Universitat Politècnica de Catalunya

Implementation and Evaluation of Support Vector Machines in AMPL

MATHEMATICAL OPTIMIZATION

Data Science and Engineering

David González and Ángel Morales

May 2025

1 Introduction

Support Vector Machines (SVMs) are a widely used tool in machine learning for binary classification tasks. In this project, we focus on implementing and understanding SVMs from the perspective of mathematical optimization.

Our implementation is based on the AMPL modelling language, which we use to define and solve both the primal and dual formulations of the SVM optimization problem. For the optimization itself, we rely on CPLEX solver. In addition, Python is used for data preprocessing, dataset generation, and various auxiliary tasks.

Throughout the project, we apply our implementation to several datasets with different properties. The first dataset is synthetic and generated using `gensvmdat`, a tool provided by our instructor. The second dataset contains real-world medical data related to diabetes diagnosis. The third and final dataset is also synthetic, but it consists of non-linearly separable observations, which requires the application of the kernel method to achieve satisfactory classification.

In each experiment, we divide the dataset into a training set and a test set, with the test set comprising 10% of the total observations. We evaluate the model's performance by computing both training and testing accuracy, and we analyze the results to assess how well the SVM generalizes to unseen data.

2 Mathematical formulation

We consider a binary classification problem with m data points $x_1, x_2, \dots, x_m$, where each $x_i \in \mathbb{R}^n$. These vectors are stored in a matrix A of size $m \times n$. Each point is associated with a label $y_i \in \{-1, +1\}$ indicating its class. The goal of a Support Vector Machine is to find a hyperplane that separates the two classes with the maximum possible margin. This central hyperplane is defined by the equation:

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

and the two margin boundaries are defined by:

$$w^T x + \gamma = +1 \quad \text{and} \quad w^T x + \gamma = -1.$$

In the ideal case where the data is linearly separable, all positive-labeled points lie beyond the $+1$ margin, and all negative-labeled points lie beyond the -1 margin, with no points falling between the margins. In this case, each point satisfies the constraint:

$$y_i(w^T x_i + \gamma) \geq 1.$$

However, to handle cases where the data is not perfectly separable, we introduce slack variables $s_i \geq 0$ for each point, which allow for violations of the margin constraints. The new condition becomes:

$$y_i(w^T x_i + \gamma) \geq 1 - s_i.$$

These slack variables measure the degree to which each point violates the margin:

- If $s_i = 0$, the point is correctly classified and lies outside the margin.
- If $0 < s_i \leq 1$, the point is inside the margin but still on the correct side of the hyperplane.
- If $s_i > 1$, the point is misclassified (on the wrong side of the hyperplane).

The margin width is $\frac{2}{\|w\|}$, so minimizing $\frac{1}{2}\|w\|^2$ corresponds to maximizing the margin. Additionally, we penalize large slacks to avoid misclassification. Therefore, the primal SVM optimization problem to solve is the following:

$$\begin{aligned} \text{minimize} \quad & \frac{1}{2}\|w\|^2 + \nu \sum_{i=1}^{m} s_i \\ \text{subject to} \quad & y_i(w^T x_i + \gamma) \geq 1 - s_i, \quad s_i \geq 0 \quad \forall i. \end{aligned}$$

The term ν is a regularization hyperparameter that controls the trade-off between maximizing the margin and minimizing classification errors. A larger value of ν prioritizes reducing misclassified points (by penalizing slack variables more heavily) over increasing the margin width.

2.1 Dual problem

Instead of solving the primal formulation of the SVM, we can work with its dual version. The dual problem is another optimization problem that is mathematically equivalent to the primal, but has some advantages: it is always convex, it can be more efficient when the number of features is large, and it enables to use the kernel trick for non-linear classification. The dual formulation for the SVM is the following:

$$\begin{aligned} \text{maximize} \quad & \sum_{i=1}^{m} \lambda_i - \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m} \lambda_i \lambda_j y_i y_j K_{ij} \\ \text{subject to} \quad & 0 \leq \lambda_i \leq \nu \quad \forall i, \text{ and } \sum_{i=1}^{m} \lambda_i y_i = 0. \end{aligned}$$

In this formulation, the variables to optimize are the Lagrange multipliers $\lambda_1 \dots, \lambda_m$, while w and γ do not appear explicitly. However, we can easily recover them from the solution. The value of w can be reconstructed as:

$$w = \sum_{i=1}^{m} \lambda_i y_i x_i.$$

To compute the bias term γ, we select a support vector from the dataset, that is, a point x_i that lies exactly on one of the margin hyperplanes. Such points correspond to Lagrange multipliers that satisfy $0 < \lambda_i < \nu$. For any of these points, the margin condition becomes an equality: $w^T x_i + \gamma = y_i$. We can then isolate γ and compute it as:

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

3 AMPL Implementation

Based on the mathematical background previously explained, the corresponding AMPL model code for the primal problem, contained in the file `svm.mod`, is:

```
param m;                        # Number of training examples
param n;                        # Number of features
param A{1..m, 1..n};            # Feature matrix
param y{1..m};                  # Labels (+1 or -1)
param v = 1;                    # Regularization parameter

var w{1..n};                    # Weight vector
var gamma;                      # Bias term
var s{1..m} ¿= 0;               # Slack variables

minimize objective:
    0.5 * sum{i in 1..n} w[i] 2  + v * sum{i in 1..m} s[i];

subject to correctclass{i in 1..m}:
    (sum{j in 1..n} w[j]*A[i,j] + gamma) * y[i] + s[i] ¿= 1;
```

The AMPL code for the dual problem, contained in the file `svm_dual.mod`, is:

```
param m;                             # Number of training examples
param K{i in 1..m, j in 1..m};       # Kernel matrix: Kij = xi T xj
param y{1..m};                       # Labels (+1 or -1)
param v = 1;                         # Regularization parameter

var lambda{1..m} ¿= 0, ¡= v;         # Dual variables

maximize dualobj:
    sum{i in 1..m} lambda[i]
    - 0.5 * sum{i in 1..m, j in 1..m}
      lambda[i] * lambda[j] * y[i] * y[j] * K[i,j];

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

4 Experiment 1: Synthetic dataset generated with gensvmdat

4.1 Dataset generation

For the first experiment we will use the **gensvmdat** tool provided by the professor, which generates a synthetic dataset of randomly distributed points with four features with labels that are suitable to be separated linearly with our implementation of the SVM. We configured the tool to generate 200 points with the seed 42 with the following command:

```
$¿ ./gensvmdat raw_data_ex1.txt 200 42
```

The resulting file contains 200 labelled points, and some of them are marked with an asterisk (*) to indicate that they are intentionally designed to be hard or impossible to classify. From the full dataset, we select the first 180 points for training the model and preserve the last 20 for testing. AMPL cannot read this file directly, so we made a simple python function that generates the .dat files excluding the last 20 observations, both for the primal and dual problems. It also removes the asterisks but returns a list with the indices of the points that had them.

4.2 Solving the primal problem

Once the dataset is fully prepared, we solve the primal formulation of the SVM executing the "**svm.run**" file in AMPL. In this case, we set the hyperparameter ν is equal to 1, although we will explore other values later to penalize misclassified points more strongly. The model returns the optimal weight vector w, the bias term γ and the slack variables s_i for all 180 training points:

$$w = (2.6410,\ 2.1910,\ 3.0778,\ 2.3872) \qquad \gamma = -5.1717$$

$$\text{Margin width} = \frac{2}{\|w\|^2} = 0.0742$$

The figure below displays the slack variables for all training points. Points originally marked with an asterisk, the ones hard to classify correctly, have their slacks marked in red.

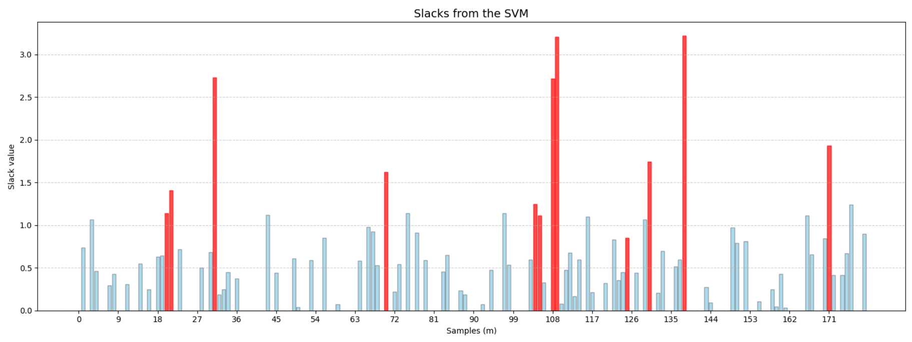

Figura 1: Slacks from the SVM

In the plot we see that many slacks are zero and the ones different from zero tend to be below 1, which indicates that the corresponding points are in the correct side of the hyperplane. We also see that all the red ones exceed 1, therefore they are in the wrong side. Some of them even have a slack greater than 2, so they are past the margin band.

4.3 Testing and accuracy

From the previous figure we see that, from all the 180 points, only 19 are badly classified. Therefore, we got a training accuracy of 89.44 %.

The 20 test points are used to evaluate the model's performance on unseen data. Although slack variables are only defined for training points, we can still easily compute the classification quality on the test points. Given the point x_i with the label y_i, the expression $r_i = (w^T x_i + \gamma)y_i$ is similar to the slacks and also indicates if the point is correctly classified. In this case, $r_i \geq 1$ indicates that the point is correctly classified and outside the margin band, $r_i \in (0, 1)$ indicates that it is in the correct side but inside the margins, $r_i \in (-1, 1)$ indicates that it is inside the margins but badly classified and $r_i \leq -1$ indicates that the point is badly classified and outside the margins.

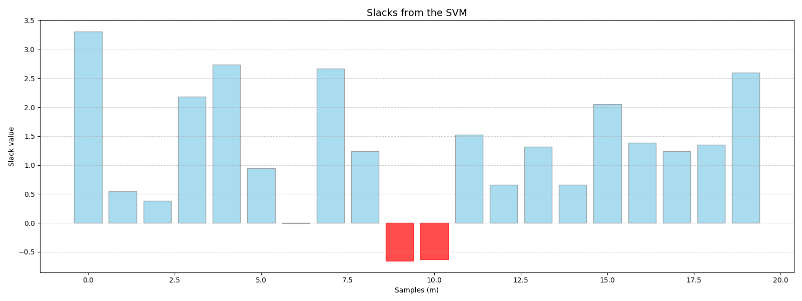

Figura 2: r values for test points

After evaluating and plotting the r values for all the points, we see that only the two red points, the ones with the asterisk, have $r_i < 0$ and therefore are badly classified. All the others are correctly classified and 12 of them have $r_i > 1$ and therefore are outside the margins bands. Since only 2 out of the 20 points are badly-classified, the model has a testing accuracy of 90%, which suggests that it performs very well overall, even in the presence of challenging data.

4.4 Solving the dual problem

After solving the dual problem, we get the Lagrange multipliers $\lambda_1 \ldots, \lambda_m$, from which we can reconstruct the value of w and γ. The resulting value for w is almost the same as the previously obtained:

$$w = \sum_{i=1}^{m} \lambda_i y_i x_i = (2.6410,\ 2.1910,\ 3.0778,\ 2.3872)$$

Most of the Lagrange multipliers are either 0 or ν (the regularization parameter), except for two values, those corresponding to indices 62 and 157. These are the **support vectors**, the only points that lie exactly on the margin and therefore satisfy the hyperplane equations. We can use them to isolate and compute the value of γ:

$$x_{62} = (0.168,\ 0.102,\ 0.574,\ 0.728),\ \ y_{62} = -1 \quad \Longrightarrow \quad \gamma = y_{62} - w^T x_{62} = -5.17169$$

$$x_{157} = (0.116, 0.828, 0.975, 0.440),\ \ y_{157} = -1 \quad \Longrightarrow \quad \gamma = y_{157} - w^T x_{157} = -5.17169$$

In both cases, we obtain the same value for γ, which perfectly matches the result from the primal problem. This confirms the consistency between both formulations, dual and primal, and highlights that everything is working as it should.

The solving time of the algorithm in the primal formulation was 0.004686 seconds, with 11 iterations. In contrast, the dual formulation took 0.006383 seconds and required 14 iterations, suggesting that solving the primal is more efficient in this case.

4.5 The impact of parameter ν

We now test the algorithm for different values of ν to evaluate its influence on the model's behavior. High values of ν prioritize reducing the slack variables, which leads to fewer classification errors but also a narrower margin. On the other hand, lower values of ν allow for wider margins and tolerate more misclassifications.

We run the algorithm for $\nu = 0.1, 1,$ and 10, and evaluate the training and testing accuracy, the proportion of points correctly classified and outside the margin, and the width of the margin. The results are shown in the following table:

Taula 1: Effect of parameter ν on performance and margin

	Train accuracy	Train acc. outside margin	Test accuracy	Test acc. outside margin	Margin width
$\nu = 0.1$	90.56%	13.89%	85.0%	20.0%	0.3358
$\nu = 1$	89.44%	50.00%	90.0%	60.0%	0.0742
$\nu = 10$	91.67%	66.67%	90.0%	75.0%	0.0247

We observe that as ν increases, both training and testing accuracy remain relatively stable. However, the proportion of points outside the margin significantly increases, and the margin width decreases. This indicates that larger values of ν penalize misclassifications more heavily while allowing less flexibility in the decision boundary, resulting in a narrower margin.

5 Experiment 2: Real-world dataset

5.1 Obtaining the dataset

For our second experiment we explore how our SVM implementation performs on real-world data. We selected the *Pima Indians Diabetes Database*, which is publicly available on Kaggle. This dataset was created by the National Institute of Diabetes and Digestive and Kidney Diseases, and its goal is to predict whether a patient has diabetes based on several diagnostic measurements.

The dataset contains 691 observations. All individuals are female patients of Pima Indian heritage, aged 21 or older. Each observation corresponds to one person and includes eight medical features, along with a binary outcome variable indicating whether the individual has diabetes or not.

To simplify the problem, we will only take into account four features: **Glucose**, **BMI**, **Age**, and **Blood-Pressure**. After preprocessing and visualizing the data, we expect those variables to allow us to make linear separation.

5.2 Solving the Primal Problem

As with the synthetic dataset, we began by splitting the data into a training set (90%) and a test set (10%). A Python script processes the `raw_diabetes.csv` file and generates two `.dat` files compatible with AMPL: one for the primal formulation and one for the dual.

After training the model using the primal formulation, we obtained the following parameters:

$$w = (0.026773,\ 0.065770,\ 0.019861,\ -0.011386), \quad \gamma = -5.909629$$

$$\text{Margin width} = \frac{2}{\|w\|^2} = 359.2866$$

In this case, the margin width is significantly larger than in the synthetic dataset. This is mainly because in the synthetic case, the features were scaled to be between 0 and 1, whereas in this real-world dataset the feature values are much larger.

5.3 Testing and accuracy

When examining the slack variables s_i, we observe that many of them are either zero or less than one, indicating correct classification. However, 158 out of the 691 training examples have $s_i > 1$, which means they are misclassified. This corresponds to a misclassification rate of 22.865%, resulting in a training accuracy of 77.135%.

We then applied the trained model to the test set and computed the r_i values. Among the 77 test observations, only 16 were misclassified, therefore we have a testing accuracy of 79.221%. This value is very close to the training accuracy, suggesting that the model generalizes well and does not exhibit over-fitting or under-fitting. The figure below shows the distribution of r_i values in the test set; those below zero correspond to misclassified points:

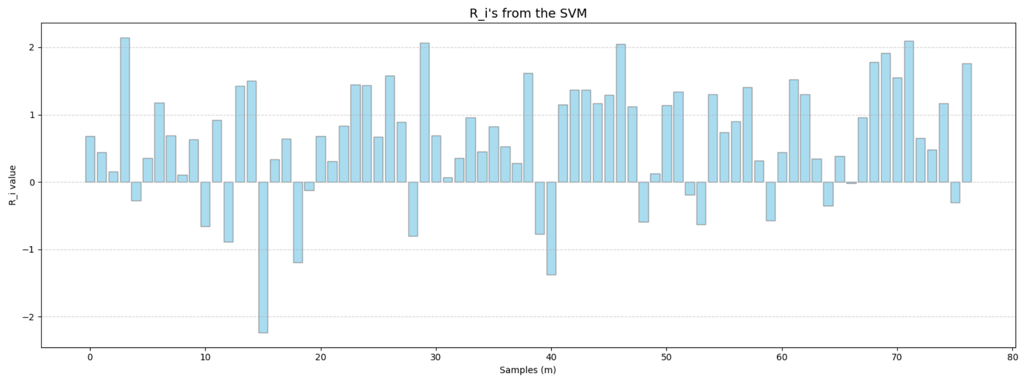

Figura 3: r values for test points

5.4 Solving the Dual Problem

After solving the dual formulation on the same dataset, we obtained the Lagrange multipliers λ_i. As in the previous case, we can reconstruct the vector w from the dual solution. As expected, the result is nearly identical to the one obtained from the primal problem:

$$w = \sum_{i=1}^{m} \lambda_i y_i x_i = (0.026785,\ 0.0657681,\ 0.019873,\ -0.011397)$$

We then used a Python script to identify the support vectors, defined as the training samples whose Lagrange multipliers satisfy $0 < \lambda_i < \nu$. These support vectors can then used to compute the bias term γ. In this case, we found five of those points, and after averaging all the values obtained for γ, we got:

$$\gamma = -5.9097144$$

which closely matches the value obtained from the primal formulation.

Overall, the dual formulation achieves consistent results with the primal one, reinforcing the equivalence between both formulations. In this case, the primal formulation took a solving time of 0.0163 seconds and did 20 iterations, while the dual formulation spent 0.0220 seconds and did 24 iterations. Again, the primal formulation had a better performance.

6 Experiment 3: Non-linearly Separable Dataset

6.1 Mathematical Background: The Kernel Method

The Kernel trick is used when the data can't be separated by a linear hyperplane. The fundamental idea behind the Kernel trick is to map the original data into another space, called feature space, using a transformation function $\phi(x)$. After this step we expect the data to be linearly separable and therefore we can apply the standard SVM method. The formulation of the dual optimization problem becomes:

$$\text{maximize} \quad \sum_{i=1}^{m} \lambda_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} \lambda_i\ \lambda_j\ y_i\ y_j\ \phi(x_i)^T \phi(x_j)$$

$$\text{subject to} \quad 0 \le \lambda_i \le \nu \quad \forall i, \text{ and } \sum_{i=1}^{m} \lambda_i y_i = 0.$$

The term $\phi(x_i)^T \phi(x_j)$ represents as a measure of similarity between $\phi(x_i)$ and $\phi(x_j)$. The key idea is to define this dot product through a kernel function $K(x_i, x_j)$ and compute it directly, instead of explicitly defining the initial function $\phi(x)$.

In our implementation, we use the Gaussian Kernel (or RBF). Similar to the regular dot product, this kernel takes its maximum value when $x_i = x_j$ and decreases as they get further apart. It is defined as:

$$K(x_i, x_j) = \exp\left(-\beta \|x_i - x_j\|^2\right),$$

To incorporate the kernel trick into the dual SVM formulation, we simply replace the dot product matrix with the kernel matrix K. In our implementation, this corresponds to modifying the input `.dat` file to contain the kernel matrix rather than the dot product matrix.
When classifying a new point or when evaluating the accuracy of the model on a test set, we use the decision function $f(x)$, defined as:

$$f(x) = \sum_{i=1}^{m} \lambda_i y_i K(x_i, x) + \gamma.$$

The function $f(x)$ is positive in the +1 region and negative in the -1 region. The margin corresponds to the interval $[-1, 1]$. Thus, for correct classification, the value $r_i = f(x_i) \cdot y_i$ must be positive, and ideally greater than 1 to ensure the point lies outside the margin.

Finally, to recover the γ parameter (the bias term), we proceed as before: we find a support vector point with $\lambda_i \in (0, \nu)$ and we isolate γ in the decision function:

$$\gamma = y_i - \sum_{j=1}^{m} \lambda_j y_j K(x_j, x_i).$$

6.2 Obtaining the dataset

For this experiment we will restrict the data to a 2-dimensional input space. This will allow us to clearly visualize the data and the decision function obtained by the algorithm. We decided to create a dataset by ourselves with a little python script using the pygame library, which allows us to place points interactively and ensure that the distribution is not linearly separable. All points are confined to the square $[0, 1] \times [0, 1]$.

The final dataset consist of 62 points, randomly split into 55 training samples and 7 test samples. In the following figure, training points are shown as circles while test points as squares. The points with label +1 are coloured in pink while the ones with label -1 are colored in green.

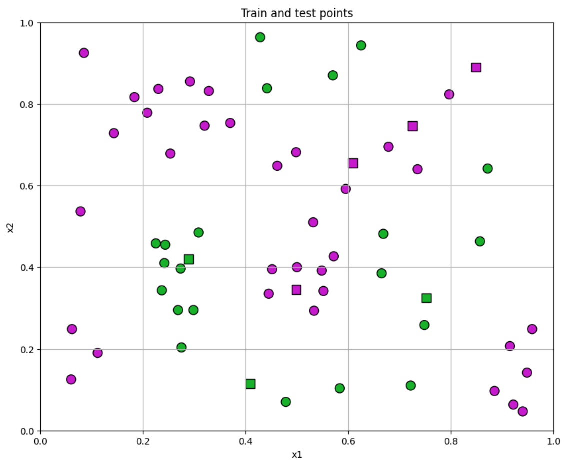

Figura 4: Non-linearly separable dataset

6.3 Solving the dual formulation and defining f

We will set the hyper-parameters to $\beta = 20$ and $\nu = 1$. After running the AMPL script `svm_dual.run`, we obtain the optimal values of the Lagrange variables λ_i. As in previous experiments, the support vectors are identified as those for which $\lambda_i \in (0, \nu)$. In this case, a significant number of points qualify as support vectors. Computing the average value of γ using these support vectors yields $\gamma = 0.23761$. With this, the decision function $f(x)$ is fully defined and can now be evaluated on any input x.

6.4 Visualization of the results and accuracy

Given that the data has only two dimensions, we can visualize the classification results and decision regions directly. We will plot the points the same way as before, but we will also represent the decision function. To do so, we construct a grid, and for each small region of the grid, we evaluate the function. We paint the corresponding point into one of four colors depending on its class and if it's inside the margins. The following figure shows the result:

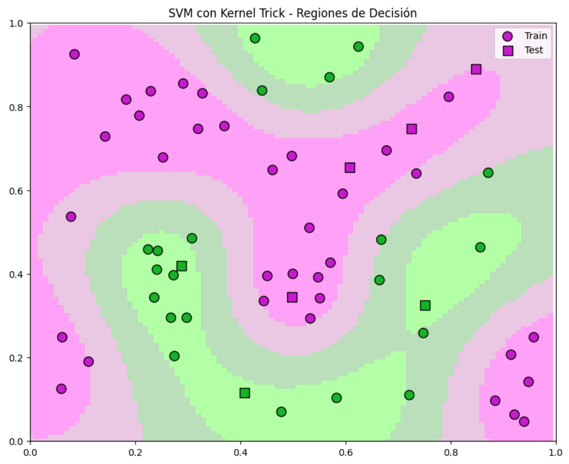

Figura 5: Results of the SVM with the Kernel trick

As shown in the figure, all training and testing points are correctly classified. A small number of points lie within the margin, but none are misclassified. Therefore, both training and test accuracies reach 100%. The results clearly illustrate the effectiveness of the kernel method in learning complex, non-linear decision boundaries that would be impossible to capture with a standard linear SVM. In fact, applying a linear SVM to this dataset would result in poor performance, as the data is not linearly separable in the original input space and therefore many points would be misclassified. Furthermore, the model exhibits neither under-fitting nor over-fitting, demonstrating a good balance between flexibility and generalization.

7 Conclusion

Throughout this project, we implemented and evaluated the primal and dual formulations of the SVM classifier using AMPL and Python. We validated the correctness and equivalence of both formulations on synthetic and real-world datasets. Additionally, by incorporating the Gaussian kernel, we extended our implementation to handle non-linearly separable data, achieving perfect classification in the final experiment.

The results confirm that the dual formulation is essential when applying kernel methods, while the primal formulation can be more efficient in linearly separable settings. Our methodology achieves consistent and robust performance, providing insights into the mathematical structure of SVMs and their practical application in classification tasks.