|  |  |
| --- | --- |
| Algorithm: SVM (Gaussian Kernel) | |
| USN : 1MS17CS143 | NAME : Sathvik K P |
| USN : 1MS17CS148 | NAME : Sathvik B |

**Description of the Algorithm:**

A support-vector machine constructs a hyperplane or set of hyperplanes in a high- or infinite-dimensional space, which can be used for classification, regression, or other tasks like outliers detection. Intuitively, a good separation is achieved by the hyperplane that has the largest distance to the nearest training-data point of any class (so-called functional margin), since in general the larger the margin, the lower the generalization error of the classifier.[4]

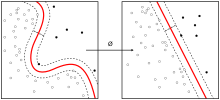
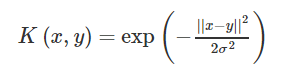
[](https://en.wikipedia.org/wiki/File:Kernel_Machine.svg)

Image of Kernel machine shown above.

Whereas the original problem may be stated in a finite-dimensional space, it often happens that the sets to discriminate are not linearly separable in that space. For this reason, it was proposed that the original finite-dimensional space be mapped into a much higher-dimensional space, presumably making the separation easier in that space. To keep the computational load reasonable, the mappings used by SVM schemes are designed to ensure that dot products of pairs of input data vectors may be computed easily in terms of the variables in the original space, by defining them in terms of a kernel function k(x,y) selected to suit the problem.

Gaussian Kernel:

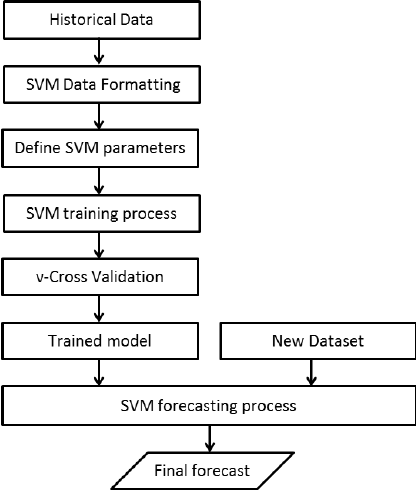
The Gaussian/RBF kernel is given as:



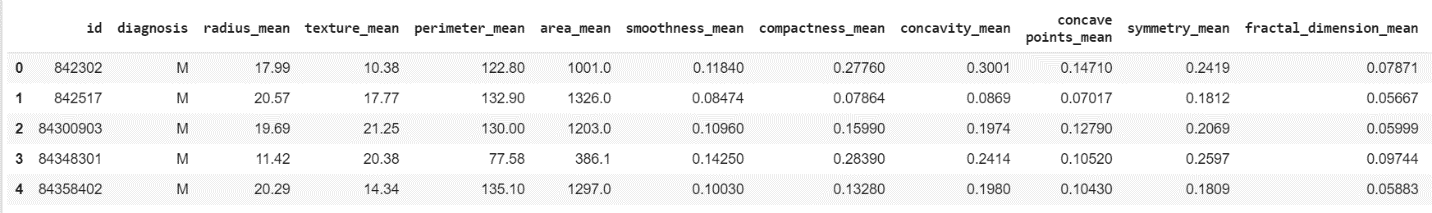
The Gaussian kernel computed with a support vector is an exponentially decaying function in the input feature space, the maximum value of which is attained at the support vector and which decays uniformly in all directions around the support vector, leading to hyper-spherical contours of the kernel function. The SVM classifier with the Gaussian kernel is simply a weighted linear combination of the kernel function computed between a data point and each of the support vectors. The role of a support vector in the classification of a data point is tempered with α, the global prediction usefulness of the support vector, and K(x,y), the local influence of a support vector in prediction at a particular data point.

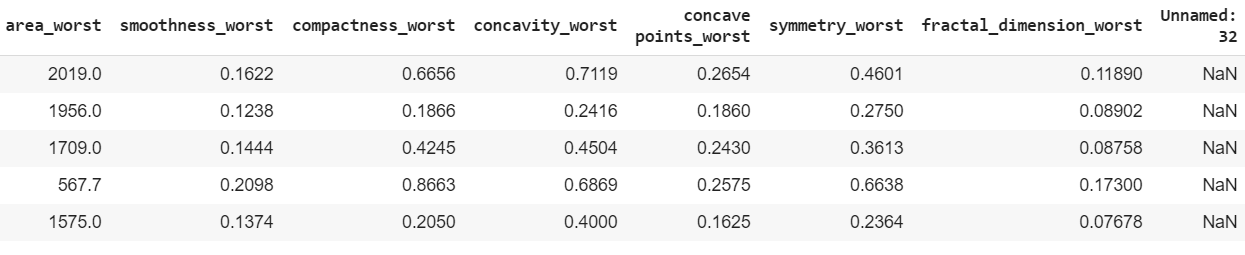
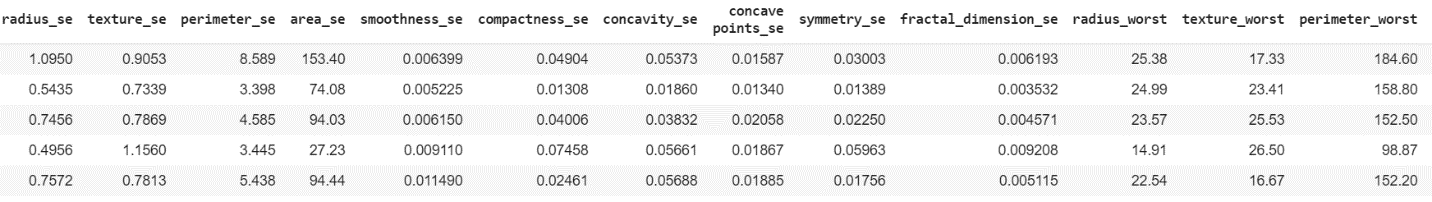
Gaussian kernels are universal kernels i.e. their use with appropriate regularization guarantees a globally optimal predictor which minimizes both the estimation and approximation errors of a classifier. Here, approximation error refers to the error incurred by limiting the space of classification models over which search is performed, and estimation error refers to error in estimation of the model parameters.

**Algorithm Pseudocode:**



**Data set Used: (Attach Screen shot of the few rows)**





**Challenges faced during the implementation of the program:**

1. Low accuracy without normalization
2. Fine tuning parameters
3. Dataset pre processing
4. Analyzing the result

**Code:**

import numpy as np # linear algebra

import pandas as pd # data processing, CSV file I/O (e.g. pd.read\_csv)

import seaborn as sns # data visualization library

import matplotlib.pyplot as plt

import time

from sklearn.preprocessing import StandardScaler

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import accuracy\_score

from sklearn.svm import SVC

from sklearn.metrics import confusion\_matrix

df.drop(['Unnamed: 32','id'],axis=1,inplace=True)

y=df.diagnosis

df['target'] = df['diagnosis'].apply(lambda x: 1 if x=='M' else 0)

df.drop(['diagnosis'],axis=1,inplace=True)

df.head()

corr\_matrix = df.corr()

corr\_matrix

scaler = StandardScaler()

df[['radius\_mean']] = scaler.fit\_transform(df[['radius\_mean']])

df[['texture\_mean']] = scaler.fit\_transform(df[['texture\_mean']])

df[['perimeter\_mean']] = scaler.fit\_transform(df[['perimeter\_mean']])

df[['area\_mean']] = scaler.fit\_transform(df[['area\_mean']])

df[['smoothness\_mean']] = scaler.fit\_transform(df[['smoothness\_mean']])

df[['compactness\_mean']] = scaler.fit\_transform(df[['compactness\_mean']])

df[['concavity\_mean']] = scaler.fit\_transform(df[['concavity\_mean']])

df[['concave points\_mean']] = scaler.fit\_transform(df[['concave points\_mean']])

df[['symmetry\_mean']] = scaler.fit\_transform(df[['symmetry\_mean']])

df[['fractal\_dimension\_mean']] = scaler.fit\_transform(df[['fractal\_dimension\_mean']])

df[['radius\_se']] = scaler.fit\_transform(df[['radius\_se']])

df[['texture\_se']] = scaler.fit\_transform(df[['texture\_se']])

df[['perimeter\_se']] = scaler.fit\_transform(df[['perimeter\_se']])

df[['area\_se']] = scaler.fit\_transform(df[['area\_se']])

df[['smoothness\_se']] = scaler.fit\_transform(df[['smoothness\_se']])

df[['compactness\_se']] = scaler.fit\_transform(df[['compactness\_se']])

df[['concavity\_se']] = scaler.fit\_transform(df[['concavity\_se']])

df[['concave points\_se']] = scaler.fit\_transform(df[['concave points\_se']])

df[['symmetry\_se']] = scaler.fit\_transform(df[['symmetry\_se']])

df[['fractal\_dimension\_se']] = scaler.fit\_transform(df[['fractal\_dimension\_se']])

df[['radius\_worst']] = scaler.fit\_transform(df[['radius\_worst']])

df[['texture\_worst']] = scaler.fit\_transform(df[['texture\_worst']])

df[['perimeter\_worst']] = scaler.fit\_transform(df[['radius\_mean']])

df[['area\_worst']] = scaler.fit\_transform(df[['area\_worst']])

df[['smoothness\_worst']] = scaler.fit\_transform(df[['smoothness\_worst']])

df[['compactness\_worst']] = scaler.fit\_transform(df[['compactness\_worst']])

df[['concavity\_worst']] = scaler.fit\_transform(df[['concavity\_worst']])

df[['concave points\_worst']] = scaler.fit\_transform(df[['concave points\_worst']])

df[['symmetry\_worst']] = scaler.fit\_transform(df[['symmetry\_worst']])

df[['fractal\_dimension\_worst']] = scaler.fit\_transform(df[['fractal\_dimension\_worst']])

y=df['target']

X=df.drop(['target'],axis=1)

X\_train,X\_test,y\_train,y\_test = train\_test\_split(X,y,test\_size=0.20,random\_state = 43)

svm = SVC(kernel="rbf", gamma="auto", C=1)

svm.fit(X\_train, y\_train)

pred\_vals = svm.predict(X\_test)

print(accuracy\_score(y\_test,pred\_vals))

cm = confusion\_matrix(y\_test, pred\_vals)

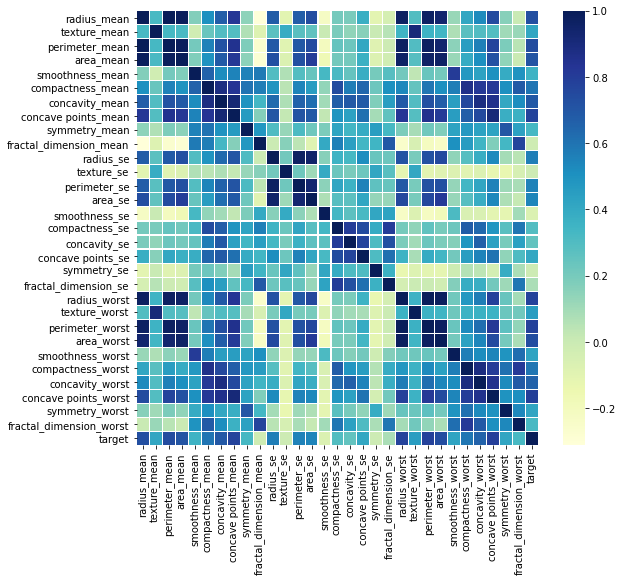
sns.heatmap(cm,annot=True)

**Output: (Screen shots)**

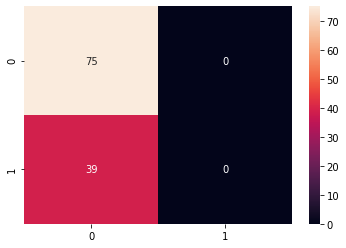
EDA:



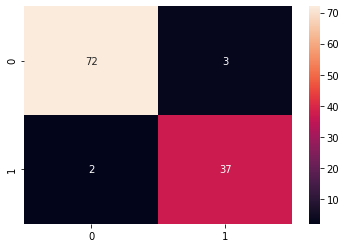
Correlation Matrix



Confusion Matrix without normalization



Confusion Matrix with normalization



As we can see, just by normalizing the data, we get a Huge increase in accuracy, from 65% to 95%. Thus the SVM classifier with a guassian kernel has fit the dataset well.

**References:**

1. <https://en.wikipedia.org/wiki/Support-vector_machine>
2. <https://www.analyticsvidhya.com/blog/2017/09/understaing-support-vector-machine-example-code/>
3. <http://cs229.stanford.edu/notes/cs229-notes3.pdf>
4. <https://alex.smola.org/teaching/pune2007/pune_4.pdf>
5. <https://www.quora.com/What-is-the-intuition-behind-Gaussian-kernel-in-SVM-How-can-I-visualize-the-transformation-function-%CF%95-that-corresponds-to-the-Gaussian-kernel-Why-is-the-Gaussian-kernel-popular>