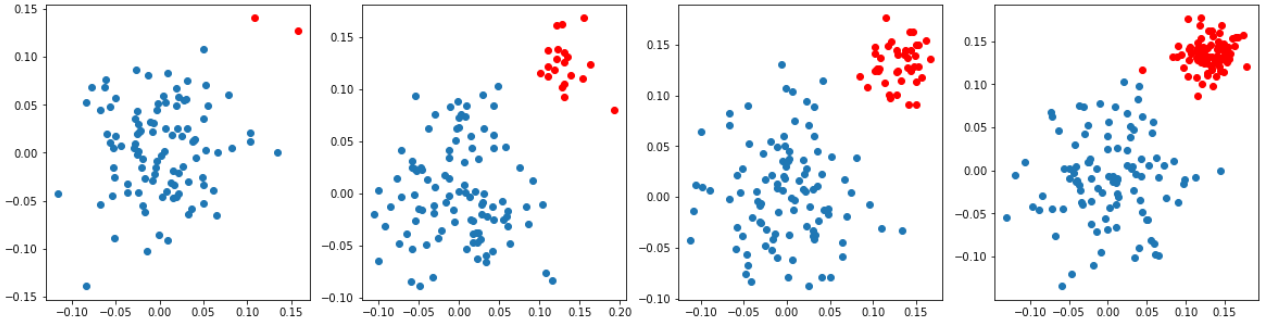
In [16]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import SGDClassifier
from sklearn.linear_model import LogisticRegression
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler, Normalizer
import matplotlib.pyplot as plt
from sklearn.svm import SVC
import warnings
warnings.filterwarnings("ignore")
```

In [17]:
```
def draw_line(coef,intercept, mi, ma):
    # for the separating hyper plane ax+by+c=0, the weights are [a, b] and the intercep
    # to draw the hyper plane we are creating two points
    # 1. ((b*min-c)/a, min) i.e ax+by+c=0 ==> ax = (-by-c) ==> x = (-by-c)/a here in pl
    # 2. ((b*max-c)/a, max) i.e ax+by+c=0 ==> ax = (-by-c) ==> x = (-by-c)/a here in pl
    points=np.array([[((-coef[1]*mi - intercept)/coef[0]), mi],[((-coef[1]*ma - interce
#     print(points[:,0], points[:,1],'\n')
    plt.plot(points[:,0], points[:,1])
```

What if Data is imabalanced

1. As a part of this task you will observe how linear models work in case of data imbalanced
2. observe how hyper plane is changs according to change in your learning rate.
3. below we have created 4 random datasets which are linearly separable and having class imbalance
4. in the first dataset the ratio between positive and negative is 100 : 2, in the 2nd data its 100:20,
in the 3rd data its 100:40 and in 4th one its 100:80

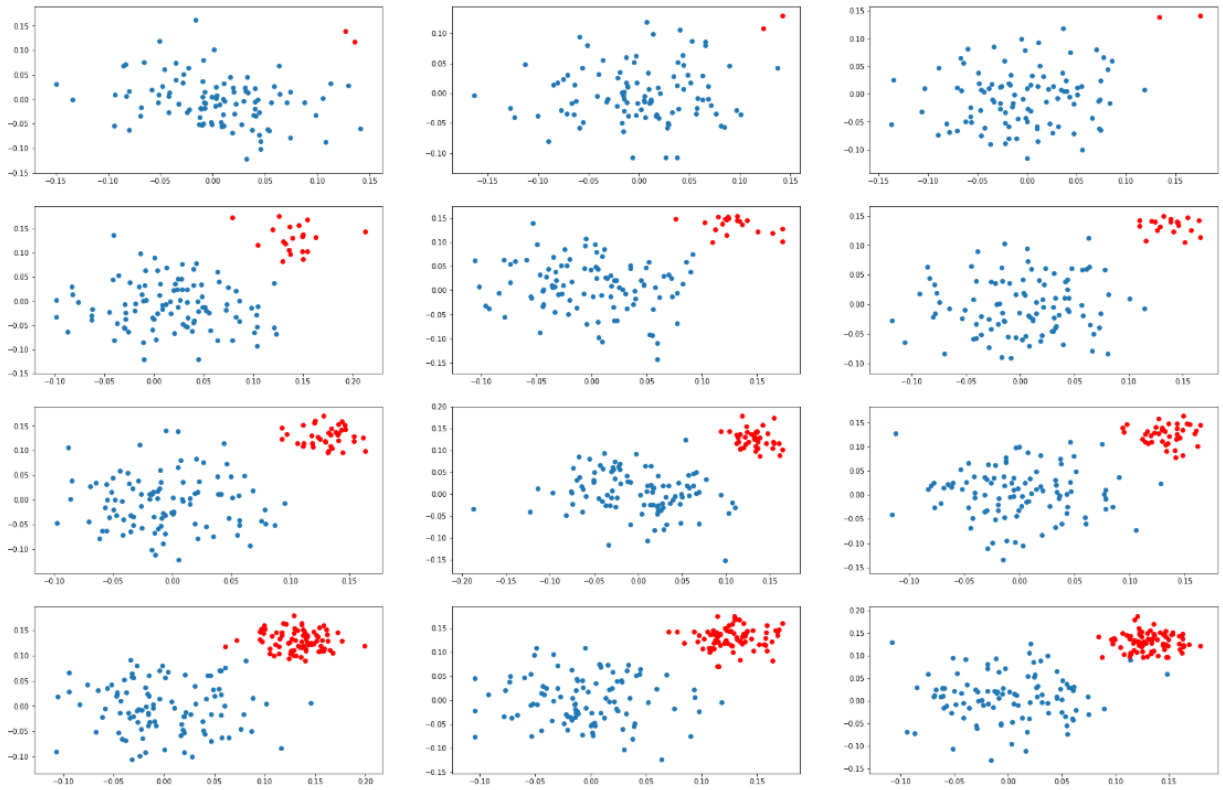
In [18]:
```
# here we are creating 2d imbalanced data points
ratios = [(100,2), (100, 20), (100, 40), (100, 80)]

plt.figure(figsize=(20,5))

for j,i in enumerate(ratios):
    plt.subplot(1, 4, j+1)
    X_p=np.random.normal(0,0.05,size=(i[0],2))
    X_n=np.random.normal(0.13,0.02,size=(i[1],2))
    y_p=np.array([1]*i[0]).reshape(-1,1)
    y_n=np.array([0]*i[1]).reshape(-1,1)
    X=np.vstack((X_p,X_n))
    y=np.vstack((y_p,y_n))
    plt.scatter(X_p[:,0],X_p[:,1])
    plt.scatter(X_n[:,0],X_n[:,1],color='red')
plt.show()
```



> your task is to apply SVM (sklearn.svm.SVC) and LR (sklearn.linear_model.LogisticRegression) with different regularization strength [0.001, 1, 100]

Task 1: Applying SVM

1. you need to create a grid of plots like this



in each of the cell[i][j] you will be drawing the hyper plane that you get after applying SVM on ith dataset and
jth learnig rate

i.e

Plane(SVM().fit(D1, C=0.001))	Plane(SVM().fit(D1, C=1))	Plane(SVM().fit(D1, C=100))
Plane(SVM().fit(D2, C=0.001))	Plane(SVM().fit(D2, C=1))	Plane(SVM().fit(D2, C=100))

```
Plane(SVM().fit(D3, C=0.001))   Plane(SVM().fit(D3, C=1))   Plane(SVM().fit(D3, C=100))

Plane(SVM().fit(D4, C=0.001))   Plane(SVM().fit(D4, C=1))   Plane(SVM().fit(D4, C=100))
```

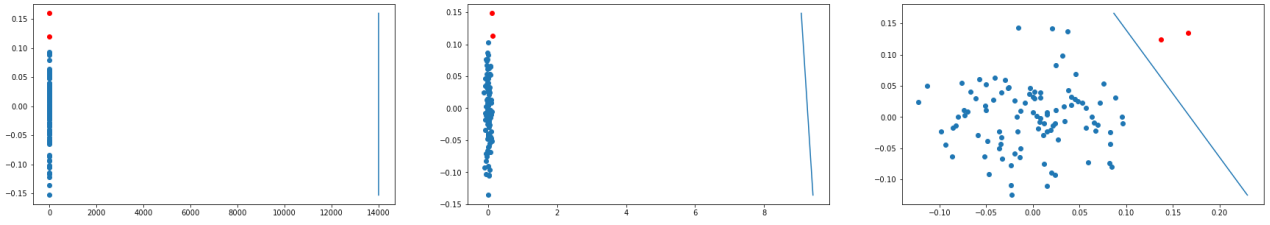
if you can do, you can represent the support vectors in different colors, which will help us understand the position of hyper plane

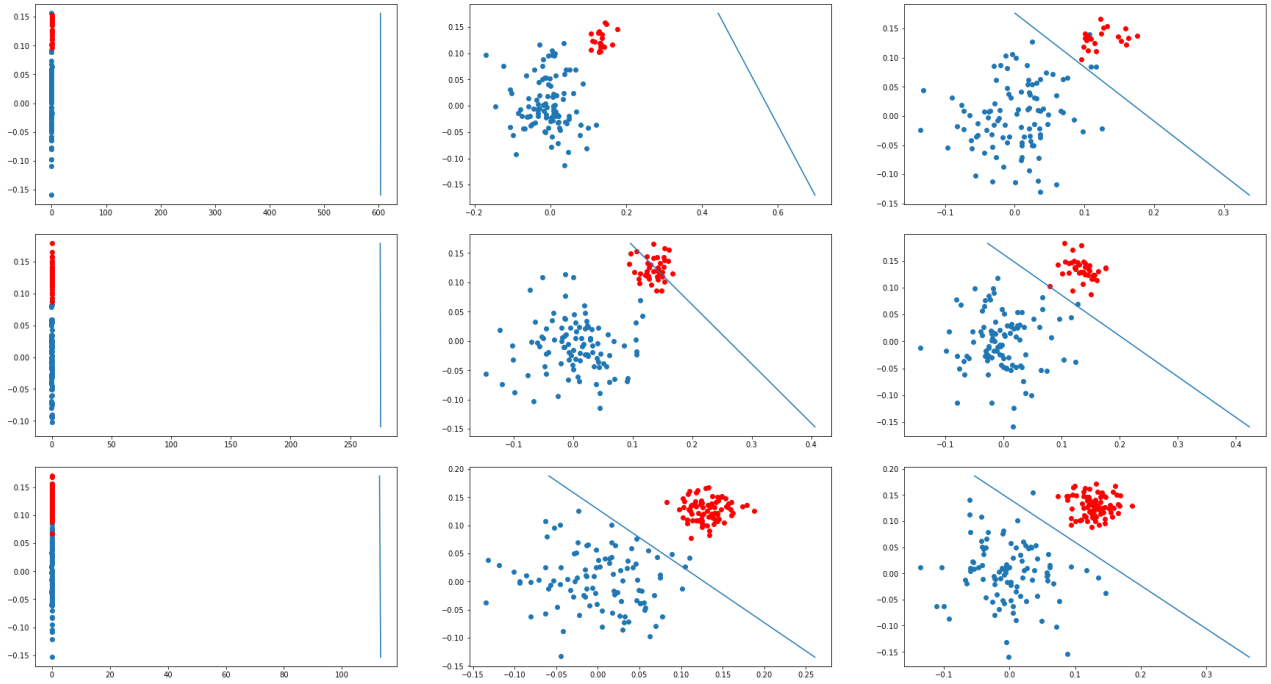
Write in your own words, the observations from the above plots, and
what do you think about the position of the hyper plane

check the optimization problem here https://scikit-learn.org/stable/modules/svm.html#mathematical-formulation

if you can describe your understanding by writing it on a paper and attach the picture, or record a video upload it in assignment.

In [19]:
```
ratios = [(100,2), (100, 20), (100, 40), (100, 80)]

c = [0.001, 1, 100]
for j,i in enumerate(ratios):

    plt.figure(figsize=(30,5))
    for k in range(len(c)):

        X_p=np.random.normal(0,0.05,size=(i[0],2))
        X_n=np.random.normal(0.13,0.02,size=(i[1],2))
        y_p=np.array([1]*i[0]).reshape(-1,1)
        y_n=np.array([0]*i[1]).reshape(-1,1)
        X=np.vstack((X_p,X_n))
        y=np.vstack((y_p,y_n))

        clf = SVC(C = c[k] , kernel='linear')
        clf.fit(X, y)

        plt.subplot(1, 3, k+1)
        plt.scatter(X_p[:,0],X_p[:,1])
        plt.scatter(X_n[:,0],X_n[:,1],color='red')
        draw_line ( clf.coef_.flatten(), clf.intercept_ , X.min(),X.max() )
    plt.show()
```





Observations :

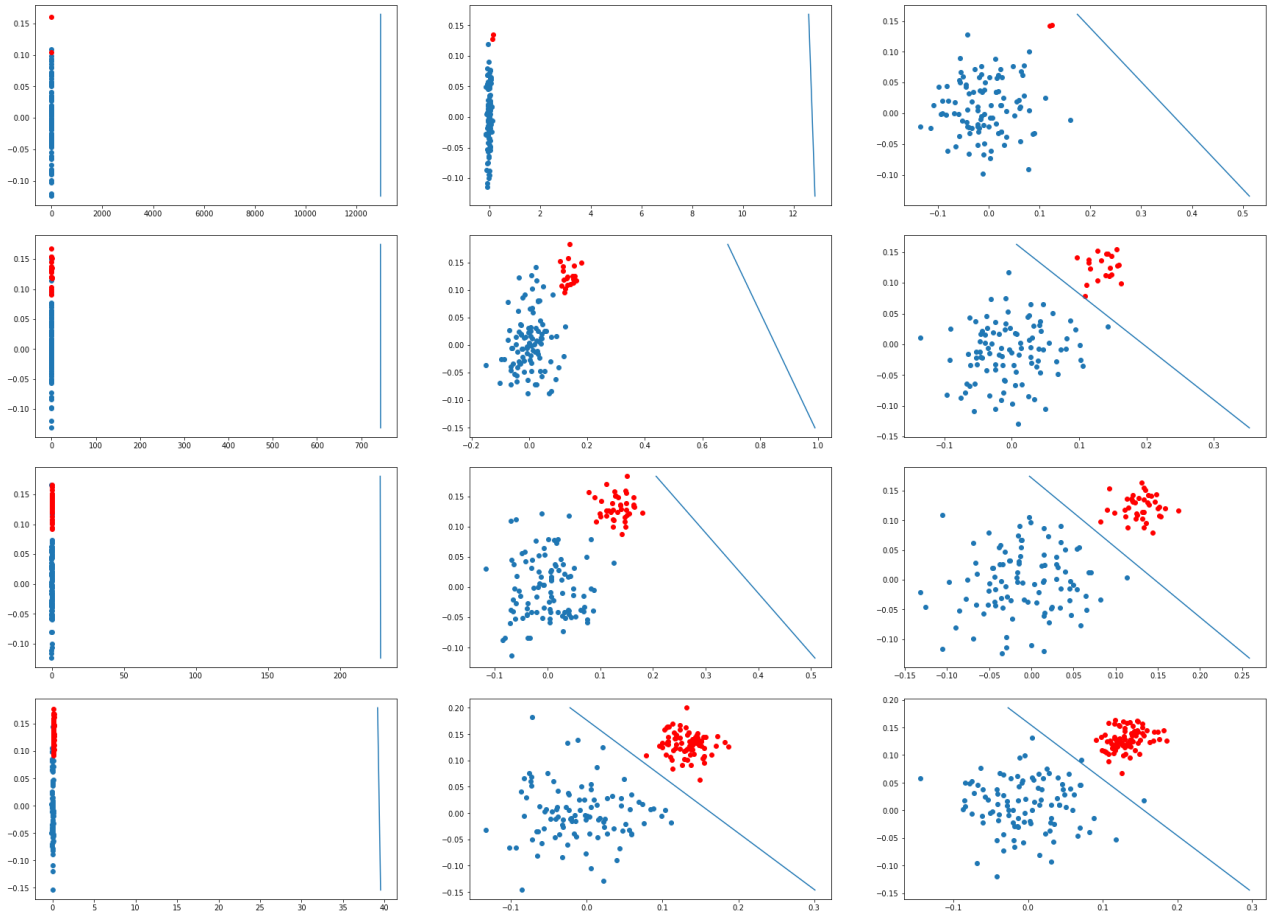
1. In the first Iteration of the highest imbalanced dataset , Only the highest value of c has been able to distinguish between both the classes.
2. Even if the imbalance reduces and the c value is low i.e. 0.001 , it is not able to distinguish the points.
3. Higher the value of c and lower imbalance yields best results.
4. C value of 100 deals best with datasets from highest imbalance to lowest imbalance. It is able to linearly seperate both classes in all scenarios.

Task 2: Applying LR

you will do the same thing what you have done in task 1.1, except instead of SVM you apply logistic regression

In [20]:
```
ratios = [(100,2), (100, 20), (100, 40), (100, 80)]

c = [0.001, 1, 100]
for j,i in enumerate(ratios):

    plt.figure(figsize=(30,5))
    for k in range(len(c)):

        X_p=np.random.normal(0,0.05,size=(i[0],2))
        X_n=np.random.normal(0.13,0.02,size=(i[1],2))
        y_p=np.array([1]*i[0]).reshape(-1,1)
        y_n=np.array([0]*i[1]).reshape(-1,1)
        X=np.vstack((X_p,X_n))
        y=np.vstack((y_p,y_n))

        clf = LogisticRegression(C = c[k] )
```

```
        clf.fit(X, y)

        plt.subplot(1, 3, k+1)
        plt.scatter(X_p[:,0],X_p[:,1])
        plt.scatter(X_n[:,0],X_n[:,1],color='red')
        draw_line ( clf.coef_.flatten(), clf.intercept_ , X.min(),X.max() )
    plt.show()
```



Observations:

1. SVC deals better with high imbalance. In the first iteration where C =100 with highest imbalance , SVC was able to differentiate the points whereas logistic Regression Simply failed.
2. In rest of the cases behaviour of SVC is similar to logistic regression.