

Kernel as the key booster of SVM performance.

Juan Manuel González Rincón

ID 23031523

Introduction

The present research aims to give a clear understanding of the classification machine learning technique SVM, focusing on the importance of the kernel for data classification in a multidimensional space according to its nature and behavior, this will help us to get more mathematical intuition on the selection of this kernel and the effects on its use.

Fundamentals of classifier based on SVM

SVM is one part from a wide range of algorithms of supervised learning based on the development made by Vladimir Vapnik at AT&T Bell Laboratories in 1995, taking further its own proposed VC theory, nowadays it is a widely used classification technique for either linear or non-linear problems with the use of the powerful Kernel trick.

The main objective of the SVM is to separate the classes with a hyperplane. Imagine a hyperplane dividing the classes, the closest points to the hyperplane are called support vectors and their distance to the hyperplane is called margins, the way to separate the classes is by increasing the width of the margins by finding the correct hyperplane.

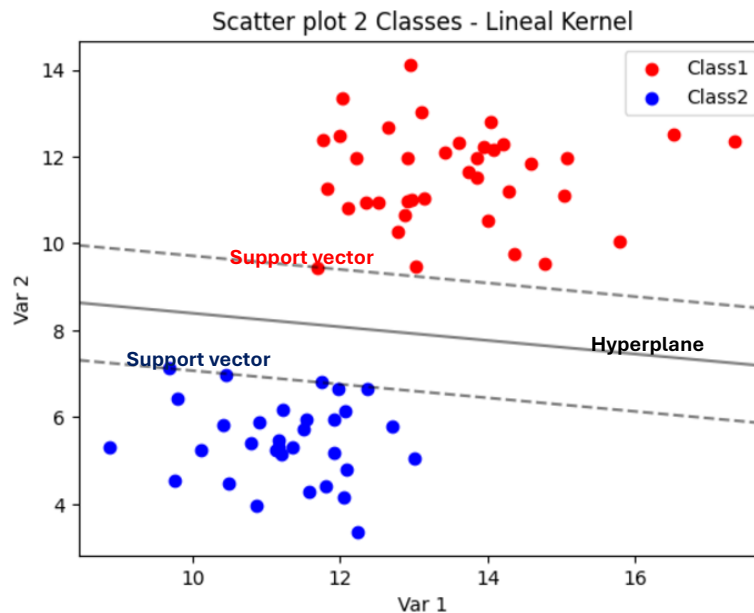


Chart1. Linear SVM.

The hyperplane is basically a function in a defined dimension, in the case of a plane (2D), the hyperplane is a line (1D), the general equation of this hyperplane is given by:

$$y = b + w^T \cdot X$$

Equation1. Hyperplane function.

With “**w**” as vectors of weights, “**b**” as the bias term and “**X**” as the variables, the classification rule is based on the location of the points with respect to the main hyperplane. With the purpose of define the rules we set two more hyperplanes, each one crossing each support vector on each side of the main hyperplane, each vector is going to be the threshold of the algorithm for making the classification, following by the functions:

$$\begin{cases} \text{For class 1 we set } w^T \cdot X + b \geq 1 \\ \text{For class } -1 \text{ we set } w^T \cdot X + b \leq -1 \end{cases}$$

Equation2. Function classification rule.

It is very important to contrast this formulation with the labeled data, for that reason, there was purposed another formula to measure how correct each point was classified by the hyperplane and at the same time summarize the previous two equations:

$$y_i(w^T \cdot X + b) \geq 1$$

Equation3. Constrain for setting support vectors.

Where “**y_i**” is the original label of the training data (1 or -1) and the result of the equation will be equal or greater that 1 in the cases in which the classification was correctly made by the algorithm and less that 1 when it is not clearly classified, this equation also gives us 0 for the support vectors.

Now, bringing back to the objective of increasing the margin between classes or between the lateral hyperplanes, it is needed to measure this margin, based on a perpendicular projection of the support vectors over a vector “**w**”, the width of the margin is given by a subtraction of the distances of each support vector.

$$d_- = \frac{x_- \cdot w}{||w||} \quad \text{and} \quad d_+ = \frac{x_+ \cdot w}{||w||}$$

Equation4. Function distances to support vector.

Basically, the margin is given by:

$$d = d_+ - d_-$$

Equation5. Function getting the width.

Considering the equation x we define the margin as:

$$d = \frac{2}{||w||} \quad \text{same as} \quad d = \frac{2}{w^T \cdot w}$$

Equation6. Function width.

Now we define our objective function to maximize, knowing that we can manage a problem of maximization as a minimization of its inverse, and considering the restriction previously defined for the equation x about the support vectors.

$$\text{Objective function and function to minimize } f(w) = \frac{1}{2} w^T \cdot w$$

$$\text{Restriction for support vectors } g(w, b) = y_i(w \cdot x_{+-} + b) - 1 = 0$$

Equation7/8. Objective function and constrain support vectors.

At this point Vapnik used the Lagrangiano for the optimization, which is basically create a function that has out initial Objective function with a weighted restriction, ending in the function:

$$L(w, b, \alpha_i) = \frac{1}{2} w^T w - \sum_{i=1}^l \alpha_i [y_i(w \cdot x_i + b - 1)] \text{ with } l \text{ as the number of support vectors}$$

Equation9. Final function to minimize with lagrancio.

After a derivation process with respect to each variable, Vapnik found one of the keys of this algorithm, the result equation depends on a vector multiplication of the variables, this insight will be helpful for the use of the kernel.

$$L(\alpha_i) = \sum_i \alpha_i - \frac{1}{2} \sum_i \sum_j \alpha_i \alpha_j y_i y_j (x_i \cdot x_j)$$

Equation10. Result of the objective function after derivation.

The model used in the chart1, was trained with the 70% of the data and then tested with the 30% remaining, using the linear hyperplane, first we need to define the kernel, as this example was clearly linearly separable, we train the model as:

```
# Then we create and train our model
model_linear=svm.SVC(kernel='linear')
model_linear.fit(x_train,y_train)

SVC

SVC(kernel='linear')

predictions=model_linear.predict(x_test)
```

Image1. SVM trained with Linear kernel

Getting the final classification clearly shown, with the rule of classification based on the hyperplane, the training data is represented with points (·) and predicted values as stars (*).

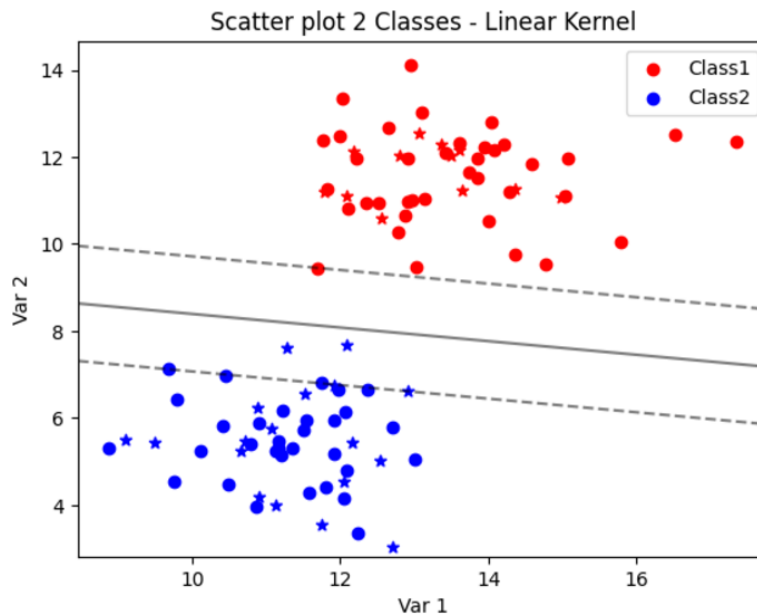


Chart2. SVM Linear kernel predictions.

The previous exercise resulted in a perfect classification with an accuracy measure of 100%, It is used the accuracy because is one of the best measures to weight the correct classification of both classes with same importance.

```
accuracy_lineal=accuracy_score(predictions,y_test)
print('Accuracy:',accuracy_lineal)
```

Accuracy: 1.0

Image2. Accuracy of predictions from SVM Linear kernel model.

Kernel trick

In the previous chart we saw two classes that were separated by a line in a 2D plane, that doesn't happen in all the problems, there are some cases where the data is not clearly separable in the original dimensionality, but we can transform the original data with some function $f(x)$ to add a new dimension where the data is now easily split, the most intuitive example is from a 1D to a 2D plane, given by $f(x) = x^2$.

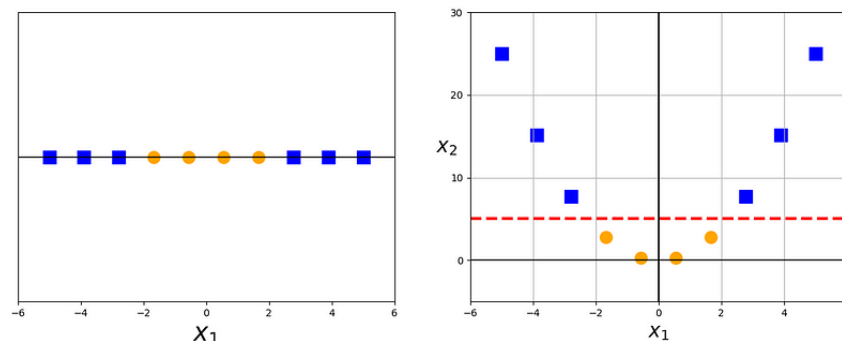


Chart3. Example of linear division of classes with vector and plane.

In this example, we have the original variables in a vector in where was difficult to classify the classes with a line, but once we apply the transformation $f(x)$ that put the variables in a plane, it was perfectly separable by a single line.

Last example was very helpful to understand how dimensionality transformation can help us to classify data, but it is not possible to simply create another function to add dimensions, that is because as we conclude before, the whole optimization problem is determined by a product of the vector of variables, in this case the idea to include the transformation to each element of the vector and then make multiplications and measure the distance in the new dimensionality will be very computational demanding, here is where the mathematicians brought the kernel functions, which are functions that implicitly calculate the scalar product in a space with different characteristics and higher dimensions without calculating the transformation element by element, the kernel also change the nature of the hyperplane depending on its type, given a transformation θ , the relation of the kernel with the dot product is given by:

$$K(x_i, x_j) = \theta(x_i) \cdot \theta(x_j)$$

Equation11. Kernel and dot product association.

Considering the kernel in the optimization function for problems that are not linearly separated the final equation will be:

$$L(\alpha_i) = \sum_i \alpha_i - \frac{1}{2} \sum_i \sum_j \alpha_i \alpha_j y_i y_j K(x_i, x_j)$$

Equation12. Result of the objective function with kernel.

As the previous exercise, let's analyze a dataset where the points are spread in a certain way, cases like more complex data with mixed classes or with an inner class and an outer class required more advanced techniques.

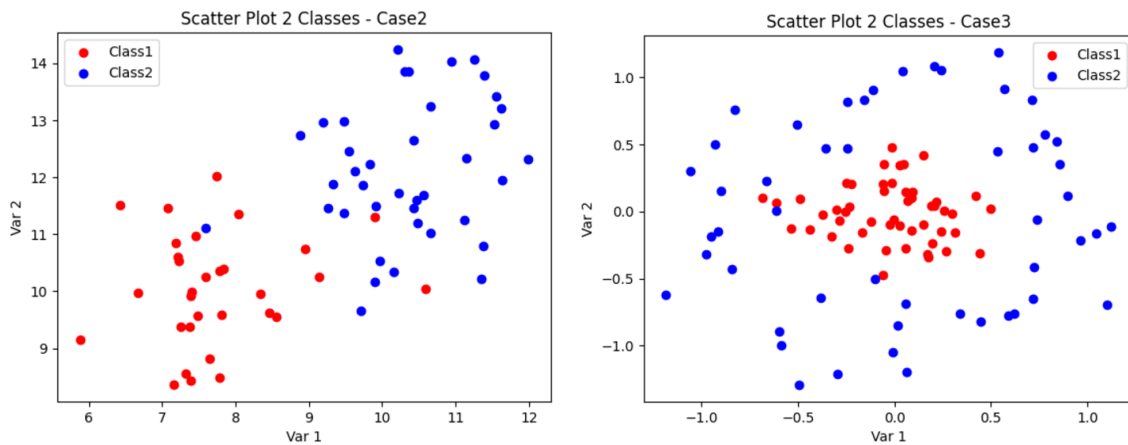


Chart4. Data Case2 with more mixed classes case and Data Case3 classes distributed with different centroids.

In this case, a simple linear hyperplane is not the most intuitive choice, that means that as in the transformation example, we will need to add a new dimension to get a clearer way to separate both classes, here is where we introduce another type of kernel, in both examples the performance of lineal kernel was not good at all Case2 accuracy = 0.7 and for Case3 accuracy=0.56.

Polynomial Kernel: Is used for non-linear relationships between the features, basically is an extension of the linear kernel but includes more features in the hyperplane function, includes the combination of the variables as xy , includes higher degree relationships as x^2 to be more able to separate more complex distribution of data all these components of the hyperplane come from the distributive rule over the kernel basic function.

$$K(x_i, x_j) = (x_i^T \cdot x_j + c)^d$$

Equation13. Polynomial kernel calculation of similarity.

Where the C is a free parameter trading off the influence of higher-order and lower-order polynomial grades and d is the maximum degree of the polynomial order use in the optimization, is important to know that the degree helps to catch more difficult trends but as increase is also more prone to overfit, at this point is important to know how the model is creating the hyperplane lines and check with the accuracy in the training and test sets.

The degree is specified in parameter d from the sklearn library as:

```
model_poly=svm.SVC(kernel='poly',degree=3,C=1)
model_poly.fit(x_train,y_train)
```

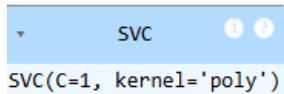


Image3. Setting SVM model with Polynomial degree 3 kernel.

With the same data of Case2, the model was trained with different degrees to see the effect in the accuracy on the test set.

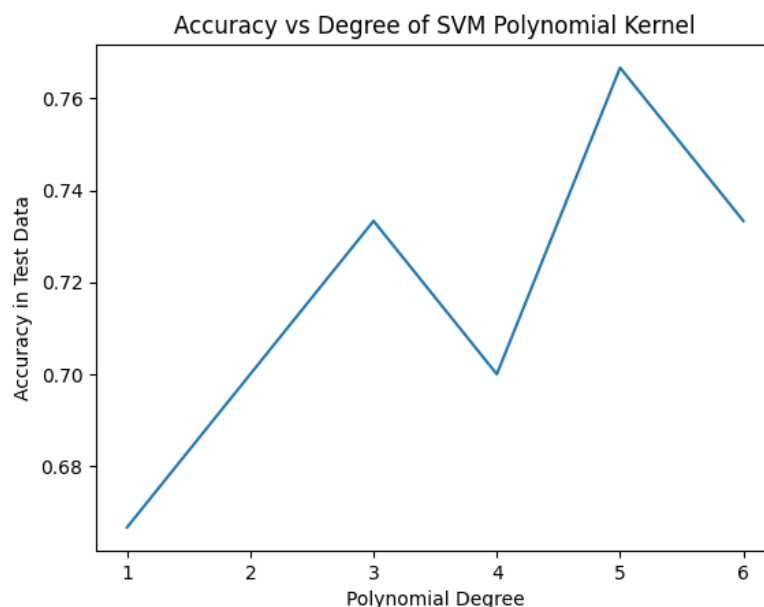


Chart5. Degree of the SVM kernel vs Accuracy

Basically, the degree 1 of the SVM polynomial kernel is the same lineal kernel while increasing the kernel let us capture more complex relationships, in this case the best way to select the model is checking the decision boundary, in this case the best one without losing accuracy was the degree 3.

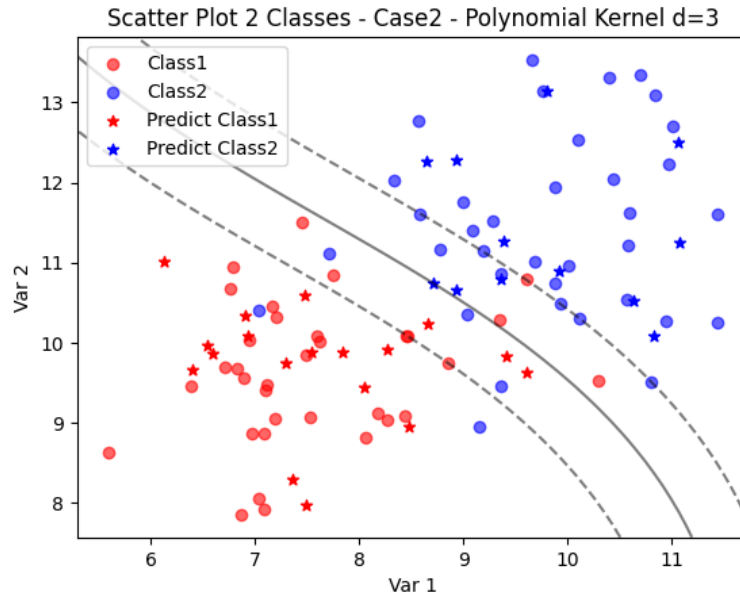


Chart5. SVM Polynomial Kernel degree 3.

The accuracy of this case was 0.73 but in the chart is clearly seen that the decision boundary changes compared to the lineal.

The Gaussian Kernel Radial Function (RBF) Kernel: Is specially used with more complex and non-linear relationships; the relationship depends on the distance between the points with the calculation of the Euclidian distance $\|x_i - x_j\|^2$, this distance is also weighted with a parameter σ which is called “reach” and gives the ponderation of the distance over the similarity of the points adjusting the trade-off between bias and variance.

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

Equation14. Gaussian kernel calculation of similarity.

In this case the similarity between the points measured with the kernel increases as the points are closer and decreases if distance increases, as we can see if we plot just the function $y = e^{-x^2}$ with y representing similarity and x representing weighted distance, we will have a better view of the relationship.

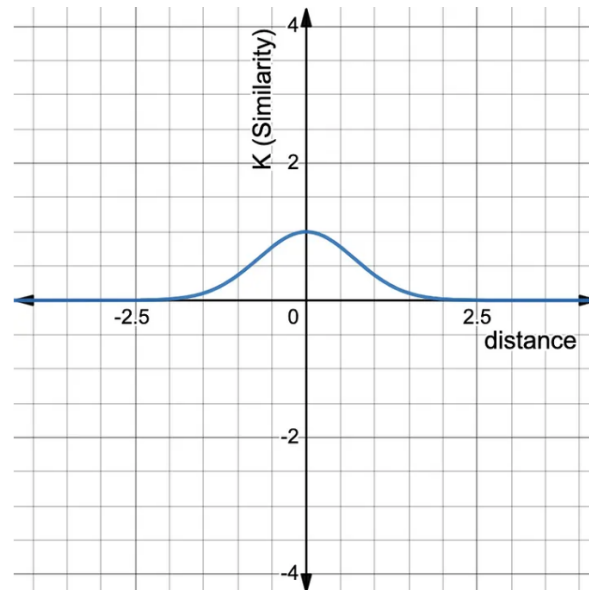


Chart6. Behavior of relationship between similarity and distance.

Using the “RBF” kernel to create a new SVM model for the previously shown data points of Case3 we have:

```
# Then we create and train our gaussian model
model_gaussian=svm.SVC(kernel='rbf')
model_gaussian.fit(x_train,y_train)
```

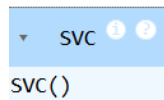


Image4. SVM trained with RBF kernel

After predicting the test values and plotting the hyperplane in a 2D chart that also contains the training data with labels, we can see the importance of using this new kernel, to compare it with the linear one.

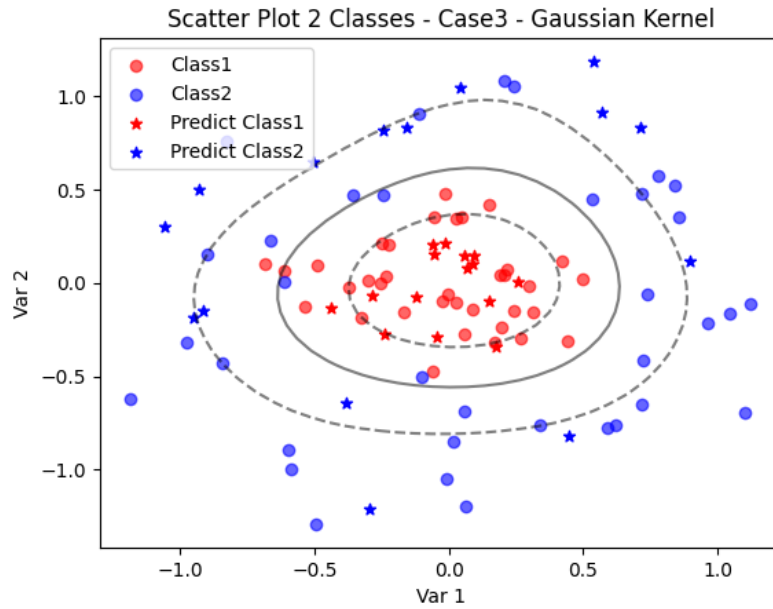


Chart7. Predictions of SVM model with Gaussian kernel.

In both cases the essential classification rule of the SVM is followed with the hyperplane as the key separator of data but the two hyperplanes are essentially different, the “RBF” is a parabolic plane, to see this it was extracted the third dimension calculated based on “RBF”.

```
z_test=model_gaussian.decision_function(x_test)
```

Image5. Calculating the z axis with the decision function.

When this dimension is put together with the original variables, it is visible how the transformation divides better the classes with Class1 over the Class2.

SVM 3D representation Gaussian Kernel

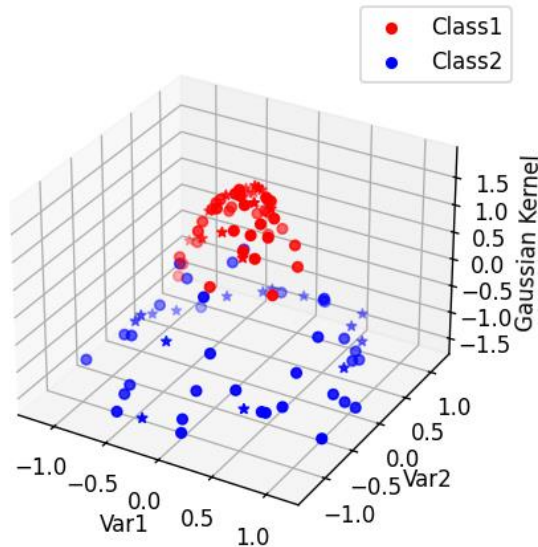


Chart8. Higher dimensionality from SVM with RBF transformation.

Bringing back the classification made by SVM using the Gaussian kernel, its accuracy was 0.93 compared to the 0.56 given by the SVM with Linear kernel.

Conclusion

Kernels are essential in Support Vector Machines (SVM) to solve non-linearly separable problems. Each kernel type has specific characteristics that make it suitable for different scenarios:

Linear Kernel: Works well for data that is separable in the original space but performs poorly with non-linear relationships. It is fast and easy to interpret but less flexible.

Polynomial Kernel: Models non-linear relationships by including higher-degree interactions between features. A low degree ($d=2$ or $d=3$) can capture complex patterns, but higher degrees increase the risk of overfitting. It is useful when data relationships are clearly polynomial.

RBF Kernel (Radial Basis Function): The most versatile kernel, transforming data into an infinite-dimensional space based on Gaussian similarities. Its γ parameter controls the range of influence of support points, making it ideal for problems with complex decision boundaries.

Github: https://github.com/Juan980207/SVM_Kernels.git

References

1. Towards Data Science. (n.d.). *Support Vector Machines (SVM)*. Retrieved from <https://towardsdatascience.com/support-vector-machines-svm-c9ef22815589>
2. Scikit-learn. (n.d.). *Exercise: Support Vector Machine Classifier on Iris Dataset*. Retrieved from https://scikit-learn.org/stable/auto_examples/exercises/plot_iris_exercise.html
3. Freie Universität Berlin. (n.d.). *Support Vector Machines: SOGA R*. Retrieved from <https://www.geo.fu-berlin.de/en/v/soga-r/Machine-Learning/SVM/index.html>
4. Towards Data Science. (n.d.). *The Kernel Trick*. Retrieved from <https://towardsdatascience.com/the-kernel-trick-c98cdbcaeb3f>
5. StatQuest. (2020, November 24). *Support Vector Machines Part 1 [YouTube video]*. Retrieved from <https://www.youtube.com/watch?v=1NxnPkZM9bc>
6. StatQuest. (2020, November 26). *Support Vector Machines Part 2 [YouTube video]*. Retrieved from https://www.youtube.com/watch?v=_PwhiWxHK8o
7. StatQuest. (2021, March 10). *Support Vector Machines Part 3 [YouTube video]*. Retrieved from https://www.youtube.com/watch?v=XyH8bdv_DS w
8. Towards Data Science. (n.d.). *Radial Basis Function (RBF) Kernel*. Retrieved from <https://towardsdatascience.com/radial-basis-function-rbf-kernel-the-go-to-kernel-acf0d22c798a>

9. AI Mind. (n.d.). *Unlocking the Power of RBF Kernel*. Retrieved from <https://pub.aimind.so/unlocking-the-power-of-rbf-kernel-advantages-local-boundaries-gammas-influence-2aebafce336a>
10. Medium. (n.d.). *SVM Kernels and Its Types*. Retrieved from <https://medium.com/@abhishekjainindore24/svm-kernels-and-its-type-dfc3d5f2dcd8>
11. Scikit-learn. (n.d.). *Tuning Parameters for RBF Kernel*. Retrieved from https://scikit-learn.org/stable/auto_examples/svm/plot_rbf_parameters.html
12. GeeksforGeeks. (n.d.). *Adjusting Degree Parameter for Polynomial Kernel in SVM*. Retrieved from <https://www.geeksforgeeks.org/how-to-adjust-the-degree-parameter-for-a-polynomial-kernel-in-svm/>