Kernel Methods

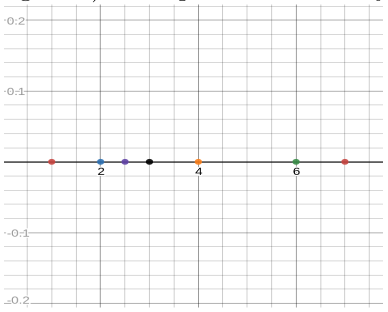
Rohit Singhatwadia

May 2018

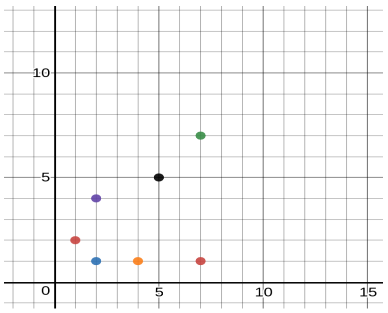
1 Introduction

In Machine Learning most of the theory and algorithms developed are good in handling linearly separable data. However most of the data analysis problem require methods capable of handling not linearly separable data.
To handle not linearly separable data we map the features of the data to a new feature space (high-dimensional). In this new feature space our data should be linearly separable. Now we can apply our traditional machine learning algorithm to it.

In the below figure consider that points having black,green and red colour are in class 1 and other points are in class 2. Now in figure 1 these points are not linearly separable, but after going to a higher dimensional space (i.e. in figure 2) these points are linearly separable.



Original points in 1D



Transformed points in 2D
We will consider an embedding map

$$\phi : x \epsilon R^n \mapsto \phi(x) \epsilon F \subset R^N$$

The choice of the map ϕ aims to convert the nonlinear relations into linear ones. After mapping it to the higher dimensions what we really need is the dot product between the features in the higher dimension space.
However if the dimension of the new feature space i.e. N is large than computing the dot product can be a expensive operation both in terms of time and space(to store the higher dimension features).
The inner products can, however, sometimes be computed more efficiently as a direct function of the input features, without going to the higher dimension. This direct function is known as kernel function.

2 Kernels

A kernel is a function which measures similarity between two objects in a way such that there exist a dot product corresponding to this similarity in some higher dimensional space. It simply takes two inputs and show how similar they are. The inputs can be anything from two integers, two real valued vectors, two strings etc. The arguably simplest example is the linear kernel, also called dot-product. Given two vectors, the similarity is the length of the projection of one vector on another. Another interesting kernel examples is Gaussian kernel. Given two vectors, the similarity will diminish with the radius of σ. The success of learning with kernels depends very strongly on the choice of kernel. It is very often problem specific. Now Let's define a kernel properly

A kernel is a function that for all x, z ϵ X satisfies

$$k(x,z) = \langle \phi(x), \phi(z) \rangle$$

where,

$$\phi : x \epsilon R^n \mapsto \phi(x) \epsilon F \subset R^N$$

By using kernels we have a shortcut. By that we mean that now there is no need to compute the features in higher dimensional space and no need to worry about the mapping ϕ. We can now directly compute the dot product in the higher dimension by our original features. This is known as the kernel trick. To explain this we will use a example. Suppose we have 2 vectors $x = (1,2,3) y = (4,5,6)$. We wish to compute there dot product in some higher dimensional space. Let the features in that space be $f(x) = (x1x1, x1x2, x1x3, x2x1, x2x2, x2x3, x3x1, x3x2, x3x3)$. For our vectors $\phi(x) = (1,2,3,2,4,6,3,6,9)$ and $\phi(y) = (16,20,24,20,25,30,24,30,36)$. Now finally there dot product is $\langle \phi(x), \phi(y) \rangle = 16+40+72+40+100+180+72+180+324 =$

1024. Now we use kernels to compute these directly $k(x,y) = (\langle x,y \rangle)^2 = (4+10+18)^2 = 1024$.

Note : The algorithm we will use after getting the dot product is independent of the choice of kernels.This signify modularity.

3 Hilbert Space

A Hilbert Space F is a abstract inner product space which follows two additional properties completeness and separable.
Completeness states that every cauchy sequence(a sequence whose elements become closer to each other as sequence progress) of elements of F converges to a element h ϵ F.
Separability states that for any $\epsilon > 0$ there is a finite set of elements $h_1 \dots h_N$ of F such that for all h ϵ F

$$min||h_i - h|| < \epsilon$$

Intuitively, by completeness we mean that there are no missing points from it. For example the set of rational number is incomplete as $(2)^{1/2}$ is missing from it. And a cauchy sequence of rational numbers can converge to it.
Separability implies that Hilbert spaces have countable orthonormal bases. Any space which is finite or countably infinite is separable as the whole space is a countable dense subset of itself. Example of an uncountable separable space is the real line, in which the rational numbers form a countable dense subset.A simple example of a space which is not separable is a uncountable discrete space.
The reason for the importance of the properties of completeness and separability is that together they ensure that Hilbert spaces are either isomorphic to R^n for some finite n or to the space L2.
Each element in this space is a function. This function is a linear combination of our input features. This is similar to our weight vector as it is a linear combination of the feature vectors of the training points. Thus finding the weight vector is equivalent to finding a equivalent element of this space.
A reproducing kernel Hilbert space (RKHS) is a Hilbert space of functions in which point evaluation is a continuous linear function .Thus this guarantee that the function element that we are looking for is linear.
Note : In an inner product space, if k is a kernel and $x_1, x_2 \epsilon X$, then

$$k(x1, x2) \leq k(x1, x1) * k(x2, x2)$$

this is called Cauchy–Schwarz inequality.

4 Gram Matrix

Suppose we have l input vectors $x_1 \dots x_l$ and we are using a kernel k to evaluate the dot product in a feature space with feature map ψ than we define gram matrix as

$$G_{ij} = \langle \phi(x_i), \phi(x_j) \rangle = k(x_i, x_j)$$

This matrix is also called kernel matrix.This matrix is symmetric. And it contain all the information needed to find the pairwise distance in the data set. But it looses the information of the original data set w.r.t origin. We loose the information like the angle at which the feature is located w.r.t origin .This signify that gram matrix are rotation invariant.

Positive semi-definite Kernels Let X be a nonempty set. A function k : $X * X \mapsto R$ which give rise to a positive semi-definite gram matrix for all values of $x \epsilon X$ is called positive semi-definite kernel.

5 Construction of the reproducing kernel Hilbert space

Here we have a function k

$$k : XxX \mapsto R$$

We assume that k satisfies the positive semi-definite property and proceed to construct a feature mapping ϕ into a Hilbert space for which is the kernel. First we will define the mapping ϕ

$$\phi : x \epsilon X \mapsto \phi(x) = k(x, .) \epsilon F$$

Now there is something unusual about this construction. Here elements of feature space will be functions. Now we use F to denote the function space. Now to define F as a vector space

$$f, g \epsilon F \Rightarrow (f + g)(x) = f(x) + g(x)$$

We now introduce a inner product on F as follows. Let $f, g \epsilon F$ be given by

$$f(x) = \sum_{i=1}^{l} \alpha_i k(x_i, x)$$

and

$$g(x) = \sum_{i=1}^{n} \beta_i k(z_i, x)$$

than their dot product is

$$\langle f, g \rangle = \sum_{i=1}^{l} \sum_{j=1}^{n} \alpha_i \beta_j k(x_i, z_j) = \sum_{i=1}^{l} \alpha_i g(x_i) = \sum_{j=1}^{n} \beta_j f(z_j)$$

In the previous expression we get the second and third equalities by substituting the value of g and f respectively.

This is real valued, bilinear and symmetric and hence satisfies the properties of a inner product, provide

$$\langle f, f\rangle \geq 0$$

But this follows from our assumption that the kernel is positive semi-definite,

$$\langle f, f\rangle = \sum_{i=1}^{l}\sum_{j=1}^{l}\alpha_i\alpha_j k(x_i, x_j) = \alpha' K\alpha \geq 0$$

where α_i and α_j are entries of vector α and K is the kernel matrix.
There is another property that follows if we take g=k(x,.)

$$\langle f, k(x,.)\rangle = \sum_{i=1}^{l}\alpha_i k(x_i, x) = f(x)$$

This fact is known as reproducing property of the kernel.Now we have to show the property of completeness and separability. Separability will follow if the input space is countable or the kernel is continuous. For completeness we consider a fixed input x and a Cauchy sequence $(f_n)_{n=1}^{\infty}$. $(f_n)_{n=1}^{\infty}$.

$$(f_n(x) - f_m(x))^2 = (\langle f_n - f_m, k(x,.)\rangle)^2 \leq ||f_n - f_m||^2 k(x, x)$$

by the Cauchy–Schwarz inequality. Thus $f_n(x)$ is a bounded sequence of real numbers and has a limit. This satisfies the completeness condition.
By using the reproducing property we can also say that

$$\langle f, \phi(x)\rangle = \langle f, k(x,.)\rangle = f(x)$$

Thus the function f can indeed be represented as the linear function defined by an inner product. This proves that the function f is linear. And since this corresponds to weights we can say that the features in the high dimensional space is linearly separable.

Given a function that satisfies the finitely positive semi-definite property we will refer to the corresponding space F_k as its Reproducing Kernel Hilbert Space (RKHS).

Note: In the early years of kernel machine learning research, it was not the notion of positive definite kernels that was being used. Instead, researchers considered kernels satisfying the conditions of Mercer's theorem.

Note: Let X be the set of all countable sequences of real numbers x $=x_1 \dots x_n \dots$ such that the sum

$$\sum_{i=1}^{\infty} x_i^2 < \infty$$

with the inner product between two sequences x and y defined by

$$\langle x, y\rangle = \sum_{i=1}^{\infty} x_i y_i$$

This space is known as L2.

6 Mercer's kernel

Mercer's theorem is also used to construct a feature space for a valid kernel just like the last section. The major difference is it defines the feature space in terms feature vector instead of function space.

Suppose K is a continuous symmetric function $K : R^m * R^m \mapsto R$, than there exist a inner product space H and a mapping $\phi : R^m \mapsto H$ so that

$$K(x_1, x_2) = \langle \phi(x_1), \phi(x_2) \rangle$$

if for all square-integrable function g

$$\int\int K(x_1, x_2)g(x_1)g(x_2)dx_1dx_2 \geq 0$$

This can be illustrated with the help of a example Let our kernel be a positive constant function $K(x_1, x_2) = c$. Now we want that

$$\int\int cg(x_1)g(x_2)dx_1dx_2 \geq 0$$

$$c\int\int g(x_1)g(x_2)dx_1dx_2 \geq 0$$

They both are the same integerals. So we can write it as

$$c(\int g(x_1)dx_1)^2 \geq 0$$

So, now we can say that this kernel satisfies mercer theorem.

7 Properties of kernels

1) **The set of positive definite kernels is a closed convex cone**
(a) If $\alpha_1 and \alpha_2 are \geq 0$ and k_1, k_2 are positive semi-definite kernels than $\alpha_1 k_1 + \alpha_2 k_2$ is positive semi-definite.
(b)If k(x,x') = $\lim_{n\mapsto\infty} k_n(x, x')$ exist for all x,x' than k is positive semi-definite.

2) **The pointwise product k_1k_2 is positive semi-definite.**

3) **The tensor product $k_1 \otimes k_2$ and direct sum $k_1 \oplus k_2$ are positive semi-definite.**

4) **Closure Properties**

(a) $k(x, y) = k_1(x, y) + k_2(x, y)$

(b) $k(x,y) = ak_1(x,y)$

(c) $k(x,y) = k_1(x,y)k_2(x,y)$

(d) $k(x,y) = f(x)f(y)$

(e)$k(x,y) = k_1(\phi(x),\phi(y))$

8 Kernel Construction

(a) $k(x,y) = p(k_1(x,y))$ where p(x) is a polynomial
(b) $k(x,y) = exp(k_1(x,y))$
(c) $k(x,y) = exp(-||x-y||^2/(2\sigma^2))$
(d) Kernels generated using closer properties

9 Examples of Kernels

9.1 Polynomial kernels

The derived polynomial kernel for a kernel k_1 is defined as

$$k(x,y) = p(k_1(x,y))$$

Frequently, it also refers to the inhomogeneous polynomial

$$k(x,y) = (\langle x,y\rangle + R)^d$$

We can expand it using Binomial Theorem. Features for each component in the sum together form the features of the whole kernel.
The polynomial kernel looks not only at the given features of input samples to determine their similarity, but also combinations of these. This kernel is not popular in svm as there RBF kernel is better. It's however used a lot in natural language processing. The most common degree is 2 since larger degree tend to overfit.
The problem with polynomial kernel is that it may suffer from numerical instability when $x^Ty + c < 1$ than $K(x,y) = (x^Ty + c)^d$ tends to zero as d increases where as $x^Ty + c > 1$ tend to infinity.
In NLP we need combination of features to determine accuracy. While applying polynomial kernel we can improve the speed by using inverted index. This inverted index table gives us for each feature the number of support vectors which contain that feature.

9.2 All-subsets kernel

It defines different combination of features. Consider a space with a feature ϕ_A for each subset A of the input features, including the empty subset. We can represent them as

$$\phi_i(x) = x_1^{i_1} \dots x_n^{i_n}$$

with the restriction that i=$(i_{1n})\epsilon 0, 1^n$. The feature ϕ_A is given by multiplying together the input features for all the elements of the subset. Now

$$\phi : x \mapsto \phi_A(x)$$

Now the corresponding kernel is

$$k(x, y) = \langle\phi(x), \phi(y)\rangle = \sum_A \prod_{i \epsilon A} x_i y_i = \prod_{i=1}^{n}(1 + x_i y_i)$$

9.3 Radial basis function kernel

For σ¿ 0, the Gaussian kernel is defined as

$$k(x, y) = exp(-||x - y||^2/(2\sigma^2))$$

We can also write it as

$$k(x, y) = exp(-\gamma||x - y||^2)$$

Now this kernel signifies a infinite dimensional feature space i.e it works as if we have transformed our features in a infinite dimensional space. Just perform the expansion of the function and this function can have up to infinite length. Although the higher order terms falls very quickly. This specify a similarity measure as when x is very close to y the kernel will attain its maximum value and when x is far away the kernel value will tend towards zero. The shape however is smooth.
A small gamma means a Gaussian with a large variance so the influence of x_j is more, i.e. if x_j is a support vector, a small gamma implies the class of this support vector will have influence on deciding the class of the vector x_i even if the distance between them is large. If gamma is large, then variance is small implying the support vector does not have wide-spread influence. Technically speaking, large σ leads to hig bias and low variance models, and vice-versa. Also the scaling by γ occurs the same amount in all directions.
RBF kernel works well in practice and it is relatively easy to tune. It is a stationary kernel, which means that it is invariant to translation. Suppose you are computing $K(x, y)$. A stationary kernel will yield the same value $K(x, y)$ for $K(x + c, y + c)$, where c may be vector.For the RBF, this is done by working on the difference of the two vectors. Also note that the linear kernel does not have this property.
Other kernels that are very close to this are exponential Kernel

$$k(x, y) = exp(-||x - y||/(2\sigma^2))$$

and laplacian Kernel

$$k(x, y) = exp(-||x - y||/(\sigma))$$

It is important to note that the observations made about the sigma parameter for the Gaussian kernel also apply to the Exponential and Laplacian kernels. Now we will comment on the VC dimension of rbf kernel. It's simply infinity. Gaussian kernels can classify any set of points exactly. We can justify this fact by saying that VC dimension correspond to the number of dichotomies we can get. This clearly depends on the effective number of parameters. This is simply the weights of the classifier and we know that this corresponds to a point in a Hilbert space. Now with rbf kernel we work with infinite dimensional space. Thus with this argument we can say that the VC dimension of rbf kernel is infinite.

9.4 ANOVA kernels

The ANOVA kernel of degree d is like the all-subsets kernel except that it is restricted to subsets of the given cardinality d. We can represent them as

$$\phi_i(x) = x_1^{i_1} \dots x_n^{i_n}$$

with the restriction that i=$(i_{1n}) \epsilon 0, 1^n$. It has another restriction that

$$\sum_{j=1}^{n} i_j = d$$

ANOVA stands for ANalysis Of VAriance.

9.5 Locality Improved kernels

Suppose we have a image and we have to use kernel on it. We can use various kernels like polynomial kernel or Gaussian RBF. But these kernels don't make use of the locality of the image. We can permute the pixels of the image and still these kernels will give the same result.By using the locality we can get a better result. One way is to use pyramidal kernel. It takes inner products between corresponding image patches, then raises the latter to some power p1 , and finally raises their sum to another power p2 . While the overall degree of this kernel is p1 p2 , the first factor p1 only captures short range interactions.

9.6 Kernels on Sets

These are defined on non-vectorial space. Consider the set of subsets P(D) of a fixed domain D. The elements of the input space are therefore sets. They could include all the subsets of D or some sub-selection of the whole power set

$$X \subset P(D)$$

Consider two elements of X $A_1, A2$ and they also are subsets of domain D. Now we can define the kernels like intersection kernel, union complement kernel which are based on basic set operations like union,intersection and complement.

9.7 Kernels for text

We will describe the bag of words model here.A bag is a set in which repeated elements are allowed, so that not only the presence of a word but also its frequency is taken into account. Here the ordering of words are ignored. So the grammatical information is lost. Also the phrases are broken and so there meaning is lost. We refer to this full set of documents as the corpus and the set of terms occurring in the corpus as the dictionary. We can represent a bag as a vector in a space in which each dimension is associated with one term from the dictionary. We store the terms which occur in the document and there corresponding frequency. Clearly the size of this vector is very large. But most of it's entries are 0.

Document Term Matrix: The document–term matrix of a corpus is the matrix whose rows are indexed by the documents of the corpus and whose columns are indexed by the terms.The $(i, j)^{th}$ entry gives the frequency of term t_j in document d_i. The term–document matrix is the transpose D' of the document–term matrix. The term-by-term matrix is given by D'D while the document-by-document matrix is DD'.

Vector space kernel: This is given by the document-by-document matrix DD'.

In order to compute the kernel k we must first convert the document into a list of the terms that they contain. This process is known as tokenisation.Each term encountered in the corpus is assigned its own unique number. This ensures that the list of terms in a document can be reordered into ascending term order together with an associated frequency count.

9.8 Kernels for strings

Spectrum kernels: In this we count how many substrings of length p they have in common. We define the spectrum of order p (or p-spectrum) of a sequence s to be the histogram of frequencies of all its substrings of length p. Comparing the p-spectra of two strings can give important information about their similarity in applications where contiguity plays an important role. We can define a kernel as the inner product of their p-spectra.
In this we construct all possible substrings of length p and find out which of them are present in our string. In this way we construct a vector corresponding to it. Now the job of the kernel is to find out the inner product of the vectors corresponding to our input strings.

All-subsequences kernel: It's working is similar to the spectrum kernel. The difference is instead of matching all possible substrings of length p, we will match all possible susequences. Clearly the vector space here is much larger than the previous one.

9.9 Convolution and structures

Here we define kernels on structured objects. Suppose the object x ϵ X is composed of $x_p \epsilon X_p$. Haussler investigated how to define a kernel between composite objects by building on similarity measures that assess their respective parts. Define the R-convolution of $k1 \ldots kp$ as

$$[k_1 * \ldots * k_p](x, y) = \sum_{x' \epsilon R(x) y' \epsilon R(y)} \prod_{p=1}^{P} k_p(x - p', y'_p)$$

where the sum runs over all possible ways R(x) and R(x') in which we can decompose x into $x_1 \ldots x_p$. R-convolution is a valid kernel.

10 Representer theorem

This theorem says that a minimizer f of a regularized empirical risk function defined over a reproducing kernel Hilbert space can be represented as a finite linear combination of kernel products evaluated on the input points in the training set data.
The representer theorem shows that solutions of a large class of optimization problems can be expressed as kernel expansions over the sample points. We define $\Omega : [0, \infty) \mapsto R$ a strictly monotonic increasing function, a set X and a arbitrary loss function c: $(X * R * R)^n \mapsto RU\infty$. Then each minimizer f ϵ H of the regularized risk functional

$$c((x_1, y_1, f(x_1)), \ldots (x_n, y_n, f(x_n))) + \Omega(||f||_H^2)$$

can be expressed in the form

$$f(x) = \sum_{i=1}^{n} \alpha_i k(x_i, x)$$

Monotonicity of Ω does not prevent the regularized risk functional from having multiple local minima. If we discard the strictness of the monotonicity, then it no longer follows that each minimizer of the regularized risk admits an expansion. Now we will explain the significance of this theorem. Even if we are solving a optimization problem in a infinite dimensional space H, containing linear combinations of kernels centered on arbitrary points of X, the solution within the span of n particular kernels centered on the training points. On solving we will find that most of the α will turn out to be zero. We will see this further in the next section. The values for which α is not zero will actually determine the solution. We use this to find the weights.

11 Support Vector Machine for Classification

In this section we will show how kernel can help us in classification.
Suppose we have a linearly separable data. We wish to find a line that can

classify the two classes. There can be many such lines. Among those we wanna pick the one which has the maximum margin. By that we mean that the distance from the closest point to the separating line should be as large as possible. This line like any other separating line have the same in-sample error. The reason we pick this line is if the data generated is not accurate the classifier with fat margin has better chance of classifying it correctly as compared to classifier with thin margin.
Now we have to find the weights for which we have the classifier with fat margin. Suppose we have input points in d dimensional space. We need to find the plane which classify the points correctly and have a fat margin.
Equation of this plane is $w^T x = 0$. Let x_n be the nearest data point to this plane. We will mention a important point before we move forward. It's obvious that for any point in the dataset not on the plane we have $|w^T x_n| > 0$. Now we can multiply our weight by any number and still we will have the same plane. So we will the pick one which is normalized. We do that enforcing the constraint that $|w^T x_n| = 1$. We will also have a bias b. Now equation of our plane is $w^T x + b = 0$ and our constraint is $|w^T x_n + b| = 1$. The vector w is perpendicular to our plane. To see this take any two points on the plane and take their difference and we will find their dot product to w as zero $w^T(x - x') = 0$. Now we have to find the distance of point x_n to the plane.
Take any point x on the plane. Take the projection of $x_n - x$ on w. This is our distance. Unit vector of w

$$w/||w||$$

Projection is the dot product of our unit vector and $x_n - x$ which is

$$|w^T x_n - w^T x|/||w|| = |w^T x_n + b - (w^T x + b)|/||w|| = 1/||w||$$

Now our optimization problem is to maximize $1/||w||$. This is similar as if we have to minimize $w^T w/2$. And our constraint is $y_n(w^T x_n + b) \geq 1$ for all n. To solve this we will use Lagrange method

$$L(w, b, \alpha) = 1/2 w_T w - \sum_{n=1}^{N} \alpha_n (y_n(w^T x_n + b) - 1)$$

Minimize w.r.t w and b maximize w.r.t each $\alpha_n \geq 0$.
Take derivatives w.r.t w and b

$$\nabla_w L = w - \sum_{n=1}^{N} \alpha_n y_n x_n = 0$$

$$\nabla_b L = - \sum_{n=1}^{N} \alpha_n y_n = 0$$

we get

$$w = \sum_{n=1}^{N} \alpha_n y_n x_n$$

$$\sum_{n=1}^{N} \alpha_n y_n = 0$$

substitute these equations in the Lagrangian. We get

$$L(\alpha) = \sum_{n=1}^{N} \alpha_n - (1/2) \sum_{n=1}^{N} \sum_{m=1}^{N} y_n y_m \alpha_n \alpha_m x_n^T x_m$$

Maximize w.r.t α such that $\alpha_n \geq 0$ for all n ϵ N and $\sum_{n=1}^{N} \alpha_n y_n = 0$ We solve this for α using quadratic programming technique. From the equation we see that the complexity depends on the number of examples N.
After solving we will get the value of α. From this we can get w by

$$w = \sum_{n=1}^{N} \alpha_n y_n x_n$$

After solving we will find that most of the values in α are zero. Here we define the KKT condition

$$\alpha_n (y_n (w^T x_n + b) - 1) = 0$$

Those points for which α_n is greater than zero are the support vectors. These are the points which actually define the margin. Since there α_n greater than zero they only define w. These are very less in number.
Solve for b using any support vector

$$y_n (w^T x_n + b) = 1$$

Now if we want to solve for nonlinear data we can go to a higher dimension space Z in which our data is linearly separable and apply the quadratic programming in that space to get our result. Our equation here is simply

$$L(\alpha) = \sum_{n=1}^{N} \alpha_n - (1/2) \sum_{n=1}^{N} \sum_{m=1}^{N} y_n y_m \alpha_n \alpha_m z_n^T z_m$$

$$w = \sum_{n=1}^{N} \alpha_n y_n z_n$$

where z are features in higher dimension. Now the support vectors we get out of this live in the Z space. There pre-image in the X space might not make a lot of sense in some cases. The margin is also maintained in the Z space.

And finally we define our hypothesis as

$$g(x) = sign(w^T z + b)$$

What we really require from the Z space is the inner product. We need it in the computation of $L(\alpha)$, b and hypothesis function. Instead of doing this we can directly compute the inner product using kernels. This won't have any effect on quadratic programming computation. Using kernels we can improve the performance as doing it the normal way will be a expensive operation.

$$L(\alpha) = \sum_{n=1}^{N} \alpha_n - (1/2) \sum_{n=1}^{N} \sum_{m=1}^{N} y_n y_m \alpha_n \alpha_m K(x_n, x_m)$$

$$g(x) = sign(\sum_{n=1}^{N} \alpha_n y_n K(x_n, x) + b)$$

Soft Margin Classifier

What we discussed here is actually the hard margin classifier. In there is no scope of making any error. We need a model in which we can make some errors. So now we define the soft margin classifier. The solution to this is very similar to the one we just discussed. In this we will also consider about margin violations. We will introduce a new factor E_n which defines the amount of margin violation done by point n. Now the total violations is

$$\sum_{n=1}^{N} E_n$$

Now our optimization problem is to minimize $w^T w/2 + C \sum_{n=1}^{N} E_n$. And our constraint is $y_n(w^T x_n + b) \geq 1 - E_n$ and $E_n \geq 0$ for all n.
Now again we will apply Lagrange method

$$L(w, b, E, \alpha, \beta) = 1/2 w_T w + C \sum_{n=1}^{N} E_n - \sum_{n=1}^{N} \alpha_n (y_n (w^T x_n + b) - 1 + E_n) - \sum_{n=1}^{N} \beta_n E_n$$

Minimize w.r.t w,b and E maximize w.r.t each $\alpha_n \geq 0$ and $\beta_n \geq 0$.
Again take the derivatives w.r.t w,b and E_n

$$\nabla_w L = w - \sum_{n=1}^{N} \alpha_n y_n x_n = 0$$

$$\nabla_b L = - \sum_{n=1}^{N} \alpha_n y_n = 0$$

$$\nabla_{E_n} L = C - \alpha_n - \beta_n = 0$$

Substituting it our solution we get reduced to

$$L(\alpha) = \sum_{n=1}^{N} \alpha_n - (1/2) \sum_{n=1}^{N} \sum_{m=1}^{N} y_n y_m \alpha_n \alpha_m x_n^T x_m$$

Maximize w.r.t α such that $\alpha_n \geq 0$ and $\alpha_n \leq C$ for all n ϵ N and $\sum_{n=1}^{N} \alpha_n y_n = 0$. Now again feed it to the quadratic programming technique and get the value of α and eventually the support vectors. Among these margin support vectors are the ones which have $E_n = 0$.

12 Regression Estimation

In this we first introduce the ϵ insensitive loss function. What it does is that it ignores errors which are less than a threshold .We introduce this loss so that we can apply the kernel trick in regression. Here we extend the idea of soft margin to regression problems. instead of requiring that yf(x) exceeds some margin value, we now require that the values yf(x) are bounded by a margin on both sides. Thus we introduce the following constraints

$$y_i - f(x_i) \leq \epsilon_i - E_i$$

$$f(x_i) - y_i \leq \epsilon_i - E_i^*$$

where E_i and E_i^* are non negative. If $y_i - f(x_i) \leq \epsilon$ than no penalty is imposed. The objective function is given by the sum of the slack variables E_i and E_i^* and $1/2||w||^2$. This is equivalent to solving

$$minimize 1/2||w||^2 + \sum_{i=1}^{n} \varphi(y_i - f(x_i))$$

where we can choose different values of loss function φ
Least Square regression

$$\varphi(E) = (1/2)E_2$$

Least absolute deviations

$$\varphi(E) = |E|$$

We can use Lagrange and quadratic programming techniques to solve this as we did before.

13 Multicategory classification

Many estimation problems cannot be described by assuming that y=+1,-1. In this case it is advantageous to go beyond simple functions f(x) depending on x only. Instead, we can encode a larger degree of information by estimating a function f(x,y). In this y obtained as a solution of an optimization problem over f(x,y). And we need to do this in such a way that our solution matches the actual solution.
The loss may be more than just a simple 0–1 loss. In the following we denote by $\delta(y, y')$ the loss incurred by estimating y' instead of y. Without loss of generality, we require that $\delta(y, y) = 0$ and that $\delta(y, y') \geq 0$ for all y, y'ϵ Y.
To deal with this we define f as

$$f(x, y) = \langle \phi(x, y), w \rangle$$

Corresponding kernel function are given by

$$k(x, y, x', y') = \langle \phi(x, y), \phi(x', y') \rangle$$

We have the following optimization problem

$$minimize(1/2)w^T w + C\sum_{i=1}^{N} E_i$$

such that

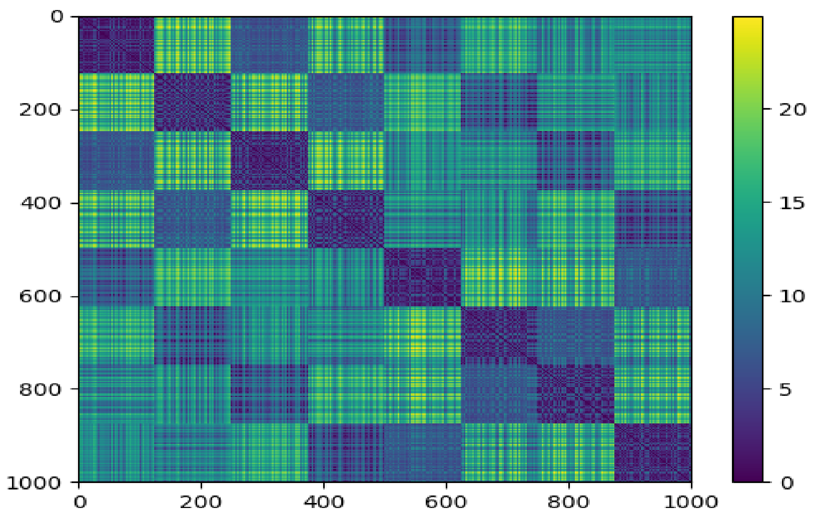
$$\langle w, \phi(x_i, y_i) - \phi(x_i, y_i) \rangle \geq \delta(y, y_i) - E_i$$

We can solve this efficiently if constraint are evaluated without heavy computation.

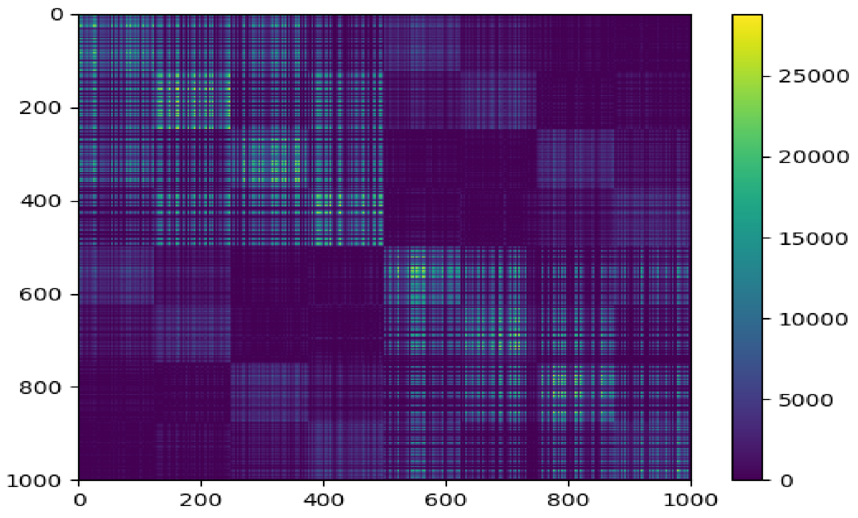
14 Results

For different type of datasets we have computed Euclidian Distance Matrix, Gram Matrix for polynomial kernel, Gram matrix for rbf kernel. For rbf kernel we have computed 3 matrix one for each of σ value. From this we can clearly say that with low value of σ rbf kernel gives high similarity value only to those points that are very near to it. For high value of σ points that are far away are also assigned good enough similarity value. This is correct behaviour as sigma corresponds to variance. Also the diagonal value have low value in euclidian matrix but high similarity value in the gram matrix of both polynomial and rbf kernel. This is correct as points which are closer are assigned similarity value.
For rbf kernel for low σ values the gram matrix is very dark as most of the points are assigned very less similarity value. For high values the matrix is very clear as the variance is high and points far away are also assigned significant similarity value. These matrices depend a lot on the type of datasets we have. In general if we have a dataset in which all points are very close than the corresponding gram matrix will be very clear as all of the points are assigned high similarity value. For better clarity refer to dataset images in the appendix.
For the two spiral dataset polynomial kernel doesn't give much information as it assigns similar similarity value to all data points. This is justified by the fact that this kernel give only 51.8 percent accuracy on two spiral data. Where as in other datasets it gives better performance and also the matrix is better in them.

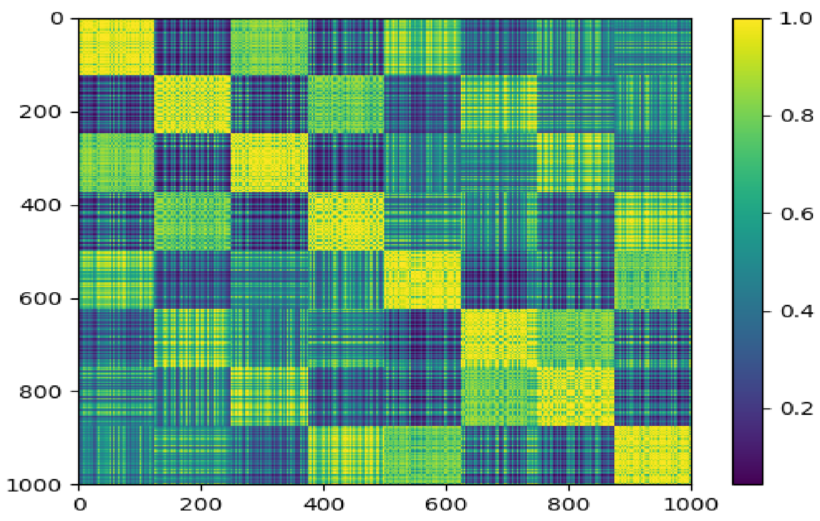
Corner Dataset



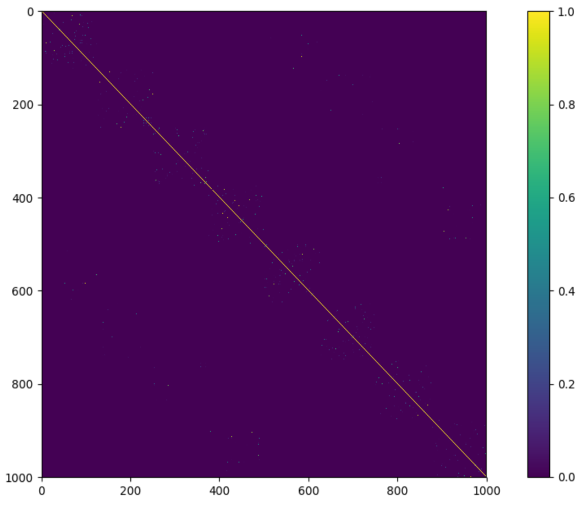
Euclidian Distance Matrix



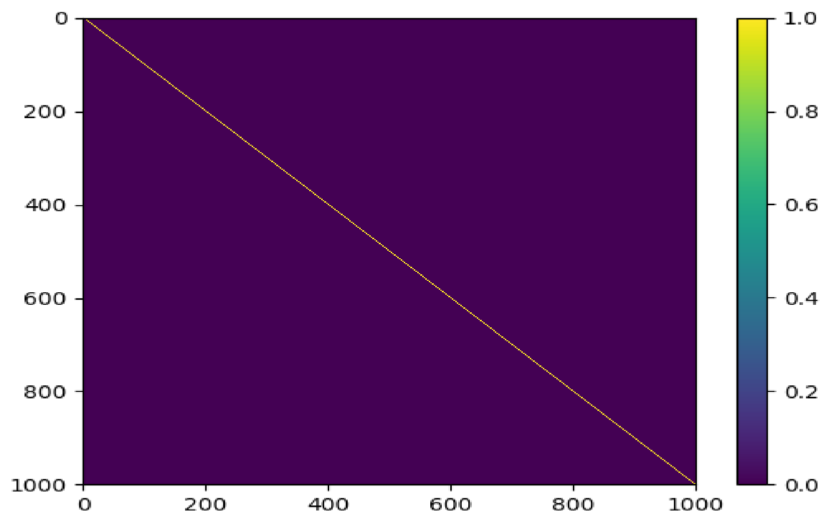
Polynomial kernel gram Matrix



RBF kernel gram Matrix with sigma 10

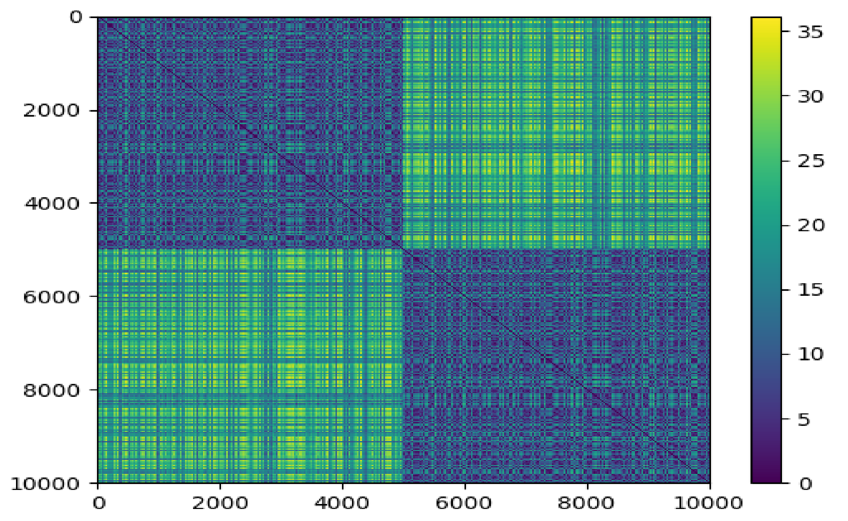


RBF kernel gram Matrix with sigma 0.1

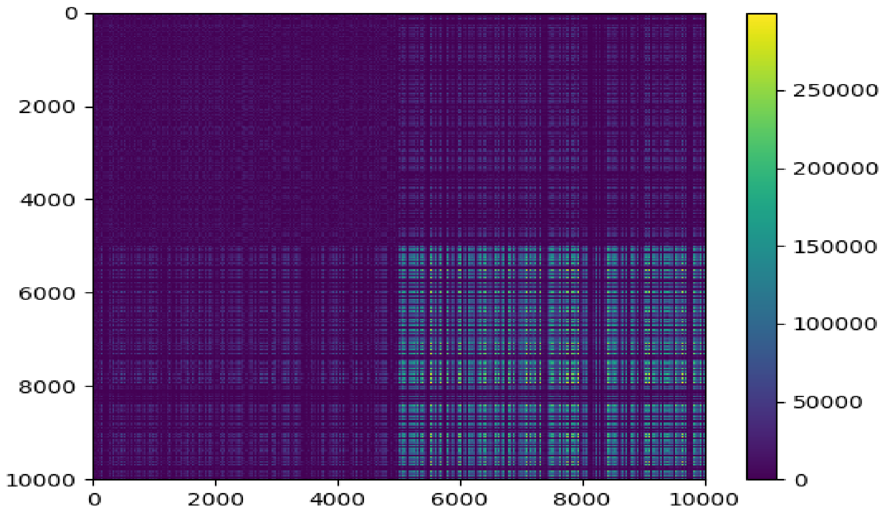


RBF kernel gram Matrix with sigma 0.001

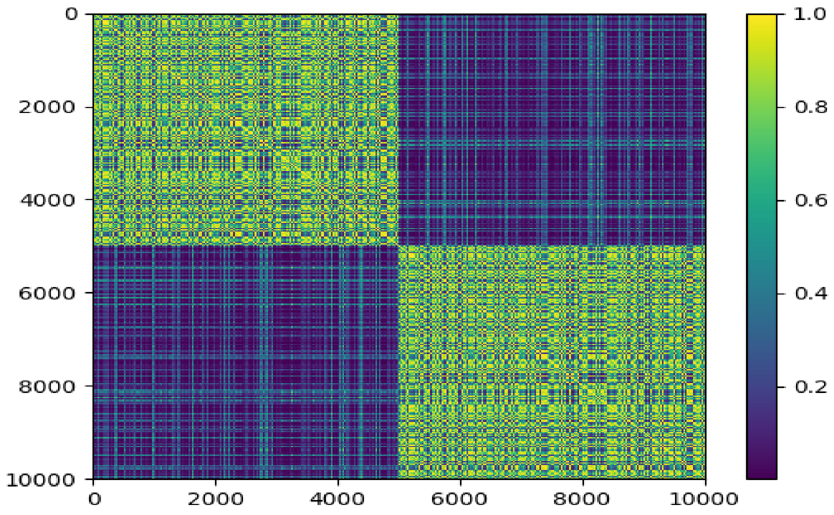
Double Moon Dataset



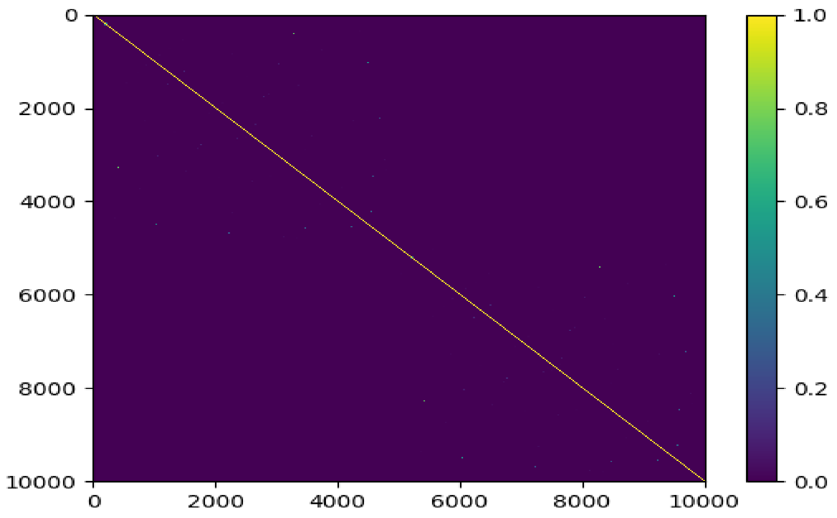
Euclidian Distance Matrix



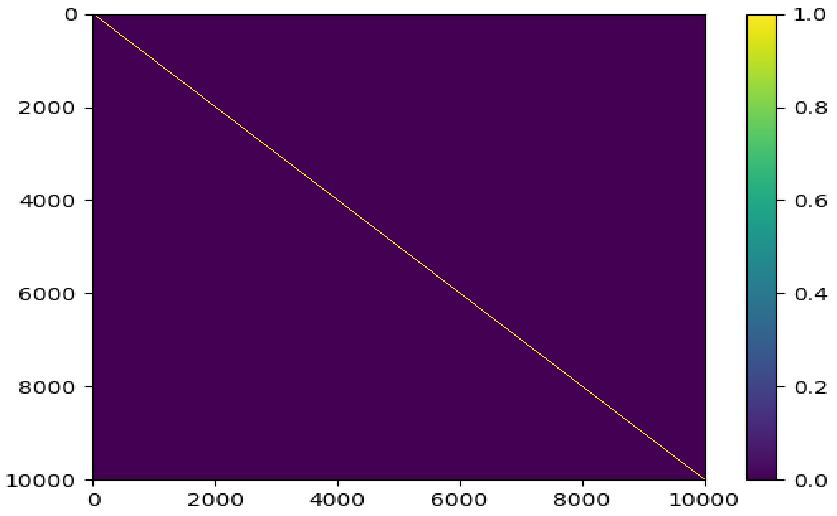
Polynomial kernel gram Matrix



RBF kernel gram Matrix with sigma 10

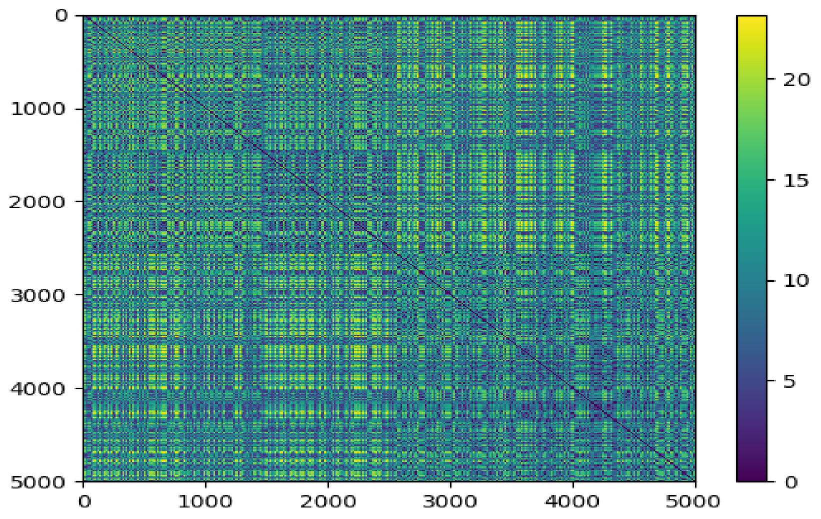


RBF kernel gram Matrix with sigma 0.1

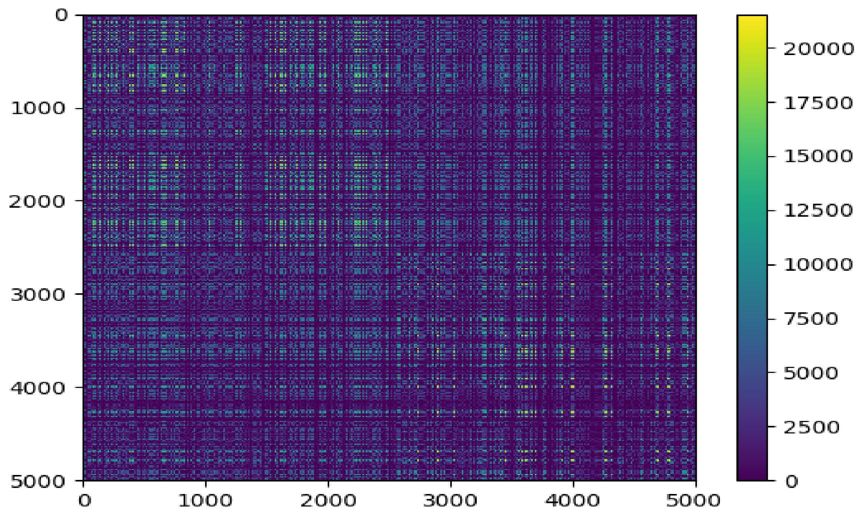


RBF kernel gram Matrix with sigma 0.001

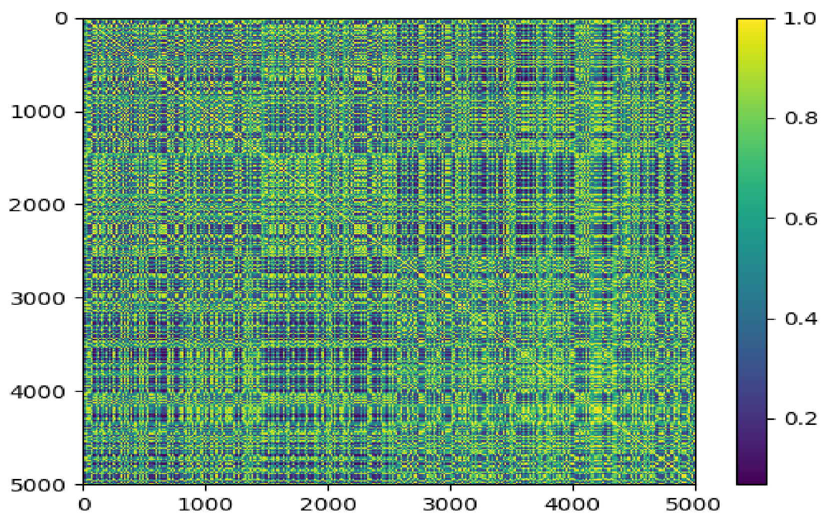
TwoSpiral Dataset

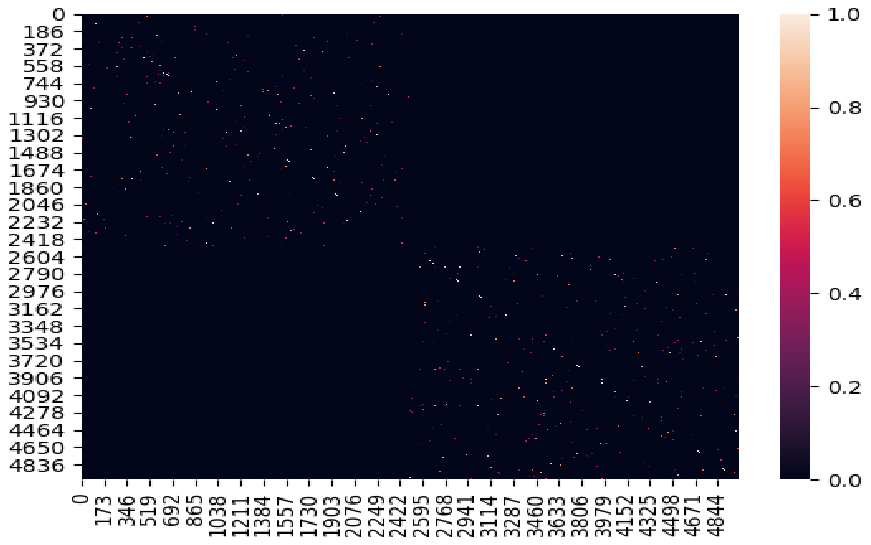


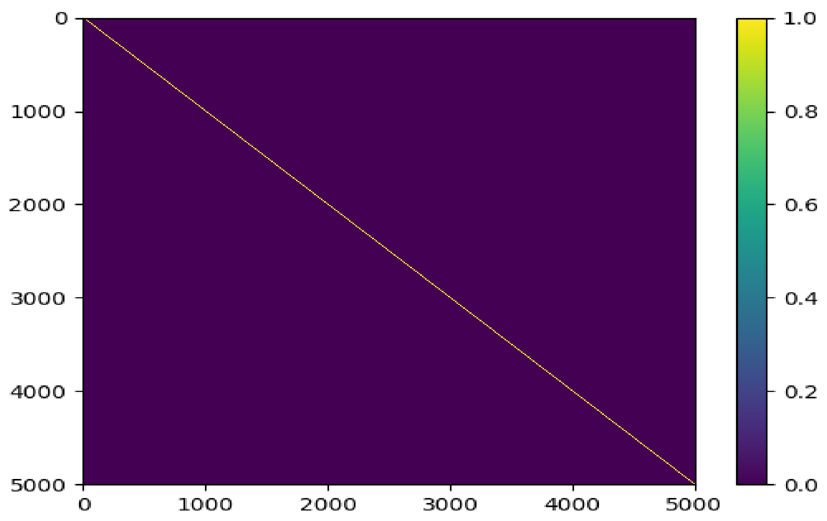
Euclidian Distance Matrix



Polynomial kernel gram Matrix

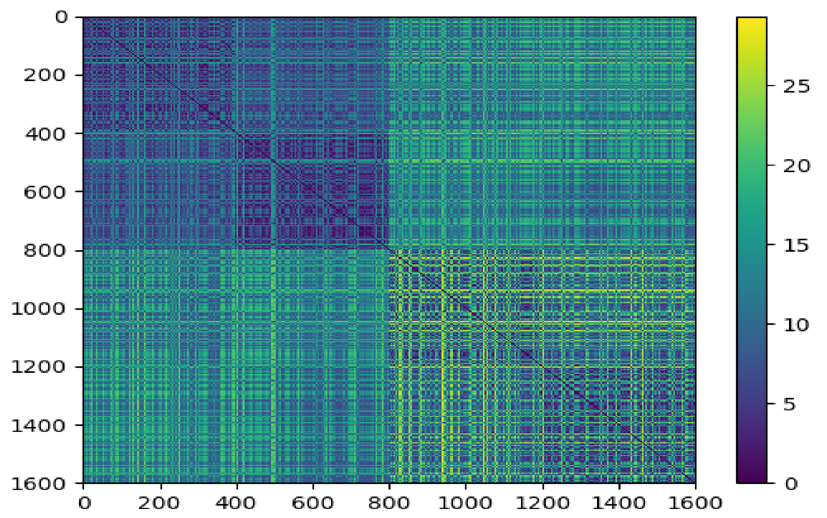

RBF kernel gram Matrix with sigma 10


RBF kernel gram Matrix with sigma 0.1

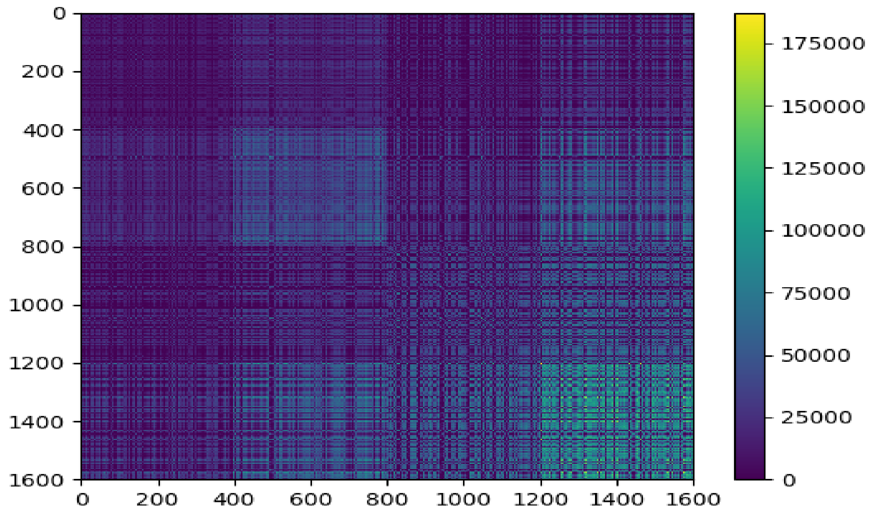


RBF kernel gram Matrix with sigma 0.001

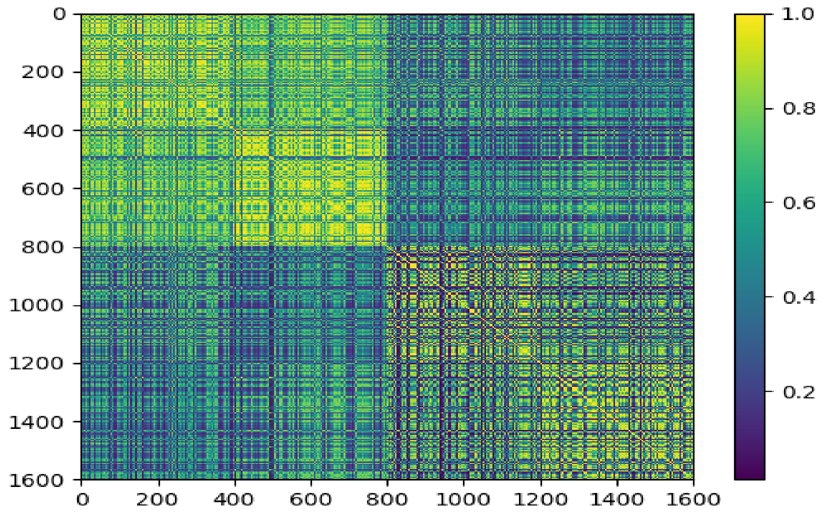
SwissRoll



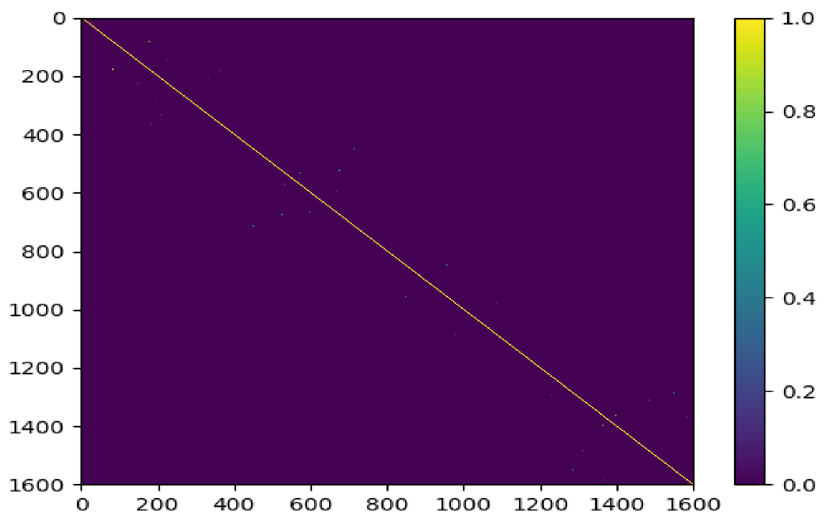
Euclidian Distance Matrix



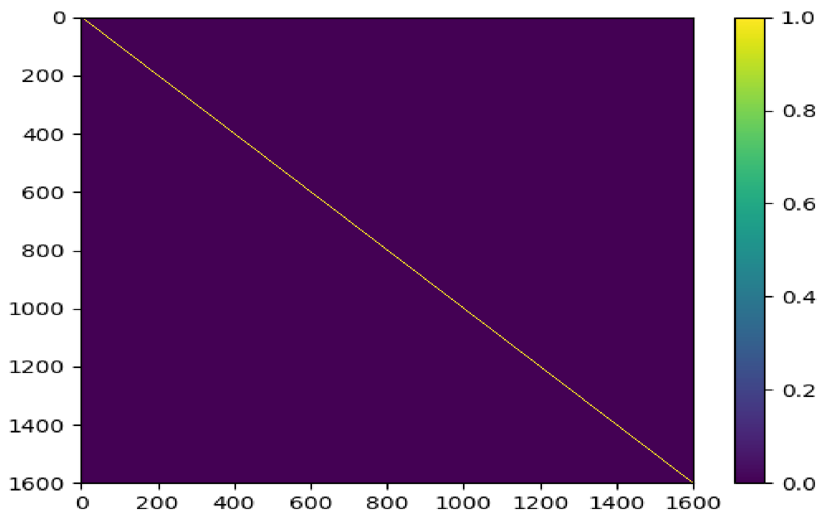
Polynomial kernel gram Matrix



RBF kernel gram Matrix with sigma 10



RBF kernel gram Matrix with sigma 0.1



RBF kernel gram Matrix with sigma 0.001

Accuracy of Generated datasets

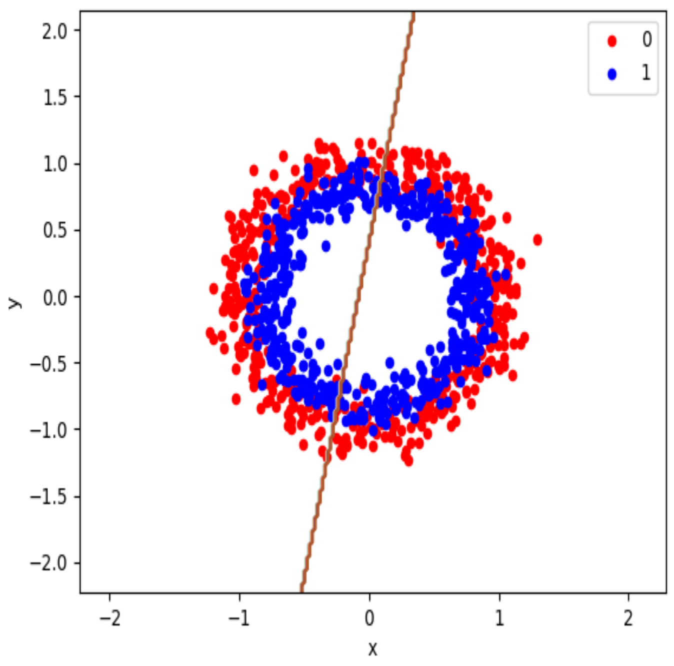
Description of data in Appendix.

Dataset	Linear Kernel	RBF Kernel	Polynomial Kernel
DoubleMoon1	100	100	100
DoubleMoon2	100	100	100
Corners	100	100	100
TwoSpirals	64.45	100	51.8
CircleData	48	81.33	97.67
DoubleMoon3	85.33	86	86.33
Blob	85	86	86
CircleData2	49.67	83.33	83.33
DoubleMoon4	78.66	78.33	73.66
SwissRoll	76.458	97.916	97.916

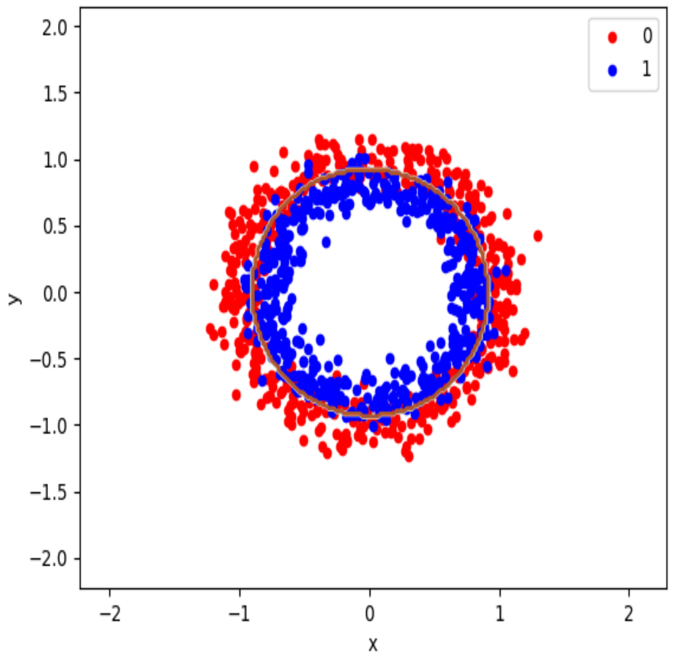
Images of classifier for Generated datasets

Cicles2 Dataset

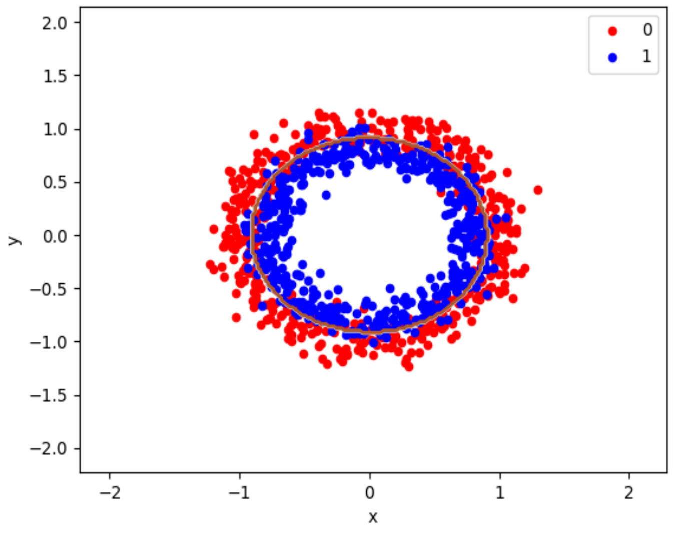
Linear Kernel



RBF Kernel

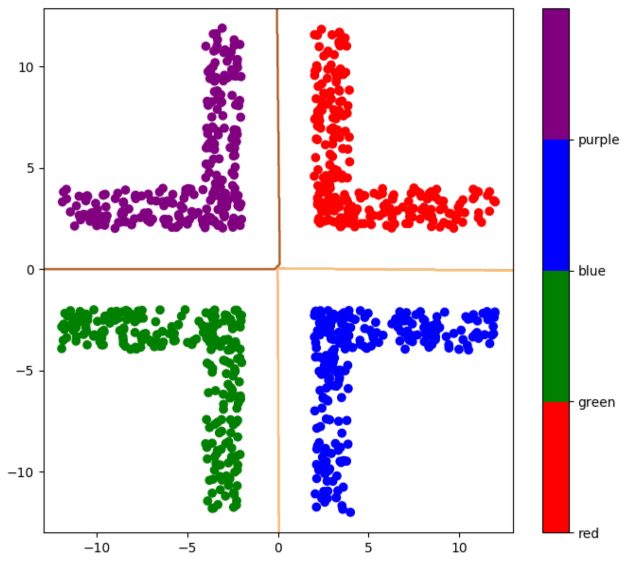


Polynomial Kernel

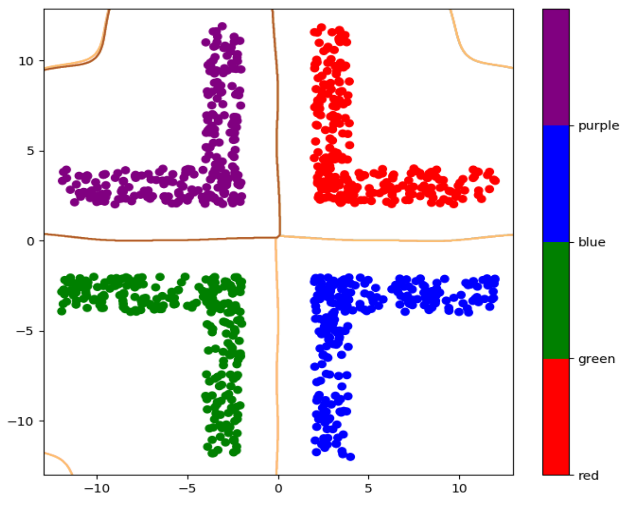


Corners Dataset

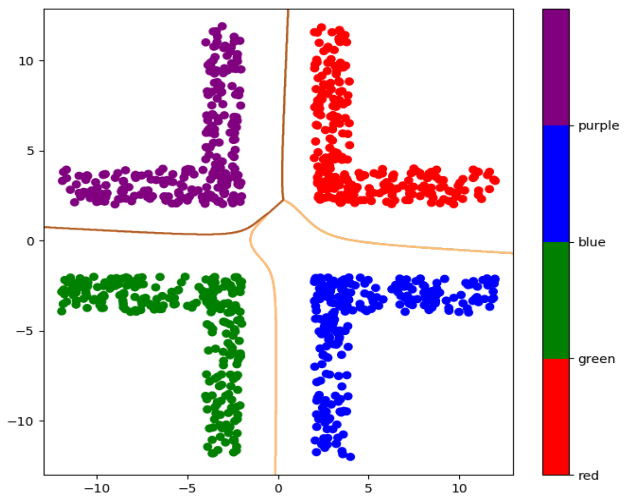
Linear Kernel



RBF Kernel

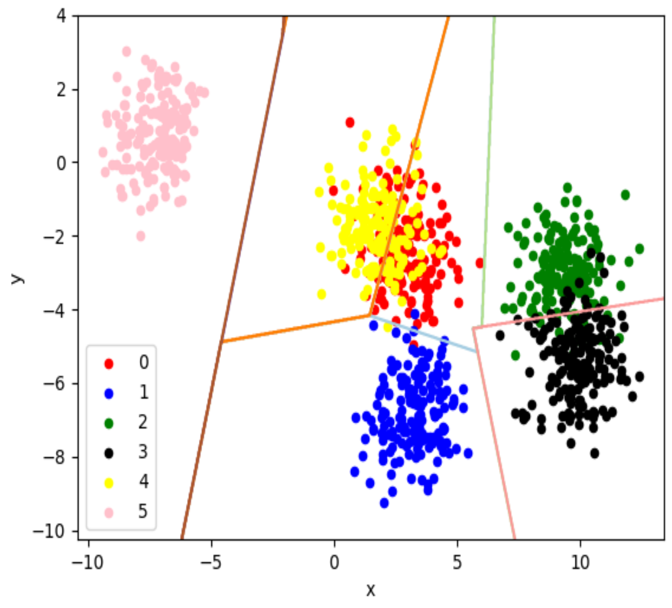


Polynomial Kernel

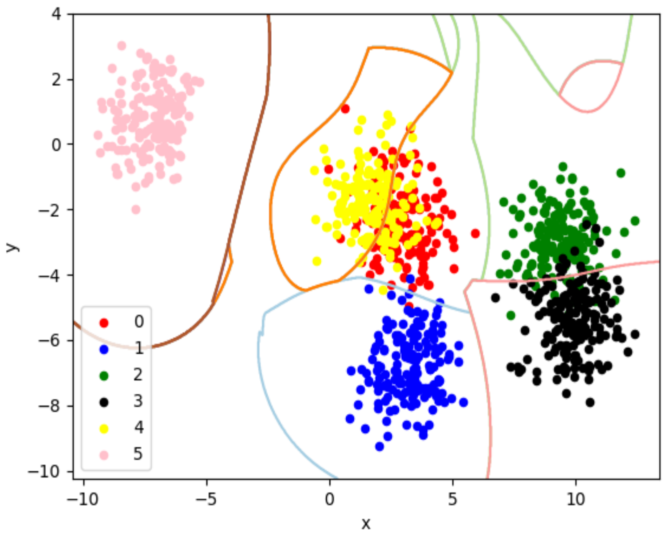


Blob Dataset

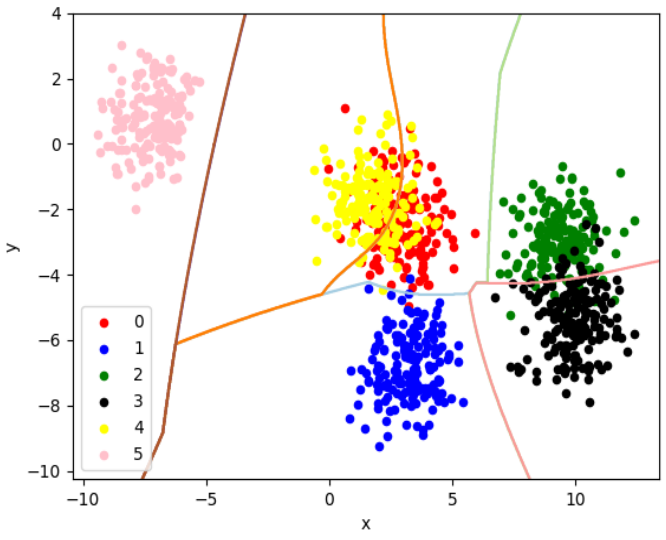
Linear Kernel



RBF Kernel

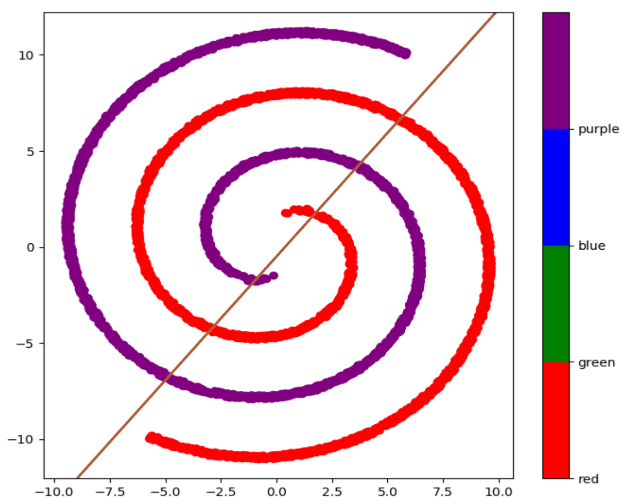


Polynomial Kernel

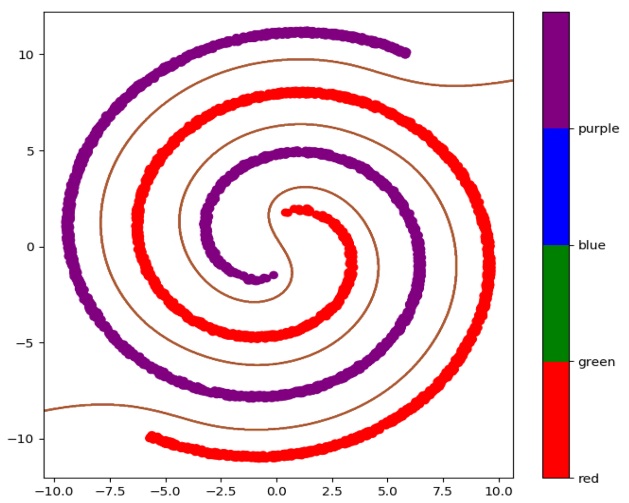


TwoSpirals Dataset

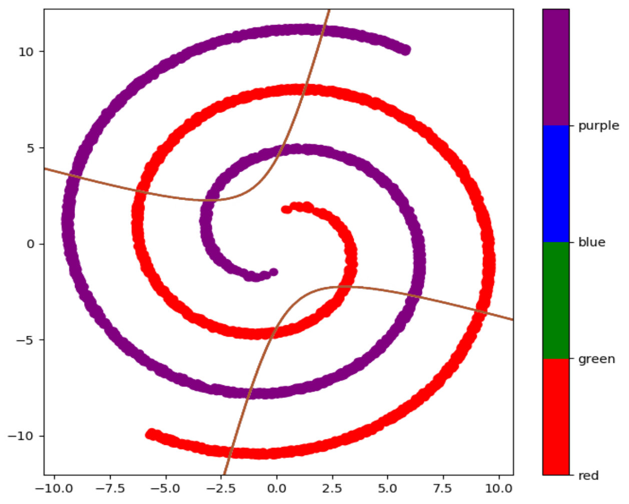
Linear Kernel



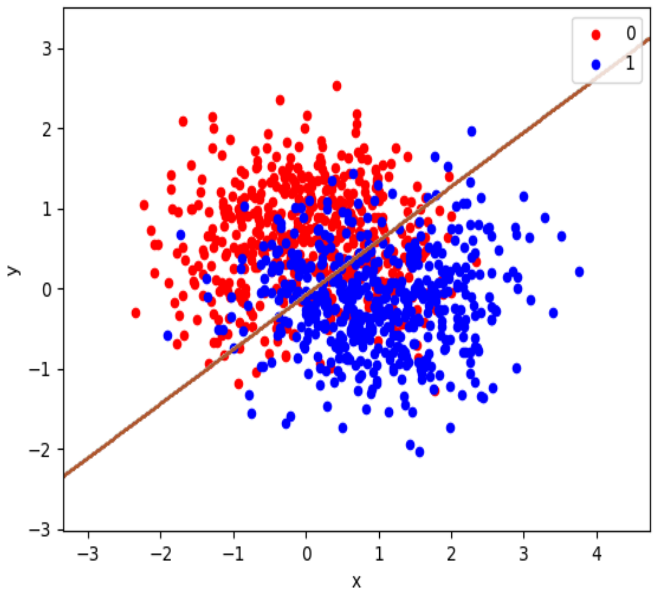
RBF Kernel



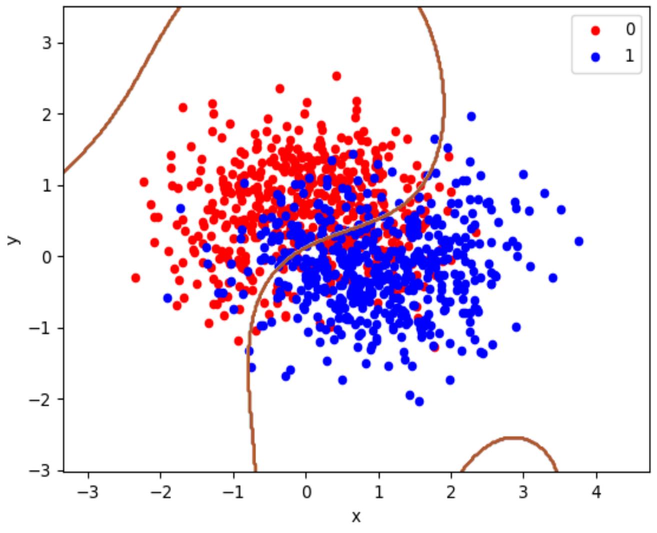
Polynomial Kernel



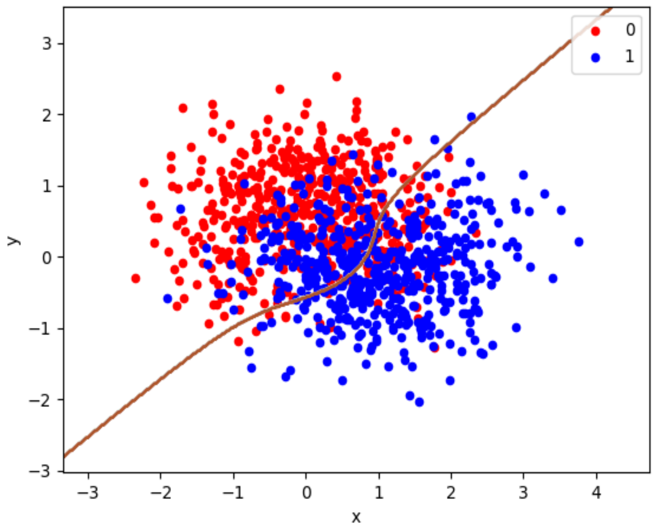
DoubleMoon4
Linear Kernel



RBF Kernel



Polynomial Kernel



Accuracy for Image Datasets

In this we we first obtain the features from the by passing them through VGG architecture. Once we obtain the features we compute the accuracy on them. Now we perform dimensionality reduction on it to obtain features of size 50. Now again we compute the accuracy on them. We perform this on 4 datasets. We have used 70-30 split.
For GHIM and C1k dataset we observe that performance of linear kernel is better than RBF kernel on original features. But the performance of RBF is better than linear on reduced features. For C10K and CIFAR10 rbf is better than linear in both the original and reduced features. Now we justify this by the fact that in higher dimensions there is a high probability for the data to be linearly separable than in lower dimension. So it's better to use linear kernel in such data when there is a better chance for the data to be linearly separable. Another benefit in that is linear kernel is much faster than RBF kernel. In GHIM and C1K the original feature size is 55296 and 49152 respectively. This is very high and it's justified that linear kernel is better in this case. However for C10K and CIFAR10 the original feature sizes are 7680 and 512 respectively. This is low as compared to GHIM and C10K and here RBF kernel will perform better.
Here the accuracy in the first row corresponds to features size 50 and second row is for original features.

GHIM Dataset

Linear Kernel	RBF kernel	Polynomial Kernel
79.067	82.67	80.2
88.43	85.16	82.73

C1K Dataset

Linear Kernel	RBF Kernel	Polynomial Kernel
82.33	86.33	78
87.33	81.33	80.66

C10K Dataset

Linear Kernel	RBF Kernel	Polynomial Kernel
60.53	61.4	60.66
68.767	69.36	63.633

CIFAR10 Dataset

Linear Kernel	RBF Kernel	Polynomial Kernel
93.48	93.51	93.42
93.45	93.46	93.47

Accuracy for different parameters for Polynomial kernel

In polynomial kernel generally use degree as 2 or 3 as with increasing degree there is a greater chance of overfitting. This can be observed from the following data.
In circle dataset polynomial with degree 2 performs better as this equation better represent the equation of circle. In double moon data polynomial with degree perform 3 better as that polynomial represent the shape better. The images for datasets are given in the appendix for clarity.

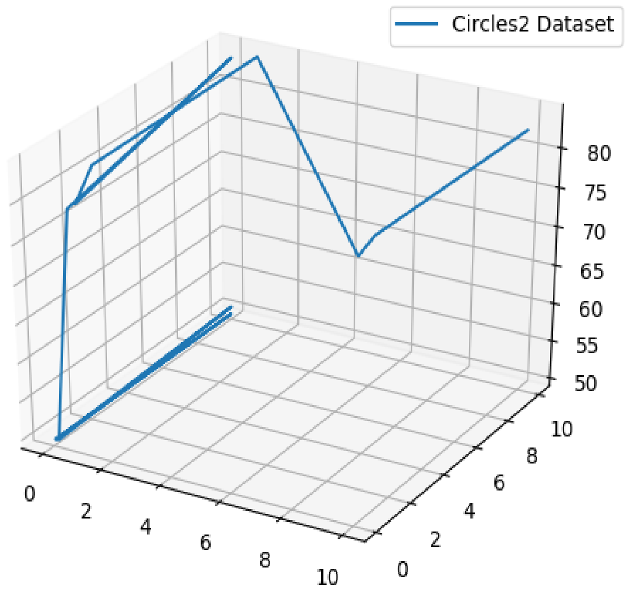
Dataset	Degree 2	Degree 3	Degree 4	Degree 5
Circle2	83.33	54	83	61.3
DoubleMoon4	62	73.67	59.33	71.67
Blob	84	86	84.33	84.66

Accuracy for different parameters for RBF kernel

We know that large value of gamma will result into high variance and low bias. The choice of C and γ depend on the dataset used. For circular data we simply need more support vectors and here the value of γ has a very important role as compared to C. Higher value of γ will result into higher variance and thus better result. For double moon 4 dataset there is too much noise and thus the value of C has a important role in this.

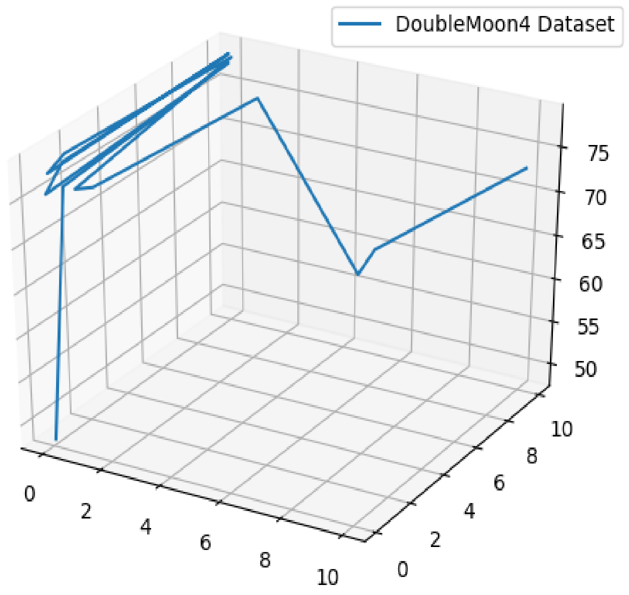
Circle2 Dataset

Gamma	C	Accuracy
0.001	0.1	49.67
0.001	1	49.67
0.001	10	49.67
0.01	0.1	49.67
0.01	1	49.67
0.01	10	50.67
0.1	0.1	49.67
0.1	1	77.33
0.1	1	83.33
1	0.1	80.66
1	1	84
1	10	84.66
10	0.1	82.33
10	1	83.33
10	10	82.66

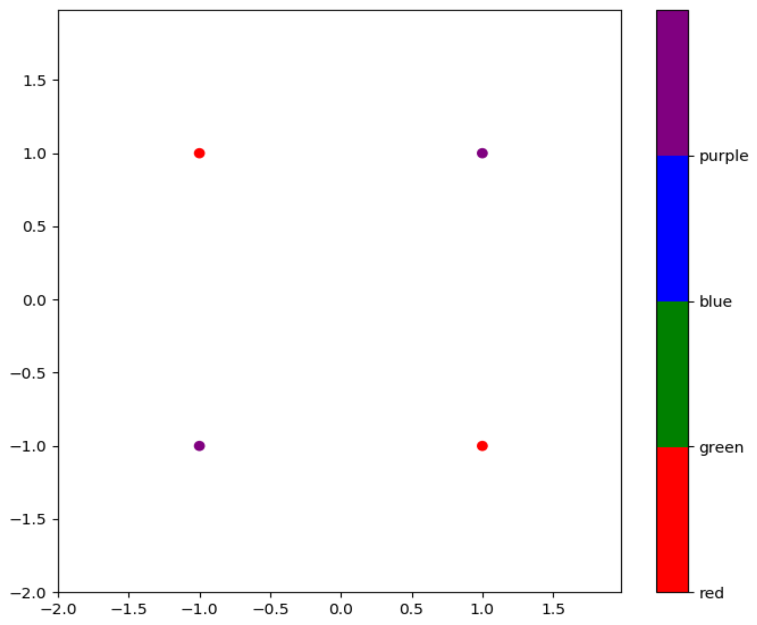


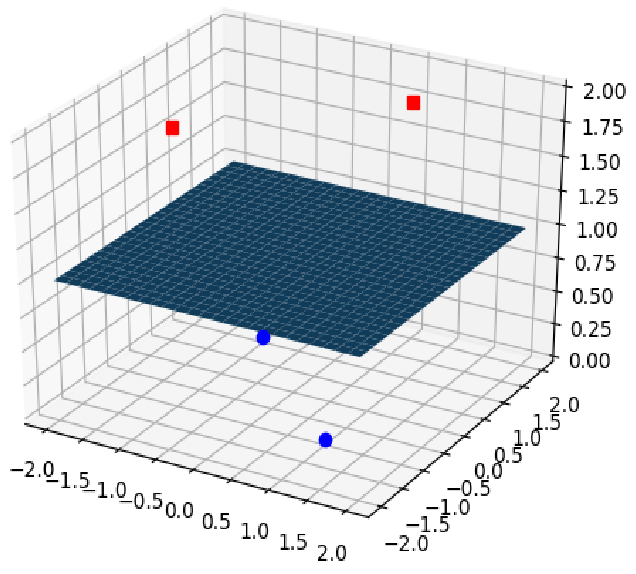
DoubleMoon4 Dataset

Gamma	C	Accuracy
0.001	0.1	48
0.001	1	75.33
0.001	10	77.6
0.01	0.1	75.67
0.01	1	78
0.01	10	78.66
0.1	0.1	78
0.1	1	79
0.1	1	78.33
1	0.1	77
1	1	76
1	10	74.33
10	0.1	75
10	1	76.33
10	10	73



XOR Problem Illustration





15 Appendix

15.1 Inner Product Space

A vector space X over the reals R is an inner product space if there exists a real-valued symmetric bilinear map, that satisfies

$$\langle x, x \rangle \geq 0$$

The bilinear map is known as the inner, dot or scalar product.

15.2 Positive semi-definite matrices

A symmetric matrix K is positive semi-definite, if it's eigen values are all non-negative. By Courant–Fisher theorem this holds if and only if

$$c' K c \geq 0$$

or

$$\sum_{i,j} c_i c j K_{ij} \geq 0$$

for all $c_i \epsilon R$. Gram and Kernel matrices are poistive semi-definite.

15.3 Dichotomy

It can be defined as a partition of a set S int two disjoint subset. Just like we define hypothesis for our input space X, we can define dichotomy in similar way on some points selected from this input space instead of the whole input space.

15.4 Growth Function

The growth function counts the most number of dichotomies on N points. It is given by $m_H(N)$. The growth function satisfies $m_H(N) \leq 2^N$ i.e all possible assignment of labels to n points.

15.5 VC Dimension

VC dimension for a hypothesis set H can be defined as the largest value of N for which $m_H(N) = 2^N$ i.e. the most points H can shatter. VC dimension correspond to the degree of freedom and thus on the effective number of parameters.

15.6 Error Measure

We define the error made on the training set by a hypothesis as E_{in}. We call this in-sample error or the empirical risk. We define the error on unseen data as E_{out}. We call this generalization error.

Our target is to minimize the generalization error. But we can't do it directly as we don't have the data for it. What we have is the in-sample error. So we want to minimize the difference between generalization error and in-sample error.
We define a hypothesis as bad if the difference between generalization error and in-sample error is greater than ϵ. If we can get the probability of picking a bad hypothesis we can get the desired result. Since we can't get it directly, we require a upper bound for it.
One solution to it is given by Heoffding's inequality

$$P[|E_{in} - E_{out}| > \epsilon] \leq 2Me^{-2\epsilon_2 N}$$

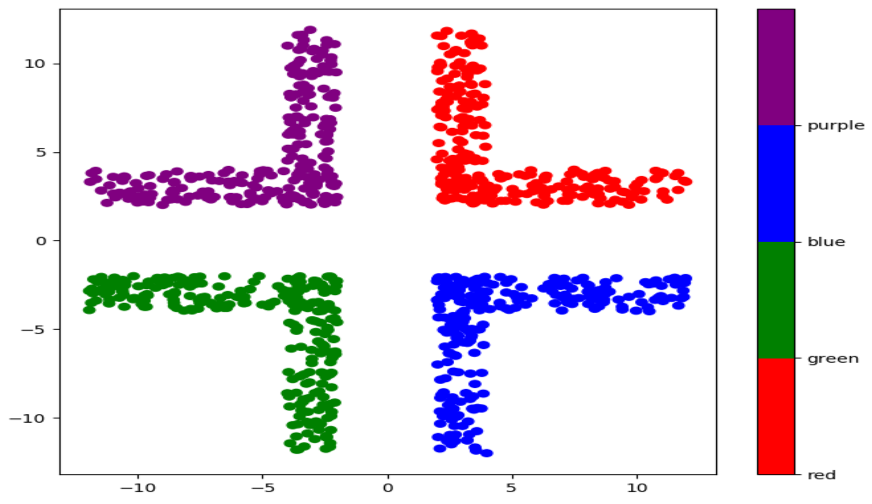
where M is the number of hypothesis. Since the value of can be very large. This inequality doesn't provide a good upper bound to us. We can reduce this by using dichotomy in place of hypothesis and thus can replace M by m i.e growth function. Now we will state without proof that m is a polynomial if there exist a breakpoint for N points.
What we did up till now as for only single hypothesis. But machine learning algorithms doesn't know about the hypothesis. It has to select on single hypothesis out of H.
One way to handle this is to use union bound inequality

$$P[|E_{in} - E_{out}| > \epsilon] \leq \sum_{h \epsilon H} P[|E_{in}(h) - E_{out(h)}| > \epsilon]$$

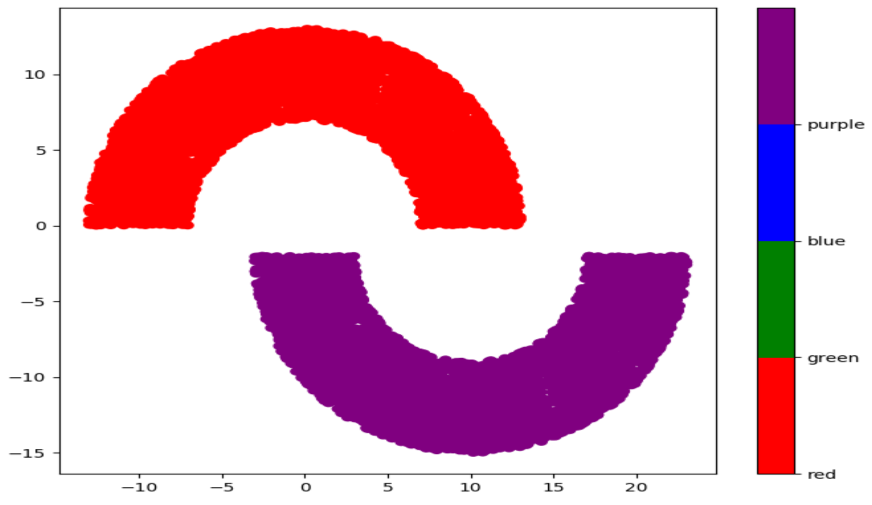
Here also our bound is very large as we are considering all such probabilities as independent i.e. there is no overlap here. However this is not the case. We can provide a much tighter bound using the VC bound. In this since we can't comment on the test data we will use a ghost dataset to provide a approximation. Without going into much details we state it.

$$P[|E_{in} - E_{out}| > \epsilon] \leq 4m_H(2N)e^{-(1/8)\epsilon_2 N}$$

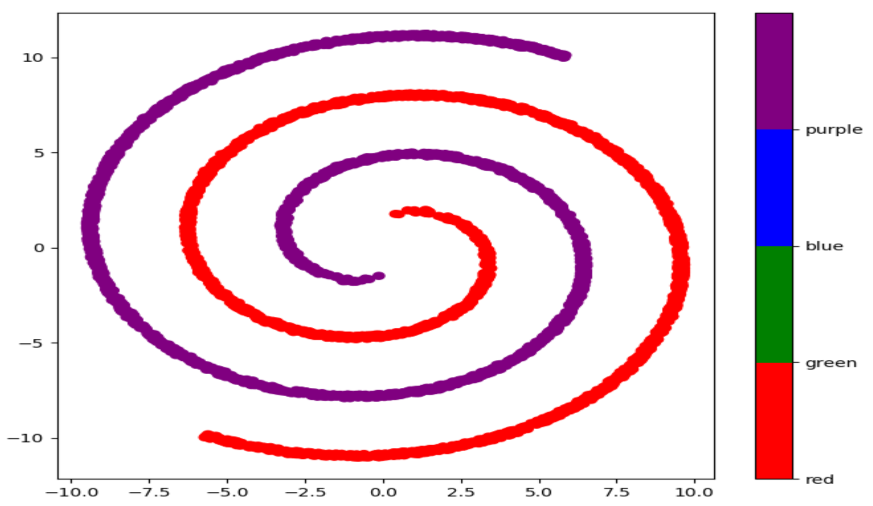
15.7 Datasets



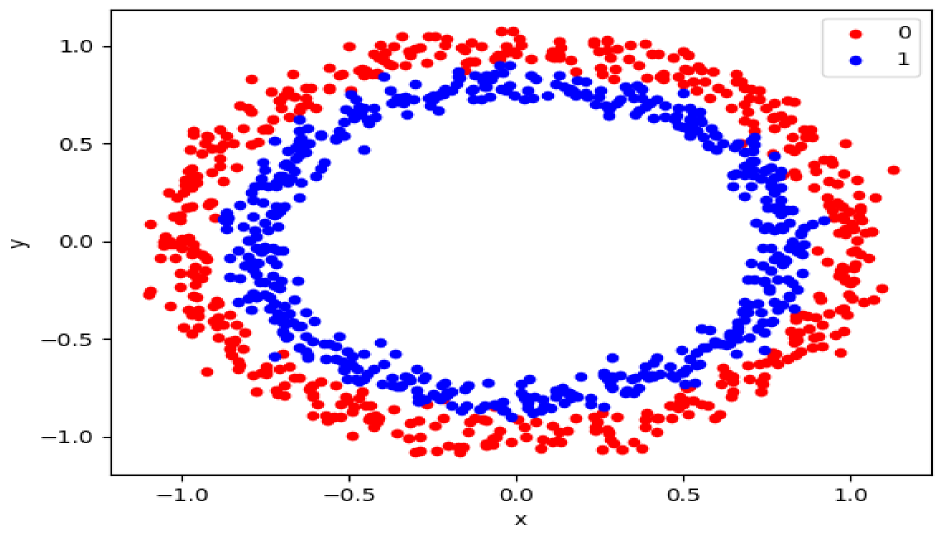
Corner Dataset



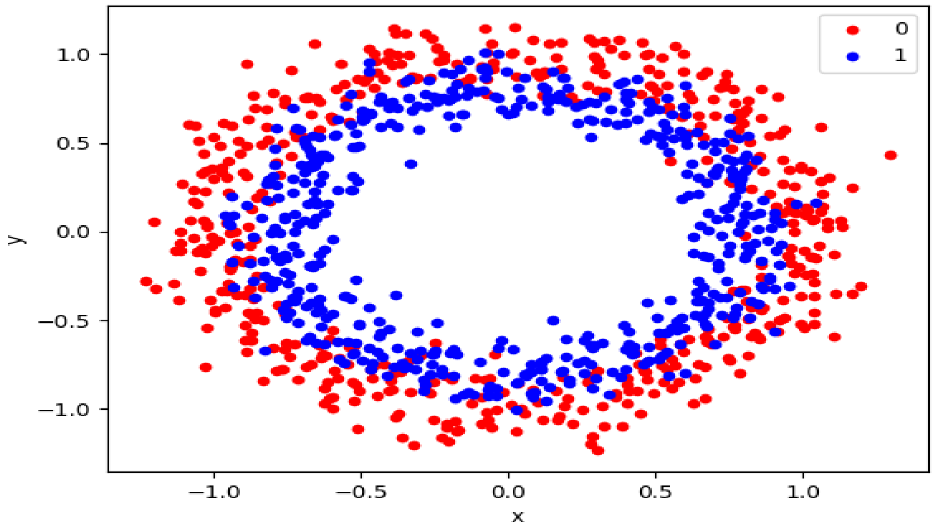
Double Moon Dataset



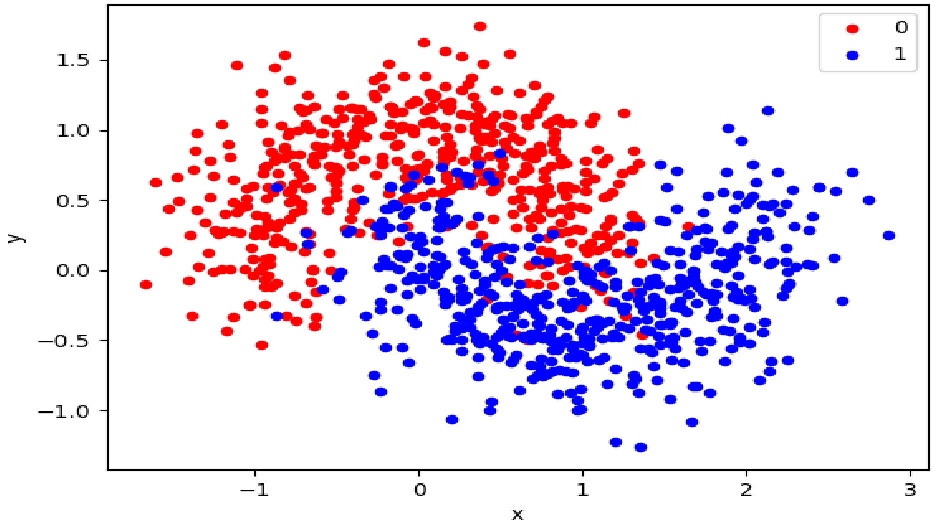
Two Spiral Dataset



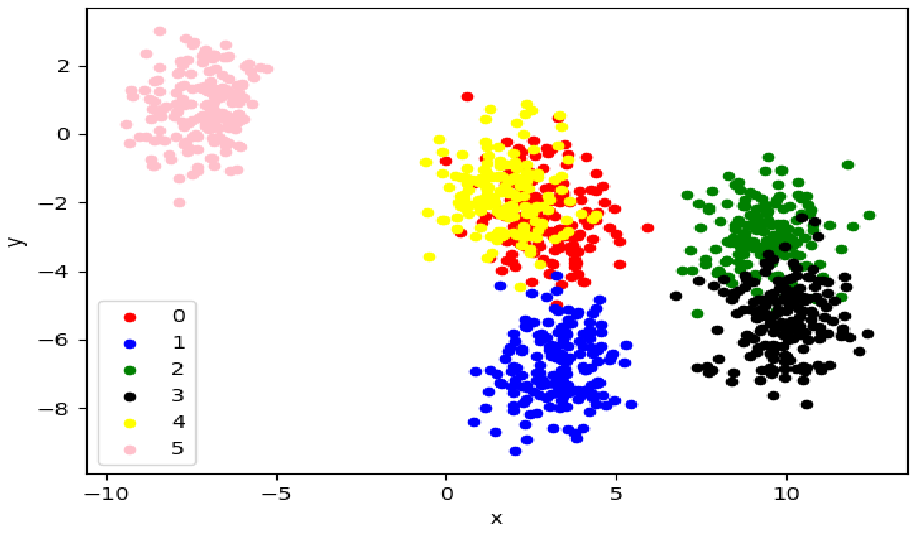
Circle Dataset



Circle2 Dataset



Double Moon 3 Dataset



Blob Dataset

References

[1] Thomas Hofmann, Bernhard Schölkopf and Alexander J. Smola [*KERNEL METHODS IN MACHINE LEARNING*]. The Annals of Statistics, 36(3):1171-1220, 2008.

[2] John Shawe-Taylor and Nello Cristianini. *Kernel Methods for Pattern Analysis*. Cambridge University Press, New York, United States of America, 2004.

[3] Pedregosa, F. and Varoquaux, G. and Gramfort, A. and Michel, V. and Thirion, B. and Grisel, O. and Blondel, M. and Prettenhofer, P. and Weiss, R. and Dubourg, V. and Vanderplas, J. and Passos, A. and Cournapeau, D. and Brucher, M. and Perrot, M. and Duchesnay, E. [*Scikit-learn: Machine Learning in Python*]. Journal of Machine Learning Research, 12:2825–2830, 2011.

[4] Mostafa Samir: Machine Learning Theory - Part 1,Part 2
`https://mostafa-samir.github.io/ml-theory-pt1/`

[5] Taku Kudo and Yuji Matsumoto [*Fast Methods for Kernel-based Text Analysis*]. Association for Computational Linguistics, 41:24-31, July 2003.