Алгоритмы анализа данных

function_name? нажать **Ctrl+Enter**.
Чтобы посмотреть исходный код - **function_name??** нажать **Ctrl+Enter**.
Как вызвать справку jupyter notebook? https://ru.stackoverflow.com/questions/629610/Как-вызвать-справку-jupyter-notebook (https://ru.stackoverflow.com/questions/629610/%D0%9A%D0%B0%D0%BA-%D0%B2%D1%8B%D0%B7%D0%B2%D0%B0%D1%82%D1%8C-%D1%81%D0%BF%D1%80%D0%B0%D0%B2%D0%BA%D1%83-jupyter-notebook)

В [1]:
```
# Compute the arithmetic mean (среднее арифметическое) along the specified axis.
np.mean?
```

```
Object `np.mean` not found.
```

В [2]:
```
# Compute the standard deviation (стандартное отклонение) along the specified axis.
np.std??
```

```
Object `np.std` not found.
```

Урок 3. Логистическая регрессия. Log Loss

Практическое 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. Посчитайте Accuracy, матрицу ошибок, точность и полноту, а также F1 score.

6. Могла ли модель переобучиться? Почему?

7*. Создайте функции eval_model_l1 и eval_model_l2 с применением L1 и L2 регуляризаций соответственно.

В [3]:
```
import numpy as np
import matplotlib.pyplot as plt
```

В [4]:
```
# Матрица признаков
X = np.array([[1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
              [1, 1, 2, 1, 3, 0, 5, 10, 1, 2],
              [500, 700, 750, 600, 1450,
               800, 1500, 2000, 450, 1000],
              [1, 1, 2, 1, 2, 1, 3, 3, 1, 2]], dtype = np.float64)

# Вектор истинных значений
y = np.array([0, 0, 1, 0, 1, 0, 1, 0, 1, 1], dtype = np.float64)
```

В [5]:
```
def sigmoid(z):
    '''
    Сигмоида σ(<w,x>):ℝ→[0,1]

    Функция σ(<w,x>) переводит полученное в скалярном произведении значение ( z = W * X )
    в вероятность, пределы которой будут лежать в промежутке от 0 до 1.

    '''

    res = 1 / (1 + np.exp(-z))
    return res
```

В [6]:
```
def calc_std_feat(x):
    '''
    Масштабирование признаков методом стандартизации

    '''

    res = (x - x.mean()) / x.std()
    return res
```

В [7]:
```
# Масштабируем матрицу признаков
X_st = X.copy()
X_st[2, :] = calc_std_feat(X[2, :])
```

В [8]:
```
X_st
```

Out[8]:
```
array([[ 1.        ,  1.        ,  1.        ,  1.        ,  1.        ,
         1.        ,  1.        ,  1.        ,  1.        ,  1.        ],
       [ 1.        ,  1.        ,  2.        ,  1.        ,  3.        ,
         0.        ,  5.        , 10.        ,  1.        ,  2.        ],
       [-0.97958969, -0.56713087, -0.46401617, -0.77336028,  0.97958969,
        -0.36090146,  1.08270439,  2.11385144, -1.08270439,  0.05155735],
       [ 1.        ,  1.        ,  2.        ,  1.        ,  2.        ,
         1.        ,  3.        ,  3.        ,  1.        ,  2.        ]])
```

1*. Задача:

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

В [9]:
```
# Не изменённый вариант функции
def calc_logloss(y, y_pred):
    '''
    Логарифмическая функция потерь

    y - вектор истинных значений
    y_pred = P+ вероятность отнесения объекта к классу +1: P(y=1|x)

    '''

    err = - np.mean(y * np.log(y_pred) + (1.0 - y) * np.log(1.0 - y_pred))
    return err
```

Изменяем функцию calc_logloss так, чтобы нули по возможности не попадали в np.log.

В [10]:
```
# Изменённый вариант функции
def calc_logloss(y, y_pred):
    '''
    Логарифмическая функция потерь  (нули не попадают в np.log.)

    y - вектор истинных значений
    y_pred = P+ вероятность отнесения объекта к классу +1: P(y=1|x)

    '''

    # Обрабатываем 0 и 1 из вектора вероятностей (0+0.01, 1-0.01).
    # Можно проварьировать параметр изменения вероятности.
    for i in range(len(y_pred)):
        if (y_pred[i] == 0):
            y_pred[i] = 1e-2
        if (y_pred[i] == 1):
            y_pred[i] -= 1e-2

    # print(len(y_pred), len(z_y_pred), z_y_pred, z_y)
    err = - np.mean(y * np.log(y_pred) + (1.0 - y) * np.log(1.0 - y_pred))
    return err
```

В [11]:
```
# Проверка
y_1 = np.array([0, 1], dtype = np.float64)
y_pred_1 = np.array([0, 0.1], dtype = np.float64)
y_1, y_pred_1
```

Out[11]:
```
(array([0., 1.]), array([0. , 0.1]))
```

В [12]:
```
calc_logloss(y_1, y_pred_1)
```

Out[12]:
```
1.1563177144237735
```

2. Задача:

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

$\alpha \cdot \nabla_w Q(w, X)$

В [13]:
```
### Logistic Regression
```

В [14]:
```
def eval_model(X, y, iterations, alpha=1e-4):
    '''
    Обучаем модель методом градиентного спуска

    iterations - количество итераций (условие прерывания алгоритма)
    alpha - скорость обучения

    '''

    # задаем начальные условия для генератора случайных чисел (повторяемость результата).
    np.random.seed(42)

    # сгенерируем вектор истинных весов
    # возвращаем несколько значений выборки из стандартного нормального распределения (X.shape[0]=4).
    W = np.random.randn(X.shape[0])
    n = X.shape[1]

    # ход градиентного спуска
    for i in range(1, iterations+1):

        # делаем предсказание z=W*X (умножаем вектор W на матрицу признаков X
        z = np.dot(W, X)

        # переводим значения z в вероятности (от 0 до 1)
        y_pred = sigmoid(z)

        # Вычисляем ошибку на i шаге Q(w,X)
        err = calc_logloss(y, y_pred)

        # Вычисляем новое значение вектора весов  W_new = W - α·∇w(Q(w,X))
        W -= alpha * (1/n * np.dot((y_pred - y), X.T))

    if i % (iterations / 10) == 0:
        print(f'i={i}, W={W}, err={err:.6f}')

    return W
```

В [15]:
```
# Неизменённая функция calc_logloss(y, y_pred):
# W - истинный вектор весов
W = eval_model(X_st, y, iterations=1000, alpha=1e-5)
W = eval_model(X_st, y, iterations=10000, alpha=1e-5)
W = eval_model(X_st, y, iterations=100000, alpha=1e-5)
```

```
i=1000, W=[ 0.49282757 -0.15007512  0.64748969  1.51727928], err=1.201313
i=10000, W=[ 0.45886981 -0.25439705  0.6453131   1.46695998], err=1.039365
i=100000, W=[ 0.25809285 -0.68188567  0.6883469   1.2412009 ], err=0.590673
```

В [16]:
```
W = eval_model(X_st, y, iterations=1000, alpha=1e-1)
W = eval_model(X_st, y, iterations=10000, alpha=1e-1)
W = eval_model(X_st, y, iterations=100000, alpha=1e-1)
```

```
i=1000, W=[-2.77136565 -0.99588853  0.56641089  3.26813012], err=0.405878
i=10000, W=[-11.27241705  -1.45342424  -2.38315559   9.49424167], err=0.252383
i=100000, W=[-35.8933164   -3.59426199  -9.64639296  29.4060647 ], err=0.116671
```

В [17]:
```
W = eval_model(X_st, y, iterations=1000, alpha=1)
W = eval_model(X_st, y, iterations=10000, alpha=1)
W = eval_model(X_st, y, iterations=100000, alpha=1)
```

```
i=1000, W=[-11.32397854  -1.45756231  -2.39926134   9.53555126], err=0.251910
i=10000, W=[-35.907977    -3.59554341  -9.65060282  29.4178451 ], err=0.116635
i=100000, W=[-143.31661218  -17.65497665  -32.28077915  127.74350957], err=0.040484
```

В [53]:
```
W = eval_model(X_st, y, iterations=1000, alpha=10)
W = eval_model(X_st, y, iterations=10000, alpha=10)
W = eval_model(X_st, y, iterations=100000, alpha=10)
```

```
i=1000, W=[-67.20839203  -6.37997719 -18.2521044   54.83753943], err=0.070169
i=10000, W=[-181.33758972  -23.14529545  -39.39598481  163.87960615], err=0.033711
i=100000, W=[-1311.91153553  -179.12343794  -259.00442698  1215.43684289], err=0.010984
```

В [52]:
```
W = eval_model(X_st, y, iterations=1000, alpha=20)
W = eval_model(X_st, y, iterations=10000, alpha=20)
W = eval_model(X_st, y, iterations=100000, alpha=20)
```

```
i=1000, W=[-155.6157941   -16.9005628   -39.74544129  134.7052772 ], err=0.118530
i=10000, W=[-388.9057345   -51.72962728  -81.01044166  356.53899761], err=0.015253
i=100000, W=[-2633.23525331  -362.18637934  -512.8012476   2445.20864538], err=0.009982
```

Вывод:

log loss минимальный при:
Скорость обучения α=20,
Количество итераций **i**=100000,

Вектор весов **W**=[-2633.23525331 -362.18637934 -512.8012476 2445.20864538],
Ошибка **err**=0.009982

3. Задача:

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).

В [20]:
```
def calc_pred_proba(W, X):
    ''' Возвращающую предсказанную вероятность класса 1 '''

    z = np.dot(W, X)
    y_pred_proba = sigmoid(z)

    return y_pred_proba
```

В [21]:
```
calc_pred_proba(W, X_st)
```

```
<ipython-input-5-d60a7b44af67>:10: RuntimeWarning: overflow encountered in exp
  res = 1 / (1 + np.exp(-z))
```

Out[21]:
```
array([2.32546235e-117, 0.00000000e+000, 1.00000000e+000, 0.00000000e+000,
       1.00000000e+000, 3.75344497e-002, 1.00000000e+000, 7.88496116e-283,
       9.99882708e-001, 1.00000000e+000])
```

4. Задача:

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

В [22]:
```
def calc_pred(W, X, porog):
    ''' Возвращающую предсказанную вероятность класса 1 '''

    y_pred = np.array([])
    z = np.dot(W, X)
    y_pred_proba = sigmoid(z)

    for i in range(len(y_pred_proba)):
        if (y_pred_proba[i] > porog):
            y_pred = np.append(y_pred, 1)
        else:
            y_pred = np.append(y_pred, 0)

    return y_pred
```

В [23]:
```
calc_pred(W, X_st, 0.5), y
```

```
<ipython-input-5-d60a7b44af67>:10: RuntimeWarning: overflow encountered in exp
  res = 1 / (1 + np.exp(-z))
```

Out[23]:
```
(array([0., 0., 1., 0., 1., 0., 1., 0., 1., 1.]),
 array([0., 0., 1., 0., 1., 0., 1., 0., 1., 1.]))
```

5. Задача:

Посчитайте Accuracy, матрицу ошибок, точность и полноту, а также F1 score.

В [24]:
```
# Ради интереса за основу примем не оптимальную модель
W = eval_model(X_st, y, iterations=1000, alpha=1e-1)
```

```
i=1000, W=[-2.77136565 -0.99588853  0.56641089  3.26813012], err=0.405878
```

В [25]:
```
y_pred = calc_pred(W, X_st, 0.5)
y_pred, y
```

Out[25]:
```
(array([0., 0., 1., 0., 1., 1., 1., 0., 0., 1.]),
 array([0., 0., 1., 0., 1., 0., 1., 0., 1., 1.]))
```

Доля правильных ответов (Accuracy):

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

В [26]:
```
def calc_accuracy(y, w, x, porog):
    '''
    Расчёт доли правильных ответов (accuracy)

    porog - порог отнесения к классу 1

    '''

    y_pred = calc_pred(W, X_st, porog)

    k = 0
    for i in range(len(y_pred)):
        if (y_pred[i] == y[i]):
            k += 1

    return k/len(y_pred)
```

В [27]:
```
# В нашем случае у нас есть
accuracy = calc_accuracy(y, W, X_st, 0.5)
accuracy
```

Out[27]:
```
0.8
```

В [28]:
```
# Проверка
import  sklearn.metrics
from sklearn.metrics import accuracy_score
```

В [29]:
```
y_pred
```

Out[29]:
```
array([0., 0., 1., 0., 1., 1., 1., 0., 0., 1.])
```

В [30]:
```
acc = accuracy_score(y, y_pred)
acc
```

Out[30]:
```
0.8
```

Матрица ошибок (Confusion matrix)

	$y = +1$	$y = -1$
$a(x) = +1$	TP	FP
$a(x) = -1$	FN	TN

В [31]:
```
def calc_confusion_matrix(y_true, y_pred):
    ''' Вычисляем матрицу ошибок (confusion matrix)
    для двух np.arrays true и pred.
    Аналог:
    "from sklearn.metrics import confusion_matrix"
    '''

    K = len(np.unique(y_true)) # Number of classes
    res = np.zeros((K, K))

    for i in range(len(y_true)):
        res[int(y_true[i])][int(y_pred[i])] += 1

    return res
```

В [32]:
```
# Вычисляем матрицу ошибок для нашего случая
conf_matrix = calc_confusion_matrix(y, y_pred)
conf_matrix
```

Out[32]:
```
array([[4., 1.],
       [1., 4.]])
```

В [33]:
```
# Проверка
from sklearn.metrics import confusion_matrix

confusion_matrix(y, y_pred)
```

Out[33]:
```
array([[4, 1],
       [1, 4]], dtype=int64)
```

Результаты совпадают

Точность (Precision) - доля истинных срабатываний от общего количества срабатываний.

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

В [34]:
```
precision = conf_matrix[0][0]/(conf_matrix[0][0] + conf_matrix[0][1])
precision
```
Out[34]: 0.8

В [35]:
```
# Проверка
from sklearn.metrics import precision_score

precision = precision_score(y, y_pred, pos_label=1)
precision
```
Out[35]: 0.8

Полнота (Recall) - доля объектов, истинно относящихся к классу "+1", которые алгоритм отнес к этому классу

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

В [36]:
```
recall = conf_matrix[0][0]/(conf_matrix[0][0] + conf_matrix[1][0])
recall
```
Out[36]: 0.8

В [37]:
```
# Проверка
from sklearn.metrics import recall_score

recall = recall_score(y, y_pred, pos_label=1)
recall
```
Out[37]: 0.8

F-мера (F1 score)

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

В [38]:
```
F = 2*(precision*recall)/(precision + recall)
F
```
Out[38]: 0.8000000000000002

В [39]:
```
# Проверка
# sklearn.metrics.f1_score - https://scikit-learn.org/stable/modules/generated/sklearn.metrics.f1_score.html
from sklearn.metrics import f1_score

f1_score(y, y_pred, average='macro')
```
Out[39]: 0.8000000000000002

6. Задача:

Могла ли модель переобучиться? Почему?

Переобучение - модель настроилась на шумы, а не на общие закономерности и тренды.

7*. Задача:

Создайте функции eval_model_l1 и eval_model_l2 с применением L1 и L2 регуляризаций соответственно.

L1-регуляризация способствует разреженности функции, когда лишь немногие факторы не равны нулю.
L2-регуляризация способствует появлению малых весовых коэффициентов модели, но не способствует их точному равенству нулю.
Оба метода помогают улучшить обобщение и ошибки, поскольку не допускают переобучения модели из-за шума в данных.
При L2-регуляризации дополнительный член является квадратичной функцией, при L1-регуляризации – модулем.

При квадратичном члене, чем ближе вы находитесь к нулю, тем меньшей становится ваша производная, пока также не приблизится к нулю. Поэтому при L2-регуляризации, когда ваша величина w уже мала, дальнейший градиентный спуск уже её сильно не изменит.
В случае модуля производная является константой с абсолютной величиной, равной единице. Формально в нуле она не определена, но мы считаем её также равной нулю. Поэтому при L1-регуляризации градиентный спуск будет стремиться к нулю с постоянной скоростью, а достигнув его, там и останется. Вследствие этого L2-регуляризация способствует малой величине весовых коэффициентов, а L1-регуляризация способствует их равенству нулю, тем самым провоцируя разрежённость.

Можно включить в свою модель сразу и L1, и L2-регуляризации. Такая модель даже имеет специальное название – **ElasticNet**. Это просто добавление и штрафа L1-регуляризации, и штрафа L2-регуляризации к вашей функции затрат.

$$\begin{aligned} J_{RIDGE} &= J + \lambda_2 |w|^2 \\ J_{LASSO} &= J + \lambda_1 |w| \\ J_{ELASTICNET} &= J + \lambda_1 |w| + \lambda_2 |w|^2 \end{aligned}$$

В [40]:
```
# W - истинный вектор весов
W = eval_model(X_st, y, iterations=1000, alpha=1e-5)
W = eval_model(X_st, y, iterations=10000, alpha=1e-5)
W = eval_model(X_st, y, iterations=100000, alpha=1e-5)
```

```
i=1000, W=[ 0.49282757 -0.15007512  0.64748969  1.51727928], err=1.201313
i=10000, W=[ 0.45886981 -0.25439705  0.6453131   1.46695998], err=1.039365
i=100000, W=[ 0.25809285 -0.68188567  0.6883469   1.2412009 ], err=0.590673
```

В [41]:
```
# Returns an element-wise indication of the sign of a number.
# Функция sign(x) возвращает единицу, если x>0,  минус единицу, если x<0, и нуль, если x=0.
# np.sign??
```

В [42]:
```
def eval_model_l1(X, y, iterations, alpha=1e-4):
    '''
    Градиентный спуск
    L1-регуляризация

    iterations - количество итераций (условие прерывания алгоритма)
    alpha - скорость обучения

    '''

    # задаем начальные условия для генератора случайных чисел (повторяемость результата).
    np.random.seed(42)

    # сгенерируем вектор истинных весов
    # возвращаем несколько значений выборки из стандартного нормального распределения (X.shape[0]=4).
    W = np.random.randn(X.shape[0])
    n = X.shape[1]

    # ход градиентного спуска
    for i in range(1, iterations+1):

        # делаем предсказание z=W*X (умножаем вектор W на матрицу признаков X
        z = np.dot(W, X)

        # переводим значения z в вероятности (от 0 до 1)
        y_pred = sigmoid(z)

        # Вычисляем ошибку на i шаге Q(w,X)
        err = calc_logloss(y, y_pred)

        # Вычисляем новое значение вектора весов (L2-регуляризация) W_new = W - α·∇w(Q(w,X)) + lambda*|W|
        W -= alpha * (1/n * np.dot((y_pred - y), X.T) + 2*np.sign(W))

    if i % (iterations / 10) == 0:
        print(f'i={i}, W={W}, err={err:.6f}')

    return W
```

В [43]:
```
# W - истинный вектор весов
W = eval_model_l1(X_st, y, iterations=1000, alpha=1e-5)
W = eval_model_l1(X_st, y, iterations=10000, alpha=1e-5)
W = eval_model_l1(X_st, y, iterations=100000, alpha=1e-5)
```

```
i=1000, W=[ 0.4728311  -0.13007488  0.627488    1.49728315], err=1.205278
i=10000, W=[ 0.25925794 -0.05452568  0.44509247  1.26736528], err=1.080879
i=100000, W=[ 3.74206502e-06  7.41569932e-06 -1.41768584e-05 -6.88292739e-06], err=0.693145
```

В [44]:
```
def eval_model_l2(X, y, iterations, alpha=1e-4):
    '''
    Градиентный спуск
    L2-регуляризация

    iterations - количество итераций (условие прерывания алгоритма)
    alpha - скорость обучения

    '''

    # задаем начальные условия для генератора случайных чисел (повторяемость результата).
    np.random.seed(42)

    # сгенерируем вектор истинных весов
    # возвращаем несколько значений выборки из стандартного нормального распределения (X.shape[0]=4).
    W = np.random.randn(X.shape[0])
    n = X.shape[1]

    # ход градиентного спуска
    for i in range(1, iterations+1):

        # делаем предсказание z=W*X (умножаем вектор W на матрицу признаков X
        z = np.dot(W, X)

        # переводим значения z в вероятности (от 0 до 1)
        y_pred = sigmoid(z)

        # Вычисляем ошибку на i шаге Q(w,X)
        err = calc_logloss(y, y_pred)

        # Вычисляем новое значение вектора весов (L2-регуляризация) W_new = W - α·∇w(Q(w,X)) + lambda*W
        W -= alpha * (1/n * np.dot((y_pred - y), X.T) + 0.1*W)

    if i % (iterations / 10) == 0:
        print(f'i={i}, W={W}, err={err:.6f}')

    return W
```

В [45]:
```
# W - истинный вектор весов
W = eval_model_l2(X_st, y, iterations=1000, alpha=1e-5)
W = eval_model_l2(X_st, y, iterations=10000, alpha=1e-5)
W = eval_model_l2(X_st, y, iterations=100000, alpha=1e-5)
```

```
i=1000, W=[ 0.4923338  -0.14992995  0.64684208  1.51576088], err=1.200410
i=10000, W=[ 0.45419937 -0.25229978  0.63885105  1.45220195], err=1.031865
i=100000, W=[ 0.23185286 -0.61912998  0.62525208  1.12672326], err=0.590607
```

В [ ]:

В [ ]:

В [ ]:

В [ ]:

В [ ]:

В [ ]:

В [ ]:

В [ ]:

L1 и L2-регуляризация для логистической регрессии - https://craftappmobile.com/l1-и-l2-регуляризация-для-логистической-регрессии (https://craftappmobile.com/l1-%D0%B8-l2-%D1%80%D0%B5%D0%B3%D1%83%D0%BB%D1%8F%D1%80%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F-%D0%B4%D0%BB%D1%8F-%D0%BB%D0%BE%D0%B3%D0%B8%D1%81%D1%82%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%BE%D0%B9-%D1%80%D0%B5%D0%B3%D1%80%D0%B5%D1%81%D1%81%D0%B8%D0%B8)
Обучение логистической регрессии - https://vc.ru/dev/72964-obuchenie-logisticheskoy-regressii (https://vc.ru/dev/72964-obuchenie-logisticheskoy-regressii)

Как написать матрицу путаницы в Python? - https://coderoad.ru/2148543/Как-написать-матрицу-путаницы-в-Python (https://coderoad.ru/2148543/%D0%9A%D0%B0%D0%BA-%D0%BD%D0%B0%D0%BF%D0%B8%D1%81%D0%B0%D1%82%D1%8C-%D0%BC%D0%B0%D1%82%D1%80%D0%B8%D1%86%D1%83-%D0%BF%D1%83%D1%82%D0%B0%D0%BD%D0%B8%D1%86%D1%8B-%D0%B2-Python)
Оценка моделей ML/DL: матрица ошибок, Accuracy, Precision и Recall - https://pythonru.com/baza-znanij/metriki-accuracy-precision-i-recall (https://pythonru.com/baza-znanij/metriki-accuracy-precision-i-recall)

В [46]:
```
import pandas as pd
y_actu = pd.Series([2, 0, 2, 2, 0, 1, 1, 2, 2, 0, 1, 2], name='Actual')
y_pred = pd.Series([0, 0, 2, 1, 0, 2, 1, 0, 2, 0, 2, 2], name='Predicted')
df_confusion_1 = pd.crosstab(y_actu, y_pred),
df_confusion = pd.crosstab(y_actu, y_pred, rownames=['Actual'], colnames=['Predicted'], margins=True)
df_confusion, df_confusion_1
```

Out[46]:
```
(Predicted  0  1  2  All
 Actual
 0          3  0  0    3
 1          0  1  2    3
 2          2  1  3    6
 All        5  2  5   12,
 (Predicted  0  1  2
  Actual
  0          3  0  0
  1          0  1  2
  2          2  1  3,))
```

В [47]:
```
#Вы также можете получить нормализованную матрицу путаницы, используя:

df_conf_norm = df_confusion / df_confusion.sum(axis=1)
df_conf_norm
```

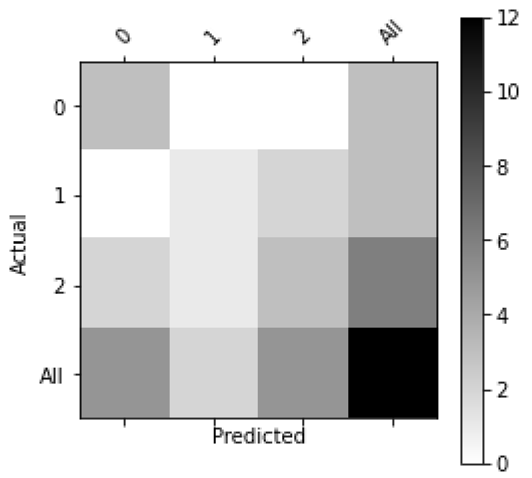
Out[47]:

Predicted Actual	0	1	2	All
0	0.500000	0.000000	0.000000	0.125
1	0.000000	0.166667	0.166667	0.125
2	0.333333	0.166667	0.250000	0.250
All	0.833333	0.333333	0.416667	0.500

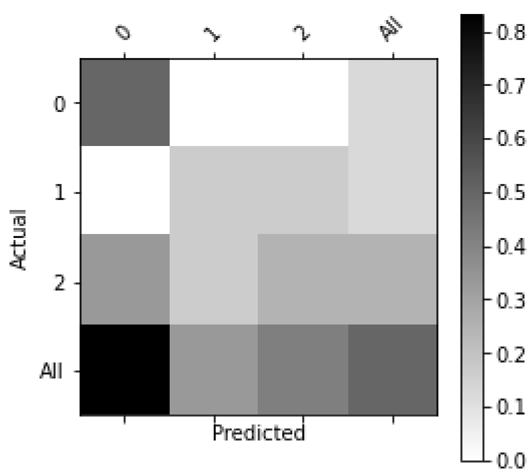
В [48]:
```
# Вы можете построить эту confusion_matrix с помощью

import matplotlib.pyplot as plt
def plot_confusion_matrix(df_confusion, title='Confusion matrix', cmap=plt.cm.gray_r):
    plt.matshow(df_confusion, cmap=cmap) # imshow
    #plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(df_confusion.columns))
    plt.xticks(tick_marks, df_confusion.columns, rotation=45)
    plt.yticks(tick_marks, df_confusion.index)
    #plt.tight_layout()
    plt.ylabel(df_confusion.index.name)
    plt.xlabel(df_confusion.columns.name)

plot_confusion_matrix(df_confusion)
```



В [49]:
```
plot_confusion_matrix(df_conf_norm)
```



В [ ]: