# *Support Vector Classification*

Machine Learning: Project 5

*Hossain, Rayhan (rhossai2)*

# Table of Contents

## Introduction

Support Vector Machine (SVM) is a simple but very effective algorithm that every machine learning expert should have in his/her arsenal. Support vector machine is highly preferred by many as it produces significant accuracy with less computation power. Support Vector Machine, abbreviated as SVM can be used for both regression and classification tasks. But, it is widely used in classification objectives. A Support Vector Machine is a discriminative classifier formally defined by a separating hyperplane. In other words, given labeled training data (supervised learning), the algorithm outputs an optimal hyperplane which categorizes new examples. In two-dimensional space this hyperplane is a line dividing a plane in two parts where in each class lay in either side.

For class project-5, this uses the benefits of SVM to classify several data sets. There was no condition given for implementing own Support Vector Classification algorithm. Therefore, developer had an option for multiple choices to experiment the data and report the classification result. Among the given problems, one was binary classification, and another two were multiple class problems. It requires both of the coarse grid search and fine grid search to experiment the data and find the best range and optimal hyper parameters.

## Dataset

The goal of this project was to apply the SVC algorithm to three different datasets. One of the problem is binary classification problem, and another two are multiple class problems. After applying the SVC, we need to measure the accuracy and the performance matrix and describe the result.

For the ionosphere dataset, there were 34 data features and two result classes. The result class labels are ‘g’ and ‘b’ which represent good and bad respectively. The name attributes were given in a separate file. But, to analyze the data, new labels were created for each data column. For the other two problems, data are almost similar. One noticeable difference is, they were with

multiple result classes. For the second problem, among the data features, three were totally irrelevant to the actual result class of the data. For the last one- Satellite dataset, the dataset is much larger. There were seven classes but 36 features for each instance. We had to determine the type of terrain from multispectral values of pixels in 3x3 neighborhoods in satellite images.

## Data Preprocessing

The given dataset is almost clean, and there is no missing data. So, there was no hassle to work with this data set. One problem is, for different data column the range was so different. So, it is necessary to do data standardization for getting better accuracy. Doing the data standardization is not a hard task. It is handled using one of the stats library. Another thing to mention, for ionosphere dataset, one column has all the values zero. So, this column was removed considering that it has no effect on the result. For the sake of simplicity to compare, the result class values are converted into binary value 0 or 1. Here, 1 represents the good class, and 0 represents the bad class. For the Vowel-Context dataset, data are different from previous in number of classes and number of features for each data point. We had 11 classes only 13 features with 3 of them being irrelevant. So, after removal, we were left with only 10 attributes. As in previous experiment our data was shuffled and split into training and validation/testing datasets.

For the data preprocessing part, task was easier to handle using the pandas data frame.

## Support Vector Machine and SVC

The objective of the support vector machine algorithm is to find a hyperplane in an N-dimensional space (N — the number of features) that distinctly classifies the data points. To separate the two classes of data points, there are many possible hyperplanes that could be chosen. Our objective is to find a plane that has the maximum margin, i.e. the maximum distance between data points of both classes. Maximizing the margin distance provides some reinforcement so that future data points can be classified with more confidence.

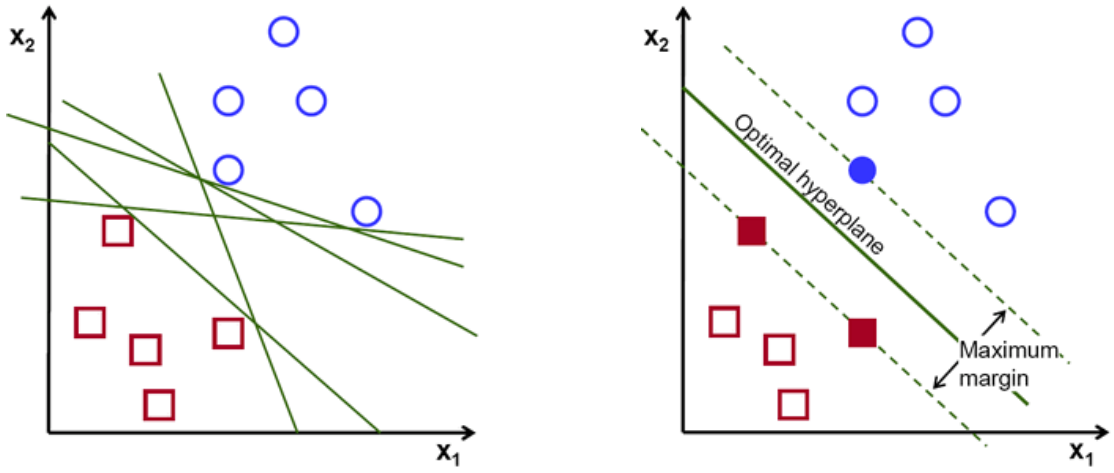


Figure-1: Possible hyperplanes

Hyperplanes are decision boundaries that help classify the data points. Data points falling on either side of the hyperplane can be attributed to different classes. Also, the dimension of the hyperplane depends upon the number of features. If the number of input features is 2, then the hyperplane is just a line. If the number of input features is 3, then the hyperplane becomes a two-dimensional plane. It becomes difficult to imagine when the number of features exceeds 3.

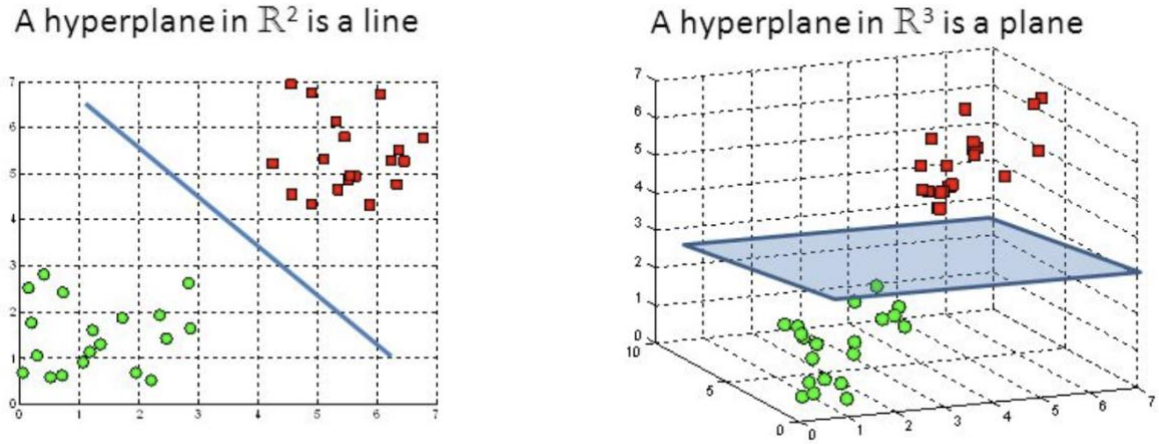


Figure-2: Hyperplanes in 2D and 3D feature space

We can also discuss a little bit more about hyperplane from the mathematical perspective. We can describe it in the following equation: $y = w^T x + b$,

where y value indicates classification label, w's are parameters of the plane, so is b which moves it in and out of the origin. This is how linear classifiers look like even in multiple dimensions with hyperplanes. The decision boundary which is the red line in our example again may not give us positive or negative values for classification by the definition, so no matter what b and w values are, it is: $w^T x + b = 0$.

For other lines, y will be described by the offset from the middle line. Two possible results could be: $w^T x + b = 1$, $w^T x + b = -1$, for lines closest to each class.

## Confusion and Performance Matrix

To evaluate the performance of our classification algorithm, we needed to compare the correct classification $\{r^t\}_{t=1}^N$ with our classifications $\{y^t\}_{t=1}^N$ (on the training, evaluation, or testing data). Therefore, implement a function that prints performance metrics determined as follows. Let TP = # true positives, TN = # true negatives, FP = # false positives, FN = # false negatives. We can print out a *confusion matrix* formatted as follows:

| | **Predicted Class** | |
|---|---|---|
| **True Class** | **benign** | **malignant** |
| **benign** | TN | FP |
| **malignant** | FN | TP |

In addition to the confusion matrix, we can also use following performance metrics:

- Accuracy = (TN + TP) / (TN + TP + FN + FP)
- TPR (true positive rate, recall, or sensitivity) = TP / (TP + FN)
- PPV (positive predictive value or precision) = TP / (TP + FP)
- TNR (true negative rate or specificity) = TN / (TN + FP)
- F1 Score = 2 × PPV × TPR / (PPV + TPR)

## Implementation and Evaluation

For the implementation, Python with Jupyter Notebook was chosen. For the SVM model, and coarse and fine grid search methods from scikit-learn were invoked. For doing the data standardization part, StandardScaler helps in a great way.

There are four kinds of known kernels that we can use in SVC-

- Linear
- Polynomial
- Radial Basis Function (RBF), defines a spherical kernel, and
- Sigmoid, has same shape with sigmoid

For choosing the value of C and gamma we need to be careful. Gamma can be also called as the 'spread' of the kernel, or kernel scale. It is a decision region. If Gamma is low, then the curve of the decision boundary is very low and therefore the decision region is too broad. From other side, when Gamma value is high, the curve is high, and we receive islands of decision-boundaries around datapoints. C parameter is the penalty for misclassifying datapoint. When it's low, we have high bias and low variance, so classification isn't penalized.

At the beginning, to understand the relations of ionosphere dataset, different columns were clustered in a two-dimensional space. It was basically done to identify the feature dependency. However, clustering two attributes or features at a time was not a fruitful idea. After analyzing the cluster results, it was clear that it would not help anyway to our real classification. Because, observing two attributes at a time was not helping to get the deeper dependency of the all data attributes. That's why, only for the first problem- the ionosphere dataset this cluster approach was applied. For the other two parts of the problems it was ignored. The final results of the clusters are given below. For the first one, we compared col_6 vs col_7. Then we compared col_5 vs col_20, col_10 vs col_15, and col_33 vs col_34 respectively. And, these classifications were done based upon the result class- good or bad.

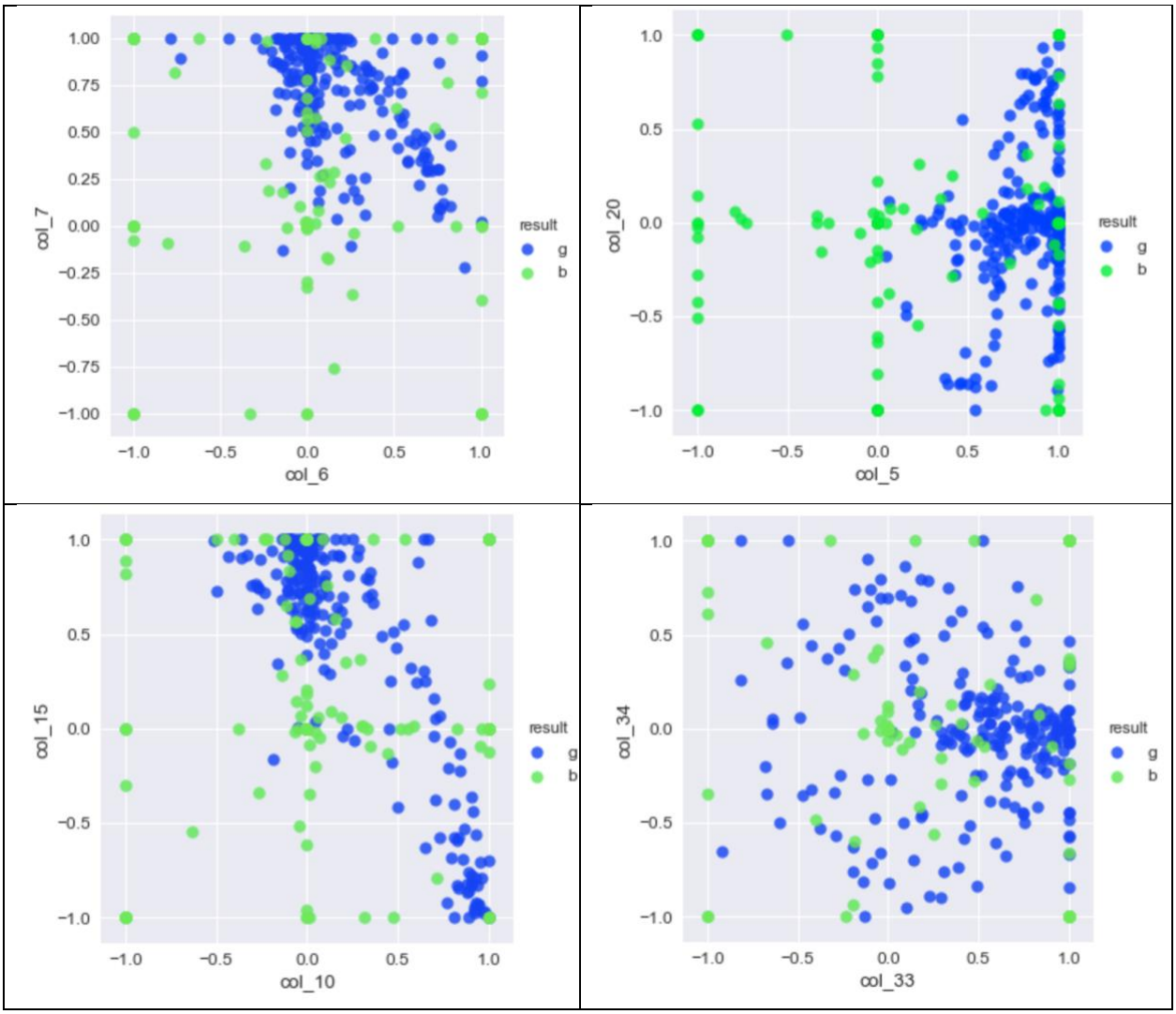


Figure-3: Classifications of two data attributes for ionosphere data

After finding that it will not help directly, we dive into the SVM modeling with all our relevant dataset. Once fitting our model, the coarse grain search was applied with different values of gamma and C. The model was fitted with the training data and tested with the validation dataset. The best point is, the accuracy for all of the values of gamma and C were above 95%. So, it was a good model. The value of the gamma and C which were chosen are given below. The values for coarse grid search is shown first.

| Gamma | 0.00001 | 0.0001 | 0.001 | 0.01 | 0.1 | 1 | 10 | 100 | 1000 | 10000 |
|---|---|---|---|---|---|---|---|---|---|---|
| C | 0.00001 | 0.0001 | 0.001 | 0.01 | 0.1 | 1 | 10 | 100 | 1000 | 10000 |

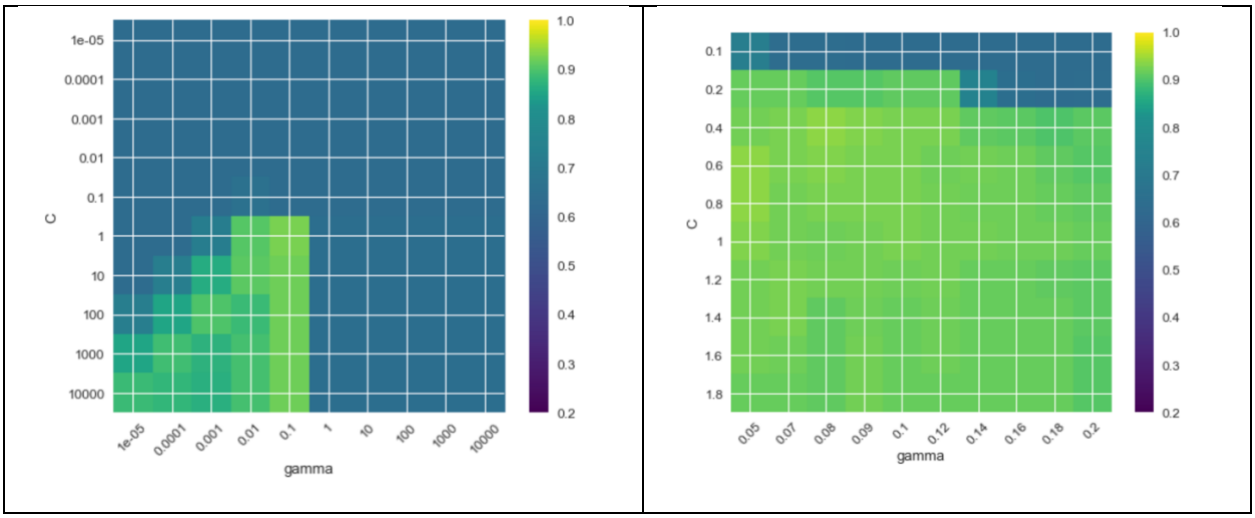


Figure-4: Problem-1 Coarse vs Fine grid search

In the above figure-4, the result of coarse vs fine grid search for ionosphere data is shown. Left side one is for coarse grid, and the right side one is for fine grid search. Here if we analyze the data, for the coarse grain, the best value of Gamma and C was 0.1 and 1 respectively. Then to look into the dipper slice, a fine parameter set was prepared. The fine parameter set for Gamma and C which was used to test is given below.

| Gamma | 0.05 | 0.07 | 0.08 | 0.09 | 0.1 | 0.12 | 0.14 | 0.16 | 0.18 | 0.20 |
|---|---|---|---|---|---|---|---|---|---|---|
| C | 0.1 | 0.2 | 0.4 | 0.6 | 0.8 | 1 | 1.2 | 1.4 | 1.6 | 1.8 |

Depending upon the best value for the coarse grid, I chose the fine grid values for Gamma and C. Applying the fine grid search, the best result was found for Gamma = 0.08, and C = 0.4 which is almost at the top left corner.

In the same way, coarse and fine grid search was applied for the problem 2 and problem 3. The results are shown below.

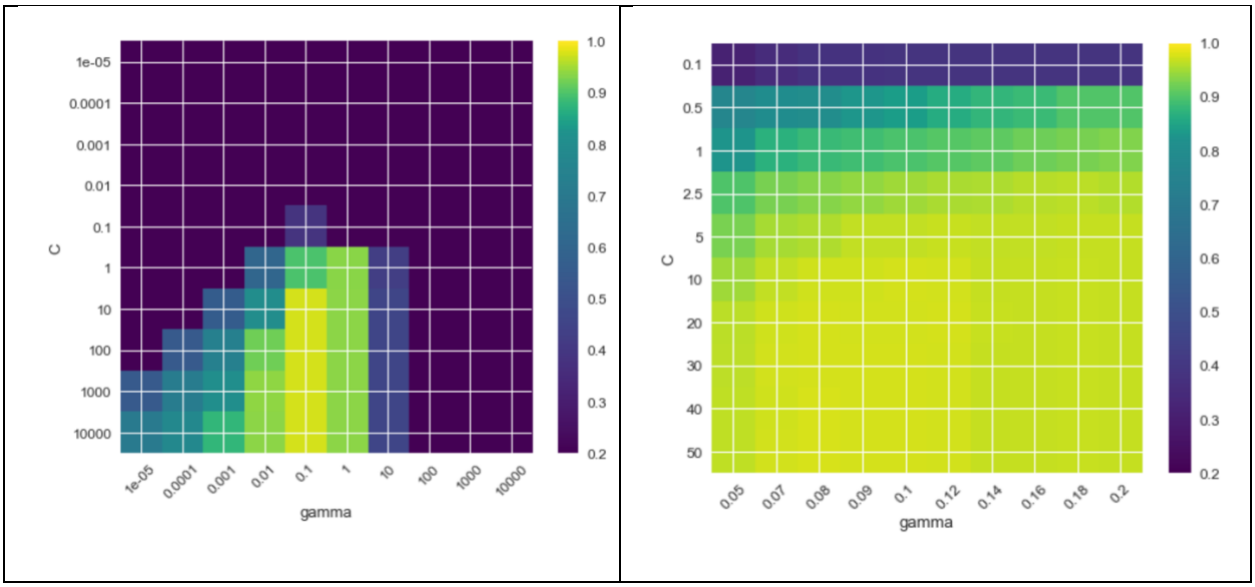


Figure-5: Problem-2 Coarse vs Fine grid search

Here for the coarse grid best of gamma and c are 0.1 and 10 respectively. The precision table is like the following one.

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0.0 | 1.00 | 1.00 | 1.00 | 25 |
| 1.0 | 0.90 | 1.00 | 0.95 | 18 |
| 2.0 | 1.00 | 1.00 | 1.00 | 29 |
| 3.0 | 0.96 | 1.00 | 0.98 | 25 |
| 4.0 | 1.00 | 0.96 | 0.98 | 26 |
| 5.0 | 0.95 | 0.91 | 0.93 | 22 |
| 6.0 | 1.00 | 1.00 | 1.00 | 21 |
| 7.0 | 1.00 | 1.00 | 1.00 | 18 |
| 8.0 | 1.00 | 1.00 | 1.00 | 21 |
| 9.0 | 1.00 | 0.88 | 0.94 | 26 |
| 10.0 | 0.89 | 1.00 | 0.94 | 17 |
| avg / total | 0.98 | 0.98 | 0.98 | 248 |

For the fine grid the best value for gamma and C are 0.08 and 40 respectively. Finally, the results for the third problem is shown below.

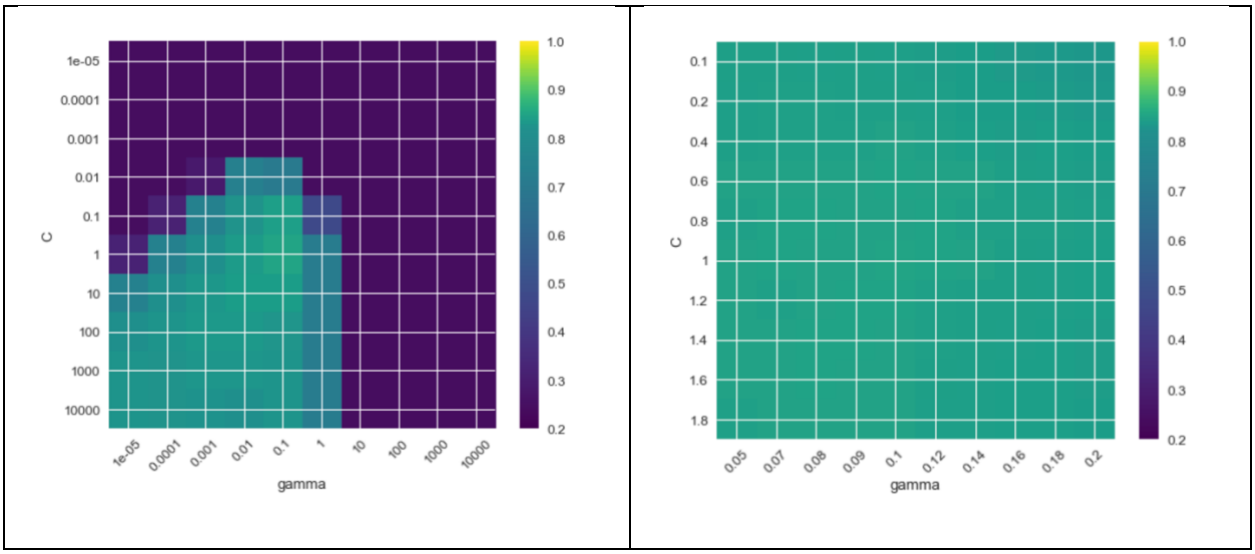


Figure-6: Problem-3 Coarse vs Fine grid search

At the final phase, I was little bit confused about the fine grid result of the third problem. It was tough to anticipate the result seeing the heat map graph. However, from the code result the best value for fine grid (right side) is, Gamma = 0.1 and C = 1.

The fine grid matrix is given below-

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 1.0 | 0.98 | 1.00 | 0.99 | 461 |
| 2.0 | 0.97 | 0.98 | 0.97 | 224 |
| 3.0 | 0.87 | 0.97 | 0.92 | 397 |
| 4.0 | 0.77 | 0.63 | 0.69 | 211 |
| 5.0 | 0.94 | 0.90 | 0.92 | 237 |
| 7.0 | 0.89 | 0.87 | 0.88 | 470 |
| avg / total | 0.91 | 0.91 | 0.91 | 2000 |

## Conclusion

In conclusion, it can be summarized that Support Vector Machine is great for dealing with high-dimensional data and produces best results in short time. Handling the algorithm for training and validation is pretty much easier. The only thing is we need to choose the C and gamma values carefully. Additionally, it also works well some smaller data size.

Note:

The Support Vector Classification algorithm is a complex concept and is little bit tough to implement from scratch. But, fortunately, we were guided to use the scikit-learn library functions. As our main purpose was to experiment the data using 3rd party library function, I searched for tutorial to learn the use of predeveloped codes. I found some YouTube videos and GitHub codes where they show the exact same work. After learning from those resources, I fit the model to our data and report the experiment result.

## Reference

[1] https://medium.com/machine-learning-101/chapter-2-svm-support-vector-machine-theory-f0812effc72

[2] https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47

[3] https://github.com/adashofdata/muffin-cupcake

[4] https://github.com/adashofdata/muffin-cupcake

[5] https://scikit-learn.org/stable/