

**Modern Education Society's  
College of Engineering, Pune-01**

<b>NAME OF STUDENT:</b>	<b>CLASS:</b>
<b>SEMESTER/YEAR:</b>	<b>ROLL NO:</b>
<b>DATE OF PERFORMANCE:</b>	<b>DATE OF SUBMISSION:</b>
<b>EXAMINED BY:</b>	<b>EXPERIMENT NO:</b>

**TITLE:** ASSIGNMENT ON EMAIL SPAM DETECTION.

**Problem Statement:**

Classify the email using the binary classification method. Email Spam detection has two states: a) Normal State – Not Spam, b) Abnormal State – Spam. Use K-Nearest Neighbors and Support Vector Machine for classification. Analyze their performance. Dataset link: The emails.csv dataset on the Kaggle <https://www.kaggle.com/datasets/balaka18/email-spam-classification-dataset-csv>.

**Objectives:**

1. Understand the Dataset & cleanup (if required).
2. Build K-Nearest Neighbour and Support vector machine models to predict the spam email.
3. Also evaluate the models & compare their respective scores like  $R^2$ , RMSE, etc.

**Pre-requisites:**

1. Knowledge of python programming.
2. Knowledge of Data Pre-processing.
3. Knowledge of K-nearest neighbour model.
4. Knowledge of Support vector machine model.

**Description:**

In this project we are classifying mails typed in by the user as either 'Spam' or 'Not Spam'. Our original dataset was a folder of 5172 text files containing the emails.

Now let us understand why we have separated the words from the mails. This is because; this is a text-classification problem. When a spam classifier looks at a mail, it searches for potential words that it has seen in the previous spam emails. If it finds a majority of those words, then it labels it as 'Spam'. Why did I say majority ? -->

CASE 1 : suppose let's take a word 'Greetings'. Say, it is present in both 'Spam' and 'Not Spam' mails.

CASE 2 : Let's consider a word 'lottery'. Say, it is present in only 'Spam' mails.

CASE 3 : Let's consider a word 'cheap'. Say, it is present only in spam.

If now we get a test email, and it contains all the three words mentioned above, there's high probability that it is a 'Spam' mail.

**About the Dataset**

The csv file contains 5172 rows, each row for each email. There are 3002 columns. The first column indicates Email name. The name has been set with numbers and not recipients' name

to protect privacy. The last column has the labels for prediction : 1 for spam, 0 for not spam. The remaining 3000 columns are the 3000 most common words in all the emails, after excluding the non-alphabetical characters/words. For each row, the count of each word(column) in that email(row) is stored in the respective cells. Thus, information regarding all 5172 emails are stored in a compact dataframe rather than as separate text files.

## K-Nearest Neighbor(KNN) Algorithm for Machine Learning

The K-nearest Neighbors (KNN) algorithm is a type of supervised machine learning algorithm used for classification, regression as well as outlier detection. It is extremely easy to implement in its most basic form but can perform fairly complex tasks. Each time there is a new point added to the data, KNN uses just one part of the data for deciding the value (regression) or class (classification) of that added point. Since it doesn't have to look at all the points again, this makes it a lazy learning algorithm. KNN is a non-parametric learning algorithm, which means that it doesn't assume anything about the underlying data. This is an extremely useful feature since most of the real-world data doesn't really follow any theoretical assumption e.g. linear separability, uniform distribution, etc.

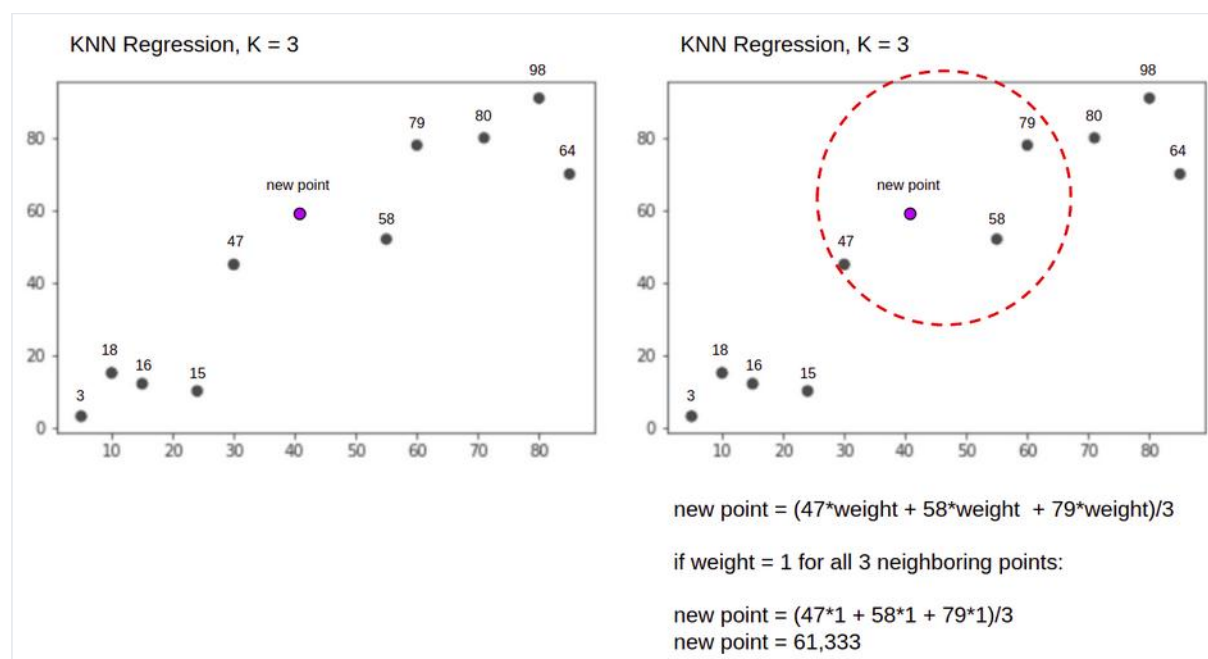
### Working of KNN:

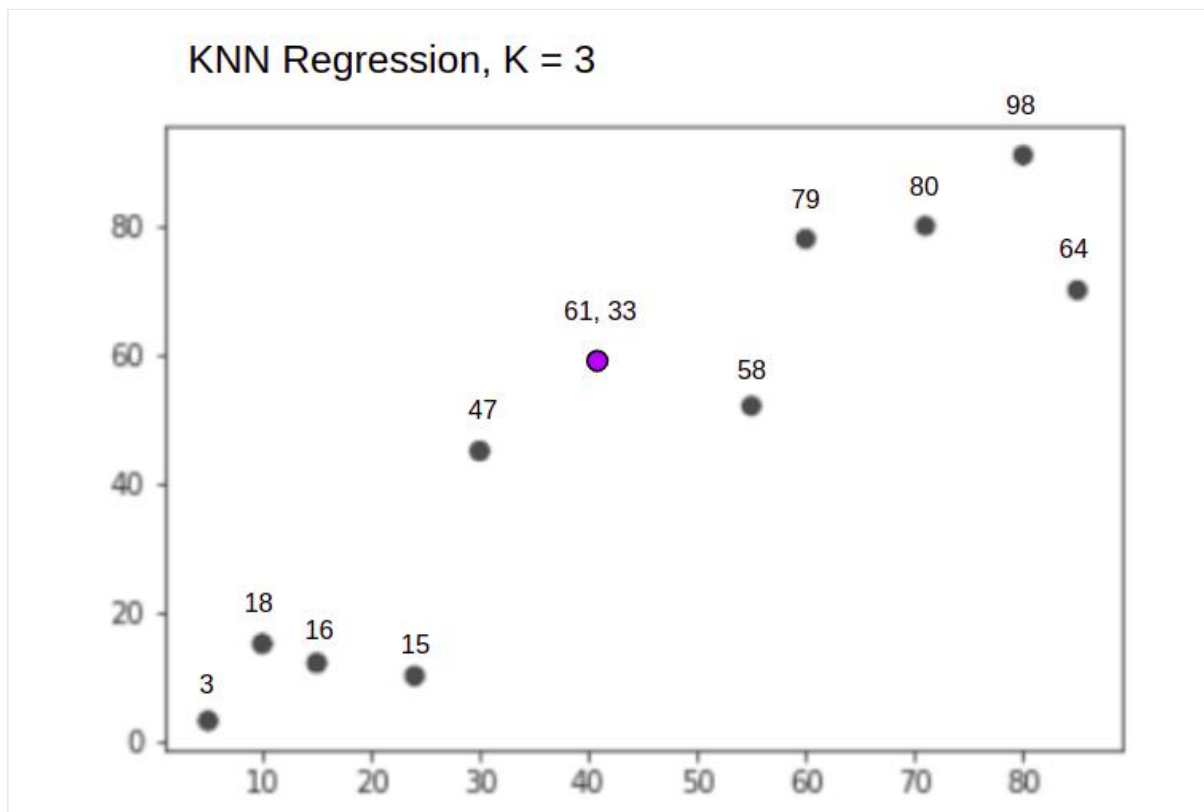
The algorithm first calculates the distance of a new data point to all other training data points. After calculating the distance, KNN selects a number of nearest data points - 2, 3, 10, or really, any integer. This number of points (2, 3, 10, etc.) is the K in K-Nearest Neighbors!

In the final step, if it is a regression task, KNN will calculate the average weighted sum of the K-nearest points for the prediction. If it is a classification task, the new data point will be assigned to the class to which the majority of the selected K-nearest points belong.

Consider a dataset with two variables and a K of 3. When performing regression, the task is to find the value of a new data point, based on the average weighted sum of the 3 nearest points.

KNN with  $K = 3$ , when used for regression:

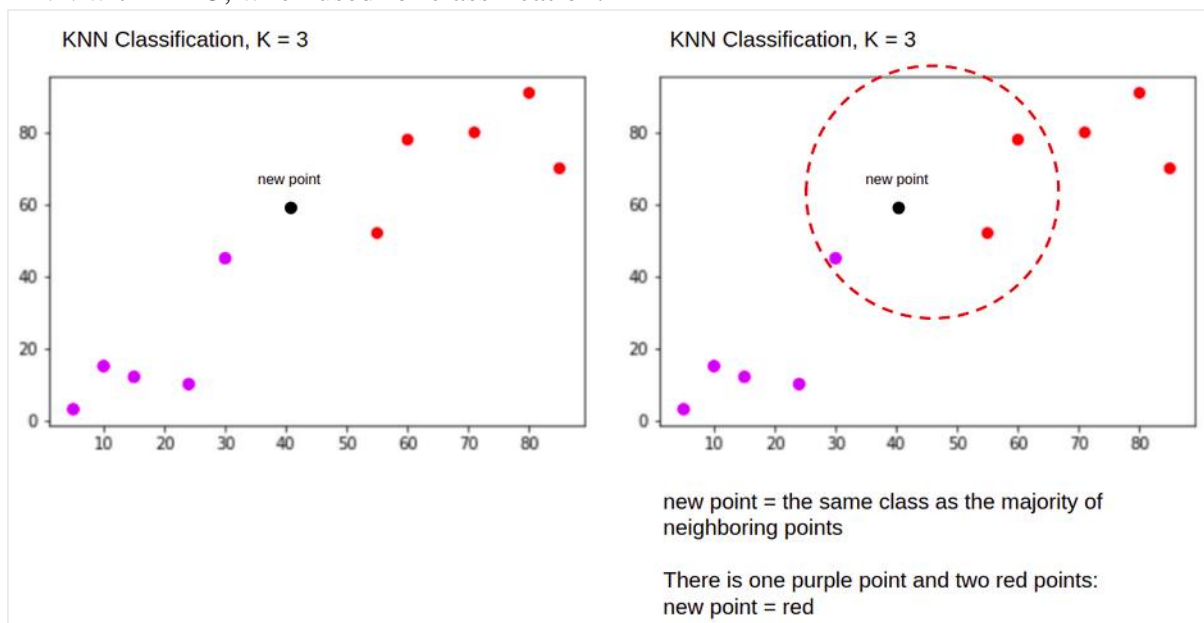


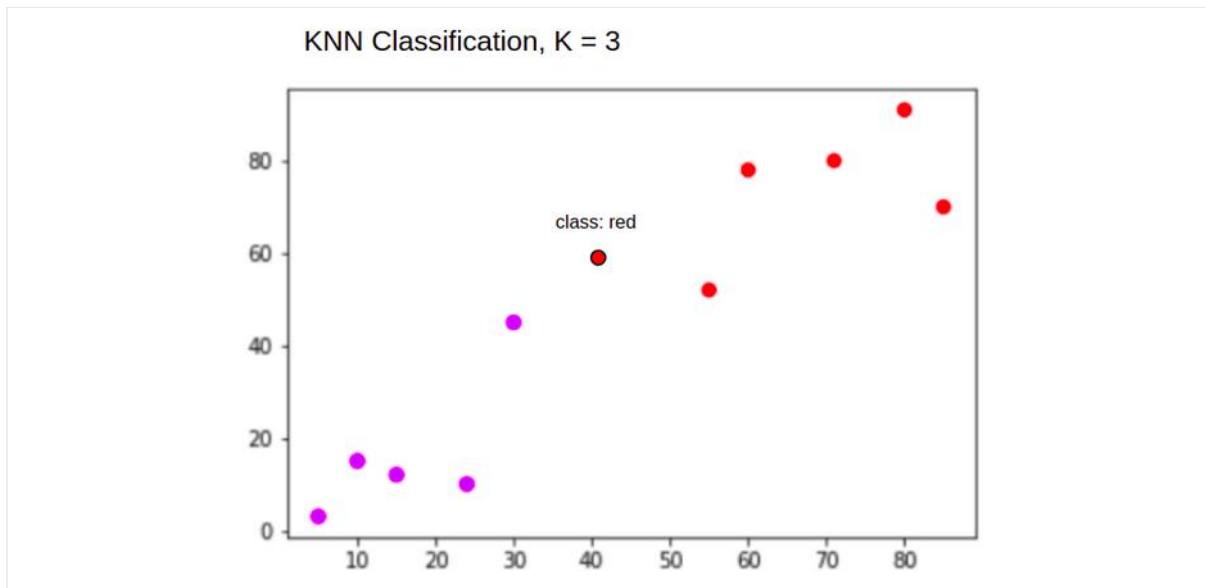


The KNN algorithm will start by calculating the distance of the new point from all the points. It then finds the 3 points with the least distance to the new point. This is shown in the second figure above, in which the three nearest points, 47, 58, and 79 have been encircled. After that, it calculates the weighted sum of 47, 58 and 79 - in this case the weights are equal to 1 - we are considering all points as equals, but we could also assign different weights based on distance. After calculating the weighted sum, the new point value is 61.33.

And when performing a classification, the KNN task to classify a new data point, into the "Purple" or "Red" class.

KNN with K = 3, when used for classification:





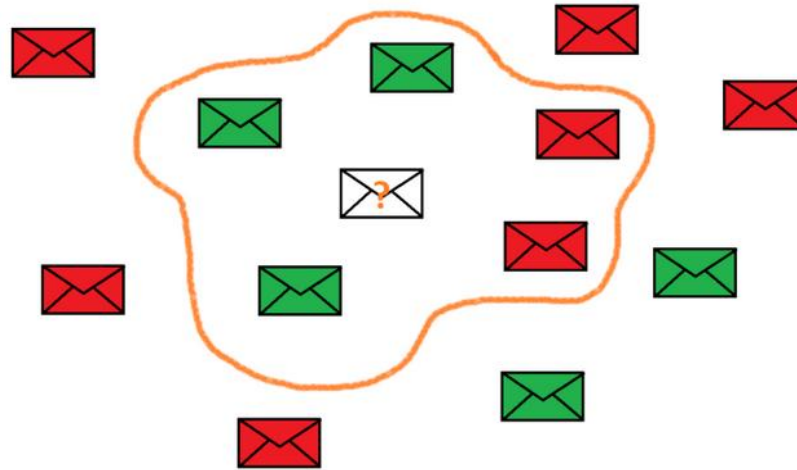
The KNN algorithm will start in the same way as before, by calculating the distance of the new point from all the points, finding the 3 nearest points with the least distance to the new point, and then, instead of calculating a number, it assigns the new point to the class to which majority of the three nearest points belong, the red class. Therefore the new data point will be classified as "Red".

### **How to select the value of K in the K-NN Algorithm?**

Defining k can be a balancing act as different values can lead to overfitting or underfitting. Lower values of k can have high variance, but low bias, and larger values of k may lead to high bias and lower variance. The choice of k will largely depend on the input data as data with more outliers or noise will likely perform better with higher values of k. Overall, it is recommended to have an odd number for k to avoid ties in classification, and cross-validation tactics can help you choose the optimal k for your dataset.

### **Steps to implement the K-NN algorithm:**

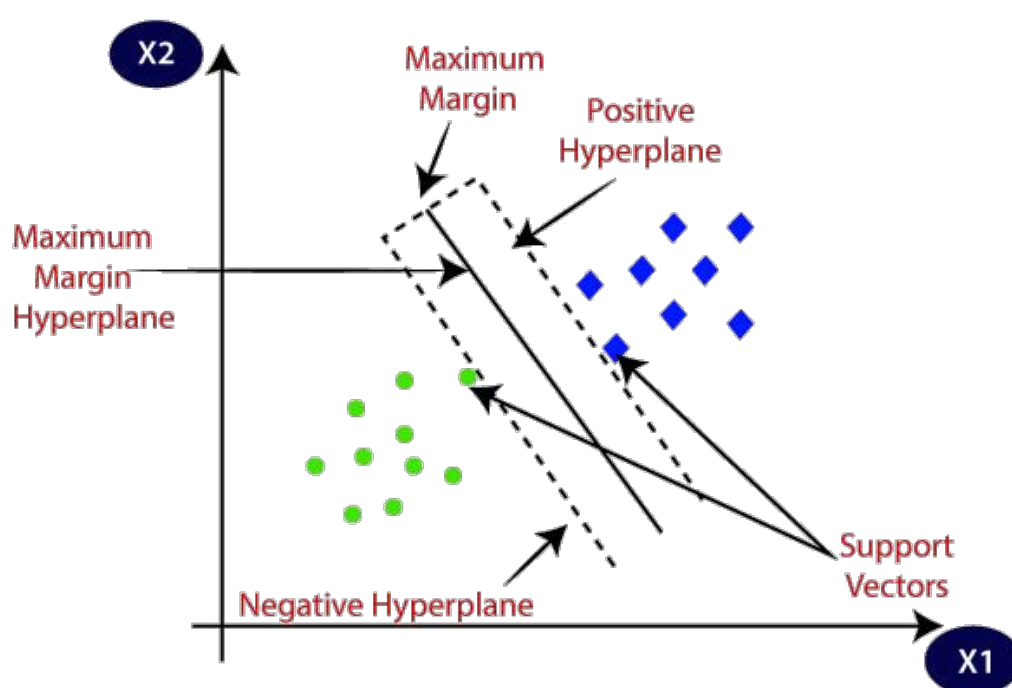
- Data Pre-processing step
- Fitting the K-NN algorithm to the Training set
- Predicting the test result
- Test accuracy of the result(Creation of Confusion matrix)
- Visualizing the test set result.



In the example shown above,  $K = 5$ ; we are comparing the email we want to classify to the nearest 5 neighbours. In this case, 3 out of 5 emails are classified as ham (non-spam), and 2 are classified as spam. Therefore, the unknown email will be given the class of the majority: ham.

### Support Vector Machine in Machine Learning:

Support Vector Machine or SVM is one of the most popular Supervised Learning algorithms, which is used for Classification as well as Regression problems. However, primarily, it is used for Classification problems in Machine Learning. The goal of the SVM algorithm is to create the best line or decision boundary that can segregate n-dimensional space into classes so that we can easily put the new data point in the correct category in the future. This best decision boundary is called a hyperplane. SVM chooses the extreme points/vectors that help in creating the hyperplane. These extreme cases are called as support vectors, and hence algorithm is termed as Support Vector Machine. Consider the below diagram in which there are two different categories that are classified using a decision boundary or hyperplane:



### SVM can be of two types:

- **Linear SVM:** Linear SVM is used for linearly separable data, which means if a dataset can be classified into two classes by using a single straight line, then such data is termed as linearly separable data, and classifier is used called as Linear SVM classifier.
- **Non-linear SVM:** Non-Linear SVM is used for non-linearly separated data, which means if a dataset cannot be classified by using a straight line, then such data is termed as non-linear data and classifier used is called as Non-linear SVM classifier.

### Hyperplane and Support Vectors in the SVM algorithm:

**Hyperplane:** There can be multiple lines/decision boundaries to segregate the classes in n-dimensional space, but we need to find out the best decision boundary that helps to classify the data points. This best boundary is known as the hyperplane of SVM.

The dimensions of the hyperplane depend on the features present in the dataset, which means if there are 2 features (as shown in image), then hyperplane will be a straight line. And if there are 3 features, then hyperplane will be a 2-dimension plane. We always create a hyperplane that has a maximum margin, which means the maximum distance between the data points.

#### Support Vectors:

The data points or vectors that are the closest to the hyperplane and which affect the position of the hyperplane are termed as Support Vector. Since these vectors support the hyperplane, hence called a Support vector.

Support vectors are data points that are closer to the hyperplane and influence the position and orientation of the hyperplane. Using these support vectors, we maximize the margin of the classifier. Deleting the support vectors will change the position of the hyperplane. These are the points that help us build our SVM. In SVM, we take the output of the linear function and if that output is greater than 1, we identify it with one class and if the output is -1, we identify it with another class. Since the threshold values are changed to 1 and -1 in SVM, we obtain this reinforcement range of values  $[-1, 1]$  which acts as margin.

### Cost Function and Gradient Updates

In the SVM algorithm, we are looking to maximize the margin between the data points and the hyperplane. The loss function that helps maximize the margin is hinge loss.

$$c(x, y, f(x)) = \begin{cases} 0, & \text{if } y * f(x) \geq 1 \\ 1 - y * f(x), & \text{else} \end{cases}$$

$$c(x, y, f(x)) = (1 - y * f(x))_+$$

Hinge loss function (function on left can be represented as a function on the right)

The cost is 0 if the predicted value and the actual value are of the same sign. If they are not, we then calculate the loss value. We also add a regularization parameter the cost function. The

objective of the regularization parameter is to balance the margin maximization and loss. After adding the regularization parameter, the cost functions looks as below.

$$\min_w \lambda \|w\|^2 + \sum_{i=1}^n (1 - y_i \langle x_i, w \rangle)_+$$

Loss function for SVM

Now that we have the loss function, we take partial derivatives with respect to the weights to find the gradients. Using the gradients, we can update our weights.

$$\frac{\partial}{\partial w_k} \lambda \|w\|^2 = 2\lambda w_k$$

$$\frac{\partial}{\partial w_k} (1 - y_i \langle x_i, w \rangle)_+ = \begin{cases} 0, & \text{if } y_i \langle x_i, w \rangle \geq 1 \\ -y_i x_{ik}, & \text{else} \end{cases}$$

Gradients

When there is no misclassification, i.e our model correctly predicts the class of our data point, we only have to update the gradient from the regularization parameter.

$$w = w - \alpha \cdot (2\lambda w)$$

Gradient Update — No misclassification

When there is a misclassification, i.e our model make a mistake on the prediction of the class of our data point, we include the loss along with the regularization parameter to perform gradient update.

$$w = w + \alpha \cdot (y_i \cdot x_i - 2\lambda w)$$

Gradient Update — Misclassification

Since SVMs can use any number of kernels, it's important that you know about a few of them.

## Kernel functions

### Linear

These are commonly recommended for text classification because most of these types of classification problems are linearly separable.

The linear kernel works really well when there are a lot of features, and text classification problems have a lot of features. Linear kernel functions are faster than most of the others and you have fewer parameters to optimize.

Here's the function that defines the linear kernel:

$$f(X) = w^T \cdot X + b$$

In this equation,  $\mathbf{w}$  is the weight vector that you want to minimize,  $\mathbf{X}$  is the data that you're trying to classify, and  $\mathbf{b}$  is the linear coefficient estimated from the training data. This equation defines the decision boundary that the SVM returns.

### Polynomial

The polynomial kernel isn't used in practice very often because it isn't as computationally efficient as other kernels and its predictions aren't as accurate.

Here's the function for a polynomial kernel:

$$f(\mathbf{X1}, \mathbf{X2}) = (a + \mathbf{X1}^T * \mathbf{X2})^b$$

This is one of the more simple polynomial kernel equations you can use.  $f(\mathbf{X1}, \mathbf{X2})$  represents the polynomial decision boundary that will separate your data.  $\mathbf{X1}$  and  $\mathbf{X2}$  represent your data.

### Gaussian Radial Basis Function (RBF)

One of the most powerful and commonly used kernels in SVMs. Usually the choice for non-linear data.

Here's the equation for an RBF kernel:

$$f(\mathbf{X1}, \mathbf{X2}) = \exp(-\gamma * \|\mathbf{X1} - \mathbf{X2}\|^2)$$

In this equation,  $\gamma$  specifies how much a single training point has on the other data points around it.  $\|\mathbf{X1} - \mathbf{X2}\|$  is the dot product between your features.

ADVERTISEMENT

### Sigmoid

More useful in neural networks than in support vector machines, but there are occasional specific use cases.

Here's the function for a sigmoid kernel:

$$f(\mathbf{X}, \mathbf{y}) = \tanh(\alpha * \mathbf{X}^T * \mathbf{y} + \mathbf{C})$$

In this function,  $\alpha$  is a weight vector and  $\mathbf{C}$  is an offset value to account for some misclassification of data that can happen.

### Steps to implement the SVM algorithm:

- Import the dataset
- Explore the data to figure out what they look like
- Pre-process the data
- Split the data into attributes and labels
- Divide the data into training and testing sets
- Train the SVM algorithm
- Make some predictions
- Evaluate the results of the algorithm

### Questions:

1. Why should we not use the KNN algorithm for large datasets?
2. Is Feature Scaling required for the KNN Algorithm? Explain with proper justification.
3. What do you mean by Hinge loss?
4. What's the "kernel trick" and how is it useful?