In [1]:

```
import pandas as pd
import numpy as np
import os

pd.set_option("display.max_rows", 100)
pd.set_option('display.max_columns', 200)

import warnings
warnings.filterwarnings('ignore')
%config InlineBackend.figure_format = 'svg'

from tqdm import tqdm

import matplotlib.pylab as plt
from mpl_toolkits.mplot3d.axes3d import Axes3D
import matplotlib.gridspec as gridspec
%matplotlib inline
plt.rcParams['figure.figsize'] = [12.0, 4.0]
# plt.rcParams['figure.dpi'] = 80


from sklearn.metrics import f1_score, recall_score, precision_score, classificat
ion_report
from sklearn.cluster import DBSCAN
from collections import Counter
from sklearn.decomposition import PCA
from sklearn import manifold
from sklearn.model_selection import KFold
```

In [9]:

```
def get_train_sample(X, y, n, bot_rate=0.1):
    X_bot = X[y==1].values
    X_user = X[y==0].values

    if bot_rate<1:
        n_bot = int(bot_rate * n)
    else:
        n_bot = int(bot_rate)
    assert n_bot < n

    bot_indexes = np.random.choice(np.arange(X_bot.shape[0]), size=n_bot, replac
e=False)
    X_train_bot = X_bot[bot_indexes]
    user_indexes = np.random.choice(np.arange(X_user.shape[0]), size=n - n_bot,
replace=False)
    X_train_user = X_user[user_indexes]
    return np.vstack((X_train_bot, X_train_user)), np.hstack((np.ones(n_bot),
np.zeros(n - n_bot))).reshape(-1, 1)

def get_random_sample(X, y, n):
    choosen_indexes = np.random.choice(X.index, size=n, replace=False)
    X_random = X.loc[choosen_indexes]
    y_random = y.loc[choosen_indexes].values
    return X_random, y_random
```

```
def plot_2d_distr_scatter(X_sample, y_sample, labels=None):
    label_1 = np.where(y_sample==1)[0]
    label_0 = np.where(y_sample==0)[0]
    plot_number = 2 if labels!=None else 1

    plt.figure(figsize=(12,5))
    gs = gridspec.GridSpec(1, plot_number)
    ax1 = plt.subplot(gs[0, 0])
    ax1.set_title('Реальное распределение ботов')
    plt.scatter(X_sample[label_1, 0], X_sample[label_1, 1], 10, c='r', label='bo
t')
    plt.scatter(X_sample[label_0, 0], X_sample[label_0, 1], 2, c='g', label='use
r')
    ax1.legend()

    if labels!=None:
        ax2 = plt.subplot(gs[0, 1])
        label_1 = np.where(labels==1)[0]
        label_0 = np.where(labels==0)[0]
        ax2.set_title('Распределение меток')
        ax2.scatter(X_sample[label_1, 0], X_sample[label_1, 1], 10, c='r',
label='label 1')
        ax2.scatter(X_sample[label_0, 0], X_sample[label_0, 1], 2, c='g',
label='label 0')
        ax2.legend()
    plt.show()


def plot_3d_distr_scatter(X_sample, y_sample, labels=None):
    label_1 = np.where(y_sample==1)[0]
    label_0 = np.where(y_sample==0)[0]
    plot_number = 2 if labels!=None else 1

    fig = plt.figure(figsize=[12,5])
    ax1 = fig.add_subplot(1, plot_number, 1, projection='3d')

    xs = X_sample[label_1, 0]
    ys = X_sample[label_1, 1]
    zs = X_sample[label_1, 2]
    ax1.scatter(xs, ys, zs,  c='r', zdir='x', s=5, label='bot')
    ax1.set_title('Реальное распределение ботов')

    xs = X_sample[label_0, 0]
    ys = X_sample[label_0, 1]
    zs = X_sample[label_0, 2]
    ax1.scatter(xs, ys, zs, c='g', zdir='x', s=2, label='user')
    ax1.legend()

    if labels != None:
        label_1 = np.where(labels==1)[0]
        label_0 = np.where(labels==0)[0]

        ax2 = fig.add_subplot(1,2,2, projection='3d')
        ax2.set_title('Распределение меток')
        xs = X_sample[label_1, 0]
        ys = X_sample[label_1, 1]
        zs = X_sample[label_1, 2]
        ax2.scatter(xs, ys, zs,  c='r', zdir='x', s=5, label='label_1')

        xs = X_sample[label_0, 0]
        ys = X_sample[label_0, 1]
```

```
        zs = X_sample[label_0, 2]
        ax2.scatter(xs, ys, zs, c='g', zdir='x', s=2, label='label_0')
        ax2.legend()
    plt.show()
```

DBSCAN

Получив все эти хорошие представления пора бы уже приступить к собственно кластеризации. Для этих целей был выбран алгоритм **DBSCAN**, который строит кластера в зависимости от плотности точек. Такой алгоритм выглядит очень удачным после того, что мы наблюдали на предыдущих картинках. Как водится, алгоритм кластеризации выделяет 2 кластера честных и нечестных транзакций. На самом деле выдаст больше, однако мы будем считать точки самого жирного класстера хорошими, а все остальные - аномальными.

Алгоритм достаточно тяжеловесный, так что так или иначе нам пригодятся и методы понижения размерности и какое-то разбиение всего множества наблюдений.

Помимо него попробую применить **One-class svm**

В принципе, если прямо гнаться за точностью, можно попробовать разбиения, при которых каждый элемент попадает сразу в **несколько выборок**, а потом оценить, как часто он попадает в подозрительные на фрод. Или просто по результатам этих кластеризаций отранжировать по подозрительности все точки

In [3]:

```
X = pd.read_csv('clean_data.csv')
y = pd.read_csv('clean_target.csv', header=None).loc[:, 0]
X.head()
```

Out[3]:

	var1	var2	var3	var4	var5	var6	var7	var8
0	-0.696436	-0.043653	1.682442	0.967586	-0.245495	0.352104	0.193170	0.0828
1	0.610419	0.159634	0.110432	0.314644	0.043550	-0.062717	-0.063533	0.0714
2	-0.695692	-0.803812	1.176228	0.266639	-0.365135	1.371065	0.638096	0.2078
3	-0.494884	-0.111096	1.189352	-0.606106	-0.007480	0.949734	0.191566	0.3167
4	-0.593198	0.526455	1.027316	0.282965	-0.295471	0.073043	0.478044	-0.227(

Без понижения размерности

Перед тем как чего-нибудь считать, пропатчим наш Дбскан и помимо правленных меток добавим ему параметр min_bot_num/max_bot_num - минимальное/максимальное число аномалий, которое он должен вернуть.

Добьемся мы этого инкрементным/декрементым изменением параметра eps на некоторое заданное значение до тех пор, пока не выполнится это условие. По идее если заморочиться, то можно и значение этого шага сделать адаптивным, но мы люди простые.

In [4]:

```
def my_db_scan(X, min_samples, eps, min_bot_num=0, max_bot_num=100000, verbose=T
rue):
    visited_eps = set()
    while(True):
        if eps in visited_eps:
            return final_labels
        visited_eps.add(eps)

        if verbose:
            print('Текущий параметр:', eps)

        dbs = DBSCAN(min_samples=min_samples, eps=eps, n_jobs=-1)
        dbs_labels = dbs.fit_predict(X)

        label_counter = Counter(dbs_labels)
        main_cluster_label = max(label_counter, key=label_counter.get)
        final_labels = dbs_labels.copy()
        final_labels[dbs_labels!=main_cluster_label] = 1
        final_labels[dbs_labels==main_cluster_label] = 0

        finded_bots_num = final_labels.sum()
        if verbose:
            print('Всего отмечено: {}'.format(finded_bots_num))
        if  finded_bots_num < min_bot_num:
            eps -= 0.1
            continue
        if finded_bots_num > max_bot_num:
            eps += 0.1
            continue
        if verbose:
            print('Всего ботов: {}'.format(y_sample.sum()))
            print('Верно отмечено: {}'.format(len([x for x in np.where(
                y_sample==1)[0] if x in np.where(final_labels==1)[0]])))
            print('Ф-мера: {}'.format(f1_score(y_pred=final_labels, y_true=y_sam
ple)))
            print('Полнота: {}'.format(recall_score(y_pred=final_labels,
y_true=y_sample)))
            print('Точность: {}'.format(precision_score(y_pred=final_labels, y_t
rue=y_sample)))
        return final_labels, eps

X_sample, y_sample = get_random_sample(X, y, 20000)
```

Замечание. metric='canberra' следует брать меньше eps. Также при увеличении числа точек, есть резон уменьшать eps. При увеличении размерности X стоит начать думать об увеличении.

In [126]:

```
%time y_labels, _ = my_db_scan(X_sample, min_samples=2, eps=6.1, min_bot_num=10,
 max_bot_num=300)
```

```
Текущий параметр: 6.1
Всего отмечено: 259
Всего ботов: 28
Верно отмечено: 20
Ф-мера: 0.1393728222996516
Полнота: 0.7142857142857143
CPU times: user 42.6 s, sys: 80 ms, total: 42.7 s
Wall time: 42.7 s
```

В принципе получилось достаточно неплохо и считалось не так уж прям медленно.

In [125]:

```
%time y_labels, _ = my_db_scan(X_sample, min_samples=2, eps=7, min_bot_num=10, m
ax_bot_num=300)
```

```
Текущий параметр: 7
Всего отмечено: 159
Всего ботов: 28
Верно отмечено: 16
Ф-мера: 0.17112299465240643
Полнота: 0.5714285714285714
CPU times: user 48.7 s, sys: 264 ms, total: 49 s
Wall time: 49 s
```

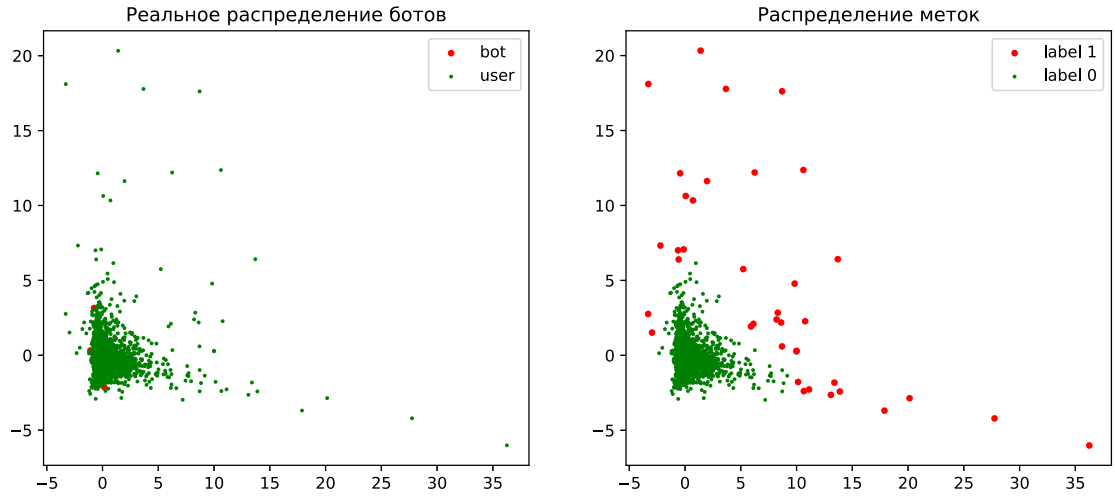
PCA-2

In [141]:

```
pca = PCA(n_components=2, random_state=42).fit(X)
X_pca = pca.transform(X_sample)

y_labels, _ = my_db_scan(X_pca, min_samples=2, eps=1, min_bot_num=10, max_bot_nu
m=300)
plot_2d_distr_scatter(X_pca, y_sample, y_labels)
```

```
Текущий параметр: 1
Всего отмечено: 38
Всего ботов: 6
Верно отмечено: 0
Ф-мера: 0.0
Полнота: 0.0
Точность: 0.0
```



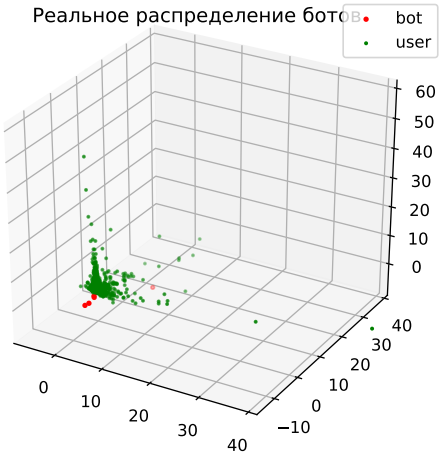
Совсем грусть и тоска. Результаты совсем удручающие, но глядя на картинки понимаешь, что ожидать большего и не приходилось.

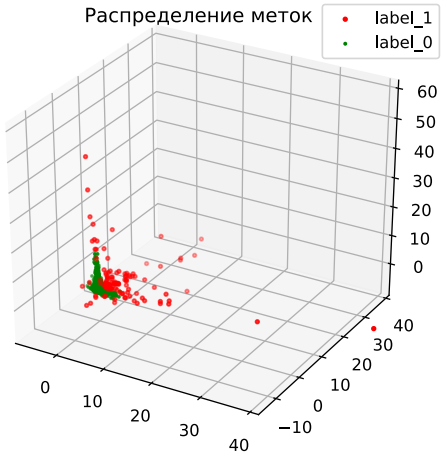
PCA-3

In [142]:

```
pca = PCA(n_components=3, random_state=42).fit(X)
X_pca = pca.fit_transform(X_sample)
y_labels, _ = my_db_scan(X_pca, min_samples=1, eps=1, min_bot_num=10, max_bot_nu
m=150)
plot_3d_distr_scatter(X_pca, y_sample, y_labels)
```

```
Текущий параметр: 1
Всего отмечено: 148
Всего ботов: 6
Верно отмечено: 5
Ф-мера: 0.06493506493506494
Полнота: 0.8333333333333334
Точность: 0.033783783783783786
```

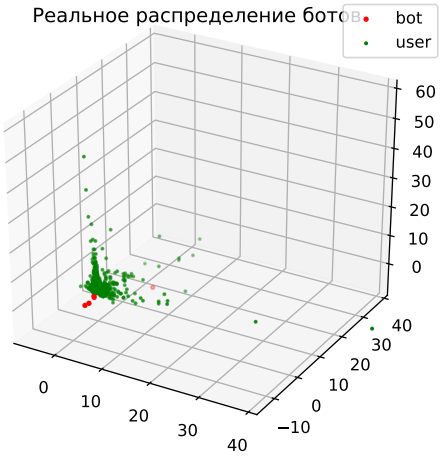


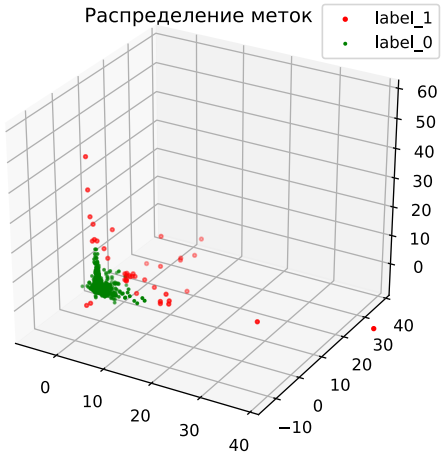


In [143]:

```
pca = PCA(n_components=3, random_state=42).fit(X)
X_pca = pca.fit_transform(X_sample)
y_labels, _ = my_db_scan(X_pca, min_samples=1, eps=2.3, min_bot_num=10, max_bot_
num=150)
plot_3d_distr_scatter(X_pca, y_sample, y_labels)
```

```
Текущий параметр: 2.3
Всего отмечено: 42
Всего ботов: 6
Верно отмечено: 3
Ф-мера: 0.125
Полнота: 0.5
Точность: 0.07142857142857142
```





Отлично, получился относительно неплохой результат, причем достаточно шустро считается.
Попробуем разбить всю нашу совокупную выборку на подвыборки, на которой и посчитать аномалии независимо

In [56]:

```
pca = PCA(n_components=3, random_state=42).fit(X)
kf=KFold(n_splits=18, random_state=42, shuffle=True)
eps = 2.3
res_labels = pd.DataFrame()
for i in tqdm(kf.split(X), total=kf.n_splits):
    X_sample = X.loc[i[1]]
    y_sample = y[i[1]]
    X_pca = pca.fit_transform(X_sample)
    y_labels, eps = my_db_scan(X_pca, min_samples=1, eps=2.3, min_bot_num=30, ma
x_bot_num=80, verbose=False)
    temp_df = pd.DataFrame(index=i[1], data=y_labels, columns=
['random_state_42'])
    res_labels = res_labels.append(temp_df)
res_labels.sort_index(inplace=True)
print('Всего ботов: {}'.format(y.sum()))
print('Всего отмечено: {}'.format(res_labels.sum()[0]))
print('Верно отмечено: {}'.format(len([x for x in np.where(
    y==1)[0] if x in np.where(res_labels.values==1)[0]])))
print('Ф-мера: {}'.format(f1_score(y_pred=res_labels.values, y_true=y)))
print('Полнота: {}'.format(recall_score(y_pred=res_labels.values, y_true=y)))
print('Точность: {}'.format(precision_score(y_pred=res_labels.values,
y_true=y)))
res_labels.to_csv('careful_pca_3.csv')
```

```
Всего ботов: 473
Всего отмечено: 993
Верно отмечено: 118
Ф-мера: 0.16098226466575716
Полнота: 0.24947145877378435
Точность: 0.11883182275931521
```

In [5]:

```
pca = PCA(n_components=3, random_state=42).fit(X)
kf=KFold(n_splits=18, random_state=42, shuffle=True)
eps = 2
res_labels = pd.DataFrame()
for i in tqdm(kf.split(X), total=kf.n_splits):
    X_sample = X.loc[i[1]]
    y_sample = y[i[1]]
    X_pca = pca.fit_transform(X_sample)
    y_labels, eps = my_db_scan(X_pca, min_samples=1, eps=eps, min_bot_num=30, ma
x_bot_num=100, verbose=False)
    temp_df = pd.DataFrame(index=i[1], data=y_labels, columns=
['random_state_42'])
    res_labels = res_labels.append(temp_df)
res_labels.sort_index(inplace=True)
print('Всего ботов: {}'.format(y.sum()))
print('Всего отмечено: {}'.format(res_labels.sum()[0]))
print('Верно отмечено: {}'.format(len([x for x in np.where(
    y==1)[0] if x in np.where(res_labels.values==1)[0]])))
print('Ф-мера: {}'.format(f1_score(y_pred=res_labels.values, y_true=y)))
print('Полнота: {}'.format(recall_score(y_pred=res_labels.values, y_true=y)))
print('Точность: {}'.format(precision_score(y_pred=res_labels.values,
y_true=y)))
res_labels.to_csv('adaptive_pca_3.csv')
```

```
100%|██████████| 18/18 [04:55<00:00, 21.24s/it]

Всего ботов: 473
Всего отмечено: 1374
Верно отмечено: 139
Ф-мера: 0.1505143475906876
Полнота: 0.2938689217758985
Точность: 0.10116448326055313
```

In [20]:

```
pca = PCA(n_components=3, random_state=42).fit(X)
kf=KFold(n_splits=18, random_state=42, shuffle=True)
eps = 1
res_labels = pd.DataFrame()
for i in tqdm(kf.split(X), total=kf.n_splits):
    X_sample = X.loc[i[1]]
    y_sample = y[i[1]]
    X_pca = pca.fit_transform(X_sample)
    y_labels, eps = my_db_scan(X_pca, min_samples=1, eps=eps, min_bot_num=30, ma
x_bot_num=150, verbose=False)
    temp_df = pd.DataFrame(index=i[1], data=y_labels, columns=
['random_state_42'])
    res_labels = res_labels.append(temp_df)
res_labels.sort_index(inplace=True)
print('Всего ботов: {}'.format(y.sum()))
print('Всего отмечено: {}'.format(res_labels.sum()[0]))
print('Верно отмечено: {}'.format(len([x for x in np.where(
    y==1)[0] if x in np.where(res_labels.values==1)[0]])))
print('Ф-мера: {}'.format(f1_score(y_pred=res_labels.values, y_true=y)))
print('Полнота: {}'.format(recall_score(y_pred=res_labels.values, y_true=y)))
print('Точность: {}'.format(precision_score(y_pred=res_labels.values,
y_true=y)))
res_labels.to_csv('agro_pca_3.csv')
```

```
100%|██████████| 18/18 [03:40<00:00, 13.85s/it]

Всего ботов: 473
Всего отмечено: 2179
Верно отмечено: 178
Ф-мера: 0.13423831070889897
Полнота: 0.3763213530655391
Точность: 0.08168884809545664
```

Отмечу, что на разбиение на 30тысяч точек оперативной памяти уже не хватает.

PCA-10

In [12]:

```
X_sample, y_sample = get_random_sample(X, y, 15000)
pca = PCA(n_components=10, random_state=42).fit(X)
X_pca = pca.fit_transform(X_sample)
y_labels, _ = my_db_scan(X_pca, min_samples=1, eps=2, min_bot_num=10, max_bot_nu
m=150)
```

```
Текущий параметр: 2
Всего отмечено: 609
Текущий параметр: 2.1
Всего отмечено: 531
Текущий параметр: 2.2
Всего отмечено: 483
Текущий параметр: 2.3000000000000003
Всего отмечено: 439
Текущий параметр: 2.4000000000000004
Всего отмечено: 398
Текущий параметр: 2.5000000000000004
Всего отмечено: 358
Текущий параметр: 2.6000000000000005
Всего отмечено: 295
Текущий параметр: 2.7000000000000006
Всего отмечено: 271
Текущий параметр: 2.8000000000000007
Всего отмечено: 246
Текущий параметр: 2.900000000000001
Всего отмечено: 222
Текущий параметр: 3.000000000000001
Всего отмечено: 213
Текущий параметр: 3.100000000000001
Всего отмечено: 190
Текущий параметр: 3.200000000000001
Всего отмечено: 176
Текущий параметр: 3.300000000000001
Всего отмечено: 163
Текущий параметр: 3.4000000000000012
Всего отмечено: 147
Всего ботов: 28
Верно отмечено: 16
Ф-мера: 0.18285714285714286
Полнота: 0.5714285714285714
Точность: 0.10884353741496598
```

In [13]:

```
pca = PCA(n_components=10, random_state=42).fit(X)
kf=KFold(n_splits=18, random_state=42, shuffle=True)
eps = 3.4
res_labels = pd.DataFrame()
for i in tqdm(kf.split(X), total=kf.n_splits):
    X_sample = X.loc[i[1]]
    y_sample = y[i[1]]
    X_pca = pca.fit_transform(X_sample)
    y_labels, eps = my_db_scan(X_pca, min_samples=1, eps=eps, min_bot_num=30, ma
x_bot_num=150, verbose=False)
    temp_df = pd.DataFrame(index=i[1], data=y_labels, columns=
['random_state_42'])
    res_labels = res_labels.append(temp_df)
res_labels.sort_index(inplace=True)
print('Всего ботов: {}'.format(y.sum()))
print('Всего отмечено: {}'.format(res_labels.sum()[0]))
print('Верно отмечено: {}'.format(len([x for x in np.where(
    y==1)[0] if x in np.where(res_labels.values==1)[0]])))
print('Ф-мера: {}'.format(f1_score(y_pred=res_labels.values, y_true=y)))
print('Полнота: {}'.format(recall_score(y_pred=res_labels.values, y_true=y)))
print('Точность: {}'.format(precision_score(y_pred=res_labels.values,
y_true=y)))
res_labels.to_csv('adaptive_pca_10.csv')
```

```
100%|██████████| 18/18 [06:00<00:00, 21.57s/it]

Всего ботов: 473
Всего отмечено: 2168
Верно отмечено: 237
Ф-мера: 0.1794774706550549
Полнота: 0.5010570824524313
Точность: 0.10931734317343174
```

MDS-2

In [22]:

```
del(X_pca)
```