In [1]:

```
import pandas as pd
import numpy as np
from matplotlib import pylab as plt
from matplotlib.colors import ListedColormap

from sklearn import datasets
from sklearn.model_selection import train_test_split
```

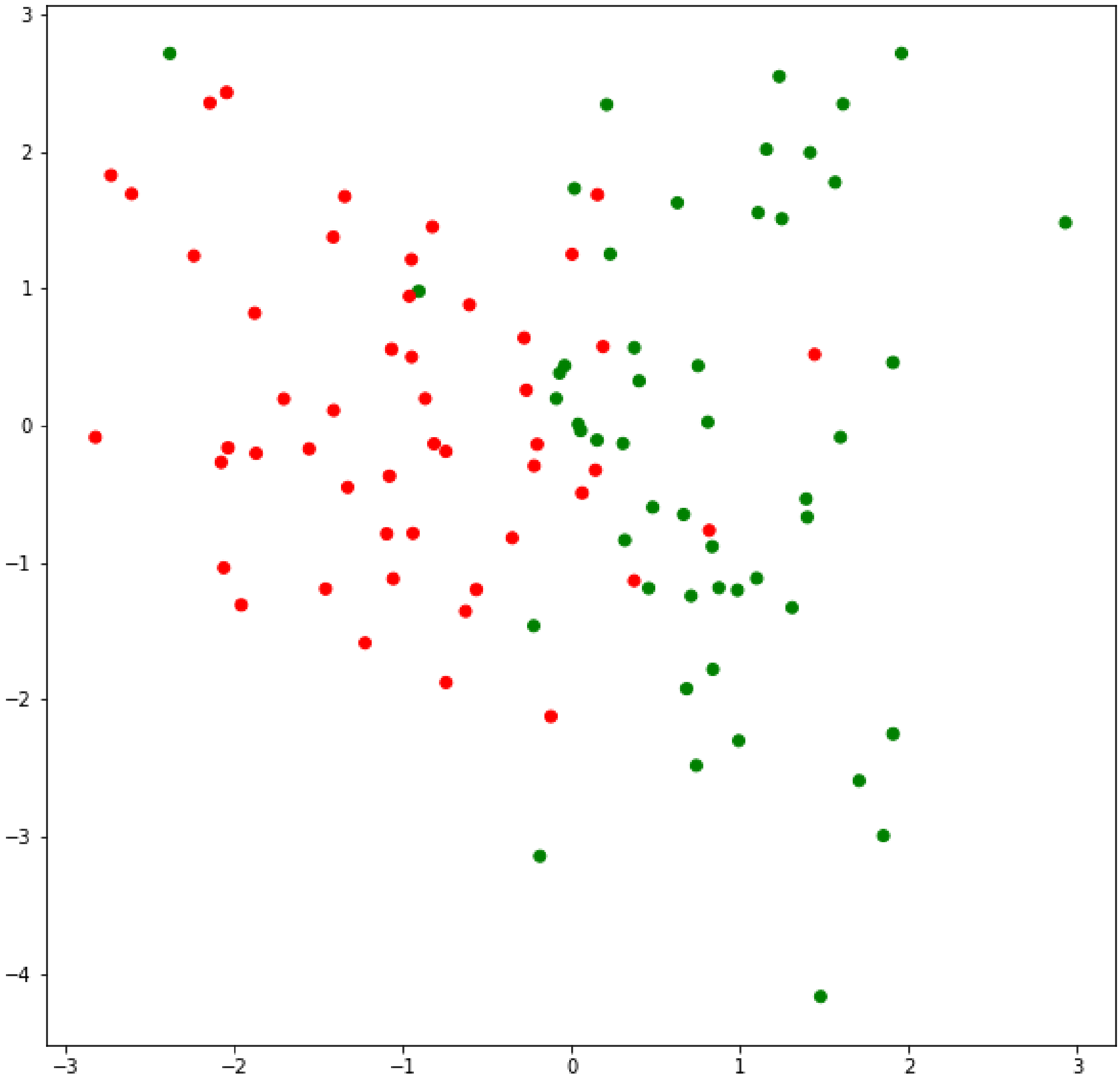
DATA IMPORT AND VISUALISATION

In [2]:

```
colors = ListedColormap(['red', 'green'])
class_data = datasets.make_classification(n_features = 4,
                                          n_informative = 3,
                                          n_redundant = 1,
                                          n_clusters_per_class = 4,
                                          random_state = 453)
plt.figure(figsize=(10,10))
plt.scatter(list(map(lambda x: x[2], class_data[0])),
            list(map(lambda x: x[1], class_data[0])),
            c = class_data[1], cmap = colors)
```

Out[2]:

```
<matplotlib.collections.PathCollection at 0x7fc349adea50>
```


3
2
1
0
−1
−2
−3
−4
−3
−2
−1
0
1
2
3

In [3]:

```
pd.DataFrame(class_data[0])
```

Out[3]:

	0	1	2	3
0	0.699120	-0.667022	1.397948	-1.260264
1	2.122415	0.198573	-0.867802	-1.375430
2	-1.402894	-0.168801	-1.556477	0.767410
3	3.180725	2.719282	-2.382140	0.810743
4	0.498283	0.520943	1.440881	0.323670
...	...	...	...	...
95	-1.490561	-0.084033	1.594531	1.077410
96	1.426655	0.198703	-0.090098	-0.823537
97	1.510295	-0.819527	-0.351522	-2.125629
98	2.618080	1.674698	-1.345389	0.015435
99	-0.289224	-1.132502	0.370808	-1.134489

100 rows × 4 columns

In [4]:

```
data = pd.DataFrame(class_data[0], columns=['p1','p2','p3','p4'])
data['class'] = class_data[1]
data
```

Out[4]:

	p1	p2	p3	p4	class
0	0.699120	-0.667022	1.397948	-1.260264	1
1	2.122415	0.198573	-0.867802	-1.375430	0
2	-1.402894	-0.168801	-1.556477	0.767410	0
3	3.180725	2.719282	-2.382140	0.810743	1
4	0.498283	0.520943	1.440881	0.323670	0
...	...	...	...	...	...
95	-1.490561	-0.084033	1.594531	1.077410	1
96	1.426655	0.198703	-0.090098	-0.823537	1
97	1.510295	-0.819527	-0.351522	-2.125629	0
98	2.618080	1.674698	-1.345389	0.015435	0
99	-0.289224	-1.132502	0.370808	-1.134489	0

100 rows × 5 columns

SVM

В качестве базового решения для меня будет являться SVM

In [5]:

```
from sklearn.svm import SVC
from sklearn.metrics import roc_auc_score, accuracy_score, f1_score, classificat
ion_report
from sklearn.model_selection import StratifiedKFold

data = data.iloc[np.random.permutation(len(data))]
X = data[['p1','p2','p3','p4']]
Y = data['class']
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2)

f1=[]
SVM = SVC(kernel='linear')
SVM.fit(X_train,y_train)
predictions_SVM = SVM.predict(X_test)

print(classification_report(y_test, predictions_SVM))
```

```
              precision    recall  f1-score   support

           0       0.90      0.90      0.90        10
           1       0.90      0.90      0.90        10

    accuracy                           0.90        20
   macro avg       0.90      0.90      0.90        20
weighted avg       0.90      0.90      0.90        20
```

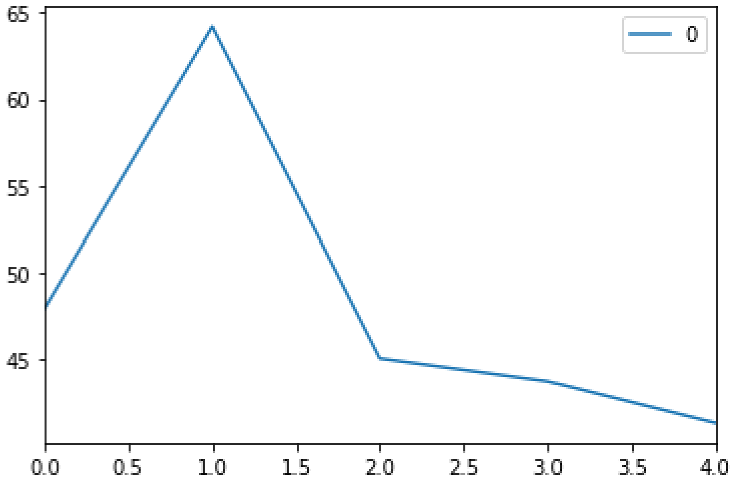
SVM по K-фолдам для двумерной задачи

При разных попытках обучить получаются разбросанные значения, проведем проверку по K-фолдам

In [6]:

```
def SVM_model(df):
    data = df.iloc[np.random.permutation(len(df))]

    X = np.array(data[['p1','p2']])
    Y = np.array(data['class'])

    skf = StratifiedKFold(n_splits=5)
    skf.get_n_splits(X, Y)
    print(skf)

    print('Stratified K-Fold:')
    f1=[] # array of f1 scores
    for train_index, test_index in skf.split(X, Y):
        X_train, X_test = X[train_index], X[test_index]
        y_train, y_test = Y[train_index], Y[test_index]

        SVM = SVC(kernel='linear')
        SVM.fit(X_train,y_train)

        predictions_SVM = SVM.predict(X_test)
        f1.append(round(f1_score(predictions_SVM, y_test, average='macro')*100,3
))
        print("SVM F1 Score -> ", f1[-1])

    # avg f1
    print()
    print(f'F1 average: {round(np.array(f1).mean(), 3)} %')
    print(f'F1 std:\t {round(np.array(f1).std(), 3)}')
    print(f'F1 var:\t {round(np.array(f1).var(), 3)}')
    print()
    pd.DataFrame(f1).plot()


print('working with data...')
SVM_model(data)
```

```
working with data...
StratifiedKFold(n_splits=5, random_state=None, shuffle=False)
Stratified K-Fold:
SVM F1 Score ->  47.917
SVM F1 Score ->  64.194
SVM F1 Score ->  45.055
SVM F1 Score ->  43.734
SVM F1 Score ->  41.333

F1 average: 48.447 %
F1 std:  8.156
F1 var:  66.515
```



My model

In [7]:

```
def job_func4D(w0, w1, w2, w3, w4, el):
    return el[0]*w0 + el[1]*w1 + el[2]*w2 + el[3]*w3 + w4

# разделяющая кривая в думерном пространстве
def job_func2D(w0, w1, el):
    return el*w1 + w0

"""по сути это и есть моя функция потерь. Я даю штрафы если элемент
класса оказывается не на своей стороне разделения"""
# сверху класс 1
def penalty_2D(real, pred, cl):
    if real > pred:
        cl_pred = 1
        return 0 if cl_pred == cl else abs(pred-real)
    elif real == pred:
        return 0
    else:
        cl_pred = 0
        return 0 if cl_pred == cl else abs(pred-real)
```

In [8]:

```
def Differential_evolution(X, Y, n_population=2500, n_generations=1600, mutation
_force=100, probability=0.8):
    #"""here you can do any optimization technique you want. I try differential
 evolution""

    dimension = 2 #для моей двумерной задачи!

    #initialize population
    w_population=[]
    for k in range(n_population):
        w_population.append(10*np.random.randint(-100, 100)*np.random.sample(siz
e=(1,dimension)))

    w_populations=[]
    n_step=0

    while n_step < n_generations:
        w_populations.append(w_population)

        #choose 3 vectors, mutate first one in the direction of (second - thir
d), compare with first
        w_index = np.random.randint(0, n_population, size=(1,3))
        w1 = np.array(w_population[w_index[0][0]])
        w2 = np.array(w_population[w_index[0][1]])
        w3 = np.array(w_population[w_index[0][2]])

        # мутировавшие гены
        w_mutant = np.add(w1, (mutation_force * np.subtract(w2,w3)))

        # создаем наследника с частью мутировавших ген, с частью ген батьки
        w_desc = []
        for k in range(dimension):
            proba = np.random.uniform()
            if proba<=probability:
                w_desc.append(w_mutant[0][k])
            else:
                w_desc.append(w1[0][k])

        w_desc = np.array([w_desc])

        #считаем у кого больше погрешность у сына или батьки
        error_1, error_2 = 0, 0
        for i in range(len(X)):
            real = X[i][1]
            pred = X[i][0]*w1[0][0] + w1[0][1]
            pred_mut = X[i][0]*w_desc[0][0]+w_desc[0][1]
            cl = Y[i]

            error_1 += penalty_2D(real, pred, cl)
            error_2 += penalty_2D(real, pred_mut, cl)

        if error_2 < error_1:
            w_population.pop(w_index[0][0])
            w_population.insert(w_index[0][0],w_desc)

        n_step+=1

    w_best = np.mean(np.array(w_population),axis=0)
    return w_best
```

In [17]:

```
def my_model(df):
    data = df.iloc[np.random.permutation(len(df))]

    X = np.array(data[['p1','p2','p3','p4']])
    Y = np.array(data['class'])

    skf = StratifiedKFold(n_splits=5)
    skf.get_n_splits(X, Y)
    print(skf)

    print('Stratified K-Fold:')
    f1_max = 0
    f1=[] # array of f1 scores
    for train_index, test_index in skf.split(X, Y):
        X_train, X_test = X[train_index], X[test_index]
        y_train, y_test = Y[train_index], Y[test_index]

        # fit
        weights = Differential_evolution(X_train,y_train)
        w1, w2 = weights.tolist()[0][0], weights.tolist()[0][1]
        # evaluate
        predictions_my_model = []
        for i in range(len(X_test)):
            predictions_my_model.append( 1 if (X[i][1] > X[i][0]*w1+w2) else 0 )
        f1.append(round(f1_score(predictions_my_model, y_test, average='macro')*
100,3))
        print("SVM F1 Score -> ", f1[-1])

        if f1_max < f1[-1]:
            f1_max = f1[-1]
            best_weights = weights

    # avg f1
    print()
    print(f'F1 average: {round(np.array(f1).mean(), 3)} %')
    print(f'F1 std:\t {round(np.array(f1).std(), 3)}')
    print(f'F1 var:\t {round(np.array(f1).var(), 3)}')
    print(f'f1_max:{f1_max}')
    return best_weights


print('working with data...')
weights = my_model(data)
```

```
working with data...
StratifiedKFold(n_splits=5, random_state=None, shuffle=False)
Stratified K-Fold:
SVM F1 Score ->  33.504
SVM F1 Score ->  58.333
SVM F1 Score ->  49.495
SVM F1 Score ->  49.495
SVM F1 Score ->  45.055

F1 average: 47.176 %
F1 std:  8.083
F1 var:  65.331
f1_max:58.333
```

In [18]:

```
w1, w2 = weights.tolist()[0][0], weights.tolist()[0][1]
w1,w2
```

Out[18]:

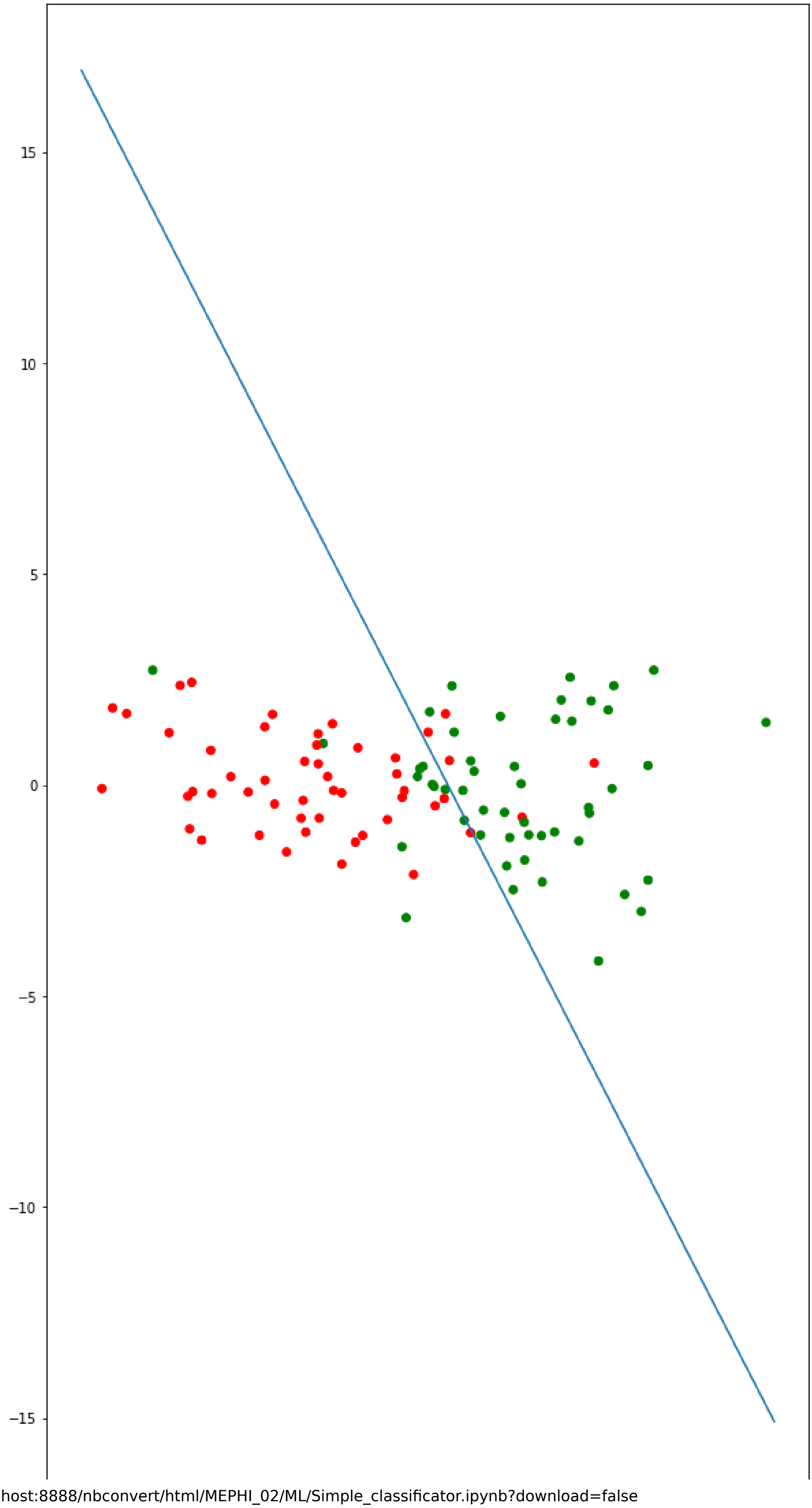
```
(-5.337142924720051, -0.9228511662476016)
```

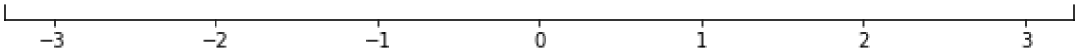
In [19]:

```
x = np.linspace(-3, 3)
plt.figure(figsize=(10,20))
plt.plot(x, [(w1*x-w2) for x in x])
plt.scatter(list(map(lambda x: x[2], class_data[0])),
            list(map(lambda x: x[1], class_data[0])),
            c = class_data[1], cmap = colors)
```

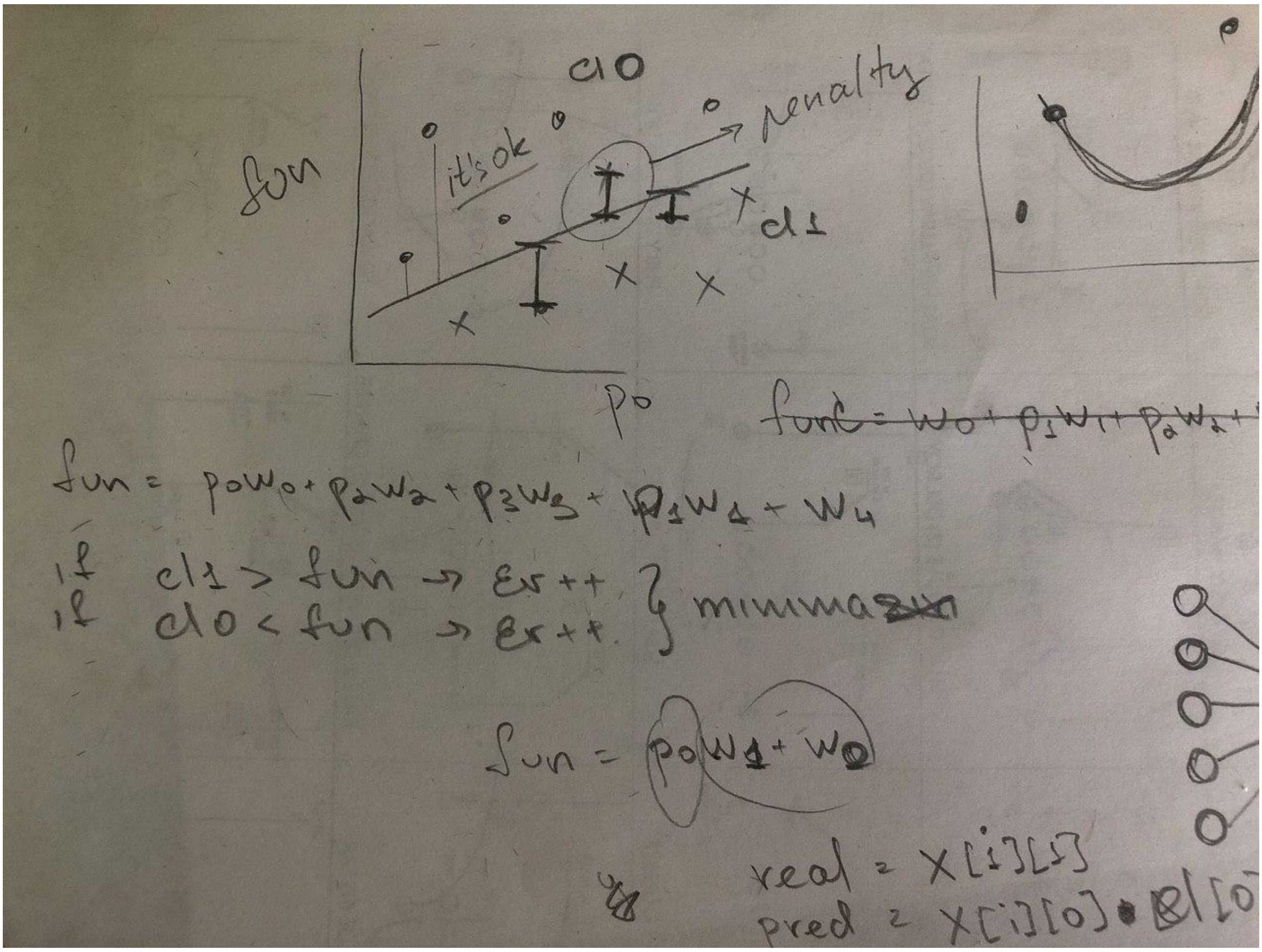
Out[19]:

```
<matplotlib.collections.PathCollection at 0x7fc3495a64d0>
```


15
10
5
0
−5
−10
−15



Я не знаю, на всякий случай прикрепил то как я думал про штрафы... все что выше линии классификатора - один клас, ниже - другой. если элемент класса оказывается не по ту сторону границы, то добавляем штраф в виде манхетеновского расстояния по у до границы



Думал как можно сделать модель с оберткой... в частности обернуть функцию потерь penalty в обертку

Оставлю тут пока что

In [ ]:

```
def benchmark(func):
    import time

    def wrapper():
        start = time.time()
        func()
        end = time.time()
        print('[*] Время выполнения: {} секунд.'.format(end-start))
    return wrapper

@benchmark
def fetch_webpage():
    import requests
    webpage = requests.get('https://google.com')

fetch_webpage()
```