udaylunawat@gmail.com

```
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")
```

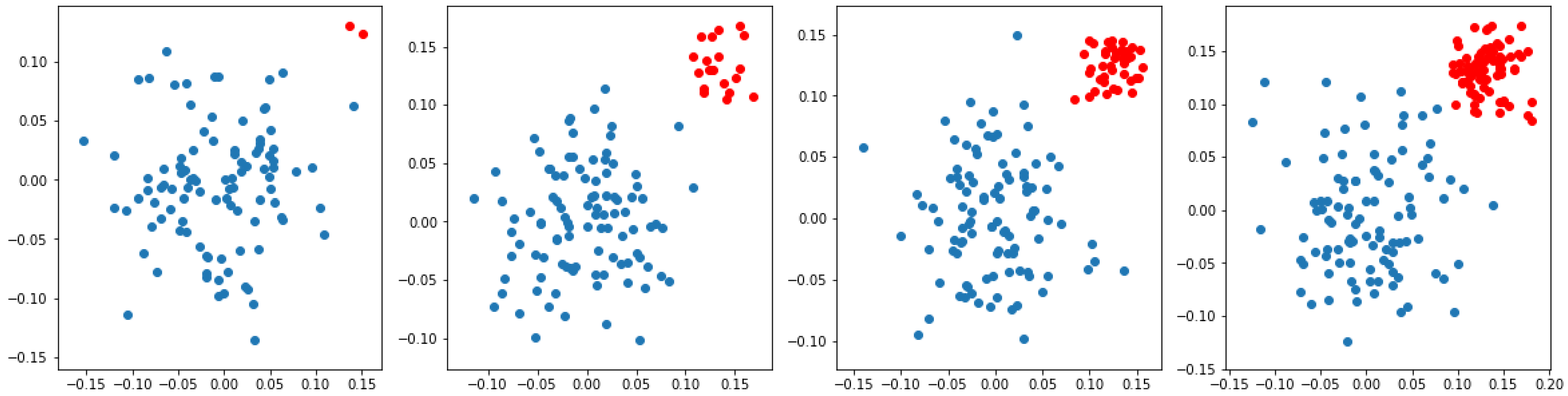
```
def draw_line(coef,intercept, mi, ma):
    # for the separating hyper plane ax+by+c=0, the weights are [a, b] and the intercept is c
    # to draw the hyper plane we are creating two points
    # 1. ((b*min-c)/a, min) i.e ax+by+c=0 ==> ax = (-by-c) ==> x = (-by-c)/a here in place of y we are keeping 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 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 ration between positive and negative is 100 : 2, in the 2nd data its 100:20, in the 3rd data its 100:40
```

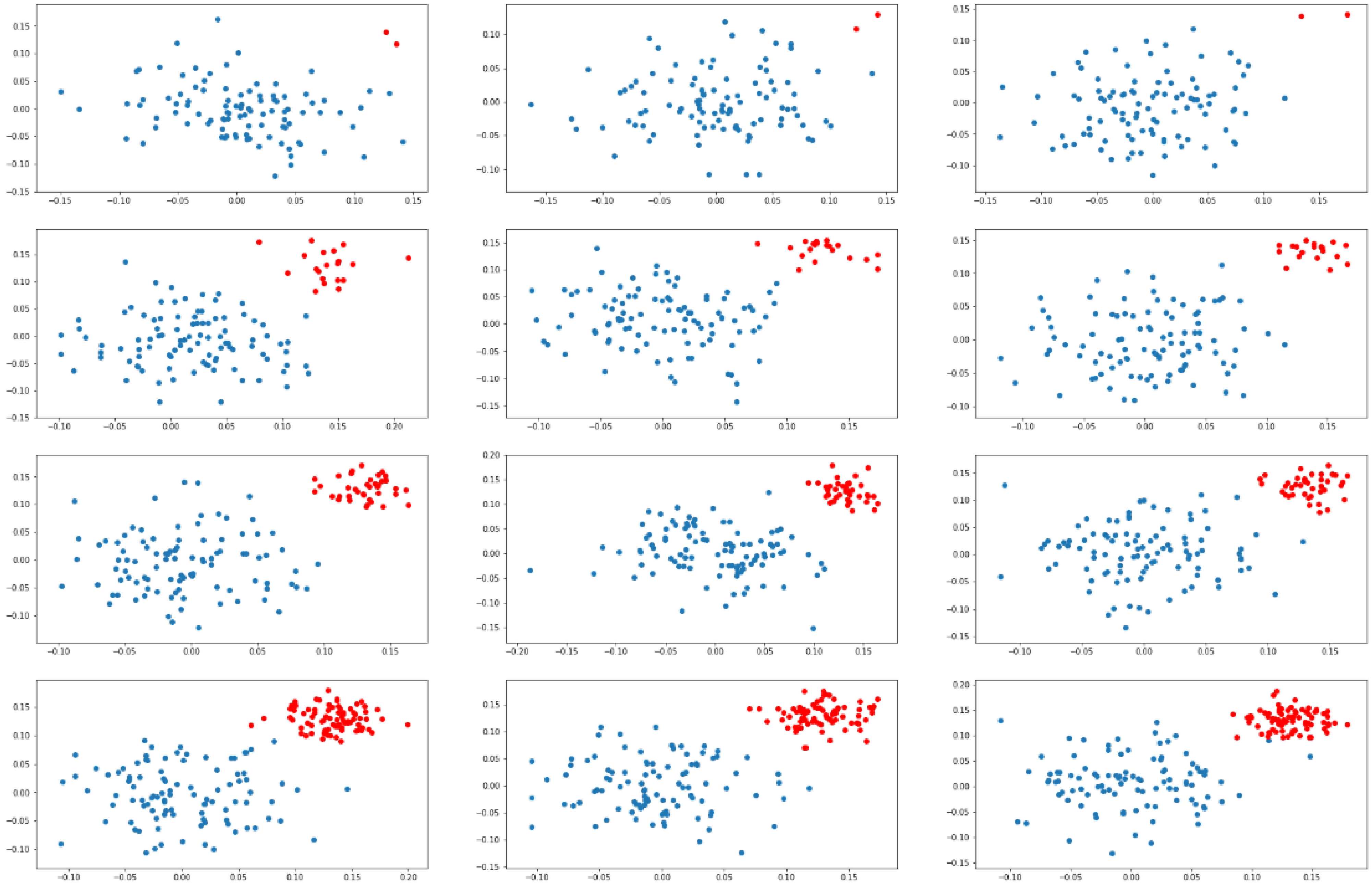
```
# here we are creating 2d imbalanced data points
ratios = [(100,2), (100, 20), (100, 40), (100, 80)]
plt.figure(figsize=(20,5))
for j,i in enumerate(ratios):
    plt.subplot(1, 4, j+1)
    X_p=np.random.normal(0,0.05,size=(i[0],2))
    X_n=np.random.normal(0.13,0.02,size=(i[1],2))
    y_p=np.array([1]*i[0]).reshape(-1,1)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

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 hy

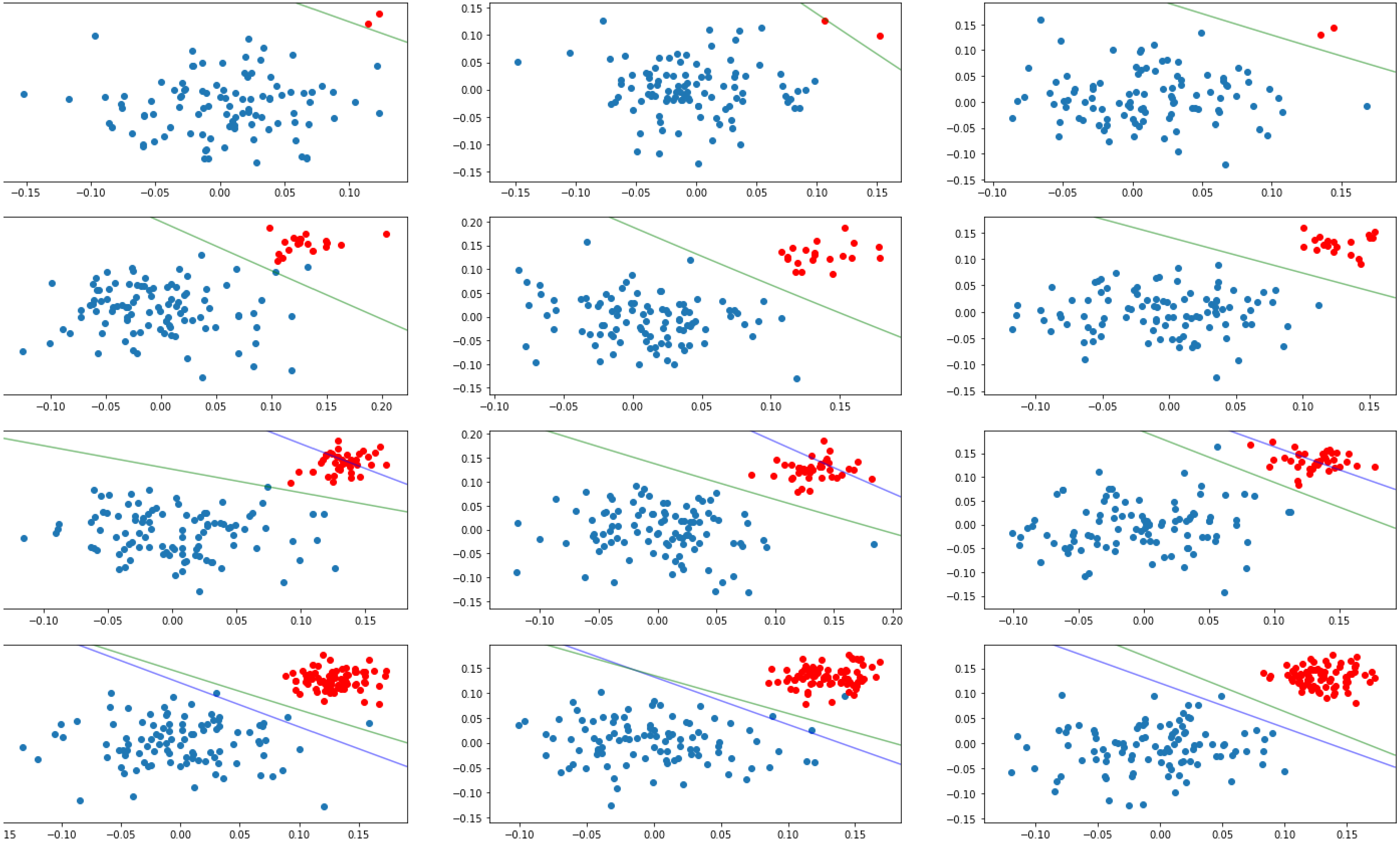
```
check the optimization problem here https://scikit-learn.org/stable/modules/svm.html#mathematical-formulation

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

```
#Source - https://jakevdp.github.io/PythonDataScienceHandbook/05.07-support-vector-machines.html
def plot_svc_decision_function(model,  color, ax=None, plot_support=False):
    """Plot the decision function for a 2D SVC"""
    if ax is None:
        ax = plt.gca()
    xlim = ax.get_xlim()
    ylim = ax.get_ylim()

    # create grid to evaluate model
    x = np.linspace(xlim[0], xlim[1], 30)
    y = np.linspace(ylim[0], ylim[1], 30)
    Y, X = np.meshgrid(y, x)
    xy = np.vstack([X.ravel(), Y.ravel()]).T
    P = model.decision_function(xy).reshape(X.shape)

    # plot decision boundary and margins
    ax.contour(X, Y, P, colors=color,
               levels=[0], alpha=0.5,
               linestyles=['-'])

    ax.set_xlim(xlim)
    ax.set_ylim(ylim)
```

```
#SVC
ratios = [(100,2), (100,2), (100,2), (100, 20), (100, 20), (100, 20), (100, 40), (100, 40), (100, 40), (100, 80), (100, 80), (100, 80)]
plt.figure(figsize=(25, 15))
plt.suptitle('SVM', fontsize =25)

for j,i in enumerate(ratios):
    plt.subplot(4, 3, 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')



    clf1 = SVC(C=0.001)
    clf1.fit(X, y)
    l1 = plot_svc_decision_function(clf1, color = 'red');

```

```
    clf2 = SVC(C=1)
    clf2.fit(X, y)
    l2 = plot_svc_decision_function(clf2, color = 'blue');

    clf3 = SVC(C=100)
    clf3.fit(X, y)
    l3 = plot_svc_decision_function(clf3, color = 'green');


plt.show()
```

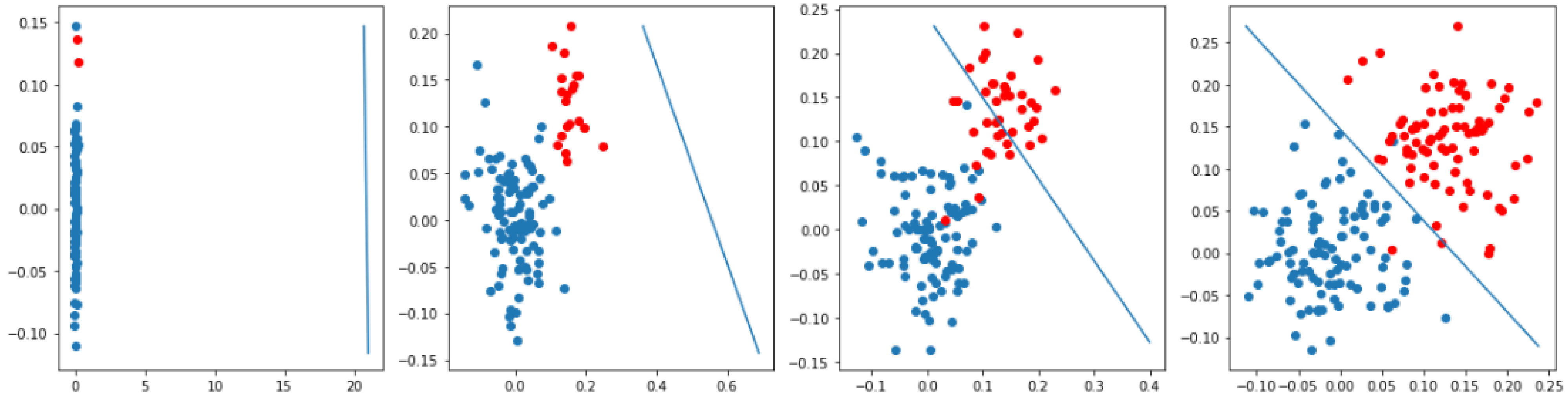
SVM



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
```

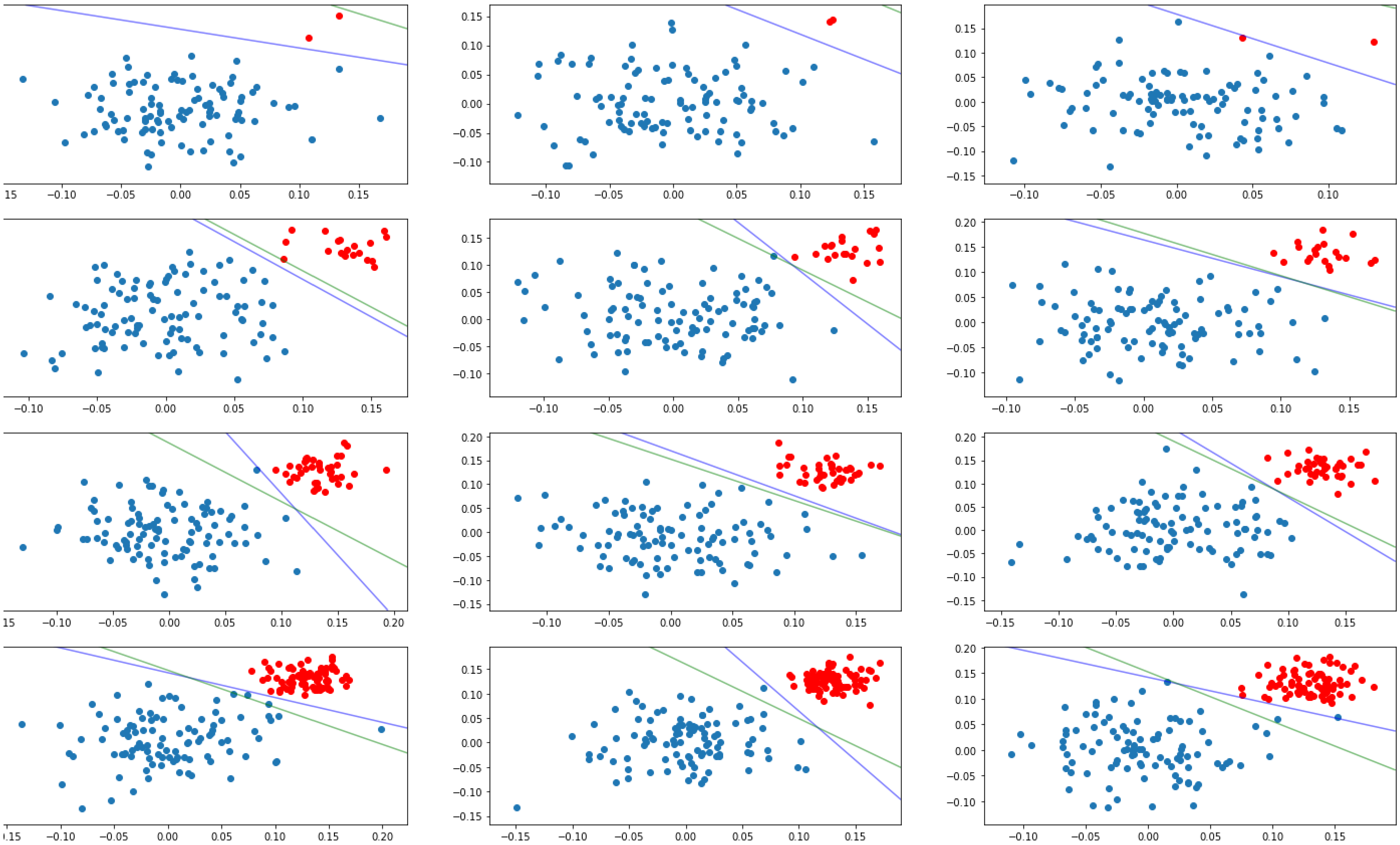
```
these are results we got when we are experimenting with one of the model
```




```python
#Logistic Regression

ratios = [(100,2), (100,2), (100,2), (100, 20), (100, 20), (100, 20), (100, 40), (100, 40), (100, 40), (100, 80), (100, 80), (100, 80)]
plt.figure(figsize=(25,15))
plt.suptitle('Logistic Regression', fontsize =25)
for j,i in enumerate(ratios):
    plt.subplot(4, 3, 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')


    clf1 = LogisticRegression(C=0.001)
    clf1.fit(X, y)
    plot_svc_decision_function(clf1, color = 'red');

    clf2 = LogisticRegression(C=1)
    clf2.fit(X, y)
    plot_svc_decision_function(clf2, color = 'blue');

    clf3 = LogisticRegression(C=100)
```

```
    clf3.fit(X, y)
    plot_svc_decision_function(clf3, color = 'green');

plt.show()
```

Logistic Regression



Observation

- Green Line (C = 100) seems to be the best fitting line.
- Red Line (C = 0.001) seems nowhere to be seen.
- Blue Line (C = 1) seems to be working just OK.
- In case of very Imbalanced data, none of the models seems to work.
- With increase in C, the model also improves.