HOMEWORK #6
LINEAR & NON-LINEAR SVM

Group Members: Onzali Suba (9690-3228-04)
and Nitya Pydipati (3219-5181-98)

Comparison with Scikit Learn

- Machine Learning Library in Python

sklearn.svm.**SVC** and sklearn.svm.**LinearSVC**

- LinearSVC is particularly meant for Linear kernels when dealing with linearly separable data, whereas SVC accepts a parameter called kernel where you can define the type of kernel transformation that you would like to apply for the non-linear separable data.
- Our algorithm is developed currently only to deal with hard classification problems. SVC and LinearSVC both consist of an argument 'C' that corresponds to the error term that needs to be included to allow certain misclassification of points.
- The different kinds of kernels available are 'linear', 'poly', 'rbf', 'sigmoid', 'precomputed'. If none is given, 'rbf' will be used.
- Also our implemented algorithm only works for binary classification whereas LinearSVC also includes support for multi-class classification through one-vs-the-rest. SVC on the other hand works on one-against-one approach for multi- class classification. If there are n classes, then a combination of their classifiers are trained and then the 'density_function_shape' parameter is used in order to provide a consistent interface to all the classifiers.

Application of SVM

- SVMs are helpful in text and hypertext categorization as their application can significantly reduce the need for labeled training instances.
- In Quantitative finance, for example predictive tasks, where x consists of features derived from a historical stock indicator time series and y is a sell or buy signal.
- Text based SVM, filter e-mail to be forwarded to either the inbox or the spam folder
- Used in Bioinformatics and research (Eg: Protein classification, Cancer classification)
- Another outstanding application of SVMs is the detection of human faces in gray-level images. The problem is to determine in an image the location of human faces and, if there are any, return an encoding of their position.
- Hand-written characters can be recognized using SVM.
- Many problems of image processing and image classification are accomplished by SVM.

Data Structures used in the algorithms

1) **Arrays (Numpy)**
   We use it in many parts of our algorithms for storing and retrieving values of data, y_predict and in many other instances.

```
X=data[:,0:2]
Y=data[:,2]
y_predict = np.zeros(len(X))
```

Challenges faced

- Understanding which kernel to choose
- Figuring out computation using quadratic functions and using constraints while doing so.
- Understanding how to convert the data points using kernel function and computing the weights in the new dimension.

Optimizations:

- Avoiding dot operator:

  for loop evaluates the function references each time and thus this leads to inefficiency. Thus we can initialize do this, so each time in the *for* loop when we calculate the value of b we don't have to reference it repeatedly:

```
b=self.b
for n in range(len(self.a)):
    b += self.sv_y[n]
    b -= np.sum(self.a * self.sv_y * Q[ind[n],sv])
```

- Instead of using for loops to find the alphas greater than 0, we have used numpy's built in functions to do so.

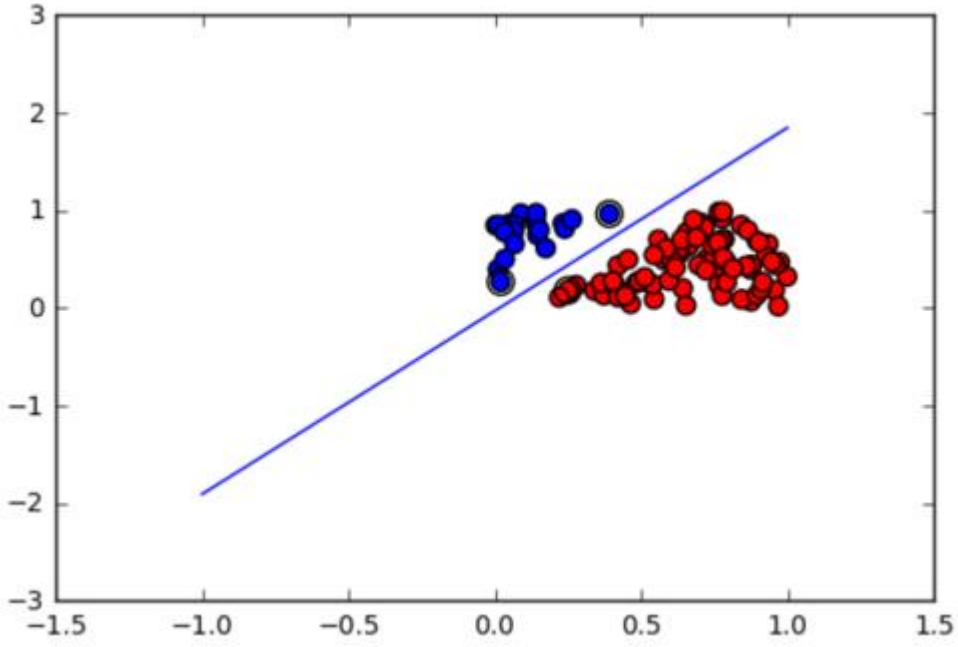
```
valid_alphas=alphas>0.00001
support_vector_indices = np.where(valid_alphas)[0]
```

Output:

- **Linear SVM:**

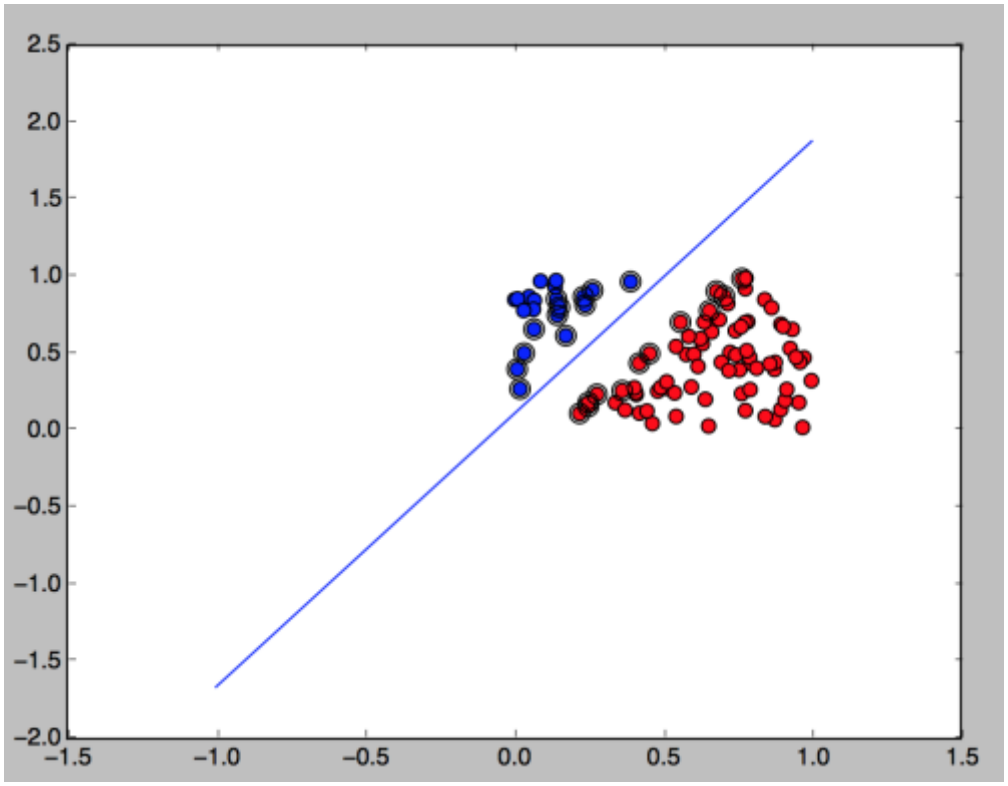
```
    pcost       dcost       gap    pres   dres
 0: -2.0636e+01 -4.3905e+01  3e+02  2e+01  2e+00
 1: -2.2372e+01 -3.7202e+01  9e+01  5e+00  5e-01
 2: -2.3112e+01 -3.8857e+01  5e+01  2e+00  2e-01
 3: -2.8318e+01 -3.3963e+01  1e+01  4e-01  4e-02
 4: -3.2264e+01 -3.3927e+01  2e+00  1e-02  1e-03
 5: -3.3568e+01 -3.3764e+01  2e-01  1e-03  1e-04
 6: -3.3737e+01 -3.3739e+01  2e-03  1e-05  1e-06
 7: -3.3739e+01 -3.3739e+01  2e-05  1e-07  1e-08
 8: -3.3739e+01 -3.3739e+01  2e-07  1e-09  1e-10
Optimal solution found.
[ 33.73875192   1.29468506  32.4440672 ]
3 support vectors out of 100 points
```

Intercept:
-0.106987267959
Weights:
[7.25005616 -3.86188932]



Graph from implemented algorithm

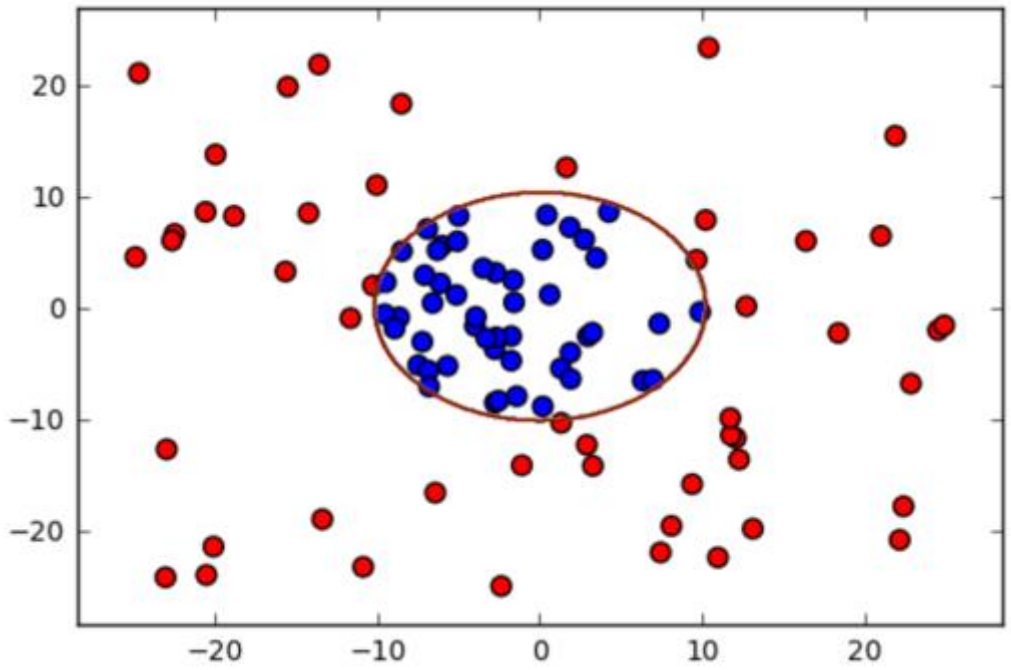
Output for Scikit:
Intercept:
[0.21848298]
Weights:
[3.59965788 -2.03198838]



Graph from Scikit Learn Library

- **Non-linear SVM:**

```
     pcost       dcost       gap    pres   dres
 0: -4.0666e+01 -1.0206e+02  5e+02  2e+01  3e+00
 1: -1.5924e+02 -2.2789e+02  3e+02  1e+01  1e+00
 2: -2.9280e+02 -3.6244e+02  3e+02  1e+01  1e+00
 3: -5.7710e+02 -6.0303e+02  4e+02  9e+00  1e+00
 4: -1.2873e+03 -1.2409e+03  5e+02  9e+00  1e+00
 5: -1.2647e+03 -1.0924e+03  7e+02  8e+00  9e-01
 6: -6.9076e+02 -4.0802e+02  1e+03  5e+00  6e-01
 7: -1.8688e+02 -2.9779e+01  4e+02  1e+00  2e-01
 8: -3.4731e+00 -5.2038e-02  1e+01  3e-02  4e-03
 9: -3.5053e-02 -3.8447e-02  1e-01  3e-04  3e-05
10: -2.1413e-02 -2.7448e-02  6e-03  8e-18  2e-13
11: -2.6166e-02 -2.6328e-02  2e-04  3e-18  2e-13
12: -2.6293e-02 -2.6295e-02  2e-06  4e-18  2e-13
13: -2.6295e-02 -2.6295e-02  2e-08  7e-18  2e-13
Optimal solution found.
6 support vectors out of 100 points
Intercept:
-16.6600524903
Weights:
[ 8.64344215e-10   1.60702131e-01   1.58698040e-01  -9.47655776e-03
  -3.85759106e-02  -1.10224094e-03]
```



Graph from Implemented Algorithm

Output for Scikit:
Intercept:
[-18.92792843]
Weights:
[-0.04937408 -0.08663716 0.109537 0.02647424]

Contributions:

Code: Colloborated

Report: Colloborated