Task 1

In [30]:

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
import scipy.stats as scp
import math
from sklearn import cross_validation, datasets, metrics, neighbors


%matplotlib inline
%pylab inline

SIZE = 1000
```

```
Populating the interactive namespace from numpy and matplotlib
```

```
/usr/local/lib/python2.7/dist-packages/sklearn/cross_validation.py:44:
 DeprecationWarning: This module was deprecated in version 0.18 in fav
or of the model_selection module into which all the refactored classes
 and functions are moved. Also note that the interface of the new CV i
terators are different from that of this module. This module will be r
emoved in 0.20.
  "This module will be removed in 0.20.", DeprecationWarning)
```

Генерация данных:

In [94]:

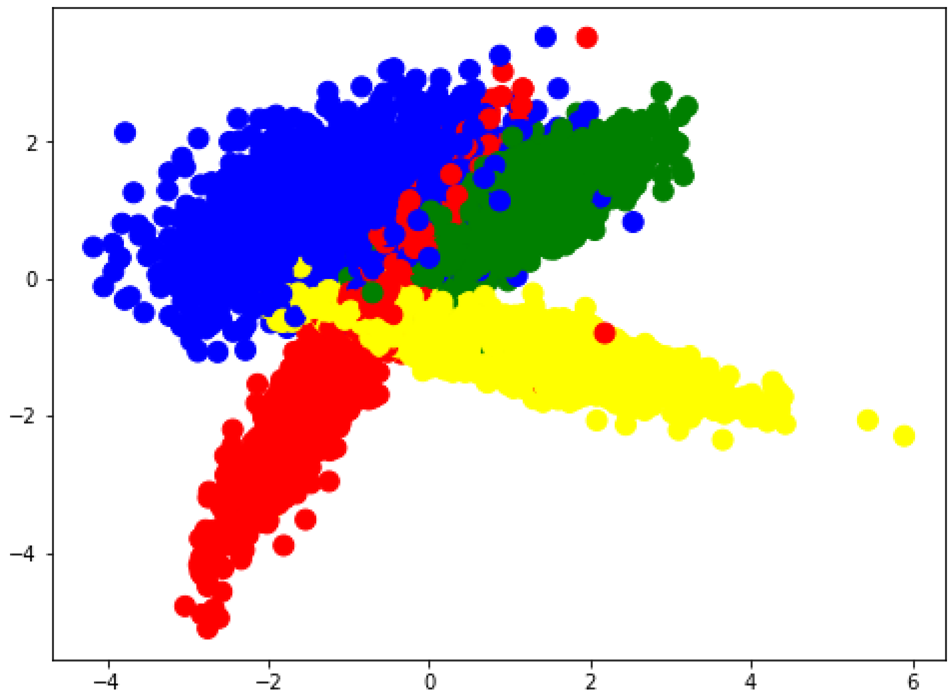
```
classification_problem = datasets.make_classification(n_samples=10000,
                                                       n_features=2,
                                                       n_informative=2,
                                                       n_classes=4,
                                                       n_redundant=0,
                                                       n_clusters_per_class=1,
                                                       random_state=3)
colors = ListedColormap(['red', 'blue', 'yellow', 'green'])
light_colors = ListedColormap(
    ['lightcoral', 'lightblue', 'lightyellow', 'lightgreen'])
```

In [95]:

```
pylab.figure(figsize=(8,6))
pylab.scatter(map(lambda x: x[0], classification_problem[0]),
              map(lambda x: x[1], classification_problem[0]),
              c=classification_problem[1], cmap=colors, s=100)
```

Out[95]:

```
<matplotlib.collections.PathCollection at 0x7f5c6b1181d0>
```



In [96]:

```
train_data, test_data, train_labels, test_labels
= cross_validation.train_test_split(classification_problem[0],

                                    classification_problem[1],

                                    test_size = 0.3,

                                    random_state = 1)
```

Визуализируем разделяющие поверхности:

In [38]:

```
def get_meshgrid(data, step=.05, border=.5,):
    x_min, x_max = data[:, 0].min() - border, data[:, 0].max() + border
    y_min, y_max = data[:, 1].min() - border, data[:, 1].max() + border
    return np.meshgrid(np.arange(x_min, x_max, step),
                       np.arange(y_min, y_max, step))
```

In [42]:

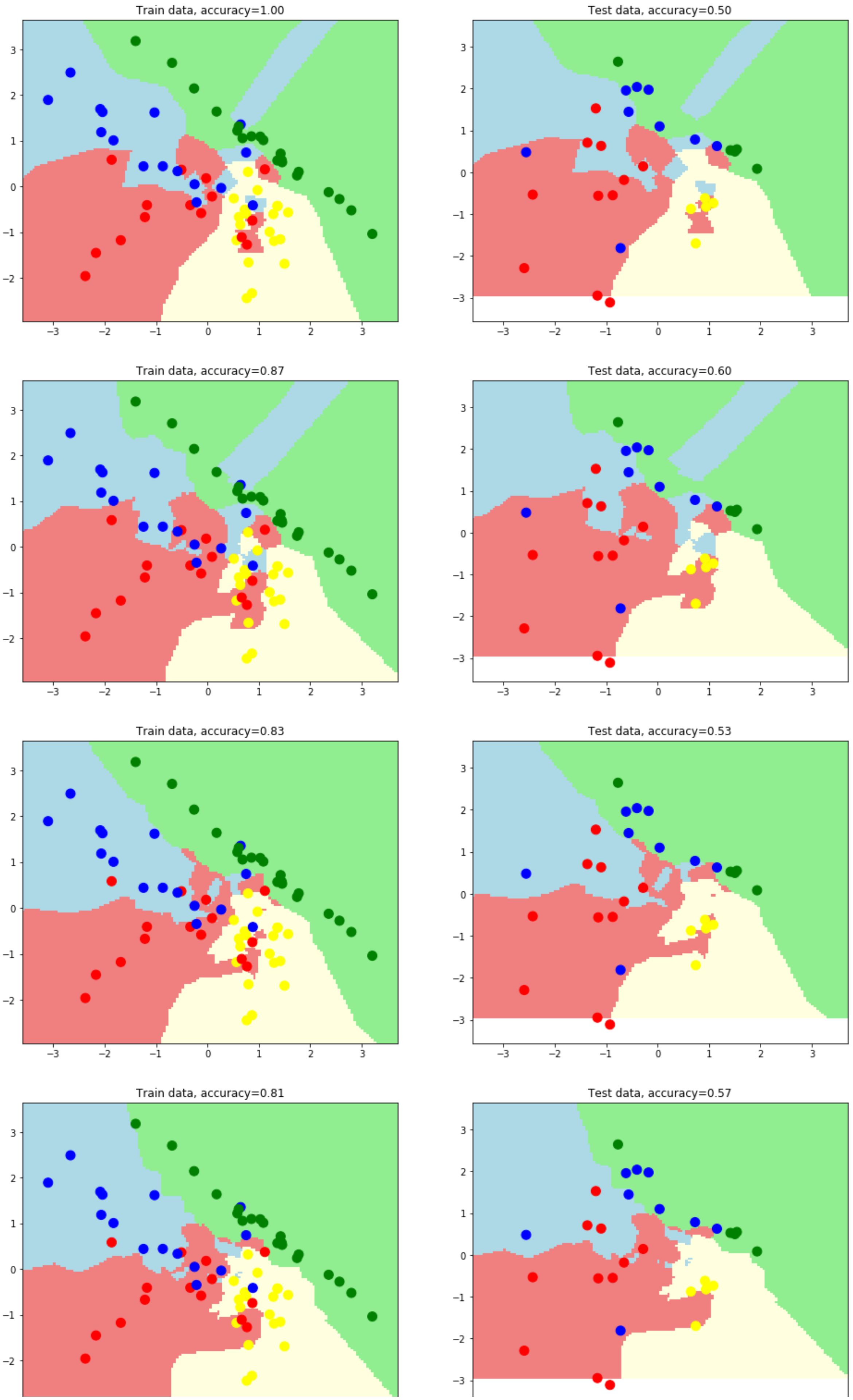
```
def plot_decision_surface(estimator, train_data, train_labels,
                          test_data, test_labels,
                          colors = colors, light_colors = light_colors):
    #fit model
    estimator.fit(train_data, train_labels)

    #set figure size
    pyplot.figure(figsize = (16, 6))

    #plot decision surface on the train data
    pyplot.subplot(1,2,1)
    xx, yy = get_meshgrid(train_data)
    mesh_predictions = np.array(
        estimator.predict(np.c_[xx.ravel(), yy.ravel()])).reshape(xx.shape)
    pyplot.pcolormesh(
        xx, yy, mesh_predictions, cmap = light_colors)
    pyplot.scatter(
        train_data[:, 0], train_data[:, 1], c = train_labels, s = 100, cmap = color
    pyplot.title(
        'Train data, accuracy={:.2f}'
                 .format(metrics.accuracy_score(train_labels,
                                                estimator.predict(train_data))))

    #plot decision surface on the test data
    pyplot.subplot(1,2,2)
    pyplot.pcolormesh(xx, yy, mesh_predictions, cmap = light_colors)
    pyplot.scatter(test_data[:, 0], test_data[:, 1],
                   c = test_labels, s = 100, cmap = colors)
    pyplot.title('Test data, accuracy={:.2f}'
                 .format(metrics.accuracy_score(test_labels,
                                                estimator.predict(test_data))))
```

In [45]:

```
def plot_surface_with_k_neighbours(k):
    estimator = neighbors.KNeighborsClassifier(n_neighbors=k)
    plot_decision_surface(
        estimator, train_data, train_labels, test_data, test_labels)
```

In [52]:

```
for i in range(1,5):
    plot_surface_with_k_neighbours(i)
```



Теперь построим график accuracy от количества соседей:

In [97]:

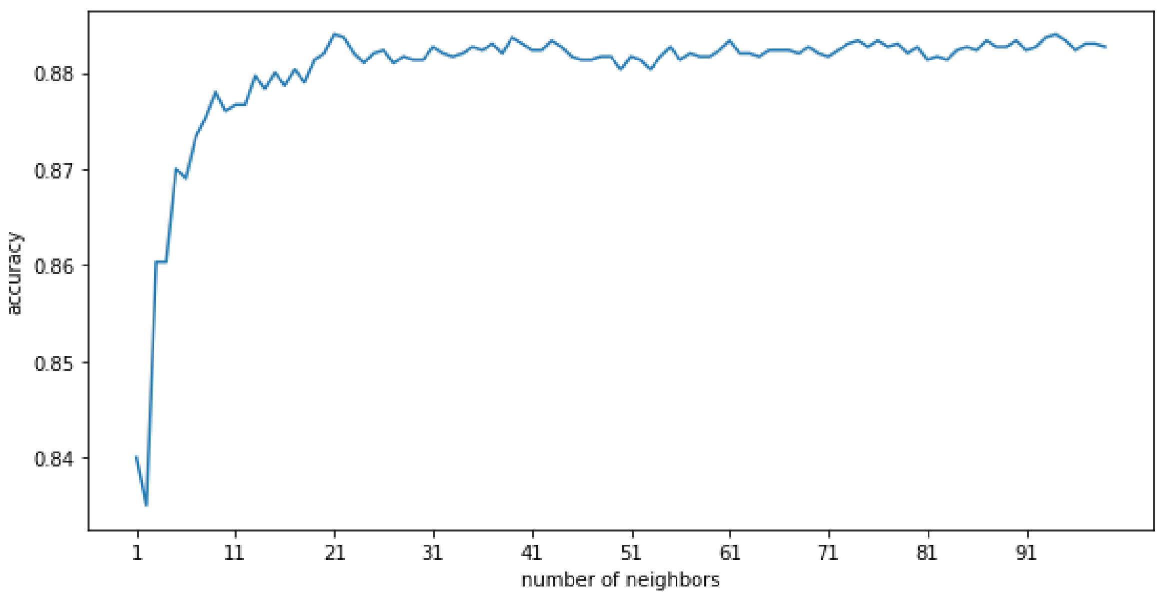
```
def get_accuracy(estimator, train_data, train_labels, test_data, test_labels,
                          colors = colors, light_colors = light_colors):

    estimator.fit(train_data, train_labels)
    return [metrics.accuracy_score(train_labels, estimator.predict(train_data)),
        metrics.accuracy_score(test_labels, estimator.predict(test_data))]

def get_accuracy_for_k_neighbours(k):
    estimator = neighbors.KNeighborsClassifier(n_neighbors=k)
    return get_accuracy(estimator, train_data,
                        train_labels, test_data, test_labels)

def get_accuracy_array():
    result = []
    for i in range(1,100):
        result.append(get_accuracy_for_k_neighbours(i))
    return np.array(result)
```

In [98]:

```
accuracy = (get_accuracy_array())[:,1]
```

In [100]:

```
fig = plt.figure(figsize=[10, 5])
plt.plot(np.arange(1, 100), accuracy)
plt.xlabel('number of neighbors')
plt.ylabel('accuracy')
plt.xticks(np.arange(1,100)[::10])
plt.show()
```



Как мы видим, наилучшая accuracy получается при $n = 20$

Task 2

In [84]:

```
from sklearn import datasets
from sklearn.model_selection import cross_val_score
from sklearn import naive_bayes
```

In [67]:

```
digits = datasets.load_digits()
```

In [68]:

```
digits
```

Out[68]:

```
{'DESCR': "Optical Recognition of Handwritten Digits Data Set\n=======
============================================\n\nNotes\n-----\nData Set
 Characteristics:\n    :Number of Instances: 5620\n    :Number of Attr
ibutes: 64\n    :Attribute Information: 8x8 image of integer pixels in
 the range 0..16.\n    :Missing Attribute Values: None\n    :Creator:
 E. Alpaydin (alpaydin '@' boun.edu.tr)\n    :Date: July; 1998\n\nThis
 is a copy of the test set of the UCI ML hand-written digits datasets
\nhttp://archive.ics.uci.edu/ml/datasets/Optical+Recognition+of+Handwr
itten+Digits\n\nThe data set contains images of hand-written digits: 1
0 classes where\neach class refers to a digit.\n\nPreprocessing progra
ms made available by NIST were used to extract\nnormalized bitmaps of
 handwritten digits from a preprinted form. From a\ntotal of 43 peopl
e, 30 contributed to the training set and different 13\nto the test se
t. 32x32 bitmaps are divided into nonoverlapping blocks of\n4x4 and th
e number of on pixels are counted in each block. This generates\nan in
put matrix of 8x8 where each element is an integer in the range\n0..1
6. This reduces dimensionality and gives invariance to small\ndistorti
ons.\n\nFor info on NIST preprocessing routines, see M. D. Garris, J.
 L. Blue, G.\nT. Candela, D. L. Dimmick, J. Geist, P. J. Grother, S.
 A. Janet, and C.\nL. Wilson, NIST Form-Based Handprint Recognition Sy
stem, NISTIR 5469,\n1994.\n\nReferences\n----------\n  - C. Kaynak (19
95) Methods of Combining Multiple Classifiers and Their\n    Applicati
ons to Handwritten Digit Recognition, MSc Thesis, Institute of\n    Gr
aduate Studies in Science and Engineering, Bogazici University.\n  -
 E. Alpaydin, C. Kaynak (1998) Cascading Classifiers, Kybernetika.\n
 - Ken Tang and Ponnuthurai N. Suganthan and Xi Yao and A. Kai Qin.\n
   Linear dimensionalityreduction using relevance weighted LDA. School
 of\n    Electrical and Electronic Engineering Nanyang Technological U
niversity.\n    2005.\n  - Claudio Gentile. A New Approximate Maximal
 Margin Classification\n    Algorithm. NIPS. 2000.\n",
 'data': array([[  0.,   0.,   5., ...,   0.,   0.,   0.],
       [  0.,   0.,   0., ...,  10.,   0.,   0.],
       [  0.,   0.,   0., ...,  16.,   9.,   0.],
       ...,
       [  0.,   0.,   1., ...,   6.,   0.,   0.],
       [  0.,   0.,   2., ...,  12.,   0.,   0.],
       [  0.,   0.,  10., ...,  12.,   1.,   0.]]),
 'images': array([[[  0.,   0.,   5., ...,   1.,   0.,   0.],
        [  0.,   0.,  13., ...,  15.,   5.,   0.],
        [  0.,   3.,  15., ...,  11.,   8.,   0.],
        ...,
        [  0.,   4.,  11., ...,  12.,   7.,   0.],
        [  0.,   2.,  14., ...,  12.,   0.,   0.],
        [  0.,   0.,   6., ...,   0.,   0.,   0.]],

       [[  0.,   0.,   0., ...,   5.,   0.,   0.],
        [  0.,   0.,   0., ...,   9.,   0.,   0.],
        [  0.,   0.,   3., ...,   6.,   0.,   0.],
        ...,
        [  0.,   0.,   1., ...,   6.,   0.,   0.],
        [  0.,   0.,   1., ...,   6.,   0.,   0.],
        [  0.,   0.,   0., ...,  10.,   0.,   0.]],

       [[  0.,   0.,   0., ...,  12.,   0.,   0.],
        [  0.,   0.,   3., ...,  14.,   0.,   0.],
        [  0.,   0.,   8., ...,  16.,   0.,   0.],
        ...,
        [  0.,   9.,  16., ...,   0.,   0.,   0.],
        [  0.,   3.,  13., ...,  11.,   5.,   0.],
        [  0.,   0.,   0., ...,  16.,   9.,   0.]],
```

```
        ...,
        [[  0.,   0.,   1., ...,   1.,   0.,   0.],
         [  0.,   0.,  13., ...,   2.,   1.,   0.],
         [  0.,   0.,  16., ...,  16.,   5.,   0.],
         ...,
         [  0.,   0.,  16., ...,  15.,   0.,   0.],
         [  0.,   0.,  15., ...,  16.,   0.,   0.],
         [  0.,   0.,   2., ...,   6.,   0.,   0.]],

        [[  0.,   0.,   2., ...,   0.,   0.,   0.],
         [  0.,   0.,  14., ...,  15.,   1.,   0.],
         [  0.,   4.,  16., ...,  16.,   7.,   0.],
         ...,
         [  0.,   0.,   0., ...,  16.,   2.,   0.],
         [  0.,   0.,   4., ...,  16.,   2.,   0.],
         [  0.,   0.,   5., ...,  12.,   0.,   0.]],

        [[  0.,   0.,  10., ...,   1.,   0.,   0.],
         [  0.,   2.,  16., ...,   1.,   0.,   0.],
         [  0.,   0.,  15., ...,  15.,   0.,   0.],
         ...,
         [  0.,   4.,  16., ...,  16.,   6.,   0.],
         [  0.,   8.,  16., ...,  16.,   8.,   0.],
         [  0.,   1.,   8., ...,  12.,   1.,   0.]]]),
 'target': array([0, 1, 2, ..., 8, 9, 8]),
 'target_names': array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])}
```

In [69]:

```
breast = datasets.load_breast_cancer()
```

In [73]:

```
breast
```

Out[73]:

```
{'DESCR': 'Breast Cancer Wisconsin (Diagnostic) Database\n===========
==================================\n\nNotes\n-----\nData Set Characteri
stics:\n    :Number of Instances: 569\n\n    :Number of Attributes: 30
 numeric, predictive attributes and the class\n\n    :Attribute Inform
ation:\n        - radius (mean of distances from center to points on t
he perimeter)\n        - texture (standard deviation of gray-scale val
ues)\n        - perimeter\n        - area\n        - smoothness (local
 variation in radius lengths)\n        - compactness (perimeter^2 / ar
ea - 1.0)\n        - concavity (severity of concave portions of the co
ntour)\n        - concave points (number of concave portions of the co
ntour)\n        - symmetry \n        - fractal dimension ("coastline a
pproximation" - 1)\n\n        The mean, standard error, and "worst" or
 largest (mean of the three\n        largest values) of these features
 were computed for each image,\n        resulting in 30 features.  For
 instance, field 3 is Mean Radius, field\n        13 is Radius SE, fie
ld 23 is Worst Radius.\n\n        - class:\n                - WDBC-Mal
ignant\n                - WDBC-Benign\n\n    :Summary Statistics:\n\n
   ===================================== ====== ======\n
                             Min    Max\n    ========================
============ ====== ======\n    radius (mean):
 6.981  28.11\n    texture (mean):                       9.71   39.28
\n    perimeter (mean):                     43.79  188.5\n    area (me
an):                          143.5  2501.0\n    smoothness (mean):
                 0.053  0.163\n    compactness (mean):
   0.019  0.345\n    concavity (mean):                     0.0    0.42
7\n    concave points (mean):                0.0    0.201\n    symmetr
y (mean):                         0.106  0.304\n    fractal dimension (me
an):              0.05   0.097\n    radius (standard error):
   0.112  2.873\n    texture (standard error):             0.36   4.88
5\n    perimeter (standard error):           0.757  21.98\n    area (s
tandard error):                   6.802  542.2\n    smoothness (standard
 error):           0.002  0.031\n    compactness (standard error):
    0.002  0.135\n    concavity (standard error):           0.0    0.3
96\n    concave points (standard error):      0.0    0.053\n    symmet
ry (standard error):            0.008  0.079\n    fractal dimension (s
tandard error):   0.001  0.03\n    radius (worst):
   7.93   36.04\n    texture (worst):                      12.02  49.5
4\n    perimeter (worst):                    50.41  251.2\n    area (w
orst):                          185.2  4254.0\n    smoothness (worst):
                  0.071  0.223\n    compactness (worst):
    0.027  1.058\n    concavity (worst):                    0.0    1.2
52\n    concave points (worst):               0.0    0.291\n    symmet
ry (worst):                     0.156  0.664\n    fractal dimension (w
orst):            0.055  0.208\n    ==================================
=== ====== ======\n\n    :Missing Attribute Values: None\n\n    :Class
 Distribution: 212 - Malignant, 357 - Benign\n\n    :Creator:  Dr. Wil
liam H. Wolberg, W. Nick Street, Olvi L. Mangasarian\n\n    :Donor: Ni
ck Street\n\n    :Date: November, 1995\n\nThis is a copy of UCI ML Bre
ast Cancer Wisconsin (Diagnostic) datasets.\nhttps://goo.gl/U2Uwz2\n\n
Features are computed from a digitized image of a fine needle\naspirat
e (FNA) of a breast mass.  They describe\ncharacteristics of the cell
 nuclei present in the image.\n\nSeparating plane described above was
 obtained using\nMultisurface Method-Tree (MSM-T) [K. P. Bennett, "Dec
ision Tree\nConstruction Via Linear Programming." Proceedings of the 4
th\nMidwest Artificial Intelligence and Cognitive Science Society,\np
p. 97-101, 1992], a classification method which uses linear\nprogrammi
ng to construct a decision tree.  Relevant features\nwere selected usi
ng an exhaustive search in the space of 1-4\nfeatures and 1-3 separati
ng planes.\n\nThe actual linear program used to obtain the separating
 plane\nin the 3-dimensional space is that described in:\n[K. P. Benne
tt and O. L. Mangasarian: "Robust Linear\nProgramming Discrimination o
```

```
f Two Linearly Inseparable Sets",\nOptimization Methods and Software
 1, 1992, 23-34].\n\nThis database is also available through the UW CS
 ftp server:\n\nftp ftp.cs.wisc.edu\ncd math-prog/cpo-dataset/machine-
learn/WDBC/\n\nReferences\n----------\n   - W.N. Street, W.H. Wolberg
 and O.L. Mangasarian. Nuclear feature extraction \n     for breast tu
mor diagnosis. IS&T/SPIE 1993 International Symposium on \n     Electr
onic Imaging: Science and Technology, volume 1905, pages 861-870,\n
  San Jose, CA, 1993.\n   - O.L. Mangasarian, W.N. Street and W.H. Wol
berg. Breast cancer diagnosis and \n     prognosis via linear programm
ing. Operations Research, 43(4), pages 570-577, \n     July-August 199
5.\n   - W.H. Wolberg, W.N. Street, and O.L. Mangasarian. Machine lear
ning techniques\n     to diagnose breast cancer from fine-needle aspir
ates. Cancer Letters 77 (1994) \n     163-171.\n',
 'data': array([[  1.79900000e+01,   1.03800000e+01,   1.22800000e+02,
 ...,
          2.65400000e-01,   4.60100000e-01,   1.18900000e-01],
       [  2.05700000e+01,   1.77700000e+01,   1.32900000e+02, ...,
          1.86000000e-01,   2.75000000e-01,   8.90200000e-02],
       [  1.96900000e+01,   2.12500000e+01,   1.30000000e+02, ...,
          2.43000000e-01,   3.61300000e-01,   8.75800000e-02],
       ...,
       [  1.66000000e+01,   2.80800000e+01,   1.08300000e+02, ...,
          1.41800000e-01,   2.21800000e-01,   7.82000000e-02],
       [  2.06000000e+01,   2.93300000e+01,   1.40100000e+02, ...,
          2.65000000e-01,   4.08700000e-01,   1.24000000e-01],
       [  7.76000000e+00,   2.45400000e+01,   4.79200000e+01, ...,
          0.00000000e+00,   2.87100000e-01,   7.03900000e-02]]),
 'feature_names': array(['mean radius', 'mean texture', 'mean perimete
r', 'mean area',
       'mean smoothness', 'mean compactness', 'mean concavity',
       'mean concave points', 'mean symmetry', 'mean fractal dimensio
n',
       'radius error', 'texture error', 'perimeter error', 'area erro
r',
       'smoothness error', 'compactness error', 'concavity error',
       'concave points error', 'symmetry error', 'fractal dimension e
rror',
       'worst radius', 'worst texture', 'worst perimeter', 'worst are
a',
       'worst smoothness', 'worst compactness', 'worst concavity',
       'worst concave points', 'worst symmetry', 'worst fractal dimen
sion'],
      dtype='|S23'),
 'target': array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 1, 1, 1, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
 0, 0,
       1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 1,
 1, 1,
       1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1,
 1, 0,
       1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1,
 1, 1,
       1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0,
 1, 1,
       0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1,
 1, 1,
       0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 0,
 0, 1,
       0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1,
 0, 1,
```

```
        0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1,
 1, 0,
        0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0,
 1, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1,
 0, 1,
        1, 0, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
 0, 1,
        1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1,
 1, 0,
        1, 0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1,
 0, 1,
        1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0,
 0, 1,
        0, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1,
 0, 1,
        1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1,
 1, 1,
        0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1,
 0, 1,
        1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1,
 1, 1,
        0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1,
 1, 1,
        1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0,
 1, 1,
        1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1,
 1, 1,
        1, 1, 1, 1, 0, 1, 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, 0, 0, 0, 0, 0, 0, 1]),
 'target_names': array(['malignant', 'benign'],
       dtype='|S9')}
```

In [90]:

```python
def get_number_of_operations(method, d_set):
    return np.mean(cross_val_score(method, d_set.data, d_set.target))
```

In [92]:

```python
print('digits dataset:')
print('bernulli accuracy = {:.2f}'.format(
    get_number_of_operations(naive_bayes.BernoulliNB(), digits)))
print('multinomial accuracy = {:.2f}'.format(
    get_number_of_operations(naive_bayes.MultinomialNB(), digits)))
print('gaussian accuracy = {:.2f}'.format(
    get_number_of_operations(naive_bayes.GaussianNB(), digits)))
```

```
digits dataset:
bernulli accuracy = 0.83
multinomial accuracy = 0.87
gaussian accuracy = 0.82
```

In [93]:

```
print('breast_canser dataset:')
print('bernulli accuracy = {:.2f}'.format(
    get_number_of_operations(naive_bayes.BernoulliNB(), breast)))
print('multinomial accuracy = {:.2f}'.format(
    get_number_of_operations(naive_bayes.MultinomialNB(), breast)))
print('gaussian accuracy = {:.2f}'.format(
    get_number_of_operations(naive_bayes.GaussianNB(), breast)))
```

```
breast_canser dataset:
bernulli accuracy = 0.63
multinomial accuracy = 0.89
gaussian accuracy = 0.94
```

Итак, максимально качество на датасете digits: 0.87 на датасете breast_canser: 0.94

Из наших данных верны следующие пункты: (b)

Task 3

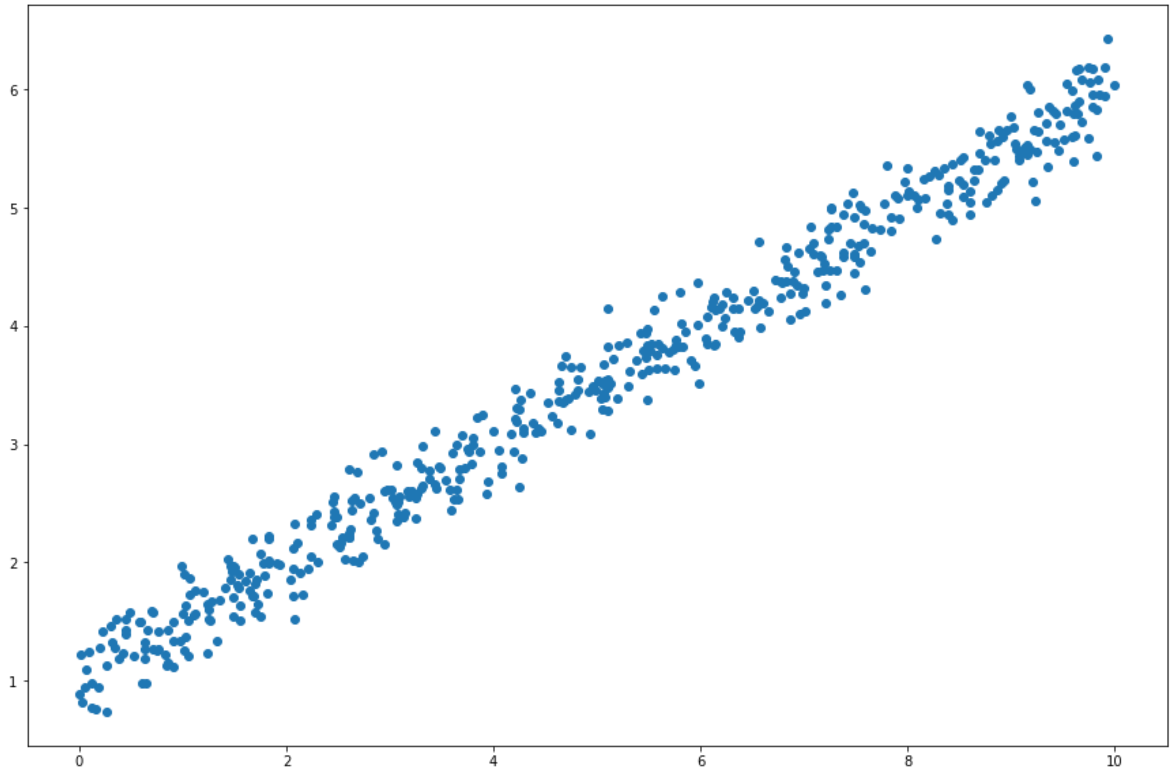
Сгенерируем выборку точек

In [107]:

```
A = 0
SCALE = 0.2
SIZE = 500

eps = np.array(scp.norm.rvs(loc=A, scale=SCALE, size=SIZE))
sampleX = np.array(scp.uniform.rvs(loc=0, scale=10, size=SIZE))
sampleY = sampleX * 0.5 + 1 + eps
```

In [108]:

```
fig = plt.figure(figsize=[15, 10])
plt.scatter(sampleX, sampleY)
plt.show()
```



Построим оценки

In [110]:

```
from scipy.optimize import minimize
```

In [129]:

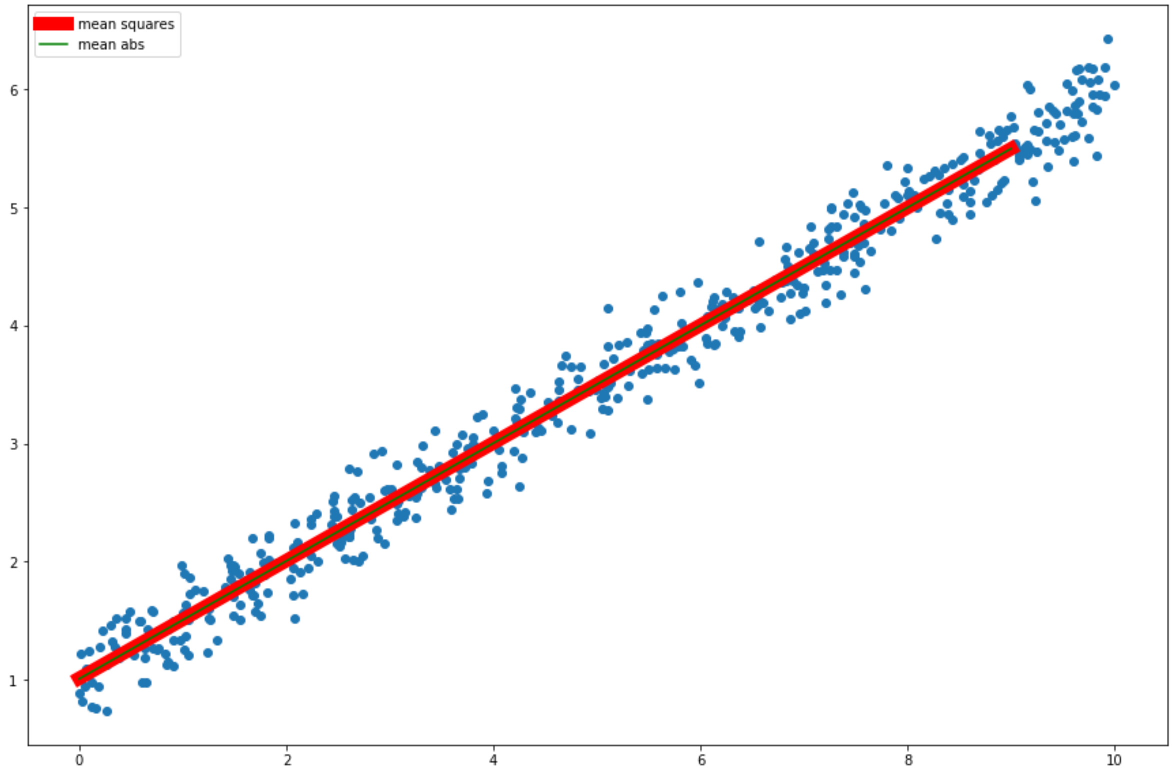
```
def func(args, x):
    k, b = args
    return k * x + b

def func_mean_sqaures(args):
    k, b = args
    return ((k * sampleX + b - sampleY) ** 2).mean()

def func_mean_abs(args):
    k, b = args
    return np.mean(np.fabs(k * sampleX + b - sampleY))
```

In [130]:

```
bnds = ((0, 10), (-10, 10))
res_mean_squares = minimize(func_min_sqaures, (2, 0), bounds=bnds)
res_mean_abs = minimize(func_min_abs, (2, 0), bounds=bnds)
```

In [135]:

```
fig = plt.figure(figsize=[15, 10])
plt.scatter(sampleX, sampleY)
plt.plot(np.arange(10), func(res_mean_squares.x, np.arange(10)),
         linewidth=10, color='red', label=r'mean squares')
plt.plot(np.arange(10), func(res_mean_abs.x, np.arange(10)),
         color='green', label=r'mean abs')
plt.legend()
plt.show()
```



Добавим выбросы

In [136]:

```
newEps = np.array(scp.norm.rvs(loc=A, scale=SCALE, size=75))
newX = np.array(scp.uniform.rvs(loc=0, scale=10, size=75))
newY = -1 + newEps
```

In [137]:

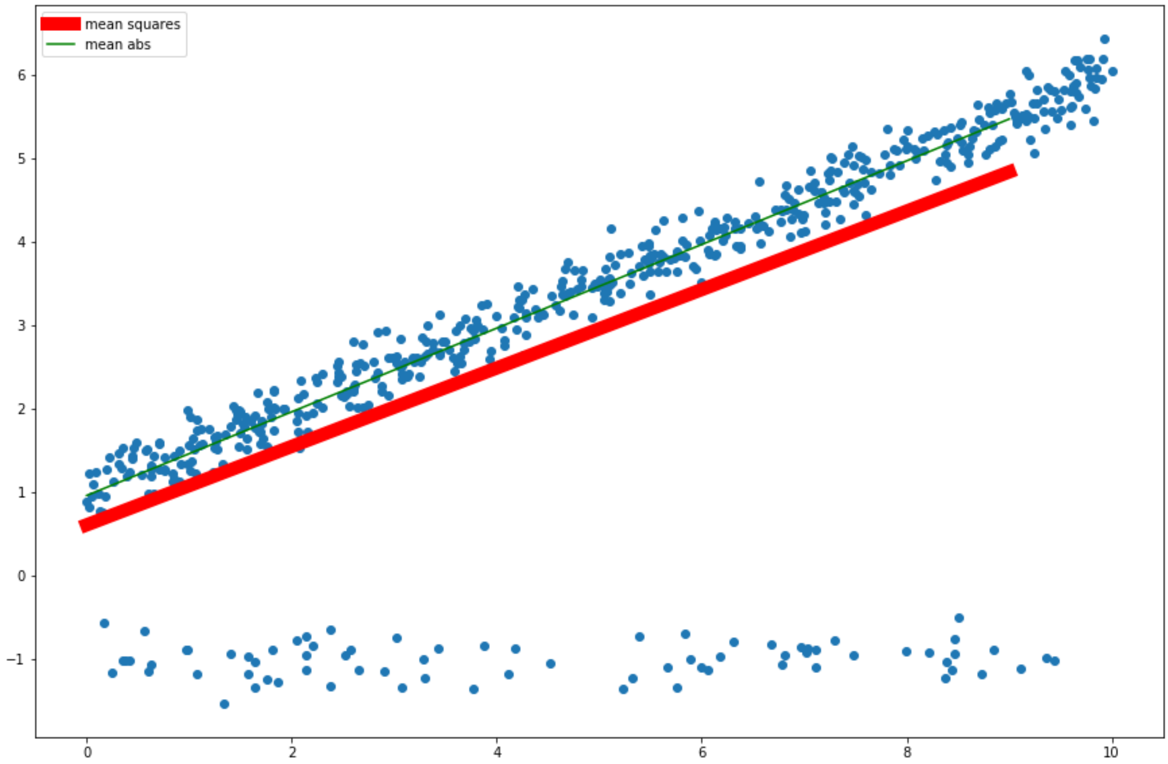
```
sampleX = np.append(sampleX, newX)
sampleY = np.append(sampleY, newY)
```

In [138]:

```
bnds = ((0, 10), (-10, 10))
res_mean_squares = minimize(func_min_sqaures, (2, 0), bounds=bnds)
res_mean_abs = minimize(func_min_abs, (2, 0), bounds=bnds)
```

In [139]:

```
fig = plt.figure(figsize=[15, 10])
plt.scatter(sampleX, sampleY)
plt.plot(np.arange(10), func(res_mean_squares.x, np.arange(10)),
         linewidth=10, color='red', label=r'mean squares')
plt.plot(np.arange(10), func(res_mean_abs.x, np.arange(10)),
         color='green', label=r'mean abs')
plt.legend()
plt.show()
```



Как мы видим, метод минимизации среднего квадрата отклонения более подвержен влиянию выбросов.

In [ ]: