1. <https://www.javatpoint.com/>
2. <https://www.geeksforgeeks.org>

* Also called Support Vector Networks
* Supervised learning
* For classification and regression (for regression called Support Vector Regression)
* Developed at [AT&T Bell Laboratories](https://en.wikipedia.org/wiki/AT%26T_Bell_Laboratories) by [Vladimir Vapnik](https://en.wikipedia.org/wiki/Vladimir_Vapnik) with colleagues (Boser et al., 1992, [Guyon](https://en.wikipedia.org/wiki/Isabelle_Guyon) et al., 1993, [Cortes](https://en.wikipedia.org/wiki/Corinna_Cortes) and Vapnik, 1995,[[2]](https://en.wikipedia.org/wiki/Support-vector_machine#cite_note-article1995-2) Vapnik et al., 1997
* Apply to linier and non linier data
* SVM is not Deep Learning (DL), however the SVM version L2-SVM(DLSVM) is recognized as SVM technique apply to DL. <https://arxiv.org/abs/1306.0239#:~:text=Recently%2C%20fully%2Dconnected%20and%20convolutional,natural%20language%20processing%2C%20and%20bioinformatics>.
* SVM can be applied using structured data and unstructured data such as text, voice or image.
* SVM has been applied in face detection, hand-writing recognition, text and hypertext categorization, image classification, bioinformatics, protein fold and remote homo-logy detection, Generalized predictive Control (GPC), Geo andEnvironment Science.
* Which kernel is best for SVM?

So, the rule of thumb is: **use linear SVMs (or logistic regression) for linear problems, and nonlinear kernels such as the Radial Basis Function kernel for non-linear problems**.

Two-class SVM, when data is linearly separable

A close-up of a diagram

Description automatically generated

Not linearly separable

A blue and green circles and red lines

Description automatically generated A graph of a function

Description automatically generated

A graph with a green line and red dots

Description automatically generated

The steps for non linear SVM

a. transform the original input data into a higher dimensional space

b. introduce slack variables

- need to minimize

A math equations with numbers

Description automatically generated

Subject to: A group of black letters

Description automatically generated

- 3 Approaches for SVM Multi class classification

a. one vs all or one vs rest

b. all pairs

c. Error Correcting output codes (ECOC)

one vs all

Scatter chart

Description automatically generated

All pairs

A picture containing text, map, traffic, light

Description automatically generated

Error Correcting output codes (ECOC)

A screenshot of a computer

Description automatically generated

Questions:

SVM 1:

Dear Students, please answer these questions:

* What are the differences between 2 classes SVM and multi-class SVM?
* what do you know regarding the sigmoid kernel?

SVM 2:

Dear Students, please answer these questions:

1. What does SVM mean as a deterministic algorithm ?
2. What do you know about Linear SVM and Non Linear SVM?

SVM Study Case:

Dear students please do this assignment based on the attached PDF file

2. ss

Also called Hyperbolic Tangent Kernel or Multilayer Perceptron (MLP) kernel.

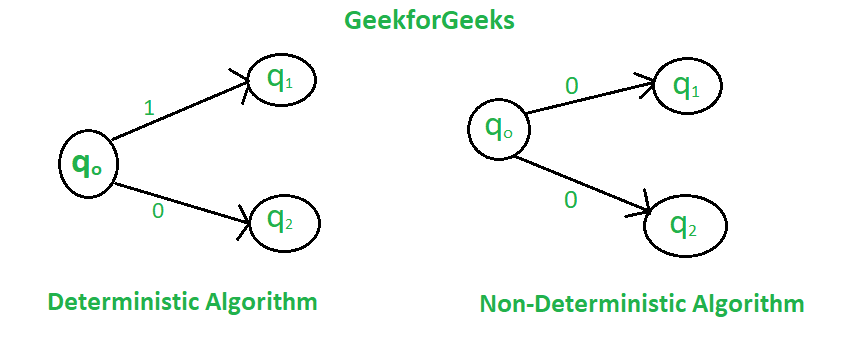
<http://crsouza.com/2010/03/17/kernel-functions-for-machine-learning-applications>

<https://www.geeksforgeeks.org/major-kernel-functions-in-support-vector-machine-svm/#:~:text=Sigmoid%20Kernel%3A%20this%20function%20is,activation%20function%20for%20artificial%20neurons>. (including phyton script)

Major Kernel Functions in SVM (**Gausian Kernel**, **Gaussian Kernel Radial Basis Function (RBF), Sigmoid Kernel, Polynomial Kernel, Linear Kernel )**

Sigmoid kernel formula: K(x,y)=tanh(y.xT y+r)

1. ss



<https://www.geeksforgeeks.org/difference-between-deterministic-and-non-deterministic-algorithms/>

**Simplest case: linearly-separable data, binary classification**

SVM algorithm can be used for **Face detection, image classification, text categorization,** etc.

Chart, scatter chart

Description automatically generated

Goal: want to find the hyperplane (i.e. decision boundary) linearly separating our classes. The boundary will have equation: wT x + b = 0.

Anything above the decision boundary should have label 1. i.e., xi s.t. wT xi + b > 0 will have corresponding yi = 1.

Similarly, anything below the decision boundary should have label −1. i.e., xi s.t. wT xi + b < 0 will have corresponding yi = −1.

The reason for this labelling scheme is that it lets us condense the formulation for the decision function to f(x) = sign(wT x + b) since f(x) = +1 for all x above the boundary, and f(x) = −1 for all x below the boundary.



1. The best hyperplane is that plane that has the maximum distance from both the classes, and this is the main aim of SVM.



1. S
2. Linear SVM

When the data is perfectly linearly separable only then we can use Linear SVM. Perfectly linearly separable means that the data points can be classified into 2 classes by using a single straight line(if 2D).

1. Non-Linear SVM

When the data is not linearly separable then we can use Non-Linear SVM, which means when the data points cannot be separated into 2 classes by using a straight line (if 2D) then we use some advanced techniques like kernel tricks to classify them. In most real-world applications we do not find linearly separable datapoints hence we use kernel trick to solve them.

* <https://www.analyticsvidhya.com/blog/2021/10/support-vector-machinessvm-a-complete-guide-for-beginners/>

## Examples of SVM Kernels

1. **Polynomial kernel** – it is mostly used in image processing.
2. **Linear Splines kernel in one-dimension –** it is used in text categorization and is helpful in dealing with large spare data vectors.
3. **Gaussian Kernel**– it is used when there is no preceding information about the data.
4. **Gaussian Radial Basis Function (RBF) –**It is commonly used where there is no previous knowledge about the data.
5. **Hyperbolic Tangent Kernel –** it is used in neural networks.
6. **Bessel Function of the First kind Kernel –** it is used to eliminate the cross term in mathematical functions.
7. **Sigmoid Kernel –** it can be utilized as the alternative for neural networks.
8. **ANOVA Radial Basis Kernel –** it is mostly used in regression problems.

**Activation function**

**A screenshot of a table

Description automatically generated**

**A graph of different colored lines

Description automatically generated**

**A table of numbers and symbols

Description automatically generated**

**A graph of different sizes and colors

Description automatically generated with medium confidence**

**A table of numbers with numbers on it

Description automatically generated A table of numbers and digits

Description automatically generated**

**SIgMoid**

**A graph with a line

Description automatically generated**

**Tanh**

**A graph with numbers and a line

Description automatically generated**

**ReLu**

**A graph with a line going up

Description automatically generated**

**Leaky Relu**

**A graph with numbers and a line

Description automatically generated**

Source:

[SVM | Support Vector Machine Algorithm in Machine Learning (analyticsvidhya.com)](https://www.analyticsvidhya.com/blog/2017/09/understaing-support-vector-machine-example-code/)

**Exercise SVM 1:**

* Using Iris dataset [UCI Machine Learning Repository: Iris Data Set](https://archive.ics.uci.edu/ml/datasets/iris)

Graphical user interface, application

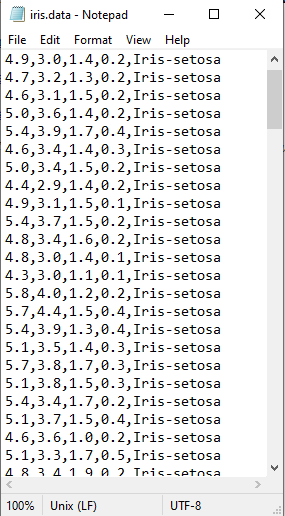
Description automatically generated

Download: data description

Graphical user interface, text

Description automatically generated

Download: dataset



* Script:

import numpy as np

import matplotlib.pyplot as plt

from sklearn import svm, datasets

# import some data to play with

iris = datasets.load\_iris()

X = iris.data[:, :2] # only take the first two features. We could

# avoid this ugly slicing by using a two-dim dataset

y = iris.target

#Create an instance of SVM and fit out data. We do not scale our

# data since we want to plot the support vectors

C = 1.0 #SVM regularization parameter

svc = svm.SVC(kernel='linear', C=1,gamma=0).fit(X, y)

# create a mesh to plot in

x\_min, x\_max = X[:, 0].min() - 1, X[:, 0].max() + 1

y\_min, y\_max = X[:, 1].min() - 1, X[:, 1].max() + 1

h = (x\_max / x\_min)/100

xx, yy = np.meshgrid(np.arange(x\_min, x\_max, h),

np.arange(y\_min, y\_max, h))

plt.subplot(1, 1, 1)

Z = svc.predict(np.c\_[xx.ravel(), yy.ravel()])

Z = Z.reshape(xx.shape)

plt.contourf(xx, yy, Z, cmap=plt.cm.Paired, alpha=0.8)

plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.Paired)

plt.xlabel('Sepal length')

plt.ylabel('Sepal width')

plt.xlim(xx.min(), xx.max())

plt.title('SVC with linear kernel')

plt.show()

Please running the script and see the result!

Script 2:

Exchange the kernel into RBF (Gaussian) Kernel

svc = svm.SVC(kernel='rbf', C=1,gamma=0).fit(X, y)

Please running the script and see the result!

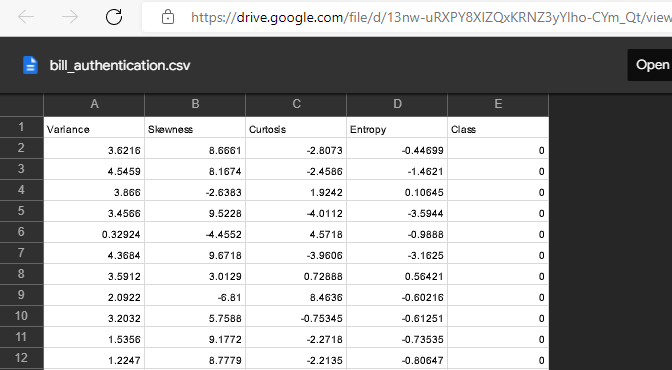
Please running the script with changing score gamma and C and print the result:

1. svc = svm.SVC(kernel='rbf', C=1,gamma=10).fit(X, y)
2. svc = svm.SVC(kernel='rbf', C=1,gamma=100).fit(X, y)
3. svc = svm.SVC(kernel='rbf', C=100,gamma=0).fit(X, y)
4. svc = svm.SVC(kernel='rbf', C=1000,gamma=0).fit(X, y)

**Exercise 2:**

Source: [Implementing SVM and Kernel SVM with Python's Scikit-Learn (stackabuse.com)](https://stackabuse.com/implementing-svm-and-kernel-svm-with-pythons-scikit-learn/)

* implementation with Python scikit learn
* dataset: banknote authentication uci machine learning
* The data is available for download at the following link:<https://drive.google.com/file/d/13nw-uRXPY8XIZQxKRNZ3yYlho-CYm_Qt/view>



* The detailed information about the data is available at the following link: <https://archive.ics.uci.edu/ml/datasets/banknote+authentication>
* Graphical user interface, application, table

  Description automatically generated
* **Attribute Information:**
* 1. variance of Wavelet Transformed image (continuous)  
  2. skewness of Wavelet Transformed image (continuous)  
  3. curtosis of Wavelet Transformed image (continuous)  
  4. entropy of image (continuous)  
  5. class (integer)

Script:

#import Scikit-learn library

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

%matplotlib inline

#To read data from CSV file

bankdata = pd.read\_csv("D:/Datasets/bill\_authentication.csv")

#to see the rows and columns and of the data

#In the output you will see (1372,5). This means that the bank note dataset has 1372 rows and 5 columns.

bankdata.shape

#print dataset

bankdata.head()

#Data Preprocessing (1) Dividing data into attributes and labels

#all the columns of the bankdata dataframe are being stored in the X variable except the "Class" column, which is the label column.

#The drop() method to drop the column

X = bankdata.drop('Class', axis=1)

#Data Preprocessing (2) dividing the data into training and testing sets

#only the class column is being stored in the y variable.

y = bankdata['Class']

#X variable contains attributes while y variable contains corresponding labels.

#divide data into training and test sets.

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

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

#SVM classifier

from sklearn.svm import SVC

svclassifier = SVC(kernel='linear')

svclassifier.fit(X\_train, y\_train)

#to make prediction

y\_pred = svclassifier.predict(X\_test)

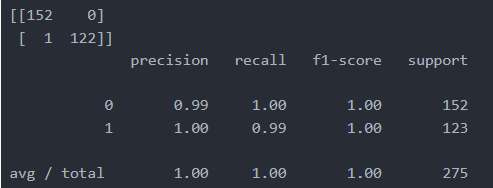
#Print confusion matrix (evaluating)

from sklearn.metrics import classification\_report, confusion\_matrix

print(confusion\_matrix(y\_test,y\_pred))

print(classification\_report(y\_test,y\_pred))

Running result:



TP=152, TN=122, FP=0, FN=1

**Implementation with iris dataset from UCI machine learning**

#import library

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

#import dataset

url = "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data"

# Assign colum names to the dataset

colnames = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width', 'Class']

# Read dataset to pandas dataframe

irisdata = pd.read\_csv(url, names=colnames)

#Preprocessing

X = irisdata.drop('Class', axis=1)

y = irisdata['Class']

#divide data into training and test sets.

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

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

#SVM classifier with polynomial kernel

from sklearn.svm import SVC

svclassifier = SVC(kernel='poly', degree=8)

svclassifier.fit(X\_train, y\_train)

#Making Predictions

y\_pred = svclassifier.predict(X\_test)

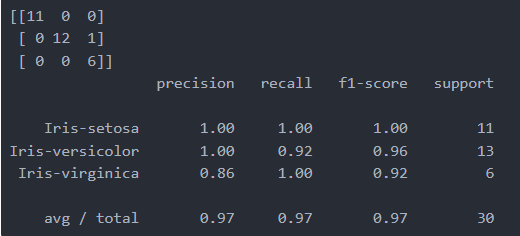
#Evaluating the Algorithm

from sklearn.metrics import classification\_report, confusion\_matrix

print(confusion\_matrix(y\_test, y\_pred))

print(classification\_report(y\_test, y\_pred))

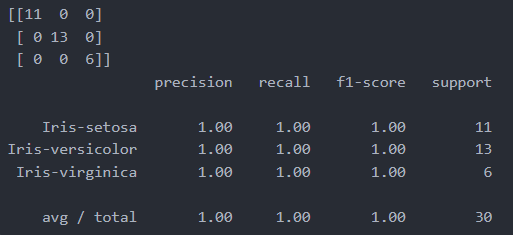
running result:



**If using Gaussian (RBF) kernel**

**svclassifier = SVC(kernel='rbf')**

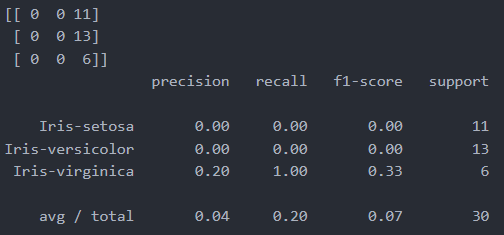
running result:



**If using Sigmoid kernel**

**svclassifier = SVC(kernel='sigmoid')**

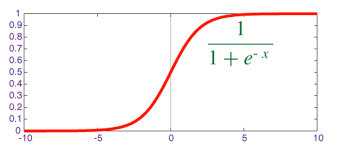
running result:



**Conclusion:**

* The worst is sigmoid kernel due to sigmoid only return 0 and 1
* Gaussian (RBF) kernel has 100% prediction rate
* Polynomial kernel has 97% prediction rate which is misclassified in class iris-virginica with 86% prediction rate.
* Usually SVM is better than Decision tree

**Sigmoid kernel**



**RBF kernel**

[Radial Basis Function (RBF) Kernel: The Go-To Kernel | by Sushanth Sreenivasa | Towards Data Science](https://towardsdatascience.com/radial-basis-function-rbf-kernel-the-go-to-kernel-acf0d22c798a)

(also called the quadratic exponential kernel, the squared exponential kernel or the Gaussian kernel)



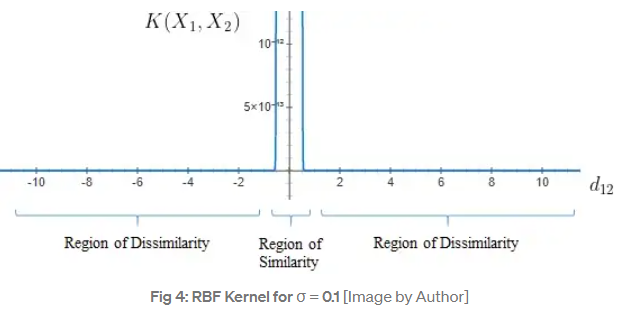
where,  
1. ‘σ’ is the variance and our hyperparameter  
2. ||X₁ - X₂|| is the Euclidean (L₂-norm) Distance between two points X₁ and X₂

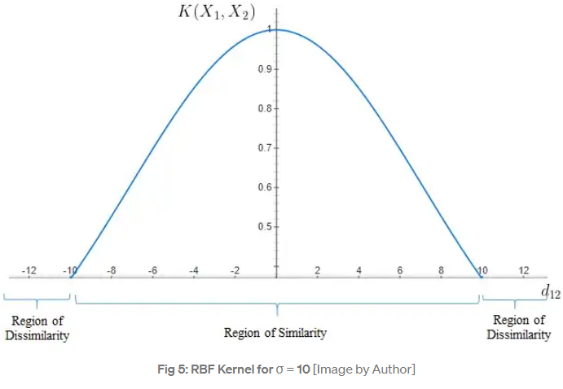
3. ||X₁ - X₂||2 is Squared Euclidean Distance

Chart

Description automatically generated with low confidence

**Fig 4: RBF Kernel for σ = 1**

****

****

**Polynomial kernel**

K(x,y)= (xTy + c)d

x,y are vectors in input space, where c is constant and when c=0 the kernel is called homogeneous

The parameter **c** can be used to control the trade-off between the fit of the training data and the size of the margin. A large **c** value will give a low training error but may result in overfitting. A small **c** value will give a high training error but may result in underfitting.

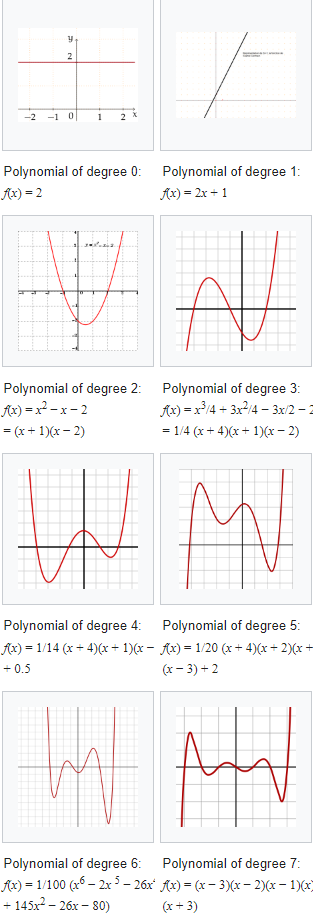
The degree **d** of the polynomial can be used to control the complexity of the model. A high degree **d** will result in a more complex model that may overfit the data, while a low degree **d** will

result in a simpler model that may underfit the data.

[SVM Kernels: Polynomial Kernel - From Scratch Using Python. - PyCodeMates](https://www.pycodemates.com/2022/10/svm-kernels-polynomial-kernel.html)

[(10) (PDF) A User’s Guide to Support Vector Machines (researchgate.net)](https://www.researchgate.net/publication/41896604_A_User%27s_Guide_to_Support_Vector_Machines/figures?lo=1)

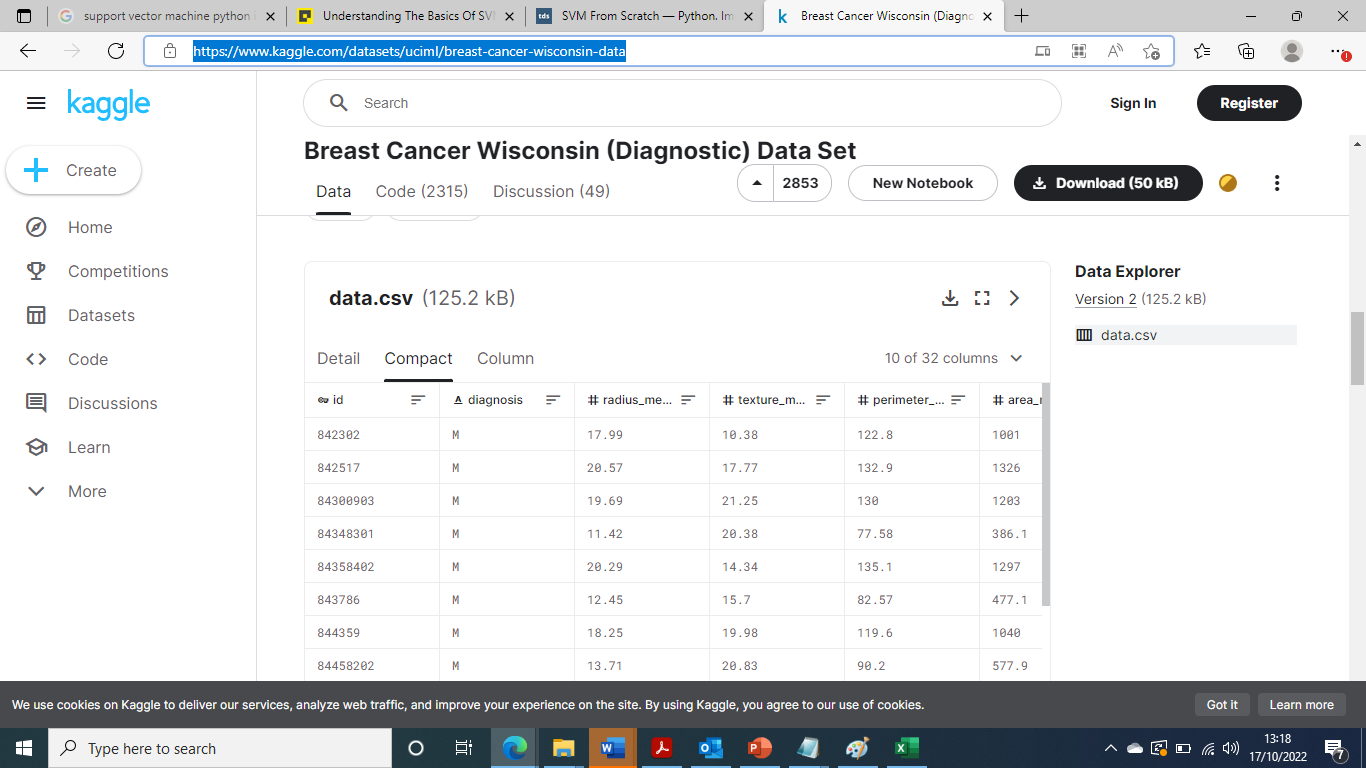
[Polynomial - Wikipedia](https://en.wikipedia.org/wiki/Polynomial)



**Exercise 3 (SVM from scratch)**

Source: [SVM From Scratch — Python. Important Concepts Summarized | by Qandeel Abbassi | Towards Data Science](https://towardsdatascience.com/svm-implementation-from-scratch-python-2db2fc52e5c2)

* Using breast cancer dataset UCI machine learning from [Breast Cancer Wisconsin (Diagnostic) Data Set | Kaggle](https://www.kaggle.com/datasets/uciml/breast-cancer-wisconsin-data)



* Apply normalization

is one of the many feature engineering techniques which is the process of converting a range of values, into a standard range of values, typically in the interval [−1, 1] or [0, 1]. It’s not a strict requirement but it improves the speed of learning (e.g. faster convergence in gradient descent) and prevents numerical overflow. Add following code in init() functions to normalize all of your features:

* Train model using Stochastic Gradient Descent (SGD) in order to minimize or maximise the cost function, where in SVM to minimize the cost

Script SGD:

**def sgd(features, outputs):**max\_epochs = 5000  
weights = np.zeros(features.shape[1])  
# stochastic gradient descent  
for epoch in range(1, max\_epochs):   
# shuffle to prevent repeating update cycles  
X, Y = shuffle(features, outputs)  
for ind, x in enumerate(X):  
ascent = calculate\_cost\_gradient(weights, x, Y[ind])  
weights = weights - (learning\_rate \* ascent)  
  
return weights

**Exercise 4 : visualisation SVM**

Source: [Understanding The Basics Of SVM With Example And Python Implementation (analyticsindiamag.com)](https://analyticsindiamag.com/understanding-the-basics-of-svm-with-example-and-python-implementation/)

#import dataset

import pandas as pd

data = pd.read\_csv("apples\_and\_oranges.csv")

#Splitting the dataset into training and test samples

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

training\_set, test\_set = train\_test\_split(data, test\_size = 0.2, random\_state = 1)

#Classifying the predictors and target

X\_train = training\_set.iloc[:,0:2].values

Y\_train = training\_set.iloc[:,2].values

X\_test = test\_set.iloc[:,0:2].values

Y\_test = test\_set.iloc[:,2].values

#Initializing SVM and fitting the training data

from sklearn.svm import SVC

classifier = SVC(kernel='rbf', random\_state = 1)

classifier.fit(X\_train,Y\_train)

#Predicting the classes for test set

Y\_pred = classifier.predict(X\_test)

#Attaching the predictions to test set for comparing

test\_set["Predictions"] = Y\_pred

#Calculating the accuracy of the predictions

from sklearn.metrics import confusion\_matrix

cm = confusion\_matrix(Y\_test,Y\_pred)

accuracy = float(cm.diagonal().sum())/len(Y\_test)

print("\nAccuracy Of SVM For The Given Dataset : ", accuracy)

#Visualizing the classifier

#Before we visualize we might need to encode the classes ‘apple’ and ‘orange’ into numericals.We can achieve that using the label encoder.

from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()

Y\_train = le.fit\_transform(Y\_train)

#After encoding , fit the encoded data to the SVM

from sklearn.svm import SVC

classifier = SVC(kernel='rbf', random\_state = 1)

classifier.fit(X\_train,Y\_train)

#Let’s Visualize!

import numpy as np

import matplotlib.pyplot as plt

from matplotlib.colors import ListedColormap

plt.figure(figsize = (7,7))

X\_set, y\_set = X\_train, Y\_train

X1, X2 = np.meshgrid(np.arange(start = X\_set[:, 0].min() - 1, stop = X\_set[:, 0].max() + 1, step = 0.01), np.arange(start = X\_set[:, 1].min() - 1, stop = X\_set[:, 1].max() + 1, step = 0.01))

plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape), alpha = 0.75, cmap = ListedColormap(('black', 'white')))

plt.xlim(X1.min(), X1.max())

plt.ylim(X2.min(), X2.max())

for i, j in enumerate(np.unique(y\_set)):

plt.scatter(X\_set[y\_set == j, 0], X\_set[y\_set == j, 1], c = ListedColormap(('red', 'orange'))(i), label = j)

plt.title('Apples Vs Oranges')

plt.xlabel('Weight In Grams')

plt.ylabel('Size in cm')

plt.legend()

plt.show()

Chart

Description automatically generated

Example image classification using SVM

[In-Depth: Support Vector Machines | Python Data Science Handbook (jakevdp.github.io)](https://jakevdp.github.io/PythonDataScienceHandbook/05.07-support-vector-machines.html)

## **Example: Face Recognition**

As an example of support vector machines in action, let's take a look at the facial recognition problem. We will use the Labeled Faces in the Wild dataset, which consists of several thousand collated photos of various public figures. A fetcher for the dataset is built into Scikit-Learn:

In [18]:

**from** **sklearn.datasets** **import** fetch\_lfw\_people

faces = fetch\_lfw\_people(min\_faces\_per\_person=60)

print(faces.target\_names)

print(faces.images.shape)

['Ariel Sharon' 'Colin Powell' 'Donald Rumsfeld' 'George W Bush'

'Gerhard Schroeder' 'Hugo Chavez' 'Junichiro Koizumi' 'Tony Blair']

(1348, 62, 47)

Let's plot a few of these faces to see what we're working with:

In [19]:

fig, ax = plt.subplots(3, 5)

**for** i, axi **in** enumerate(ax.flat):

axi.imshow(faces.images[i], cmap='bone')

axi.set(xticks=[], yticks=[],

xlabel=faces.target\_names[faces.target[i]])

