Week 4 Santander case study

In [1]:
```
import pandas as pd
import numpy as np
from sklearn import preprocessing
import seaborn as sns
```

In [2]:
```
trn = pd.read_csv("C:/Users/harsh/Documents/Data Science Course/ML/week 4/trai
n.csv")
#Importing training data
```

In [3]:
```
tst = pd.read_csv("C:/Users/harsh/Documents/Data Science Course/ML/week 4/tes
t.csv")
```

Using the preprocessing feature to normalize and clean and transform data for better estimation later on throughout the code. Initializing the data sets

Observing the structure of the training data

Counting the number of 0s and 1s

Observing the correlation of each class with one another

I have used the var_2 and var_6 columns randomly to train the data with target column.

Also, later on I have tested the trained model with var_1 and var_5

Observing the 0s and 1s of target with a correlation plot of var_2 and var_6. Upon observing, the 1s are prominent for higher values of var_2 and var_6.

Using the minmax, fit, transform, scaler to normalize and clean data.

In [4]:
```
trn.head()
#Overview of the data showing 5 rows and 202 columns
```

Out[4]:

	ID_code	target	var_0	var_1	var_2	var_3	var_4	var_5	var_6	var_7	...	var_
0	train_0	0	8.9255	-6.7863	11.9081	5.0930	11.4607	-9.2834	5.1187	18.6266	...	4.4
1	train_1	0	11.5006	-4.1473	13.8588	5.3890	12.3622	7.0433	5.6208	16.5338	...	7.6
2	train_2	0	8.6093	-2.7457	12.0805	7.8928	10.5825	-9.0837	6.9427	14.6155	...	2.9
3	train_3	0	11.0604	-2.1518	8.9522	7.1957	12.5846	-1.8361	5.8428	14.9250	...	4.4
4	train_4	0	9.8369	-1.4834	12.8746	6.6375	12.2772	2.4486	5.9405	19.2514	...	-1.4

5 rows × 202 columns

In [5]:
```
trn['target'].value_counts()
#Counting the number of 0s and 1s in target class.
```

Out[5]:
```
0    179902
1     20098
Name: target, dtype: int64
```

In [6]:
```
minmax=preprocessing.MinMaxScaler(feature_range=(1,5))
#Normalizing the data using MinMaxScaler
```

In [7]:
```
trn.columns
#Printing the classes in the training data set
```

Out[7]:
```
Index(['ID_code', 'target', 'var_0', 'var_1', 'var_2', 'var_3', 'var_4',
       'var_5', 'var_6', 'var_7',
       ...
       'var_190', 'var_191', 'var_192', 'var_193', 'var_194', 'var_195',
       'var_196', 'var_197', 'var_198', 'var_199'],
      dtype='object', length=202)
```

In [8]:
```
trn.corr()
#Printing the correlation with each and every class of the data
```

Out[8]:

	target	var_0	var_1	var_2	var_3	var_4	var_5	var_6	
target	1.000000	0.052390	0.050343	0.055870	0.011055	0.010915	0.030979	0.066731	-
var_0	0.052390	1.000000	-0.000544	0.006573	0.003801	0.001326	0.003046	0.006983	
var_1	0.050343	-0.000544	1.000000	0.003980	0.000010	0.000303	-0.000902	0.003258	
var_2	0.055870	0.006573	0.003980	1.000000	0.001001	0.000723	0.001569	0.000883	-
var_3	0.011055	0.003801	0.000010	0.001001	1.000000	-0.000322	0.003253	-0.000774	
...	...	...	...	...	...	...	...	...	
var_195	0.028285	0.002073	-0.000785	-0.001070	0.001206	0.003706	-0.001274	0.001244	
var_196	0.023608	0.004386	-0.000377	0.003952	-0.002800	0.000513	0.002880	0.005378	
var_197	-0.035303	-0.000753	-0.004157	0.001078	0.001164	-0.000046	-0.000535	-0.003565	
var_198	-0.053000	-0.005776	-0.004861	-0.000877	-0.001651	-0.001821	-0.000953	-0.003025	
var_199	0.025434	0.003850	0.002287	0.003855	0.000506	-0.000786	0.002767	0.006096	-

201 rows × 201 columns

Choosing two classes randomly to train the model. I chose var_2 and var_6

In [9]:
```
X=trn.iloc[:,[4,8]]
#Pulling data for var_2 and randomly var_6 and training the svm on this data
```

In [10]:
```
X.head()
```

Out[10]:

	var_2	var_6
0	11.9081	5.1187
1	13.8588	5.6208
2	12.0805	6.9427
3	8.9522	5.8428
4	12.8746	5.9405

In [11]:
```
X_test=tst.iloc[:,[2,6]]
#Testing the data with var_1,var_5
```

In [12]:
```
X_test.head()
```

Out[12]:

	var_1	var_5
0	7.7798	-2.3805
1	1.2543	-4.0117
2	-10.3581	9.8052
3	-1.3222	3.1744
4	-0.1327	-8.5848

In [13]:
```
Y=trn.iloc[:,1]
#Using target as classsiifer
```

In [14]:
```
Y.head()
```

Out[14]:
```
0    0
1    0
2    0
3    0
4    0
Name: target, dtype: int64
```

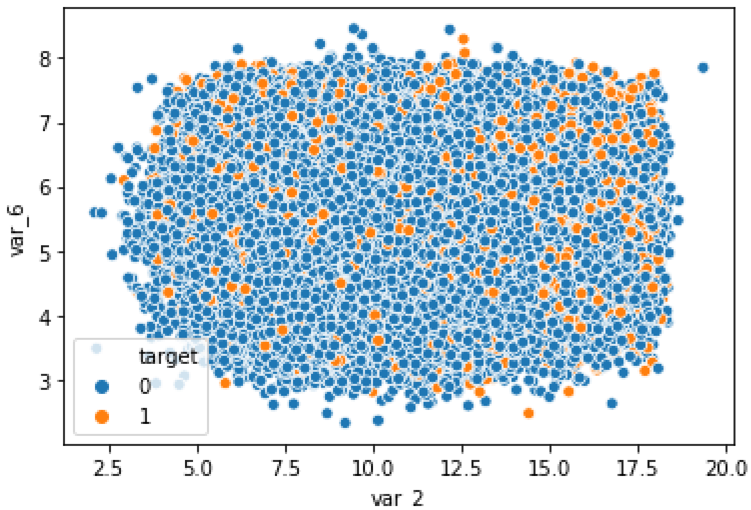
In [15]:
```
Y_test=tst.iloc[:,2]
```

In [16]:
```
Y_test.head()
```

Out[16]:
```
0     7.7798
1     1.2543
2   -10.3581
3    -1.3222
4    -0.1327
Name: var_1, dtype: float64
```

In [17]:
```
sns.scatterplot(x='var_2', y='var_6', hue = 'target', data=trn)
#Visualising the target classifier between two variables #This gives a rough i
dea on how many zeroes are theres.
#1s are deciding factor. signicant 1s after value 13 for var_2. significant 1s
after value 6 for var_2
```

Out[17]: <matplotlib.axes._subplots.AxesSubplot at 0x1a7450e5d88>



Normalizing and cleaning the data

In [18]:
```
minmax.fit(X).transform(X)
#Normalizing the data
```

Out[18]:
```
array([[3.27223412, 2.81719232],
       [3.72494039, 3.14641663],
       [3.31224363, 4.01317946],
       ...,
       [2.9483984 , 2.44790506],
       [3.66729327, 3.42646384],
       [3.32101602, 2.92551308]])
```

Now, fitting the SVM Model on the given data

In [19]:
```
from sklearn.svm import SVC
#Importing SVC from python libraries
```

In [20]:
```
svc = SVC(kernel='rbf', probability=True)
#Introducing first hyperparameter kernel 'rbf'
```

In [21]:
```
svc.fit(X,Y)
#Fitting the SVC on X(x_train) and Y(y_train)
```

Out[21]:
```
SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='scale', kernel='rbf',
    max_iter=-1, probability=True, random_state=None, shrinking=True, tol=0.0
01,
    verbose=False)
```

In [22]:
```
y_pred=svc.predict(X_test)
#Testing the using test data by predicting
```

In [23]:
```
y_pred
#Displaying the prediction
```

Out[23]:
```
array([0, 0, 0, ..., 0, 0, 0], dtype=int64)
```

In [24]:
```
from sklearn.metrics import confusion_matrix,accuracy_score,classification_rep
ort
#Importing the metrics required to discuss the efficacy of the SVM algorithm
```

In [25]:
```
accuracy_score(y_pred, Y, normalize=False)
#Priting the accuracy score
```

Out[25]:
```
179902
```

In [26]:
```
print(confusion_matrix(y_pred,Y))
#Printing the confusion matrix
```

```
[[179902  20098]
 [     0      0]]
```

In [27]:
```
y_train_pred=svc.predict(X)
#predicting using the train data
```

In [28]:
```
y_train_pred #Displaying the prediction
```

Out[28]:
```
array([0, 0, 0, ..., 0, 0, 0], dtype=int64)
```

In [29]:
```
print(confusion_matrix(y_train_pred, Y))
#Printing the confusion matrix
```

```
[[179902  20098]
 [     0      0]]
```

In [30]:
```
print("\nAccuracy Score: %f" % (accuracy_score(Y,y_pred) * 100))
#Printing the accuracy score, Higher the accuracy the better the model..
```

```
Accuracy Score: 89.951000
```

In [31]:
```
print(classification_report(Y,y_pred))
#Printing the classification report. We can see the precison and recall values
for both the target classifiers.
#0s have better metrics all over.
```

```
              precision    recall  f1-score   support

           0       0.90      1.00      0.95    179902
           1       0.00      0.00      0.00     20098

    accuracy                           0.90    200000
   macro avg       0.45      0.50      0.47    200000
weighted avg       0.81      0.90      0.85    200000

C:\Users\harsh\anaconda3\lib\site-packages\sklearn\metrics\_classification.p
y:1272: UndefinedMetricWarning: Precision and F-score are ill-defined and bei
ng set to 0.0 in labels with no predicted samples. Use `zero_division` parame
ter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

The Accuracy score turned out to be 89.95 and the modle trained only for 0s of target classifier. 1s are ignored.

In [34]:
```
from sklearn.model_selection import GridSearchCV
#Using grid searchcv to compare the different hyperparameters
```

In [35]:
```
param_grid = {'C':[0.1,1,10], 'gamma': [1,0.1,0.01], 'kernel':['rbf']}
#Intialising the hyperparameters by specifying range for C and gamma with rbf
 kernel
```

In [36]:
```
grid = GridSearchCV(SVC(), param_grid, verbose = 3, refit=True,cv=3)
grid.fit(X,Y)
#Executing gridsearch with the specified classfiers while applying SVC
```

```
Fitting 3 folds for each of 9 candidates, totalling 27 fits
[CV] C=0.1, gamma=1, kernel=rbf ......................................

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent worke
rs.

[CV] .......... C=0.1, gamma=1, kernel=rbf, score=0.900, total= 2.9min
[CV] C=0.1, gamma=1, kernel=rbf ......................................

[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:  2.9min remaining:    0.
0s

[CV] .......... C=0.1, gamma=1, kernel=rbf, score=0.900, total= 1.9min
[CV] C=0.1, gamma=1, kernel=rbf ......................................

[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:  4.8min remaining:    0.
0s
```

```
[CV] .......... C=0.1, gamma=1, kernel=rbf, score=0.900, total= 1.8min
[CV] C=0.1, gamma=0.1, kernel=rbf ....................................
[CV] ........ C=0.1, gamma=0.1, kernel=rbf, score=0.900, total= 1.7min
[CV] C=0.1, gamma=0.1, kernel=rbf ....................................
[CV] ........ C=0.1, gamma=0.1, kernel=rbf, score=0.900, total= 1.7min
[CV] C=0.1, gamma=0.1, kernel=rbf ....................................
[CV] ........ C=0.1, gamma=0.1, kernel=rbf, score=0.900, total= 1.9min
[CV] C=0.1, gamma=0.01, kernel=rbf ...................................
[CV] ....... C=0.1, gamma=0.01, kernel=rbf, score=0.900, total= 1.9min
[CV] C=0.1, gamma=0.01, kernel=rbf ...................................
[CV] ....... C=0.1, gamma=0.01, kernel=rbf, score=0.900, total= 1.8min
[CV] C=0.1, gamma=0.01, kernel=rbf ...................................
[CV] ....... C=0.1, gamma=0.01, kernel=rbf, score=0.900, total= 1.7min
[CV] C=1, gamma=1, kernel=rbf ........................................
[CV] ............ C=1, gamma=1, kernel=rbf, score=0.900, total= 3.4min
[CV] C=1, gamma=1, kernel=rbf ........................................
[CV] ............ C=1, gamma=1, kernel=rbf, score=0.900, total= 3.3min
[CV] C=1, gamma=1, kernel=rbf ........................................
[CV] ............ C=1, gamma=1, kernel=rbf, score=0.900, total= 2.7min
[CV] C=1, gamma=0.1, kernel=rbf ......................................
[CV] .......... C=1, gamma=0.1, kernel=rbf, score=0.900, total= 1.8min
[CV] C=1, gamma=0.1, kernel=rbf ......................................
[CV] .......... C=1, gamma=0.1, kernel=rbf, score=0.900, total= 2.3min
[CV] C=1, gamma=0.1, kernel=rbf ......................................
[CV] .......... C=1, gamma=0.1, kernel=rbf, score=0.900, total= 1.7min
[CV] C=1, gamma=0.01, kernel=rbf .....................................
[CV] ......... C=1, gamma=0.01, kernel=rbf, score=0.900, total= 1.7min
[CV] C=1, gamma=0.01, kernel=rbf .....................................
[CV] ......... C=1, gamma=0.01, kernel=rbf, score=0.900, total= 1.7min
[CV] C=1, gamma=0.01, kernel=rbf .....................................
[CV] ......... C=1, gamma=0.01, kernel=rbf, score=0.900, total= 1.7min
[CV] C=10, gamma=1, kernel=rbf .......................................
[CV] ........... C=10, gamma=1, kernel=rbf, score=0.900, total=21.6min
[CV] C=10, gamma=1, kernel=rbf .......................................
[CV] ........... C=10, gamma=1, kernel=rbf, score=0.900, total=13.3min
[CV] C=10, gamma=1, kernel=rbf .......................................
[CV] ........... C=10, gamma=1, kernel=rbf, score=0.900, total=12.8min
[CV] C=10, gamma=0.1, kernel=rbf .....................................
[CV] ......... C=10, gamma=0.1, kernel=rbf, score=0.900, total= 4.8min
[CV] C=10, gamma=0.1, kernel=rbf .....................................
[CV] ......... C=10, gamma=0.1, kernel=rbf, score=0.900, total= 4.8min
[CV] C=10, gamma=0.1, kernel=rbf .....................................
[CV] ......... C=10, gamma=0.1, kernel=rbf, score=0.900, total= 5.0min
[CV] C=10, gamma=0.01, kernel=rbf ....................................
[CV] ........ C=10, gamma=0.01, kernel=rbf, score=0.900, total= 2.3min
[CV] C=10, gamma=0.01, kernel=rbf ....................................
[CV] ........ C=10, gamma=0.01, kernel=rbf, score=0.900, total= 2.7min
[CV] C=10, gamma=0.01, kernel=rbf ....................................
[CV] ........ C=10, gamma=0.01, kernel=rbf, score=0.900, total= 2.3min

[Parallel(n_jobs=1)]: Done  27 out of  27 | elapsed: 107.4min finished
```

Out[36]:
```
GridSearchCV(cv=3, error_score=nan,
             estimator=SVC(C=1.0, break_ties=False, cache_size=200,
                           class_weight=None, coef0=0.0,
                           decision_function_shape='ovr', degree=3,
                           gamma='scale', kernel='rbf', max_iter=-1,
                           probability=False, random_state=None, shrinking=Tr
ue,
                           tol=0.001, verbose=False),
             iid='deprecated', n_jobs=None,
             param_grid={'C': [0.1, 1, 10], 'gamma': [1, 0.1, 0.01],
                         'kernel': ['rbf']},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=None, verbose=3)
```

In [37]:
```
grid.best_params_
#Printing the best hyperparameters which are 0.1 for C and 1 for gamma
```

Out[37]:
```
{'C': 0.1, 'gamma': 1, 'kernel': 'rbf'}
```

In [38]:
```
grid.best_score_
#Printing the best score
```

Out[38]:
```
0.8995100000199757
```

I have used grid search algorithm (which is almost brute force) for hyperparameter optimization. When data have high dimensions and have curse of dimensionality, this method is not recommended to use as evident from compilation times. I have tried created 36 fits earlier but it making the already resource intensive computation more cumbersome. So, I have stuck to 3 folds of each hyperparameter and to create 27 fits.

As the Above SVC fit so long to compile, I have only stuck with one regualarization parameter for the linear kernel

In [39]:
```
param_grid = {'C':[0.1], 'kernel':['linear']}
#Intialising the hyperparameters by specifying range for C and gamma with line
ar kernel
```

In [40]:
```
grid = GridSearchCV(SVC(), param_grid, verbose = 3, refit=True,cv=3)
grid.fit(X,Y)
#Executing gridsearch with the specified classfiers while applying SVC
```

```
Fitting 3 folds for each of 1 candidates, totalling 3 fits
[CV] C=0.1, kernel=linear ............................................

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent worke
rs.

[CV] ................ C=0.1, kernel=linear, score=0.900, total=  56.0s
[CV] C=0.1, kernel=linear ............................................

[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:   55.9s remaining:    0.
0s

[CV] ................ C=0.1, kernel=linear, score=0.900, total=  43.1s
[CV] C=0.1, kernel=linear ............................................

[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:  1.7min remaining:    0.
0s

[CV] ................ C=0.1, kernel=linear, score=0.900, total= 1.1min

[Parallel(n_jobs=1)]: Done   3 out of   3 | elapsed:  2.8min finished
```

Out[40]:
```
GridSearchCV(cv=3, error_score=nan,
             estimator=SVC(C=1.0, break_ties=False, cache_size=200,
                           class_weight=None, coef0=0.0,
                           decision_function_shape='ovr', degree=3,
                           gamma='scale', kernel='rbf', max_iter=-1,
                           probability=False, random_state=None, shrinking=Tr
ue,
                           tol=0.001, verbose=False),
             iid='deprecated', n_jobs=None,
             param_grid={'C': [0.1], 'kernel': ['linear']},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=None, verbose=3)
```

In [41]:
```
grid.best_params_
#Printing the best hyperparameters which are 0.1 for C and 1 for gamma
```

Out[41]:
```
{'C': 0.1, 'kernel': 'linear'}
```

In [42]:
```
grid.best_score_
#Printing the best score which is gamma
```

Out[42]:
```
0.8995100000199757
```

In [43]:
```
C_range=[0.1,1,10]     #specifying classifiers in order to plot
gamma_range =[10,.1,0.01]
```

In [44]:
```
 #Building an iterative loop to test each score of the classifiers
classifiers = []
for C in C_range:
    for gamma in gamma_range:
        clf = SVC(C=C, gamma=gamma, kernel="rbf")
        clf.fit(X,Y)
        classifiers.append((C,gamma,clf))
```

Printing the margin differentiators using the contour function. Specifying the upper and lower limit of the grid using mesh grid function. I have used the ravel function to flatten the data. Visualizing each parameter correlation using iterative loops. I have used only rbf kernel to visualize the data.

In [45]:
```
#Data visualisation
#Method 1
import matplotlib.pyplot as plt
from matplotlib.colors import Normalize
```

In [46]:
```
plt.figure(figsize=(12,10))
xxx, yyy = np.meshgrid(np.linspace(-0.01,20,200),np.linspace(-0.01,20,200))
#Specifying the size and range
```

```
<Figure size 864x720 with 0 Axes>
```

In [47]:
```
#Flattening data using ravel
xxx.ravel()
yyy.ravel()
```

Out[47]:
```
array([-1.e-02, -1.e-02, -1.e-02, ...,  2.e+01,  2.e+01,  2.e+01])
```

In [48]:
```
 #Specifying the scatter plot column 1 and column2
pre = X.iloc[:,0]
gl = X.iloc[:,1]
```

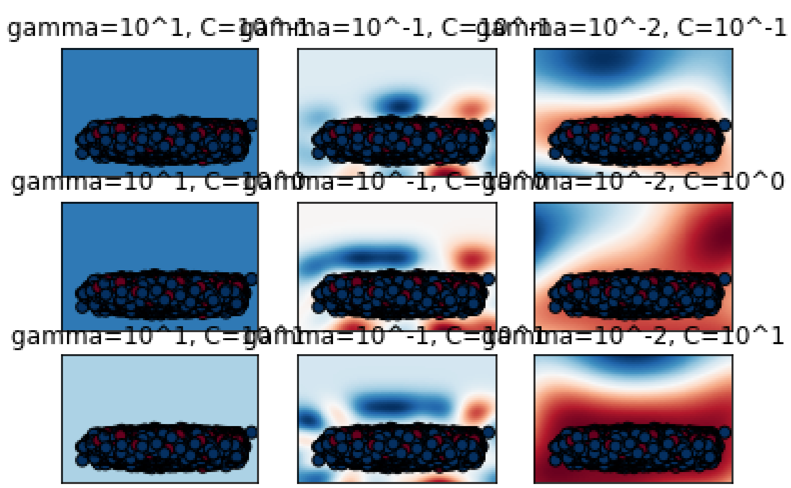
In [49]:
```
#Building iterative loop to visualize the SVC for each value of Hyperparameter
s C and gamma
for(k,(C,gamma,clf)) in enumerate(classifiers):

    W = clf.decision_function(np.c_[xxx.ravel(),yyy.ravel()])
    W = W.reshape(xxx.shape)

    plt.subplot(len(C_range), len(gamma_range), k+1)
    plt.title("gamma=10^%d, C=10^%d" %(np.log10(gamma),np.log10(C)))

    plt.pcolormesh(xxx,yyy, -W, cmap=plt.cm.RdBu)
    plt.scatter(pre,gl, c=Y, cmap=plt.cm.RdBu_r,edgecolors='k')
    plt.xticks(())
    plt.yticks(())
    plt.axis('tight')
    ay= plt.gca()
    ay.contour(xxx, yyy, W, colors='k', levels=[-0.1, 1, 10], alpha=0.5,  #dif
ferentiating lines to distinguish support vectors
           linestyles=['--', '-', '--'])
```

```
C:\Users\harsh\anaconda3\lib\site-packages\ipykernel_launcher.py:17: UserWarn
ing: No contour levels were found within the data range.
C:\Users\harsh\anaconda3\lib\site-packages\ipykernel_launcher.py:17: UserWarn
ing: No contour levels were found within the data range.
C:\Users\harsh\anaconda3\lib\site-packages\ipykernel_launcher.py:17: UserWarn
ing: No contour levels were found within the data range.
C:\Users\harsh\anaconda3\lib\site-packages\ipykernel_launcher.py:17: UserWarn
ing: No contour levels were found within the data range.
C:\Users\harsh\anaconda3\lib\site-packages\ipykernel_launcher.py:17: UserWarn
ing: No contour levels were found within the data range.
C:\Users\harsh\anaconda3\lib\site-packages\ipykernel_launcher.py:17: UserWarn
ing: No contour levels were found within the data range.
C:\Users\harsh\anaconda3\lib\site-packages\ipykernel_launcher.py:17: UserWarn
ing: No contour levels were found within the data range.
```



In [50]:
```
#Method 2
#Using Vstack, 'rbf' kernel and ravel. but not going through iterative loops.
 Summarizing in single graph.
model = SVC(kernel='rbf').fit(X,Y)
ax= plt.gca()
xx=np.linspace(-0.01,20,200)
yy=np.linspace(-0.01,20,200)
YY, XX=np.meshgrid(yy,xx)
xy=np.vstack([XX.ravel(),YY.ravel()]).T
ZZ = model.decision_function(xy).reshape(XX.shape)
ax.contour(xx, yy, ZZ, colors='k', levels=[-0.1, 1, 10], alpha=0.5,
           linestyles=['--', '-', '--'])
ax.scatter(model.support_vectors_[:,0],model.support_vectors_[:,1], s=20, line
width =1)
```

```
C:\Users\harsh\anaconda3\lib\site-packages\ipykernel_launcher.py:11: UserWarn
ing: No contour levels were found within the data range.
  # This is added back by InteractiveShellApp.init_path()
```

Out[50]: <matplotlib.collections.PathCollection at 0x1a761ed9b48>