Supporting Vector Machines

ECE273 Final Project

Yuanping Zhang yuz613@eng.ucsd.edu A53224871
Boyang Zhang boz083@eng.ucsd.edu A53248620

Project Objective

The objective of this project is to create a supporting vector machine that can be applied to linearly separable, linearly non-separable, and non-linear dataset for binary classification. The algorithm is written in Matlab and uses CVX toolbox for solving convex optimization problems.

SVM Introduction

Supporting vector machine is a supervised learning algorithm typically used for binary classification. It provides a decision boundary that separate the data into two categories. The goal is to find the decision boundary that maximizes the margin, i.e. the distance of the closest points in each category to the decision boundary.

SVM has a few properties that makes it a popular model for classification applications. The complexity of the program does not depend on the dimension of the data points. The solution only depends on key data points (support vectors), even when the dataset is large. The problem can be convert into a convex optimization problem, for which we have tools to directly solve it. The algorithm can be applied to data of different features (linear, non-linear) with minor adjustments. These properties will be shown in more details in the next section.

These properties allow SVM to be implemented in various real-world applications. Here are some of the examples:

- Text and hypertext categorization
- Image classification (objects of interest vs. background information)
- Hand-written character recognition (signature authentication)
- Bioinformatics (protein classification, cancer classification)

Methods and Algorithms

Our approach to creating the support vector machine model is to convert the problem of maximizing the decision boundary into a convex optimization problem, then solve it using the CVX toolbox. First, we need to understand the three different cases of SVM problems.

i. Linearly Separable

For each point in the dataset, $(\mathbf{x}_i, \mathbf{y}_i)$, where $\mathbf{x}_i \in R^n$ and $\mathbf{y}_i \in \{-1, +1\}$. In this case, where the dataset is linearly separable, there exists some hyperplane that separates points with y = -1 from y = +1. We suppose all data points satisfy the following inequality constraints:

$\mathbf{w}^T\mathbf{x}_i + b \geq 1$ for $\mathbf{y}_i = 1$

$\mathbf{w}^T\mathbf{x}_i + b \leq -1$ for $\mathbf{y}_i = -1$

That is all points with +1 y-value are on one side of (or on) the hyperplane H_1: $\mathbf{w}^T\mathbf{x}_i + b = 1$. All points with -1 y-value are on the other side of (or on) the hyperplane H_2: $\mathbf{w}^T\mathbf{x}_i + b = -1$. $\mathbf{w}$ is the vector orthogonal to the decision boundary. Our goal then becomes to maximize the margin between two hyperplanes. Consider two points that are

on H1 and H2 respectively, $p_1 = (x^+, 1)$ $p_2 = (x^-, -1)$. The margin between the two hyperplanes has the form:

$$M = \frac{(x^+ - x^-) \cdot w}{|w|} = \frac{2}{|w|}$$

To maximize the margin is equivalent to minimizing: $0.5\mathbf{w}^T\mathbf{w}$

Notice that we can simplify the constraint by combining the two inequalities above into one: $\mathbf{y_i}(\mathbf{x_i} \bullet \mathbf{w} + b) \leq 1$. Now constructing SVM becomes the following quadratic optimization problem and solve for **w** and b.

Minimize $0.5\mathbf{w}^T\mathbf{w}$

Subject to $\mathbf{y_i}(\mathbf{x_i} \bullet \mathbf{w} + b) - 1 \leq 0 \quad \forall i$

Notice that both the objective function and the constraints are convex. Therefore, we can write it as a dual problem by introducing positive Lagrange multipliers α_i and corresponding KKT conditions. The stationarity in the KKT requires the gradient of the Lagrangian equals to zero with respect to **w** and b.

Maximize:

$$g(\boldsymbol{\alpha}) = \sum_i \alpha_i - \frac{1}{2}\sum_{i,j} \alpha_i \alpha_j y_i y_j \boldsymbol{x_i} \cdot \boldsymbol{x_j}$$

KKT:

$$\boldsymbol{w} = \sum_i \alpha_i y_i \boldsymbol{x_i}\ ;\ \sum_i \alpha_i y_i = 0;$$

$$\alpha_i \geq 0;$$

$\mathbf{y_i}(\mathbf{x_i} \bullet \mathbf{w} + b) - 1 \leq 0 \quad \forall i$

$\alpha_i(\mathbf{y_i}(\mathbf{x_i} \bullet \mathbf{w} + b) - 1) = 0 \quad \forall i$

Notice that we only have one set of Lagrange multipliers since there is no equality constraints in the primal problem. Once we have the problem in this dual form, we can use CVX to solve the convex optimization problem of minimizing $-g(\boldsymbol{\alpha})$ subject to the constraints.

In the solutions, we only need to keep the points which have $\alpha_i > 0$, namely the supporting vectors. They determine the hyperplanes H_1, H_2. For the other points which have $\alpha_i = 0$, removing them will not change the solution. This "sparseness" property is one of the advantage of SVM. We only need certain points even when the dataset is large.

Once we have solved for all non-zero α_i, we can obtain the decision boundary by calculating **w** and b.

$$\boldsymbol{w} = \sum_i \alpha_i y_i \boldsymbol{x_i}$$

$$b = y_i - \boldsymbol{w}^T \boldsymbol{x_i} \quad \forall i, s.t.\ \alpha_i > 0$$

After successfully trained the SVM by solving the convex problems mentioned above, we can test the accuracy of our SVM by comparing the sign of y-values from a set of testing points with $\text{sgn}(\mathbf{w}^T\mathbf{x}+b)$, i.e. if the point is lying on the correct side of the decision boundary.

ii. Linearly Non-Separable

When there is noise in the dataset, we cannot linearly separate all the points with negative y-value from positive ones in the dataset. The previous algorithm will yield no feasible solution. We need a new algorithm to tolerate these points. We refer to this version of SVMs as soft-margin.

To tolerate these points, we introduce slackness variables ε_i to relax the inequality constraints:

$\mathbf{y_i}(\mathbf{x_i} \bullet \mathbf{w} + b) - 1 + \varepsilon_i \leq 0 \quad \forall i$

The sum of all slackness variables represents an upper bound of training errors. We can choose a parameter C to assign the penalty to errors. Higher C corresponds to higher penalty. The objective function becomes:

$$0.5\mathbf{w}^T\mathbf{w} + C\sum_i \varepsilon_i$$

If we follow the same steps as above to write the problem in dual form, the slackness variable will disappear, but $\alpha_i$now has an upper bound C.

Maximize:

$$g(\boldsymbol{\alpha}) = \sum_i \alpha_i - \frac{1}{2}\sum_{i,j} \alpha_i \alpha_j y_i y_j \boldsymbol{x_i} \cdot \boldsymbol{x_j}$$

Subject to:

$$\sum_i \alpha_i y_i = 0;$$

$$C \geq \alpha_i \geq 0;$$

The solution still has the same form (same way to calculate **w** and b). This means the only change we have to make in our algorithm is adding a upper bound for α_i when solving the convex optimization problem.

iii. Non-Linear

When the decision function is not linear, we can use the kernel trick. To observe more obvious separation between points in the dataset, we map them into higher dimension $\mathcal{H}$:

$$\phi(\boldsymbol{x}): \mathbb{R}^n \rightarrow \mathcal{H}$$

Recall that in the linear case, the training points only exist in the dot products in both dual form and the solution. This property allows us to use the dot product of the mapping of each point in non-linear cases, which are called kernels. We do not need to know the mapping explicitly and can train the SVM in the same way as above. The resulting **w** will also be in the higher dimension $\mathcal{H}$. However, this doesn't affect the training result (if the mapping created enough separation between points of two classification).

In order to make sure there exists a pair of dimension and mapping function, the kernel function has to satisfy the Mercer's condition. The focus of this project is on creating a functional SVM algorithm, so we will not delve into details of Mercer's condition, but instead limit our choice of kernels to the following common types:

Polynomial: $K(\boldsymbol{x_i}, \boldsymbol{x_j}) = (1 + \boldsymbol{x_i^T x_j})^p$

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

Sigmoidal: $K(\boldsymbol{x_i}, \boldsymbol{x_j}) = \tanh(ß_1 + ß_0 \boldsymbol{x_i^T x_j})$

In our algorithm, the choice of kernels includes linear (typical dot product), polynomial, and RBF. The results are shown in the following sections.

Dataset

For the sake of controllability on examining the accuracy and robustness of the our SVM algorithm, we generated synthetic data for all three cases.

For the linear case, we generate a set of points in $\mathbb{R}^2$ with coordinates randomly chosen between 0 and 100. The points with coordinates sum up to larger than 100 are classified as +1, and the rest are classified as -1. For a clear observation of the linear separable case, we also manually created a margin between two types by increasing or decreasing coordinates of each point by 10(increase for +1 type and decrease for -1 type).

For the linearly non-separable case, we used the same method as above to generate data points. To generate noise, instead of creating a clear margin between the two types, for each point in the +1 category we subtracted one of the coordinate's value by 5, so there will be points of +1 class in the region where the sum of coordinates is smaller than 100.

For the non-linear cases, we again use random points with coordinates between 0 and 100. For a point (x_1,x_2), if $x_2 <- |x_1 -$

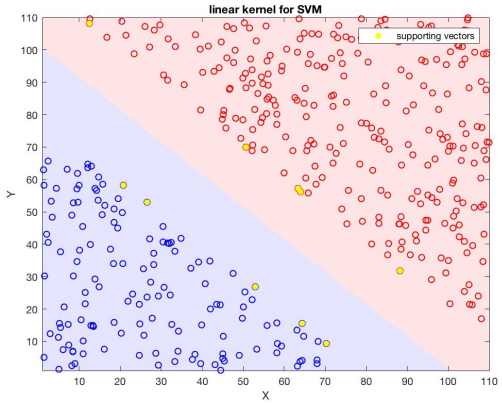
$50| + 33$, it is classified as +1, and -1 otherwise, i.e. whether the point is above or below an upside-down triangle.

Another dataset we use is separate the two types of point according to a circle that's centered at (50,50). Data points are classified as +1 when inside the circle and -1 when outside the circle.

Results

i. Linearly Separable

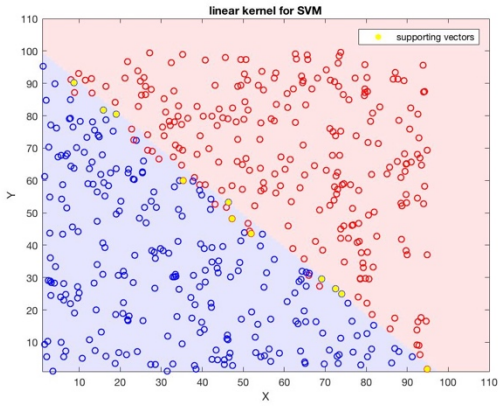
This first graph shows the result of applying our SVM algorithm on linearly separable dataset with 500 points.



The red dots are the points of category +1 and blue dots are points of category -1. The support vectors are highlighted in yellow. The line between the red and blue region is the decision boundary. Since we have control over generating training data, we know that the decision boundary should be the hyperplane where the sum of the point's coordinates equal to 100. The SVM performed as we expected.

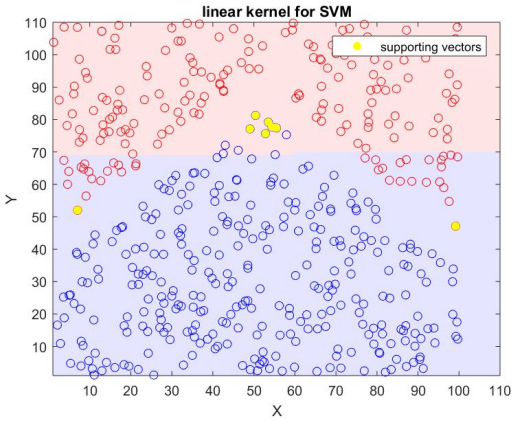
ii. Linearly Non-separable

The figure below shows the result of applying our SVM to 500 linearly non-separable data points.



Although there are training points in the wrong region, the slack variables tolerate the error and allow the algorithm to find the optimal decision boundary. This solves the overfitting issue in a linearly non-separable dataset.

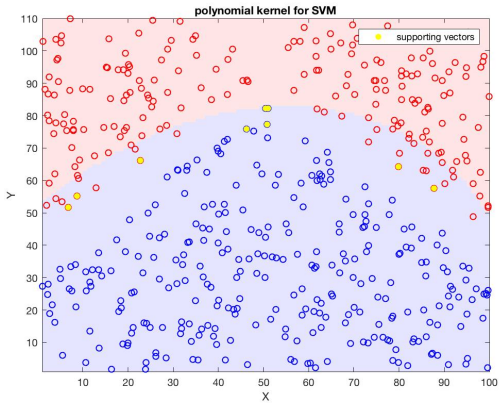
For dataset that are non-linear, there will be too much "noise" when we use linear kernel, even though with the slack variables still allow us to find a decision boundary. In the figure below, the training dataset is classified using the absolute value function mentioned in the last section. It is obvious to observe that the training result is not ideal.



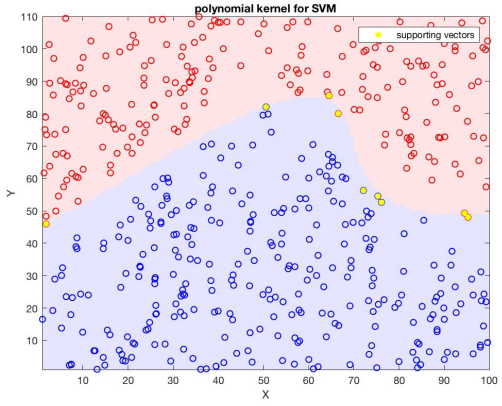
iii. Polynomial Kernel

For these cases, the non-linear kernels generate much better results. The figure below shows the training result of SVM with polynomial kernel of power 2. We can see

that the result is much better than the linear case. We can also observe the characteristic of a polynomial function. The decision boundary roughly resembles a second-degree polynomial function. This also leads to the inaccuracy of the result, (some of the points of +1 class end up in the blue region when far from center), since our generated data has a faster drop off away from the center.

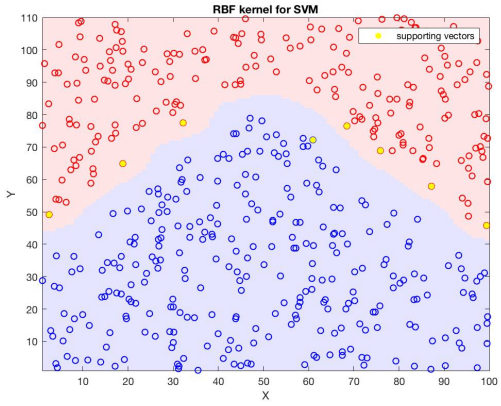


The dependence on choice of power in the polynomial is more obvious when we increase the index. The figure below shows the training result with polynomial of degree 4.

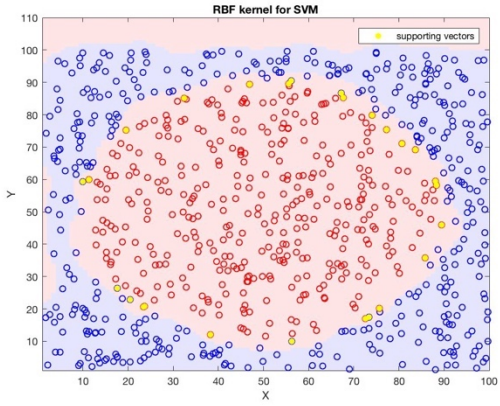


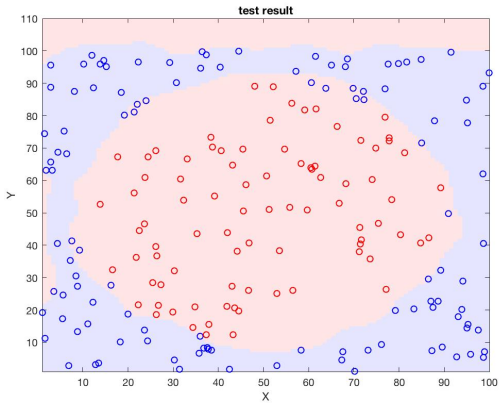
iv. Gaussian RBF Kernel

Another kernel we used for nonlinear data is Gaussian radial basis function. Using the same dataset mentioned above, the decision boundary resembles the absolute value function closer, since it is not limited to the polynomial function. The supporting vectors are clearly situated on either sides of the decision boundary and maximizing the margin between.

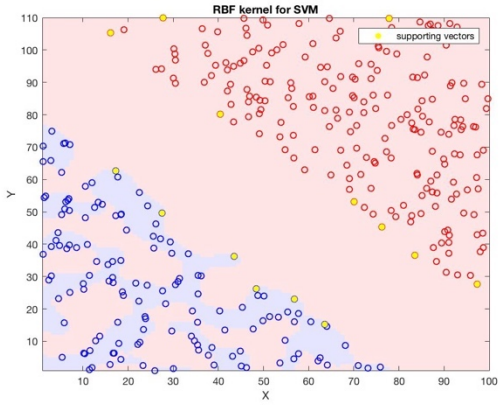


The benefit of using RBF kernel is more evident when the dataset does not resemble any polynomials. For example, the figures below show the training and testing results of applying SVM with RBF kernel on the circular dataset mentioned in the last section. SVM is able to find the decision boundary accurately. This reflects in the testing with minor misclassification. The training set has 800 points and testing set has 200 points.



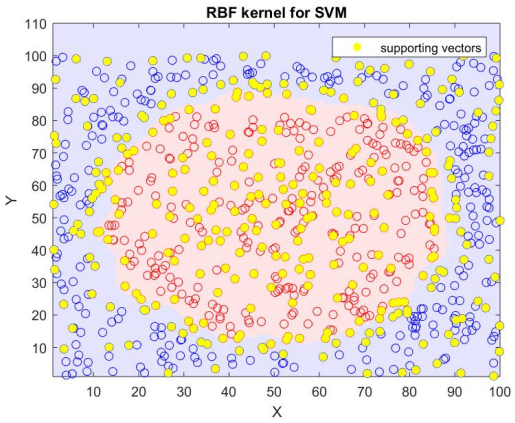


However, RBF kernel also has its downside. The computing time is much longer than linear and polynomial kernels. When the data is linear, RBF kernel's training result is also not as accurate as the simple linear kernel. The figures below show the training result of applying RBF kernel on linearly separable dataset. It is clear to observe that the error of mislabeling points of -1 class is very high in this case. The RBF kernel is trying to develop a decision boundary that accurately envelops every single data point in this scenario. With more training points the results might be more accurate, but the computing time will also be significantly longer. This shows the importance of having a general idea of the dataset and choose the right kernel for the SVM.

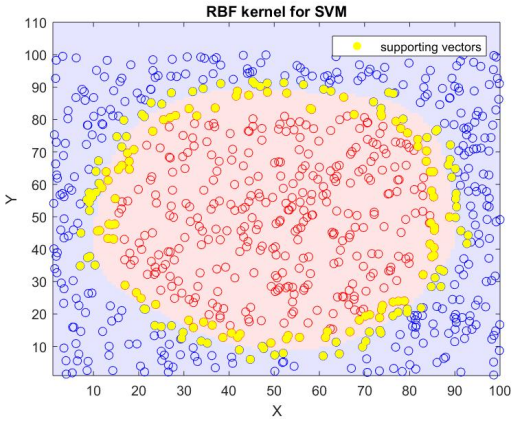


v. σ-value for RBF Kernel

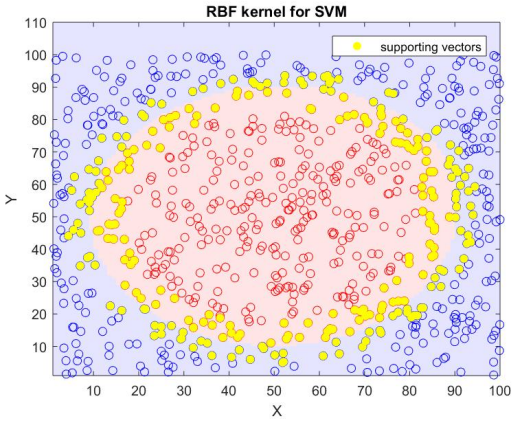
For RBF kernel we have the choice of σ values and it is important to understand the effect on the training result. The three figures below are all training results using the same dataset and RBF kernel but with different σ value.



σ =5



σ =15

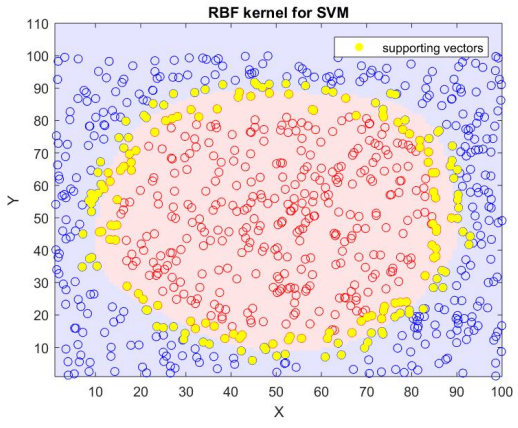


σ = 50

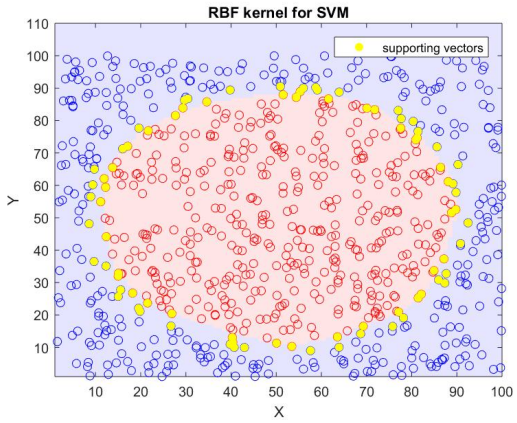
Comparing the three figures side by side, it is easy to see there is an optimal choice of σ for efficiency, i.e. least amount of support vectors while determining the decision boundary correctly. When the σ-value is small, there are more support vectors and the distance from the decision boundary is also larger on average. When σ-value is large, there are also a large number of support vectors but situated closer to the decision boundary in the R^2 dimension. Considering the boundary for classification is well defined in the dataset in this case, the decision boundary is not largely affected by the σ-value. However, when the boundary becomes harder to find for RBF kernel, σ will play an even more important role in optimizing performance of the SVM. For this particular dataset, the optimal value for σ is around 15.

vi. Constraints on Slackness Variable

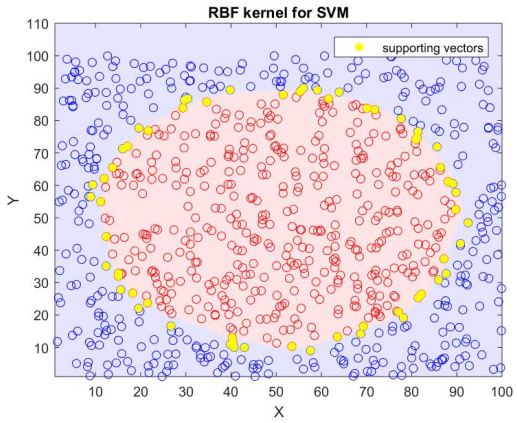
As discussed in the algorithm section, the parameter C controls how much training error we tolerate. A higher C represents a harder margin. The results are shown below using RBF kernel. We chose σ = 15 for all three cases below for optimal results.



C = 1



C = 10



C = 25

As C increases the number of support vectors decreases. We can interpret it as the margin of the two hyperplanes separating the data points of two category in higher dimension decreases as C increases, i.e. we are getting a harder margin. Again, in this case, the RBF kernel allows the SVM to find the decision boundary accurately, so increasing C does not affect the result significantly. The choice of C is a trade of between the efficiency of the SVM (fewer support vectors) and the capability of the SVM to successfully identify the decision boundary. When the dataset becomes even more complicated, it is important to iterate over C to find the optimal choice.

vii. Higher Dimensions

As mentioned in the introduction section, the algorithm we are using have no limit on the dimension of the dataset. Since it is difficult to represent 3 or higher dimensional results on this 2-d paper, we test the accuracy of our training results by calculating the percentage of mislabeled points when applying our SVM with corresponding testing dataset.

Using the linear kernel, the SVM trained and tested with 3-D version of the linearly separable dataset results in an accuracy of 100%. It is expected consider the large margin we manually created.

For 3-D linearly non-separable data, the SVM results in 93.8% accuracy.

For SVM with RBF kernel, it was able to achieve 97% accuracy on the linearly non-separable dataset, and 95% on 3-D sphere dataset.

Conclusion

In conclusion, we successfully created a SVM algorithm for both linear and non-linear dataset for binary classification. We also examined the effect of different parameters in each method and identified the optimal choice for each parameter. There are a few directions that we believe are interesting for future exploration.

All of the dataset we used in this project were synthetically generated, for the sake of exploring the robustness and accuracy of the SVM in specific areas. It will be interesting to see how the model perform with real world data or dataset has more irregularities.

Another interesting area that one can expand on this project is finding more rigorous methods of finding the right choice of kernels and parameters such as σ or C, instead of iterating all possible values.

Especially when the data becomes more complicated in higher dimensions, we can imagine even RBF kernel would fail or requires a significant computing time. Having a method for choosing kernels will be very important in that case.

References

1. C. J. C. Burges, "A Tutorial on Support Vector Machines for Pattern Recognition", *Data Mining and Knowledge Discovery*, vol. 2, no. 2, pp. 121-167, June 1998.
2. I. Steinwart and A. Christmann, "Support Vector Machines", Springer, 2008.
3. Michael P. S. Brown William Noble Grundy, David Lin, Nello Cristianini, Charles Sugnet, Manuel Ares, Jr., David Haussler, "Support Vector Machine Classification of Microarray Gene Expression Data"
4. T. Joachims*, "Text categorization with Support Vector Machines: learning with many relevant features"* , ECML – 98
5. Lipo Wang, *"Support Vector Machines: Theory and Applications"*