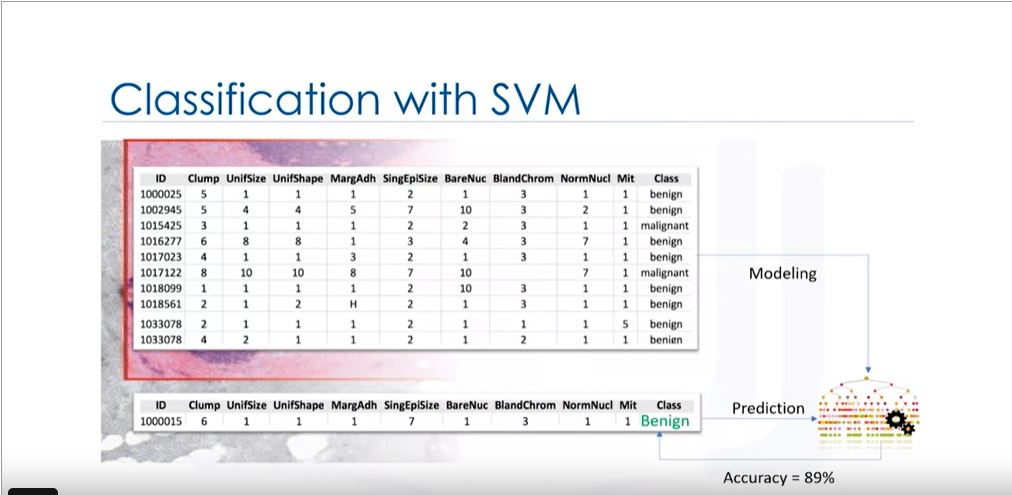
**Support vector machine (SVM):**

Let’s suppose you have the dataset of the patients who are suffering from the cancer . The dataset contains the cell information with different cell characteristics and two class of cells i.e. Benign and malignant. Now we can use this cell characteristics to give the early indication whether the cell is benign or malignant. For that we have to train the dataset and then model the SVMs to pass the test set and predict the characteristics of the cell.



Definition:

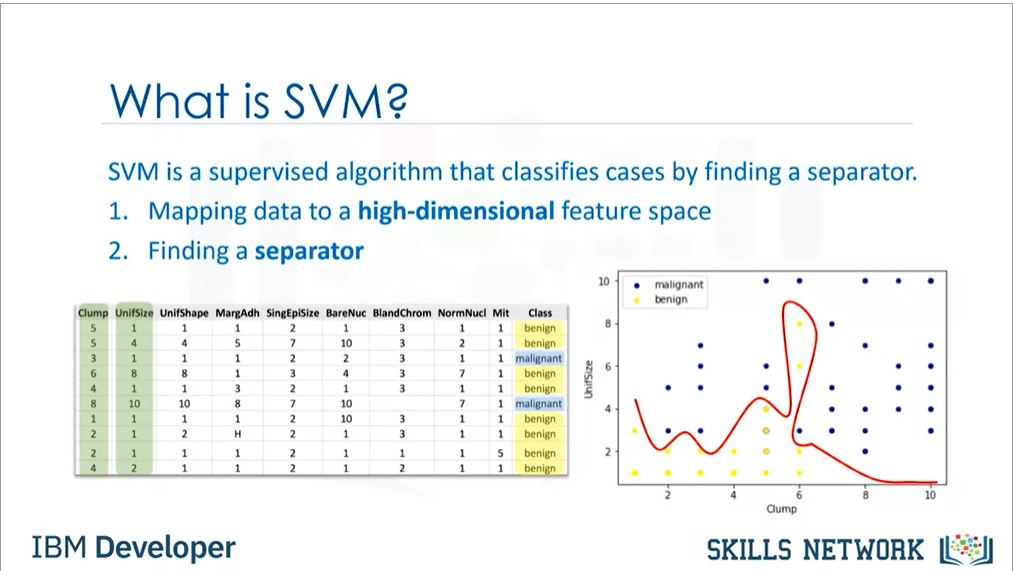
SVM is a supervised algorithm that classifies the cases by finding the separator.

How it works?:

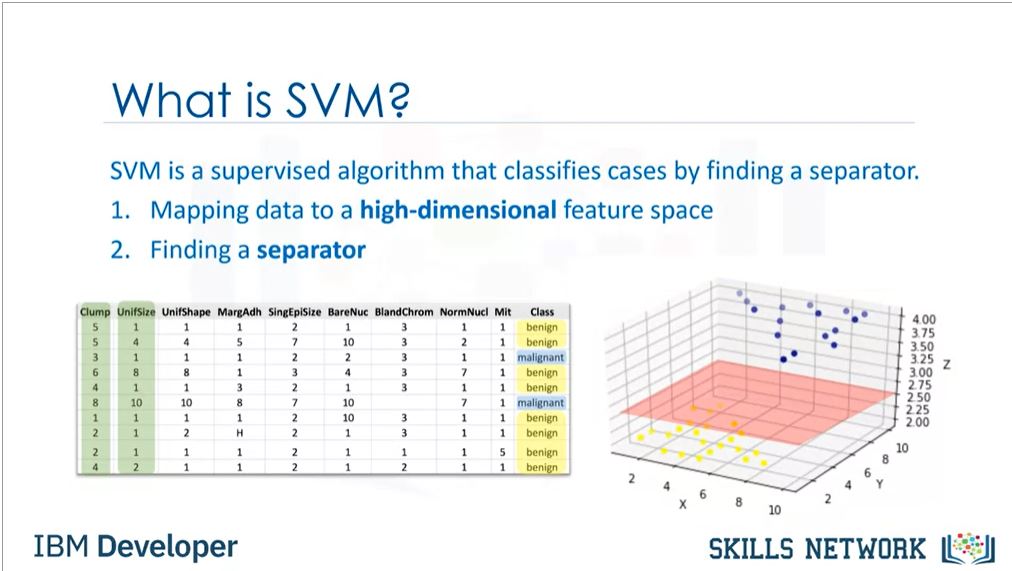
1. Mapping the data to the high dimensional feature space.
2. Finding the separator.

The data should be separated in such a way that the separator can be drawn as a hyper plane.

For example we have distribution of benign and malignant cells with feature set Clump and uniSize.

But as we plot the graph we can conclude that this is linearly non separable data set i.e. we cannot draw a linear line to separate the benign and malignant cells. 

Hence in most cases of real life dataset we have non linearable dataset.So we can transfer the data to the higher dimensional space. From that example we can convert 2d space to 3d space. By doing so we can classify the separator as hyper plane as we can separate the malignant and benign cells by the linear separator.

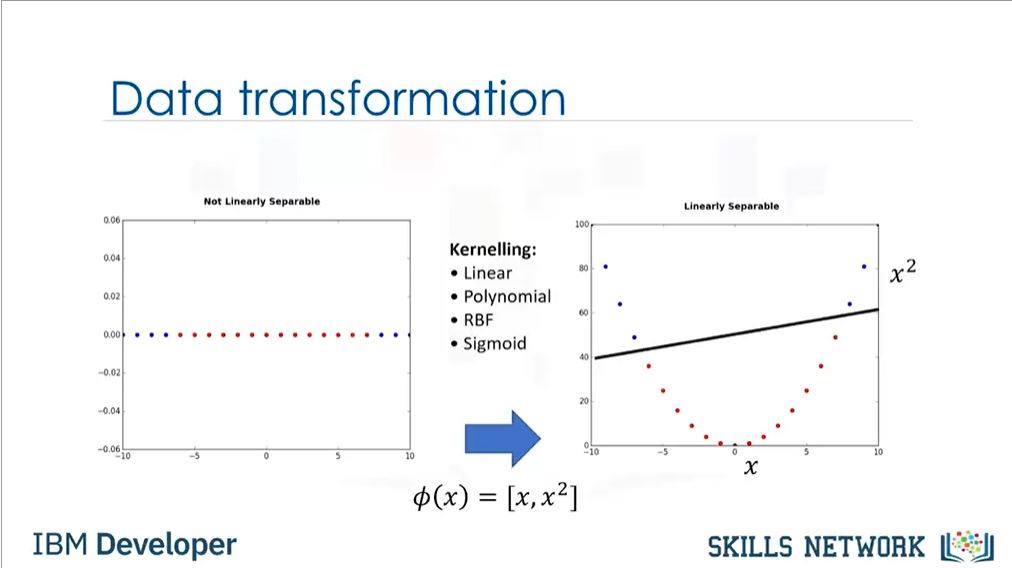


Therefore SVM algorithm outputs an optimal hyper plane that categorizes new examples.

Now we will see that how the data transformation works. Consider a example of non linearly separable date set in 1d space . Now to transform this data we will use a function to convert it into 2d space such that output is [x,x^2].

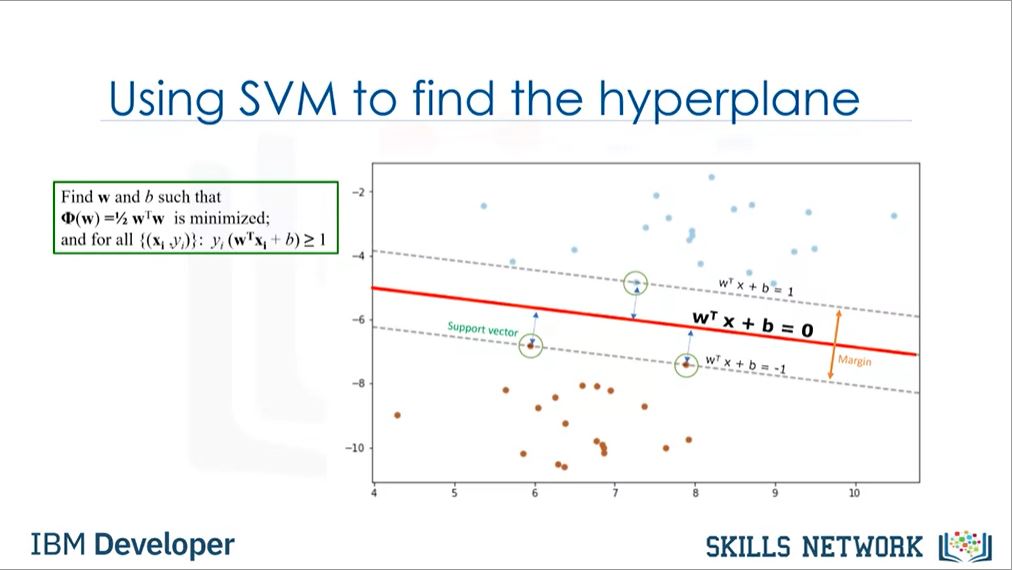
But doing so we can convert this data into linearly separable form.

Hence we can draw a line two separate the two characteristic output. Hence we can use the line to predict the test case output of new cases. Hence the mapping of data into higher dimensional space is know as **“Kernelling”**. Its types are Linear,polynomial, Radial basis function RBF, Sigmoid etc.

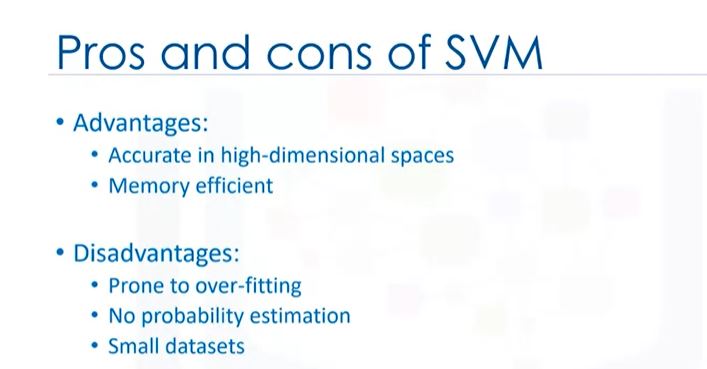


Now how to do we find the right or optimized separator after transformation of the data?

So basic idea of separator is to make a hyperplane which can divide the two class of sets for example benign and malignant.

Now consider 2d space . In that the hyper plane is generally a line . Hence this line segregates the two classes . The points above line represent a class and below the separator line represents another class. 

So hyperplane is to made in such a way that the margin i.e. the space between the line and closest examples should be more. Now we here consider the support vectors i.e. the nearest example to the hyper plane. And hence other examples can be ignored. Thus this algorithm is memory efficient.



**Coding:**

#import the libraries

import pandas as pd

import pylab as pl

import numpy as np

import scipy.optimize as opt

from sklearn import preprocessing

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

%matplotlib inline

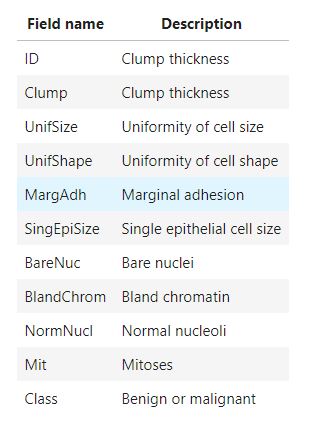
import matplotlib.pyplot as plt

#Load the data

!wget -O cell\_samples.csv https://s3-api.us-geo.objectstorage.softlayer.net/cf-courses-data/CognitiveClass/ML0101ENv3/labs/cell\_samples.csv

cell\_df = pd.read\_csv("cell\_samples.csv")

cell\_df.head()

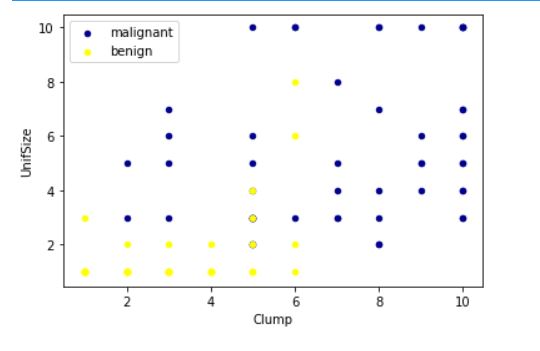


#Plot the graph

ax = cell\_df[cell\_df['Class'] == 4][0:50].plot(kind='scatter', x='Clump', y='UnifSize', color='DarkBlue', label='malignant');

cell\_df[cell\_df['Class'] == 2][0:50].plot(kind='scatter', x='Clump', y='UnifSize', color='Yellow', label='benign', ax=ax);

plt.show()



#Change data type

cell\_df = cell\_df[pd.to\_numeric(cell\_df['BareNuc'], errors='coerce').notnull()]

cell\_df['BareNuc'] = cell\_df['BareNuc'].astype('int')

cell\_df.dtypes

# Get X and Y set

feature\_df = cell\_df[['Clump', 'UnifSize', 'UnifShape', 'MargAdh', 'SingEpiSize', 'BareNuc', 'BlandChrom', 'NormNucl', 'Mit']]

X = np.asarray(feature\_df)

X[0:5]

cell\_df['Class'] = cell\_df['Class'].astype('int')

y = np.asarray(cell\_df['Class'])

y [0:5]

#Train test split

X\_train, X\_test, y\_train, y\_test = train\_test\_split( X, y, test\_size=0.2, random\_state=4)

print ('Train set:', X\_train.shape, y\_train.shape)

print ('Test set:', X\_test.shape, y\_test.shape)

#The SVM algorithm offers a choice of kernel functions for performing its processing. Basically, mapping data into a higher dimensional space is called kernelling. The mathematical function used for the transformation is known as the kernel function, and can be of different types, such as:

1.Linear

2.Polynomial

3.Radial basis function (RBF)

4.Sigmoid

Each of these functions has its characteristics, its pros and cons, and its equation, but as there's no easy way of knowing which function performs best with any given dataset, we usually choose different functions in turn and compare the results. Let's just use the default, RBF (Radial Basis Function) for this lab.

#Modelling

from sklearn import svm

clf = svm.SVC(kernel='rbf')

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

yhat = clf.predict(X\_test)

yhat [0:5]

**Evaluation**

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

import itertools

def plot\_confusion\_matrix(cm, classes,

normalize=False,

title='Confusion matrix',

cmap=plt.cm.Blues):

"""

This function prints and plots the confusion matrix.

Normalization can be applied by setting `normalize=True`.

"""

if normalize:

cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]

print("Normalized confusion matrix")

else:

print('Confusion matrix, without normalization')

print(cm)

plt.imshow(cm, interpolation='nearest', cmap=cmap)

plt.title(title)

plt.colorbar()

tick\_marks = np.arange(len(classes))

plt.xticks(tick\_marks, classes, rotation=45)

plt.yticks(tick\_marks, classes)

fmt = '.2f' if normalize else 'd'

thresh = cm.max() / 2.

for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):

plt.text(j, i, format(cm[i, j], fmt),

horizontalalignment="center",

color="white" if cm[i, j] > thresh else "black")

plt.tight\_layout()

plt.ylabel('True label')

plt.xlabel('Predicted label')

# Compute confusion matrix

cnf\_matrix = confusion\_matrix(y\_test, yhat, labels=[2,4])

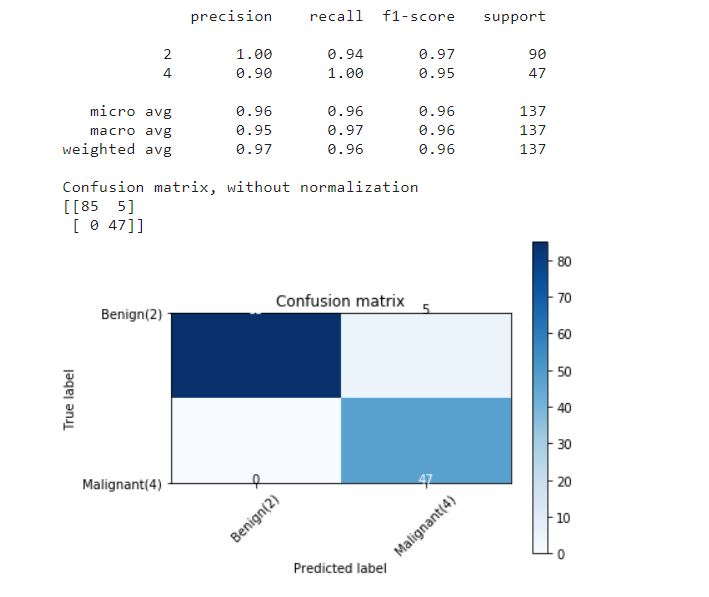
np.set\_printoptions(precision=2)

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

# Plot non-normalized confusion matrix

plt.figure()

plot\_confusion\_matrix(cnf\_matrix, classes=['Benign(2)','Malignant(4)'],normalize= False, title='Confusion matrix')



#You can also easily use the **f1\_score** from sklearn library:

from sklearn.metrics import f1\_score

f1\_score(y\_test, yhat, average='weighted')



#Lets try jaccard index for accuracy:

from sklearn.metrics import jaccard\_similarity\_score

jaccard\_similarity\_score(y\_test, yhat)

