Support vector machine classifier in AMPL

Benet Ramio and Pau Amargant

GCED, UPC

June 10, 2023

Abstract

We present a support vector machine classifier implemented in AMPL, a mathematical optimization language. We implement and compare the primal and dual formulations, and the linear and Gaussian Kernels, on synthetic and real datasets. We show that the linear kernel can achieve high accuracy on linearly separable data, and that the primal formulation is faster than the dual. On non-linearly separable data, the Gaussian kernel proves necessary and is able to properly separate the two classes. We use the AMPL Python API and scikit-learn tools to streamline the model fitting and demonstrate the power and flexibility of AMPL for solving SVM and other optimization problems.

Seeds: 42

Contents

1 Introduction

Support vector machines are a powerful and widely used technique for supervised classification problems. However, implementing and solving SVMs can be sometimes challenging and treating them as a blackbox can often lead to bad results.

In this project, we aim to implement and solve a SVM using AMPL, a mathematical optimization language that allow us to express the SVM problem concisely and solve it through one of the many solvers available nowadays. The model is then tested against multiple datasets in order to study its performance when in front of various situations; two synthetics datasets and a real-world dataset.

Throughout the project both the Primal and Dual formulations of the SVM problem are implemented. A linear and a Gaussian kernel are used to classify linearly and non linearly separable data respectively.

2 Theory and Modelling

We begin by introducing the core concepts of support vector machines and how they have been implemented using the AMPL optimitzation modelling language.

2.1 Primal formulation

Support vector machines are a kind of discriminative classification model which aims to classify observations into two classes . The input space consists of a set of observations and labels $x_i \in \mathbb{R}^n$, $y_i \in \{0, 1\}$ $i = 1 \dots m$. A separating hyperplane $w^\intercal x + \gamma = 0$ is fitted to the data in order to separate the observations into two classes, together with two parallel planes separated from it by a margin. Given a set of observations we aim to find the plane which maximizes the distance between the two parallel planes $w^\intercal x + \gamma \geq +1$ and $w^\intercal x + \gamma \leq 1$. To allow for misclassifications a soft margin must be introduced. This is done by using slack variables $s_i \geq 0$, $i = 1, \dots, m$ w. To account for this change the objective function is modified as to also minimize the classification errors modified by a parameter $\nu \in \mathbb{R}$. Equation 2.1 shows the primal SVM problem.

$$\min_{w,\gamma,s \in \mathbb{R}^{N+1+m}} \quad \frac{1}{2}||w||_2 + \nu \sum_{i=1}^{m} s_i \tag{1}$$

$$\text{s.t} \quad y_i(w^\intercal \phi(x_i) + \gamma) \geq 1 \qquad i = 1, \dots, m \tag{2}$$

$$s_i \geq 0 \qquad i = 1, \dots, m \tag{3}$$

2.2 Dual formulation

The SVM problem can be expressed in its dual formulation using its dual function, as shown in **??** furthermore, given that it is a convex differentiable problem *Wolfe duality* allows us to solve it through the dual formulation. Throught manipulation the problem can be reduced into:

$$\max_{\lambda} \quad \lambda^\intercal e - \frac{1}{2}\lambda^\intercal Y A A^\intercal Y \lambda \tag{4}$$

$$\text{s.t.} \quad \lambda^\intercal Y e = 0 \tag{5}$$

$$0 \leq \lambda \leq \nu \tag{6}$$

$$\tag{7}$$

This formulation only finds the optimal λ values, therefore the hyperplane parameters w and γ have to be retrieved using the fact that $w = A^\intercal Y \lambda = \sum_{i=1}^{m} \lambda_i y_i \phi(x_i)$. We can obtain γ by identifying a support vector.

$$w = \sum_{i=1}^{m} \lambda_i y_i \phi(x_i) \tag{8}$$

$$\gamma = \frac{1}{y_i} - w^\intercal \phi(x_i) \quad i \in SV \tag{9}$$

The w expression divided into misclassified points, support vectors and non-binding points can be used to interprete the importance of the ν value. Furthermore, non-binding points don't affect the parameters of the hyperplane. The importance that the misclassified vectors have on the solution is directly proporcional to ν. With a large value of ν misclassifications have a larger penalty (as shown in 2.1) and they have a larger weight on calculating the hyperplane w.

2.3 Gaussian Kernel

The previous formulations perform well in front of linearlyly separable dataset. When working with non-linearly separable data a kernel an improvement can be achieved by using a mapping function $\phi : X \subseteq \mathbb{R}^n \to F \subseteq \mathbb{R}^N$, usually with $N >> n$ and training the model in the higher dimensional space. Thanks to the nature of SVMs a kernel can be used without needing to calculate the values of X in the higher dimensional space by the use of the so called *Kernel trick*. This allows us to use infinitely dimensional kernels such as the Gaussian Kernel [**?**]. We define the kernel function $K(x_i, x_j) = \phi(x_i)^\intercal \phi(x_j)$ and subtitute into the dual formulation:

$$\max_{\lambda} \quad \lambda^\intercal e - \frac{1}{2}\lambda^\intercal Y K Y \lambda \tag{10}$$

$$\text{s.t.} \quad \lambda^\intercal Y e = 0 \tag{11}$$

$$0 \leq \lambda \leq \nu \tag{12}$$

In this case the values of w can not be obtained, as it is infinitely dimensional, but we can still do predictions by taking advatange of:

$$w^\intercal x_k + \gamma = \sum_{i=1}^{m} \lambda_i y_i \phi(x_i)^\intercal \phi(x_k) + \gamma = \sum_{i=1}^{m} \lambda_i y_i \ker(x_i, x_j) + \gamma \tag{13}$$

Where the kernel is given by:

$$\ker(x_i, x_j) = e^{-\frac{1}{2\sigma^2}||x_i - x_j||^2} \tag{14}$$

2.4 Methodology

In 2 the AMPL models used have been shown and explained. However, it is necessary to discuss how the *AMPL API* [**?**] and *scikit-learn* have been utilized to streamline the model fitting process and analyze the results more easily. The AMPL API, known as *amplpy* is an interface that gives access the features of AMPL from within Python. This integration enables the usage of AMPL models while being able to input and extract data using popular Python libraries such as NumPy and pandas DataFrames. This means that the data preprocessing and post-processing tasks can be performed efficiently in Python, while the model generation and solver interaction are handled by AMPL with minimal additional overhead. It is worth noting that because of using amplpy we have used IPOPT in place of the commercial solvers .

Furthermore, the AMPL API has been combined with the capabilities of *scikit-learn* estimators. The SVM model has been implemented into a custom estimator with the `fit(X,y)` and `predict(X)` methods. By implementing the SVM as a custom estimator within scikit-learn, it becomes compatible with scikit-learn tools such as GridSearch and CrossValidation, enabling the search for optimal classifier parameters.

3 AMPL codes

Both formulations and the problem have been implemented as an AMPL model. The dual formulation has been implemented using both a linear and a Gaussian kernel. As can be observed included code, the three models begin by declaring the parameters and variables of the model as specified by the SVM formulation. The parameters are passed to the models using the AMPL Python API. In the primal model the variables which are optimized are w, γ and s while in the dual formulation λ is optimized.It is worth noting that in the case of the gaussian kernel, the kernel matrix is prepcalculated in python and then passed to AMPL.

3.1 Codes

Listing 1: Primal formulation

```
# PARAMETERS
param m;  # number of rows
param n; # number of columns
param nu; # nu parameter
param X {1..m,1..n}; # data matrix
param Y {1..m}; # labels

# VARIABLES
var w{1..n};
var gamma;
var s{1..m};

# OBJECTIVE function
minimize fx:
   1/2*sum{i in 1..n}w[i]**2 + nu*sum{i in 1..m}s[i];

# CONSTRAINTS
subject to gx1 {i in 1..m}:
   Y[i]*((sum{j in 1..n}X[i,j]*w[j])+gamma)+s[i] >= 1;

subject to gx2 {i in 1..m}:
   s[i] >= 0;
```

Listing 2: Dual linear formulation

```
# PARAMETERS
param m;  # number of rows
param n; # number of columns
param nu; # nu parameter
param X {1..m,1..n}; # data matrix
param Y {1..m}; # labels

# VARIABLES
var lambda{1..m} >= 0, <= nu;

# OBJECTIVE function
maximize q:
   sum{i in 1..m}lambda[i]
   -1/2*sum{i in 1..m, j in 1..m}lambda[i]*Y[i]*
   lambda[j]*Y[j]*(sum{k in 1..n}X[i,k]*X[j,k]);
```

```
# CONSTRAINTS
subject to hx:
   sum{i in 1..m}lambda[i]*Y[i] = 0;
```

Listing 3: Dual kernel formulation

```
# PARAMETERS
param m;  # number of rows
param n; # number of columns
param nu; # nu parameter
param sigma; # sigma parameter
param X {1..m,1..n}; # data matrix
param Y {1..m}; # labels
param K {1..m,1..m}; # kernel matrix

# VARIABLES
var lambda{1..m} >= 0, <= nu;

# OBJECTIVE function
maximize q:
   sum{i in 1..m}lambda[i]
   -1/2*sum{i in 1..m, j in
      1..m}lambda[i]*Y[i]*lambda[j]*Y[j]*K[i,j];

# CONSTRAINTS
subject to hx:
   sum{i in 1..m}(lambda[i]*Y[i]) = 0;
```

4 Model evaluation

4.1 Synthetic linearly separable

We have tested the syntetic linearyly separable data for the primal, dual linear and dual kernel problems for various sizes (m) and ν's which can be found in the appendix Listing 4.

In Figure 1, the logaritmic time for each dataset size is presented. We can see that, generally, as the size grows, the time needed to solve the problem also increases for all the formulations. Also, we can observe that both dual problems exhibit similar execution times, which are significantly greater than that of the primal problem. From appendix Figures 6 and 7 we can see that the primal problem time increases linearly while for the dual problems, quadratically., making it computationally expensive for large datasets.

The quadratic increase in complexity in the dual formulation could be attributed to the combined effect of two factors: matrix operations and constraint evaluation. As the dataset size grows, the number of variables and constraints in the dual problem increases, leading to larger matrices that need to be computed and manipulated. This increase in matrix size results in increased computational complexity. Additionally, for each data point in the dataset, the constraints need to be evaluated, and as the dataset size increases, the number of constraints to evaluate also grows.

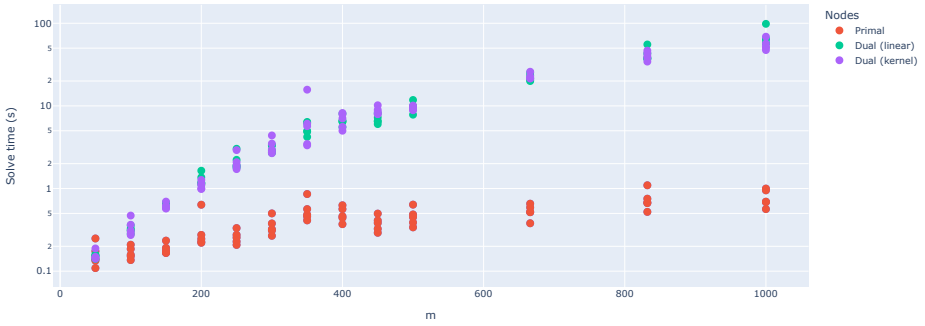

Figure 1: *Solve time (log) for different dataset sizes and formulations*

We continue analyzing the iterations. As we can see in Figure 2, we can observe that the primal formulation generally requires more iterations than both dual approaches, even though it takes less time to execute. This indicates that each iteration in the primal is much faster than in the dual.

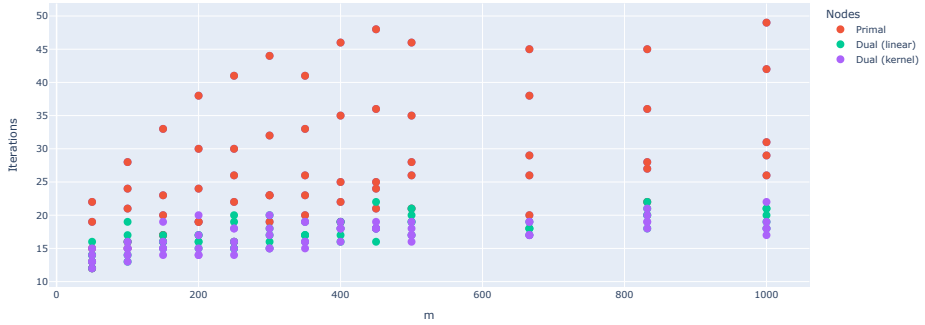

Figure 2: *Iterations for different dataset sizes and formulations*

In terms of accuracies, it is evident from the heatmap shown in Figure 3 that both the primal and dual linear models yield identical accuracies for each dataset size and value of ν. This observation serves as evidence that the primal plane is indeed recovered from the dual solution [1]. However, in the case of the kernel solution, since the data is linearly separable in the input space, applying a feature space transformation generally leads to worse results. Nevertheless, there are some exceptional scenarios, such as when ν is set to 0.5 with dataset sizes of 100 and 450, where the performance is better with the kernel trick.

[1] In the following section, we will verify it sowing the values of w and γ of a concrete dataset with fixed m and ν

Figure 3: *Accuracy heatmap*

4.1.1 Analysis of a concrete dataset

In this section, we are going to analyse a concrete dataset with fixed size and ν in order to verify that the planes from the primal formulation is correctly recovered solving the dual linear problem [2].

In Table 1 we can see the output from our primal and dual linear formulations as well as the *sklearn* SVC model. It is observed that the values of γ and w are the same for both the primal and dual linear formulations, while for the *sklearn* model γ and w are not exactly the same but with very small difference [3]. We can say that the tree solutions yield the same solution, which can be verified by noting that the accuracies are also the same. Additionally, as we have already seen when comparing different dataset sizes, it is evident that the primal formulation requires more iterations to solve the optimization problem. However, despite the higher number of iterations, the primal solution exhibits a shorter execution time compared to the dual solution. This indicates that each iteration in the primal formulation is significantly faster than in the dual formulation. In the case of *sklearn* model, it iterates way more with a very quicker execution time thanks to using *libsvm*, which provides an efficient implementation of SVM classifiers.

type	accuracy	γ	w	it	time (s)
primal	0.98	-6.0461	[2.5755,3.0139,3.071,3.4197]	22	0.286294
dual_linear	0.98	-6.0461	[2.5755,3.0139,3.071,3.4197]	15	1.567805
sklearn SVC	0.98	-6.0445	[2.5746,3.0135,3.0705,3.4186]	241	0.002696

Table 1: *Accuracy, plane, iterations, and execution time for each formulation and sklearn SVC model*

[2] We have chosen $m = 250$ (size) and $\nu = 0.5$ since it gave great performance in terms of accuracy and takes a reasonable execution time to execute

[3] The small difference might be due to numerical precision or a different implementation of the SVM, as scikit-learn uses libsvm

4.2 Real linearly separable

Next we will see test the tree formulations of the SVM problem in a real linearly separable dataset. We choose the famous Iris dataset, which is a widely recognized dataset in the field of machine learning and statistics. Introduced by Ronald Fisher in 1936, it comprises 150 samples of iris flowers, each characterized by four features: sepal length, sepal width, petal length, and petal width. These measurements are recorded in centimetres, and each sample is labelled with one of three species: setosa, versicolor, or virginica.

We can see the plot of the first two principal components of PCA in Figure 4. The setosa specie is yellow while the versicolor and virginica are blue.As we can see, the setosa specie is linearly separable to the other species, so we will test our formulations as well as the *sklearn* SVC model to distinguish between this two groups. Before starting, it is relevant to point out that the choice of ν in this case is irrelevant, since the data is linearly separable without misclassification. In this case, a hard margin formulation would also work.

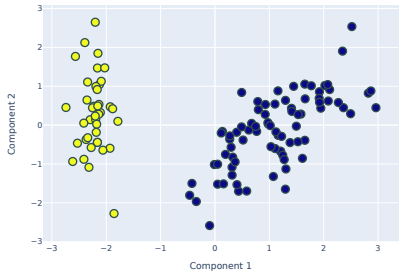

Figure 4: *First two principal components of PCA of the iris dataset with the setosa specie in yellow while the versicolor and virginica in blue*

Table 2 presents the results of three different formulations and the *sklearn* SVC model with their corresponding outcomes. Remarkably, due to the linear separability and absence of misclassifications in our data, we achieved a perfect accuracy for all formulations and *sklearn* model. Furthermore, it is noteworthy that the plane obtained from the primal formulation matches precisely with the one derived from the dual formulation, indicating the successful implementation of the optimization problems. As in the case of synthetic generated data, this plane is no exactly the same but very close to the one obtained with the *sklearn* model, probably due to internal optimizations the model does in order to have a faster execution time. However, a difference we can notice from the analysed case in section 4.1.1 is that the *sklearn* model does much fewer iterations than all the other formulations. In the case of the dual Gaussian solution, the accuracy remains perfect.

type	accuracy	γ	w	it	time (s)
primal	1.0	1.451	[-0.046,0.5217,-1.0032,-0.4642]	26	0.230969
dual_linear	1.0	1.451	[-0.046,0.5217,-1.0032,-0.4642]	15	0.392686
dual_gaussian	1.0	-0.265	—	14	0.667636
sklearn SVC	1.0	1.447	[-0.0458, 0.5222,-1.0029,-0.4641]	6	0.000990

Table 2: *Accuracy, plane, iterations, and execution time for each formulation*

4.3 Linearly non separable

In order analyse the performance of the classifier in a linearly nonseparable dataset we use the Swiss roll synthetic dataset. It is a three-dimensional datast with the shape of a rolled.up sheet of paper, resembling a rolled-up Swiss pastry. We have used the python implemenation, which generates the set of N points in 3D space together with a set of values Y for the points. The values of Y can be interpeted as being continous when the swiss roll is unrolled and labels for the points can be set by using expression 15, in which a is an arbitrary threshold. In our case the median has been used.

If we fit a support vector classifier to the data using a linear kernel it becomes apparent that the model is not appropiate for the data. Using $\nu = 1$ an accuracy of 0.592 is obtained. As shown in figure 5 the separating plane is not able to properly classify the points.

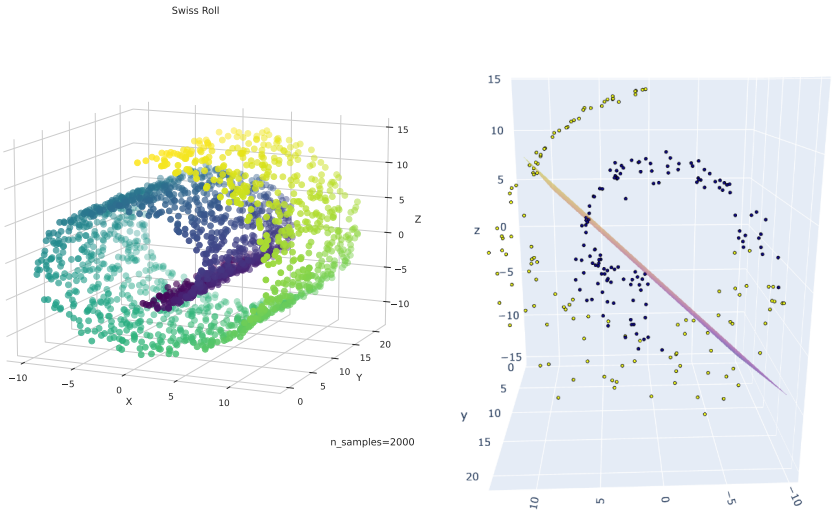

Figure 5: *Comparison of Swiss roll and labeled swiss roll with the fitted separating plane*

On the other hand, when using the dual formulation of the support vector classifier together with a gaussian kernel, also with $\nu = 1$, a much higher accuracy of $0,992$ is obtained. This proved that the gaussian kernel function allows us to classify non linearly separable data.

The optimal ν parameter can be found by training the model with different ν values, performing cross-validation to estimate the generalization error, and choosing the ν value which

gives us the higher accuracy. In our case the maximum accuracy achieved is 0.992 when $\nu = 5$, but it is worth noting that as shown in table 3 similar results are obtained with other ν values. The number of iterations and execution time is relatively similar in all cases.

Furthermore, using the scikit-learn SVC classifier and radial basis function kernel, the same accuracy has been obtained.

ν	0.1	0.5	1	5	10
Accuracy	0.492	0.964	0.992	0.992	0.992
Time (S)	30.54	33.38	34.64	31.92	33.52
Iterations	17	12	15	15	16

Table 3: *Accuracy values for different values of ν using a train-test split*

5 Conclusions

After having fitted the various versions of the Support Vector Classifier model on different datasets, we have been able to determine the performance of each model under different circumstances.

We have observed that when in front of a linearly separable dataset, the model is able to properly separate the two classes even with few training data. When using the proper hyperparameter, the three formulations are able to achieve an accuracy of more than 90%, with ν being important when having less than 200 samples. On the other hand, despite doing more iterations, the primal model has proved much faster than the dual formulation.

When in front of a non-linearly separable data, the Gaussian kernel has proved superior. The linear kernel has not been able to separate the data, while the Gaussian kernel has achieved an accuracy of 0.992. Nonetheless, we have also observed the increased computing time required.

The ν parameter has only proved relevant on non-linearly separable data or when misclassified points are present, as in those cases the soft margin is necessary.

When it comes to computing performance, *scikit-learn* has been shown to be an order of magnitude faster due to utilising *libsvm* which is specifically targeted at solving SVM problems while the *IPOPT* is a solver for a broader class of optimization problems.

Overall, the results have shown us the theoretical models translate to a real environment and how depending on the nature of the data the kernel and hyperparameters must be properly chosen.

6 Appendices

6.1 Relevant plots

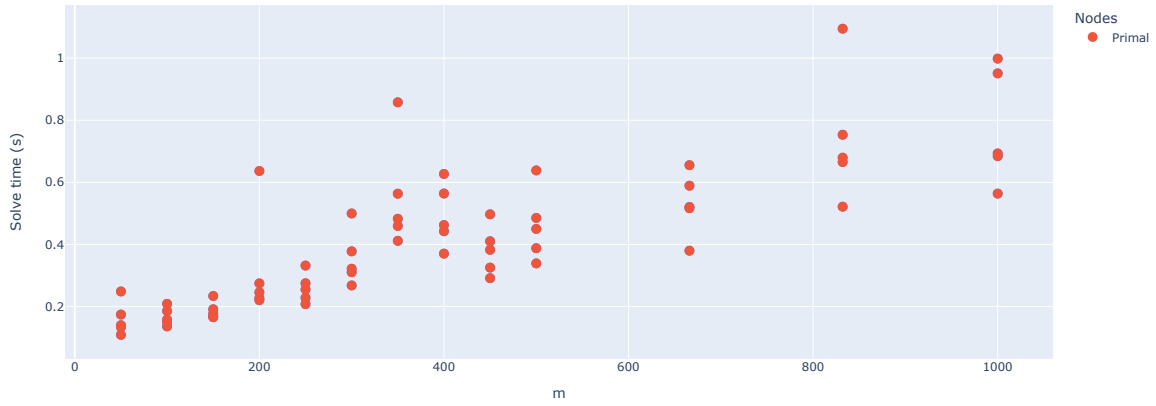

Figure 6: *Primal formulation execution time for different dataset sizes*

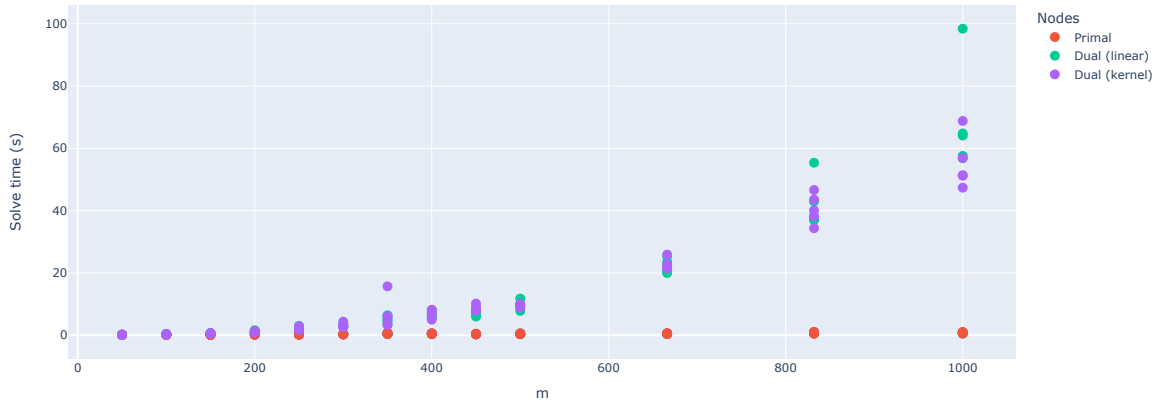

Figure 7: *Solve time for different dataset sizes and formulations*

6.2 Relevant equations

$$t_i = \begin{cases} 1 & y_i < a \\ -1 & y_i \geq a \end{cases} \tag{15}$$

6.3 Listings

```
m_values = [50, 100, 150, 200, 250, 300, 350, 400, 450, 500, 666, 832, 1000]
nu_values = [0.5, 1, 2, 5, 10]
```

Listing 4: *Sizes (m) and ν values tried to analyse the formulations*