Support Vector Machine (SVM) Implementation using AMPL: A Comparative Study

Carlos Arbonés and Joel Solé

GCED, UPC.
Mathematical Optimization.

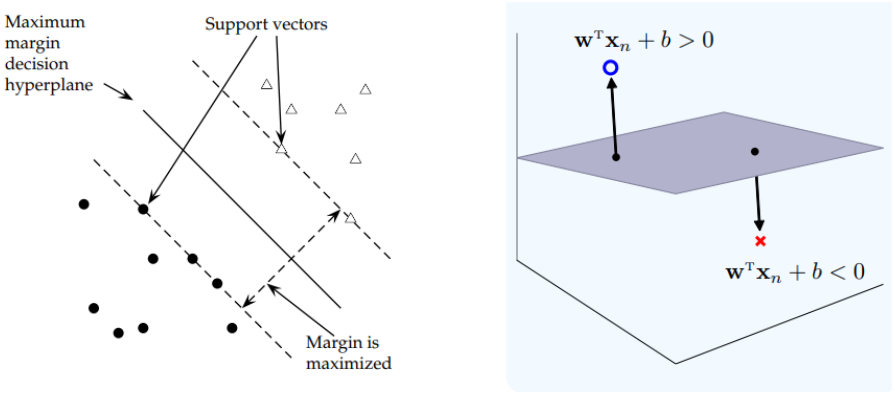
Contents

1 Introduction

In this project, we will implement the techniques of primal and dual optimization for *Support Vector Machines* (SVM). We will develop the necessary code to execute these algorithms and evaluate their accuracy on various datasets, including both proprietary and external datasets. Furthermore, we will validate the performance of both models on the different datasets and compare the results. Additionally, we will analyze and compare the hyperplanes generated by these models, ensuring their equivalence. Finally, we will assess the applicability of the SVM implementation on a non-separable dataset using the RBF or Gaussian Kernel.

2 Implementation

The objective is to separate the data, represented as $x \in \mathbb{R}^n$, into two distinct classes, which we will denote as $\{+1, -1\}$. The goal of SVM is to find the surface (hyperplane) that best separates the data, meaning it maximizes the distance to any data point (figure 2.1a). This distance is referred to as the classifier's *margin*. A classifier with a large margin makes fewer misclassifications because the farther the points are from the hyperplane, the more confident the classification will be.



(a) *Margin and support vectors* [1] (b) *Example of classification* [2]

Fig. 2.1: *Representation of the SVM*

2.1 Primal quadratic formulation

Given m pairs $(\mathbf{x}_i, y_i) \in \mathbb{R}^n \times \{+1, -1\}$, we will calculate the hyperplane defined by $(\mathbf{w}, \gamma)$ that maximizes the distance between the parallel planes $\mathbf{w}^T\mathbf{x}+\gamma \geq 1$ and $\mathbf{w}^T\mathbf{x}+\gamma \leq -1$.

It can be shown that the margin between the two planes is given by $\frac{2}{\|\mathbf{w}\|^2}$. Therefore, our objective is to maximize this margin in order to have more robust predictions. We can verify that maximizing the margin is equivalent to minimizing $\|\mathbf{w}\|^2$. Hence, we can express it as follows:

$$\max \frac{2}{\|\mathbf{w}\|^2} = \min \|\mathbf{w}\|^2 = \min \frac{1}{2}\mathbf{w}^T\mathbf{w}$$

On the other hand, the constraints we need to impose are related to the classifier:

$$\mathbf{w}^T\mathbf{x} + \gamma \geq +1 \text{ for } y_i = +1$$
$$\mathbf{w}^T\mathbf{x} + \gamma \leq -1 \text{ for } y_i = -1$$

When dealing with linearly inseparable data, we need to relax the constraints of the problem by introducing artificial variables, denoted as $s_i \geq 0$, for each data point, known as *slack* variables. This allows us to accept errors in predictions in order to make the problem solvable. The resulting constraints can be considered as soft constraints and are defined as follows:

$$y_i(\mathbf{w}^T\mathbf{x}_i + \gamma) \geq 1 - s_i \quad \text{for } i = 1, \ldots, m$$

By incorporating these constraints, we also need to account for the value of the slack variables when minimizing the objective function. In this case, we penalize the slack variables, as otherwise, the problem would always be solved by taking very large slack values. However, this would increase the prediction errors. Thus, the formulation of the problem becomes:

$$\begin{aligned} \min_{\mathbf{w},\gamma,\mathbf{s}\in\mathbb{R}^{n+1+m}} \quad & \frac{1}{2}\mathbf{w}^T\mathbf{w} + \nu\sum_{i=1}^{m}\mathbf{s}_i & \\ \text{s.t.} \quad & y_i(\mathbf{w}^T\mathbf{x}_i + \gamma) + \mathbf{s}_i \geq 1 & i = 1, \ldots, m \\ & \mathbf{s}_i \geq 0 & i = 1, \ldots, m \end{aligned}$$

An implementation in *AMPL* can be seen in listing 1.

2.2 Dual quadratic formulation

Given a primal problem, we have the following definitions:

Lagrangian function

$$L(x, \lambda, \mu) = f(x) + \lambda^T h(x) + \mu^T g(x)$$

Dual function

$$q(\lambda, \mu) = \min_x L(x, \lambda, \mu)$$
$$x \in X$$

Dual problem

$$\max_{\lambda,\mu} q(\lambda, \mu)$$
$$\mu \geq 0$$

According to Theorem 1, the dual function is concave, and therefore the first-order conditions are sufficient conditions. The point that maximizes the function will have a gradient of zero. Thus, we can rewrite the problem as follows:

Dual problem*

$$\max_{x,\lambda,\mu} L(x,\lambda,\mu)$$
$$\nabla_x L(x,\lambda,\mu) = 0$$
$$\mu \geq 0$$

According to Theorem 2, the dual problem provides a lower bound on the minimum value of the function. However, in the case of SVM, which is a convex problem, Theorem 3 states that the obtained value will be equal to the function's minimum, and therefore, we will also obtain the same parameters.

$$\max_{\mathbf{w},\gamma,\mathbf{s},\lambda,\mu} L(\mathbf{w},\gamma,\mathbf{s},\lambda,\mu)$$
$$\nabla_w L(\mathbf{w},\gamma,\mathbf{s},\lambda,\mu) = 0$$
$$\nabla_\gamma L(\mathbf{w},\gamma,\mathbf{s},\lambda,\mu) = 0$$
$$\nabla_\mathbf{s} L(\mathbf{w},\gamma,\mathbf{s},\lambda,\mu) = 0$$
$$\lambda \geq 0,\ \mu \geq 0$$

After some manipulation the dual problem becomes:

$$\begin{aligned} \max_{\lambda} \quad & \lambda^T \mathbf{e} - \frac{1}{2}\lambda^T Y A A^T Y \lambda \\ \text{s.t.} \quad & \lambda^T Y \mathbf{e} = 0 \\ & 0 \leq \lambda \leq \nu \end{aligned}$$

An implementation in *AMPL* can be seen in listing 2.

It can be observed that the problem can be rewritten by substituting $Q = YAA^TY$, where Q will have a size of $m \times m$. When a large number of data points are considered for optimization, the computational complexity can become a challenge due to the quadratic calculations and potentially disproportionate matrix dimensions. For instance, if 10,000 data points are used for SVM optimization, the matrix Q would have $10^4 \times 10^4$ elements, rendering it computationally infeasible.

2.2.1 Retrieving w and γ from the dual solution λ

Once the dual problem is solved we want to recover the plane $\mathbf{w}^T\phi(\mathbf{x}) + \gamma = 0$ from the vector of lambdas λ in order to make predictions on new incoming data and classify them. From equation 2 we have that:

$$\mathbf{w} = \sum_{i=1}^{m} \lambda_i y_i \phi(\mathbf{x})$$

We can consider the following partition of points:

- **Missclassified points (MC)**: the set of points that are misclassified.
- **Support vectors (SV):** the set of points that fall on the decision boundary
- **Non-binding points (NB):** the points that, if removed, do not affect the algorithm and do not alter the plane.

In figure 4.2 there is an exemplification of the types of points. It can be shown that:

$$\begin{cases} s_i > 0, \lambda_i = \nu & \text{if } i \in MC \\ s_i = 0, \lambda_i \geq 0 & \text{if } i \in SV \\ s_i = 0, \lambda_i = 0 & \text{if } i \in NB \end{cases}$$

and we can express it as:

$$\mathbf{w} = \sum_{i=1}^{m} \lambda_i y_i \phi(\mathbf{x}) = \sum_{i \in MC}^{m} \nu y_i \phi(\mathbf{x}) + \sum_{i \in SV}^{m} \lambda_i y_i \phi(\mathbf{x})$$

Therefore, the most important points are those with the maximum λ_i, which define the plane for optimization, in this case, the MC points.

To calculate γ, we simply need to take a point $i \in SV$, satisfying $0 \leq \lambda \leq \nu$ and satisfying the constraint with equality:

$$y_i(\phi(x_i)^T w + \gamma) = 1$$
$$\gamma = \frac{1}{y_i} - \phi(x_i)^T w$$

An implementation in *AMPL*[1] of retrieving w and γ can be seen in listing 4.

2.3 Results and analysis

In this section, we will measure the accuracy of the SVM on different datasets that contain varying data and dimensions, as well as the separation hyperplanes created by the SVM algorithm to separate the data. In the two datasets presented in this section, we will not be able to visualize the hyperplanes since both datasets have more than three dimensions (features).

For each optimization we perform, we will also consider the parameters provided by AMPL, such as execution time, objective function value, and total number of algorithm iterations.

The accuracy of the models will be calculated in Python due to its ease of computation and data extraction. The implementation can be seen in Listing 6.

The optimization of the formulations has been performed using the CPLEX solver [3], as it is particularly effective for quadratic and convex problems.

2.3.1 Methodology

For both datasets, we will first perform cross-validation to determine the optimal value of the hyperparameter ν (which regulates the slack values), i.e., the parameter that maximizes the accuracy. The methodology for this is explained in more detail in the following subsection.

After selecting the best value of ν, we will train the SVM using all the training data and test it on the reserved test samples.

To obtain both the separation hyperplanes and predictions for the test data, we will extract them from the results of the optimization performed with AMPL. In the case of the primal form, they can be directly extracted. For the dual form, as explained earlier, the process has been described.

[1] Please note that we have added a value of $\epsilon = 10^{-6}$ since the optimization package used (CPLEX) assigns a very small value (e.g., 2.75189e-10) to solutions that are equal to 0.

2.3.2 Cross validation

To find the appropriate value of ν for each dataset, we have divided the training set into two subsets: one for training and one for validation using *train_test_split()* function in *Python*. Using the training dataset, we have tested different values [2] of ν, specifically 0.2, 0.5, 1, 2, 5, 10, 20, 50, and 100.

In total, we have performed 9 optimization problems, each corresponding to a different value of ν. For each optimization, we have evaluated the accuracy on the validation data and selected the ν that maximizes the accuracy.

It is worth mentioning that for simplicity, we have only performed this cross-validation. For example, K-fold cross-validation could have been conducted to obtain the value of ν. However, in this project, our focus is more on understanding how SVM works rather than achieving the best possible results.

2.3.3 Generated dataset

The dataset used in this case is artificially generated and contains non-linearly separable data, resulting in classification errors. The generator produces data in four dimensions. To perform the optimization for both the primal and the dual forms, we have generated 1000 samples[3].

To select the value of ν, we have conducted the aforementioned cross-validation, and the chosen value is $\nu = 0.2$. (You can refer to the cross-validation results table in Table 5).

Once the optimization has been performed using both the primal and dual methods, we have extracted the respective parameters to verify that they are indeed the same. As shown in Table 1, the values are exactly the same for both models. However, it is also evident that the dual formulation requires significantly more computational time and 5 additional iterations compared to the primal formulation.

Table 1: *Result comparison of primal and sual solution in test generated dataset*

model	ν	γ	time(s)	accuracy	fobj	iterations
primal	0.2000	-5.4408	0.0320	0.9160	86.5232	12
dual	0.2000	-5.4408	0.9690	0.9160	86.5232	17

The optimal values for the variables in both the primal and dual formulations are as follows:

$$\mathbf{w}^T = (2.57068, 2.90207, 2.86961, 2.60609)$$
$$\gamma = -5.44081$$

The generated hyperplane is:

$$w^T x + \gamma = 2.57\mathbf{x}_1 + 2.9\mathbf{x}_2 + 2.86\mathbf{x}_3 + 2.6\mathbf{x}_4 - 5.44 = 0$$

[2] These values may not be the optimal ones. We have tried a range of different values to ensure a comprehensive comparison.

[3] More points could have been chosen to have more training data, but the dual form has quadratic cost, and the calculations could become more complex

2.3.4 Diabetes dataset

In this subsection, we have obtained a dataset from the internet to test SVM. We require a dataset that is suitable for binary classification. In this case, we have chosen the *Diabetes Dataset* [4], where the objective is to diagnostically predict whether a patient has diabetes based on certain diagnostic measurements included in the dataset.

The dataset consists of 768 patients and 8 explanatory variables, including features such as *pregnancies, glucose, blood pressure* and more. An example of 5 random samples from the dataset, along with all the variables, can be seen in Table 2.

Table 2: *Example of some diabetes dataset samples*

Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DPF	Age
2.0	84.0	0.0	0.0	0.0	0.0	0.3	21.0
9.0	112.0	82.0	24.0	0.0	28.2	1.3	50.0
1.0	139.0	46.0	19.0	83.0	28.7	0.7	22.0
0.0	161.0	50.0	0.0	0.0	21.9	0.3	65.0
6.0	134.0	80.0	37.0	370.0	46.2	0.2	46.0

In this case, we have also performed cross-validation to determine the best value for ν, and the results can be seen in Table 6. We observe that all values of ν yield the same accuracy, and therefore we randomly select one, such as $\nu = 0.2$. We have also extracted the information for both the primal and the dual from the results, obtaining the following:

Table 3: *Result comparison of primal and sual solution in test diabetes dataset*

model	ν	γ	time(s)	accuracy	fobj	iterations
primal	0.2000	-6.5818	0.0310	0.7532	63.1217	22
dual	0.2000	-6.5818	0.4690	0.7532	63.1217	18

We can observe from Table 3 that the primal formulation is still much faster in terms of execution compared to the dual formulation. Although the primal formulation requires 4 more iterations than the dual formulation, these iterations are faster.

The optimal values for the variables in both the primal and dual formulations are as follows:

$$\mathbf{w}^T = (0.0571, 0.0277, 0.0124, 0.00084, 0.00127, 0.0724, 0.4003, 0.0251)$$
$$\gamma = -6.58175$$

The generated hyperplane is:

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

3 The Kernel trick in SVM

Kernel methods in SVM allow us to perform nonlinear classification by applying a nonlinear transformation to the data points before applying SVM. The primary purpose of using kernels is to overcome the limitation of SVM, which creates a flat boundary (hyperplane) to separate data points into classes. This nonlinear transformation can create a non-linear decision boundary, allowing SVM to classify data points more accurately.

However we need to choose an appropriate nonlinear transformation, which can be a complex task. Moreover, increasing the dimension of the output of the transformation leads to higher computational requirements.

The kernel trick allows us to compute the inner product between the transformed data points without explicitly knowing the transformation itself. The kernel function represents this inner product and quantifies the similarity between data points after the nonlinear transformation.

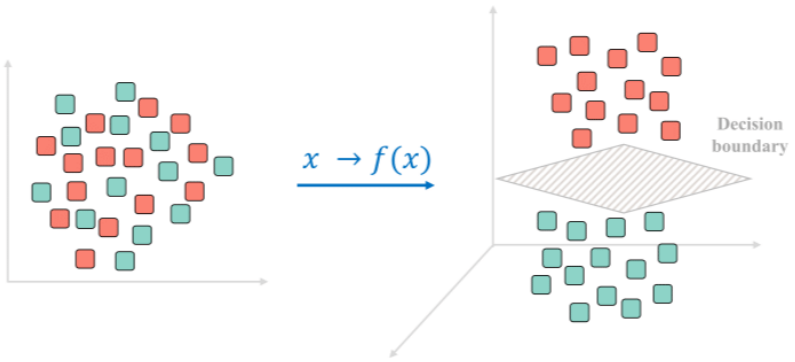


Fig. 3.1: *Example of kernel transformation* [5]

$$K_{ij} = K(x_i, x_j) = \phi(x_i)^T \cdot \phi(x_j) = \sum_{l=1}^{N} \phi_l(x_i)\phi_l(x_j), \quad i, j = 1, \ldots, m$$

In this application we used the Gaussian Kernel:

$$K(x_i, x_j) = \exp\left(-\frac{\|x_i - x_j\|^2}{2\sigma^2}\right)$$

The kernel trick can be employed in the dual problem by replacing the term AA^T with K. By doing so, we can achieve the desired nonlinear transformation without the need to explicitly compute it:

$$\begin{aligned} \max_{\lambda} \quad & \lambda^T \mathbf{e} - \frac{1}{2}\lambda^T Y K Y \lambda \\ \text{s.t.} \quad & \lambda^T Y \mathbf{e} = 0 \\ & 0 \leq \lambda \leq \nu \end{aligned}$$

An implementation in *AMPL* can be seen in listing 3.

3.1 Classification without the explicit hyperplane

To perform optimization, we have observed that it is not necessary to know the transformation $\phi(x)$ to define the kernel matrix. However, once we have the solution of the dual problem with kernel trick (the vector of lambdas), we want to be able to classify new points based on:

$$\begin{cases} \mathbf{w}^T\phi(\mathbf{x}) + \gamma > 0 & (+1) \\ \mathbf{w}^T\phi(\mathbf{x}) + \gamma < 0 & (-1) \end{cases}$$

We can achieve classification without needing the transformation:

$$\mathbf{w}^T\phi(\mathbf{x}) = \left(\sum_{i=1}^{m} \lambda_i y_i \phi(\mathbf{x}_i)^T\right)\phi(\mathbf{x}) = \sum_{i=1}^{m} \lambda_i y_i (\phi(\mathbf{x}_i)^T\phi(\mathbf{x})) = \sum_{i=1}^{m} \lambda_i y_i K(x_i, x)$$

To calculate γ for an point $k \in SV$ we have:

$$\gamma = \frac{1}{y_k} - \phi(\mathbf{x}_k)^T\mathbf{w} = \frac{1}{y_k} - \sum_{i=1}^{m} \lambda_i y_i K(x_i, x_k)$$

An implementation in *AMPL*[4] of how to make classifications without having the transformation can be seen in listing 5

The Gaussian kernel transformation has a transformation $\phi(x) \in \mathbb{R}^\infty$, which facilitates data classification. Having more dimensions for our points makes it easier to find a hyperplane that allows for good classification.

3.2 Dataset

In this section, we created a linearly non-separable dataset using the *make_swiss_roll()* function in titPython. We generated 1000 samples, which by default have a dimension of 3 for classification purposes. When generating the dataset, the points are not initially labeled as -1 or 1, but have real values. To address this, we assigned points with a y value greater than the mean to class 1, and those with a y value less than the mean to class -1.

An implementation example of how to create the dataset can be seen in Listing 7.

3.3 Results and accuracy

In this subpart, we did not perform cross-validation, and we selected a value of $\nu = 1$ for the optimization. The results obtained can be observed in Table 4.

By employing the proposed kernel application, we achieve a remarkable 100% accuracy in our classification task. The utilization of this kernel successfully separates the data points, enabling an exact classification.

Table 4: *Result comparison of primal and sual solution in test diabetes dataset*

kernel	ν	w	γ	acc	time	fobj	iter
Gaussian	1	cannot be found	0.2741	100.0	1.2340	36.1062	19
Linear	1	[0.0334685, 0.00486892, -0.112087]	-0.1710	60.0	0.5310	645.0155	15

[4]Note that we have used a $\sigma = n$ (number of variables) for the hyperparameter.

The objective function is also much lower with the Gaussian kernel. As for the execution time and number of iterations, we observe that the Gaussian kernel takes more than double the time of the linear kernel and requires 4 additional iterations.

In comparison, when utilizing the linear kernel in the dual problem with the same dataset, we observe a significantly lower accuracy of 60%.

These results highlight the superior performance of the Gaussian kernel over the linear kernel. The Gaussian kernel, operating in an infinite-dimensional space, generates a hyperplane that precisely separates the data without any misclassification. In contrast, the dual formulation without the kernel trick achieves a classification accuracy of only 60% using a three-dimensional hyperplane[5] defined by

$$w = (0.0334685, 0.00486892, -0.112087)$$

and

$$\gamma = -0.171039$$

[5]Since it is a 3-dimensional hyperplane, we can visualize the hyperplane and the points we want to separate. The representation can be seen in Figure 4.1.

4 Appendix

4.1 Code

4.1.1 AMPL

Listing 1: Implementation of primal quadratic formulation

```
### Parameters ###
param n >= 1, integer; # number of variables
param m >= 1, integer; # number of samples
param nu;           # hyperparameter
param y{i in 1..m}; # predicted label vector
param A{1..m, 1..n}; # matrix of data

### Variables ###
var w {1..n};   # weights
var gamma;        # intercept
var s {1..m};     # slacks: soft constrains

minimize fobj_primal:
  1/2*(sum{i in 1..n} w[i]^2) + nu*(sum{i in 1..m} s[i]);
subject to classify{i in 1..m}:
  -y[i]*(sum{j in 1..n} w[j]*A[i, j] + gamma) - s[i] + 1 <= 0;
subject to slacks{i in 1..m}:
  -s[i] <= 0;
```

Listing 2: Implementation of dual quadratic formulation

```
param n >= 1, integer; # number of variables
param m >= 1, integer; # number of samples
param nu;          # hyperparameter
param y{i in 1..m}; # predicted label vector
param A{1..m, 1..n}; # matrix of data

var la{1..m};

maximize fobj_dual:
  (sum{i in 1..m} la[i]) - (1/2*sum{i in 1..m, j in 1..m}
     la[i]*y[i]*la[j]*y[j]*(sum{k in 1..n} A[i,k]*A[j,k]));
subject to h:
  sum{i in 1..m} la[i]*y[i] = 0;
subject to c{i in 1..m}:
  0 <= la[i] <= nu;
```

Listing 3: Implementation of dual quadratic formulation with kernel trick

```
param n >= 1, integer; # number of variables
param m >= 1, integer; # number of samples
param nu;         # hyperparameter
param sigma;      # hyperparameter
param y{i in 1..m}; # predicted label vector
param A{1..m, 1..n}; # matrix of data

var la{1..m};

maximize fobj_dual:
   (sum{i in 1..m} la[i]) -
   (1/2*sum{i in 1..m, j in 1..m}
      la[i]*y[i]*la[j]*y[j]*exp(-sum{k in 1..n} ((A[i,k] -
      A[j,k])^2) / (2 * sigma^2)));
subject to h:
   sum{i in 1..m} la[i]*y[i] = 0;
subject to c{j in 1..m}:
   0 <= la[j] <= nu;
```

Listing 4: Implementation of retrieving parameters from the dual solution

```
param w{1..n};
param gamma;
for {i in 1..n}{
   let w[i] := (sum{j in 1..m} y[j]*la[j]*A[j,i]);
}
for {i in 1..m}{
   if la[i] > 1e-6 and la[i] < nu - 1e-6 then {
      let gamma := (1/y[i]) - (sum{j in 1..n} A[i, j]*w[j]);
      break;
   }
}
```

Listing 5: Implementation of predictions with kernel trick

```
param gamma;
param pred{i in 1..TEST_SIZE};

for {k in 1..m}{
     if la[k] > 1e-6 and la[k] < nu - 1e-6 then {
        let gamma := (1/y[k]) - (sum{i in 1..m}
           la[i]*y[i]*exp(-sum{j in 1..n} ((A[i,j] - A[k,j])^2)
           / (2 * sigma^2)));
        break;
     }
   }
for {k in 1..TEST_SIZE}{
   let pred[k] := (sum{i in 1..m} la[i]*y[i]*exp(-sum{j in
      1..n} ((A[i,j] - test[k,j])^2) / (2 * sigma^2))) + gamma;
}
```

4.1.2 Python

Listing 6: Accuracy computation

```
def getAccuracy(w, gamma, X, y):
   y_pred = np.dot(X, w) + gamma
   y_pred = np.where(y_pred > 0, 1, -1)
   return np.sum(y_pred == y)/len(y)
```

Listing 7: Generating dataset with non separable points

```
from sklearn.datasets import make_swiss_roll

Xk, yk = make_swiss_roll(n_samples=1000, noise=0.0,
    random_state=42)
Xk = pd.DataFrame(Xk).astype(float)
yk = np.where(yk > yk.mean(), 1, -1)
yk = pd.DataFrame(yk).astype(float)
```

Listing 8: Plane generator

```
import matplotlib.pyplot as plt

x = np.linspace(-10, 10, 100)
y = np.linspace(-10, 10, 100)
X, Y = np.meshgrid(x, y)
Z = (-wk[0]*X - wk[1]*Y - gamma_k)/wk[2]

fig = plt.figure(figsize=(10, 10))
ax = fig.add_subplot(111, projection='3d')
ax.plot_surface(X, Y, Z, alpha=0.5)
ax.scatter(Xk_test[0], Xk_test[1], Xk_test[2], c=yk_test[0],
    s=50, cmap='viridis')
plt.show()
```

4.2 Tables

Table 5: *Cross validation results for primal formulation in the generated dataset*

ν	γ	**time(s)**	**accuracy**	**fobj**	**iterations**
0.2000	-5.0463	0.0310	0.9650	73.0943	11
0.5000	-6.5464	0.0310	0.9550	158.0650	12
1	-7.7613	0.0320	0.9550	290.6490	11
2	-8.8408	0.0160	0.9500	546.6816	11
5	-10.1251	0.0310	0.9600	1298.6117	12
10	-10.7485	0.0160	0.9600	2541.8396	13
20	-10.9371	0.0310	0.9500	5024.3618	11
50	-11.0366	0.0310	0.9500	12469.2929	11
100	-11.0400	0.0150	0.9500	24877.0333	12

Table 6: *Cross validation results for primal formulation in the diabetes dataset*

ν	γ	**time(s)**	**accuracy**	**fobj**	**iterations**
0.2	-6.7835	0.0320	0.7886	51.0649	19
0.5	-6.6654	0.0310	0.7886	127.4677	23
1	-6.6566	0.0320	0.7886	254.7691	23
2	-6.6452	0.0310	0.7886	509.3672	21
5	-6.6430	0.0310	0.7886	1273.1407	19
10	-6.6430	0.0310	0.7886	2546.0936	18
20	-6.6430	0.0160	0.7886	5091.9994	18
50	-6.6430	0.0160	0.7886	12729.7169	18
100	-6.6430	0.0150	0.7886	25459.2460	19

4.3 Figures

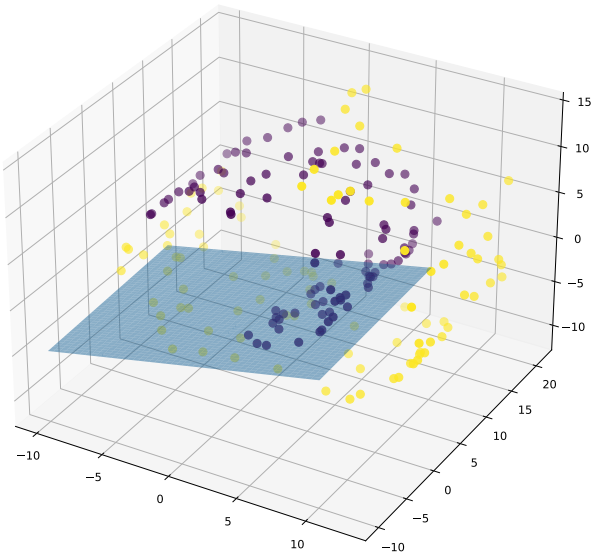


Fig. 4.1: *Classification hyperplane of lineal kernel optimization given by the equation:*

$$0.03x_1 + 0.004x_2 - 0.11x_3 - 0.17 = 0$$

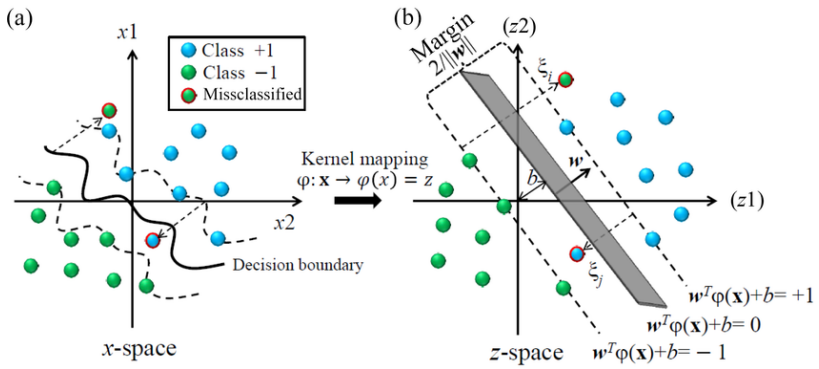


Fig. 4.2: *Point partition: Missclassified points (red contour), Support Vectors (b lue and green in the dashed line), and Non-Binding points (good classified)* [6]

4.4 Text

Theorem 1 (Concavity of $q(\lambda, \mu)$)**.** *The dual function*

$$q(\lambda, \mu) = \min_{x \in X} L(x, \lambda, \mu) = \min_{x \in X} f(x) + \lambda^T h(x) + \mu^T g(x)$$

is concave (in the region where is finite, that is, the minimum exists).

Theorem 2 (Weak duality)**.** *Let x be a feasible point of primal problem ($h(x) = 0$, $g(x) \leq 0$, $x \in X$) and (λ, μ) a feasible point of dual problem ($\mu \geq 0$) then:*

$$q(\lambda, \mu) \leq f(x)$$

Theorem 3 (Strong duality)**.** *If X is a convex set, $f(x)$ and $g(x)$ are convex functions, $h(x) = Ax - b$, under certain constraints qualifications:*

$$q(\lambda^*, \mu^*) = f(x^*)$$

Derivation of the Dual SVM problem

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

$$\mathbf{w} - (\lambda^T Y A)^T = 0 \tag{2}$$

$$\lambda^{\mathbf{T}} Y \mathbf{e} = 0 \tag{3}$$

$$\nu\mathbf{e} - \lambda - \mu = 0 \tag{4}$$

$$\lambda \geq 0,\ \mu \geq 0 \tag{5}$$

$$(1) =_{(2)} \frac{1}{2}\lambda^T Y A A^T Y \lambda + \nu\mathbf{e}^T\mathbf{s} + \lambda^T(-Y(AA^T Y\lambda + \gamma\mathbf{e}) - \mathbf{s} + \mathbf{e}) - \mu^T\mathbf{s}$$

$$= \frac{1}{2}\lambda^T Y A A^T Y \lambda + \nu\mathbf{e}^T\mathbf{s} - \lambda^T Y A A^T Y \lambda - \gamma\lambda^T Y\mathbf{e} - \lambda^T\mathbf{s} + \lambda^T\mathbf{e} - \mu^T\mathbf{s}$$

$$= -\frac{1}{2}\lambda^T Y A A^T Y \lambda + (\nu\mathbf{e}^T - \lambda^T - \mu^T)\mathbf{s} - \gamma\lambda^T Y\mathbf{e} + \lambda^T\mathbf{e} =_{(3)(4)} -\frac{1}{2}\lambda^T Y A A^T Y \lambda + \lambda^T\mathbf{e}$$

$$(4) \implies {}_{(5)}\nu\mathbf{e} - \lambda \leq 0 \implies \lambda \leq \nu\mathbf{e}$$

Classification of points and their respective values of λ_i

Through the values of λ_i, we can determine the type as follows:

1. **For Misclassified (MC) points:** If $s_i > 0$, one of the optimality conditions is $s_i\mu_i = 0$ (complementarity). Therefore, if a point is misclassified, we have $\mu = 0$. From Equation 4, we have $\nu - \lambda_i - \mu_i = 0 \implies \nu - \lambda_i = \mu_i = 0 \implies \lambda_i = \nu$.
2. **For Non-Bounding (NB) points:** Another complementarity condition is $\lambda_i(y_i(\phi(x_i)^T w + \gamma) + s_i - 1) = 0$. Since these points are correctly classified, we have $s_i = 0$. Thus, $\lambda_i(y_i(\phi(x_i)^T w + \gamma) - 1) = 0$. As these points lie outside the decision boundary, it represents an inactive constraint $(y_i(\phi(x_i)^T w + \gamma) - 1) > 0$, and therefore, $\lambda_i = 0$.
3. **For Support Vectors (SV):** Using the same reasoning as before, the constraint becomes active as these points lie on the separation planes, resulting in $(y_i(\phi(x_i)^T w + \gamma) - 1) = 0$. We have $\lambda \geq 0$, and by the constraint $0 \leq \lambda_i \leq \nu$.

References

[1] C.D. Manning, P. Raghavan, H. Schütze. Support vector machines and learning theory (2008). URL https://nlp.stanford.edu/IR-book/pdf/15svm.pdf

[2] S. Arlot, A. Celisse. A survey of cross-validation procedures for model selection (2010). URL https://edisciplinas.usp.br/pluginfile.php/5078086/course/section/5978681/chapSVM.pdf

[3] IBM. IBM ILOG CPLEX Optimization Studio. https://www.ibm.com/products/ilog-cplex-optimization-studio (Accessed 2023). IBM website

[4] A.D. Khare. Diabetes dataset. https://www.kaggle.com/datasets/akshaydattatraykhare/diabetes-dataset (2021). Accessed: May 17, 2023

[5] M. Mahdianpari. Support vector machine vs. random forest for remote sensing image classification: A meta-analysis and systematic review (2020). URL https://www.researchgate.net/figure/An-SVM-example-for-non-linearly-separable-data-with-the-kernel-trick_fig3_344437400

[6] F. Khan, F. Enzmann, M. Kersten, Multi-phase classification by a least-squares support vector machine approach in tomography images of geological samples. Full-text available (2016)