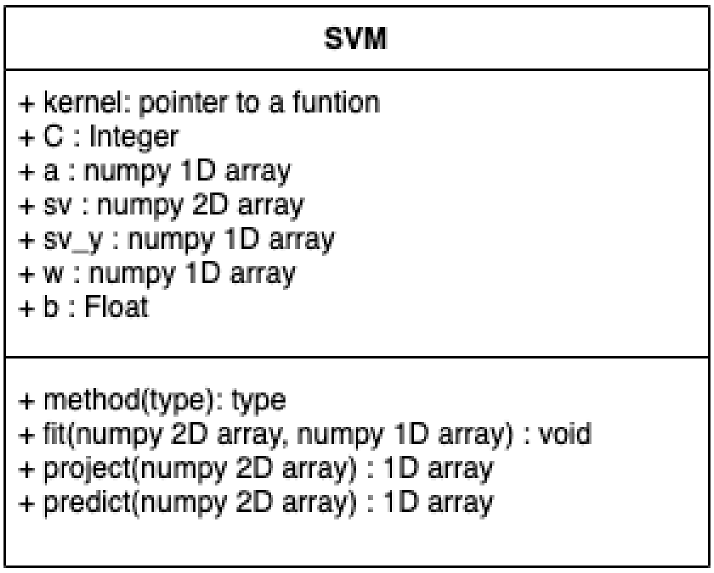
# INF-552
# MACHINE LEARNING FOR DATA SCIENCE

## PROGRAMMING ASSIGNMENT 6:
## Support Vector Machines

| NAME | USC-ID | EMAIL |
|---|---|---|
| Amitabh Rajkumar Saini | 7972003272 | amitabhr@usc.edu |
| Shilpa Jain | 4569044628 | shilpaj@usc.edu |
| Sushumna Khandelwal | 7458911214 | sushumna@usc.edu |

## PART 1: IMPLEMENTATION

- **Data Structures Defined/Used**
  - Dataset represented as numpy 2-D Array
  - Weights represented as numpy 1D array
  - Bias represented as float value



**SVM**

+ kernel: pointer to a funtion
+ C : Integer
+ a : numpy 1D array
+ sv : numpy 2D array
+ sv_y : numpy 1D array
+ w : numpy 1D array
+ b : Float

+ method(type): type
+ fit(numpy 2D array, numpy 1D array) : void
+ project(numpy 2D array) : 1D array
+ predict(numpy 2D array) : 1D array

### Support Vector Machine

A support-vector machine constructs a hyperplane or set of hyperplanes in a high- or infinite-dimensional space, which can be used for classification, regression, or other tasks like outliers detection. Intuitively, a good separation is achieved by the hyperplane that has the largest distance to the nearest training-data point of any class (so-called functional margin), since in general the larger the margin, the lower the generalization error of the classifier.

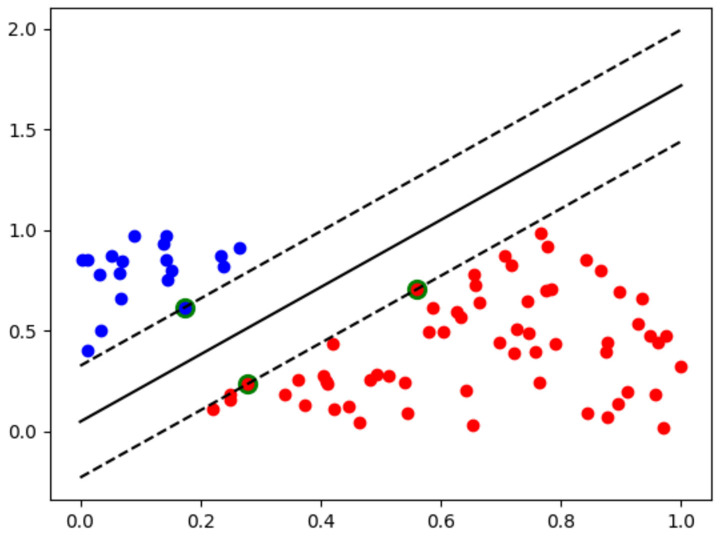
**Run and Output the result (Our Implementation)**

- **Linear SVM Kernel**

  **Output**:

```
--------------------------------LINEAR SVM KERNEL----------------------------------

             pcost       dcost       gap    pres   dres
 0: -1.6949e+01 -3.6032e+01  2e+02  2e+01  2e+00
 1: -1.4736e+01 -2.9453e+01  6e+01  3e+00  3e-01
 2: -1.5278e+01 -3.1893e+01  4e+01  1e+00  2e-01
 3: -1.8533e+01 -2.5727e+01  1e+01  5e-01  6e-02
 4: -2.1562e+01 -2.5201e+01  4e+00  4e-15  3e-15
 5: -2.4418e+01 -2.4647e+01  2e-01  6e-15  3e-15
 6: -2.4615e+01 -2.4617e+01  3e-03  6e-15  3e-15
 7: -2.4617e+01 -2.4617e+01  3e-05  1e-14  3e-15
 8: -2.4617e+01 -2.4617e+01  3e-07  6e-15  3e-15

Optimal solution found.

3 support vectors out of 80 points
-------Classifier--------
Alpha:
[12.28963366 12.32738322 24.61701775]
Bias:
0.17154933335516453
Weight:
[ 6.01928777 -3.60585752]
Center Margin Equation:
6.019287772752118 x1 + -3.605857518533089 x2 +
0.17154933335516453 = 0
Support Vector:
[[0.55919837 0.70372314]
 [0.27872572 0.23552777]
 [0.17422964 0.6157447 ]]
-------------------------

----Metrics----
TP: 18
FP: 0
TN: 2
FN: 0
Accuracy: 100.0
---------------
```

**Visualisation:**



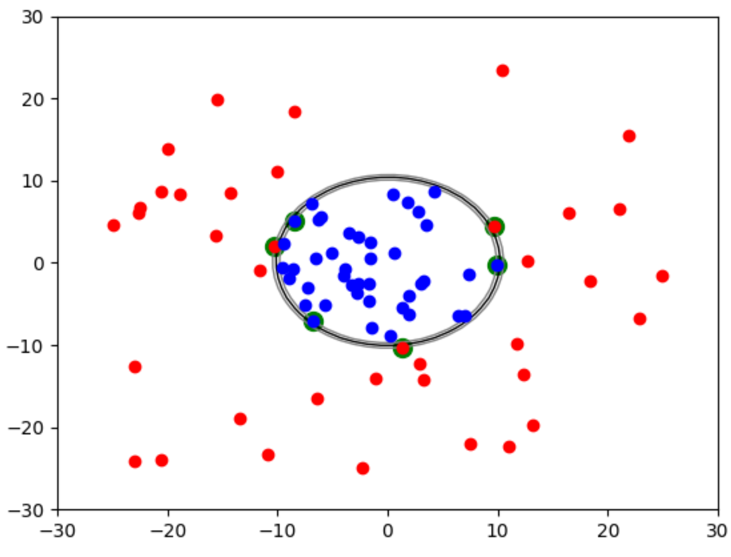
- **Non-Linear SVM Kernel**

  **Output:**

```
--------------------NON LINEAR POLYNOMIAL SVM KERNEL -------------------

     pcost       dcost       gap    pres   dres
 0: -1.6275e+04 -6.3269e+07  2e+08  8e-01  6e-08
 1: -1.1216e+04 -2.6538e+07  4e+07  2e-01  7e-08
 2: -7.7602e+03 -1.0750e+07  2e+07  6e-02  3e-08
 3: -5.4928e+03 -6.0365e+06  1e+07  3e-02  2e-08
 4: -3.0897e+03 -3.0826e+06  5e+06  1e-02  1e-08
 5: -1.1458e+03 -1.5090e+06  3e+06  6e-03  5e-09
 6:  4.5780e+02 -1.4069e+05  2e+05  3e-04  3e-09
 7:  4.2015e+02 -1.3004e+04  2e+04  2e-05  3e-10
 8:  2.4119e+02 -1.1270e+03  1e+03  5e-07  1e-11
 9:  4.1497e+01 -5.0998e+01  9e+01  5e-09  2e-12
10:  5.8934e+00 -6.6447e+00  1e+01  2e-16  1e-12
11:  8.1641e-01 -9.8806e-01  2e+00  2e-16  5e-13
12:  1.0343e-01 -1.5300e-01  3e-01  2e-16  2e-13
13:  6.9073e-03 -3.1339e-02  4e-02  2e-16  7e-14
14: -4.1710e-03 -3.3555e-02  3e-02  2e-16  1e-13
15: -1.9978e-02 -2.4474e-02  4e-03  1e-16  7e-14
16: -2.2597e-02 -2.2700e-02  1e-04  2e-16  1e-13
17: -2.2659e-02 -2.2663e-02  4e-06  2e-16  9e-14
18: -2.2661e-02 -2.2662e-02  5e-07  2e-16  8e-14
19: -2.2661e-02 -2.2661e-02  2e-08  1e-16  9e-14

Optimal solution found.
6 support vectors out of 80 points
-------Classifier--------
Alpha:
[0.00239794 0.00502193 0.01490102 0.01605393 0.00660751
0.00034022]
```

```
Bias:
-13.945230389341006
Weight:
None
Center Margin Equation:
0.1342486273684903 x1^2+ 0.12275891537307657 x2^2 + -
0.03195447732856952 x1 + -0.15262274882918492 x2 + -
0.012767054228946445 x1x2 + 3.2755670247188026e-07 + -
13.945230389341006 = 0

Support Vector:
[[  9.90143538  -0.31483149]
 [ -6.80002274  -7.02384335]
 [ -8.47422847   5.15621613]
 [-10.260969     2.07391791]
 [  1.3393313  -10.29098822]
 [ -9.46760885   2.36139525]]
-------------------------

----Metrics----
TP: 10
FP: 1
TN: 9
FN: 0
Accuracy: 95.0
---------------
```

**Visualization**



## CHALLENGES

- Fixing the value of C and choosing proper kernel function for maximum efficiency required a lot of trails.
- Finding the right function which would solve QP problem.
- Understanding and implementing CVXOPT library functions for solving QP problems efficiently.
- Finding the right function in the numpy library that offers a good implementation of matrix operations.

## CODE LEVEL OPTIMIZATION

- Usage of CVXOPT library functions helped to smoothen the whole process of finding the QP problems a lot.
- Most of the functions are vectorized, that helped in increasing the speed efficiently.
- Extensive use of Pandas and Numpy enabled us vectorize and perform matrix operations.
- Choosing polynomial kernel of degree 2 and the value of C is fixed to 1000.

## PART 2:  SOFTWARE FAMILIARIZATION

**Library Function:**

**sklearn.svm.SVC**

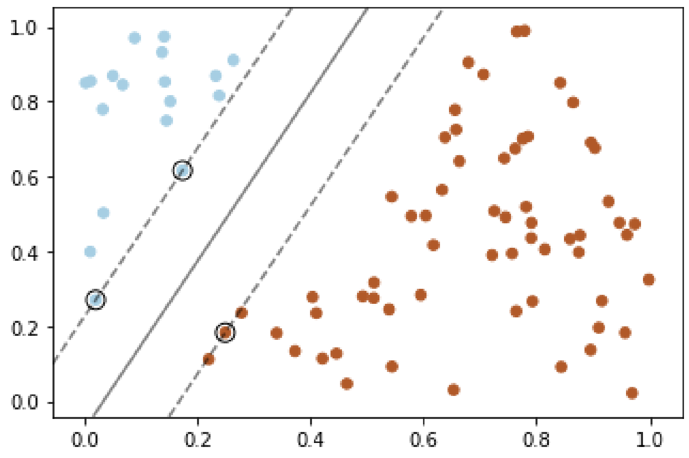
*class* sklearn.svm.**SVC**(*C=1.0*, *kernel='rbf'*, *degree=3*, *gamma='scale'*, *coef0=0.0*, *shrinking=True*, *probability=False*, *tol=0.001*, *cache_size=200*, *class_weight=None*, *verbose=False*, *max_iter=1*, *decision_function_shape='ovr'*, *break_ties=False*, *random_state=None*)

**Output:**

**SVM with Linear Kernel :**

```
weights:
[[ 7.45838668 -3.3139653 ]]
Intercept:
[-0.25902653]

Support Vectors:
array([[0.02066458, 0.27003158],
       [0.17422964, 0.6157447 ],
       [0.24979414, 0.18230306]])
```

**Visualization:**



**SVM with Non-Linear Kernel :**

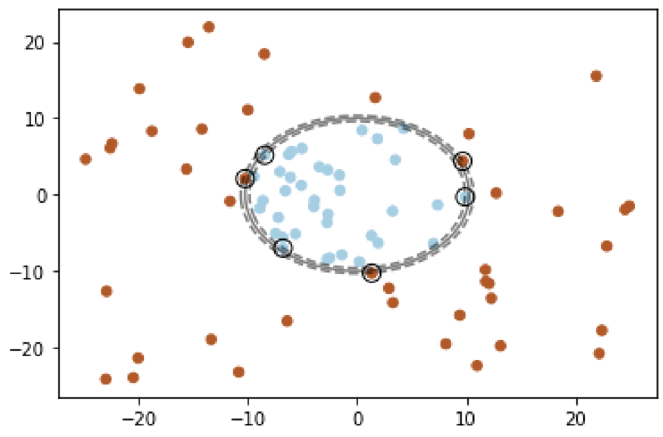
**Output:**

Support Vectors:

```
array([[ -8.47422847,   5.15621613],
       [  9.90143538,  -0.31483149],
       [ -6.80002274,  -7.02384335],
       [  9.67917724,   4.3759541 ],
       [-10.260969  ,   2.07391791],
       [  1.3393313 , -10.29098822]])
```

```
Intercept: [-13.59526067]
```

**Visualization:**



**Improvements:**

Sklearn employs a function sparsify() to convert the matrix in to a sparse matrix. Converts the coef_ member to a scipy.sparse matrix, which for models can be much more memory- and storage-efficient than the usual numpy.ndarray representation. We can consider this in our program to improve the runtime. We received better accuracy using the library implemention.

## <u>Comparison between our Implementation and sklearn library:</u>

Sklearn employs a function sparsify() to convert the matrix in to a sparse matrix which for models can be much more memory and storage-efficient than the usual numpy.ndarray representation. We can consider this in our program to improve the runtime.
Most of the algorithms in Sklearn are implemented using Vectorization and thus they are faster than our program. Also, library usage is good for quick implementation. It was easy to write the equation of hyperplane in our implementation.

## PART 3: APPLICATIONS

SVMs can be used to solve various real-world problems:

- SVMs are helpful in text and hypertext categorization, as their application can significantly reduce the need for labeled training instances in both the standard inductive and transductive settings. Some methods for shallow semantic parsing are based on support vector machines.
- Classification of images can also be performed using SVMs. Experimental results show that SVMs achieve significantly higher search accuracy than traditional query refinement schemes after just three to four rounds of relevance feedback.
- Classification of satellite data like SAR data using supervised SVM.
- Hand-written characters can be recognized using SVM.
- The SVM algorithm has been widely applied in the biological and other sciences. They have been used to classify proteins with up to 90% of the compounds classified correctly. Permutation tests based on SVM weights have been suggested as a mechanism for interpretation of SVM models. Support-vector machine weights have also been used to interpret SVM models in the past.

## PART 4: CONTRIBUTION

The project was planned and implemented by all group members.

1. We all discussed the design of the project.
2. The code was built in group together with discussion and peer reviews within the group.
3. Library function was studied by each member individually and collaborated to compare and analyse the difference between our results and library functions output on Linear and Polynomial SVM.
4. Design of data set, usage of data structures and kernel were all discussed and then coded.