Universitat Politècnica de Catalunya

Barcelona School of Informatics

Master in Innovation and Research in Informatics

Optimization Techniques for Data Mining

- Support Vector Machine -

Author
Marc Mendez
Ricard Meyerhofer

Lecturer
Jordi Castro

December 7, 2019

Index

1 Introduction

The goal of this assignment is to implement a Support Vector Machine in primal and dual quadratic form. Once implemented in AMPL, we will validate the SVM's with different datasets and verify that the separation hyperplane from the dual model coincides with the primal one. But what is a SVM? Informally, a SVM is a method that is one of the best known and most widely methods used for classification and regression analysis. More concretely, a SVM model is a representation of data in space, mapped in a way that separate categories are divided by a clear gap that is as wide as possible (which is not always possible so we might need to leave a soft-margin).

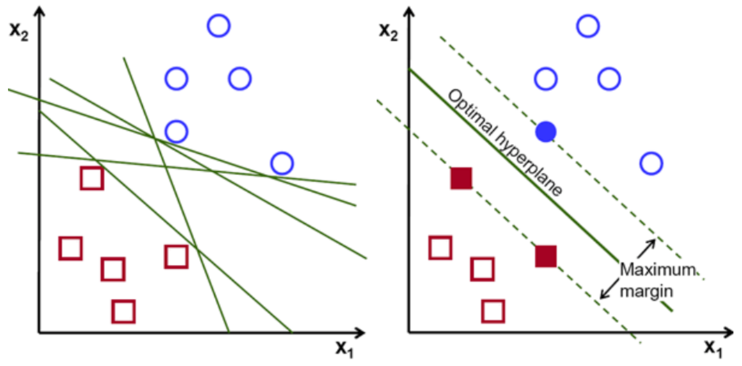

Figure 1: Hyperplanes that are a solution VS optimal

Prior to explain each of the forms, we are now going to formally introduce our problem. We want to classify m points of $\mathbb{R}^4$, represented by the matrix $A \in \mathbb{R}^{m\times 4}$. Each point belongs to the class $+1$ or -1 depending on its classification in a diagonal matrix $Y \in \mathbb{R}^{m\times 4}$ the diagonal element $y_i = +1$ if the point i belongs to the class $+1$, otherwise i belongs to the class -1. However, we are not only interested in classifying correctly, we are interested in maximizing our margin so that our classifier is more robust. We define margin as the distance between the two delimiting planes ($M = ||x^+ - x^-|| = \frac{2}{||w||_2}$). When both classes are linearly separable, the plane is $x^T w + \gamma = 0$ where γ determines the location relative to the origin. In case that the classes are not linearly separable, the planes are limiting the smooth margin. To measure the error, we have the vector $s_i = 1..m$ which means the error of classification in the point i. When they are separable, s can be 0.

2 SVM Primal

If we take the aforementioned problem, we can see that this is a constraint optimization problem. We want to maximize our margin subject to the fact that all the data is correctly classified. If instead of thinking of the problem as a maximization we think of it as a minimization, we can write it the minimization of the inverse of the margin.

$$\min_{w,\gamma,s\in \mathbb{R}^{n+1+m}} \frac{1}{2}||w||_2^2 + \nu e^T s$$

subject to:

$$Y(Aw + e\gamma) + s \geq e,$$
$$s \geq 0$$

Where ν is a parameter chosen by the user, e is a vector of ones, A is the input matrix. As aforementioned, Y corresponds to the class assigned, w, γ, s are decision variables. We implemented this form in AMPL by using the scalar form:

$$\min_{w,\gamma,s\in \mathbb{R}^{n+1+m}} \frac{1}{2}w^T w + \nu \sum_{j=1}^{m} s_j$$

where $w^T w$ is equal to $\sum_i^n w_i^2$

subject to:

$$y_i \cdot (\sum_{i=1}^{n} w_n \cdot x_{i,n} + \gamma) + s_i \geq 1$$
$$s_i \geq 0$$

3 SVM Dual

By solving the Lagrangian dual of the above problem, we have the following equation. Note that we should obtain the same solution using the primal and the dual which we will prove later. In the dual, the objective function is defined as:

$$\max_{\lambda\in \mathbb{R}^{m}} \lambda^T e - \frac{1}{2}\lambda^T Y A A^T Y \lambda$$

subject to:

$$\lambda^T Y e = 0$$

$$0 \leq \lambda \leq \nu$$

To solve this with AMPL, we have passed it to scalar form.

$$\sum_{i=1}^{m} \lambda_i - \frac{1}{2} \sum_{i}^{m} \sum_{j}^{m} \lambda_i y_i \lambda_j y_j K_{ij}$$

subject to:

$$\sum_{i=1}^{m} \lambda_i y_i = 0$$

$$0 \leq \lambda \leq \nu$$

where K_{ij} is the Kernel matrix that is equal to AA^T.

4 Modelling

In this section we are going to explain how we have trained and validated our results. Furthermore, we will conclude on the hyperplanes of the dual and primal form and if they are or not different. In addition, we are going to use a simplified version of the Iris dataset to test our models.

4.1 Training and test phase

In our case, we created two files: one training and one test set. Our training set consists of 100 individuals while the test set consists of 200 individuals (both with different seeds).

We created both datasets from the script facilitated us (gensvmdat). The data that the script outputs have the following properties:

- Each row is a data point.
- First four are random numbers between 0 and 1 (which will be our number of variables x)
- The result of adding up these values is greater than 2, the fifth value is 1 otherwise -1 (which will be our y).

- Fifth column can also contain an "*" which states that it means that the point is wrongly classified.

One example of the aforementioned could be:

```
# Sample of 7 rows with size=100 and seed=54321
0.415 0.356 0.896 0.921 1.0
0.878 0.907 0.309 0.331 1.0
0.988 0.876 0.653 0.604 1.0
0.626 0.677 0.131 0.958 1.0
0.641 0.426 0.421 0.207 -1.0
0.683 0.028 0.558 0.084 -1.0
0.877 0.358 0.131 0.292 -1.0
```

In order to validate that our results are correct what we did was to apply the values of ν, w, γ and s from the training test. This can be done because we know that we can evaluate a data point against a hyperplane with the following formula:

$$w^T x + \gamma = 0$$

where if the result is bigger than 0 (above the hyperplane), the fifth column should be a 1, otherwise -1. This result that we got has been compared with the real value and at this way, we could validate the performance of our test set.

We have that γ and s come from the solution but the ν is a variable and affects at the quality of the solution. Therefore, we need to fix the ν so that we can evaluate our performance. To do so, we are going to pick the one that gives us the best performance for our training set which in this particular case has been $\nu = 4$ that gives us an accuracy of 94% in the training set and a 93% of accuracy in the test set.

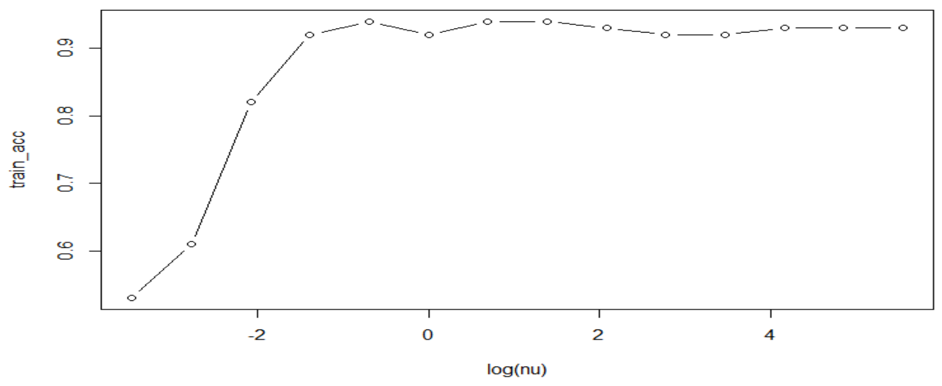

Figure 2: ν vs train precision

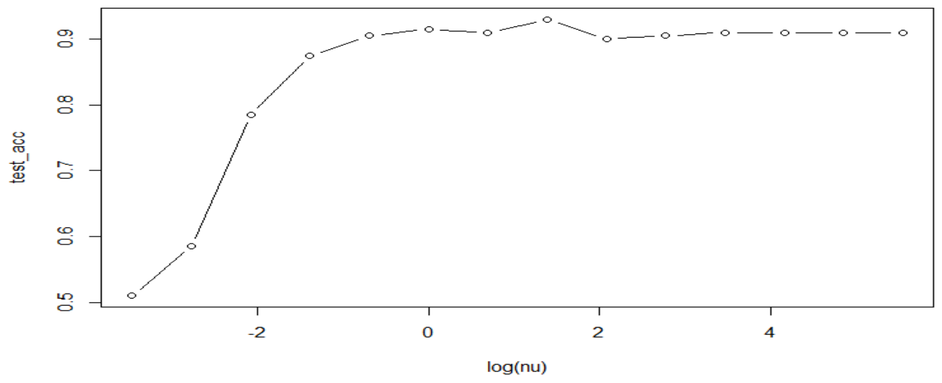

Figure 3: ν vs test results

4.1.1 Hyperplanes of the Dual and Primal forms

As we previously mentioned, the SVM is a complex problem and the dual form is nothing more than the Langrangian dual of the primal. Therefore, it can be demonstrated that by solving the dual, we obtain the same solution than by using the primal. This is because once we find our λ, we can compute the corresponding hyperplane by using the following formulas

$$w = A^T Y \lambda$$
$$\gamma = w^T x_i \frac{-1}{y_i}$$

Note that to calculate γ, the point that we use from x_i has to be a support vector which means that it has to fulfill ($0 \leq \lambda_i$) and ($0 = s_i$).

This can be found in our results since the dual and the primal give us the same exact results.

4.2 Test phase with Iris Dataset

On this section we will use our AMPL model with a dataset that is not generated by the *gensvmdat* scipt. In particular, the iris dataset, which is well known on the data science community as one of the most used datasets for the begginers.

The data on this dataset consists of a version of the Iris dataset which has 100 samples of 2 iris species(Iris virginica and Iris versicolor). Each one of the samples consists in 4 features: length and width of the sepals and petals in centimeters. We will use our primal form to calculate the hyperplane as both formulations return the same w and γ.

We will make the same test that we made on the previous dataset. Iterate on ν to see which value can obtain the best accuracy.

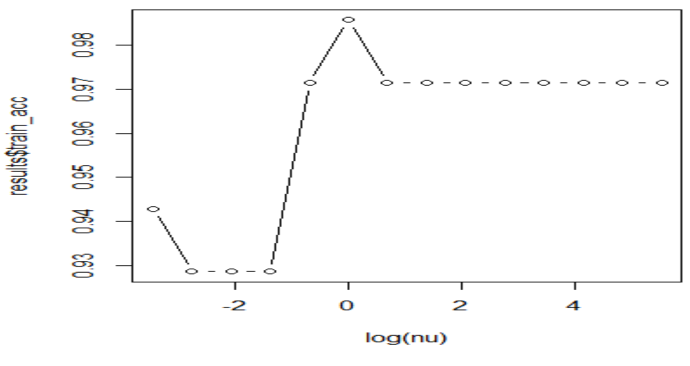

Figure 4: Train iris accuracy

This plot above is showing the train accuracy on the iris dataset. For the value of $\nu = 1$, we obtain a 98,57% of accuracy as the best value.

In the next step, we will compute the test accuracy and analyze if our model has a good behaviour. We could omit all the other values of ν and only execute the test for the value of 1 as it was the best result, but we will also include all the other values just to see the evolution of accuracy for ν values on the test data.

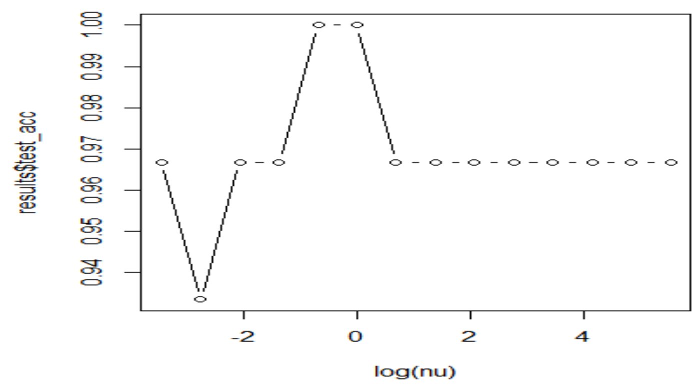

Figure 5: Test iris accuracy

We can see on the plot that for 2 ν values we obtain a 100% accuracy. This values are $\nu = 0$ and $\nu = 1$.

For low values of ν, we obtain less accuracy, but if we increase too much the value, we will see that is not the most optimum accuracy, with the plot, we can know what are the best ν values for our particular set of data, this doesn't mean that for every dataset this value of nu is going to get the best results but only in our data sample.

5 Conclusions

As aforementioned, implementing the dual leads to the same solution as the one obtained in the primal. Therefore, we can decide which form should be used depending on which one is more convenient to solve a specific problem.

To classify a point using the primal, we must compute the scalar product $w^T x$ in order to see where is the data point. This can have a very large computational cost in case of having very large number of dimensions. However, when we are using the dual form, it is way more efficient, specially if we are working with low amount the support vectors as they make λ have a value of $\lambda = 0$. This will make the dual form require less kernel evaluations and reducing the computational cost a lot when having large inputs.

In addition, as we now have a scalar product only involving data vectors, we can kernelize the SVM by applying the kernel trick.

Finally, it can be seen that AMPL has some problems with numerical types. We had some issues such as float problems where one number equal to another, would be considered different(F.e. we had $256.00 < 256.00000$ was retrieving a true). This affects our results from the dual formulation on the small values of ν(0.03125-0.5) since there are some points that where being considered as support vectors when they should not, giving a wrong gamma and causing to obtain a different hyperplane, the difference was quite low on most of the cases.

6 AMPL Code

Dual

```
#########
#Dual run
#########
reset;

cd('C:/Users/Meyerhofer/Desktop/UNI/OTDM/SVM-project/');
model 'dual.mod';
data 'data.dat';
#display nu;

option solver "ampl_mswin64/cplex";
solve;

var w{i in 1..numVars};

for{i in 1..numVars} {
   let w[i] := sum{j in 1..numPoints}lambda[j]*y[j]*x[j,i];
}

var gamma;

for{i in 1..numPoints} {
   if lambda[i]>=0.0001 && (lambda[i] + 0.0001) < nu then
      { let gamma := (1/y[i]) - (sum{j in 1..numVars} w[j]*x[i,j]);
      #display i;
      #display gamma;
      };

}


#display w;
#display gamma;
#display lambda;
display w > 'output/dual/w.txt';
display gamma > 'output/dual/gamma.txt';
display lambda > 'output/dual/lambda.txt';


var results {i in 1..numPoints} = if (sum {j in 1..numVars} w[j] *
    x[i,j] + gamma >= 0) then 1 else -1;

var correct;
for{k in 1..numPoints} {
   if results[k] == y[k] then
      { let correct := correct + 1;}
}

#display results;
display correct/numPoints;

display correct/numPoints > 'output/dual/acc_train.txt';

#validation, we use same w's and gamma. we change x,y's

param numPoints_test = 30;


var test{m in 1..numPoints_test, n in 1..dimensions};
var x_test{m in 1..numPoints_test, n in 1..numVars};
var y_test{m in 1..numPoints_test};

read {m in 1..numPoints_test, n in 1..dimensions} test[m,n] <
    "data/iris-test.dat";

for{i in 1..numPoints_test} {
   for{j in 1..numVars} {
      let x_test[i,j] := test[i,j];
   }
   let y_test[i] := test[i,numVars+1];
}

var results_test {i in 1..numPoints_test} = if (sum {j in
    1..numVars} w[j] * x_test[i,j] + gamma >= 0) then 1 else -1;

var correct_test;
```

```
for{k in 1..numPoints_test} {
   if results_test[k] == y_test[k] then
      { let correct_test := correct_test + 1;}
}


#display results_test;
#display correct_test/numPoints_test;
display correct_test/numPoints_test > 'output/dual/acc_test.txt';
#########
#Dual mod
#########
param numPoints;
param numVars;
param dimensions;

param nu;
param train{m in 1..numPoints, n in 1..dimensions};
param x{m in 1..numPoints, n in 1..numVars};
param y{m in 1..numPoints};
param K{i in 1..numPoints, j in 1..numPoints} = sum{k in
    1..numVars}x[i,k]*x[j,k];

var lambda{m in 1..numPoints} >= 0, <= nu;

maximize obj_function:
   sum{m in 1..numPoints} lambda[m] - (1/2)*sum{mi in 1..numPoints,
       mj in 1..numPoints}
       lambda[mi]*y[mi]*lambda[mj]*y[mj]*K[mi,mj];

subject to Contraint1:
   sum{m in 1..numPoints} lambda[m]*y[m] = 0;
```

Primal

```
#########
#Primal run
#########
reset;
```

```
cd('C:/Users/Meyerhofer/Desktop/UNI/OTDM/SVM-project/');
model 'primal.mod';
data 'data.dat';
#display nu;

option solver "ampl_mswin64/cplex";
solve;

#display w;
#display s;
#display gamma;


display w > 'output/primal/w.txt';
display s > 'output/primal/s.txt';
display gamma > 'output/primal/gamma.txt';


var results {i in 1..numPoints} = if (sum {j in 1..numVars} w[j] *
    x[i,j] + gamma >= 0) then 1 else -1;

var correct;
for{k in 1..numPoints} {
   if results[k] == y[k] then
      { let correct := correct + 1;}
}

#display results;
display correct/numPoints;

display correct/numPoints > 'output/primal/acc_train.txt';

#validation, we use same w's and gamma. we change x,y's

param numPoints_test = 30;


var test{m in 1..numPoints_test, n in 1..dimensions};
var x_test{m in 1..numPoints_test, n in 1..numVars};
var y_test{m in 1..numPoints_test};
```

```
read {m in 1..numPoints_test, n in 1..dimensions} test[m,n] <
    "data/iris-test.dat";

for{i in 1..numPoints_test} {
   for{j in 1..numVars} {
      let x_test[i,j] := test[i,j];
   }
   let y_test[i] := test[i,numVars+1];
}

var results_test {i in 1..numPoints_test} = if (sum {j in
    1..numVars} w[j] * x_test[i,j] + gamma >= 0) then 1 else -1;

var correct_test;
for{k in 1..numPoints_test} {
   if results_test[k] == y_test[k] then
      { let correct_test := correct_test + 1;}
}


#display results_test;
display correct_test/numPoints_test > 'output/primal/acc_test.txt';
display correct_test/numPoints_test;
#########
#Primal mod
#########
param numPoints;
param numVars;
param dimensions;

param nu;
param train{m in 1..numPoints, n in 1..dimensions};
param x{m in 1..numPoints, n in 1..numVars};
param y{m in 1..numPoints};

var s{m in 1..numPoints} >= 0;
var w{n in 1..numVars};
var gamma;
```

```
minimize obj_function:
   (1/2) * sum{n in 1..numVars} w[n]^2 + nu*sum{m in 1..numPoints}
      s[m];

subject to Constraint1 {m in 1..numPoints}:
   y[m]*((sum{n in 1..numVars} w[n]*x[m,n]) + gamma) + s[m] >= 1;
```

Data.dat

```
data;

param numPoints := 70;
param numVars := 4;
param dimensions := 5;
read nu < "data/nu.txt";


read {m in 1..numPoints, n in 1..dimensions} train[m,n] <
    "data/iris-train.dat";

for{i in 1..numPoints} {
   for{j in 1..numVars} {
      let x[i,j] := train[i,j];
   }
   let y[i] := train[i,numVars+1];
}
```