Solving SVM using IPM

Derrick Kim
Advised by Mutiara Sondjaja

May 18, 2020

Abstract

The purpose of this paper is to understand Interior Point Method and the challenges involved if the number of input points is scaled up. More specifically, on a training data set that has binary labels, we formulate a Support Vector Machine Quadratic Program, and attempt to solve it. We explain why we cannot use standard Linear Programming algorithms on the SVM Quadratic Program, and why we therefore use the KKT conditions to convert it to a linear system. At this point, we still face problems because of the existence of inequalities in the constraints, and ultimately use a log barrier to ensure that the iterations of the Interior Point Method stay within the feasible region. With the newly evolved constraints, we try to find the optimal hyperplane that classifies the modified problem.

1 Introduction

Separating points into classes is an interesting problem. Classification is essentially taking a set of input points and labels and constructing a hyperplane to divide the labeled points. In an ideal setting, the points are linearly separable, but this is not always the case. There are scenarios where a clear cut hyperplane cannot divide the labeled points. This paper attempts a novel approach to solve scenarios where the labeled points may not be linearly classifiable.

There are many different types of classification algorithms, such as K-Nearest-Neighbors, Logistic Regression, Naive Bayes Classifier, but this paper focuses on Support Vector Machines(SVM). We approach SVMs as a convex optimization problem using Karush-Kuhn-Tucker Conditions and solve it using the Interior Point Method.

2 Background

We introduce a few standard concepts, which will be used in our approach. This section acts a general introduction to these concepts, and does not go into the details of our specific implementation. Our unified framework which implements these concepts is shown in a later section.

2.1 Quadratic Programming

A linearly-constrained quadratic programming problem is a constrained optimization problem where we minimize a convex quadratic function subject to linear constraints. We specifically study the form with k variables, and l constraints as follows.

$$\begin{aligned} \min \quad & f(\vec{v}) \\ \text{s.t.} \quad & g_i(\vec{v}) \leq 0 \\ & h_i(\vec{v}) = 0 \end{aligned} \tag{1}$$

where,

- $\vec{c}, \vec{v} \in \mathbb{R}^k$
- $\mathrm{Q} \in \mathbb{R}^{(k)\mathrm{v}(k)}$ is a symmetric semi-definite matrix

- $f(\vec{v}) = \frac{1}{2}\vec{v}^T Q\vec{v} + \vec{c}^T\vec{v}$
- $g_i(\vec{v}), h_i(\vec{v})$ are linear constraints

2.2 Karush Kuhn Tucker (KKT) Conditions

Given a general function that we want to minimize, the optimal solution necessarily has to follow the KKT conditions. The KKT conditions are derived from the standard Lagrangian multiplier approach. While the Lagrangian multiplier approach works only for equality constraints, the KKT conditions generalize to inequality constraints as well. Given an optimization problem as follows.

$$\begin{aligned} \min \quad & f(\vec{v}) \\ \text{s.t.} \quad & g_i(\vec{v}) \leq 0 \\ & h_i(\vec{v}) = 0 \end{aligned}$$

Let the numbers of inequalities and equalities be l and l' respectively.

2.2.1 Stationarity

$$\text{For minimizing } f(\vec{v}^*) : \nabla f(\vec{v}^*) + \sum_{i=1}^{l} \mu_i \nabla g_i(\vec{v}^*) + \sum_{j=1}^{l'} \lambda_j \nabla h_j(\vec{v}^*) = 0$$

2.2.2 Primal Feasibility

$$g_i(\vec{v}^*) \leq 0, i \in \{1...l\}$$
$$h_i(\vec{v}^*) = 0, i \in \{1..l'\}$$

2.2.3 Dual Feasibility

$$\mu_i \geq 0, i \in \{1...l\}$$

2.2.4 Complementary Slackness

$$\mu_i g_i(\vec{v}^*) = 0, i \in \{1...l\}$$

2.3 Newton's Method

Newton's method is an iterative method that helps us find the root of a function. We start with an initial guess that is a good approximation of the root. In the one dimensional case, we approximate the function by its tangent at the initial point. The x-intercept of the tangent line gives us a better approximation of the root. Iteratively, this gets us closer to the root. In our case, we have a higher dimensional function with a higher dimensional root. Analogously, we use the Jacobian matrix instead of the tangent line.
Given a function $F : \mathbb{R}^u -> \mathbb{R}^v = (f_1...f_v)$, the Jacobian matrix is defined to be a v x u matrix. Specifically, it is written as

$$\nabla F(\vec{Z}) = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_u}{\partial x_u} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_v}{\partial x_1} & \cdots & \frac{\partial f_v}{\partial x_u} \end{bmatrix}$$

We want to find $\vec{Z}$ such that

$$\vec{F}(\vec{Z}) = \vec{d}, Z \in \mathbb{R}^u, \vec{d} \in \mathbb{R}^v$$

This can be done using the Newton's method to find the root of $\vec{F}(\vec{Z}) - \vec{d}$. Rewriting this as an iterative method, we get the following.

$$\vec{Z}^{(i+1)} = \vec{Z}^{(i)} + [\nabla F(\vec{Z}^{(i)}]^{-1}(\vec{d} - \vec{F}(\vec{Z}^{(i)})) \tag{2}$$

2.4 Interior Point Method(IPM)

Interior Point Method is a way of solving convex optimization problems. Unlike previous methods, they are not restricted by linearity. We use a barrier function to encode the convex feasible region. Unlike conventional programming algorithms which traverse the boundary, the Interior Point Method traverses the interior of the feasible region.

Given a function $f(\vec{v})$, we wish to minimize $f(\vec{v})$ subject to some constraints[1].

$$c_i(\vec{v}) \geq 0 \text{ for } i = \{1 \dots m\},\ \vec{v} \in \mathbb{R}^n,\ \text{where } f : \mathbb{R}^n \to \mathbb{R}, c_i : \mathbb{R}^n \to \mathbb{R}$$

The logarithmic barrier function associated with $f(\vec{v})$ is

$$B(\vec{v}, \beta) = f(\vec{v}) - \beta \sum_{i=1}^{m} \log(c_i(\vec{v})) \tag{3}$$

Here $\beta > 0$ and serves as a scalar, and as β converges to zero, the minimum of $B(x, \beta)$ should converge to a solution. Here, the domain of the log barrier is from 0 to infinity, so we implicitly ensure that the constraint is greater than 0.

$$g_b = g - \beta \sum_{i=1}^{m} \frac{\nabla c_i(\vec{v})}{c_i(\vec{v})} \tag{4}$$

where g is the gradient of the original function $f(\vec{v})$, and ∇c_i is the gradient of c_i. We also introduce a dual variable λ, where

$$c_i(\vec{v})\lambda_i = \mu, \forall i = \{1...m\} \tag{5}$$

We try to find those $(\vec{v}_\mu, \lambda_m u)$ for which the gradient of the barrier function is zero. Applying (5) to (4), we get an equation for the gradient:

$$g - \nabla F(\vec{Z})^T \lambda = 0 \tag{6}$$

where the matrix A is the Jacobian of the constraints $c(x)$.
Applying Newton's method to (5) and (6), we get an equation for $(\vec{v}, \lambda)$. Because of (6) and because we wish to minimize $f(\vec{v})$, the condition $\lambda \geq 0$ should be enforced at each step. This can be done by choosing an appropriate α :

$$(\vec{v}, \lambda) \to (\vec{v} + \alpha p_{\vec{v}}, \lambda + \alpha p_\lambda) \tag{7}$$

3 SVM Quadratic Program Formulation

Support Vector Machines(SVM) is a popular classification algorithm to classify a set of points by finding a hyperplane with the largest separation between the classes. This is effectively maximizing the minimum of distances to points in a class.

We relax the constraints as the data points need not be linearly separable and work on a formulation shown below. Alternatively, we could use an appropriate kernel function to take the points to a new space, where they are linearly separable. We can then find a good separating hyperplane in the new space. We can convert it back by inverting the kernel function. Thus, effectively, we are classifying certain types of non-linear classifiable data. But this is for possible future work, and will not be discussed in this paper.

SVM can be formulated as a convex quadratic program[6].The classification algorithm typically involves a feature vector, $\phi(x)$. This will typically include a bias component ($\phi_0(x) = 1$) to maintain dimensionality. In this paper, note $\phi(x) = x$.

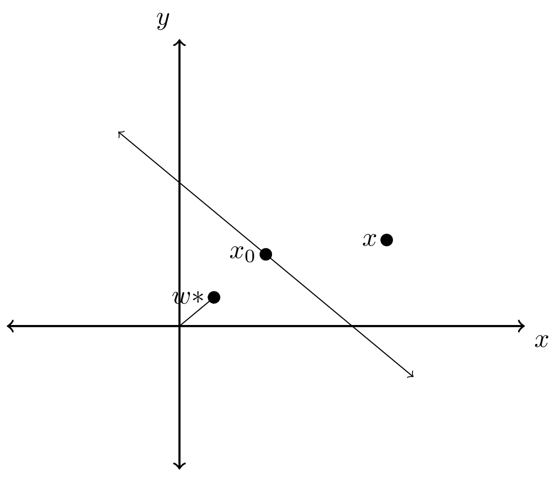


Figure 1: Example of a distance between a point and a hyperplane

We formally define a hyperplane in $\mathbb{R}^d$ using an example shown in Figure 1. Let $w*$ be the unit vector perpendicular to the hyperplane. Consider an arbitrary point x. The goal is to find the distance between x and the hyperplane. If we define x_0 on the hyperplane, then this distance corresponds to length of $x - x_0$ in direction of w^*, which equals

$$w^{*T}(x - x_0) = w^{*T}x - w^{*T}x_0 = \frac{w^Tx - w^Tx_0}{||w||} = \frac{w^T(x) - b}{||w||} \tag{8}$$

where b is the y-intercept.

Furthermore, $w^T\phi(x) + b$ and x's label y must have the same sign. So, if we use the feature vector $\phi(x)$ instead of x,

$$H_d(\phi(x)) = \frac{y[w^T\phi(x) - b]}{||w||} \tag{9}$$

We have defined a hyperplane. Consider the input data set of n points $x_1 ... x_n \in \mathbb{R}^d$ and corresponding labels $y_1 ... y_n \in \{-1, 1\}$ for classification. If this data set was separable, note that there is an infinite number of decision boundaries or separating hyperplanes.

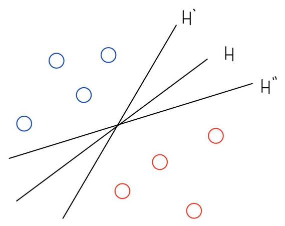


Figure 2: Possible Linear Separators for given data points

In Figure 2, the ideal line, H, would be a decision boundary in the exact *middle* of the two classes. H' or H'' will also separate the data points correctly, but we want a decision boundary that correctly classifies the training data and is as far away from every training point as possible.

In Figure 3, we can see that the line H is exactly in the *middle* of the two classes.

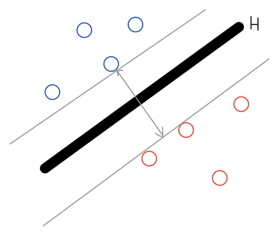


Figure 3: Support Vectors for Linear Separator. Note that H here maximizes the minimum distance to the two classes

3.0.1 Margin

In our data set, consider the point from each class that is closest to the decision boundary. The distance between this point and the decision boundary is called the *margin*, and this is shown in Figure 4.

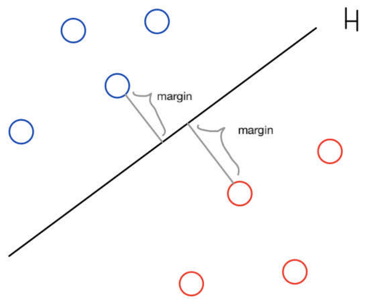


Figure 4: Illustration of Margin

Note that the margin is:

$$margin = \min_{i \in \{1...n\}} \frac{y_i[w^T\phi(x_i) - b]}{||w||} \tag{10}$$

To find a hyperplane that separates the two classes while also maximizing the margin, we pick a line that would be the farthest from the closest points of each class. We now re-scale the margin. There are two possible methods shown below.

$$\begin{cases} ||w|| =1 \\ \min y[w^T\phi(x) - b] \end{cases} \tag{11}$$

Or:

$$\begin{cases} \min ||w|| \\ y[w^T\phi(x) - b] = 1 \end{cases} \tag{12}$$

Since we are looking for the optimal line, it is not influenced by setting $y[w^T\phi(x) - b] = 1$ as we are just scaling the system. The later method is more appropriate here as we will introduce more constraints and slack variables later. Hence, the points closest to the decision boundary are at distance 1.

The above formulation works for linearly separable data. But non-linearly separable data is more interesting, and in such cases, we introduce a slack variable ξ_i that relaxes our constraints. This means, we discount certain points such as outliers, and try to still come up with a respectable line that does a relatively

good job of separating the data.
For every point, the relaxed constraint can be posed as follows.

$$y_i(w^T x_i - b)) - 1 + \xi_i \geq 0, \forall i \in \{1...n\} \tag{13}$$

Then, our quadratic program becomes as follows.

Let $\vec{v} = (w_1...w_d, b, \xi_1, ...\xi_n)$
and for some constant C.

$$\begin{aligned} \min_{w,b,\xi} \quad & \frac{1}{2}\sum_{j=1}^{d} w_j^2 + C\sum_{i=1}^{n} \xi_i \\ \text{s.t.} \quad & y_i(w^T x_i - b)) - 1 + \xi_i \geq 0 \\ & \xi_i \geq 0 \\ & \forall i \in \{1...n\} \end{aligned} \tag{14}$$

4 Solving the SVM Quadratic Program Formulation

We have formulated the SVM Quadratic Program Formulation so far, and we now are trying to solve it. Since it is a quadratic program, we cannot directly use standard Linear Programming algorithms. Therefore, we use KKT conditions to convert it to a linear system. But this system has problems because there are inequalities in the constraints. Therefore, we use the log barrier to enforce the feasible regions given by the constraints. Therefore, we try to find the optimal hyperplane using the Interior Point Method on this modified problem. This section deals with the details of this implementation.

4.1 Input

Consider an input data set of n points $x_1...x_n \in \mathbb{R}^d$ and corresponding labels $y_1...y_n \in \{-1, 1\}$.Here $x_i = (x_i^1...x_i^d)$ and let $m = d + 1$. The input may or may not be linearly separable, but this is not a problem because we have slack variables to handle this.

4.2 Modified Quadratic Program Formulation

In this subsection, we convert the SVM formulation in the previous section(14) to the quadratic program formulation in the background(1).

We define the matrix $\tilde{A} \in \mathbb{R}^{n\times(m+2n)}$ as follows.

$$\tilde{A} = \left.\underbrace{\begin{bmatrix} x_1^{(1)}y_1 & \dots & x_1^{(d)}y_1 & -y_1 \\ x_2^{(1)}y_2 & \dots & x_2^{(d)}y_2 & -y_2 \\ \vdots & & \vdots & \vdots \\ x_n^{(1)}y_n & \dots & x_n^{(d)}y_n & -y_n \end{bmatrix}}_{m+2n}\right) n$$

And we define the matrix $A \in \mathbb{R}^{n\times(m+4n)}$ as follows.

$$A = \left.\underbrace{\begin{bmatrix} \tilde{\mathbf{A}} & \mathbf{I_n} & -\mathbf{I_n} \end{bmatrix}}_{m+4n}\right) n$$

We use an additional slack variable s_i to convert the constraints of the SVM Quadratic Program formulation into equalities from inequalities.

Let $\vec{v} = (w_1 \dots w_d, b, \xi_1 ... \xi_n, s_1 ... s_n)$
and for some constant C.

$$\begin{aligned} \min_{w,b,\xi} \quad & \frac{1}{2}\sum_{j=1}^{d} w_j^2 + C\sum_{i=1}^{n}\xi_i \\ \text{s.t.} \quad & y_i(w^T x_i - b)) - 1 + \xi_i - s_i = 0 \\ & \xi_i \geq 0 \\ & s_i \geq 0 \\ & \forall i \in \{1...n\} \end{aligned} \tag{15}$$

This can be re-written as:

$$\begin{aligned} \min \quad & \frac{1}{2}\vec{v}^T Q \vec{v} + \vec{c}^T \vec{v} \\ \text{s.t.} \quad & A\vec{v} = \vec{b} \\ & \xi_i \geq 0 \\ & s_i \geq 0 \\ & \forall i \in \{1...n\} \end{aligned} \tag{16}$$

where

$$\vec{v} = \begin{bmatrix} \vec{w} \\ \vec{\xi} \\ \vec{s} \end{bmatrix} \in \mathbb{R}^{m+2n}, \vec{w} = \begin{bmatrix} w_1 \\ \vdots \\ w_d \\ b \end{bmatrix}, \vec{\xi} = \begin{bmatrix} \xi_1 \\ \vdots \\ \xi_n \end{bmatrix}, \vec{s} = \begin{bmatrix} s_1 \\ \vdots \\ s_n \end{bmatrix}, \vec{c} = \begin{bmatrix} \vec{0}_m \\ c_1 \\ \vdots \\ c_n \\ \vec{0}_n \end{bmatrix}, \vec{b} = \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix} \in \mathbb{R}^n$$

$$I_{d0} = \left.\underbrace{\begin{bmatrix} 1 & 0 & \dots & 0 \\ 0 & \ddots & \ddots & \vdots \\ \vdots & \ddots & 1 & \vdots \\ 0 & \dots & \dots & 0 \end{bmatrix}}_{m}\right) m, \; Q = \left.\underbrace{\begin{bmatrix} I_{d0} & O_{mn} & O_{mn} \\ O_{nm} & O_{nn} & O_{nn} \\ O_{nm} & O_{nn} & O_{nn} \end{bmatrix}}_{m+2n}\right) m+2n$$

where $m = d + 1$ and O is the zero matrix, noted by its size in its subscript.

4.3 Primal Log Barrier Problem

We cannot directly solve the modified quadratic program(16) because $\vec{\xi}, \vec{s}$ have to be non-negative . To solve this problem, we convert the previous formulation(16) to a primal log barrier formulation(17).This enforces that $\vec{\xi} \geq 0, \vec{s} \geq 0$ and makes it implicit in the fact that the $dom(\log) = (0, \infty)$. We apply the KKT conditions to the conditions to get a linear system(19,20) which leads to the final formulation(21,22).

Adding the log-barrier function gives us

$$\begin{aligned} \min \quad & \vec{v}^T Q \vec{v} + C^T \vec{v} - \beta(\sum \log \xi_i + \sum \log s_i) \\ \text{s.t.} \quad & A\vec{v} = \vec{b} \end{aligned} \tag{17}$$

Note that β is any non-negative constant.

4.3.1 Lagrangian Function

$$L(\vec{v}, \vec{\alpha}) = \frac{1}{2}\vec{v}^T Q\vec{v} + \vec{C}^T\vec{v} - \vec{\lambda}^T(A\vec{v} - \vec{b}) - \beta(\sum_{i=1}^{n} \log(\xi_i) + \sum_{i=1}^{n} \log(s_i)) \tag{18}$$

Notes:

1. Implicitly assumes $\xi_i, s_i > 0$
2. Here β is any non-negative constant or a *penalty* for ξ(or s) being close to 0. The closer β is to 0, the closer the primal log barrier formulation is to the original formulation.

4.3.2 KKT Conditions

1. Stationarity: $Q^T\vec{v} + \vec{c} - A^T\vec{\lambda} - \beta \begin{bmatrix} \vec{0} \\ \frac{1}{\vec{\xi}} \\ \frac{1}{\vec{s}} \end{bmatrix} = \vec{0}$

$$\tag{19}$$

2. Primal Feasibility: $A\vec{v} = \vec{b}$

$$\tag{20}$$

The Dual Feasibility and Complementary Slackness conditions are not applicable here because the constraints only have equalities.

4.3.3 Converting to Linear Formulation

Let $\vec{\lambda} \in \mathbb{R}^n, \vec{\gamma} \in \mathbb{R}^n, \vec{\mu} \in \mathbb{R}^n$, where $\gamma_i = \frac{\beta}{\xi_i}$, $\mu_i = \frac{\beta}{s_i}$

$$\vec{\gamma} = \begin{bmatrix} \frac{\beta}{\xi_1} \\ \vdots \\ \frac{\beta}{\xi_n} \end{bmatrix}, \vec{\mu} = \begin{bmatrix} \frac{\beta}{s_1} \\ \vdots \\ \frac{\beta}{s_n} \end{bmatrix}, \vec{v} = \begin{bmatrix} \vec{w} \\ \xi_1 \\ \vdots \\ \xi_n \\ s_1 \\ \vdots \\ s_n \end{bmatrix} \in \mathbb{R}^{m+2n}$$

1. Stationarity: $Q^T\vec{v} + \vec{c} - A^T\vec{\lambda} - \begin{bmatrix} \vec{0} \\ \vec{\gamma} \\ \vec{\mu} \end{bmatrix} = \vec{0}$
2. Primal Feasibility: $A\vec{v} = \vec{b}$
3. $\gamma_i\xi_i = \beta, \mu_i s_i = \beta$

   Note that the closer β is to 0, the more these look like complementary slackness.
4. $\xi \geq 0, s \geq 0$

Note that these are implicitly required.
Also note: $Q \in \mathbb{R}^{(m+2n)\times(m+2n)}, \vec{v} \in \mathbb{R}^{m+5n}, \vec{c} \in \mathbb{R}^{m+2n}, A \in \mathbb{R}^{n\times(m+2n)}, \vec{b} \in \mathbb{R}^n$

4.3.4 Final Formulation

We can alternatively write the formulation from section 4.3.3 as a function(21) on which we can apply newton's method.

$$\text{Let } \vec{Z} = \begin{bmatrix} \vec{w} \\ \vec{\xi} \\ \vec{s} \\ \vec{\gamma} \\ \vec{\mu} \end{bmatrix} \in \mathbb{R}^{m+5n}, F(\vec{Z}) = \begin{bmatrix} Q^T\vec{v} + \vec{c} - A^T\vec{\lambda} - \begin{bmatrix} \vec{0} \\ \vec{\gamma} \\ \vec{\mu} \end{bmatrix} \\ A\vec{v} \\ \gamma_i\xi_i \\ \mu_i s_i \end{bmatrix}$$

We wish to find a $\vec{Z}^*$ such that

1.
$$F(\vec{Z}^*) = \begin{bmatrix} \vec{0} \\ \vec{b} \\ \beta \\ \vdots \\ \beta \end{bmatrix} = \vec{d}_\beta \tag{21}$$

2.
$$\xi^* \geq 0, s^* \geq 0 \tag{22}$$

Note that the β's will be relaxed or in other words not necessarily equal to the original β.
$\therefore$ Our goal is to find Z^* such that $F(\vec{Z}^*) = \vec{d}_0$ and $\xi^* \geq 0, s^* \geq 0$.

4.4 Generate $\vec{Z}_0$

We are about to apply Interior Point Method to the final formulation(21,22). So, we need a starting point, $\vec{Z}_0$. This starting point may not be in the feasible region. We add appropriate null vectors to make ξ_i, s_i non-negative(22), thus ensuring that we are in the feasible region.

4.4.1 Find $\vec{Z}_0$

In other words, $\vec{Z}_0$ satisfies:

$$Q^T\vec{v} + \vec{c} - A^T\vec{\lambda} - \begin{bmatrix} \vec{0} \\ \gamma \\ \mu \end{bmatrix} = \vec{0}$$

$$A\vec{v} = \vec{b}$$

This can also be written as:

$$M\vec{Z}_0 = \begin{bmatrix} \vec{0} \\ \vec{b} \end{bmatrix} = \vec{d} \tag{23}$$

where $M \in \mathbb{R}^{(m+3n)\text{x}(m+5n)}$.
Note the matrix M might not be invertible. So least squares can be used to find required $\vec{Z}_0$.

4.4.2 Modify $\vec{Z}_0$ such that:

We have two conditions for $\vec{Z}_0$. The first condition ensures that we are close to the root. The second condition ensures that we are in the feasible region. They are as follows.

1.
$$\gamma_i\xi_i \approx \beta, \mu_i s_i \approx \beta \tag{24}$$

2.
$$\xi_i \geq 0, s_i \geq 0 \tag{25}$$

First, we try to achieve condition 2(25) by finding $\vec{r}$ where $M\vec{r} = \begin{bmatrix}\vec{0}\\ \vec{0}\end{bmatrix}$. This means $\vec{r}$ is in the null space of M.

For every negative ξ_i(or s_i), we do the following: we find a null vector $\vec{r}$ with the coordinate corresponding to negative ξ_i(or s_i) $= 1$ and coordinates corresponding to other $\xi_i = 0$ and $s_i = 0$.

$$\text{If } \vec{Z}_{new} = \vec{Z_0} + t\vec{r}, \text{ then } M(\vec{Z_0} + t\vec{r}) = M\vec{Z_0} + tM\vec{r} = \begin{bmatrix}\vec{0}\\ \vec{b}\end{bmatrix} + t\begin{bmatrix}\vec{0}\\ \vec{0}\end{bmatrix} = \begin{bmatrix}\vec{0}\\ \vec{b}\end{bmatrix}$$

where t is a scaling constant. We repeat this until $\xi_i \geq 0, s_i \geq 0$ is satisfied.

Condition 1(24) is taken care of by using Newton's method.

4.5 Pseudocode

We cannot find an exact root for this problem, so we introduce an error bound and if we are within the error bound, we stop. We call the error bound tolerance. We wrap up our algorithm as a pseudocode below.

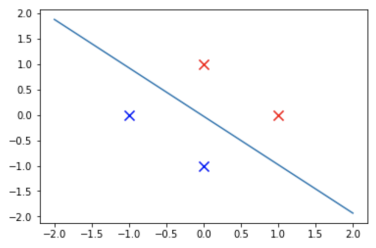
```
Algorithm 1 Implementation
Require: $|F(\vec{Z}) - \vec{d_0}| < tolerance$
Ensure: $\xi_i, s_i \geq 0$
  1. Initialize
        1.1 Take n points as input
        1.2 Take tolerance as input
        1.3 $\beta \leftarrow \beta_0$
  2. $\vec{Z} \leftarrow$ Generate $\vec{Z_0}$
        2.1 Find $\vec{Z_0}$ such that $M\vec{Z} = \vec{\vec{d}}(23)$
              2.1.1 If M is invertible, then: $\vec{Z_0} = M^{-1}\vec{\vec{d}}$
              2.1.2 Else: use standard least squares method to find $M$
        2.2 Modify $\vec{Z_0}$
              2.2.1 Add appropriate Null Vector according to section 4.4.2.
              2.2.2 Use Newton's Method
  3. Iteratively update $\vec{Z}$ using Interior Point Method until required condition(first line of Algorithm 1) is reached
        3.1 Update: $\beta \leftarrow \beta *$ discount factor
        3.2 $\vec{Z} \leftarrow$ Do one iteration of Newton's Method using $\vec{d_\beta}(2)$
  4. Return $\vec{Z}$
```

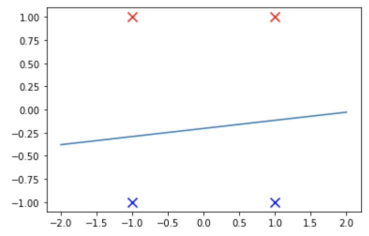
5 Results

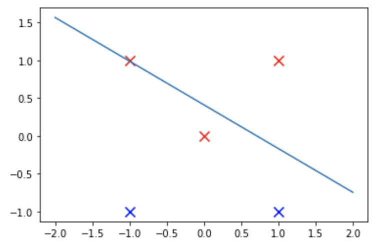
Here is a table that summarizes the results of implementing the algorithm on custom sample cases and a data set consisting of 862 points [5] [4]. The codes can be found here[2][3].

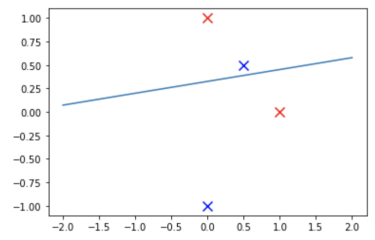
Dataset	Number of features	Number of points	Tolerance	Number of IPM iterations	Time
Sample 1	2	4	0.1	100	941 milliseconds
Sample 2	2	4	0.1	100	1.12 seconds
Sample 3	2	5	1	100	2.24 seconds
Sample 4	2	4	0.1	100	918 milliseconds
Four Class	2	7 out of 862	0.1	100	23.1 seconds

Here are pictures of the samples and the data set.


(a) Sample 1


(b) Sample 2


(c) Sample 3


(d) Sample 4

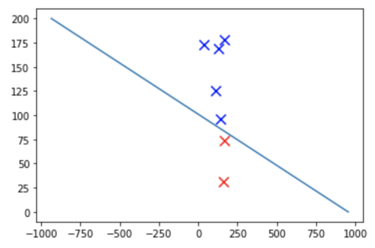


Figure 5: Four Class

5.1 Discussion of the Challenges

1. Our Interior Point Method is not very efficient. As we increase the input size, the matrices become huge and the number of computations drastically increases.
2. The algorithm is not guaranteed to converge.
3. A standard initial β might not work for all types of input.
4. The $\vec{Z}$ has to be in the vicinity of the optimal solution. Otherwise, it might go to a suboptimal solution.

6 Future Works

This section is written in response to the discussion of challenges in section 5.1. Each item is a response to the respective item in section 5.1. These are possible works in the future as a continuation of this project.

1. The algorithm in this paper can be optimized. We can try to figure out a way to efficiently calculate the least squares and the Jacobian matrix,which are causing such a long time to classify the points.
2. We can search for proofs (if there are any) that talk about converging to an optimal solution.
3. We can experiment more thoroughly with a wider range of βs.
4. We can research on methods to ensure that $\vec{Z}$ has to be in the vicinity of the optimal solution if it is possible.

References

[1] Boyd and Vandenberghe. Interior point method. https://web.stanford.edu/class/ee364a/lectures/barrier.pdf, 2019.

[2] Derrick Kim. Code for fourclasses. https://github.com/egk265/SVM_IPM/blob/master/code_for_four_class.py, month = , year = 2019,.

[3] Derrick Kim. Code for samples. https://github.com/egk265/SVM_IPM/blob/master/code_for_samples.py, 2020.

[4] Chih-Jen Lin. Libsvm four class. https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary/fourclass, 2019.

[5] Chih-Jen Lin. Libsvms binary class. https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary.html#fourclass%E2%80%A9, 2020.

[6] Ameet Talwalkar. Support vector machines. https://www.cs.cmu.edu/~atalwalk/teaching/winter17/cs260/lectures/lec11.pdf, 2019.