере уверенности в ответе.

Функционал ошибки в линейной классификации

Как и в случае линейной регрессии, для обучения алгоритма линейной классификации требуется измерять ошибку. По аналогии с средней абсолютной ошибкой и среднеквадратичной ошибкой в случае линейной классификации можно использовать естественный подход: так как возможных ответов конечное число, можно требовать полного совпадения предсказанного класса $a(x_i)$ и истинного y_i. Тогда в качестве функционала ошибки можно использовать долю неправильных ответов:

$$Q(a, X) = \frac{1}{l} \sum_{i=1}^{l} [a(x_i) \neq y_i]$$

или, используя понятие отступа,

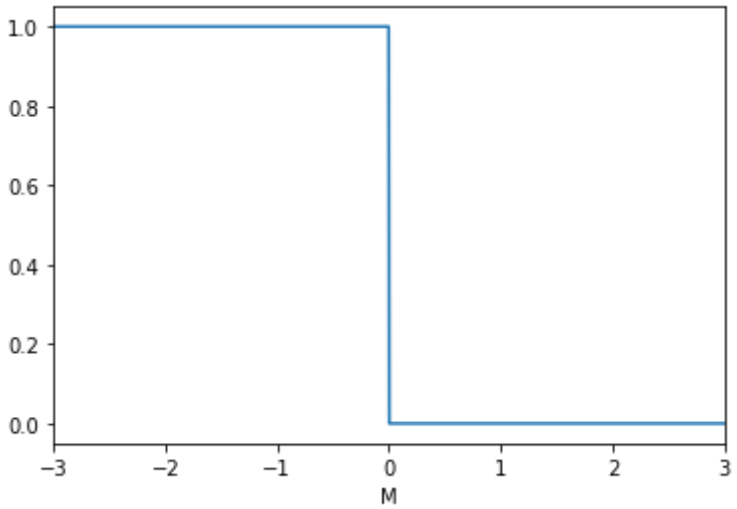
$$Q(a, X) = \frac{1}{l} \sum_{i=1}^{l} [M_i < 0] = \frac{1}{l} \sum_{i=1}^{l} [y_i \langle w, x_i \rangle < 0].$$

Функция, стоящая под знаком суммы, называется *функцией потерь*. График ее в зависимости от отступа будет иметь пороговый вид:

Ввод [1]:
```
import matplotlib.pyplot as plt
import numpy as np

%matplotlib inline
```

Ввод [2]:
```
def loss_function(x):
    return 0 if x > 0 else 1
```

Ввод [3]:
```
dots = np.linspace(-3, 3, 1000)
q_zero_one_loss = [loss_function(x) for x in dots]

plt.xlabel('M')
plt.xlim(-3, 3)
plt.plot(dots, q_zero_one_loss);
```



Она называется *пороговой функцией потерь* или 1/0 функцией потерь. Как мы видим, она негладкая, поэтому градиентные методы оптимизации к ней неприменимы. Для упрощения оптимизации используют гладкие оценки сверху этой функции, то есть такие функции, что

$$[M_i < 0] \leq \tilde{L}(M_i).$$

Тогда минимизировать уже нужно эту новую функцию:

$$Q(a, X) \leq \tilde{Q}(a, X) = \frac{1}{l} \sum_{i=1}^{l} \tilde{L}(M_i) \to \min_w.$$

Примерами могут быть:

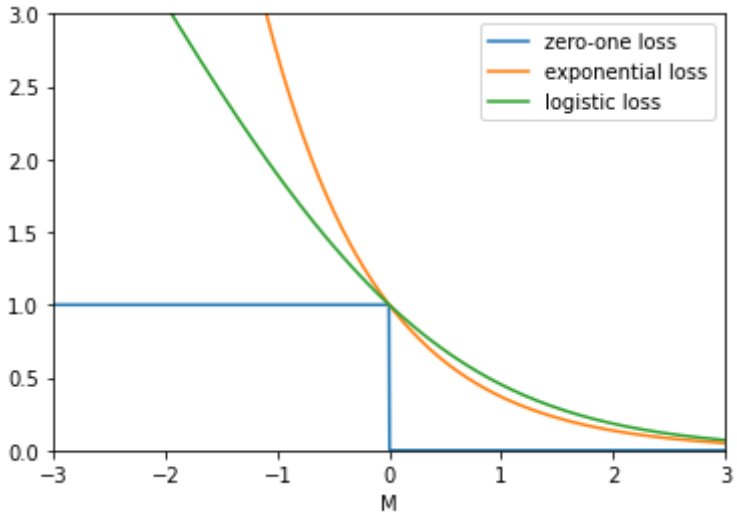
- *экспоненциальная функция потерь* $\tilde{L}(M_i) = \exp(-M_i)$
- *логистическая функция потерь* $\tilde{L}(M_i) = \log(1 + \exp(-M_i))$
- и др. (см. доп. материалы)

Реализуем их и построим соответствующие графики.

Ввод [4]:
```
def exp_loss_func(x):
    return np.exp(-x)
```

Ввод [5]:
```
def logistic_loss(x):
    return np.log2(1 + np.exp(-x))
```

Ввод [6]:
```
q_exp_loss = [exp_loss_func(x) for x in dots]
q_logistic_loss = [logistic_loss(x) for x in dots]

plt.xlabel('M')
plt.xlim(-3, 3)
plt.ylim(0, 3)
plt.plot(dots, q_zero_one_loss)
plt.plot(dots, q_exp_loss)
plt.plot(dots, q_logistic_loss)
plt.legend(['zero-one loss', 'exponential loss', 'logistic loss']);
```



Все они оценивают функцию потерь сверху и при этом хорошо оптимизируются.

Логистическая регрессия

Логистическая регрессия - частный случай линейного классификатора, обладающий одной полезной особенностью - помимо отнесения объекта к определенному классу она умеет прогнозировать вероятность P того, что объект относится к этому классу.

Во многих задачах такая особенность является очень важной. Например, в задачах кредитного скоринга (предсказание, вернет клиент кредит или нет) прогнозируют вероятность невозврата кредита и на основании нее принимают решение о выдаче или невыдаче.

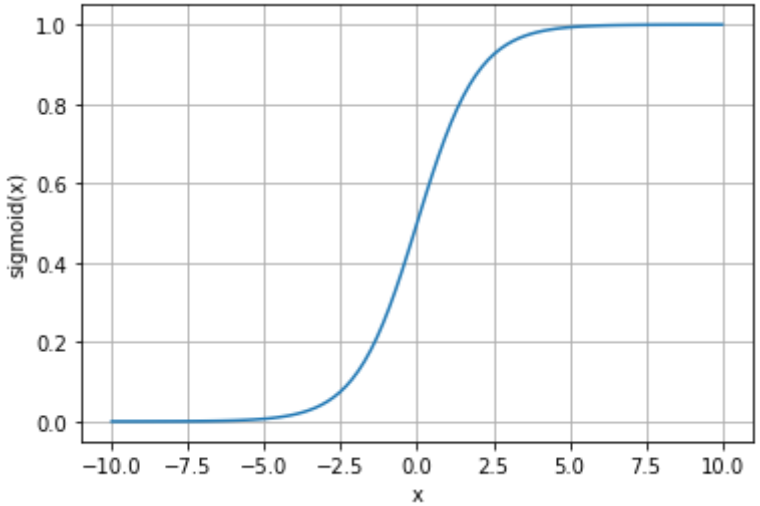
Пусть в каждой точке пространства объектов $\mathbb{X}$ задана вероятность того, что объект x будет принадлежать к классу "+1" $P(y = 1|x)$ (условная вероятность $y = 1$ при условии x). Она будет принимать значения от 0 до 1, и нам нужно каким-то образом ее предсказывать, но пока мы умеем только строить прогноз методами линейной регрессии с помощью некоего алгоритма $b(x) = \langle w, x_i \rangle$. У него есть проблема, связанная с тем, что скалярное произведение $\langle w, x_i \rangle$ не всегда возвращает значения в отрезке [0, 1]. Чтобы достичь такого условия, можно использовать некую функцию $\sigma : \mathbb{R} \to [0, 1]$, которая будет переводить полученное в скалярном произведении значение в вероятность, пределы которой будут лежать в промежутке от 0 до 1. В модели логистической регрессии в качестве такой функции берется сигмоида, которая имеет вид:

$$\sigma(z) = \frac{1}{1 + exp(-z)}.$$

Изобразим ее график.

Ввод [7]:
```
def sigmoid(x):
    return 1 / (1 + np.exp(-x))
```

Ввод [8]:
```
dots = np.linspace(-10, 10, 100)
sigmoid_value = list(map(sigmoid, dots))

plt.xlabel('x')
plt.ylabel('sigmoid(x)')
plt.grid()
plt.plot(dots, sigmoid_value);
```

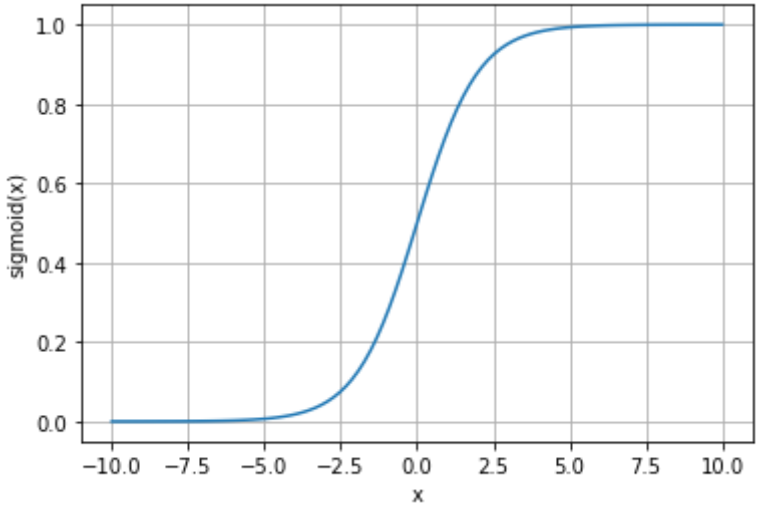


$$\sigma(z) = \frac{1}{(1 + exp(-z))^{-1}} = \frac{1}{1 + \frac{1}{exp(z)}} = \frac{exp(z)}{exp(z) + 1}$$

$$\sigma(z) = \frac{exp(z)}{1 + exp(z)}.$$

Ввод [9]:
```
def sigmoid_2(x):
    return np.exp(x) / (1 + np.exp(x))
```

Ввод [10]:
```
dots = np.linspace(-10, 10, 100)
sigmoid_value_2 = list(map(sigmoid_2, dots))

plt.xlabel('x')
plt.ylabel('sigmoid(x)')
plt.grid()
plt.plot(dots, sigmoid_value_2);
```



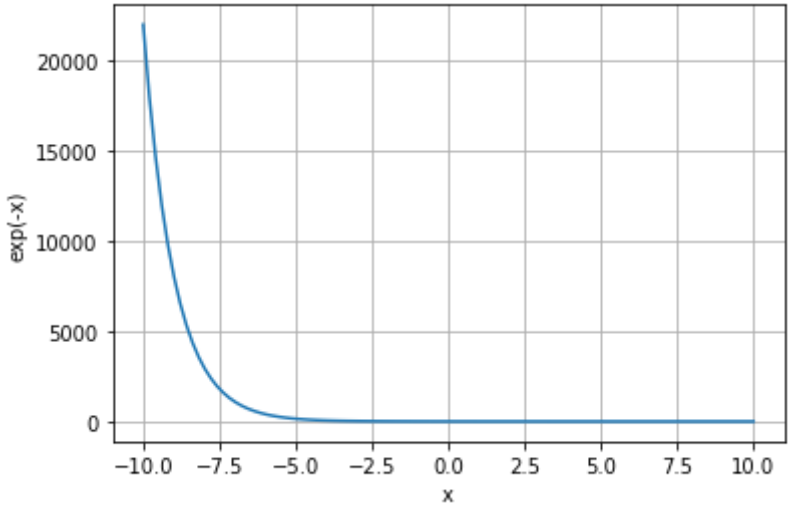
При использовании такой функции $\tilde{b}(x_i) = \sigma(\langle w, x_i \rangle)$ получаем, что вероятность отнесения объекта к классу "+1" $P(y = 1|x)$, которую для краткости обозначим p_+, будет равняться

$$p_+ = \sigma(\langle w, x_i \rangle) = \frac{1}{1 + exp(-\langle w, x_i \rangle)},$$

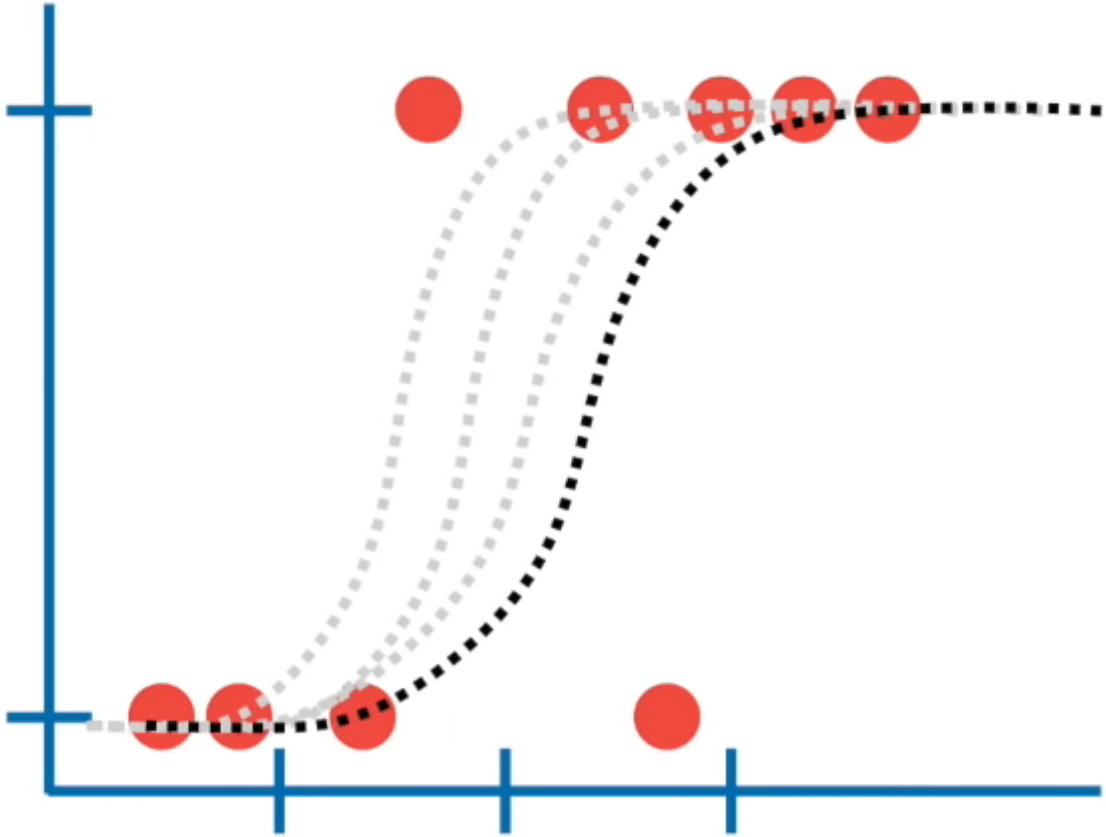
Чем больше будет скалярное произведение $\langle w, x_i \rangle$, тем выше будет предсказанная вероятность.

Ввод [11]:

```
dots = np.linspace(-10, 10, 100)
exp_value = list(map(lambda x: np.exp(-x), dots))

plt.xlabel('x')
plt.ylabel('exp(-x)')
plt.grid()
plt.plot(dots, exp_value);
```



Метод максимального правдоподобия



Далее для обучения этой модели нам потребуется использовать *метод максимального правдоподобия* (см. доп. материалы). Его сущность заключается в выборе гипотезы, при которой вероятность получить имеющееся наблюдение максимальна.

С точки зрения реализуемого алгоритма вероятность того, что в выборке встретится объект x_i с классом y_i, равна

$$P(y = y_i|x_i) = p_+^{[y_i=+1]}(1 - p_+)^{[y_i=-1]}.$$

Исходя из этого, правдоподобие выборки (т.е. вероятность получить такую выборку с точки зрения алгоритма) будет равняться произведению вероятностей получения каждого имеющегося ответа:

$$P(y|X) = L(X) = \prod_{i=1}^{l} p_+^{[y_i=+1]}(1-p_+)^{[y_i=-1]}.$$

Правдоподобие можно использовать как функционал для обучения алгоритма, однако, удобнее взять от него логарифм, так как в этом случае произведение превратится в сумму, а сумму гораздо проще оптимизировать. Также, в отличие от рассмотренных ранее функций потерь, правдоподобие требуется максимизировать для обучения алгоритма, а не минимизировать. Поэтому для большего удобства перед правдоподобием ставят минус, поскольку функции потери в задачах регрессии принято минимизировать. В итоге получим:

$$\ln \prod_{i=1}^{l} p_+^{[y_i=+1]}(1-p_+)^{[y_i=-1]} =$$

$$= -\sum_{i=1}^{l} \ln(p_+^{[y_i=+1]}(1-p_+)^{[y_i=-1]})$$

$$-\ln L(X) = -\sum_{i=1}^{l}([y_i=+1]\ln p_+) + [y_i=-1]\ln(1-p_+))).$$

Данная функция потерь называется *логарифмической функцией потерь (log loss)* или *кросс-энтропией*.

В случае, когда имеются классы 1 и -1:

$$p_+ = \sigma(\langle w, x\rangle)$$
$$p_- = \sigma(-\langle w, x\rangle)$$

$$p = \sigma(y\langle w, x\rangle)$$

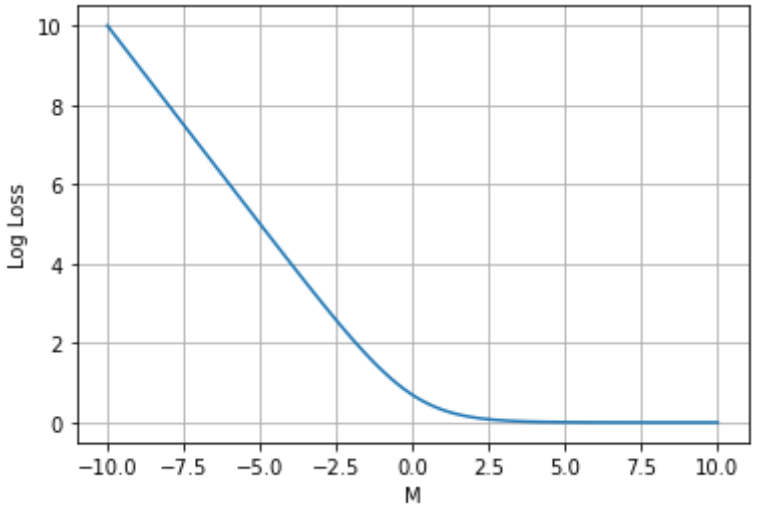
$$\ln L = -\sum_{i=1}^{l} \ln(p_+^{[y_i=+1]}(1-p_+)^{[y_i=-1]}) =$$

$$= -\sum_{i=1}^{l} \ln(\sigma(y\langle w, x\rangle)) = -\sum_{i=1}^{l} \ln(\frac{1}{1+exp(-y\langle w, x\rangle}})$$

$$= \sum_{i=1}^{l} \ln(1+exp(-y\langle w, x\rangle))$$

То есть в случае логистической регрессии обучение сводится к минимизации этого функционала.

Ввод [12]:
```
dots = np.linspace(-10, 10, 100)
log_loss_value = list(map(lambda x: - np.log(1 / (1 + np.exp(-x))), dots))
log_loss_value = list(map(lambda x: np.log(1 + np.exp(-x)), dots))

plt.xlabel('M')
plt.ylabel('Log Loss')
plt.grid()
plt.plot(dots, log_loss_value);
```



В общем виде log loss запишется как

$$-\ln L(X) = -\sum_{i=1}^{l}(y_i \ln \frac{1}{1+exp(-\langle w, x_i\rangle)} + (1-y_i)\ln(1-\frac{1}{1+exp(-\langle w, x_i\rangle)})).$$

$$-\ln L(X) = -\sum_{i=1}^{l}(y_i \ln(\sigma) + (1-y_i)\ln(1-\sigma)).$$

Производные

Сигмоида

$$\sigma(z) = \frac{1}{1 + exp(-z)}$$

$$\frac{d\sigma(z)}{dz} = -\frac{1}{(1 + exp(-z))^2}(-exp(-z)) = \frac{exp(-z)}{(1 + exp(-z))^2}(*) = \frac{1}{1 + exp(-z)}(1 - \frac{1}{1 + exp(-z)}) = \sigma(1 - \sigma)$$

$(*)\frac{exp(-z)+1-1}{(1+exp(-z))^2} = \frac{1+exp(-z)-1}{(1+exp(-z))^2} = \frac{1+exp(-z)}{(1+exp(-z))^2} - \frac{1}{(1+exp(-z))^2} =$

$= \frac{1}{(1+exp(-z))} - \frac{1}{(1+exp(-z))}\frac{1}{(1+exp(-z))} = \frac{1}{(1+exp(-z))}(1 - \frac{1}{(1+exp(-z))})$

Логлосс

$$\frac{dL}{dw} = -\sum_{i=1}^{l}(\frac{y_i}{\sigma} - \frac{1 - y_i}{1 - \sigma})\frac{d\sigma(z)}{dz} = -\sum_{i=1}^{l}\frac{(1 - \sigma)y_i - \sigma(1 - y_i)}{\sigma(1 - \sigma)}\frac{d\sigma(z)}{dz} = -\sum_{i=1}^{l}\frac{(y_i - \sigma y_i - \sigma + \sigma y_i)}{\sigma(1 - \sigma)}\frac{d\sigma(z)}{dz}$$

$$= -\sum_{i=1}^{l}\frac{y_i - \sigma}{\sigma(1 - \sigma)}\frac{d\sigma(z)}{dz} = -\sum_{i=1}^{l}\frac{y_i - \sigma}{\sigma(1 - \sigma)}\sigma(1 - \sigma) = \sum_{i=1}^{l}\sigma - y_i = \frac{1}{1 + exp(-\langle w, x\rangle)} - Y = X^T(\sigma - Y)$$

$\frac{d\langle w,x\rangle}{dw} = \frac{dXw}{dw} = X^T$

Реализация логистической регрессии

Напишем алгоритм логистической регрессии.

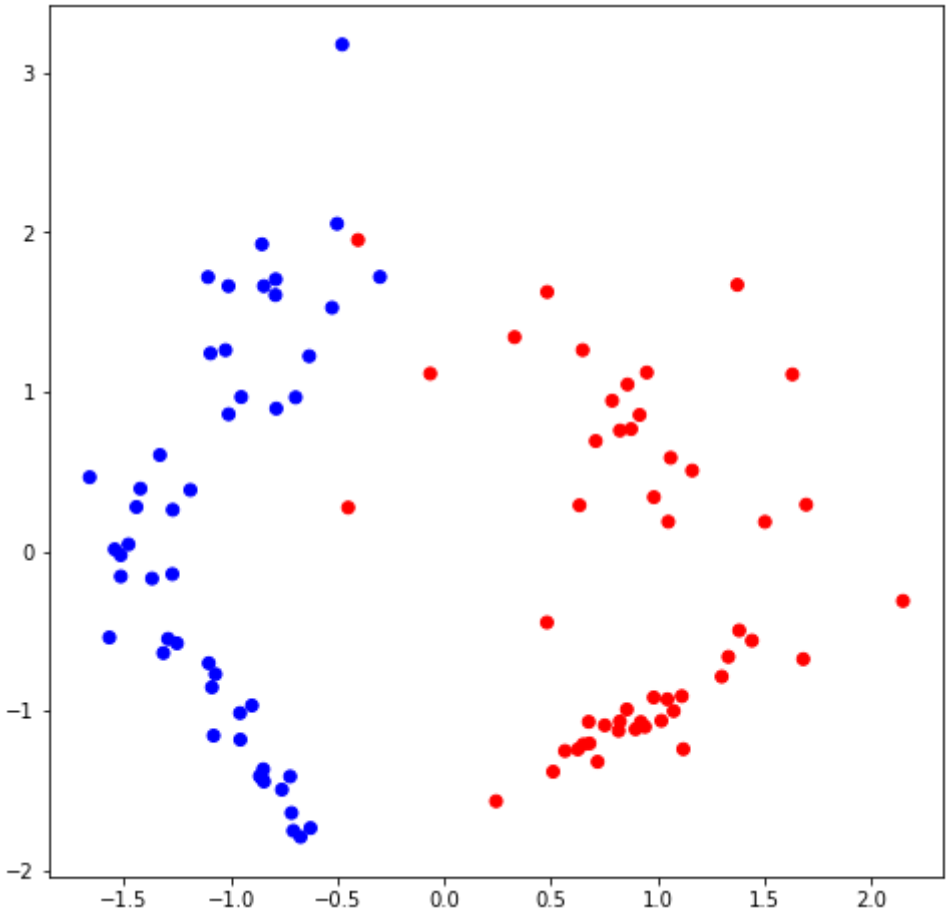
Ввод [13]:

```
from sklearn import datasets
from matplotlib.colors import ListedColormap

# сгеренируем данные с помощью sklearn.datasets
X, y = datasets.make_classification(n_samples=100, n_features=2, n_informative=2,
                                    n_redundant=0, n_classes=2, random_state=1)
# X, y = datasets.make_blobs(centers=2, cluster_std=2.5, random_state=12)
X, y
```

Out[13]:

```
(array([[ 1.30022717, -0.7856539 ],
        [ 1.44184425, -0.56008554],
        [-0.84792445, -1.36621324],
        [-0.72215015, -1.41129414],
        [-1.27221465,  0.25945106],
        [ 0.78694271,  0.94294879],
        [ 0.81695766, -1.12478707],
        [ 1.6822707 , -0.67596877],
        [-1.07223343, -0.7701513 ],
        [-0.06539297,  1.11257376],
        [-1.33232952,  0.60245671],
        [-0.6963714 ,  0.96382716],
        [ 0.82340614,  0.7561926 ],
        [-1.5155534 , -0.15804853],
        [-0.78971776,  1.70347973],
        [ 0.85887841,  1.04457966],
        [-0.45001335,  0.27345841],
        [ 0.7099324 ,  0.69020919],
        [-0.79095935,  1.60495551],
```

Ввод [14]:
```
# и изобразим их на графике
colors = ListedColormap(['blue', 'red'])

plt.figure(figsize=(8, 8))
plt.scatter(X[:, 0], X[:, 1], c=y, cmap=colors);
```



Далее разделим выборку на обучающую и тестовую. При реальной работе, если нет специфических требований по сохранению порядка выборки, ее полезно перемешивать, так как данные в ней могут быть каким-либо образом отсортированы. Это может негативно сказаться на процессе обучения.

Ввод [15]:
```
np.random.permutation(X.shape[0])
```

Out[15]:
```
array([20, 21, 61, 47, 34, 95, 28,  2, 81, 27, 59,  4, 77, 25, 57, 39, 84,
       76, 52, 15,  3, 54, 98, 99, 91,  7, 41, 50, 71,  6, 96, 51, 53, 23,
       17, 73, 97, 26, 70, 35, 83, 62, 30, 55, 87, 63, 36, 14, 33, 11, 67,
       56, 88, 44, 74, 93,  0, 45, 92,  8, 49, 43, 10, 58, 48,  5, 18,  1,
       82, 32, 46, 12, 31, 72, 65, 78, 37, 29, 86, 60, 90, 16, 68, 94, 69,
       89, 19, 75, 38, 13, 85, 42,  9, 64, 80, 66, 24, 79, 40, 22])
```

Ввод [16]:
```
# перемешивание датасета
np.random.seed(12)
shuffle_index = np.random.permutation(X.shape[0])
X_shuffled, y_shuffled = X[shuffle_index], y[shuffle_index]

# разбивка на обучающую и тестовую выборки
train_proportion = 0.7
train_test_cut = int(len(X) * train_proportion)

X_train, X_test, y_train, y_test = \
    X_shuffled[:train_test_cut], \
    X_shuffled[train_test_cut:], \
    y_shuffled[:train_test_cut], \
    y_shuffled[train_test_cut:]

print("Размер массива признаков обучающей выборки", X_train.shape)
print("Размер массива признаков тестовой выборки", X_test.shape)
print("Размер массива ответов для обучающей выборки", y_train.shape)
print("Размер массива ответов для тестовой выборки", y_test.shape)
```

```
Размер массива признаков обучающей выборки (70, 2)
Размер массива признаков тестовой выборки (30, 2)
Размер массива ответов для обучающей выборки (70,)
Размер массива ответов для тестовой выборки (30,)
```

Реализуем функцию потерь log loss с одновременным расчетом градиента.

Оптимизировать функционал ошибки будем с помощью градиентного спуска, его вид в случае использования такой функции потерь будет:

$$w_{n+1} = w_n - \eta \frac{1}{l} X^T (A - Y),$$

где $A = \frac{1}{1+exp(-\langle w, x_i \rangle)}$.

$$L(X) = -\sum_{i=1}^{l} (y_i \ln(\sigma) + (1 - y_i) \ln(1 - \sigma)).$$

$$L(X) = \sum_{i=1}^{l} \ln(1 + exp(-y\langle w, x \rangle))$$

Ввод [17]:
```
def log_loss(w, X, y):
    m = X.shape[0]
    # используем функцию сигмоиды, написанную ранее
    A = sigmoid(np.dot(X, w))

    # labels 0, 1
    loss = -1.0 / m * np.sum(y * np.log(A) + (1 - y) * np.log(1 - A))

    # labels -1, 1
#     temp_y = np.where(y == 1, 1, -1)
#     loss = 1.0 / m * np.sum(np.log(1 + np.exp(-temp_y * np.dot(X, w))))

    grad = 1.0 / m * X.T @ (A - y)

    return loss, grad
```

Реализуем градиентный спуск

Ввод [18]:
```
def optimize(w, X, y, n_iterations, eta):
    # потери будем записывать в список для отображения в виде графика
    losses = []

    for i in range(n_iterations):
        loss, grad = log_loss(w, X, y)
        w = w - eta * grad

        losses.append(loss)

    return w, losses
```

и функцию для выполнения предсказаний

Ввод [19]:
```
def predict(w, X):

    m = X.shape[0]

    y_predicted = np.zeros(m)

    A = np.squeeze(sigmoid(np.dot(X, w)))

    # За порог отнесения к тому или иному классу примем вероятность 0.5
    for i in range(A.shape[0]):
        if (A[i] > 0.5):
            y_predicted[i] = 1
        elif (A[i] <= 0.5):
            y_predicted[i] = 0

    return y_predicted
```

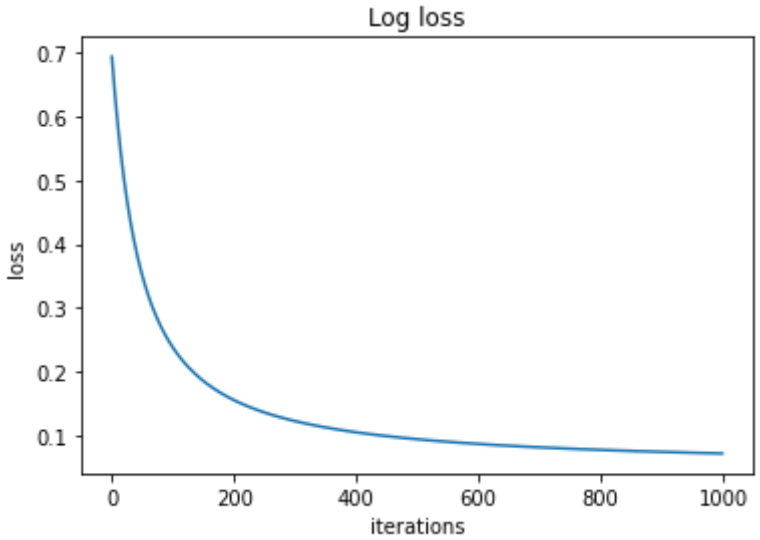
Ввод [20]:

```
# инициализируем начальный вектор весов
w0 = np.zeros(X_train.shape[1])

n_iterations = 1000
eta = 0.05

w, losses = optimize(w0, X_train, y_train, n_iterations, eta)

y_predicted_test = predict(w, X_test)
y_predicted_train = predict(w, X_train)

# В качестве меры точности возьмем долю правильных ответов
train_accuracy = np.mean(y_predicted_train == y_train) * 100.0
test_accuracy = np.mean(y_predicted_test == y_test) * 100.0

print(f"Итоговый вектор весов w: {w}")
print(f"Точность на обучающей выборке: {train_accuracy:.3f}")
print(f"Точность на тестовой выборке: {test_accuracy:.3f}")
```

```
Итоговый вектор весов w: [3.72659902 0.22383415]
Точность на обучающей выборке: 98.571
Точность на тестовой выборке: 96.667
```

Покажем, как менялась при этом функция потерь.

Ввод [21]:
```
plt.title('Log loss')
plt.xlabel('iterations')
plt.ylabel('loss')
plt.plot(range(len(losses)), losses);
```



[Визуализация (https://scikit-learn.org/stable/auto_examples/linear_model/plot_iris_logistic.html#sphx-glr-auto-examples-linear-model-plot-iris-logistic-py)](https://scikit-learn.org/stable/auto_examples/linear_model/plot_iris_logistic.html#sphx-glr-auto-examples-linear-model-plot-iris-logistic-py) логистической регрессии

Ввод [26]:
```
np.arange(x_min, x_max, h).shape
```

Out[26]:
```
(241,)
```

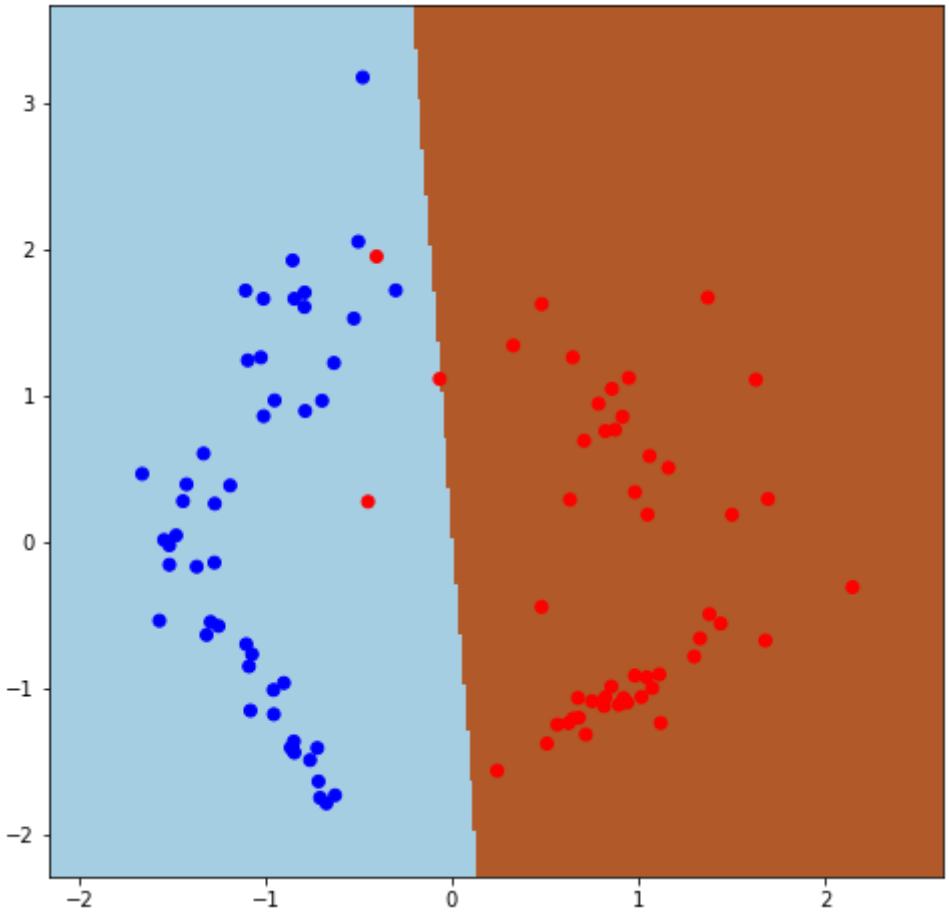
Ввод [25]:
```
Z.shape
```

Out[25]:
```
(299, 241)
```

Ввод [24]:

```
plt.figure(figsize=(8, 8))

x_min, x_max = X[:, 0].min() - .5, X[:, 0].max() + .5
y_min, y_max = X[:, 1].min() - .5, X[:, 1].max() + .5
h = .02  # step size in the mesh
xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))
Z = predict(w, np.c_[xx.ravel(), yy.ravel()])

# Put the result into a color plot
Z = Z.reshape(xx.shape)
plt.pcolormesh(xx, yy, Z, cmap=plt.cm.Paired)

plt.scatter(X[:, 0], X[:, 1], c=y, cmap=colors);
```

```
<ipython-input-24-7292eef48c77>:11: MatplotlibDeprecationWarning: shading='flat' when X and Y have the same dimension
s as C is deprecated since 3.3.  Either specify the corners of the quadrilaterals with X and Y, or pass shading='aut
o', 'nearest' or 'gouraud', or set rcParams['pcolor.shading'].  This will become an error two minor releases later.
  plt.pcolormesh(xx, yy, Z, cmap=plt.cm.Paired)
```



Оценка качества классификации

Как и в случае линейной регрессии, в задачах классификации требуется оценивать качество обученной модели. Для этого существует большое количество подходов.

Наиболее очевидным и простым способом является расчет *доли правильных ответов*:

$$accuracy(a, x) = \frac{1}{l}\sum_{i=1}^{l}[a(x_i) = y_i].$$

Проблемы accuracy:

```
1) Дисбаланс классов

кот - 950 наблюдений
голубь - 50 наблюдений
a(x) = кот

accuracy?

2) Ошибки могут иметь разную цену
```

200 людей

Модель 1:	Модель 2:
100 людям одобрила кредит	50 людям одобрила кредит
80 вернули 20 не вернули	48 вернули 2 не вернули

Матрица ошибок

Удобно представлять ответы в виде комбинации истинного ответа и ответа алгоритма. При этом получается так называемая *матрица ошибок*.

	$y = +1$	$y = -1$
$a(x) = +1$	True Positive TP	False Positive FP
$a(x) = -1$	False Negative FN	True Negative TN

В матрице сверху отложены истинные ответы, слева - ответы алгоритма. Когда алгоритм относит объект к классу "+1", говорят, что он *срабатывает*, а когда к "-1", - *пропускает*. Если алгоритм сработал (дал положительный ответ) и объект действительно относится к классу "+1", говорят, что имеет место верное срабатывание/верный положительный ответ (True Positive, TP), а если объект не относится к классу "+1", это ложное срабатывание (False Positive, FP). Если алгоритм пропускает объект, а его истинный класс "+1", это ложный пропуск/ложный негативные ответ (False Negative, FN), а если истинный класс объекта "-1", имеет место истинный пропуск (True Negative, TN). При такой классификации уже есть два вида ошибок - ложные срабатывания и ложные пропуски. По главной диагонали в матрице ошибок располагаются верные ответы, по побочной - неверные.

Точность и полнота

В классификации часто используются две метрики - *точность* и *полнота*.

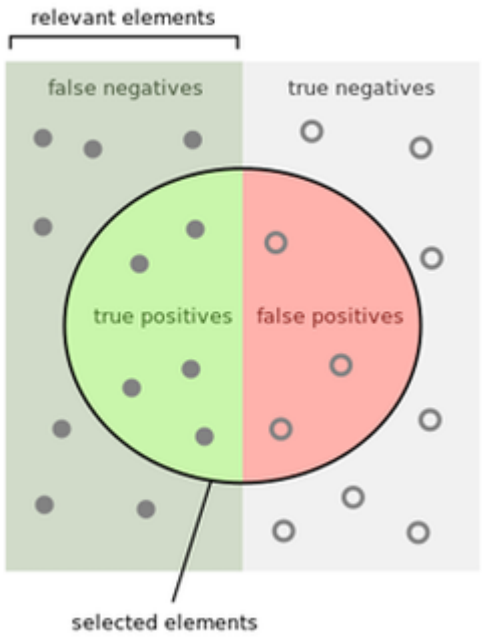
Точность (precision) представляет из себя долю истинных срабатываний от общего количества срабатываний. Она показывает, насколько можно доверять алгоритму классификации в случае срабатывания

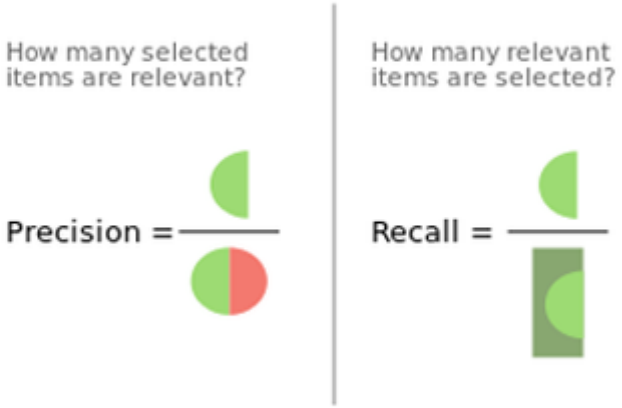
$$precision(a, X) = \frac{TP}{TP + FP}.$$

Полнота (recall) считается как доля объектов, истинно относящихся к классу "+1", которые алгоритм отнес к этому классу

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

здесь $TP + FN$ как раз будут вместе составлять весь список объектов класса "+1".





Пример

Пусть у нас есть выборка из 100 объектов, из которых 50 относится к классу "+1" и 50 к классу "-1" и для этой работы с этой выборкой мы рассматриваем две модели: $a_1(x)$ с матрицей ошибок

	$y=+1$	$y=-1$
$a_1(x)=+1$	40	10
$a_1(x)=-1$	10	40

и $a_2(x)$ с матрицей ошибок:

	$y=+1$	$y=-1$
$a_2(x)=+1$	22	2
$a_2(x)=-1$	28	48

Для первого алгоритма

$$precision(a_1, X) = 0.8$$
$$recall(a_1, X) = 0.8$$

Для второго алгоритма

$$precision(a_2, X) = 0.92$$
$$recall(a_2, X) = 0.44$$

Как мы видим, точность второй модели очень высока, но при этом сильно снижена полнота. Поэтому нужно правильно формировать бизнес-требования к модели, какой именно показатель должен быть определяющим. Например, если в задаче кредитного скоринга банк ставит цель возврата 90% кредитов, задачей ставится максимизация полноты при условии точности не ниже 0.9. А если при распознавании спама стоит требование, например, распознавать 95% спам-писем, задача состоит в максимизации точности при условии полноты не ниже 0.95.

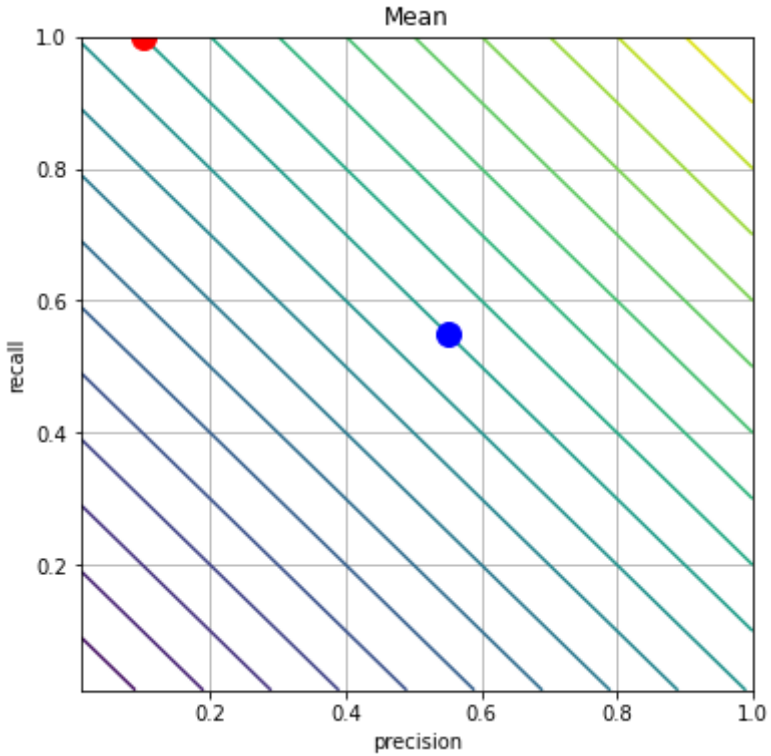
Однако, такое ограничение есть не всегда, и в остальных случаях требуется максимизировать и полноту и точность. Есть различные варианты объединения их в одну метрику, одним из наиболее удобных из них является *F-мера*, которая представляет собой среднее гармоническое между точностью и полнотой

$$F = \frac{2 \cdot precision \cdot recall}{presision + recall}.$$

В отличие от, например, среднего арифметического, если хотя бы один из аргументов близок к нулю, то и среднее гармоническое будет близко к нулю. По сути, F-мера является сглаженной версией минимума из точности и полноты (см. графики).

Ввод [27]:

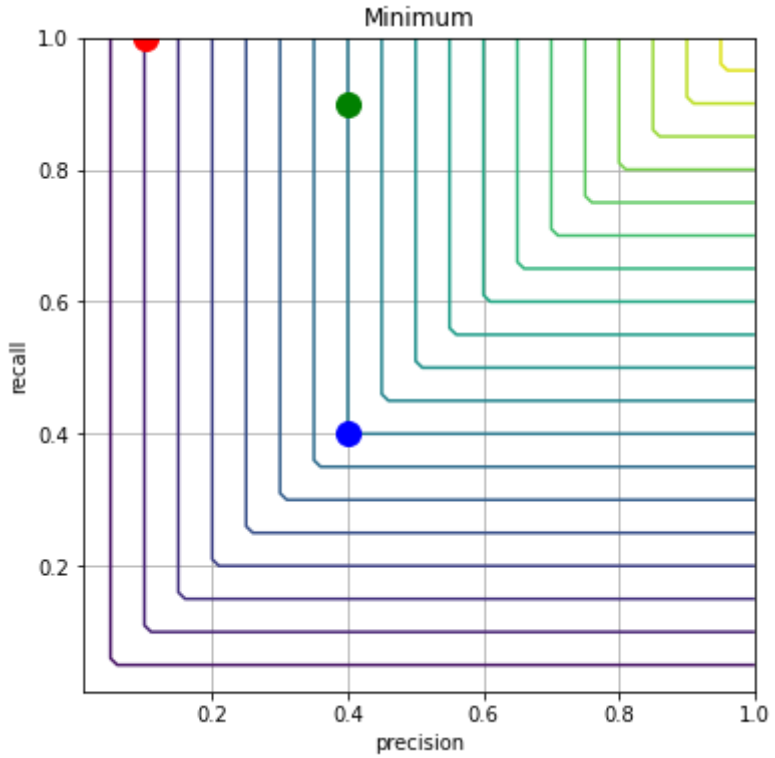
```
precisions, recalls = np.meshgrid(np.linspace(0.01, 1, 100), np.linspace(0.01, 1, 100))

mean_levels = np.empty_like(precisions)
for i in range(precisions.shape[0]):
    for j in range(precisions.shape[1]):
        mean_levels[i, j] = 1/2 * (precisions[i, j] + recalls[i, j])

plt.figure(figsize=(6, 6))
plt.title('Mean')
plt.xlabel('precision')
plt.ylabel('recall')
plt.grid()
plt.contour(precisions, recalls, mean_levels, levels=20)
plt.plot(0.1, 1, 'ro', ms=12)
plt.plot(0.55, 0.55, 'bo', ms=12);
```



	red	blue
precision	0.1	0.55
recall	1	0.55
mean	0.55	0.55

Ввод [28]:

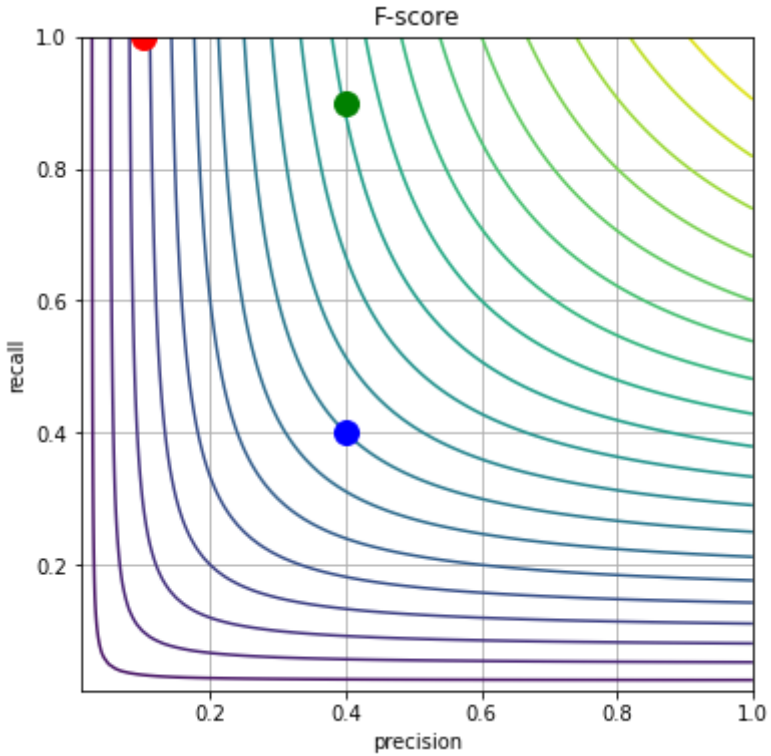
```
precisions, recalls = np.meshgrid(np.linspace(0.01, 1, 100), np.linspace(0.01, 1, 100))

min_levels = np.empty_like(precisions)
for i in range(precisions.shape[0]):
    for j in range(precisions.shape[1]):
        min_levels[i, j] = min([precisions[i, j], recalls[i, j]])

plt.figure(figsize=(6, 6))
plt.title('Minimum')
plt.xlabel('precision')
plt.ylabel('recall')
plt.grid()
plt.contour(precisions, recalls, min_levels, levels=20)
plt.plot(0.1, 1, 'ro', ms=12)
plt.plot(0.4, 0.4, 'bo', ms=12)
plt.plot(0.4, 0.9, 'go', ms=12);
```



	red	blue	green
precision	0.1	0.4	0.4
recall	1	0.4	0.9
min	0.1	0.4	0.4

Ввод [29]:

```
f_levels = np.empty_like(precisions)
for i in range(precisions.shape[0]):
    for j in range(precisions.shape[1]):
        f_levels[i, j] = 2 * precisions[i, j] * recalls[i, j] / (precisions[i, j] + recalls[i, j])

plt.figure(figsize=(6, 6))
plt.title('F-score')
plt.xlabel('precision')
plt.ylabel('recall')
plt.grid()
plt.contour(precisions, recalls, f_levels, levels=20)
plt.plot(0.1, 1, 'ro', ms=12)
plt.plot(0.4, 0.4, 'bo', ms=12)
plt.plot(0.4, 0.9, 'go', ms=12);
```



	red	blue	green
precision	0.1	0.4	0.4
recall	1	0.4	0.9
f-score	0.1818	0.4	0.55

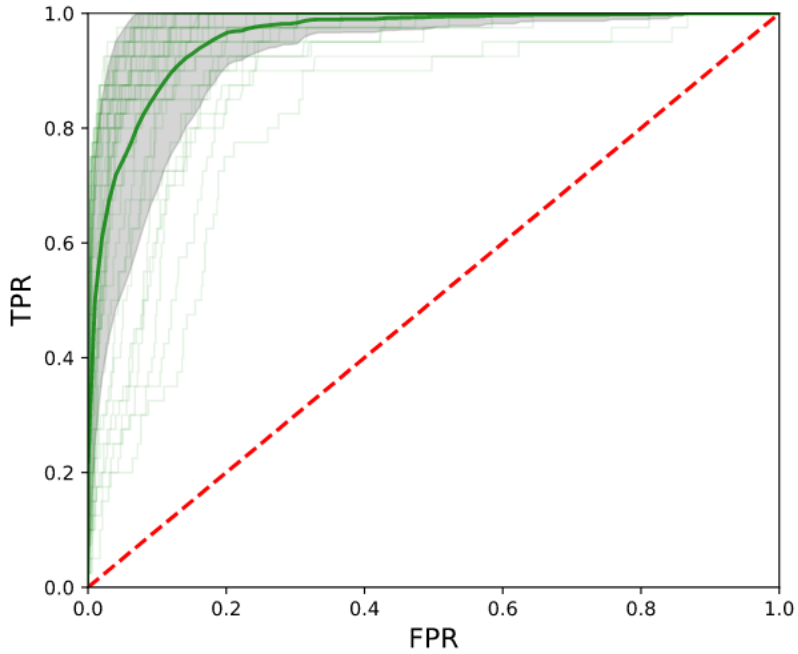
Существует также усовершенствованная версия F-меры F_β:

$$F_\beta = (1 + \beta^2)\frac{precision \cdot recall}{\beta^2 \cdot precision + recall}.$$

Параметр β здесь определяет вес точности в метрике. При $\beta = 1$ это среднее гармоническое, умноженное на 2 (чтобы в случае $precision = 1$ и $recall = 1$ $F_1 = 1$). Его изменение требуется, когда необходимо отдать приоритет точности или полноте, как это было показано в примерах ранее. Чтобы важнее была полнота, β должно быть меньше 1, чтобы важнее была точность - больше.

ROC-кривая

Итак, мы научились определять вероятность отнесения объекта к тому или иному классу и метрики, которые характеризуют качество работы алгоритма $a(x) = [b(x) > t]$, и теперь, чтобы конвертировать ее в бинарную метку (сделать выводы о принадлежности к классу), нужно определить значение порога вероятности t, при котором объект нужно относить к соответствующему классу. Естественным кажется вариант, при котором порог равен 0.5, но он не всегда оказывается оптимальным. Зачастую интерес представляет сам вещественнозначный алгоритм $b(x)$, а порог будет выбираться позже в зависимости от требований к точности и полноте. В таком случае появляется потребность в измерении качества семейства алгоритмов $a(x) = [b(x) > t]$ с различными t.



Есть способы оценки модели в целом, не привязываясь к конкретному порогу. Первый из них основанна использовании *ROC-кривой*. Такая кривая строится в следующих координатах:

по оси x откладывается доля ложных срабатываний (False Positive Rate) - отношение числа ложных срабатываний к общему размеру отрицательного класса:

$$FP$$

В качестве примера возьмем выборку из шести объектов, которым алгоритм $b(x)$ присвоил оценки принадлежности к классу 1:

$b(x)$	**0**	**0.1**	**0.2**	**0.3**	**0.5**	**0.6**
y	0	0	1	1	0	1

Ввод [ ]:
```
отсечка 0

b(x) = 0, 0.1, 0.2, 0.3, 0.5, 0.6
y =    0   0   1    1    0    1

a(x) - 1   1   1    1    1    1
```

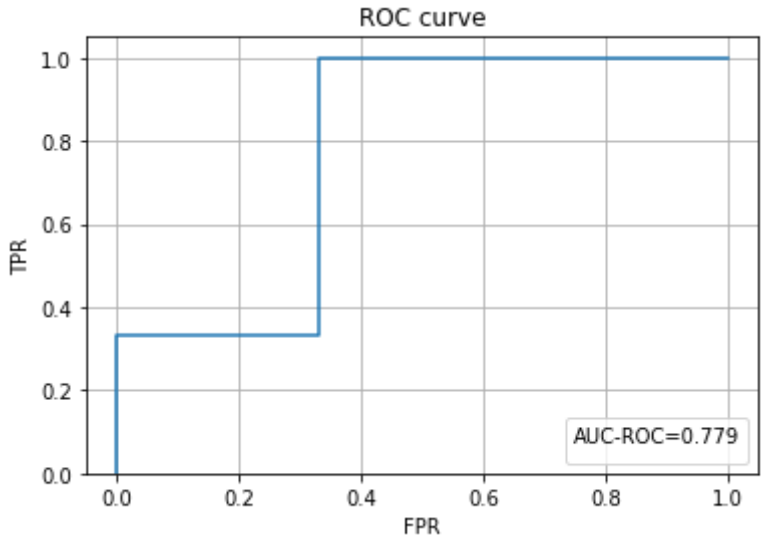
	0	0.1	0.2	0.3	0.5	0.6	0.99
TPR	1	1	1	0.66	0.33	0.33	0
FPR	1	0.66	0.33	0.33	0.33	0	0

Теперь пойдем по порядку справа налево:

1. Сначала выбираем самый большой порог, при котором ни один объект не будет отнесен к первому классу. При этом доля верных срабатываний и доля ложных срабатываний равны нулю. Получаем точку (0, 0).
2. Далее снижая порог до 0,6, один объект будет отнесен к первому классу. Доля ложных срабатываний останется нулевой, доля верных срабатываний станет 1/3.
3. При дальнейшем уменьшении порога до 0,5 второй справа один объект будет отнесен к первому классу. TPR останется 1/3, FPR также станет равна 1/3.
4. Далее при снижении порога до 0.3 TPR станет 2/3, FPR останется 1/3.
5. При пороге 0.2 TPR станет равна 1, FPR останется 1/3.
6. При пороге 0.2 5 объектов будут отнесены алгоритмом к классу 1, TPR останется 1, FPR станет 2/3.
7. При дальнейшем уменьшении порога все объекты будут отнесены к первому классу, и TPR и FPR станут равны 1.

Построим соответствующий график

Ввод [30]:
```
from numpy import trapz  # используем эту функцию для расчета площади под кривой

TPR = [0, 0.33, 0.33, 0.66, 1, 1, 1]
FPR = [0, 0, 0.33, 0.33, 0.33, 0.66, 1]

AUC_ROC = trapz(TPR, x = FPR, dx=0.1)

plt.title('ROC curve')
plt.ylim(0, 1.05)
plt.xlabel('FPR')
plt.ylabel('TPR')
plt.grid()
plt.legend(' ', title=f'AUC-ROC={AUC_ROC:.3f}', loc='lower right')
plt.plot(FPR, TPR);
```



ROC кривая всегда идет из точки (0,0) в точку (1,1). При этом в случае наличия идеального классификатора с определенным порогом доля его верных ответов будет равна 1, а доля ложных срабатываний - 0, то есть график будет проходить через точку (0,1). Таким образом, чем ближе к этой точке проходит ROC-кривая, тем лучше наши оценки и лучше используемое семейство алгоритмов. Таким образом мерой качества оценок принадлежности к классу 1 может служить площадь под ROC-кривой. Такая метрика называется AUC-ROC (Area Under Curve - площадь под кривой ROC). В случае идеального алгоритма $AUC - ROC = 1$, а в случае худшего приближается к $\frac{1}{2}$.

Критерий AUC-ROC можно интерпретировать как вероятность того, что если выбрать случайные положительный и отрицательный объект выборки, положительный объект получит оценку принадлежности выше, чем отрицательный.

Обычно объектов гораздо больше, чем в нашем примере, поэтому кривая в реальных задачах выглядит несколько иначе - в ней больше точек.

AUC-ROC не очень устойчив к несбалансированным выборкам. Допустим, нам нужно выбрать 100 релевантных документов из выборки в 1000000 документов. И у нас есть алгоритм, который дает выборку из 5000 документов, 90 из которых релевантны. В этом случае

$$TPR = \frac{TP}{TP + FN} = \frac{90}{90 + 10} = 0.9$$

$$FPR = \frac{FP}{FP + TN} = \frac{4910}{4910 + 994990} = 0.00491$$

,

Что является показателями очень хорошего алгоритма - AUC-ROC будет близка к 1, хотя на самом деле 4910 из 5000 выданных документов являются нерелевантными.

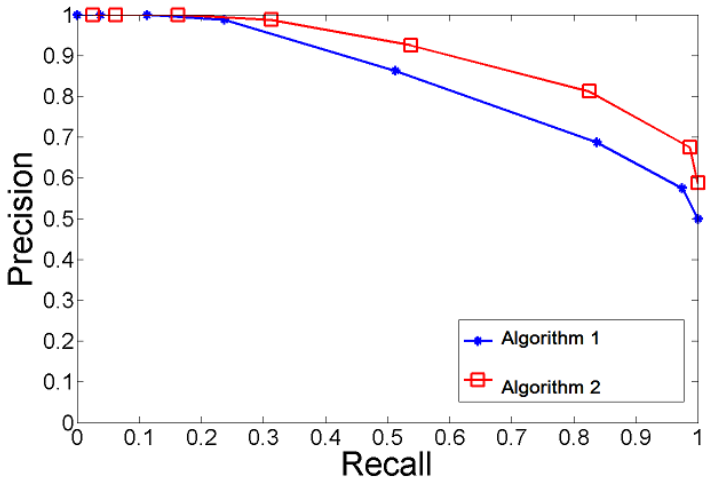
Чтобы посмотреть реальное положение дел, рассчитаем точность и полноту:

$$precision = \frac{TP}{TP + FP} = \frac{90}{90 + 4910} = 0.018$$

$$recall = TPR = 0.9.$$

Здесь уже видно, что алгоритм является недостаточно точным.

Таким образом, если размер положительного класс значительно меньше отрицательного, AUC-ROC может давать неадекватную оценку качества алгоритма, так как измеряет долю ложных срабатываний относительно общего числа отрицательных объектов, и если оно большое, доля будет мала, хотя в абсолютном значении количество ложных срабатываний может заметно превышать количество верных срабатываний.



Избавиться от такой проблемы можно используя другой метод - *кривую точности-полноты (PR-кривую)*. По оси x откладывается полнота, по оси y - точность, а точка на графике, аналогично ROC-кривой, будет соответствовать конкретному классификатору с некоторым значением порога.

Возьмем использованный нами для постройки ROC-кривой набор данных и аналогичным образом построим PR-кривую.

$b(x)$	**0**	**0.1**	**0.2**	**0.3**	**0.5**	**0.6**
y	0	0	1	1	0	1

Ввод [ ]:
```
pr = tp / (tp + fp)
rec = tp / (tp + fn)
```

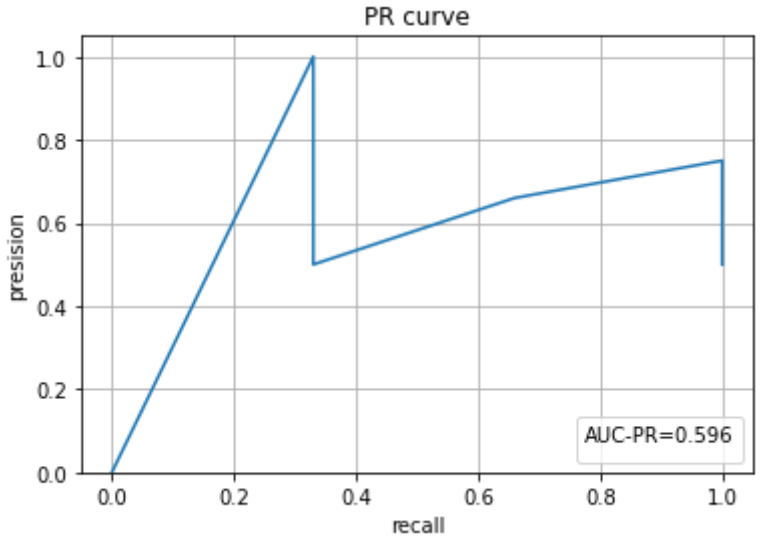
Ввод [ ]:
```
отсечка 0

b(x) = 0, 0.1, 0.2, 0.3, 0.5, 0.6
y =    0   0   1    1    0    1

a(x) = 1   1   1    1    1    1
```

	0	0.1	0.2	0.3	0.5	0.6	0.9
$Precision$	0.5	0.6	0.75	0.66	0.5	1	0
$Recall$	1	1	1	0.66	0.33	0.33	0

Она всегда стартует в точке (0,0) и заканчивается в точке (1, r), где r - доля положительных объектов в выборке. В случае наличия идеального классификатора, у которого точность и полнота 100%, кривая пройдет через точку (1,1). Таким образом, чем ближе к этой точке кривая проходит, тем лучше оценки. Так что, как и в случае ROC-кривой, можно ввести метрику качества в виде площади под PR-кривой AUC-PR.

Ввод [31]:

```
precision = [0, 1, 0.5, 0.66, 0.75, 0.6, 0.5]
recall = [0, 0.33, 0.33, 0.66, 1, 1, 1]

AUC_PR = trapz(precision, x = recall, dx=0.1)

plt.title('PR curve')
plt.ylim(0, 1.05)
plt.xlabel('recall')
plt.ylabel('presision')
plt.grid()
plt.legend(' ', title=f'AUC-PR={AUC_PR:.3f}', loc='lower right')
plt.plot(recall, precision);
```



Практическая часть

Ввод [32]:
```
import numpy as np
import matplotlib.pyplot as plt
```

Ввод [33]:
```
X = np.array([ [   1,    1,  500,    1],
               [   1,    1,  700,    1],
               [   1,    2,  750,    2],
               [   1,    5,  600,    1],
               [   1,    3, 1450,    2],
               [   1,    0,  800,    1],
               [   1,    5, 1500,    3],
               [   1,   10, 2000,    3],
               [   1,    1,  450,    1],
               [   1,    2, 1000,    2]], dtype=np.float64)

y = np.array([0, 0, 1, 0, 1, 0, 1, 0, 1, 1], dtype=np.float64)
```

Ввод [34]:
```
X
```

Out[34]:
```
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, 5.00e+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]])
```

Ввод [35]:
```
y
```

Out[35]:
```
array([0., 0., 1., 0., 1., 0., 1., 0., 1., 1.])
```

Ввод [36]:
```
def standard_scale(x):
    res = (x - x.mean()) / x.std()
    return res
```

Ввод [37]:
```
X_st = X.copy()
X_st[:, 2] = standard_scale(X[:, 2])
```

Ввод [38]:
```
X_st
```

Out[38]:
```
array([[ 1.        ,  1.        , -0.97958969,  1.        ],
       [ 1.        ,  1.        , -0.56713087,  1.        ],
       [ 1.        ,  2.        , -0.46401617,  2.        ],
       [ 1.        ,  5.        , -0.77336028,  1.        ],
       [ 1.        ,  3.        ,  0.97958969,  2.        ],
       [ 1.        ,  0.        , -0.36090146,  1.        ],
       [ 1.        ,  5.        ,  1.08270439,  3.        ],
       [ 1.        , 10.        ,  2.11385144,  3.        ],
       [ 1.        ,  1.        , -1.08270439,  1.        ],
       [ 1.        ,  2.        ,  0.05155735,  2.        ]])
```

Ввод [39]:
```
def calc_logloss(y, y_pred):
    err = - np.mean(y * np.log(y_pred) + (1.0 - y) * np.log(1.0 - y_pred))
    return err
```

Ввод [40]:
```
# Пример применения
y1 = np.array([1, 0])
y_pred1 = np.array([0.8, 0.1])
calc_logloss(y1, y_pred1)
```

Out[40]:
```
0.164252033486018
```

Ввод [41]:
```
# Плохой пример применения
y1 = np.array([1, 0])
y_pred1 = np.array([1, 0.2])
calc_logloss(y1, y_pred1)
```

```
<ipython-input-39-7d5907c1794a>:2: RuntimeWarning: divide by zero encountered in log
  err = - np.mean(y * np.log(y_pred) + (1.0 - y) * np.log(1.0 - y_pred))
<ipython-input-39-7d5907c1794a>:2: RuntimeWarning: invalid value encountered in multiply
  err = - np.mean(y * np.log(y_pred) + (1.0 - y) * np.log(1.0 - y_pred))
```

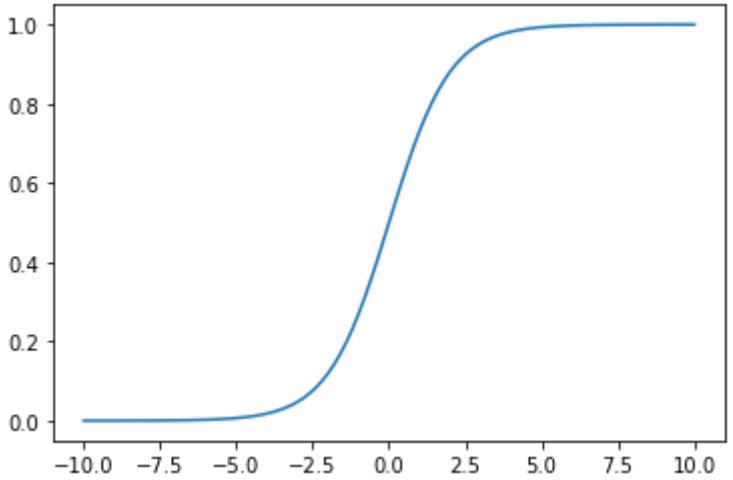
Out[41]:
```
nan
```

Ввод [42]:
```
def sigmoid(z):
    res = 1 / (1 + np.exp(-z))
    return res
```

Ввод [43]:
```
z = np.linspace(-10, 10, 101)
```

Ввод [44]:
```
probabilities = sigmoid(z)
```

Ввод [45]:
```
plt.plot(z, probabilities)
plt.show()
```



Logistic Regression

Ввод [46]:
```
def eval_model(X, y, iterations, eta=1e-4):
    np.random.seed(42)
    W = np.random.randn(X.shape[1])
    n = X.shape[0]

    for i in range(iterations):
        z = np.dot(X, W)
        y_pred = sigmoid(z)
        err = calc_logloss(y, y_pred)

        dQ = 1/n * X.T @ (y_pred - y)
        W -= eta * dQ
        if i % (iterations / 10) == 0:
            print(i, W, err)
    return W
```

Ввод [47]:
```
W = eval_model(X_st, y, iterations=500, eta=1e-4)
```

```
0 [ 0.49667621 -0.13840939  0.6476858   1.52297324] 1.1785958344356262
50 [ 0.494784   -0.14564801  0.6475462   1.52014828] 1.1657985749255426
100 [ 0.49290109 -0.15285535  0.64740132  1.51733474] 1.1531112685708473
150 [ 0.49102761 -0.16003088  0.64725118  1.51453281] 1.140535275330502
200 [ 0.48916364 -0.16717404  0.64709581  1.51174267] 1.1280719326917483
250 [ 0.48730929 -0.17428428  0.64693524  1.50896452] 1.1157225565960736
300 [ 0.48546465 -0.18136107  0.64676951  1.50619853] 1.1034884426224387
350 [ 0.48362982 -0.18840385  0.64659868  1.5034449 ] 1.0913708674192037
400 [ 0.48180488 -0.19541206  0.64642281  1.50070383] 1.0793710903721336
450 [ 0.47998993 -0.20238516  0.64624195  1.49797551] 1.0674903554915993
```

Домашнее задание

1. *Измените функцию calc_logloss так, чтобы нули по возможности не попадали в np.log.

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

3. Создайте функцию calc_pred_proba, возвращающую предсказанную вероятность класса 1 (на вход подаются W, который уже посчитан функцией eval_model и X, на выходе - массив y_pred_proba).

4. Создайте функцию calc_pred, возвращающую предсказанный класс (на вход подаются W, который уже посчитан функцией eval_model и X, на выходе - массив y_pred).

5. *Реализуйте функции для подсчета Accuracy, матрицы ошибок, точности и полноты, а также F1 score.

6. Могла ли модель переобучиться? Почему?

Проект *:

1. https://www.kaggle.com/c/gb-tutors-expected-math-exam-results (https://www.kaggle.com/c/gb-tutors-expected-math-exam-results) регрессия
2. https://www.kaggle.com/c/gb-choose-tutors (https://www.kaggle.com/c/gb-choose-tutors) классификация

Дополнительные материалы

1. Функции потерь для классификации (https://en.wikipedia.org/wiki/Loss_functions_for_classification)
2. Метод максимального правдоподобия: Сложное описание (https://habr.com/ru/company/ods/blog/323890/#metod-maksimalnogo-pravdopodobiya) / Простое описание (https://www.youtube.com/watch?v=2iRIqkm1mug)
3. Встроенные датасеты Sklearn (https://scikit-learn.org/stable/modules/classes.html#module-sklearn.datasets)
4. Площаль под кривой numpy.trapz (https://docs.scipy.org/doc/numpy-1.13.0/reference/generated/numpy.trapz.html)
5. Видео про метрику Accuracy (https://youtu.be/CCH-1gJo_z0)
6. Видео про метрики Precision, Recall (https://youtu.be/AfnBHL832Eg)
7. Видео про метрику F-score (https://youtu.be/PeE3Fkt5W3Q)
8. Видео про метрику PR-AUC (https://youtu.be/QW-09jHQH-w)

Summary

- Логистическая регрессия - частный случай линейной классификации - предсказывает вероятность отнесения объекта к основному классу, что зачастую очень важно при интерпретации
- Для "отображения" действительных предсказаний линейной модели в "вероятностный" интервал [0,1] применяют сигмоиду
- Для обучения логистической регрессии используют логарифмическую функцию потерь (log-loss), полученную методом максимального правдоподобия (maximum likelihood estimation)
- Оптимизируем log-loss классическим градиентным спуском, в котором берем градиент log-loss'a
- Основными метриками качества классификатора являются Accuracy, Precision, Recall, ROC-AUC, PR-AUC, F-мера
- Нужно быть внимательным при работе с этими метриками и хорошо понимать, как они работают и как между собой связаны, иначе выводы могут получиться неверными

Опередления

Масштабирование данных

Классификация — задача, в которой имеется множество объектов, разделённых некоторым образом на классы.

Линейный классификатор — алгоритм классификации, основанный на построении линейной разделяющей поверхности.

Отступ (для классификатора) — эвристика, оценивающая то, насколько объект принадлежит классу, насколько эталонным представителем он является.

Логистическая регрессия

Логистическая регрессия — метод построения линейного классификатора, позволяющий оценивать апостериорные вероятности принадлежности объектов классам.

Риск – отношение вероятности «положительный эффект» к вероятности «отрицательный эффект».

Логит – натуральный логарифм отношения вероятности «положительный эффект» к вероятности «отрицательный эффект».

Метрики качества классификации

Accuracy – доля правильных ответов.

$$accuracy(a, x) = \frac{1}{l}\sum_{i=1}^{l}[a(x_i) = y_i].$$

Точность (precision) – долю истинных срабатываний от общего количества срабатываний.

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

Полнота (recall) – доля объектов, истинно относящихся к выбранному классу, которые алгоритм отнес к этому классу.

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

F-мера – среднее гармоническое между точностью и полнотой.

$$F_\beta = (1 + \beta^2)\frac{precision \cdot recall}{\beta^2 \cdot precision + recall}.$$

ROC-кривая (receiver operating characteristic) — график, позволяющий оценить качество бинарной классификации, отображает соотношение между долей объектов от общего количества носителей признака, верно классифицированных как несущие признак (TPR), и долей объектов от общего количества объектов, не несущих признака, ошибочно классифицированных как несущие признак (FPR) при варьировании порога решающего правила.

PR-кривая — график, позволяющий оценить качество бинарной классификации, отображает соотношение между Precision и Recall.