8A_solution

April 10, 2022

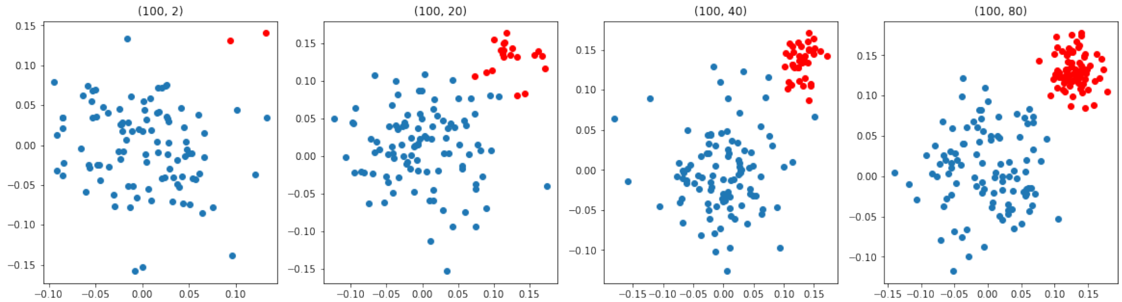
[1]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import SGDClassifier
from sklearn.linear_model import LogisticRegression
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler, Normalizer
import matplotlib.pyplot as plt
from sklearn.svm import SVC
import warnings
warnings.filterwarnings("ignore")
from IPython.display import Image as img
```

[2]:
```
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])
```

1 What if Data is imabalanced

[3]:
```
# 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)
    plt.title(str(i))
    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()
```

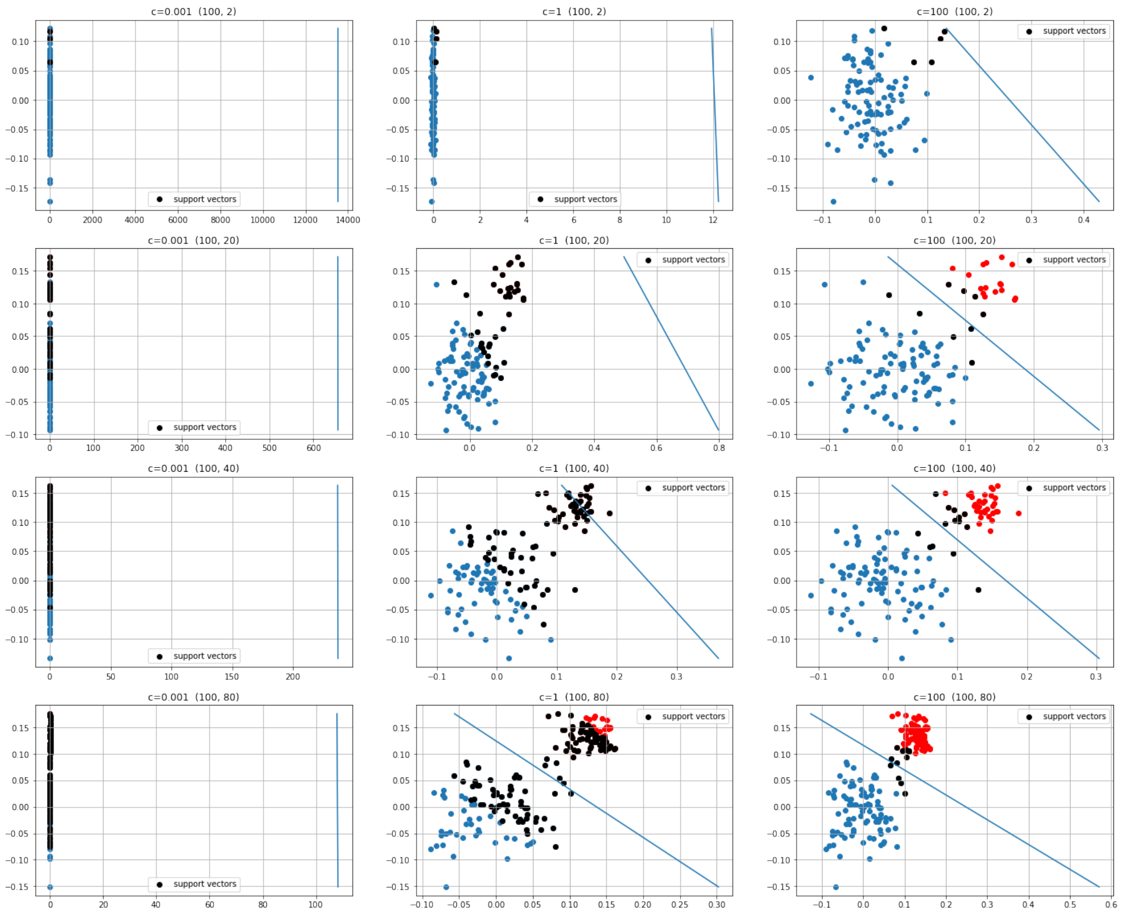


your task is to apply SVM (sklearn.svm.SVC) and LR (sklearn.linear_model.LogisticRegression) with different regularization strength [0.001, 1, 100]

1.1 Task 1: Applying SVM

[4]:
```
np.random.seed(15)
s=0
ratios = [(100,2), (100, 20), (100, 40), (100, 80)]
rate= [0.001, 1, 100]
plt.figure(figsize=(24,20))
for j,i in enumerate(ratios):
    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))
    for k in range(3):
        s=s+1
        plt.subplot(4,3,s)
        plt.title("c="+str(rate[k])+"  "+str(i))
        plt.grid()
        plt.scatter(X_p[:,0],X_p[:,1])
        plt.scatter(X_n[:,0],X_n[:,1],color='red')
        clf=SVC(kernel="linear",C=rate[k],random_state=15)
        clf.fit(X,y)  # GETTING THE INTERCEPT AND WEIGHT COEFFICIENT
        weight=clf.coef_
        intercept=clf.intercept_
```

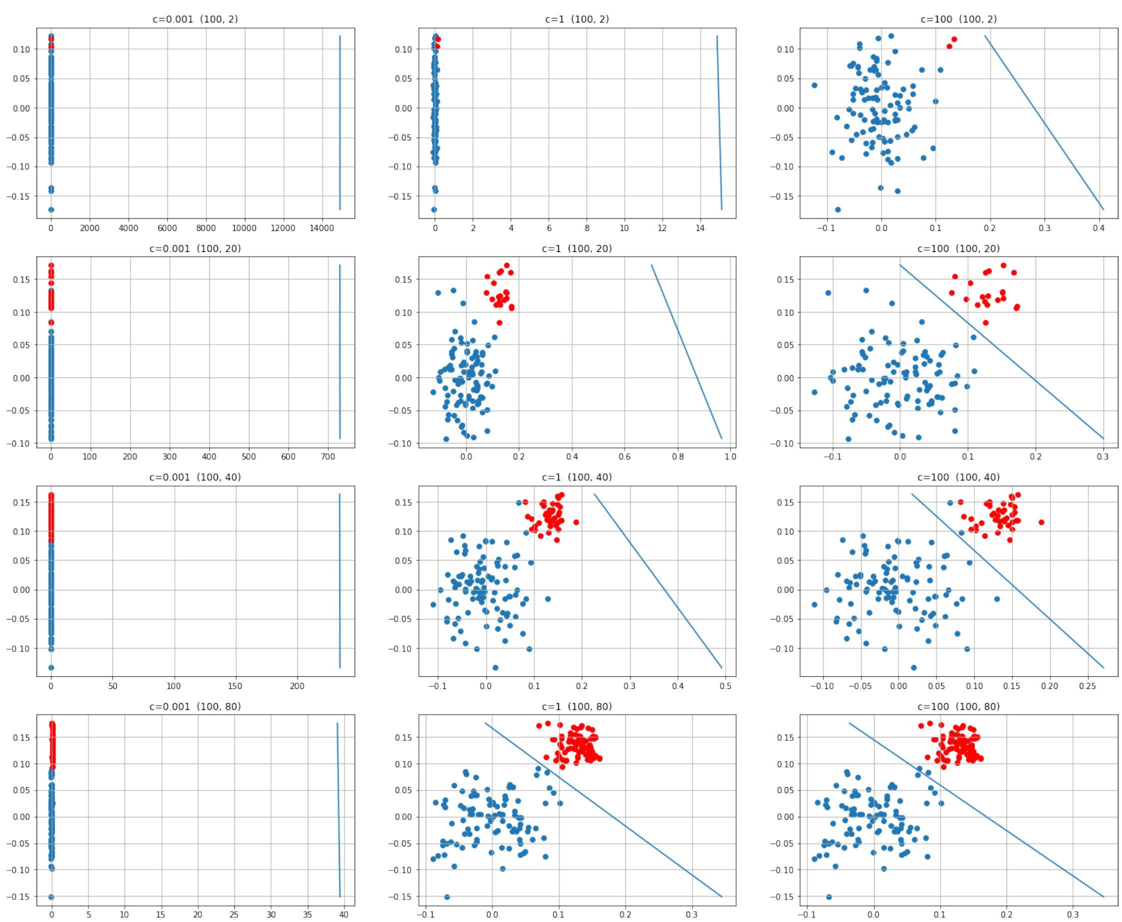
```
        sv=clf.support_vectors_

        plt.scatter(sv[:,0],sv[:,1],color="black",label='support vectors')
        plt.legend()
        mi=min(X[:,1])
        mx=max(X[:,1])
        draw_line(weight[0],intercept,mi,mx)
```



1.2 Task 2: Applying LR

[5]:
```
np.random.seed(15)
s=0
ratios = [(100,2), (100, 20), (100, 40), (100, 80)]
rate= [0.001, 1, 100]
plt.figure(figsize=(24,20))
for j,i in enumerate(ratios):
    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))
for k in range(3):
    s=s+1
    plt.subplot(4,3,s)
    plt.title("c="+str(rate[k])+"  "+str(i))
    plt.grid()
    plt.scatter(X_p[:,0],X_p[:,1])
    plt.scatter(X_n[:,0],X_n[:,1],color='red')
    clf = LogisticRegression(C=rate[k],random_state=15)
    clf.fit(X,y)
    weight=clf.coef_
    intercept=clf.intercept_
    mi=min(X[:,1])
    mx=max(X[:,1])
    draw_line(weight[0],intercept,mi,mx)
```



1.3 Observations:

we know that when c is very less the margin will be big which results in smooth decision curve and the model will underfit, as we can see in the graphs when c=0.001& c=1

when c is very large the margin will be small and there is a chance for model to overfit, but in our case when c=100 it seems it fitted well.

For C = 0.001

- for Dataset 1 (100:2) model is under-fitted
- for Dataset 2 (100:20) model is under-fitted than dataset 1
- for Dataset 3 (100:40) model is more under-fitted than dataset 2
- for Dataset 4 (100:80) model is by far most under-fitted

For C = 1

- for Dataset 1 (100:2) model is under-fitted
- for Dataset 2 (100:20) model is under-fitted than dataset 1
- for Dataset 3 (100:40) model is more under-fitted than dataset 2
- for Dataset 4 (100:80) model is highly under-fitted

For C = 100

- for Dataset 1 (100:2) model is slightly under-fitted, but better than previous c =0.01 and c=1
- for Dataset 2 (100:20) model is fitted well
- for Dataset 3 (100:40) model is very slightly over-fitted
- for Dataset 4 (100:80) model is fitted well