In [1]:
```
import pandas as pd
import numpy as np
import scipy.stats as sps
import matplotlib.pyplot as plt
```

In [12]:
```
f = open('data.txt')

pre = -1, -1

x = []
y = []
z = []
ans = []

x.append([])
y.append([])
z.append([])

for line in f:
    arr = line.split(',')

    if (len(arr) != 6):
        continue

    num = arr[0]
    d = arr[1]

    if (pre[0] == -1):
        ans.append(d)
        pre = num, d

    if (pre[0] != num or pre[1] != d):
        x.append([])
        y.append([])
        z.append([])
        pre = num, d
        ans.append(d)

    arr[5] = arr[5].split(';')[0]

    x[-1].append(arr[3])
    y[-1].append(arr[4])
    z[-1].append(arr[5])
```

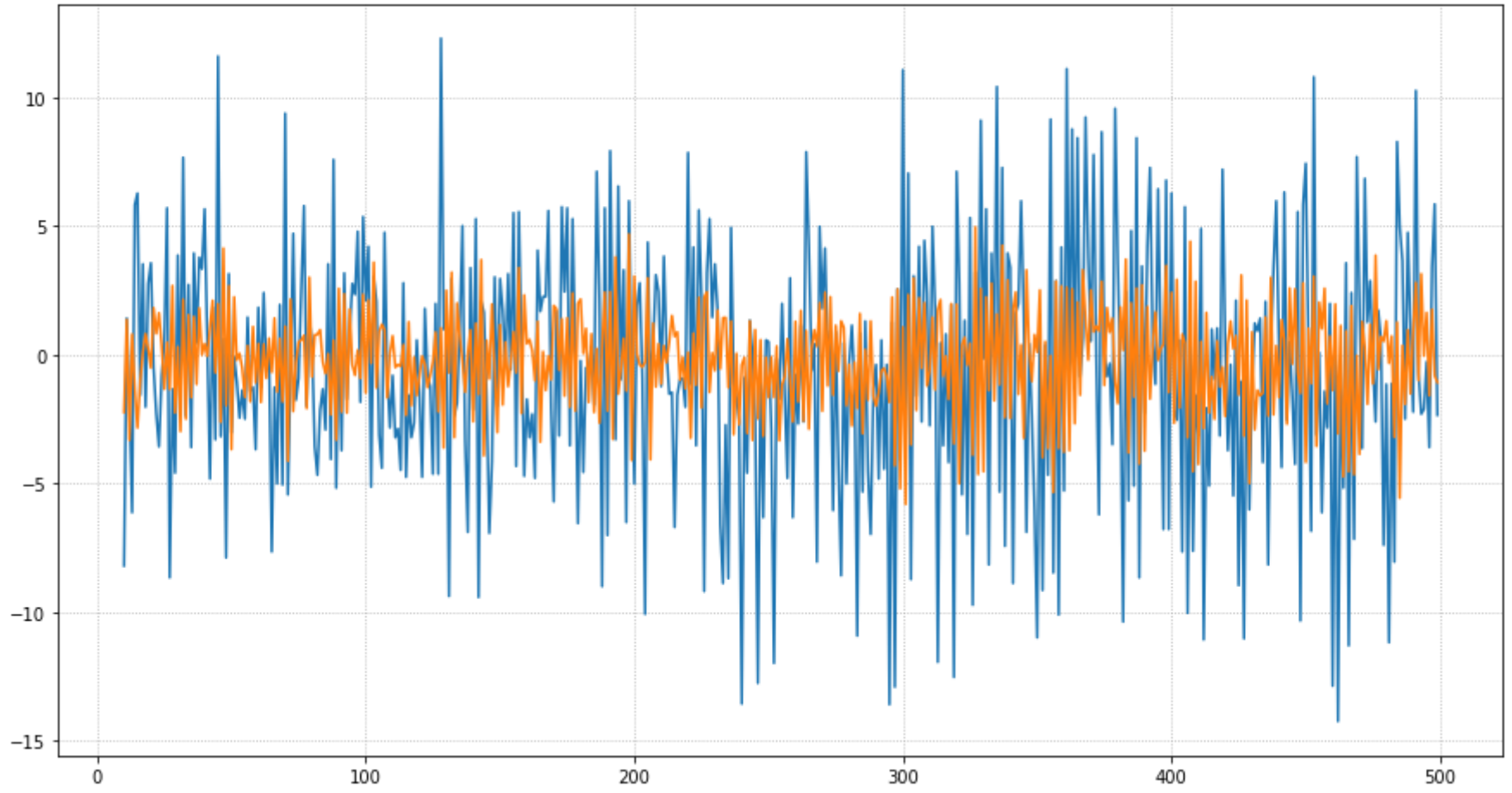
In [248]:
```
from tqdm import tqdm

def conv(x):
    for i in range(len(x)):
        try:
            x[i] = float(x[i])
        except:
            x[i] = 0
    return np.array(x)

def score(A, label, c):
    ans = 0

    for i in range(len(A)):
        pred = np.sum(A[i] * c)

        ans += (pred - label[i]) ** 2
    '''
    return ans'''

    pred = A @ np.array([c]).T

    pred = pred.T[0]

    return np.sum((pred - label) * (pred - label))

def get_gr(A, label, c):
    c = np.array(c)

    pred = A @ np.array([c]).T
    pred = pred.T[0]

    gr = (np.array([pred - label]) * 2) @ A
    gr = gr[0]

    gr /= len(A)
    return gr

def auto_cor(x, p):
    c = np.zeros(p)
```

```
    deep = 100

    A = []
    label = []

    for i in range(len(x) - p):
        A.append(x[i: i + p])
        label.append(x[i + p])

    A = np.array(A)
    label = np.array(label)

    for it in range(deep):
        gr = -get_gr(A, label, c)

        alpha = 0.2
        s = 0.1

        beta = 0.5
        cur = score(A, label, c)

        while (score(A, label, c + gr * s) > cur - s * alpha * np.sum(gr * gr)):
            s *= beta

        c += gr * s

    return c

def predict(x, c):
    ans = []

    for i in range(len(x)):
        if (i < len(c)):
            ans.append(x[i])
        else:
            ans.append(np.sum(x[i - len(c) : i] * c))

    return ans

def encode(label):
    g = []
    for i in range(len(label)):
```

```
        if (label[i] == 'Walking'):
            g.append(0)
        elif (label[i] == 'Jogging'):
            g.append(1)
        elif (label[i] == 'Upstairs'):
            g.append(2)
        elif (label[i] == 'Downstairs'):
            g.append(3)
        elif (label[i] == 'Sitting'):
            g.append(4)
        else:
            g.append(5)

    return np.array(g)

def decode(it):
    if (it == 0):
        return 'Walking'
    elif (it == 1):
        return 'Jogging'
    elif (it == 2):
        return 'Upstairs'
    elif (it == 3):
        return 'Downstairs'
    elif (it == 4):
        return 'Sitting'
    else:
        return 'Standing'

def info(ans, y_ans, method):
    print(method, ':')
    print("Суммарное качество =", np.sum(ans == encode(y_ans)) / len(ans) * 100, '%')
    ey = encode(y_ans)

    for i in range(6):
        ind = ey == i

        print("Качество для ", decode(i), "=", np.sum(ans[ind] == ey[ind]) / np.sum(ind) * 100, '%')
```

In [14]:
```
c = auto_cor(conv(x[0]), 10)
```

In [15]:
```
plt.figure(figsize=(15, 8))
plt.plot(np.arange(len(x[0]))[10:500], conv(x[0])[10:500])
plt.plot(np.arange(len(x[0]))[10:500], predict(conv(x[0]), c)[10:500])
plt.grid(ls=':')
plt.show()
```



In [19]:
```
def get_param(x):
    gr = np.linspace(np.min(x), np.max(x), 11)

    res = []

    for i in range(10):
        ind = x >= gr[i]
        res.append(np.sum(x[ind] < gr[i + 1]))

    return np.array(res) / len(x)

pt = []
y_train = []

for i in tqdm(range(350)):
    if (len(x[i]) < 200):
        continue

    x[i] = conv(x[i])
    y[i] = conv(y[i])
    z[i] = conv(z[i])

    for j in range(0, len(x[i]) // 200):
        xx = x[i][j * 200: (j + 1) * 200]
        yy = y[i][j * 200: (j + 1) * 200]
        zz = z[i][j * 200: (j + 1) * 200]
        pt.append(np.concatenate((auto_cor(xx, 10), auto_cor(yy, 10),
                                  auto_cor(zz, 10), [np.mean(xx)], [np.mean(yy)],
                                  [np.mean(zz)], [np.std(xx)], [np.std(yy)], [np.std(zz)],
                                  [np.mean(xx * xx + yy * yy + zz * zz)])))
        y_train.append(ans[i])
    break
```

```
  0%|          | 0/350 [00:00<?, ?it/s]
```

```
[-0.6946377   5.012288    4.903325   -0.61291564 -1.1849703   1.3756552
 -0.61291564 -0.50395286 -8.430995    0.95342433 -8.19945     1.4165162
 -1.879608   -6.1291566   5.829509    6.2789803  -1.56634     3.5276701
 -2.0294318   2.7649305   3.568531   -0.50395286 -2.3018389  -3.568531
 -0.8036005   0.50395286  5.706926   -8.662541   -1.334794   -4.5900574
  3.8681788  -1.7978859   7.668256   -2.3699405   2.7240696  -3.5957718
  3.9499009   0.46309182  3.7864566   3.336985    5.6660647   0.23154591
```

```
 -4.8216033   1.8387469  -3.2961242  11.604536   -3.173541    0.61291564
 -7.8861814   3.1463003  -3.0237172  -0.08172209 -1.0351465  -2.4516625
 -1.3756552  -2.4925237   1.4573772  -1.4165162  -1.2666923  -3.6774938
  1.8387469  -1.2666923   2.4108016  -0.61291564  0.04086104 -7.6546354
 -1.2666923  -5.012288    1.9477097  -5.053149    9.384419   -5.434519
 -0.61291564  4.7126403  -1.7570249  -0.9125633   2.6014864   5.7886477
 -1.9885708   1.4165162   0.42223078 -3.568531   -4.671779   -2.1383946
 -1.334794   -2.9147544   3.5276701  -4.0588636   7.5865335  -5.175732
  1.1849703  -3.718355    3.173541   -1.7297841   0.84446156  2.7649305
  2.3426998   4.7943625  -1.8387469   5.366417    2.1111538   4.2086873
 -5.134871    3.0237172   2.070293   -3.1054392  -4.399372    4.7535014
 -0.6946377  -2.8330324  -0.8036005  -3.214402   -2.8738933  -4.4810944
  2.7921712  -4.7535014  -1.56634    -3.214402   -2.6423476   0.5720546
 -1.525479   -4.7535014   1.7978859  -0.8036005  -1.2258313  -4.630918
  1.9885708  -4.630918   12.299174    0.9942854   1.0351465  -9.384419
  2.9556155  -2.3699405  -1.879608    1.1168685   5.012288   -3.336985
 -6.891896    3.3778462  -2.1383946   5.284695   -9.425281    2.070293
  1.6480621  -1.56634    -6.932757   -4.140586    3.0237172  -1.9885708
  2.9556155   1.7297841  -0.42223078  3.1463003  -0.5720546   5.5162406
 -4.3312707   5.5571017   0.50395286 -4.7126403  -1.7297841  -3.214402
 -2.3018389  -4.7943625   4.0588636   1.6889231   2.2609777   2.2609777
  5.597963   -2.070293   -5.706926    1.6889231  -3.1463003   5.7477865
  2.4516625   5.706926   -3.5276701   5.284695    0.08172209 -6.5513873
 -0.23154591 -4.5628166  -0.50395286 -0.50395286  0.313268   -1.9477097
  7.1234417   2.152015   -9.00305     5.706926   -7.014479    7.9270425
 -0.         -3.2961242   6.5513873  -0.95342433  3.2961242  -6.510526
  5.9793324  -0.5720546 ]
[ -5.012288     1.920469     2.7921712   -0.95342433 -10.079058
   4.372132    -1.3075534    0.38136974   3.1054392    2.4516625
  -1.2258313    3.8273177    0.08172209  -1.525479    -1.4573772
  -6.701211    -1.56634     -0.95342433  -0.9125633   -2.0294318
   7.8589406   -1.1441092    4.1814466   -3.5276701    5.6252036
   2.2201166   -9.193735     3.173541     5.284695     1.4573772
   3.5276701    1.7297841   -6.5922484   -8.880466    -2.7240696
  -8.689782     4.944186    -1.1168685   -1.6480621   -2.5606253
 -13.565866    -0.88532263  -4.5900574    1.334794     0.08172209
  -2.4925237  -12.762266    -0.5720546   -6.3198414    0.5720546
   0.50395286  -2.9147544  -11.985906    -0.14982383  -1.7297841
   1.9885708   -1.56634     -4.7943625    2.982856    -6.3198414
  -0.7627395   -2.6832085    1.1441092   -2.6014864    7.8861814
   4.2086873   -0.61291564   0.42223078  -8.049625     4.9850473
   1.7978859    4.140586    -0.10896278   0.92618364  -6.0474343
```

```
  0.88532263  -4.3312707   -8.580819     0.46309182  -5.012288
 -0.3405087    1.1441092   -1.1441092  -10.923519     0.84446156
 -5.325556     1.3075534   -4.5219555   -6.973618    -1.334794
 -0.38136974  -4.8216033    0.5720546   -4.440233    -0.38136974
-13.593107     1.2666923  -12.912089     2.5606253   -4.099725
 11.073342    -3.8681788    7.0553403   -8.730643     3.0645783
 -1.607201     4.2086873   -2.6014864    4.440233     2.3699405
 -2.7513103    4.9850473    1.7978859  -11.9450445    0.46309182
 -3.5276701    0.8036005   -4.1814466    1.9477097  -12.53072
  7.1234417    2.7921712   -5.434519     1.334794    -6.973618
  5.325556    -9.724928     3.214402    -2.1111538    9.112013
 -0.14982383   5.6660647   -8.158588     3.9499009   -0.10896278
 10.419566    -5.325556     7.273266    -7.43671      3.9499009
  3.3778462   -8.880466     1.607201     1.9885708    5.9793324
  1.5390993   -6.891896     0.42223078  -2.3426998   -6.0201936
-10.991621     1.56634     -9.152874     0.38136974  -4.671779
  9.152874    -8.471856     2.8738933  -10.106298     4.1814466
 -5.284695    11.114203    -1.4165162    8.771504    -1.334794
  8.430995    -0.84446156   0.6537767    9.234595     3.7864566
  0.53119355   7.7772183   -0.14982383  -6.2108784    8.662541
  0.08172209  -0.84446156  -0.3405087   -3.486809     9.575105
  3.568531    -0.6946377  -10.378705     2.4516625   -5.6660647
  4.8216033   -5.09401      8.430995    -8.662541     3.445948
 -2.2609777    4.2086873    7.273266     0.38136974  -1.1441092
  6.4424243    0.14982383  -6.782933     6.782933    -6.782933  ]
```

In [598]:
```
test = []
y_test = []

for i in tqdm(range(350, len(x))):
    if (len(x[i]) < 200):
        continue

    x[i] = conv(x[i])
    y[i] = conv(y[i])
    z[i] = conv(z[i])

    for j in range(0, len(x[i]) // 200):
        xx = x[i][j * 200: (j + 1) * 200]
        yy = y[i][j * 200: (j + 1) * 200]
        zz = z[i][j * 200: (j + 1) * 200]
        test.append(np.concatenate((auto_cor(xx, 10), auto_cor(yy, 10),
                                    auto_cor(zz, 10), [np.mean(xx)], [np.mean(yy)],
                                    [np.mean(zz)], [np.std(xx)], [np.std(yy)], [np.std(zz)],
                                    [np.mean(xx * xx + yy * yy + zz * zz)])))
        y_test.append(ans[i])
```

```
100%|██████████| 52/52 [20:06<00:00, 23.19s/it]
```

In [43]:
```
from sklearn.neighbors import KNeighborsClassifier as KNN
```

In [632]:
```
knn = KNN(n_neighbors=5, n_jobs=-1)
knn.fit(pt, encode(y_train))

res = knn.predict(test)
#print(res)
#print(encode(y_test))
res = np.array(res)
print(np.sum(res == encode(y_test)) / len(res))
```

```
0.7314211212516297
```

In [42]:
```
from sklearn.ensemble import RandomForestClassifier
```

In [633]:
```
cf = RandomForestClassifier(n_estimators=100, n_jobs=-1)
cf.fit(pt, encode(y_train))

res = cf.predict(test)
#print(res)
#print(encode(y_test))
res = np.array(res)
print(np.sum(res == encode(y_test)) / len(res))
```

```
0.8448500651890483
```

МЕТОПТЫ!

In [20]:
```
def one_hot(y, n_classes):
    # делаем вектор из 10 координат с 0 везде кроме правильного ответа
    tmp = np.zeros(
        (len(y), n_classes),
        dtype=np.uint8
    )
    tmp[range(len(tmp)), y] = 1
    return tmp
```

In [21]:
```
def softmax(W, b, x):
    tmp = np.exp(np.dot(x, W.T) + b)
    return (tmp.T / tmp.sum(axis=1)).T

def loss (y, pred):
    return -np.sum(y * np.log(pred), axis=1)
```

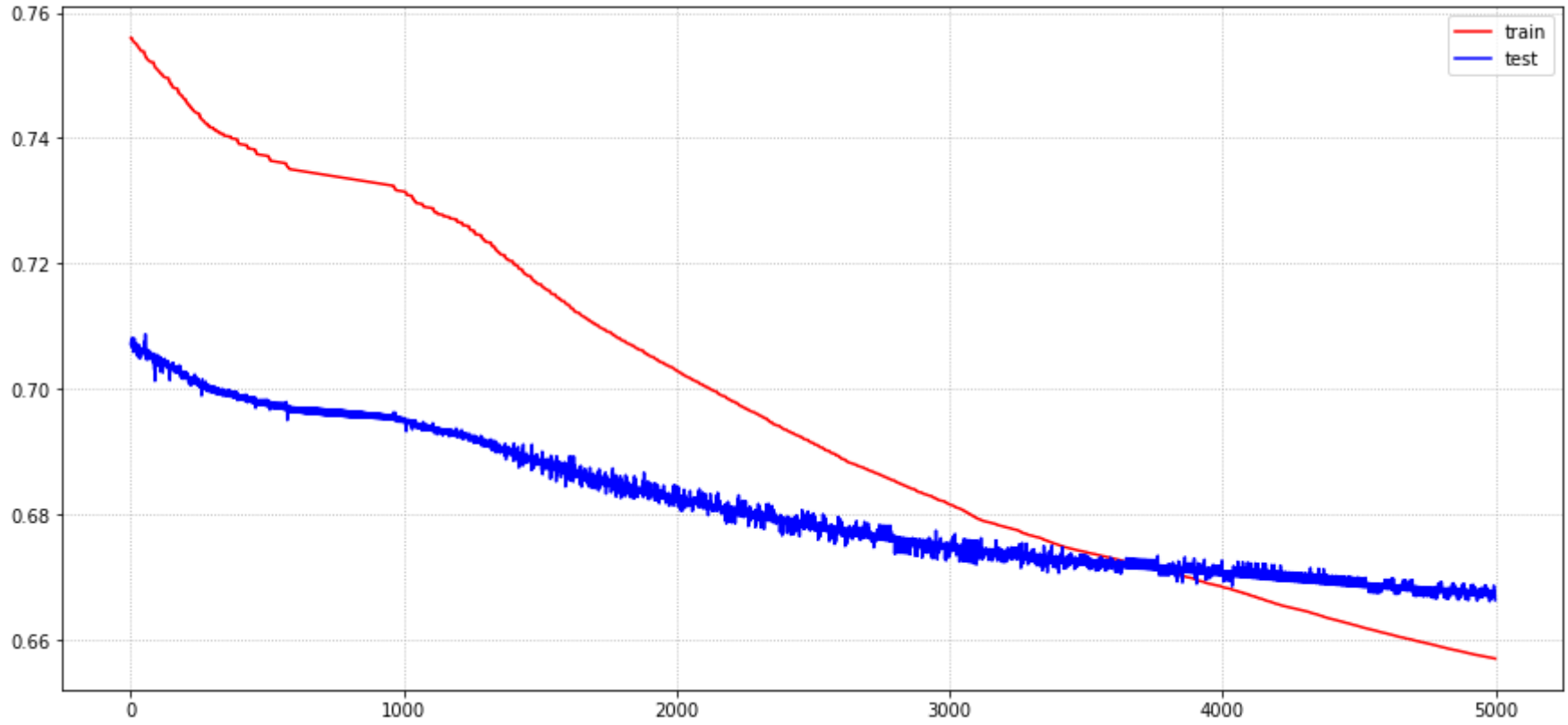
In [22]:
```
def compute_gradients(out, y, x):
    return np.hstack((np.array([np.sum(y) * out - y]).T @ np.array([x]), np.array([np.sum(y) * out - y]).T))

def gradients(W,b, x,y):
    sm = softmax(W,b,x)
    e = [ compute_gradients(a,c,b) for a,c,b in zip(sm,y,x) ]
    return np.mean(e,axis=0).T.flatten()
```

In [23]:
```
def armijo (W, b, x, y, dW, db, alpha=0.5, beta=0.5):
    dx = -gradients(W, b, x, y)
    s = 0.01
    n = len(W)
    m = len(W[0])

    while (np.mean(loss(y_tr, softmax(W + s * dW, b + s * db, X_train))) >
           np.mean(loss(y_tr, softmax(W, b, X_train))) - alpha * s * np.sum(dx ** 2)):
        s *= beta

    return s
```

In [622]:
```
W = np.zeros((6, 37))
b = np.zeros(6)

losses_train=[]
losses_valid=[]

y_tr = one_hot(encode(y_train), 6)
y_te = one_hot(encode(y_test), 6)

for i in tqdm(range(5000)):
    grad = gradients(W, b, pt, y_tr)
    dW = -grad[:37*6].reshape((37, 6)).T
    db = -grad[37*6:]
    s = armijo(W, b, pt, y_tr, dW, db)

    W += s * dW
    b += s * db

    losses_train.append(loss(y_tr, softmax(W, b, pt)))
    losses_valid.append(loss(y_te, softmax(W, b, test)))
```

```
100%|██████████| 5000/5000 [19:37<00:00,  4.25it/s]
```

In [623]:
```
plt.figure(figsize=(15, 7))
plt.plot(np.arange(5000), np.mean(losses_train, axis=1), color='red', label='train')
plt.plot(np.arange(5000), np.mean(losses_valid, axis=1), color='blue', label='test')
plt.grid(ls=':')
plt.legend()
plt.show()
```



In [625]:

```python
res = np.argmax(softmax(W, b, test), axis=1)
print(res)
print(encode(y_test))
res = np.array(res)
print(np.sum(res == encode(y_test)) / len(res))
```

```
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 0 0 1 1 1 1 0 0
 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0
 0 0 0 0 0 0 5 5 5 5 5 5 5 5 5 5 5 4 4 4 4 4 4 4 4 4 4 4 4 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 0 0 0 0 1 1 1 1 1 0 0 1 1 1 1 1 1 1 0 0 1 0
 0 0 0 0 1 5 5 5 5 5 5 5 5 5 5 4 4 4 4 4 4 4 4 4 4 4 4]
[2, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 3, 3, 3, 2, 2, 2, 3, 3, 3, 2, 2, 2, 3, 3, 3, 2, 2, 2, 3, 3, 3, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 2, 2, 2, 3, 3, 2, 2,
2, 2, 3, 3, 2, 2, 2, 2, 3, 3, 2, 2, 2, 2, 3, 3, 3, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4,
```

```
4, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 2, 2, 2, 3, 3, 3, 2, 2, 2, 2, 2, 2, 3, 3, 3, 2, 2, 2, 2, 2, 3, 3, 3, 2, 2, 2,
2, 3, 3, 3, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4]
0.833116036505867
```

In [642]:
```
file = open('sample', 'w')

for i in range(len(pt)):
    file.write(str(pt[i]) + ' ' + str(y_train[i]))
    file.write('\n')

for i in range(len(test)):
    file.write(str(test[i]) + ' ' + str(y_test[i]))
    file.write('\n')
```

In [194]:
```
file = open('output.txt', 'r')

X = []
Y = []

for line in file:
    arr = line.split(' ')
    X.append(np.array(arr[:-1], dtype=np.float))
    Y.append(arr[-1][:-1])

X = np.array(X)
Y = np.array(Y)

for i in range(37):
    X[:, i] -= np.mean(X[:, i])
    X[:, i] /= np.std(X[:, i])
```

In [195]:
```
import sklearn.model_selection
```

In [196]:
```
X_train, X_test, Y_train, Y_test = sklearn.model_selection.train_test_split(X, Y, test_size=0.2)
```

In [250]:
```
knn = KNN(n_neighbors=1, n_jobs=-1)
knn.fit(X_train, encode(Y_train))

res = knn.predict(X_test)
res = np.array(res)
info(res, Y_test, 'KNN')
```

```
KNN :
Суммарное качество = 97.60765550239235 %
Качество для  Walking = 98.82352941176471 %
Качество для  Jogging = 99.40119760479041 %
Качество для  Upstairs = 93.80530973451327 %
Качество для  Downstairs = 87.34177215189874 %
Качество для  Sitting = 98.0392156862745 %
Качество для  Standing = 100.0 %
```

In [252]:
```
cf = RandomForestClassifier(n_estimators=100, n_jobs=-1)
cf.fit(X_train, encode(Y_train))

res = cf.predict(X_test)
res = np.array(res)
info(res, Y_test, 'Random Forest')
```

```
Random Forest :
Суммарное качество = 96.7464114832536 %
Качество для  Walking = 99.29411764705883 %
Качество для  Jogging = 99.7005988023952 %
Качество для  Upstairs = 88.49557522123894 %
Качество для  Downstairs = 81.0126582278481 %
Качество для  Sitting = 96.07843137254902 %
Качество для  Standing = 100.0 %
```

In [199]:
```
import warnings
warnings.filterwarnings("ignore")
from sklearn.model_selection import train_test_split, GridSearchCV
```

In [217]:
```
#grid = {'max_depth':(range(15, 16)), 'criterion':('gini', 'entropy')}
grid = {'n_neighbors':(range(1, 15))}
cls = GridSearchCV(knn, grid, cv=5, scoring='accuracy', n_jobs=-1)
cls.fit(X_train, encode(Y_train))
```

Out[217]:
```
GridSearchCV(cv=5, error_score='raise',
       estimator=KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=-1, n_neighbors=5, p=2,
           weights='uniform'),
       fit_params=None, iid=True, n_jobs=-1,
       param_grid={'n_neighbors': range(1, 15)}, pre_dispatch='2*n_jobs',
       refit=True, return_train_score='warn', scoring='accuracy',
       verbose=0)
```

In [218]:
```
print(np.sum(cls.predict(X_test) == encode(Y_test)) / len(X_test))
```

```
0.9760765550239234
```

In [219]:
```
cls.cv_results_
```

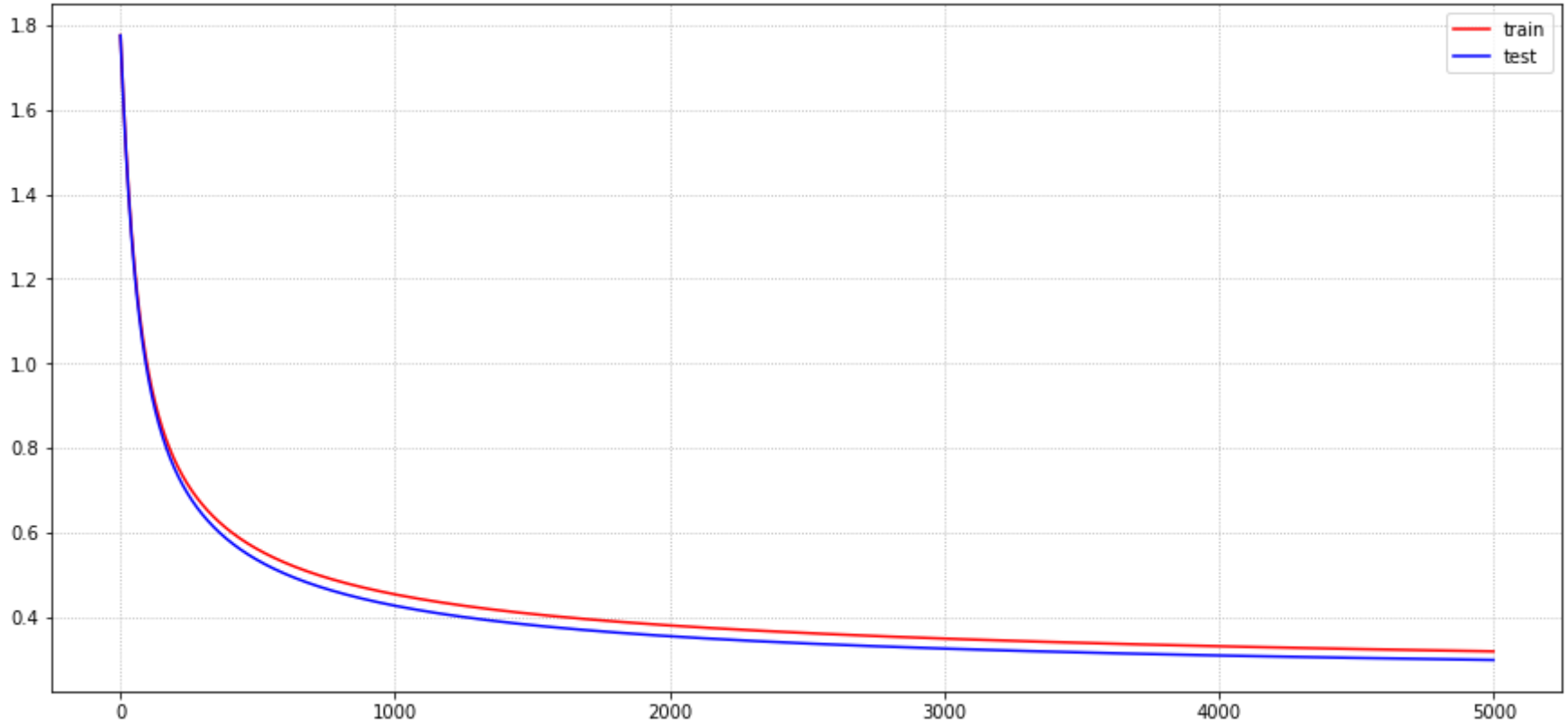
Out[219]:
```
{'mean_fit_time': array([0.00772004, 0.01142941, 0.00761161, 0.00911994, 0.00844159,
       0.01058393, 0.01141152, 0.01100278, 0.01283875, 0.00735903,
       0.00696735, 0.00706344, 0.00780215, 0.00850635]),
 'mean_score_time': array([0.21159182, 0.30665607, 0.39147954, 0.44733534, 0.37007589,
       0.34863667, 0.34052587, 0.31108074, 0.37422357, 0.41103759,
       0.30762467, 0.39355097, 0.43561902, 0.39311376]),
 'mean_test_score': array([0.97607656, 0.96363636, 0.97129187, 0.96291866, 0.96196172,
       0.95909091, 0.95478469, 0.95239234, 0.95239234, 0.95239234,
       0.95095694, 0.94952153, 0.94832536, 0.94665072]),
 'mean_train_score': array([1.        , 0.98295411, 0.98690159, 0.97846857, 0.97996343,
       0.972309  , 0.97200881, 0.96746386, 0.9659686 , 0.96237951,
       0.96112325, 0.95873072, 0.95789379, 0.95532167]),
 'param_n_neighbors': masked_array(data=[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14],
              mask=[False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False],
       fill_value='?',
            dtype=object),
 'params': [{'n_neighbors': 1},
  {'n_neighbors': 2},
  {'n_neighbors': 3},
  {'n_neighbors': 4},
  {'n_neighbors': 5},
  {'n_neighbors': 6},
  {'n_neighbors': 7},
  {'n_neighbors': 8},
  {'n_neighbors': 9},
  {'n_neighbors': 10},
  {'n_neighbors': 11},
  {'n_neighbors': 12},
  {'n_neighbors': 13},
  {'n_neighbors': 14}],
 'rank_test_score': array([ 1,  3,  2,  4,  5,  6,  7,  8,  8,  8, 11, 12, 13, 14],
      dtype=int32),
 'split0_test_score': array([0.97494033, 0.96658711, 0.97136038, 0.97016706, 0.96062053,
       0.96300716, 0.96300716, 0.95584726, 0.95346062, 0.95465394,
       0.95107399, 0.94988067, 0.95107399, 0.9522673 ]),
 'split0_train_score': array([1.        , 0.98294434, 0.98653501, 0.97905446, 0.97755835,
       0.97336924, 0.9700778 , 0.96588869, 0.96229803, 0.95990425,
       0.95751047, 0.95541592, 0.95481747, 0.95302214]),
```

```
'split1_test_score': array([0.97613365, 0.96539379, 0.97374702, 0.96778043, 0.9725537 ,
       0.96658711, 0.96062053, 0.96062053, 0.96658711, 0.96539379,
       0.96181384, 0.96062053, 0.95346062, 0.9522673 ]),
'split1_train_score': array([1.        , 0.98084979, 0.98533812, 0.97546379, 0.97845601,
       0.97187313, 0.9700778 , 0.96768402, 0.9673848 , 0.96080192,
       0.9593058 , 0.95571514, 0.95571514, 0.95062837]),
'split2_test_score': array([0.97488038, 0.9569378 , 0.97009569, 0.95813397, 0.9569378 ,
       0.95095694, 0.94497608, 0.9437799 , 0.94138756, 0.94497608,
       0.9437799 , 0.94258373, 0.94258373, 0.94019139]),
'split2_train_score': array([1.        , 0.98355263, 0.98803828, 0.9805622 , 0.98205742,
       0.9742823 , 0.97338517, 0.96800239, 0.96800239, 0.9632177 ,
       0.96232057, 0.96022727, 0.96052632, 0.95843301]),
'split3_test_score': array([0.97482014, 0.96642686, 0.97122302, 0.96043165, 0.95683453,
       0.94844125, 0.94724221, 0.94604317, 0.94604317, 0.94484412,
       0.94844125, 0.94484412, 0.94364508, 0.93884892]),
'split3_train_score': array([1.        , 0.98445906, 0.98774656, 0.9790795 , 0.97967723,
       0.97011357, 0.97280335, 0.96622833, 0.96712493, 0.96503288,
       0.96353855, 0.9614465 , 0.95905559, 0.95666467]),
'split4_test_score': array([0.97961631, 0.96282974, 0.97002398, 0.95803357, 0.96282974,
       0.96642686, 0.95803357, 0.95563549, 0.95443645, 0.95203837,
       0.94964029, 0.94964029, 0.95083933, 0.94964029]),
'split4_train_score': array([1.        , 0.98296473, 0.98684997, 0.9781829 , 0.98206814,
       0.97190675, 0.97369994, 0.96951584, 0.96503288, 0.96294082,
       0.96294082, 0.96084877, 0.95935445, 0.95786013]),
'std_fit_time': array([0.00115475, 0.00630899, 0.0024064 , 0.00297003, 0.00304962,
       0.00441387, 0.00786017, 0.00668601, 0.00622472, 0.00200637,
       0.00078376, 0.00153124, 0.00114131, 0.00441269]),
'std_score_time': array([0.05436279, 0.07401184, 0.06779678, 0.03383113, 0.1481679 ,
       0.01654463, 0.02167329, 0.06306111, 0.0809592 , 0.04434047,
       0.04952764, 0.06055202, 0.06720453, 0.06992444]),
'std_test_score': array([0.00183319, 0.00360865, 0.00134805, 0.00508388, 0.00577024,
       0.00780633, 0.00728562, 0.00639863, 0.00858611, 0.00757013,
       0.00596189, 0.00622229, 0.0043619 , 0.00590939]),
'std_train_score': array([0.        , 0.00118756, 0.00095786, 0.00168583, 0.00184137,
       0.00142757, 0.0016027 , 0.00130819, 0.00208939, 0.00182613,
       0.00232061, 0.00261469, 0.00221918, 0.00300808])}
```

In [220]:
```
W = np.zeros((6, 37))
b = np.zeros(6)

losses_train=[]
losses_valid=[]

y_tr = one_hot(encode(Y_train), 6)
y_te = one_hot(encode(Y_test), 6)

for i in tqdm(range(5000)):
    grad = gradients(W, b, X_train, y_tr)
    dW = -grad[:37*6].reshape((37, 6)).T
    db = -grad[37*6:]
    s = armijo(W, b, X_train, y_tr, dW, db)

    W += s * dW
    b += s * db

    losses_train.append(loss(y_tr, softmax(W, b, X_train)))
    losses_valid.append(loss(y_te, softmax(W, b, X_test)))
```

```
100%|██████████| 5000/5000 [17:11<00:00,  4.85it/s]
```

In [221]:
```
plt.figure(figsize=(15, 7))
plt.plot(np.arange(5000), np.mean(losses_train, axis=1), color='red', label='train')
plt.plot(np.arange(5000), np.mean(losses_valid, axis=1), color='blue', label='test')
plt.grid(ls=':')
plt.legend()
plt.show()
```



In [249]:
```
res = np.argmax(softmax(W, b, X_test), axis=1)
#print(res[:200])
#print(encode(Y_test)[:200])
res = np.array(res)
info(res, Y_test, 'Log Regression')
```

```
Log Regression :
Суммарное качество = 91.10047846889952 %
Качество для  Walking = 96.0 %
Качество для  Jogging = 99.40119760479041 %
Качество для  Upstairs = 78.76106194690266 %
Качество для  Downstairs = 43.037974683544306 %
Качество для  Sitting = 92.15686274509804 %
Качество для  Standing = 97.67441860465115 %
```