SVM Implementation from Scratch using SMO Algorithm

Sai Srinadhu K
Pre-final year Undergraduate
2015csb1015@iitrpr.ac.in
IIT Ropar

Abstract

Support vector machines are supervised learning models that analyze data used for classification and regression analysis. In this project I implemented SVM from scratch using SMO algorithm for training SVM. Observations are made for Hard margin, Soft margin and Kernel SVM's on Artificial Data. Then taking a real world data-set breast cancer, my implementation is compared with that of inbuilt LIBSVM, though its not better than that of inbuilt one, my implementation gave accuracy within 5% of inbuilt one with a significance of 0.05.

Theory of SVM & Background for SMO

The problem is given a data set

$$X = (x_i, y_i)_{i=1}^{N}$$

where class labels are binary $+1$ and -1. Find w and w_0, s.t:

$$y_i(w^T x_i + w_0) >= 1, \forall i$$

The aim is to maximize margin (distance from hyper plane to closest instances) $2/||w||$ which is equivalent to minimize $||w||^2/2$ with the above given constraints. We will use Lagrange multipliers and KKT conditions to solve this problem. Lagrange multipliers:

$$min\ f(x),$$

subject to

$$h_i(x) = 0$$
$$g_i(x) \leq 0, \forall i$$

Lagrangian multipliers for inequalities is α_i and equalities h_i.The Lagrangian function is

$$L(x, \lambda, \alpha) = f(x) + \sum_i \lambda_i h_i(x) + \sum_i \alpha_i g_i(x)$$

KKT conditions are:

$$\Delta L(x^*, \lambda^*, \alpha^*) = 0$$
$$\alpha_i \geq 0$$
$$g_i(x^*) \leq 0$$
$$\alpha_i g_i(x^*) = 0$$
$$h_i(x^*) = 0$$

The dual problem becomes

$$L_d = max_{\alpha_i} -\frac{1}{2}\sum_i \sum_j \alpha_i \alpha_j y_i y_j x_i^T x_j + \sum_i \alpha_i$$
$$\text{s.t} \sum_i \alpha_i y_i = 0$$
$$\alpha_i \geq 0$$

This dual problem will be solved by our SMO algorithm. Support vectors are those points for which Lagrange multipliers value is more than 0. This is a discriminant based method. Till now this is all Hard Margin SVM.

Let's see the case of Soft Margin SVM's, here data isn't linearly separable our aim is to find the hyper plane with least error. Now constraints change to

$$y_i(w^T x_i + w_0) >= 1 - \epsilon_i$$

and $\epsilon_i \geq 0\ \forall i$. The error function becomes

$$min \frac{1}{2}||w||^2 + C\sum_i \epsilon_i$$

where C is a penalty factor that stresses the importance on decreasing the error. The dual problem becomes

$$L_d = max_{\alpha_i} -\frac{1}{2}\sum_i \sum_j \alpha_i \alpha_j y_i y_j x_i^T x_j + \sum_i \alpha_i$$
$$\text{s.t} \sum_i \alpha_i y_i = 0$$
$$0 \leq \alpha_i \leq C, \forall i$$

This dual problem will be solved by our SMO algorithm. Then we go to case where data isn't linearly separable and we use kernel functions to separate the data.

The idea for usage of kernels is that is if the data isn't linearly separable in current space, perhaps it's linearly separable in some high dimension space. Instead of taking dot product in present dimension we then need to take dot product in higher dimensions which can be very expensive if we have to calculate the transformation to that space. This

is where kernel comes, it gives dot product of vectors in present dimension in that of transformed one which solves our problem, since we only need dot product in that higher dimension. More formally a kernel matrix K or Gram matrix is

$$K(x_i, x_j) = \Phi(x_i)^T \Phi(x_j)$$

. We needn't know what transformation is if we can define a kernel matrix which does our job. Note that all functions can't be kernel functions. For any function to be kernel function it has to satisfy Mercer's theorem. Some widely used kernels are

$$\phi(x, x') = (x^T x' + 1)^q$$

$$\Phi(x, x') = \exp -\frac{||x - x'||^2}{2s^2}$$

The first one is polynomial kernel and second one is Gaussian kernel. Both can be easily shown to satisfy mercer's condition with a cute observation that the transformed space of Gaussian kernel is infinite. kernel is like a similarity measure, more similar more value less similar less value. Using kernel functions in dual problem the $x_i^T x_j$ will change to $K(x_i, x_j)$.

So all in all if we have Lagrangian multipliers, our whole problem is solved and we are done. SMO algorithm deals with them,

SMO Algorithm

SVM computes linear classifier of the form $f(x) = w^T x + b$ where we predict 1 if $f(x) \geq 0$ else -1. By dual problem it can be represented as

$$f(x) = \sum_i \alpha_i y^{(i)} \langle x^{(i)}, x \rangle + b \tag{1}$$

where it can be substituted by kernel if needed. The Sequential Minimal Optimization (SMO) algorithm gives an efficient way of solving dual problem given in above section.
The SMO algorithm takes two Lagrangian multipliers and optimizes function value jointly for both of the multipliers. b parameter is adjusted based on these new Lagrangian multipliers. This process is repeated till their convergence.

In the next 3 sub-sections we will see, how to choose pair of Lagrangian multipliers,how to optimize the objective function and calculate b.

Selecting Parameters:

This section deals with heuristic for choosing pair in simplified SMO algorithm. Here the heuristic for choosing pair of Lagrangian multipliers is much simple. Iterate over all α_i,if it doesn't fulfill KKT conditions choose randomly a α_j which isn't same as α_i and then optimize the objective function for this pair. If none of Lagrangian multipliers change for few iterations over all $\alpha_i^{'s}$, then we exit our algorithm and say we are done. Note that this may not always always lead to global optimum for probability that $\exists$ a pair which could be optimized even now is non-zero but very small depending on number of features and max passes which we will define in next section. This is a much simpler version than full SMO algorithm but if it gives results comparable to SMO, I think it would be great given how simple it's to code up as you will see soon. In the next sub-section we deal with how to optimize the objective function.

Optimizing Objective Function:

Let's say we choose α_i and α_j to optimize which must satisfy the constraint $0 \leq \alpha_j \leq C$ for which we need lower and upper bounds, we get them like this:
$if\ y^{(i)} \neq y^{(j)}$:

$$L = max(0, \alpha_j - \alpha_i), H = min(C, C + \alpha_j - \alpha_i) \tag{2}$$

$else$:

$$L = max(0, \alpha_j + \alpha_i - C), H = min(C, \alpha_j + \alpha_i) \tag{3}$$

It can be derived that the optimal value is:

$$\alpha_j = \alpha_j - \frac{y^{(i)}(E_i - E_j)}{\eta} \tag{4}$$

where

$$E_k = f(x^{(k)}) - y^{(k)} \tag{5}$$

$$\eta = 2\langle x^{(i)}, x^{(j)} \rangle - \langle x^{(i)}, x^{(i)} \rangle - \langle x^{(j)}, x^{(j)} \rangle \tag{6}$$

where $y^{(k)}$ can be calculated using (1). Now we clip the value of α_i to lie in the interval $[L, H]$

$$\alpha_i = \begin{cases} H, \ if\ \alpha_j > H \\ \alpha_j \ \text{ if } \ L \leq \alpha_j \leq H \\ L, \ if\ \alpha_j < L \end{cases} \tag{7}$$

Finally α_i can be solved like this

$$\alpha_i = \alpha_i + y^{(i)} y^{(j)} (\alpha_j^{(old)} - \alpha_j) \tag{8}$$

The full algorithm handles the case where $\eta = 0$ but here if it occurs we assume we can't make any progress. In the next section we deal with calculating b threshold.

Threshold b:

For calculating threshold b, we calculate b_1 and b_2 for α_i and α_j respectively. If both lie within penalty parameter both will be same and which will be value of b else we take for which Lagrangian multiplier lies in $(0, C)$.

$$b_1 = b - E_i - y^{(i)}(\alpha_i - \alpha_i^{(old)})\langle x^{(i)}, x^{(i)} \rangle - y^{(j)}(\alpha_j - \alpha_j^{(old)})\langle x^{(i)}, x^{(j)} \rangle$$

$$b_2 = b - E_j - y^{(i)}(\alpha_i - \alpha_i^{(old)})\langle x^{(i)}, x^{(j)} \rangle - y^{(j)}(\alpha_j - \alpha_j^{(old)})\langle x^{(j)}, x^{(j)} \rangle$$

Now computing the threshold b:

$$b = \begin{cases} \mathrm{b}_1, \ \text{if } \ 0 < \alpha_i < C \\ \mathrm{b}_2 \ \text{ if } \ 0 < \alpha_j < C \\ (\mathrm{b}_1 + b_2)/2, \ \text{otherwise} \end{cases} \tag{9}$$

Pseudo Code for Simplified SMO:

Input:
C: regularization parameter
tol: tolerance
maxpasses: max times to iterate without changing
$(x^{(1)}, y^{(1)}), ...(x^{(m)}, y^{(m)})$:training data

Output: α and b

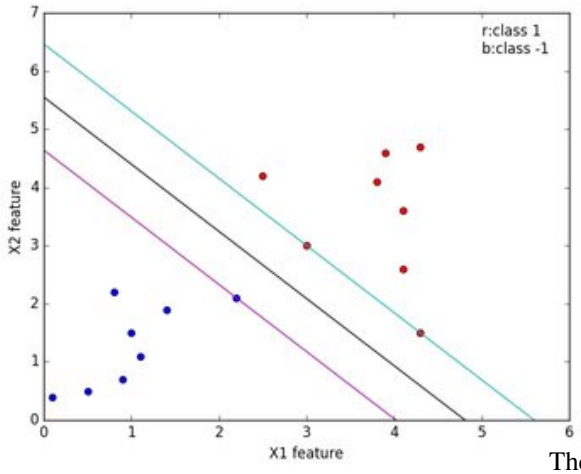
Initialize $\alpha_i = 0, b = 0$;
Initialize $passes = 0$;
while $passes < maxpasses$ **do**
  numchangedalpha=0;
  foreach $i = 1, ..., m$ **do**
    $E_i = f(x^{(i)}) - y^{(i)}$ using (1);
    if $(y^{(i)}E_i < -tol$ & $\alpha_i < C)$ *or*
     $(y^{(i)}E_i > tol$ & $\alpha_i > 0)$ **then**
      select $j \neq i$ randomly;
      calculate $E_j = f(x^{(j)}) - y^{(j)}$ using (1);
      save old ones $\alpha_i^{(old)} = \alpha_i$ and $\alpha_j^{(old)} = \alpha_j$;
      Compute L and H by (2) and (3);
      if $(L == H)$ **then**
        Continue to next i;
      else
        pass;
      end
      compute η by (6);
      if $(\eta \geq 0)$ **then**
        Continue to next i;
      else
        pass;
      end
      Compute and clip the value of α_j by (7) and (8);
      if $(|\alpha_j - \alpha_j^{(old)}| < tol)$ **then**
        Continue to next i;
      else
        pass;
      end
      update value of α_i by (8) ;
      compute b by (9);
      numchangedalpha+=1;
    else
      continue;
    end
  end
  if *(numchangedalpha=0)* **then**
    passes+=1;
  else
    passes=0;
  end
end

Algorithm 1: Simplified SMO Algorithm

Understanding the algorithm and derivations of equations in algorithm takes lot of effort but after that implementation is pretty straight forward. In the next section experiments conducted with my implementation are discussed.

Experiments & Discussion

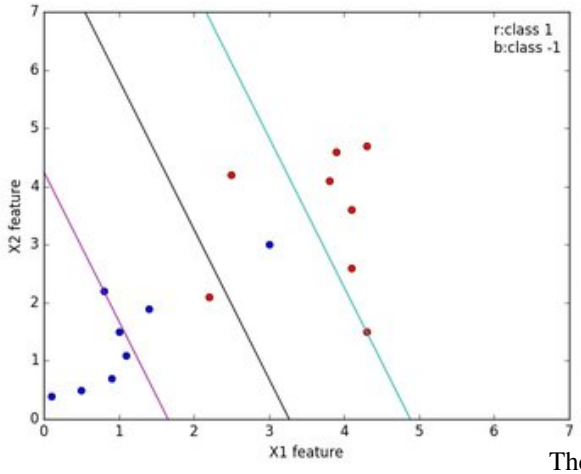
First Hard-Margin SVM is looked into with an artificial data set which is linearly separable. Lagrangian multipliers are got through my implementation of SMO.There are 3 support vectors for this data set



The red points are one class and blue are other class. Black line is optimally separating Hyper-plane. The two red points on cyan line and one blue point of magenta line are support vectors for this data set

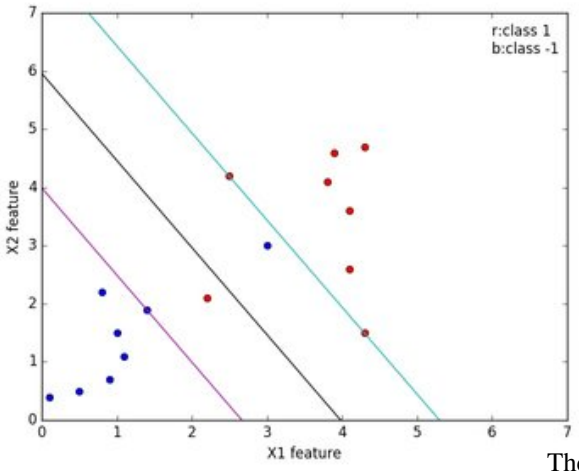
Figure 1: Hard Margin SVM for Linearly separable data-set

Next it's Soft-Margin SVM is looked into with an artificial data-set which is not linearly separable. Varying values of C observations are made.



The red points are one class and blue are other class. Black line is optimally separating Hyper-plane. There are 6 support vectors and value of C=0.25

Figure 2: Soft Margin SVM (C=0.25)



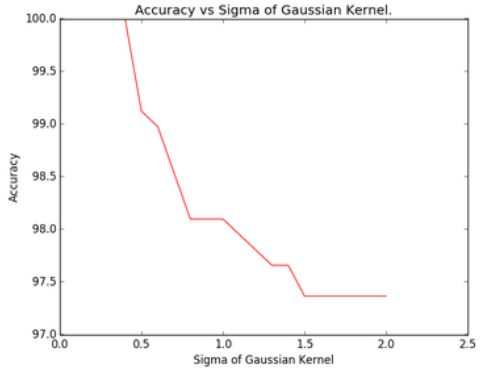
The red points are one class and blue are other class. Black line is optimally separating Hyper-plane. There are 5 support vectors and value of C=1.0

Figure 3: Soft Margin SVM (C=1.0)

We can observe that soft margin error for small C(0.25) is more than that for large value of C(1.0) which is what is expected since larger the C value smaller is the hyper-plane it looks, which can also be observed from plots.

Then taking a artificial data which is not linearly separable in present space, kernel functions are used to separate them in transformed space, accuracy is observed with tuning C,sigma,degree of polynomial kernel. We will discuss the same tuning with real-world data-set and observations will be made there.

The real world data-set used is that of breast cancer one obtained from UCI repository. It's a two class data-set with 10 features for each data point. The data obtained was already normalized to $[-1, 1]$. There are total of 683 data points. The observations are as follows:



The accuracy decreased with increasing σ till 2.0

Figure 4: Accuracy vs σ of Gaussian Kernel (C=1.0)

If the distance between x and x' is very large compared to σ then kernel value tends to 0 for that. σ plays a role of amplifier of distance between data points. Smaller sigma tends to make a local classifier, larger sigma tends to make a much more general classifier. This might be the reason for the observation as local classification (lower σ) is giving higher accuracy compared to other one. Number of support vectors doesn't follow any particular pattern for which plot is included in code. The next plot is accuracy vs degree of polynomial kernel.

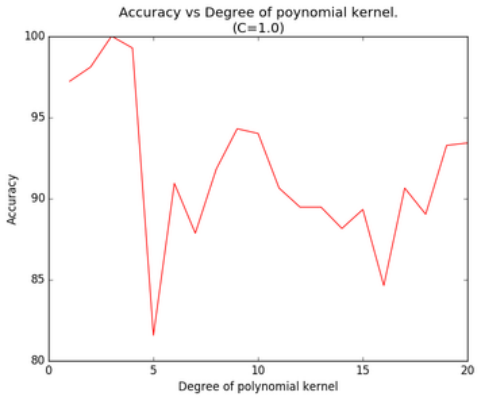


Figure 5: Accuracy vs degree of Polynomial Kernel (C=1.0)

Higher degree kernels allow a more flexible decision boundary. Degree 3 seems to be the best fit for this data-set. The reason for the above observation might be that, at the degrees of low accuracy, it might have happened that the margin it's trying to learn in that dimension is mis-classifying some data points for which increasing C may result in better training accuracy. Number of support vectors here also doesn't follow any particular pattern for which plot is included in code. The next plot is for Accuracy vs C for Gaussian kernel.

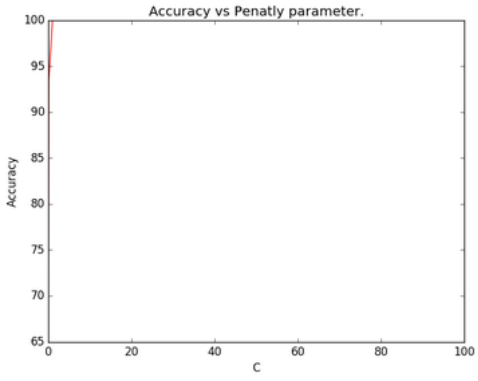


Figure 6: Accuracy vs C (Gaussian Kernel) (degree=3)

At small C, it's tries to learn higher width margin which might lead to some mis-classifications and as C is increased it tries to learn smaller width margin to an extent and then increasing C will not usually have much affect, so in higher C we might expect a increase in accuracy which is what was precisely observed here. Now we will come to support vectors.

After an increase in C from a limit, the hyper-plane's

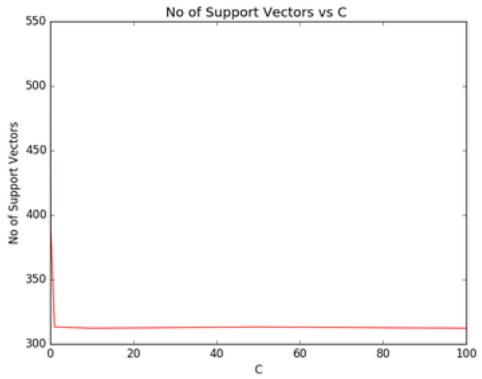


Figure 7: No of Support Vectors vs C (Gaussian Kernel) (degree=3)

may not change very much leading to almost no change in support vectors which is what is observed above.

Now coming to comparison between LIBSVM and my implementation.The best model is saved for my implementation (C=1.0, $\sigma = 0.4$, rbf). Then using inbuilt svm library the best model is saved (c=2.0, $\sigma = 1.5$, rbf). Best model is taken as the one which gave 100% accuracy on training data.
10-fold cross validation is done on both inbuilt one and non-inbuilt one.

Fold number	Own accuracy	Inbuilt accuracy
1	94.1	88.23
2	89.7	97.05
3	95.58	98.52
4	83.8	94.117
5	89.7	95.58
6	95.5	97.058
7	97.058	97.058
8	94.117	98.529
9	95.58	100.0
10	95.58	97.058

Performing a paired t-test, It's extremely easy to see that the hypothesis: "Inbuilt has better results than Own Implementation with a significance level of 0.05" can be accepted. For my implementation average accuracy is 93.071 and standard deviation is 3.89, for inbuilt one is 96.32 and 3.10 respectively. Now performing paired t-test on new hypothesis "Absolute value of difference of % accuracy between my implementation and inbuilt one is at most 5%". This can be accepted with significance level of 0.05 (can be shown with some basic calculations). This tells us that my implementation's accuracy isn't not so far from full SMO's but some dis-advantages of my implementation compared to that of full SMO are: mine takes good amount of time for bigger data-sets compared to inbuilt one and it may not always converge to global optimum like that of full SMO.

Conclusion

The main result or conclusion of this project is that the simplified SMO (one implemented by me) is not so far of from inbuilt SMO with regard to accuracy or more formally saying "Absolute value of difference of % accuracy between my implementation and inbuilt one is at most 5% can be accepted with significance level of 0.05".

Future work

Using this as a base for implementing full SMO algorithm. Looking at support vector regression and structured support vector machine. One idea I plan to explore is that, I feel any noise less data-set of 2-class classification has a kernel function which linearly separates it, proving or disproving this theorem and giving a procedure/algorithm of getting the kernel for any noise less data-set if it's possible.

References

1.http://image.diku.dk/imagecanon/material/cortes_vapnik95.pdf
2.https://pdfs.semanticscholar.org/59ee/e096b49d66f39891eb88a6c84cc89acba12d.pdf
3.http://cs229.stanford.edu/materials/smo.pdf
4.http://jonchar.net/notebooks/SVM/
5.http://haohanw.blogspot.in/2014/03/ml-how-sigma-matters-in-svm-rbf-kernel.html
6.https://neerajkumar.org/writings/svm/
7.https://www.youtube.com/watch?v=AbVtcUBlBok
8.https://en.wikipedia.org/wiki/Support_vector_machine
9.https://dl.acm.org/citation.cfm?id=1961199
10.https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary.html#breast-cancer
11.http://scikit-learn.org/stable/modules/svm.html#svm-classification