In [3]:
```
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
import plotly
from sklearn.preprocessing import StandardScaler, Normalizer
import matplotlib.pyplot as plt
from sklearn.svm import SVC
import warnings
warnings.filterwarnings("ignore")
```

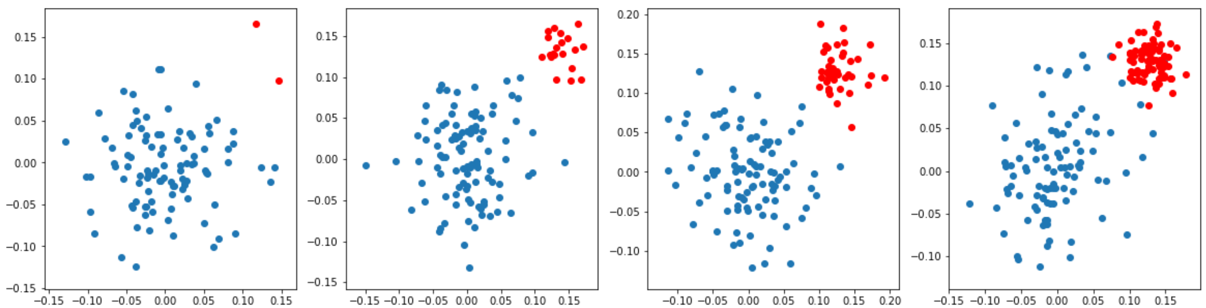
In [4]:
```
def draw_line(coef,intercept, mi, ma):
    # for the separating hyper plane ax+by+c=0, the weights are [a, b]
 and the intercept is c
    # to draw the hyper plane we are creating two points
    # 1. ((b*min-c)/a, min) i.e ax+by+c=0 ==> ax = (-by-c) ==> x = (-by
-c)/a here in place of y we are keeping the minimum value of y
    # 2. ((b*max-c)/a, max) i.e ax+by+c=0 ==> ax = (-by-c) ==> x = (-by
-c)/a here in place of y we are keeping the maximum value of y
    points=np.array([[((-coef[1]*mi - intercept)/coef[0]), mi],[((-coef
[1]*ma - intercept)/coef[0]), ma]])
    plt.plot(points[:,0], points[:,1])
```

What if Data is imabalanced

```
1. As a part of this task you will observe how linear models wor
k 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 se
parable and having class imbalance
```

4. in the first dataset the ration 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

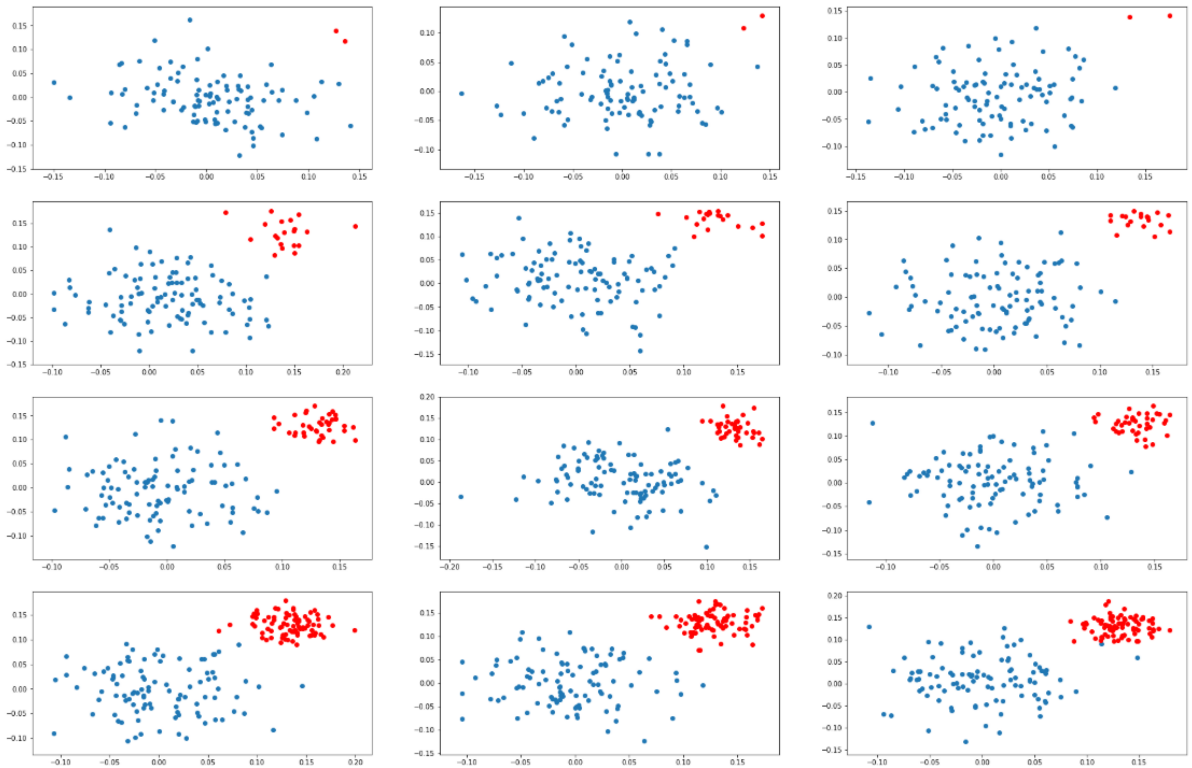
In [5]:
```
# here we are creating 2d imbalanced data points
ratios = [(100,2), (100, 20), (100, 40), (100, 80)]
plt.figure(figsize=(20,5))
for j,i in enumerate(ratios):
    plt.subplot(1, 4, j+1)
    X_p=np.random.normal(0,0.05,size=(i[0],2))
    X_n=np.random.normal(0.13,0.02,size=(i[1],2))
    y_p=np.array([1]*i[0]).reshape(-1,1)
    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](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

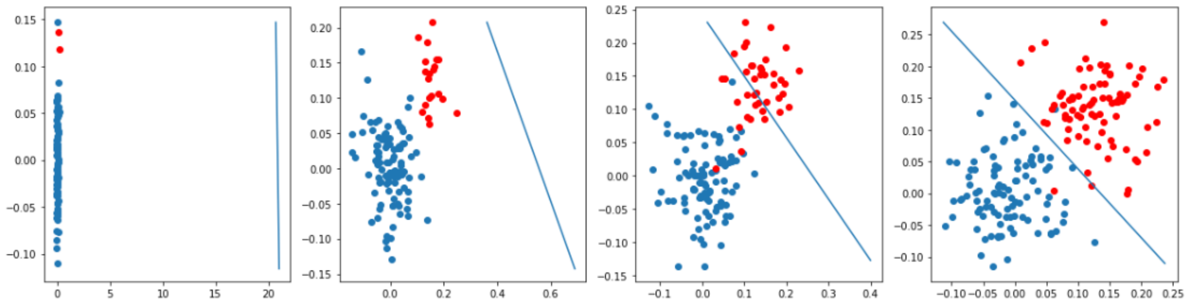
check the optimization problem here https://scikit-learn.org/stable/modules/svm.html#mathematical-formulation

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

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

these are results we got when we are experimenting with one of the model



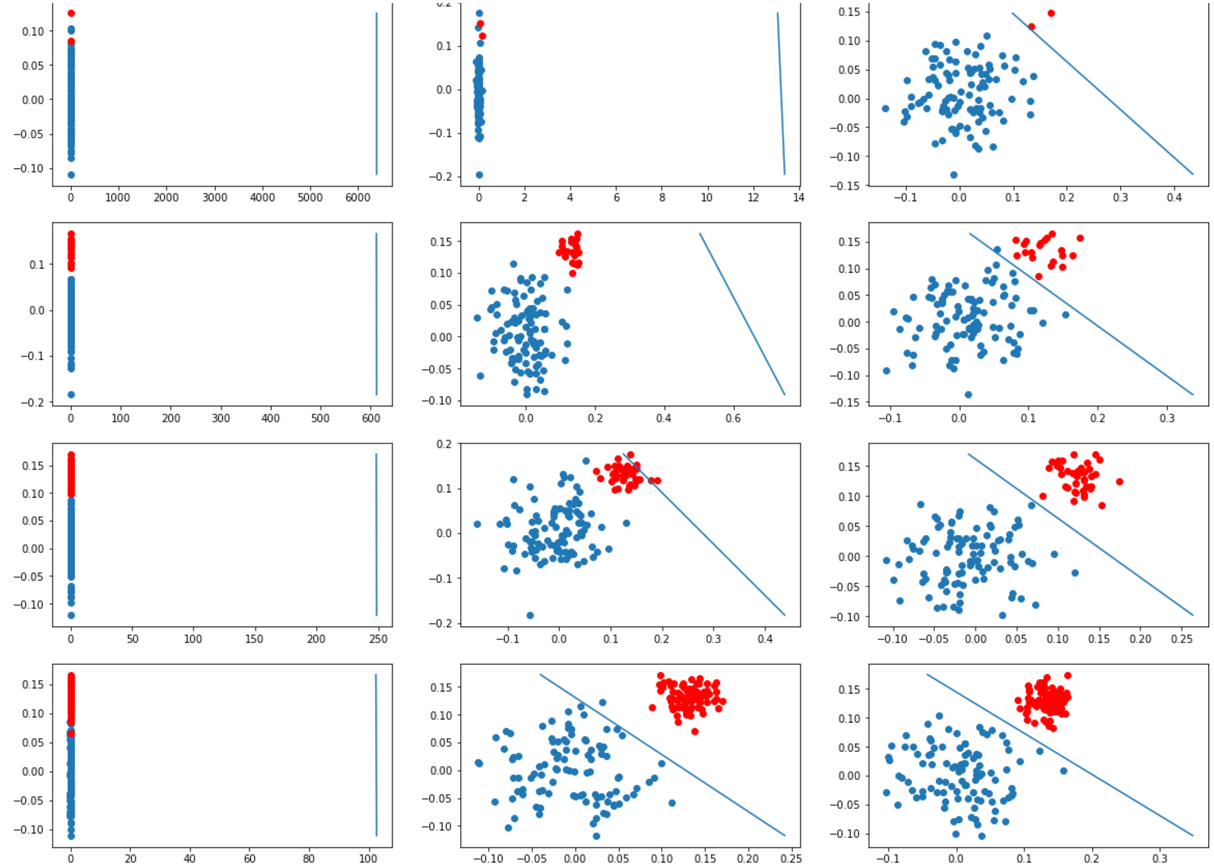
In [0]:
```
#you can start writing code here.
```

Task - 1A

In [8]:
```
from sklearn.svm import SVC
def svm_para(X,y,stren):
    clf = SVC(kernel='linear',C = stren,gamma='auto')
    clf.fit(X,y)
    #print(clf.coef_[0],clf.intercept_)
    return(clf.coef_[0],clf.intercept_)
```

In [10]:
```
#you can start writing code here.
```

```
# here we are creating 2d imbalanced data points
import matplotlib.pyplot as plot
c = [0.001, 1, 100]
ratios = [(100,2), (100, 20), (100, 40), (100, 80)]
k = 0
plt.figure(figsize=(20,15))

for j,i in enumerate(ratios):
    for value  in c:
        plt.subplot(4, 3, k+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))
        coeff, intercept = svm_para(X,y,value)

        draw_line(coeff,intercept,min(X[:,1]),max(X[:,1]))

        plt.scatter(X_p[:,0],X_p[:,1])
        plt.scatter(X_n[:,0],X_n[:,1],color='red')
        k += 1
plt.show()
```

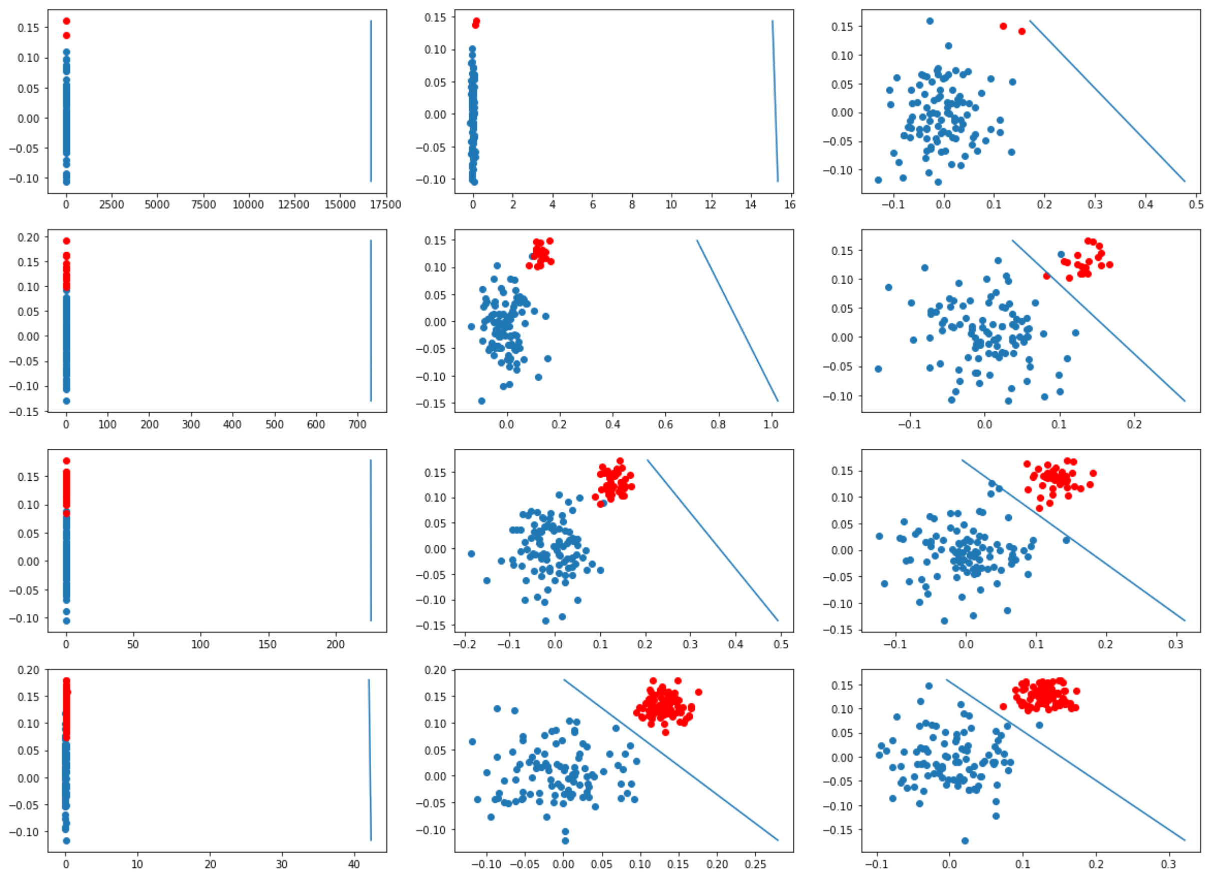


Task - 2

In [11]:
```
from sklearn.linear_model import LogisticRegression
def lr_para(X,y,stren):
    clf = LogisticRegression(C=stren)
    clf.fit(X,y)
    return(clf.coef_[0],clf.intercept_)
```

In [12]:
```
c = [0.001, 1, 100]
ratios = [(100,2), (100, 20), (100, 40), (100, 80)]
```

```
k = 0
plt.figure(figsize=(20,15))

for j,i in enumerate(ratios):
    for value in c:
        plt.subplot(4, 3, k+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))

        coeff, intercept = lr_para(X,y,value)
        draw_line(coeff,intercept,min(X[:,1]),max(X[:,1]))

        plt.scatter(X_p[:,0],X_p[:,1])
        plt.scatter(X_n[:,0],X_n[:,1],color='red')
        k += 1
plt.show()
```



Observation

- when the value of c is very low the training data does not impact the decision boundary majorly. This lead to underfitting
- Even if we have a balance regularization parameter i,e., in this case c = 100 its does not works well with the imbalnced dataset.
- So,the performance is better if we have balance or almost balanced dataset