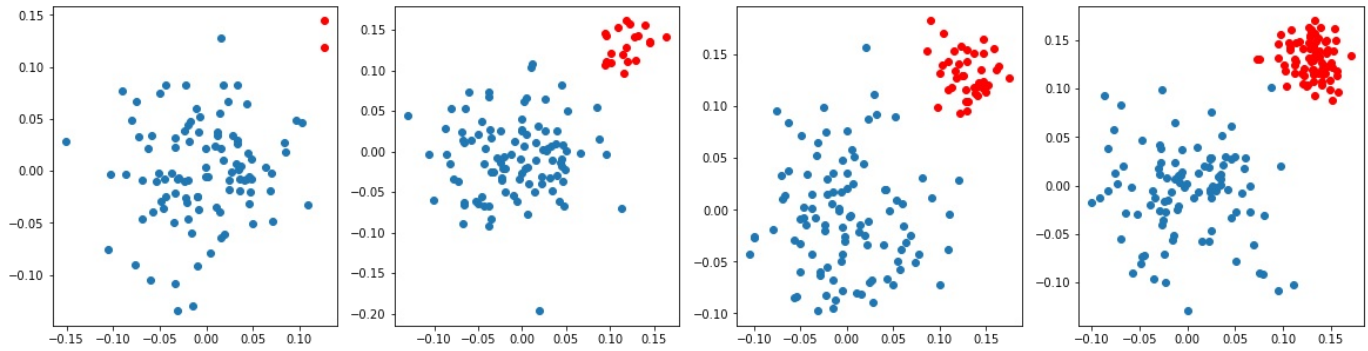
In [30]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import SGDClassifier
from sklearn.linear_model import LogisticRegression
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler, Normalizer
import matplotlib.pyplot as plt
from sklearn.svm import SVC
import warnings
warnings.filterwarnings("ignore")
```

In [90]:
```
def draw_line1(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 th
    # 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 th
    # print(coef.item(0))
    # print(coef.item(1))
    points=np.array([[((-coef.item(1)*mi - intercept)/coef.item(0)), mi],[((-coef.item(1)*ma - intercept)/coef.it
    #print(points[:,1])
    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 [4]:
```
# 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()
```

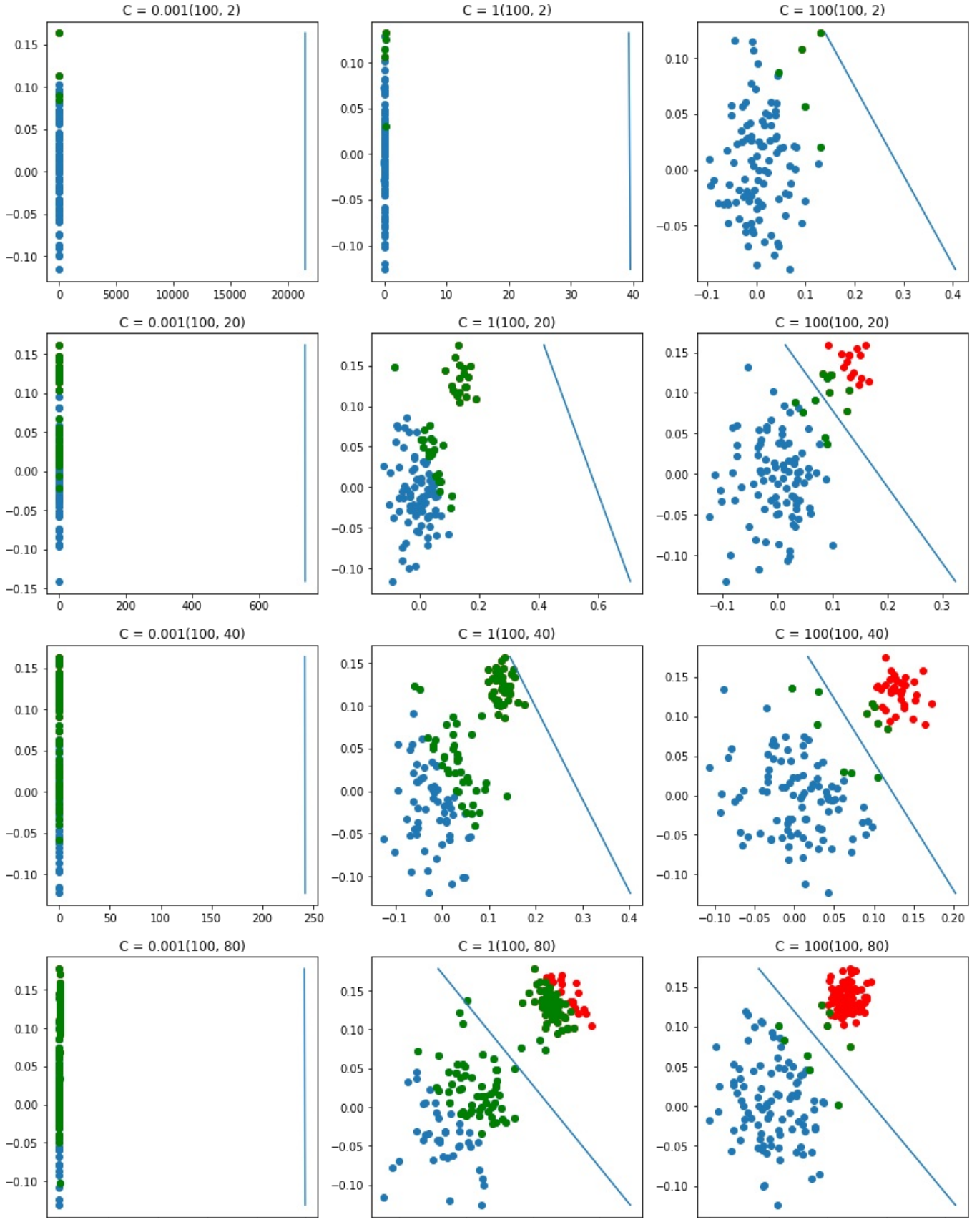


> your task is to apply SVM (sklearn.svm.SVC) and LR (sklearn.linear_model.LogisticRegression) with different regularization strength [0.001, 1, 100]

Task 1: Applying SVM

In [100…
```
#https://towardsdatascience.com/support-vector-machines-imbalanced-data-feb3ecffbb0e
C = [0.001, 1, 100]
plt.figure(figsize=(15, 20))
ratios = [(100, 2), (100, 20), (100, 40), (100,80)]
no = 1
for j,i in enumerate(ratios):
    for c in C:
      SVM = SVC(C = c, kernel = 'linear')
      plt.subplot(4, 3, no)
```

```
        no += 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))
        #print(max(X[:,1]))
        y=np.vstack((y_p,y_n))
        SVM.fit(X, y)
        support_vectors = SVM.support_vectors_
        plt.scatter(X_p[:,0],X_p[:,1])
        plt.scatter(X_n[:,0],X_n[:,1],color='red')
        plt.scatter(support_vectors[:,0], support_vectors[:,1], color = 'green') #ploting support vectors in green
        plt.title("C = " + str(c) + str(i))
        #print(list(SVM.coef_))
        #print(SVM.coef_[0])
        #print(SVM.intercept_)
        draw_line1(SVM.coef_[0], SVM.intercept_, min(X[:, 1]), max(X[:, 1]))
plt.show()
```



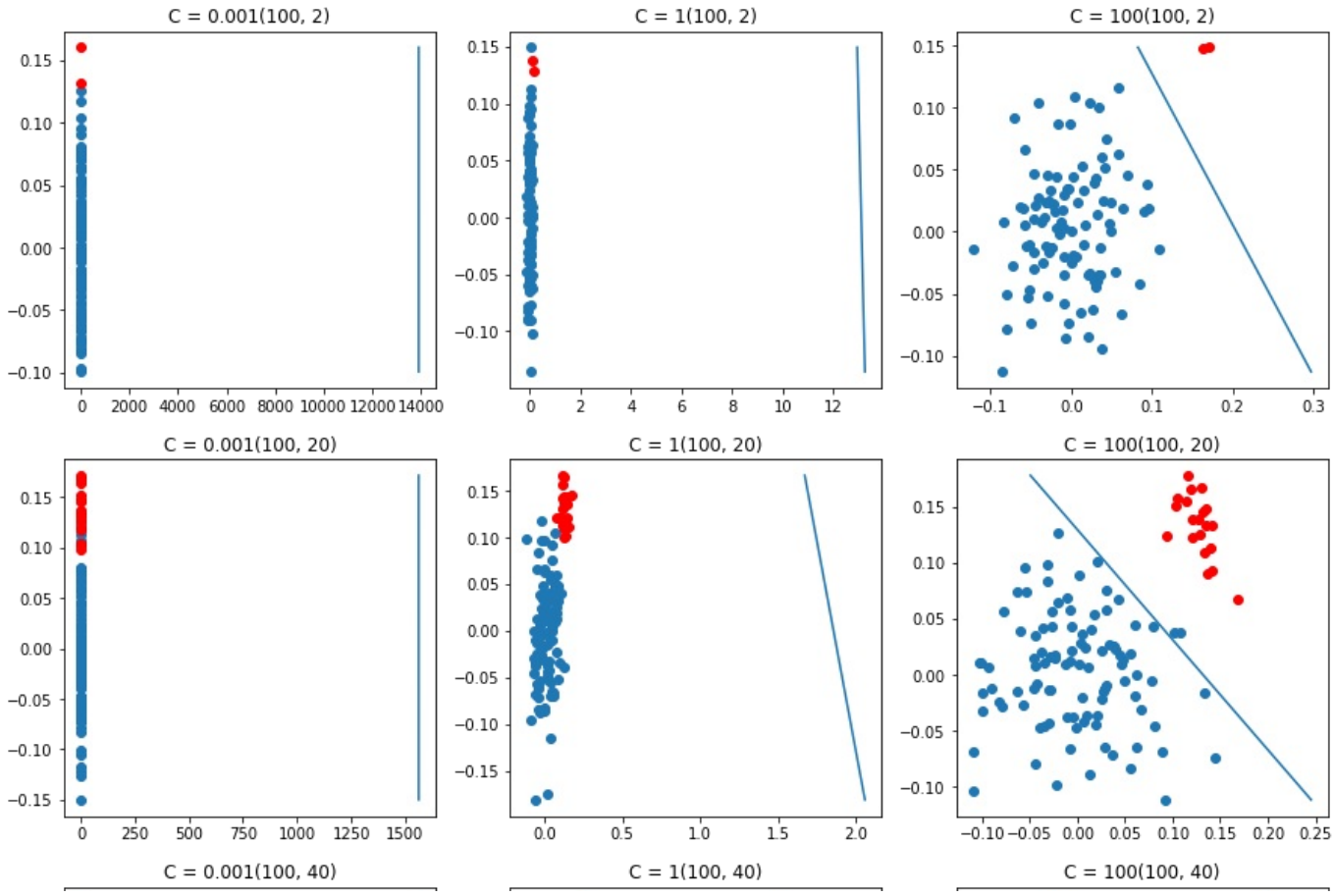
Observations on working of SVM in imbalanced data

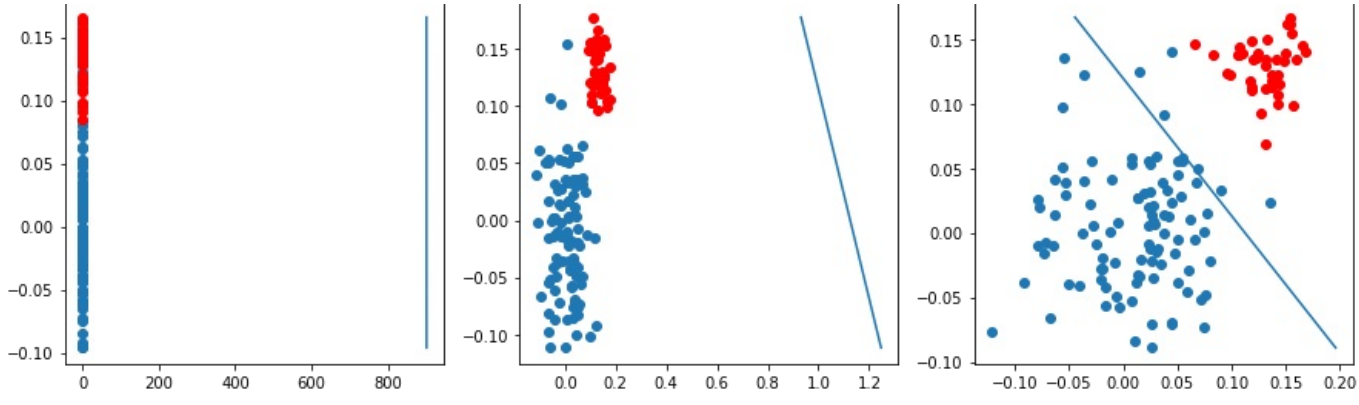
1. Classes are much much imbalanced (100, 2) : In this case hyperplane created by SVM should not classify the classes. one class dominates all the vectors and support vectors. whatever change in a hyperparameter C, model doesnt work well. So our data should not much imbalaced
2. Classes are much imbalanced (100, 20) : In this case hyperplane try to classify the points when C = 100. but other values for C = [0.001, 1], model doesnt classify
3. Classes are imbalanced (100, 40) : In this case for the C = 100, it classify the classes for other two not classified
4. Classes are slightly imbalanced (100, 80) : In this case, for C = [1, 100], model classifies well.
5. Dont use minimum value for hyperplane. Data is imbalanced or not, min value for hyplane not work well in SVM

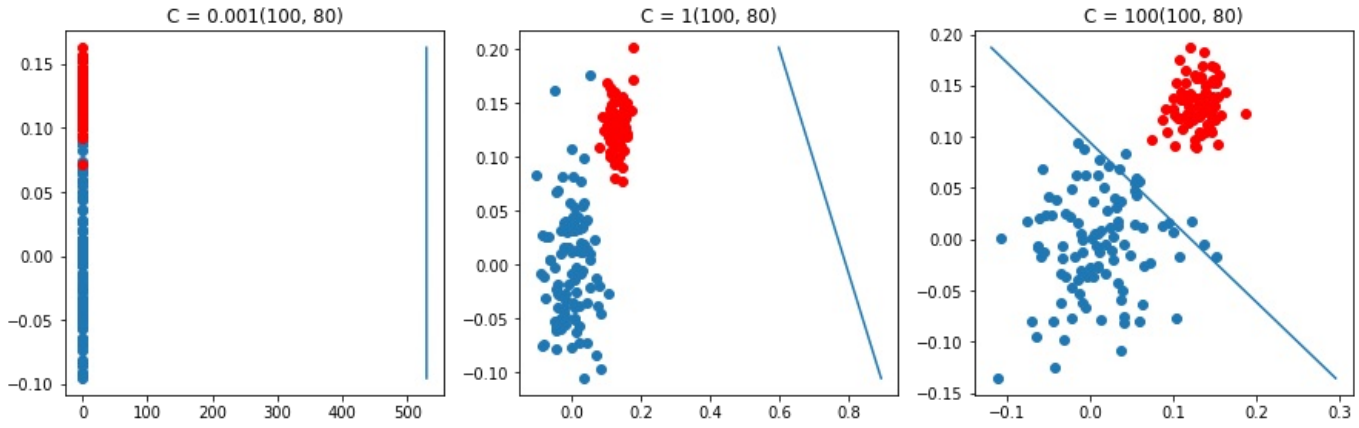
In [ ]:

Task 2: Applying LR

In [99]:
```
#you can start writing code here.
C = [0.001, 1, 100]
plt.figure(figsize=(15, 20))
ratios = [(100, 2), (100, 20), (100, 40), (100,80)]
no = 1
for j,i in enumerate(ratios):
    for c in C:
      Log_reg = LogisticRegression(C = c)
      plt.subplot(4, 3, no)
      no += 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))
      #print(max(X[:,1]))
      y=np.vstack((y_p,y_n))
      Log_reg.fit(X, y)
      plt.scatter(X_p[:,0],X_p[:,1])
      plt.scatter(X_n[:,0],X_n[:,1],color='red')
      plt.title("C = " + str(c) + str(i))
      #print(list(SVM.coef_))
      #print(SVM.coef_[0])
      #print(SVM.intercept_)
      draw_line1(Log_reg.coef_[0], SVM.intercept_, min(X[:, 1]), max(X[:, 1]))
plt.show()
```







Observations on Logistic Regression

All Points mentioned in SVM applicable for logistic regression except

1. in lightly imbalaced

data C = 1(100,80) doesnot noe classify the points

1. Either data is imbalanced C = 100 classifies the points well in logistic regression.

Comparitively SVM is much better than logistic regression by means of misclassification.

In [ ]: