SANKET NITIN WAGH

Problem Statement:

This assignment is based on Support Vector Machines. We will be implementing the Support Vector Machine algorithm on hard as well as soft margins. Experiments performed with different datasets (self-generated as well as those available on the UCI machine learning repository). Results are compared based on the accuracy values. The datasets were generated randomly.

Proposed Solution:

The algorithm implantation is phased out into data generation functions, training functions and prediction functions. The predictions are used to develop accuracy values. Each result has been plotted using matplotlib. Each implementation is done on separate files to avoid confusion.

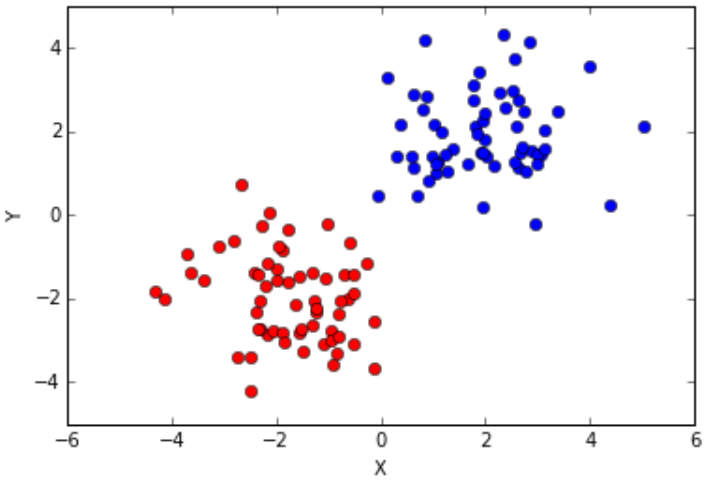
Implementation Details:

The user defined function data() is going to generate and plot the data. The user feeds the function with the number of features, the size of the data (how many data points to be generated) and the deviation that has to be produced. The number of dimensions will be set to 2 here since we are asked to generate 2D dataset. Using the deviation we generate a linearly non-separable as well as linearly separable datasets.

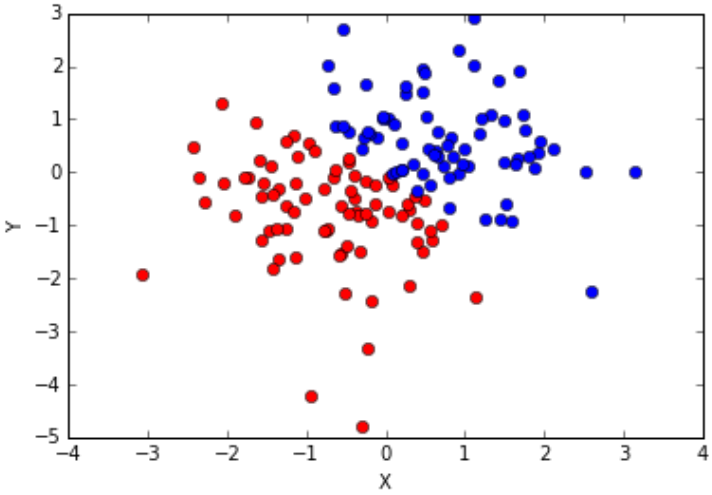
Depending on the type of margin we are going to implement on there will be either a hard() or soft() function that I have created that will train the data. This is done by implementing the algorithm. *We minimize the dual problem in this function and generate alpha values.* The alpha values are in the form of matrices and are used to determine the support vectors. Also when it comes to using the gram kernel I have integrated the kernel function and polynomial function into the training component. Once predictions are complete the accuracies are obtained and tabulated. The support vectors are plotted onto the existing plot.

Results and Discussion:

1. Data Generation:
    a. Linearly non-separable dataset:

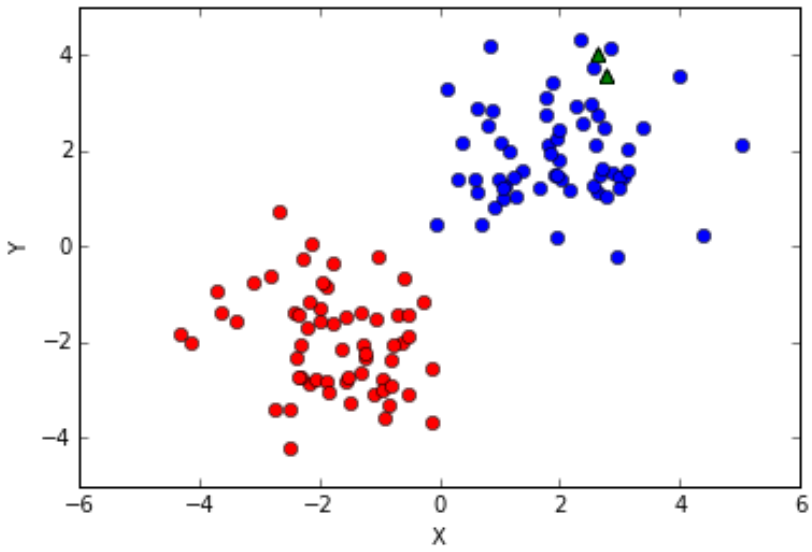


b. Linearly separable dataset:

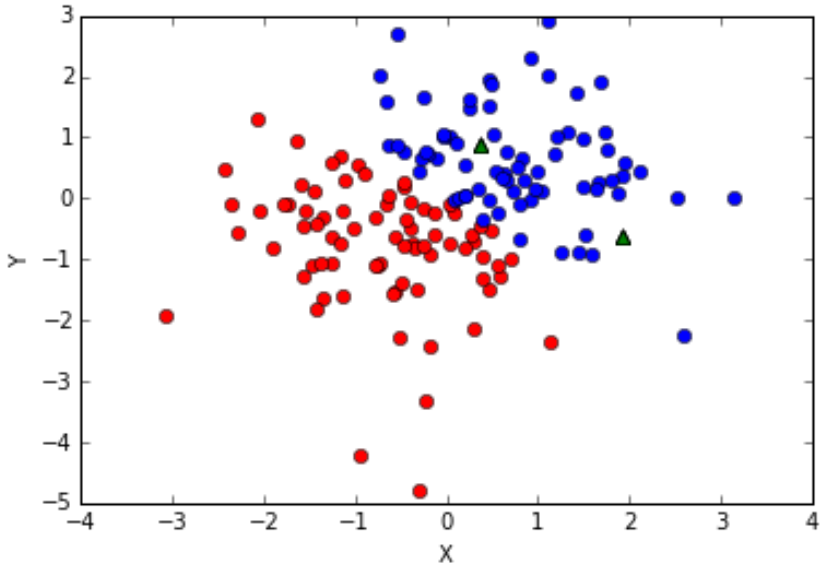


As mentioned in the design and implantation section the data is generated. I have used the random.rand() from the numpy library.

2. Hard margin implementation:
    a. The below figure represents the separable data with corresponding support vectors.



b. The below figure represents the non-separable data with corresponding support vectors.



- The accuracies obtained were as follows:
  - For the linear separable dataset generated we obtained 100% accuracy.
  - For the linear non-separable dataset generated we obtained 97.33% accuracy.
  - When the polynomial function was implemented, for the linear separable dataset generated we obtained 100% accuracy.
  - Also, when the polynomial function was implemented for the linear non-separable dataset generated we obtained 58.67% accuracy.
  - When the Gaussian kernel function was implemented, for the linear separable dataset generated we obtained 100% accuracy again.

- Also, when the Gaussian kernel function was implemented for the linear non-separable dataset generated we obtained 10% accuracy. I feel this is because the predictions were susceptible to noise values and outliers. As you can see from the graph there are a few outliers present.
- Overall conclusion suggests that the predictions were accurate and the model performed excellent on separable data and inefficiently on non-separable data.
- When I used cross validation. Each fold yielded the exact same accuracy value.

3. Derivation:

Objective:

⇒ To maximize L_D.

Objective function given:

$$L_P = \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{m}\xi_i - \sum_{i=1}^{m}\alpha_i(y^{(i)}(w^T x_i + w_0) - (1-\xi_i)) - \sum_{i=1}^{m}\beta_i\xi_i$$

① We must find w

Using the minimization equation given:

$$\frac{\partial L_P}{\partial w} = 0$$

$$\Rightarrow 2\times\frac{1}{2}\|w\| + 0 - \sum \alpha_i x_i y_i = 0$$

$$\Rightarrow \boxed{w = \sum_{i=1}^{m}\alpha_i x_i y_i}$$

② $\Rightarrow$ so for $w_0 \Rightarrow \frac{\partial L_P}{\partial w_0} = 0$

$$\Rightarrow -\sum_{i=1}^{m}\alpha_i y_i = 0$$

$$\Rightarrow \boxed{w_0 = \sum_{i=1}^{m}\alpha_i y_i}$$

③ $\frac{\partial L_P}{\partial \xi_i} = 0 \Rightarrow \boxed{C - \alpha_i - \beta_i = 0.}$

By substituting the equations obtained & by overlooking any neg-pos contradictions if any:

$$\Rightarrow \boxed{L_D = -\frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m}\alpha_i \alpha_j y_i y_j x_i^T x_j + \sum_{i=1}^{m}\alpha_i}$$

(Proof)

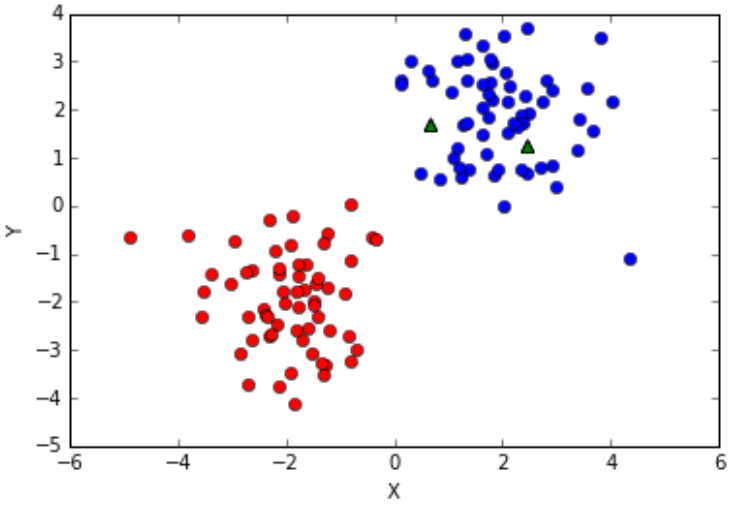
$$L_P = \frac{1}{2}\left(\sum_{i=1}^{m}\alpha_i y_i x_i^T\right)\left(\sum_{i=1}^{m}\alpha_i y_i x_i\right) + C\sum_{i=1}^{m}\xi_i$$

$$- \sum_{i=1}^{m}\alpha_i y_i\left(\left(\sum_{i=1}^{m}\alpha_i y_i x_i^T\right)x_i\right) + \sum_{i=1}^{m}\alpha_i y_i w_0$$

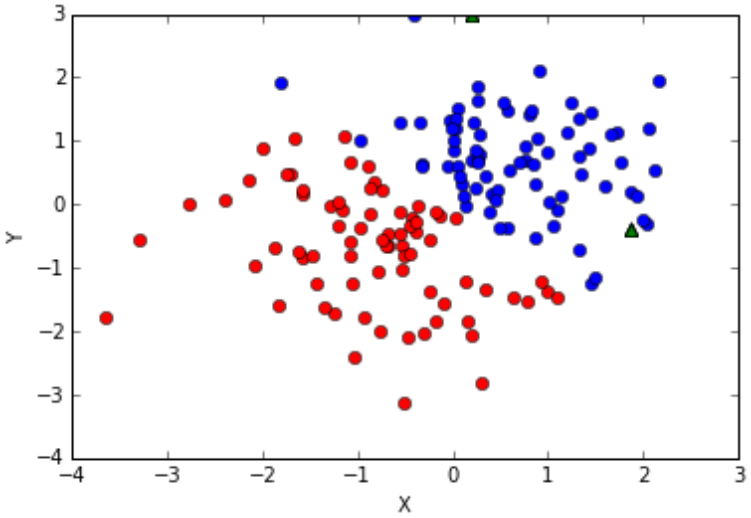
$$+ \sum_{i=1}^{m}\alpha_i - \sum_{i=1}^{m}\alpha_i \xi_i + \sum_{i=1}^{m}\beta_i \xi_i$$

& also, where $\alpha_i \geq 0$; $\sum_{i=1}^{m}\alpha_i y_i = 0$;
$\beta_i \geq 0$;
$C - \alpha_i - \beta_i = 0$
$0 \leq \alpha_i \leq C$

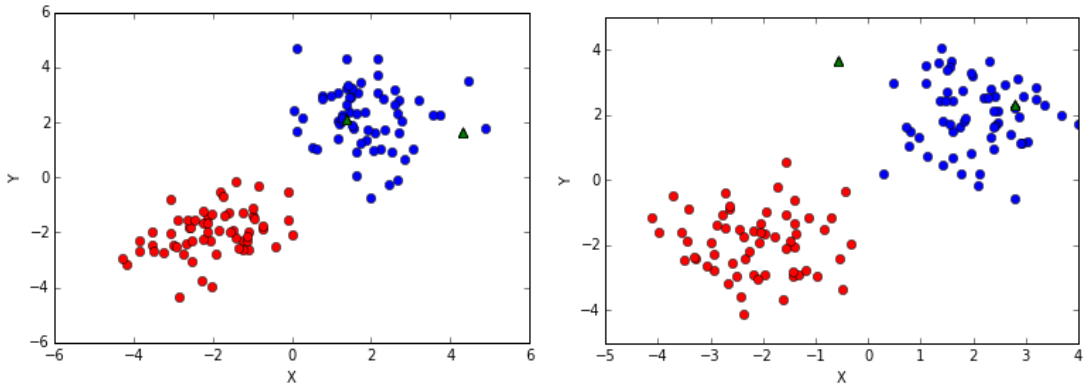
4. Soft margin implementation:
    a. The below figure represents the separable data with corresponding support vectors.



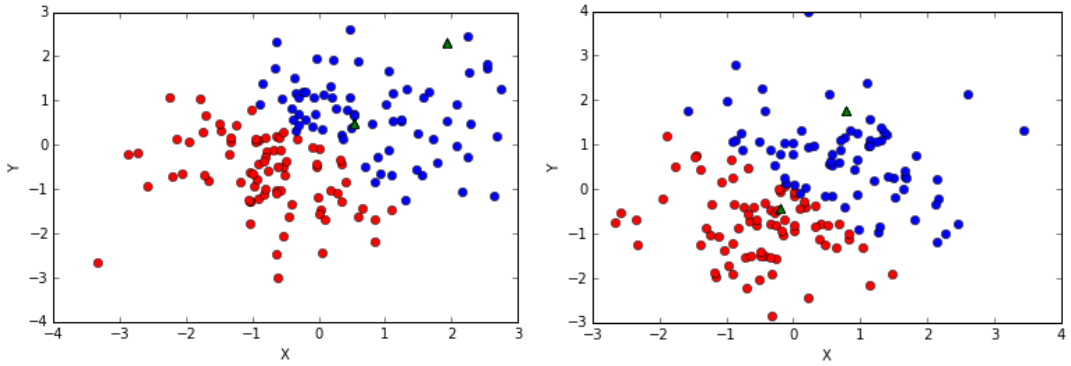
    b. The below figure represents the non-separable data with corresponding support vectors.



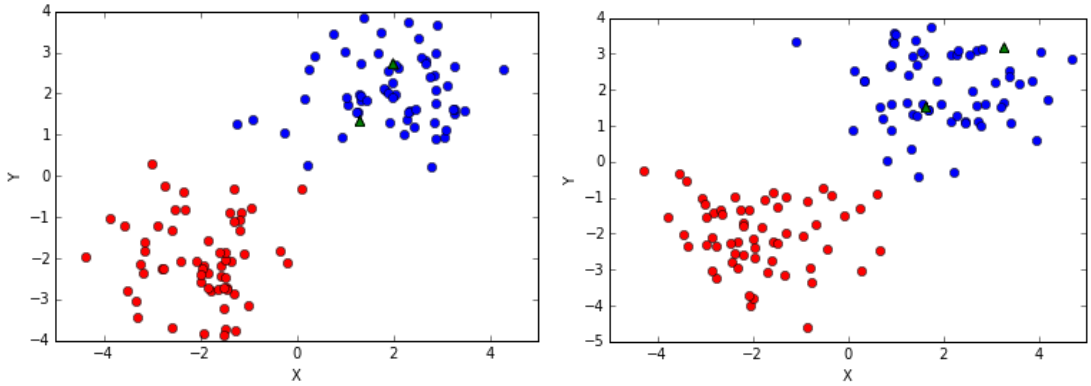
5. Hard margin: Gaussian and Polynomial Kernel functions:
    a. Separable datasets: Gaussian(left); Polynomial(right) :



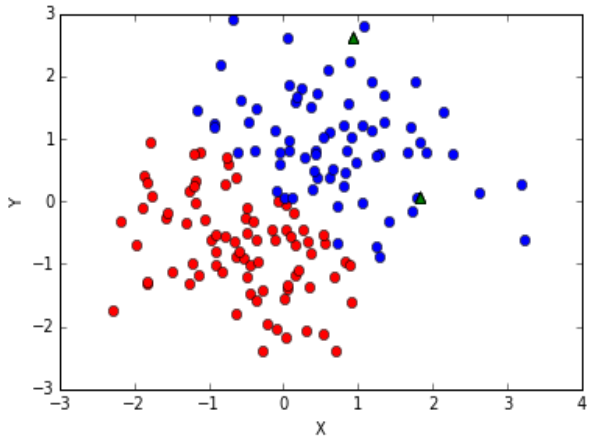
b. Non-separable datasets: Gaussian(left); Polynomial(right) :

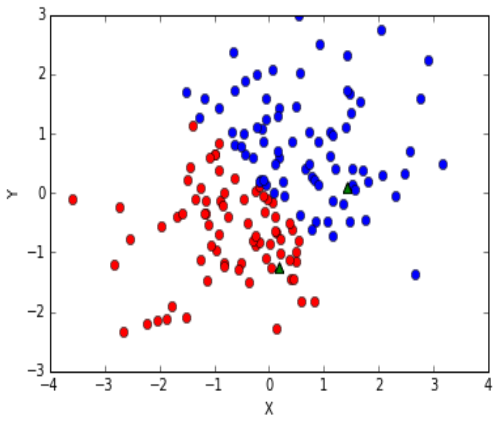


6. Soft Margin: Gaussian and Polynomial Kernel functions:
   a. Separable datasets: Gaussian(left); Polynomial(right) :



   b. Non-separable datasets: Gaussian(left); Polynomial(right) :





➔ In all the graphs above all the support vectors have been marked via green triangle.

- The accuracies obtained were as follows:
  - For the linear separable dataset generated we obtained 100% accuracy.
  - For the linear non-separable dataset generated we obtained 99.33% accuracy.
  - When the polynomial function was implemented, for the linear separable dataset generated we obtained 100% accuracy.
  - Also, when the polynomial function was implemented for the linear non-separable dataset generated we obtained 98% accuracy.
  - When the Gaussian kernel function was implemented, for the linear separable dataset generated we obtained 100% accuracy again.
  - Also, when the Gaussian kernel function was implemented for the linear non-separable dataset generated we obtained 98.67% accuracy.
  - Another test suggested that the accuracy dropped to 87% for non-separable data.
  - Overall conclusion suggests that the predictions were accurate and the model performed excellent.
  - When I used cross validation. Each fold yielded the exact same accuracy value same as the case with dealing with hard margins.

➔ It is fairly conclusive that handling data by employing soft margins yields better results. More accurate results.

Tests on external data sets: I used the Iris dataset and the Banknote Authentication dataset. I can predict that these two are not going to perform effectively.

Let us have a look with respect to employing hard margins:

- The Gaussian yielded very poor accuracies when Iris was used. An accuracy of 25.3 on one test and 8% on another (with inclusion of missing values).
- The Polynomial kernel achieved 25% accuracy and 33% accuracy under similar conditions.
- When the Gaussian kernel was implemented on banknote data it yielded 14.47% accuracy.

- When the polynomial kernel was implemented on banknote data it yielded 14.47% accuracy.

Let us have a look with respect to employing soft margins:

- Both the Gaussian and the Polynomial kernel yielded very poor accuracies when Iris was used. An accuracy of 25.33 each.
- When the Gaussian kernel was implemented on banknote data it yielded 29.4% accuracy.
- When the polynomial kernel was implemented on banknote data it yielded 44.46% accuracy.

➔ This does not mean the dataset is bad. Just that the distribution is different. We could definitely alter the labels according to the algorithm that we are using and yield good results. Since SVM deals with 1 and -1 only the results are sure to improve upon making appropriate changes to how the labels are classified and/or used. Also accuracy can be improved based on how the parameters are input.

➔ The most important conclusion is that soft margin implementation yields better results for majority of the time.

7. When large amount of samples are used we get a dense dataset but more or less ineffective results. The parameter input is to be altered according to the size of the data we generate to obtain efficient results.
   Another solution would be to have a very good split of training and testing data. Also in this case the C limiter was used with low values. If we want to further improve the efficiency of the result we must use the C limiter according to the size of the data. If we have 1000 samples, C should equal to 10, and so on.