In [1]:

```
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 [88]:

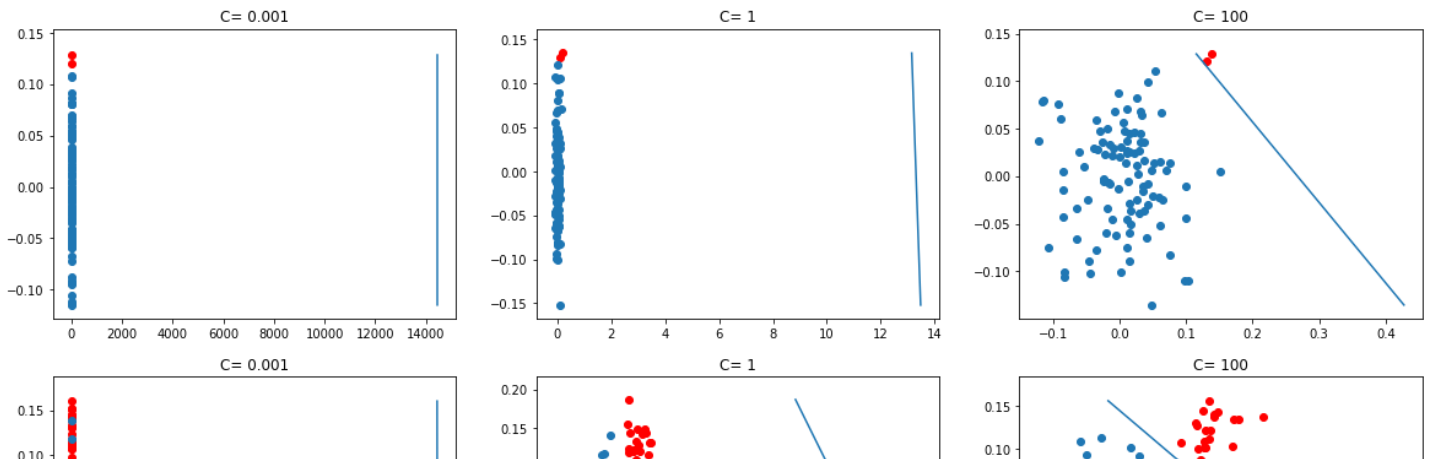
```
def draw_line(coef,intercept, mi, ma):
    # for the separating hyper plane ax+by+c=0, the weights are [a, b] and the intercept
is c
    # to draw the hyper plane we are creating two points
    # 1. ((b*min-c)/a, min) i.e ax+by+c=0 ==> ax = (-by-c) ==> x = (-by-c)/a here in plac
e of y we are keeping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

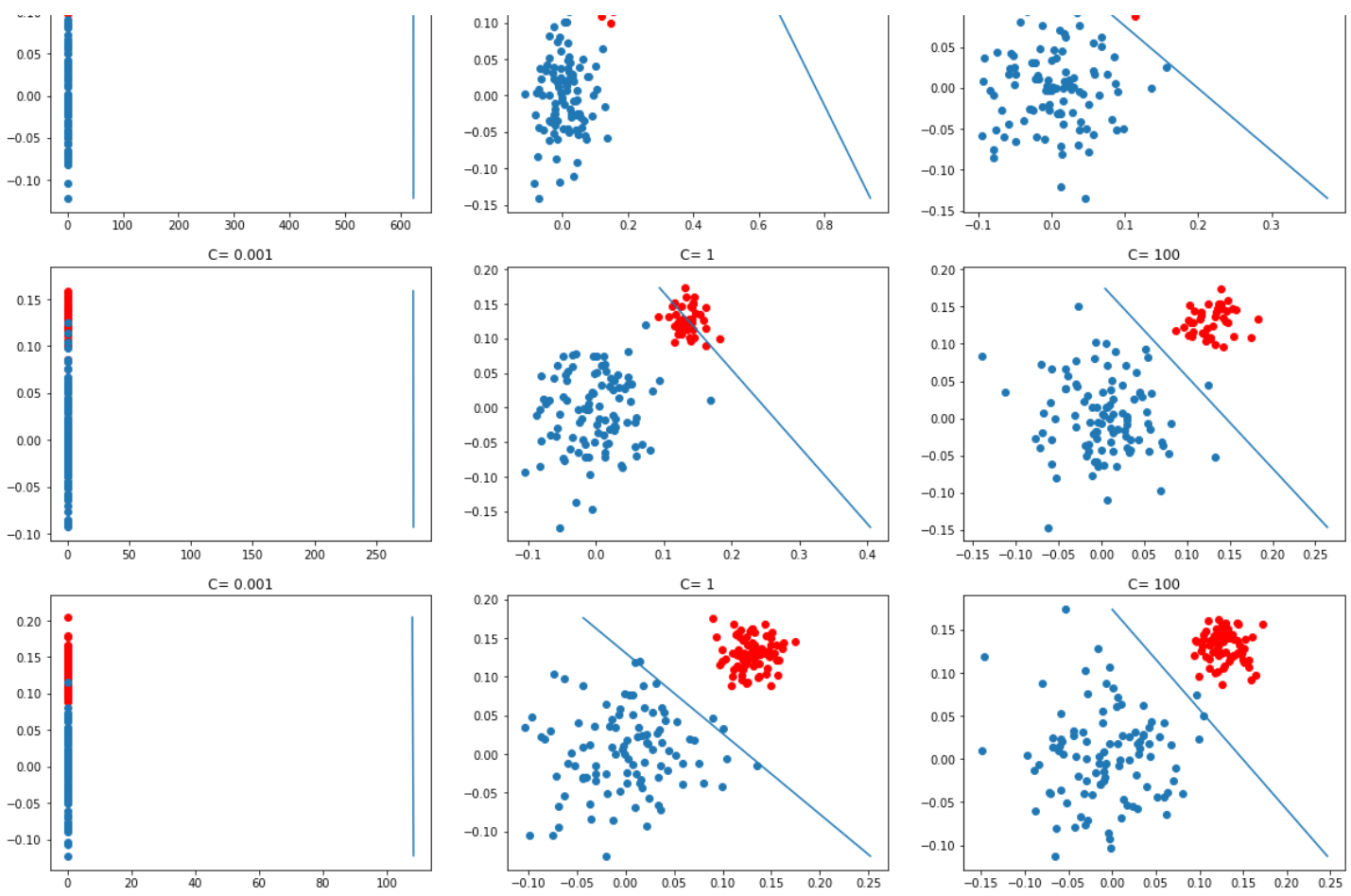
In [118]:

```
ratios = [(100,2), (100, 20), (100, 40), (100,80)]
plt.figure(figsize=(20,20))
v=0
c = [0.001,1,100]

for j,i in enumerate(ratios):
    for k in range(3):
            plt.subplot(4, 3, v+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))
            clf=SVC(kernel='linear',C=c[k])
            clf.fit(X,y)
            plt.scatter(X_n[:,0],X_n[:,1],color='red')
            plt.scatter(X_p[:,0],X_p[:,1])
            plt.title('C= '+ str(c[k]))
            v+=1

            draw_line(clf.coef_[0],clf.intercept_, max(X[:,1]), min(X[:,1]))
plt.show()
```



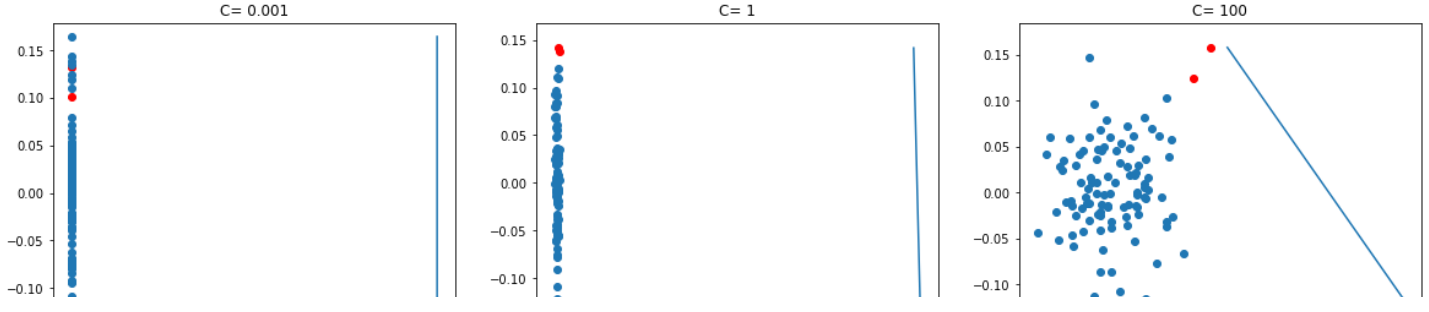


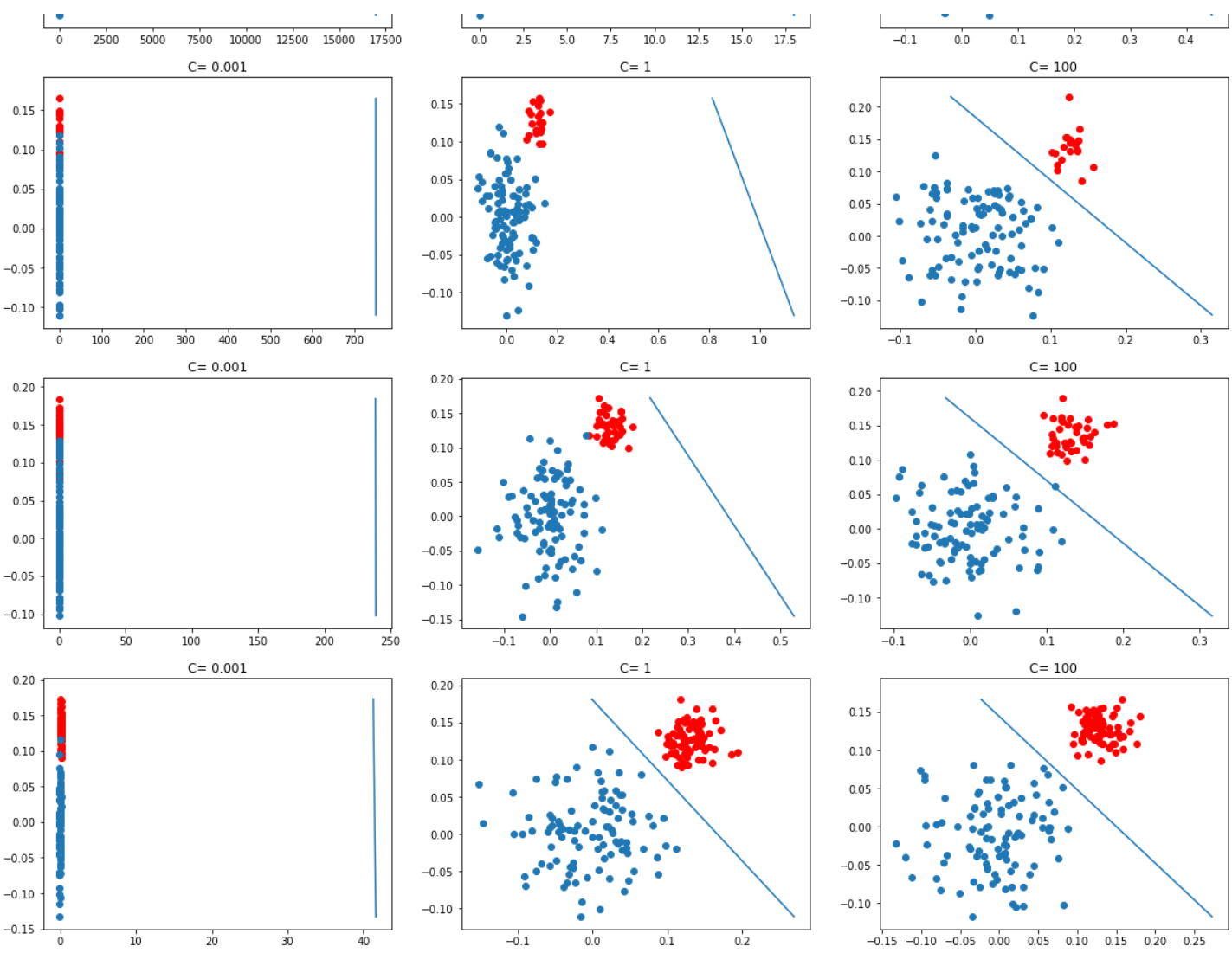
1. **very low c value(0.001) works very bad at imbalanced dataset as well as balanced data we see hyperplane is too far away**
2. **as c value(1) increases hyperplane makes better classification if data is balanced however for imbalanced data it works poorly.**
3. **as c(100) increases vastly, it works fairly good on highly imbalanced data .**

In [123]:

```
plt.figure(figsize=(20,20))
v=0
for j,i in enumerate(ratios):
    for k in range(3):
            plt.subplot(4, 3, v+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))
            lr=LogisticRegression(C=c[k])
            lr.fit(X,y)
            plt.scatter(X_n[:,0],X_n[:,1],color='red')
            plt.scatter(X_p[:,0],X_p[:,1])
            plt.title('C= '+ str(c[k]))
            v+=1

            draw_line(lr.coef_[0],lr.intercept_, max(X[:,1]), min(X[:,1]))
plt.show()
```





1. **very low c value(0.001) is very bad for imbalanced data as well as balanced data.**
2. **as c increases it becomes good for as ratio of len(datasets_1_and_2) becomes more than 0.8**
3. **large c value(100) hyperplane works bad if data is highly imbalanced but works fairly well as imbalanced data becomes less**