Polytechnic University of Catalonia

Data Science and Engineering

Support Vector Machine in AMPL

Alina Castell Blasco - 49188689S

June 2023

Contents

1 Introduction

The goal of this project is to implement the primal and dual quadratic formulations of the support vector machine classifier with AMPL. As studied in class, the SVM classifier aims to find two parallel hyperplanes so that they can separate points of two different classes; as well as maximising the distance between these two hyperplanes and minimising the classification error.

The SVM will be applied to the generated dataset *gensvmdat* and to the *diabetes* dataset found in a OpenML repository mentioned in the references. For both these linearly separable datasets the primal and dual problems will be implemented; as well as for a linearly nonseparable dataset which will also be implemented with a RBF or Gaussian kernel.

2 SVM implementation with AMPL

This section provides a concise overview of three key problems: the primal problem, the dual problem, and the kernel RBF problem. It further includes the implementation details for each problem using AMPL (.mod files).

2.1 Primal Formulation

So, for Primal Problem with linearly separable data in feature space the SVM formula is the following:

$$\min_{(w,\gamma,s)\in R^{N+1+m}} \frac{1}{2} w^T w + \nu \sum_{i=1}^{m} s_i$$

$$s.to\ y_i(w^T \phi(x_i) + \gamma) s_i \geq 1,\ i = 1, ..., m$$

$$s_i \geq 0,\ i = 1, ..., m$$

The aim is to classify m points $x_i \in R^n, i = 1, ..., m$ such that every point x belongs to one of the two classes linearly separated by the plane $w^T x + \gamma = 0$. Thus, the points will be classified in

$$if\ w^T x + \gamma > 0\ then\ x \in C1$$

$$if\ w^T x + \gamma \leq 0\ then\ x \in C2$$

The distance between these two parallel hyperplanes is to be maximised; where w is the normal to the separation hyperplane and γ determines its location with respect to the origin.

Now, this formula is transformed into standard matrix form:

$$\min_{(w,\gamma,s)\in R^{N+1+m}} \frac{1}{2} w^T w + \nu e^T s$$

$$s.to\ -Y(Aw^T + \gamma e) - s + e \leq 0$$

$$-s \leq 0$$

Being

$$A = \begin{bmatrix} \phi(x_1)^T \\ \vdots \\ \phi(x_m)^T \end{bmatrix} \in R^{m \times n}, Y = \begin{bmatrix} y_1 & & \\ & \ddots & \\ & & y_m \end{bmatrix} \; and \; e = \begin{bmatrix} 1_1 \\ \vdots \\ 1_m \end{bmatrix}$$

If the Lagrange multipliers are added to this primal formulation in matrix form, the dual formulation is generated; this theoretical formulation is shown in the next section.

In the next section the implementation of the code will be shown.

2.2 Dual Formulation

The dual formulation in standard matrix form is:

$$\max_{w,\gamma,s,\lambda,\mu} \frac{1}{2} w^T w + \nu e^T s + \lambda^T (-Y(Aw + \gamma e) - s + e) - \mu^T s$$

$$\begin{aligned} w - (\lambda^T Y A)^T &= 0 \\ \lambda^T Y e &= 0 \\ \nu e - \lambda - \mu &= 0 \\ \lambda \geq 0, \; \mu &\geq 0 \end{aligned}$$

Now, the previous formula is replaced by the definitions $w = (\lambda^T Y A)^T$ and $\nu e - \lambda = \mu$, which leads to a matrix form that is again transformed into a scalar one:

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

$$\begin{aligned} \sum_{i=1}^{m} \lambda_i y_i &= 0 \\ 0 \leq \lambda_i \leq \nu, i &= 1, ..., m \end{aligned}$$

Being $K = AA^T$.

The following code shows the implementation in AMPL of the Primal and Dual problem (they are written in the same '.mod' file, since they will be executed by the same '.run' file):

```
# Parameters
# Points and dimension
param m >= 1, integer;
param n >= 1, integer;
param nu >= 0;
```

```
6
7  # Train parameters
8  param y {1..m};
9  param A {1..m,1..n};
10
11 # Test parameters
12 param m_te >= 1, integer;
13 param n_te >= 1, integer;
14 param y_te {1..m_te};
15 param A_te {1..m_te,1..n_te};
16
17 # Variables
18 var gamma;
19 var w {1..n};
20 var s {1..m} >= 0;
21 var lambda {1..m} >= 0, <= nu;
22
23
24 ## PRIMAL ##
25 # Problem
26 minimize primal: 1/2*sum{i in 1..n} (w[i]^2) + nu * sum{j in 1..m} s[j];
27 subject to consp1 {j in 1..m}:
28       -y[j] *(sum{i in 1..n} A[j,i] * w[i] + gamma) - s[j] + 1 <= 0;
29 subject to consp2 {j in 1..m}: -s[j] <= 0;
30
31
32 ## DUAL ##
33 # Problem
34 maximize dual: sum{j in 1..m} lambda[j] - 1/2 * sum{i in 1..m} (sum{j in 1..m}
35   (lambda[i] * y[i] * lambda[j] * y[j] * sum{k in 1..n} (A[i,k]*A[j,k])));
36 subject to consd1: sum{j in 1..m} (lambda[j] * y[j]) = 0;
```

The '.run' file generated in order to check if both their formulations provide the same separation hyperplane, can be found in the Annex section. This '.run' file also indicated the solver used which is *cplex* solver. In this file the SV classifier accuracy is also evaluated, as well as the prediction of hits, accuracy and the confusion matrix.

2.3 Kernel RBF Formulation

The purpose of dealing with a Gaussian Kernel is to improve the data classification precision. This kernel allows a classification of linearly non-separable data as input. Therefore, data is transformed to another input space using the map function $\phi(x)$, which maps the data into this space. The Gaussian Kernel expression is:

$$K(x,y) = \phi(x)^T \phi(y) = e^{-\frac{||x-y||^2}{2\sigma^2}}$$

While the explicit expression of $\phi(x)$ may not always be known, this is not a problem because the Gaussian Kernel is enough for the necessary calculations in classification. This can be observed in the following expressions (where x^* represents the point from our initial dataset that we want to classify):

$$w = \sum_{i=1}^{m} \lambda_i y_i \phi(x_i) \Rightarrow w^T \phi(x) = \sum_{i=1}^{m} \lambda_i y_i \phi(x_i)^T \phi(x^*) = \sum_{i=1}^{m} \lambda_i y_i K(x_i, x^*)$$

$$\gamma = \frac{1}{y_i} - w^T \phi(x_i) = \frac{1}{y_i} - \sum_{j=1}^{m} \lambda_j y_j \phi(x_j)^T \phi(x^*) = \frac{1}{y_i} - \sum_{j=1}^{m} \lambda_j y_j K(x_j, x_i)$$

Here is the AMPL implementation of the Kernel Formulation of the Dual problem:

```
# Parameters
# Points and dimension
param m >= 1, integer;
param n >= 1, integer;
param nu >= 0;
param sigma2 = 100/2;
param y {1..m};
param A {1..m,1..n};

# Test parameters
param m_te >= 1, integer;
param n_te >= 1, integer;
param y_te {1..m_te};
param A_te {1..m_te,1..n_te};

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

# Problem
maximize kernel: sum{i in 1..m} lambda[i] - (1/2)*sum{i in 1..m}
      (sum{j in 1..m} (lambda[i]*y[i]*lambda[j]*y[j]* exp(-sum{k in 1..n}(
      ((A[i,k]-A[j ,k])^2) / (2*sigma2) ))));
subject to cons1: sum{i in 1..m} lambda[i]*y[i] = 0;
```

3 Analysis of the results for each dataset

This section contains a brief introduction to each dataset studied, the results obtained shown in the output and their analysis. It also makes reference to the execution time of each formulation, this time has been calculated with a python code that can be found in the Annex section.

3.1 Gensvmdat dataset

The dataset used for this section has been generated through the function provided in class "*./gensvmdat file.dat p seed*" with the values $p = 100$ and $seed = 12345$. To the generated file, some transformations are made in order to adapt it to an AMPL readable format (see the python code into the attached documentation). Some of this transformations include removing the guiding '*' as linearly non-separable. The resulting data file contains two matrices and vectors for training and test data of size 75 and 25 samples, respectively. As for the optimum ν value to obtain maximum accuracy, some trials have been made: if $\nu = 0.8$ accuracy is 88%, if $\nu = 2$ accuracy is 88% and

finally, from the value $\nu = 4$ forward accuracy becomes 92%. In this next table the relation between times, accuracy and ν values are shown:

ν	0.8	2	4	8	10
Time	0.03s	0.03s	0.03s	0.03s	0.03s
Accuracy	88%	88%	92%	92%	92%

Table 1: Relation between execution time, accuracy and ν values of *gensvmdat* dataset.

Thus, since the last three values coincide in relation between execution time and accuracy, the chosen value is $\nu = 10$.

The next output results shows the comparison of hyper-planes, gamma values and accuracy values between both formulations:

```
Primal                  Dual
w [*] :=                wd [*] :=
1  5.25418              1  5.25418
2  5.89321              2  5.89321
3  4.61371              3  4.61371
4  5.23367              4  5.23367

gamma = -10.417         gammad = -10.417
accuracy_p = 0.92       accuracy_d = 0.92
```

It can be observed that the hyper-planes coincide, as well as gamma values and accuracy values; which means that both formulations behave identically, at least with this particular dataset. Also note that both their objective function coincide with *objective 217.5770724*. The next values of the prediction confusion matrix based on the test dataset further analyses the accuracy:

```
Confusion Matrix
1 1   11    (TruePos)
1 2    1    (FalsePos)
2 1    1    (FalseNeg)
2 2   12    (TrueNeg)
```

The information extracted is that the proportion of True Positives and False Positives is very similar to the True Negatives and False Negatives one, respectively. The favorable part is that an accuracy value of 92% is great.

3.2 Repository dataset

This original dataset contains information about some parameters that may result in producing diabetes and the actual outcome, if the person is or is not diabetic. The goal is to achieve the maximum accuracy on the prediction based on the parameters. The variables that this study perceives as factors for diabetes are contained in the matrix 'A' written in, their actual content is:

- Number of Pregnancies
- Level of Glucose
- Blood Pressure
- Skin Thickness
- Level of Insulin
- BMI
- Diabetes Pedigree Function
- Age

The dataset contains 768 observations, in order to transform the 'csv' file into a format that AMPL is able to read, the data has been divided randomly into 650 samples for training and 118 for test. The previous factors are contained in the matrix 'A', as well as the ν value. The best ν is obtained by trying different values within 0.5 and 10 shown in the table. Each execution time value is a mean of three trials per ν. After several trials, the conclusion obtained is that the accuracy value is maintained despite changing the ν value. Therefore, the value taken will be the one with less execution time in mean; being the $\nu = 2$.

ν	0.8	2	4	8	10
Time	1.02s	0.96s	0.97s	0.97s	1.01s
Accuracy	78%	78%	78%	78%	78%

Table 2: Relation between execution time, accuracy and ν values of *diabetes* dataset.

Comparison of hyper-planes, gamma values and accuracy values between both formulations:

```
Primal                  Dual
w [*] :=                wd [*] :=
1  -0.0881721           1  -0.0881721
2  -0.0301229           2  -0.0301229
3   0.0101876           3   0.0101876
4   0.0042158           4   0.0042158
5   0.000743078         5   0.000743078
6  -0.0702191           6  -0.0702191
7  -0.000642371         7  -0.000642371
8  -0.0116693;          8  -0.0116693;

gamma = 6.69471         gammad = 6.69471
accuracy_p = 0.788136   accuracy_d = 0.788136
```

As expected, both formulations generate the exact same hyper-plane, the exact same γ and same accuracy as well. Moreover, both objective functions coincide *objective 305.9173049*. An accuracy

of the 78.8136% is a considerably good precision of the optimum, this value is further explained in the next confusion matrix:

```
Confusion Matrix
1 1   73    (TruePos)
1 2   19    (FalsePos)
2 1    6    (FalseNeg)
2 2   20    (TrueNeg)
```

The interpretation of the results in terms of the diabetes dataset is that 73% of the people who conducted the study were correctly classified as diabetic and 20% correctly classified as non-diabetic. However, 19% (almost the same size as the previous group) was wrongly classified as diabetic; thus, the actual proportion of non-diabetic is 39%. Finally, the 6% of them were classified as non-diabetic when actually they were.

In addition, the kernel formulation has been executed for this dataset as a way to obtain more information (with a slightly modified '.run' file). The result has been fairly wrong. The accuracy obtained is of the 36.44% which means that the proportion of incorrectly classified diabetic people is higher than the correct one. This formulation definitely does not work for linearly separable data, the next section shows that it does work for linearly non-separable data.

3.3 Linearly Non-separable dataset

For this part, the dataset used has been generated with a Python code that makes use of the 'make_swiss_roll()' function from the 'sklearn.datasets' module. The generated data is be saved in a file named 'swissroll.dat'. This file is structured to include the training and test matrices ('A', 'A_te') that contain 3 variables with 100 observations each. Additionally, the training and test vectors ('y', 'y_te') that contain a single column of 100 observations classified in 1 or -1 using a criterion based on the mean value. See the Python code in the Annex section.

In order to prove the efficiency of the Kernel formulation with linearly non-separable data, its classification precision will be compared with that of the primal and dual formulations. For the kernel formulation different values of nu have been tried as well. The $\nu = 0.8$ gives an accuracy of 94%, $\nu = 2$ gives one of 92%, $\nu = 4$ gives one of 94% and when $\nu = 10$ is maintained at 94%; this is shown in the next table:

ν	0.8	2	4	8	10
Time	0.45s	0.44s	0.45s	0.45s	0.45s
Accuracy	94%	92%	94%	94%	94%

Table 3: Relation between execution time, accuracy and ν values of *non-linear* dataset.

The best relation between execution time and accuracy is taken by the ν values 4,8 and 10; so, the value that will be taken for the next outputs will be $\nu = 10$.

The execution has been solved with the 'cplex' solver, indicated in the Kernel's '.run' file, shown in the Annex section; the result is the following:

```
gammad = -0.129093
Prediction kernel:
hits = 95
accuracy = 0.95 (= hits/m = 95/100)
Confusion Matrix
1 1   45    (TruePos)
2 1    5    (FalseNeg)
2 2   50    (TrueNeg)
```

The result shows a great accuracy obtained by the Kernel formulation; the transformation of the data input space allows a classification of this data almost perfect. As for the confusion matrix it can be extracted that 45% of the data is correctly classified to positive, 50% of it to negative and only 5% is misplaced in the negative side and non have been misplaced to the positive.

Output of the execution of primal and dual formulations:

```
Primal                  Dual
w [*] :=                wd [*] :=
1  -0.0053653           1  -0.0053653
2   0.0117737           2   0.0117737
3  -0.127943;           3  -0.127943;

gamma = -0.388594       gammad = -0.389782
accuracy_p = 0.67       accuracy_d = 0.67
```

As for the comparison between primal and dual, it can be seen that both hyper-planes and accuracy values are exactly the same and the γ values are very similar (they differ in the third decimal). Regarding the comparison between the lineal and non-lineal SVM formulation, the next confusion matrix shows some interesting information:

```
Confusion Matrix
1 1   31    (TruePos)
1 2   14    (FalsePos)
2 1   19    (FalseNeg)
2 2   36    (TrueNeg)
```

Both the confusion matrix and accuracy values worsen with respect to the non-lineal (Kernel-RBF) formulation. This time there is a considerable amount of sample incorrectly classified, the number of false positive is half the size of the true positives; which could have huge repercussions if used for serious matters.

Consequently, it can be considered that the use of a non-linear Kernel can be highly beneficial in cases where the data is not separable. This is due to the fact that data points, x, are represented through a mapping function that alters their distribution, making them more separable.

4 Conclusion

Taking into account the topics discussed throughout this project, it is crucial to thoroughly analyze the working dataset before implementing the SVM problem. It has been observed that both the linear and dual models of Support Vector Machines yield the same results, with the same separating hyperplane. While a significant distinction between using a linear or nonlinear SVM has also been observed.

Consistent with the theoretical classes, as the dataset size increases, it becomes easier to compute the dual problem, which yields identical outcomes to the primal problem. The linear SVMs are easier to implement since they do not require building a kernel. They perform well with linear datasets and require less effort; however, when applied to nonlinear data, the accuracy significantly drops. For instance, the accuracy obtained using the linearly non-separable dataset by the primal SVM was of 67%, which is substantially lower than the 95% accuracy achieved with the kernel SVM. This approach has allowed to classify data that would otherwise be challenging to solve using the primal or dual methods.

In summary, the key lesson learned from this project is the necessity of understanding the nature of the data to implement the most appropriate SVM approach. Additionally, the knowledge about SVMs and their various formulations, applications and analysis has expanded.

5 References

Dataset extracted from Github Repository.

Documents used for coding in AMPL: Command Guide.

Information about SVM extracted from the TowardsDataScience Article.

6 Annex

This section contains the run files for the Primal, Dual and Kernel problems; as well as the Python code used for generating linearly non-separable data and the one used for calculatin execution time.

Primal and Dual '.run' file:

```
reset;
model primdual.mod;
data diabetes.dat;
option solver cplex;
print "Here goes Primal!";
problem PRIMAL: primal, w, gamma, s, consp1, consp2;
solve PRIMAL;
display w, gamma;
print " Here goes Dual!";
problem DUAL: dual, lambda, consd1;
solve DUAL;


```

```
# Extraction of w from the dual solution for lineal Kernel
param wd {1..n};
let {i in 1..n} wd[i]:= sum{j in 1..m} lambda[j]*y[j]*A[j,i];
display wd;

# Obtain SV classifiers
param sv default 0;
param class {1..m} default 0;
for{i in 1..m} {
  if lambda[i] >= 10^(-7) and nu - lambda[i] >= 10^(-7) then let sv:= sv+1;
  if lambda[i] >= 10^(-7) and nu - lambda[i] >= 10^(-7) then let class[i]:=1;
}
display sv;
display class;

# Extraction of gamma from the dual solution
param gammad;
let gammad:= 1/sv*(sum{i in 1..m} class[i]* (y[i]- sum{j in 1..m} lambda[j]*y[j]*
      (sum{k in 1..n} A[j,k]*A[i,k])));
display gammad;


# Validation

# Prediction of kernel
print "Prediction primal:";
param prediction {1..m_te};

# Classification of target values 1, -1
for{j in 1..m_te} {
  if gamma + sum{i in 1..n_te} w[i]*A_te[j,i] >= 0
    then let prediction[j] := 1;
  else let prediction[j] := -1;
}
#display prediction, y_te;

# Accuracy
param hits_p default 0;
for {i in {1..m_te}} {
  if prediction[i] == y_te[i] then
    let hits_p := hits_p + 1;
}
display hits_p;
param accuracy_p = hits_p / m_te;
display accuracy_p;


# Prediction of kernel
print "Prediction dual: ";

# Classification of target values 1, -1
for{j in 1..m_te} {
  if gammad + sum{i in 1..n_te} wd[i]*A_te[j,i] >= 0
    then let prediction[j] := 1;
  else let prediction[j] := -1;
}
# display prediction, y_te;

```

```

# Accuracy
param hits_d default 0;
for {i in {1..m_te}} {
  if prediction[i] == y_te[i] then
    let hits_d := hits_d + 1;
}
display hits_d;
param accuracy_d = hits_d / m_te;
display accuracy_d;


# Definition of confusion matrix
param confmat{1..2,1..2} default 0;
for {i in 1..m_te}{
  if prediction[i] == y_te[i] and prediction[i]=1 then let confmat[1,1]:= confmat
    [1,1] + 1;
  }
for {i in 1..m_te}{
  if prediction[i] == y_te[i] and prediction[i]=-1 then let confmat[2,2]:= confmat
    [2,2] + 1;
  }
for {i in 1..m_te}{
  if prediction[i]*y_te[i]==-1 and prediction[i]=1 then let confmat[1,2]:= confmat
    [1,2] + 1;
  }
for {i in 1..m_te}{
  if prediction[i]*y_te[i]==-1 and prediction[i]=-1 then let confmat[2,1]:= confmat
    [2,1] + 1;
  }

# Display of confusion matrix
printf"Confusion Matrix. confmat[1,1] is TP, confmat[2,2] is TN, confmat[1,2] is FP
    and confmat[2,1] is FN";
display confmat;
```

Kernel-RBF '.run' file:

```
reset;
model kernel_prob.mod;
data diabetes.dat;
option solver cplex;
print " Here goes Kernel!";
problem KERNEL: kernel, lambda, cons1;
solve KERNEL;
#display lambda;


# Obtain SV classifiers
param sv;
param class {1..m} default 0;
for{i in 1..m} {
  if lambda[i] >= 10^(-7) and nu - lambda[i] >= 10^(-7) then let sv:= sv+1;
  if lambda[i] >= 10^(-7) and nu - lambda[i] >= 10^(-7) then let class[i]:=1;
}
display sv;
display class;

```

```
21
22 # Generate Gaussian Kernel RBF
23 param l default 0;
24 param r default 0;
25 param norm default 0;
26 for {i in 1..(m+m_te)}{
27 for {j in 1..(m+m_te)}{
28 for {k in 1..n}{
29   if i <= m
30   then let l := A[i, k];
31   else let l := A_te[i-m, k];
32   if j <= m_te
33   then let r := A[j, k];
34   else let r := A_te[j-m, k];
35   let norm:= norm + (l-r)^2;
36   }
37   let K[i,j]:= exp( -1/(2*sigma2)*norm);
38   let norm:=0;
39 }}
40
41 # Extraction of gamma from the dual solution
42 param gammad;
43 let gammad:= 1/sv*(sum{j in 1..m}class[j]*(y[j] - (sum{i in 1..n} lambda[i]*y[i]*K[i
       ,j+m])));
44 display gammad;
45
46
47 # Validation
48
49 # Prediction of kernel
50 print "Prediction kernel: ";
51 param prediction {1..m_te};
52
53 # Classification of target values 1, -1
54 for{j in 1..m_te} {
55   if gammad +sum{i in 1..m} y[i]*lambda[i]*K[i,j+m] >= 0
56     then let prediction[j] := 1;
57   else let prediction[j] := -1;
58 }
59 # display prediction, y_te;
60
61 # Accuracy
62 param hits default 0;
63 for {i in {1..m_te}} {
64   if prediction[i] == y_te[i] then
65     let hits := hits + 1;
66 }
67 display hits;
68 param accuracy = hits / m_te;
69 display accuracy;
70
71
72 # Definition of confusion matrix
73 param confmat{1..2,1..2} default 0;
74 for {i in 1..m_te}{
75   if prediction[i] == y_te[i] and prediction[i]=1 then let confmat[1,1]:= confmat
       [1,1] + 1;
76   }
```

```
for {i in 1..m_te}{
  if prediction[i] == y_te[i] and prediction[i]=-1 then let confmat[2,2]:= confmat
    [2,2] + 1;
  }
for {i in 1..m_te}{
  if prediction[i]*y_te[i]==-1 and prediction[i]=1 then let confmat[1,2]:= confmat
    [1,2] + 1;
  }
for {i in 1..m_te}{
  if prediction[i]*y_te[i]==-1 and prediction[i]=-1 then let confmat[2,1]:= confmat
    [2,1] + 1;
  }

# Display of confusion matrix
printf"Confusion Matrix. confmat[1,1] is TP, confmat[2,2] is TN, confmat[1,2] is FP
    and confmat[2,1] is FN";
display confmat;
```

Python code that uses *sklearn.datasets*:

```
from sklearn.datasets import make_swiss_roll
import numpy as np

# Generate the data
X_train, y_train = make_swiss_roll(n_samples=100, random_state=0)
X_test, y_test = make_swiss_roll(n_samples=100, random_state=1)

# Calculate the mean for the training and testing y
mean_train = np.mean(y_train)
mean_test = np.mean(y_test)

# Set data dimention
n = X_train.shape[1]
m = X_train.shape[0]
m_te = X_test.shape[0]

# Generate training data matrix
A = "\n".join([f"\t{i+1}" for i in range(n)])
t1 = "\n".join([f"\t{i+1}\t{X_train[i, 0]}\t{X_train[i, 1]}\t{X_train[i, 2]}" for i
    in range(m)])

# Generate y training vector
y = "\n".join([f"\t{i+1}\t{int(y_train[i] > mean_train) if y_train[i] > mean_train
    else -1}" for i in range(m)])

# Generate test data matrix
A_te = "\n".join([f"\t{i+1}" for i in range(n)])
t2 = "\n".join([f"\t{i+1}\t{X_test[i, 0]}\t{X_test[i, 1]}\t{X_test[i, 2]}" for i in
    range(m_te)])

# Generate y test vector
y_te = "\n".join([f"\t{i+1}\t{int(y_test[i] > mean_test) if y_test[i] > mean_test
    else -1}" for i in range(m_te)])

# Save the data to the file
with open('swiss.dat', 'w') as f:
    f.write("param A :\n" + A + " :=\n" + t1 + ";\n")
    f.write("param y :=\n" + y + ";\n")
```

```
    f.write("param A_te :\n" + A_te + " :=\n" + t2 + ";\n")
    f.write("param y_te :=\n" + y_te + ";\n")
```

Python code that calculates execution time:

```
import subprocess
import time

ampl_path = "/Users/alinacastellblasco/Desktop/data_sci/Q4/ampl_macos64/ampl"
run_file = "/Users/alinacastellblasco/Desktop/SVM-Project_OM/svm_kernel.run"

# Start the timer
start_time = time.time()

# Run the AMPL script using subprocess
subprocess.call([ampl_path, run_file])

# Calculate the execution time
execution_time = time.time() - start_time

# Print the execution time
print("Execution time: {:.2f} seconds".format(execution_time))
```