**PATTERN RECOGNITION**

**16CS441**

**LAB MANUAL**

|  |
| --- |
|  |
|  |
|  |

**IV B.Tech (CSE) I Semester**

**A.Y:2019-20 Odd SEMESTER**

**Department of Computer Science and Engineering**

****

**Vadlamudi, AP, India. 522 213**

**Name of the Faculty: Jyostna Devi Bodapati,**

**Asst. Professor, Department of CSE**

**PATTERN RECOGNITION (16CS441)**

**LIST of Lab Programs as per lab cycle**

Develop applications to implement the following:

1. Read student data and classify the students into regular and irregular students.
2. Classify the data using minimum distance classifier.
3. Classify the data using K-NN classifier
4. Classify objects using linear discriminant functions.
5. Cluster given data using K-means clustering algorithm
6. Apply dimension reduction using PCA, KPCA, LDA, Fisher’s LDA and perform comparative study
7. Classify using SVM, FCM and perform comparative study.

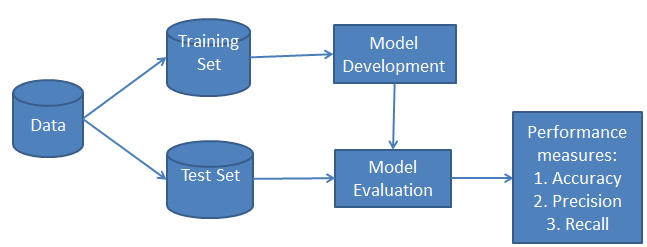
**INTRODUCTION**

**Classification:**

Classification is an important topic in statistics and machine learning. The purpose of a classifier is to assign a class to an unknown sample. Output is one of the possible values in a set of discrete value.

**Applications:** Identify gender of a given person,

The classification has two phases, a learning phase, and the evaluation phase. In the learning phase, classifier trains its model on a given dataset and in the evaluation phase, it tests the classifier performance. Performance is evaluated on the basis of various parameters such as accuracy, error, precision, and recall.



**Categorize classification algorithms based on the number of classes:**

* binary classification problem: Possibility of having only 2 classes
* multi-class classification: More than 2 classes
* Multi-label classification: A sample may belong to more than one class.

**In classification problems, we use two types of algorithms (dependent on the kind of output it creates):**

* Class output : Algorithms like SVM and KNN create a class output. For instance, in a binary classification problem, the outputs will be either 0 or 1
* Probability output : Algorithms like Logistic Regression, Random Forest, Gradient Boosting, Adaboost etc. give probability outputs. Converting probability outputs to class output is just a matter of creating a threshold probability.

**Categorize classification based on estimating parameters for each class:**

* **Parametric classifiers:** A classifier is a parametric classifier, meaning that it constructs a model for each class that is used in the training.

**Ex:** MLP

* **Non-parametric classifiers:** A classifier is a non-parametric classifier, meaning that it does not construct a model for the classes it tries to distinguish. The model remembers the whole training set and during classification the instance is classified online.

**Ex:** KNN,

**Evaluation metrics for classification:** different kinds of metrics are available to evaluate our models. The choice of metric completely depends on the type of model and the implementation plan of the model.

To evaluate any technique we generally look at 3 important aspects:

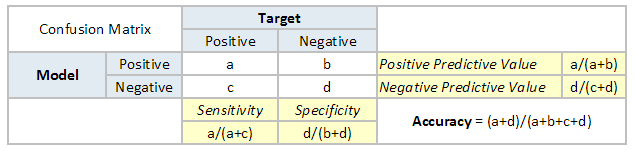
1. Ease to interpret output

2. Calculation time

3. Predictive Power

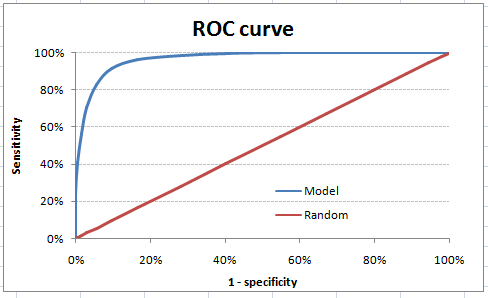
**Confusion Matrix:** Confusion matrix is a table which describes the performance of a prediction model. It is an N X N matrix, where N is the number of classes being predicted. Following are different evaluation metrics that can be computed based on the confusion matrix.

* Accuracy : the proportion of the total number of predictions that were correct.
* Positive Predictive Value or Precision : the proportion of positive cases that were correctly identified.
* Negative Predictive Value : the proportion of negative cases that were correctly identified.
* Sensitivity or Recall : the proportion of actual positive cases which are correctly identified.
* Specificity : the proportion of actual negative cases which are correctly identified.

**[](https://www.analyticsvidhya.com/blog/wp-content/uploads/2015/01/Confusion_matrix.png)**

**Area Under the ROC curve (AUC – ROC):** When we need to check or visualize the performance of the classification problem, we use AUC (Area Under The Curve) ROC (Receiver Operating Characteristics) curve. It tells how much the model is capable of distinguishing between classes.  Higher the AUC, better the model is at distinguishing between classes.

The ROC curve is the plot between sensitivity and (1- specificity). (1- specificity) is also known as false positive rate and sensitivity is also known as True Positive rate. Following is the ROC curve for the case in hand.

**[](https://www.analyticsvidhya.com/blog/wp-content/uploads/2015/01/ROC.png)**

**Gini Coefficient:** Gini coefficient is sometimes used in classification problems. Gini coefficient can be straigh away derived from the AUC ROC number. Gini is nothing but ratio between area between the ROC curve and the diagnol line & the area of the above triangle. Following is the formulae used :

Gini = 2\*AUC – 1

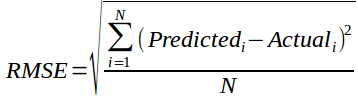
Gini above 60% is a good model.

**Regression:**

Regression determines the strength of the relationship between one dependent variable (usually denoted by Y) and a series of other changing variables (known as independent variables). In simple words it finds a mathematical relationship between the dependent variable and independent variable. Output is continuous.

**Applications:** predicting the weather using the data-set of the weather conditions in the past, predict sales for a company based on previous sales.

**Root Mean Squared Error (RMSE):** RMSE is the most popular evaluation metric used in regression problems. It follows an assumption that error are unbiased and follow a normal distribution. RMSE metric is given by:

****

**Clustering:**

A cluster refers to a collection of data points aggregated together because of certain similarities.

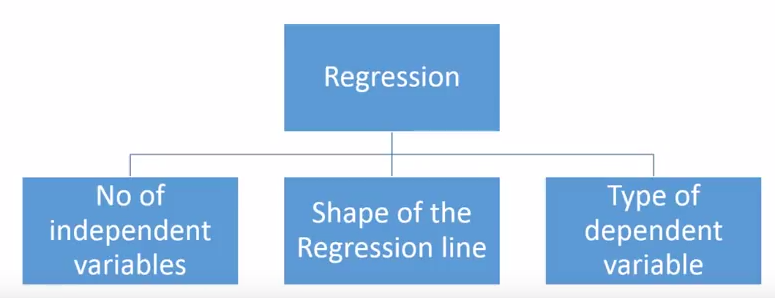
Unlike supervised learning, clustering is considered an unsupervised learning method since we don’t have the ground truth to compare the output of the clustering algorithm to the true labels to evaluate its performance.

**Cycle1- Logistic Regression**

**Objective:** Classify given data using logistic Regression

**Background Theory:** Regression analysis is a form of predictive modelling technique which investigates the relationship between a dependent (target) and independent variable (s). This technique is used for forecasting, time series modelling and finding the [causal effect relationship](https://www.analyticsvidhya.com/blog/2015/06/establish-causality-events/) between the variables.

**Types of regression:** There are various kinds of regression techniques available to make predictions. These techniques are mostly driven by three metrics



**1. A linear regression** refers to a regression model establishes a relationship between **dependent variable (Y)** and one or more **independent variables (X)** using a **best fit straight line** (also known as regression line). In this technique, the dependent variable is continuous, independent variable(s) can be [continuous or discrete](https://en.wikipedia.org/wiki/Continuous_and_discrete_variables), and nature of regression line is linear.

1. **Single Variable Linear Regression:** It is a technique used to model the relationship between a single input independent variable (feature variable) and an output dependent variable using a linear model i.e a line.

The general form is:

**Single variable Linear regression:** Y = a + bX + u

* Y = the variable that you are trying to predict (dependent variable).
* X = the variable that you are using to predict Y (independent variable).
* a = the intercept.
* b = the slope.
* u = the regression residual.



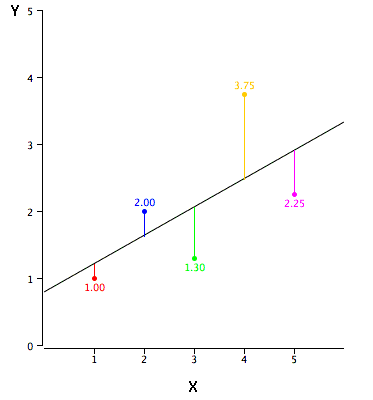
1. **Multiple regression:** The difference between simple linear regression and multiple linear regression is that, multiple linear regression has (>1) independent variables, whereas simple linear regression has only 1 independent variable. The model remains linear in that the output is a linear combination of the input variables. We can model a multi-variable linear regression as the following:

Y = a + b1X1+ b2X2 + b3X3 + ... + btXt + u

The above function does not include any non-linearities and so is only suited for modelling linearly separable data.

**How to find the best fit line:**

This task can be easily accomplished by Least Square Method. It is the most common method used for fitting a regression line. It calculates the best-fit line for the observed data by minimizing the sum of the squares of the vertical deviations from each data point to the line. Because the deviations are first squared, when added, there is no cancelling out between positive and negative values. [least square, regression line](https://www.analyticsvidhya.com/wp-content/uploads/2015/08/Least_Square.png)

[](https://www.analyticsvidhya.com/wp-content/uploads/2015/08/reg_error.gif)

We can evaluate the model performance using the metric **R-square**.

A few key points about Linear Regression:

* Fast and easy to model and is particularly useful when the relationship to be modeled is not extremely complex and if you don’t have a lot of data.
* Very intuitive to understand and interpret.
* Linear Regression is very sensitive to outliers.

**2. Logistic Regression:** Logistic regression is a classification algorithm used to assign observations to a discrete set of classes. Unlike linear regression which outputs continuous number values, logistic regression transforms its output using the logistic sigmoid function to return a probability value which can then be mapped to two or more discrete classes.

* To predict whether an email is spam (1) or (0)
* Whether the tumor is malignant (1) or not (0)

**Types of Logistic Regression**

* **Binary Logistic Regression:** The categorical response has only two 2 possible outcomes. Example: Spam or Not, pass or fail
* **Multinomial Logistic Regression:** Three or more categories without ordering. Example: Predicting which food is preferred more (Veg, Non-Veg, Vegan), Find whether the given picture of the animal belongs to Cats, Dogs, Sheep.
* **Ordinal Logistic Regression:** Three or more categories with ordering. Example: Movie rating from 1 to 5, Low, Medium, High

**Model**

Output = 0 or 1

Hypothesis => Z = WX + B

hΘ(x) = sigmoid (Z)

***Sigmoid Function***

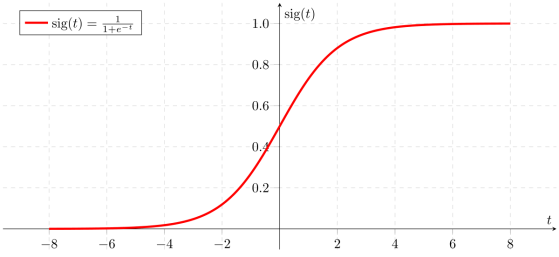
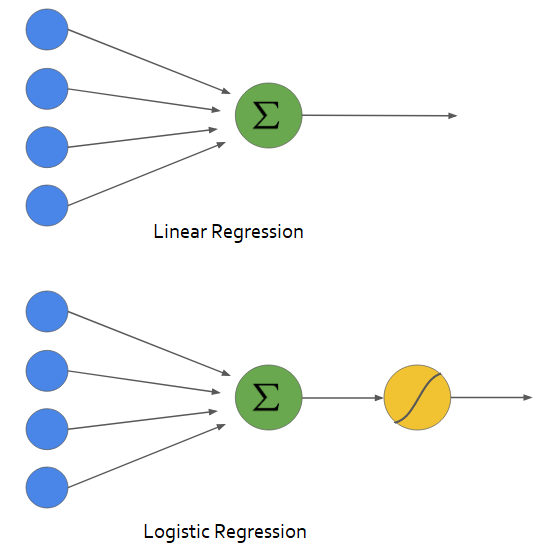


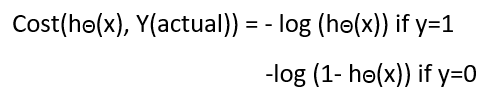
Figure 2: Sigmoid Activation Function

The corresponding output of the sigmoid function is a number between 0 and 1. The middle value is considered as threshold to establish what belong to the class 1 and to the class 0. In particular, an input producing an outcome greater than 0.5 is considered belong to the class 1. Conversely, if the output is less than 0.5, then the corresponding input is classified as belonging to 0 class

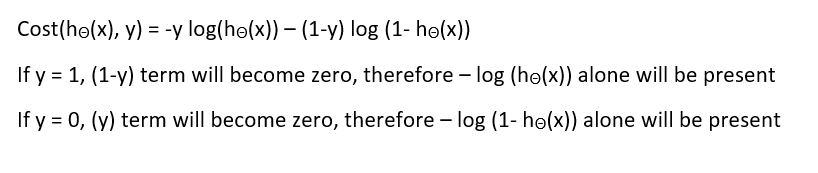


**Decision Boundary:** Decision boundary can be linear or non-linear. Polynomial order can be increased to get complex decision boundary.

**Cost Function:**



**Simplified cost function:**

****

**Python Functions to implement Logistic Regression:**

* LinearRegression is a class from **Scikit-learn**

Step1: #import the LinearRegression class: from sklearn.linear\_model import LogisticRegression

Step2: # instantiate the model (using the default parameters): model = LogisticRegression()

Step3: # fit the model with traning data: model.fit(X\_train,y\_train)

Step4: # Test the model with test data: y\_pred=model.predict(X\_test)

**Python code for logistic regression:**

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

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

# Assign colum names to the dataset

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

# Read dataset to pandas dataframe

dataset = pd.read\_csv(url, names=names)

# Split the data into features and labels

X = dataset.iloc[:, :-1].values

y = dataset.iloc[:, 4].values

# Split the data into train 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)

from sklearn.linear\_model import LogisticRegression

# Define regression Model and apply it on test data

reg = LogisticRegression()

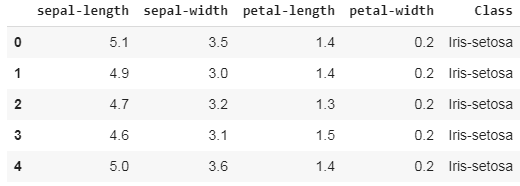
reg.fit(X\_train, y\_train)

score = reg.score(X\_test, y\_test)

print(score)

**Summary of the dataset:** We use the famous iris data set for our KNN example. The dataset consists of four attributes: sepal-width, sepal-length, petal-width and petal-length. These are the attributes of specific types of iris plant. The task is to predict the class to which these plants belong. There are three classes in the dataset: Iris-setosa, Iris-versicolor and Iris-virginica. This dataset can be downloaded from the following link: "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data"

Following are few of the samples from the dataset:

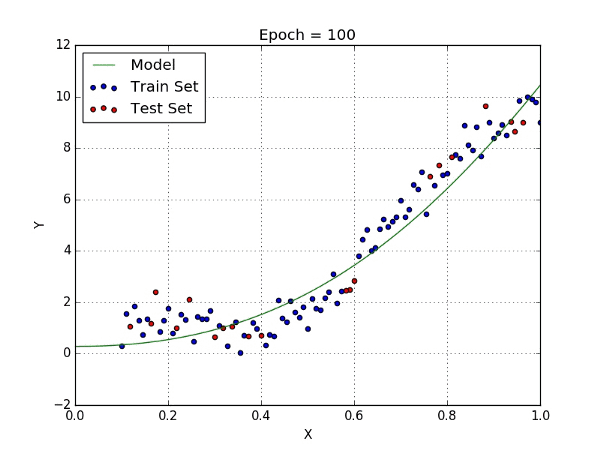
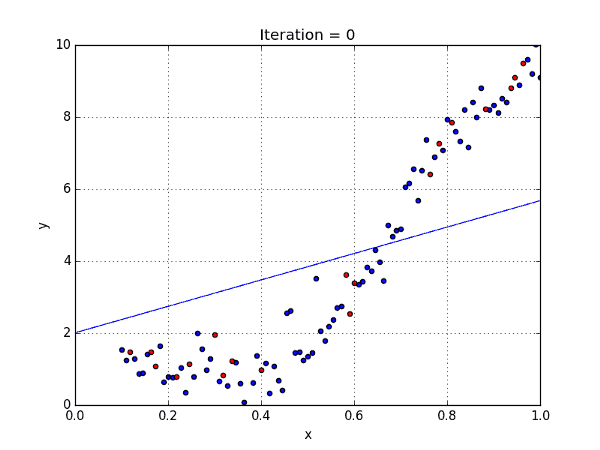


**Expected output: Accuracy – 94.4%**

**3. Polynomial Regression:** When we want to create a model that is suitable for handling non-linearly separable data, we will need to use a polynomial regression. It is rather a curve that fits into the data points. For a polynomial regression, the degree of few of the independent variables is greater than 1. For example, we can have something like:

Y = a1\*X1 + (a2)²\*X2 + (a3)⁴\*X3 ……. an\*Xn + b

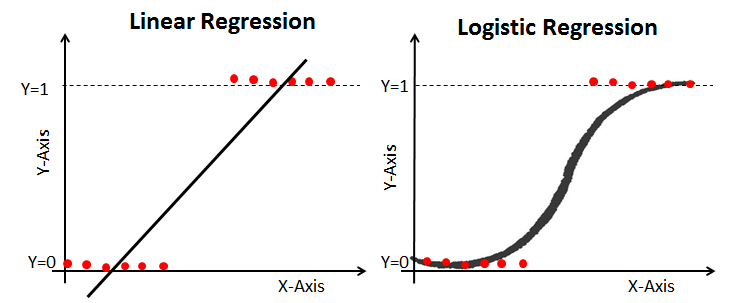
We can have some variables have exponents, others without, and also select the exact exponent we want for each variable. However, selecting the exact exponent of each variable naturally requires some knowledge of how the data relates to the output. See the illustration below for a visual comparison of linear vs polynomial regression.



Linear vs Polynomial Regression with data that is non-linearly separable

A few key points about Polynomial Regression:

* Able to model non-linearly separable data; linear regression can’t do this. It is much more flexible in general and can model some fairly complex relationships.
* Full control over the modelling of feature variables (which exponent to set).
* Requires careful design. Need some knowledge of the data in order to select the best exponents.
* Prone to over fitting if exponents are poorly selected.

****

Properties of Logistic Regression:

* The dependent variable in logistic regression follows Bernoulli Distribution.
* Estimation is done through maximum likelihood.
* No R Square, Model fitness is calculated through Concordance, KS-Statistics.

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**Cycle 2- Minimum Distance Classifier**

**Objective:** Classify given data using Minimum Distance Classifier

**Background Theory:** As the classification is based on the minimum distance calculation, this method is called minimum distance classification procedure. Minimum distance procedure performs efficiently if pattern classes can be represented by a single prototype or by several prototypes around which the patterns form clusters. The simplest type of minimum distance classifier is the one wherein the patterns of all classes are very close to each other and each class can be represented by a single prototype.

Following figure shows the concept of a minimum distance classifier.

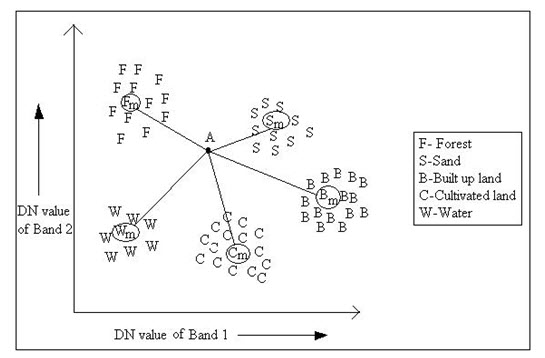


Fig. 12.1. Minimum distance to means classification strategy.

**Algorithm:**

**Step1: Training**

The kth class  is represented by its mean vector  and covariance matrix which can be estimated from the training samples: \begin{displaymath}{\bf m}_k=\frac{1}{K_k}\sum_{i=1}^{K_k} {\bf x}_i^{(k)}\;\;\;\;(k=1,\cdots,C) \end{displaymath} and \begin{displaymath}{\bf\Sigma}_k=\frac{1}{K_k}\sum_{i=1}^{K_k} ({\bf x}_i^{(k)}-{\bf m}_k)({\bf x}_i^{(k)}-{\bf m}_k)^T \end{displaymath}

**Step2: Classification**

A given pattern ${\bf x}$ of unknown class is classified to $\omega_k$ if its distance to $\omega_k$ is smaller than those to all other classes:  \begin{displaymath}
{\bf x}\in \omega_k\;\;\;\mbox{iff}\;\;\;d_M({\bf x},\omega_k)
=min \{d_M({\bf x},\omega_i)\;\;i=1,\cdots,C \}
\end{displaymath}

**Assumption:** Use a single prototype for each class ωi (usually the class’s mean mi)

**Key idea:** We don’t really need to know the distances to the prototypes—we just want to know which is closest.

The minimum distance classifier will classify x at class, Ci for which distance is minimum.

The objective function is = minimizing ||x- mi|| is the same as

= minimizing ||x- mi||2

= minimizing (x − mi) T (x − mi)

= minimizing xTx − 2xTmi + mTimi

= minimizing −2 xTmi + mTimi (since xTx is constant)

= maximizing xTmi – 1/2mTimi

For two-class case:

g1(x) = xTm1 – 1/2mT1m1

g2(x) = xTm2 – 1/2mT2m2

Create a single decision function:

g(x) = g1(x) − g2(x)

=( xTm1 – 1/2mT1m1 ) – (xTm2 – 1/2mT2m2 )

= xT(m1 − m2) – 1/2 (mT1m1 − mT2m2)

**Linear Decision Boundaries:**

g(x) = wTx + w0

where w = m1 − m2 and w0 = – 1/2 (mT1m1 − mT2m2)

g(x) > 0 Assign x to ω1

g(x) < 0 Assign x to ω2

g(x) = 0 Undecided

**Three approaches for multiple classes:**

1. Construct a single discriminant for each class that separates ωi from “not ωi”.

2. Construct a discriminant gij between each pair of classes ωi and ωj :

gij(x) = gi(x) − gj(x)

3. Construct a single discriminant gi(x) for each class ωi , and assign x to class ωi if gi(x) > gj(x) for all other classes ωj

**Python implementation:**

import csv

import numpy as np

import sys

# load csv file into numpy as matrices

a = np.loadtxt('wine\_train.csv',dtype=float, delimiter=',')

# Define the sample numbers, feature numbers, class numbers

(sampleNum,featureNum)=a.shape

print('Number of samples',sampleNum)

print('Number of features',featureNum)

classNum = len(np.unique(a[:,13]))

print('Number of classes',classNum)

# Train the classifier to find two features that have minimum error rate.

training\_data=a[:,0:13]

minError=sys.maxsize

selectedFeature=(0,1)

for s in range(0,featureNum-3):

for t in range(s+1, featureNum-2):

for x in range(0,sampleNum):

distance\_Between\_Class1\_and\_DataPoints= np.square(training\_data[x,s]-classMean[0,s])+np.square(training\_data[x,t]-classMean[0,t])

distance\_Between\_Class2\_and\_DataPoints= np.square(training\_data[x,s]-classMean[1,s])+np.square(training\_data[x,t]-classMean[1,t])

distance\_Between\_Class3\_and\_DataPoints= np.square(training\_data[x,s]-classMean[2,s])+np.square(training\_data[x,t]-classMean[2,t])

if(min(distance\_Between\_Class1\_and\_DataPoints,distance\_Between\_Class2\_and\_DataPoints,distance\_Between\_Class3\_and\_DataPoints)==distance\_Between\_Class1\_and\_DataPoints):

label\_train\_done[x]=1

elif (min(distance\_Between\_Class1\_and\_DataPoints,distance\_Between\_Class2\_and\_DataPoints,distance\_Between\_Class3\_and\_DataPoints)==distance\_Between\_Class2\_and\_DataPoints):

label\_train\_done[x]=2

else:

label\_train\_done[x]=3

errorNum=(label\_train\_done != a[:,13]).sum()

errorRate=errorNum/sampleNum

if(minError!=min(errorRate,minError)):

selectedFeature=(s+1,t+1)

minError=min(errorRate,minError)

# print(errorNum)

# print(sampleNum)

# print(errorNum/sampleNum)

print (selectedFeature)

print (minError)

#plotDecBoundaries(a[:,0:13],a[:,13],classMean[:,0:2])

**Dataset Summary:** Wine dataset is used for this experiment. This dataset contains 2 classes, 178 records with 13 different features ('Alcohol', 'Malic acid', 'Ash', 'Alcalinity of ash', 'Magnesium', 'Total phenols', 'Flavanoids', 'Nonflavanoid phenols', 'Proanthocyanins', 'Color intensity', 'Hue', 'OD280/OD315 of diluted wines', 'Proline' ).

**Expected output:**

Misclassification rate = 0.07

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**Cycle-3 :: K-Nearest Neighbour Classifier**

**Objectives:** The purpose of this laboratory session is to introduce the simplest classifier: k-Nearest Neighbor classifier. The classifier is applied on a small image dataset for with multiple classes.

**Theoretical Background**: One the simplest supervised classifier is the k-NN classifier. It is a non-parametric multi-class classifier. It makes the decision about the test sample based on the k nearest neighbors from a labeled training dataset. The next figure shows samples belonging to 3 different classes shown in different colors (yellow, red, black). Test sample is represented as a blue square. A circle enclosing the 5 closest neighbors indicates the region which is used to infer the class of the test sample. It can be labeled as a type of instance-based learning, or lazy learning, since the classifier function is only approximated locally and all computation is deferred until classification.

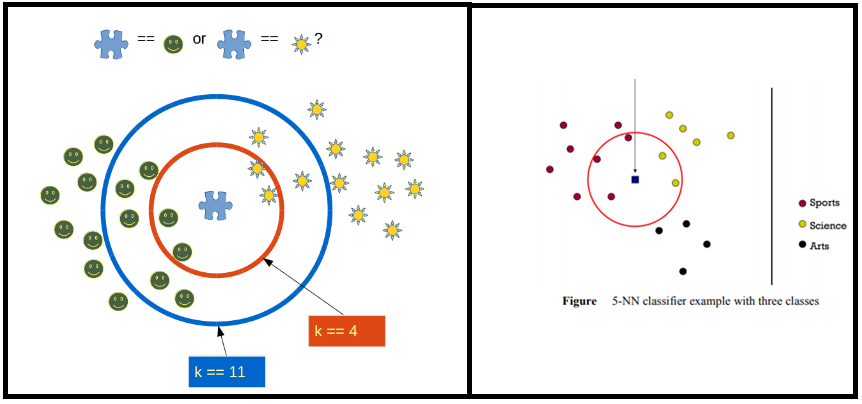


Figure : Illustartion of KNN based classification

**Classifier algorithm:**

We can implement a KNN model by following the below steps:

1. Load the data
2. Initialize the value of k
   1. Calculate the distance between test data and each example of the training data.  Use any Distance metric to find the distance. Ex: Euclidean distance, Chebyshev, cosine, etc.
   2. Sort the calculated distances in ascending order based on distance values
   3. Get top k closest training samples from the sorted array
   4. Get the most frequent class and return it as the predicted class

**Implementation in python:**

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

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

# Assign colum names to the dataset

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

# Read dataset to pandas dataframe

dataset = pd.read\_csv(url, names=names)

# Split the data into features and labels

X = dataset.iloc[:, :-1].values

y = dataset.iloc[:, 4].values

# Split the data into train 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)

from sklearn.neighbors import KNeighborsClassifier

classifier = KNeighborsClassifier(n\_neighbors=5)

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

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

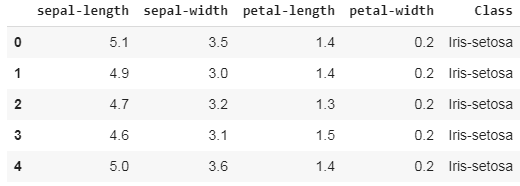
from sklearn.metrics import classification\_report, confusion\_matrix

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

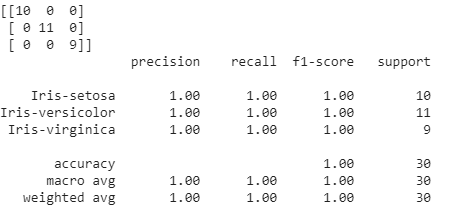
print(classification\_report(y\_test, y\_pred))

**Dataset:** We use the famous iris data set for our KNN example. The dataset consists of four attributes: sepal-width, sepal-length, petal-width and petal-length. These are the attributes of specific types of iris plant. The task is to predict the class to which these plants belong. There are three classes in the dataset: Iris-setosa, Iris-versicolor and Iris-virginica. This dataset can be downloaded from the following link: "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data"

Following are few of the samples from the dataset:



Sample output:



**Assignment:**

a. Apply normalization and check the results.

b. Test the algorithm with different distance functions.

c. Iterate k-value from 1 to 40 and draw graphs showing accuracy vs k and error vs k

d. Implement KNN on your down dataset.

**Pros and Cons of KNN:** In this section we'll present some of the pros and cons of using the KNN algorithm.

**Pros**

1. It is extremely easy to implement
2. As said earlier, it is [lazy learning](https://en.wikipedia.org/wiki/Lazy_learning) algorithm and therefore requires no training prior to making real time predictions. This makes the KNN algorithm much faster than other algorithms that require training e.g SVM, [linear regression](https://stackabuse.com/linear-regression-in-python-with-scikit-learn/), etc.
3. Since the algorithm requires no training before making predictions, new data can be added seamlessly.
4. There are only two parameters required to implement KNN i.e. the value of K and the distance function (e.g. Euclidean or Manhattan etc.)

**Cons**

1. The KNN algorithm doesn't work well with high dimensional data because with large number of dimensions, it becomes difficult for the algorithm to calculate distance in each dimension.
2. The KNN algorithm has a high prediction cost for large datasets. This is because in large datasets the cost of calculating distance between new point and each existing point becomes higher.
3. Finally, the KNN algorithm doesn't work well with categorical features since it is difficult to find the distance between dimensions with categorical features

**Cycle 4 :: Linear discriminant functions (LDA)**

**Objective:** Classify given data using linear discriminant functions

**Background theory:**

Two types of Probabilistic classification approaches:

* Generative
* Discriminative

**Discriminant function:** Various procedures are used to determining discriminant functions (some of them are statistical). They don’t require knowledge of the forms of underlying probability distributions as in the case of of generative models. Discriminant function is a popular way of representing a classifier.

* A discriminant function gi(x) is designed for each class ωi (i=1,…..,c).
  + x is assigned to class ωi if gi(x) > gj(x) **∀**j ≠ i
* Decision surfaces (boundaries) can also be found using discriminant functions.
  + Boundary of the regions ℛi and ℛj : ∀x, gi(x) = gj(x)

**Probabilistic Discriminant Functions:**

* Maximum likelihood: gi(x) = p(x / ωi )
* Bayesian Classifier: gi(x) = p( ωi / x)
* Expected Loss (Conditional Risk): gi(x) = - R(ai / x)

**linear discriminant functions:** Decision boundaries are linear in x ,or linear in some given set of functions of x.

g(x) = wTx + w0  …….(1)

Where w is the weight vector w0 the bias or threshold weight. In general there are c such functions.

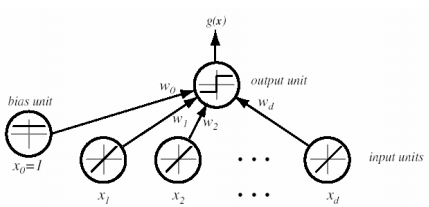
**Simple Linear Two Category Classifier:**

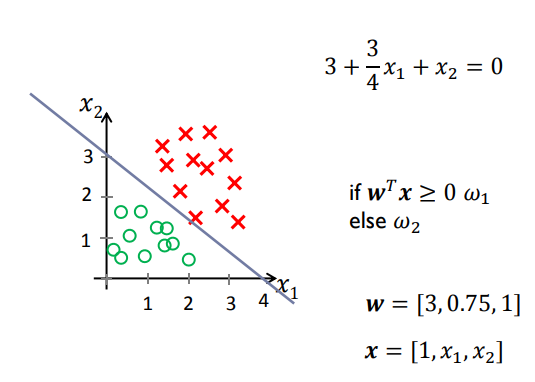
With a discriminant function of the form (1) use:

• Decide ω1 if g(x) = wTx + w0 > 0

and ω1 if g(x) = wTx + w0 < 0

* Decide g(x) = 0 then x is assigned to either of the classes

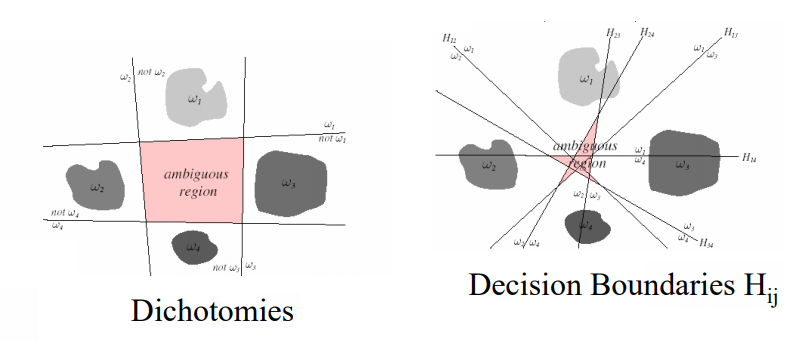




Multi-Category Classifier: Regard the problem as c two-class problems and two methods can be used:

* Separate points assigned to ωi from those not assigned to ωi
* Use one hyperplane for each pair for classes Will need c(c-1)/2 discriminant functions.

Using dichotomies for four-class problem:

****

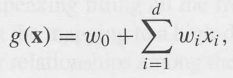
**Multi-category case: Linear Machine**

* We define c linear discriminant functions
* assign x to ωi if gi(x) > gj(x) ∀ j ≠ i;
* in case of ties, the classification is undefined

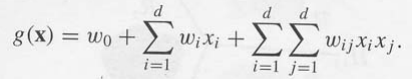
In this case, the classifier is a “linear machine”. A linear machine divides the feature space into c decision regions, with gi(x) being the largest discriminant if x is in the region Ri .

**Generalized Linear Discriminant Functions**:

• Linear Discriminant functions can be written as



• Adding additonal terms we get quadratic discriminant function:



**Non- linear discriminant functions:**

• Functions yi(x) are called φ-functions

• They map points in d-dimensional space into points in higher dimensional space

**LDF Methods for classification:** Many classification methods are based on LDFs:

* Mean Squared Error
* Sum of Squared Error}
* Perceptorn
* Fisher Linear Discriminant Analysis (LDA)
* SVM

**Python implementation for perceptron:**

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

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

# Assign colum names to the dataset

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

# Read dataset to pandas dataframe

dataset = pd.read\_csv(url, names=names)

# Split the data into features and labels

X = dataset.iloc[:, :-1].values

y = dataset.iloc[:, 4].values

# Split the data into train 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)

from sklearn.linear\_model import Perceptron

from sklearn.metrics import accuracy\_score

# Create a perceptron object with the parameters: 40 iterations (epochs) over the data, and a learning rate of 0.1

ppn = Perceptron(n\_iter=40, eta0=0.1, random\_state=0)

# Train the perceptron

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

# Apply the trained perceptron on the X data to make predicts for the y test data

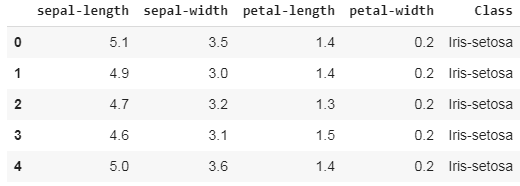
y\_pred = ppn.predict(X\_test)

# View the accuracy of the model, which is: 1 - (observations predicted wrong / total observations)

print('Accuracy: %.2f' % accuracy\_score(y\_test, y\_pred))

**Dataset:** We use the famous iris data set for our KNN example. The dataset consists of four attributes: sepal-width, sepal-length, petal-width and petal-length. These are the attributes of specific types of iris plant. The task is to predict the class to which these plants belong. There are three classes in the dataset: Iris-setosa, Iris-versicolor and Iris-virginica. This dataset can be downloaded from the following link: "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data"

Following are few of the samples from the dataset:



**Expected output:** Accuracy - 87%

**Cycle-5: K-means clustering algorithm**

**Objective:** Cluster given data using K-means clustering algorithm.

**Back ground theory:** K-means clustering is one of the simplest and popular unsupervised machine learning algorithms. Typically, unsupervised algorithms make inferences from datasets using only input vectors without referring to known, or labeled, outcomes.

**k-means clustering algorithm:**

In the clustering problem, we are given a training set {x (1), . . . , x**(m)**}, and want to group the data into a few cohesive “clusters.” Here, x (i) ∈ Rn as usual; but no labels y(i) are given. So, this is an unsupervised learning problem. The k-means clustering algorithm works as follows:

1. Initialize cluster centroids µ1, µ2, . . . , µk ∈ Rn randomly.

2. Repeat until convergence: {

For every i, set

c(i) := argminj ||x(i) − µj ||2 .

For each j, set



}

In the algorithm above, k (a parameter of the algorithm) is the number of clusters we want to find; and the cluster centroids µj represent our current guesses for the positions of the centers of the clusters. To initialize the cluster centroids (in step 1 of the algorithm above), we could choose k training examples randomly, and set the cluster centroids to be equal to the values of these k examples. (Other initialization methods are also possible.)

The inner-loop of the algorithm repeatedly carries out two steps: (i) “Assigning” each training example x (i) to the closest cluster centroid µj , and (ii) Moving each cluster centroid µj to the mean of the points assigned to it.

**Implementation:**

Following functions KMeans() is used along with different parameters:

KMeans(algorithm='auto', copy\_x=True, init='k-means++', max\_iter=300, n\_clusters=2, n\_init=10, n\_jobs=None, precompute\_distances='auto', random\_state=None, tol=0.0001, verbose=0)

**Python code:**

#Import libraries

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.cluster import KMeans

%matplotlib inline

#Generate random data

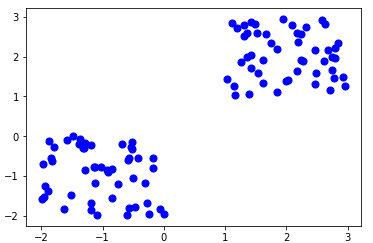
X= -2 \* np.random.rand(100,2)

X1 = 1 + 2 \* np.random.rand(50,2)

X[50:100, :] = X1

plt.scatter(X[ : , 0], X[ :, 1], s = 50, c = 'b')

plt.show()



#Apply k-means

from sklearn.cluster import KMeans

Kmean = KMeans(n\_clusters=2)

Kmean.fit(X)

#cluster Centers

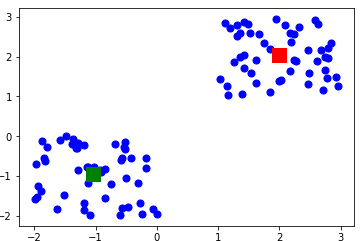
Kmean.cluster\_centers\_

plt.scatter(X[ : , 0], X[ : , 1], s =50, c='b')

plt.scatter(-1.03599643, -0.94414139, s=200, c='g', marker='s')

plt.scatter(1.99411517, 2.04101292, s=200, c='r', marker='s')

plt.show()



# predict a test sample

sample\_test=np.array([-3.0,-3.0])

second\_test=sample\_test.reshape(1, -1)

Kmean.predict(second\_test)

**Training Dataset:** A total of 100 data points has been generated and divided into two groups, of 50 points each.

**Test sample:** [-3, 3]

**Expected output:** Test sample must be assigned to Cluster1

**Cycle-6: Dimension reduction**

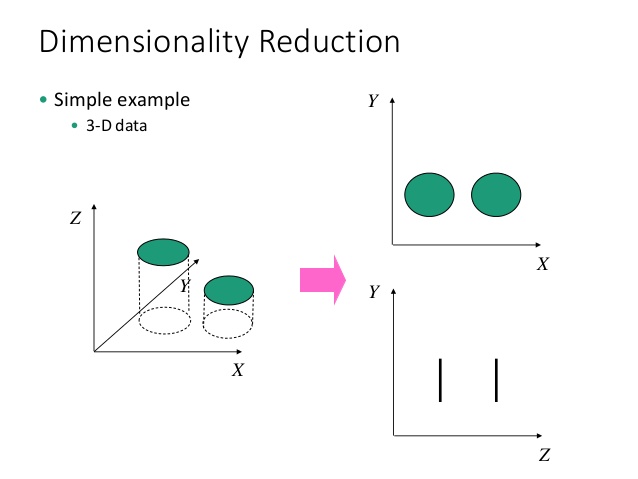
**Objective:** Apply dimension reduction using PCA, KPCA, LDA, Fisher’s LDA and perform comparative study.

## Background theory: The higher the number of features, the harder it gets to visualize the training set and then work on it. Sometimes, most of these features are correlated, and hence redundant. This is where dimensionality reduction algorithms come into play. Dimensionality reduction is the process of reducing the number of random variables under consideration, by obtaining a set of principal variables. It can be divided into feature selection and feature extraction.

* **Feature selection:** In this, we try to find a subset of the original set of variables, or features, to get a smaller subset which can be used to model the problem. It usually involves three ways:
  1. Filter
  2. Wrapper
  3. Embedded
* **Feature extraction:** This reduces the data in a high dimensional space to a lower dimension space, i.e. a space with lesser no. of dimensions.

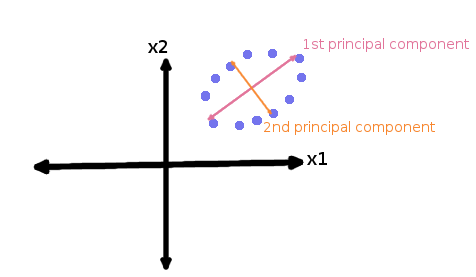
The various methods used for dimensionality reduction include:

* Principal Component Analysis (PCA)
* Kernel PCA
* Linear Discriminant Analysis (LDA)
* Generalized Discriminant Analysis (GDA)



**Principal Component Analysis:** In this technique, variables are transformed into a new set of variables, which are linear combination of original variables. These new set of variables are known as **principle components.** They are obtained in such a way that first principle component accounts for most of the possible variation of original data after which each succeeding component has the highest possible variance.

The second principal component must be orthogonal to the first principal component. In other words, it does its best to capture the variance in the data that is not captured by the first principal component. For two-dimensional dataset, there can be only two principal components. Below is a snapshot of the data and its first and second principal components. You can notice that second principle component is orthogonal to first principle component.

[](https://www.analyticsvidhya.com/wp-content/uploads/2015/07/PCA.png)

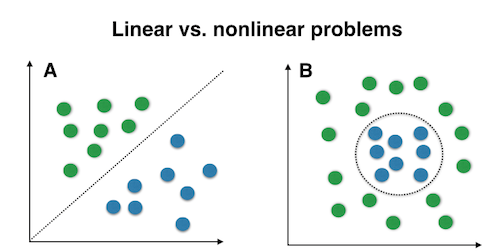
The principal components are sensitive to the scale of measurement, now to fix this issue we should always standardize variables before applying PCA. Applying PCA to your data set loses its meaning. If interpretability of the results is important for your analysis, PCA is not the right technique for your project.

The standard PCA approach can be summarized in six simple steps:

1. [Take the whole dataset consisting of d-dimensional samples ignoring the class labels](https://sebastianraschka.com/Articles/2014_pca_step_by_step.html#drop_labels)
2. [Compute the d-dimensional mean vector](https://sebastianraschka.com/Articles/2014_pca_step_by_step.html#mean_vec) (i.e., the means for every dimension of the whole dataset)
3. [Compute the scatter matrix (alternatively, the covariance matrix) of the whole data set](https://sebastianraschka.com/Articles/2014_pca_step_by_step.html#sc_matrix)
4. [Compute eigenvectors (e1,e2,...,...,ed) and corresponding eigenvalues (λ1,λ2,...,,...,λd)](https://sebastianraschka.com/Articles/2014_pca_step_by_step.html#eig_vec)
5. [Sort the eigenvectors by decreasing eigenvalues and choose k eigenvectors with the largest eigenvalues to form a d×k dimensional matrix W](https://sebastianraschka.com/Articles/2014_pca_step_by_step.html#sort_eig)(where every column represents an eigenvector)
6. [Use this d×k eigenvector matrix to transform the samples onto the new subspace.](https://sebastianraschka.com/Articles/2014_pca_step_by_step.html#transform) This can be summarized by the mathematical equation: y=WTx (where x is a d×1dimensional vector representing one sample, and y is the transformed k×1-dimensional sample in the new subspace.)

## Nonlinear dimensionality reduction using PCA (KPCA):

The “classic” PCA approach described above is a linear projection technique that works well if the data is linearly separable. However, in the case of linearly inseparable data, a nonlinear technique is required if the task is to reduce the dimensionality of a dataset.



## Kernel functions and the kernel trick: The basic idea to deal with linearly inseparable data is to project it onto a higher dimensional space where it becomes linearly separable. Let us call this nonlinear mapping function ϕ so that the mapping of a sample x can be written as x→ϕ(x), which is called “kernel function”. Now, the term “kernel” describes a function that calculates the dot product of the images of the samples x under ϕ.

κ(xi,xj)=ϕ(xi)ϕ(xj)T

In other words, the function ϕ maps the original d-dimensional features into a larger, k-dimensional feature space by creating non-linear combinations of the original features. For example, if x consists of 2 features:

x=[x1 x2]T and x∈ Rd

x′ = ϕ(xi) =[x1  x2 x1x2 x12 x1x23…]T where x ∈ Rk (k>>d)

Often, the mathematical definition of the RBF kernel is written and implemented as

κ(xi,xj)=exp(−γ∥xi−xj∥2) where γ=1/2σ2 is a free parameter that is to be optimized.

## PCA-vs-KernelPCA.png

## Implementing the RBF kernel PCA step-by-step:

In order to implement the RBF kernel PCA we just need to consider the following two steps.

#### 1. Computation of the kernel (similarity) matrix: In this first step, we need to calculate

κ(xi,xj)=exp(−γ∥xi−xj∥2)

for every pair of points. E.g., if we have a dataset of 100 samples, this step would result in a symmetric 100x100 kernel matrix.

#### 2. Eigen decomposition of the kernel matrix: Since it is not guaranteed that the kernel matrix is centered, we can apply the following equation to do so:

K′=K−1NK−K1N+1NK1N

where 1N is (like the kernel matrix) a N×N matrix with all values equal to 1N. Now, we have to obtain the eigenvectors of the centered kernel matrix that correspond to the largest eigenvalues. Those eigenvectors are the data points already projected onto the respective principal components.

1. **Apply Logistic regression on the original data**

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

# Assign colum names to the dataset

names = ['Class label', 'Alcohol', 'Malic acid', 'Ash', 'Alcalinity of ash', 'Magnesium', 'Total phenols', 'Flavanoids', 'Nonflavanoid phenols', 'Proanthocyanins', 'Color intensity', 'Hue', 'OD280/OD315 of diluted wines', 'Proline']

#1. import the data

#pls find the data here:

#https://raw.githubusercontent.com/rasbt/python-machine-learning-book/master/code/datasets/wine/wine.data

import pandas as pd

url = "https://raw.githubusercontent.com/rasbt/python-machine-learning-book/master/code/datasets/wine/wine.data"

# Read dataset to pandas dataframe

df\_wine = pd.read\_csv(url, names=names)

###2. Observe

print(df\_wine.shape)

print(df\_wine.head())

classNum = len(np.unique(y))

print('Number of classes',classNum)

#3. APPLY Logisitc regression

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

X, y = df\_wine.iloc [:, 1:].values, df\_wine.iloc[:, 0].values

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=0)

from sklearn.linear\_model import LogisticRegression

import matplotlib.pyplot as plt

lr0 = LogisticRegression()

lr0 = lr0.fit(X\_train, y\_train)

print(lr0.score(X\_test,y\_test))

**Dataset Summary:** Wine dataset is used for this experiment. This dataset contains 2 classes, 178 records with 13 different features ('Alcohol', 'Malic acid', 'Ash', 'Alcalinity of ash', 'Magnesium', 'Total phenols', 'Flavanoids', 'Nonflavanoid phenols', 'Proanthocyanins', 'Color intensity', 'Hue', 'OD280/OD315 of diluted wines', 'Proline' ).

**Expected Output:** 0.944444444444

**Task1: Dimensionality reduction for visualization of the data**

**LDA for visualization**

#Apply standard scalar

from sklearn.preprocessing import StandardScaler

sc = StandardScaler()

X\_train\_std = sc.fit\_transform(X\_train)

X\_test\_std = sc.transform(X\_test)

###We apply LDA for X\_train with y\_train

from sklearn.discriminant\_analysis import LinearDiscriminantAnalysis as LDA

lda = LDA(n\_components=2)

X\_train\_lda = lda.fit\_transform(X\_train\_std, y\_train)

lr = LogisticRegression()

lr = lr.fit(X\_train\_lda, y\_train)

###Here we define a function to visualize the decision boundaries

from matplotlib.colors import ListedColormap

import numpy as np

def plot\_decision\_regions(X, y, classifier, resolution=0.02):

# setup marker generator and color map

markers = ('s', 'x', 'o', '^', 'v')

colors = ('red', 'blue', 'lightgreen', 'gray', 'cyan')

cmap = ListedColormap(colors[:len(np.unique(y))])

# plot the decision surface

x1\_min, x1\_max = X[:, 0].min() - 1, X[:, 0].max() + 1

x2\_min, x2\_max = X[:, 1].min() - 1, X[:, 1].max() + 1

xx1, xx2 = np.meshgrid(np.arange(x1\_min, x1\_max, resolution),

np.arange(x2\_min, x2\_max, resolution))

Z = classifier.predict(np.array([xx1.ravel(), xx2.ravel()]).T)

Z = Z.reshape(xx1.shape)

plt.contourf(xx1, xx2, Z, alpha=0.4, cmap=cmap)

plt.xlim(xx1.min(), xx1.max())

plt.ylim(xx2.min(), xx2.max())

# plot class samples

for idx, cl in enumerate(np.unique(y)):

plt.scatter(x=X[y == cl, 0], y=X[y == cl, 1],

alpha=0.8, c=cmap(idx),

marker=markers[idx], label=cl)

plot\_decision\_regions(X\_train\_lda, y\_train, classifier=lr)

plt.xlabel('LD 1')

plt.ylabel('LD 2')

plt.legend(loc='lower left')

plt.tight\_layout()

plt.show()

###We apply LDA for X\_test with y\_test

X\_test\_lda = lda.transform(X\_test\_std)

plot\_decision\_regions(X\_test\_lda, y\_test, classifier=lr)

plt.xlabel('LD 1')

plt.ylabel('LD 2')

plt.legend(loc='lower left')

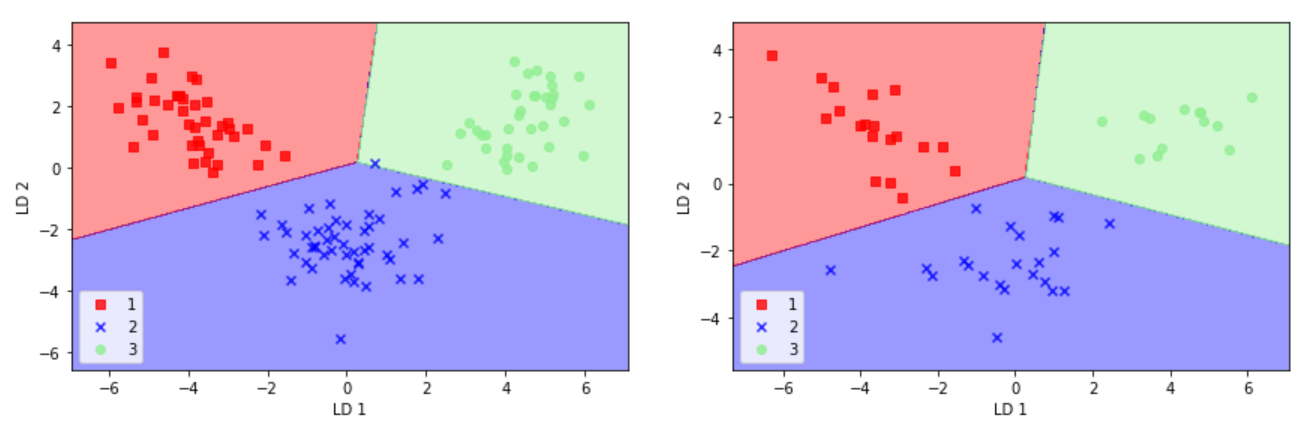
plt.tight\_layout()

# plt.savefig('./images/lda4.png', dpi=300)

plt.show()

print(lr.score(X\_test\_lda,y\_test))

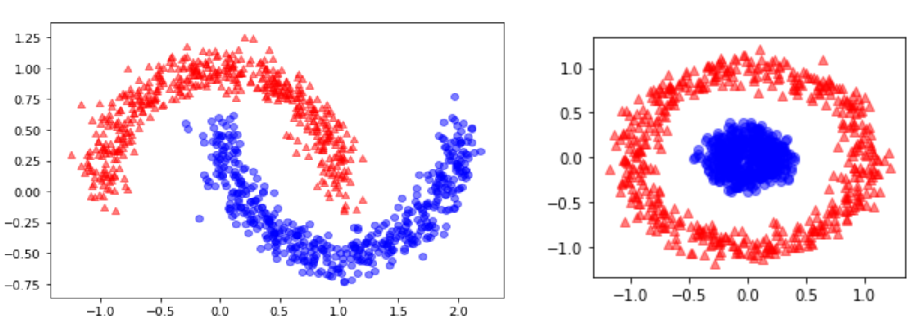
###1.0 Overfitting might be the problem here

****

**Figure: Train data and Test data in the reduced dimension after applying LDA**

**Task2: Dimensionality reduction for better separability in the projected space**

**# PCA on Moon data:** From sklearn package download moon and circle datasets. Following figure shows the pictorial representation of the datasets. This type of data can not classified using linear classifiers as they are non-linearly separable.



**Figure: Toy datasets – Moon and Circle**

###We do it with PCA

scikit\_pca = PCA(n\_components=2)

X\_spca = scikit\_pca.fit\_transform(X)

fig, ax = plt.subplots(nrows=1,ncols=2, figsize=(7,3))

ax[0].scatter(X\_spca[y==0, 0], X\_spca[y==0, 1],

color='red', marker='^', alpha=0.5)

ax[0].scatter(X\_spca[y==1, 0], X\_spca[y==1, 1],

color='blue', marker='o', alpha=0.5)

ax[1].scatter(X\_spca[y==0, 0], np.zeros((500,1))+0.02,

color='red', marker='^', alpha=0.5)

ax[1].scatter(X\_spca[y==1, 0], np.zeros((500,1))-0.02,

color='blue', marker='o', alpha=0.5)

ax[0].set\_xlabel('PC1')

ax[0].set\_ylabel('PC2')

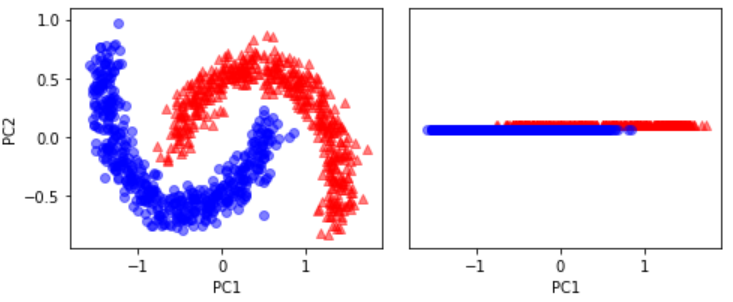
ax[1].set\_ylim([-1, 1])

ax[1].set\_yticks([])

ax[1].set\_xlabel('PC1')

plt.tight\_layout()

plt.show()

****

**Figure: Moon data projected to 1-D using PCA**

**Observations:** We can observe that the data can classified using linear classifier but most of the data can be misclassified.

###We do it with Kenel PCA

X\_kpca = rbf\_kernel\_pca(X, gamma=15, n\_components=2)

fig, ax = plt.subplots(nrows=1,ncols=2, figsize=(7,3))

ax[0].scatter(X\_kpca[y==0, 0], X\_kpca[y==0, 1],

color='red', marker='^', alpha=0.5)

ax[0].scatter(X\_kpca[y==1, 0], X\_kpca[y==1, 1],

color='blue', marker='o', alpha=0.5)

ax[1].scatter(X\_kpca[y==0, 0], np.zeros((500,1))+0.02,

color='red', marker='^', alpha=0.5)

ax[1].scatter(X\_kpca[y==1, 0], np.zeros((500,1))-0.02,

color='blue', marker='o', alpha=0.5)

ax[0].set\_xlabel('PC1')

ax[0].set\_ylabel('PC2')

ax[1].set\_ylim([-1, 1])

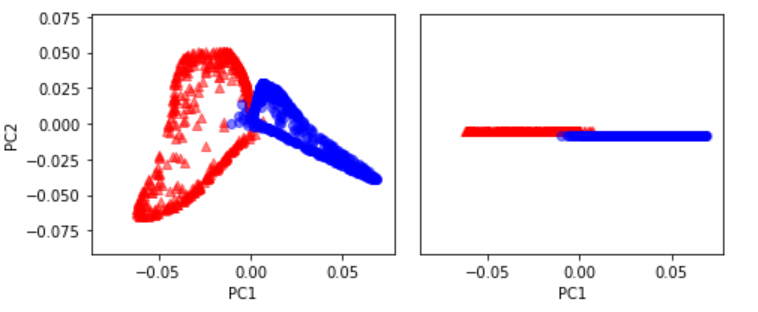
ax[1].set\_yticks([])

ax[1].set\_xlabel('PC1')

plt.tight\_layout()

# plt.savefig('./figures/circles\_3.png', dpi=300)

plt.show()

****

**Figure: Data projected to 2-D and 1-D using KPCA**

**Observations:** We can observe that the data can classified using with fewer misclassifications than the previous case.

**Cycle 7 :: SVM, FCM**

**Objective:** Classify using SVM, FCM and perform comparative study

**SVM – Background theory:**

A Support Vector Machine (SVM) is a discriminative classifier formally defined by a separating hyperplane. In other words, given labeled training data (supervised learning), the algorithm outputs an optimal hyperplane (maximum margin hyperplane) which categorizes new examples. In two dimentional space this hyperplane is a line dividing a plane in two parts where in each class lay in either side.

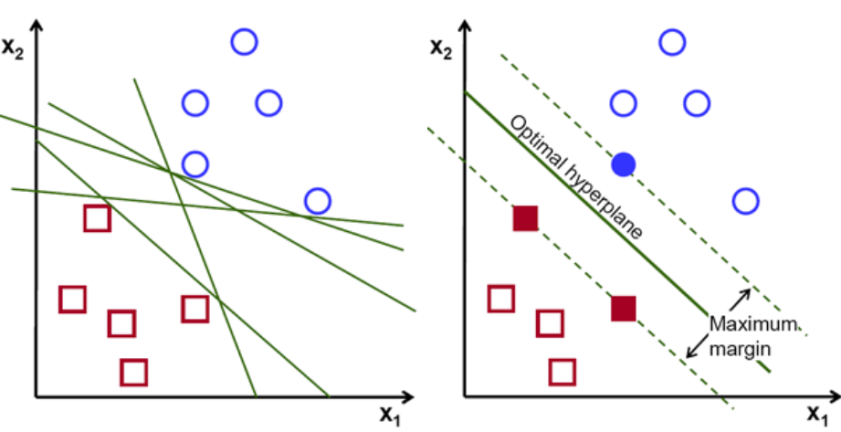
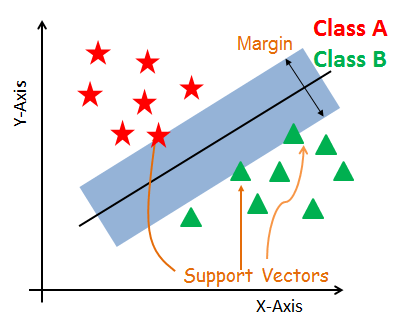
 

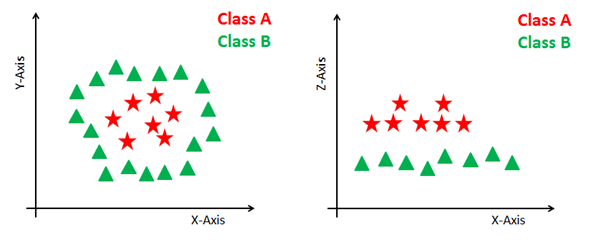
Figure: Difference between separating hyperplanes and maximum margin hyperplane

#### Support Vectors: Support vectors are the data points, which are closest to the hyperplane. These points will define the separating line better by calculating margins. These points are more relevant to the construction of the classifier.

#### Hyperplane: A hyperplane is a decision plane which separates between a set of objects having different class memberships.

#### Margin: A margin is a gap between the two lines on the closest class points. This is calculated as the perpendicular distance from the line to support vectors or closest points. If the margin is larger in between the classes, then it is considered a good margin, a smaller margin is a bad margin.

#### Dealing with non-linear and inseparable planes: Some problems can’t be solved using linear hyperplane, as shown in the figure below (left-hand side). In such situation, SVM uses a kernel trick to transform the input space to a higher dimensional space as shown on the right. The data points are plotted on the x-axis and z-axis (Z is the squared sum of both x and y: z=x^2=y^2). Now you can easily segregate these points using linear separation.



## SVM Kernels

The SVM algorithm is implemented in practice using a kernel. A kernel transforms an input data space into the required form. SVM uses a technique called the kernel trick. Here, the kernel takes a low-dimensional input space and transforms it into a higher dimensional space. In other words, you can say that it converts nonseparable problem to separable problems by adding more dimension to it. It is most useful in non-linear separation problem. Kernel trick helps you to build a more accurate classifier.

* **Linear Kernel** A linear kernel can be used as normal dot product any two given observations. The product between two vectors is the sum of the multiplication of each pair of input values.

K(x, xi) = sum(x \* xi)

* **Polynomial Kernel** A polynomial kernel is a more generalized form of the linear kernel. The polynomial kernel can distinguish curved or nonlinear input space.

K(x,xi) = 1 + sum(x \* xi)^d

Where d is the degree of the polynomial. d=1 is similar to the linear transformation. The degree needs to be manually specified in the learning algorithm.

* **Radial Basis Function Kernel** The Radial basis function kernel is a popular kernel function commonly used in support vector machine classification. RBF can map an input space in infinite dimensional space.

K(x,xi) = exp(-gamma \* sum((x – xi^2))

Here gamma is a parameter, which ranges from 0 to 1. A higher value of gamma will perfectly fit the training dataset, which causes over-fitting. Gamma=0.1 is considered to be a good default value. The value of gamma needs to be manually specified in the learning algorithm

**Python code:**

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

# Assign colum names to the dataset

names = ['Class label', 'Alcohol', 'Malic acid', 'Ash', 'Alcalinity of ash', 'Magnesium', 'Total phenols', 'Flavanoids', 'Nonflavanoid phenols', 'Proanthocyanins', 'Color intensity', 'Hue', 'OD280/OD315 of diluted wines', 'Proline']

#1. import the data

#pls find the data here:

#https://raw.githubusercontent.com/rasbt/python-machine-learning-book/master/code/datasets/wine/wine.data

import pandas as pd

url = <https://raw.githubusercontent.com/rasbt/python-machine-learning-book/master/code/datasets/wine/wine.data>

# Read dataset to pandas dataframe

df\_wine = pd.read\_csv(url, names=names)

X, y = df\_wine.iloc [:, 1:].values, df\_wine.iloc[:, 0].values

###2. Observe

print(df\_wine.shape)

#print(df\_wine.head())

classNum = len(np.unique(y))

print('Number of classes',classNum)

#3. Split Train & 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=0.3, random\_state=0)

**#Linear kernel:**

#Import Library

from sklearn import svm

# Create SVM classification object

model = svm.SVC(kernel='linear')

#Train the model using the training sets

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

#Predict the response for test dataset

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

print(model.score(X\_test,y\_test))

**Dataset Summary:** Wine dataset is used for this experiment. This dataset contains 2 classes, 178 records with 13 different features ('Alcohol', 'Malic acid', 'Ash', 'Alcalinity of ash', 'Magnesium', 'Total phenols', 'Flavanoids', 'Nonflavanoid phenols', 'Proanthocyanins', 'Color intensity', 'Hue', 'OD280/OD315 of diluted wines', 'Proline' ).

**Expected output: Accuracy: 98%**

**#ploynomial kernel:**

#Import Library

from sklearn import svm

# Create SVM classification object

model = svm.SVC(kernel='poly', degree=3, coef0=5, C=100, gamma = 0.01)

#Train the model using the training sets

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

#Predict the response for test dataset

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

print(model.score(X\_test,y\_test))

**Expected output: Accuracy: 100%**

**#RBF kernel:**

#Import Library

from sklearn import svm

# Create SVM classification object

model = svm.SVC(kernel='rbf', C=1, gamma=0.00213)

#Train the model using the training sets

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

#Predict the response for test dataset

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

print(model.score(X\_test,y\_test))

**Expected output: Accuracy: 74%**

**#Sigmoid kernel:**

#Import Library

from sklearn import svm

# Create SVM classification object

model = svm.SVC(kernel='sigmoid', C=100, gamma=0.001, degree=3)

#Train the model using the training sets

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

#Predict the response for test dataset

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

print(model.score(X\_test,y\_test))

**Expected output: Accuracy: 40%**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Kernel | | | |
|  | Linear | Poly | Sigmoid | RBF |
| Accuracy | 98% | 100% | 40% | 74% |

**Tuning parameters of SVM:**

Parameters of SVC() function:

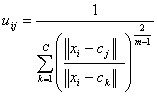
* **C** : float, optional (default=1.0), Penalty parameter C of the error term.
* **kernel** : string, optional (default=’rbf’), Specifies the kernel type to be used in the algorithm. It must be one of ‘linear’, ‘poly’, ‘rbf’, ‘sigmoid’, ‘precomputed’. If none is given, ‘rbf’ is used.
* **degree**: int, optional (default=3), Degree of kernel function. It is significant only in ‘poly’ and ‘sigmoid’.
* **gamma** : float, optional (default=0.0), Kernel coefficient for ‘rbf’ and ‘poly’. If gamma is 0.0 then 1/n\_features will be used instead.
* **coef0** : float, optional (default=0.0), Independent term in kernel function. It is only significant in ‘poly’ and ‘sigmoid’.
* **probability: boolean, optional (default=False)** : Whether to enable probability estimates. This must be enabled prior to calling predict\_proba.
* **tol: float, optional (default=1e-3)** : Tolerance for stopping criterion.

**Fuzzy C-Means (FCM):**

The Fuzzy c-means (FCM) is a method of clustering which allows one piece of data to belong to two or more clusters. It is based on minimization of the following objective function:

https://home.deib.polimi.it/matteucc/Clustering/tutorial_html/images/image019.gif,     https://home.deib.polimi.it/matteucc/Clustering/tutorial_html/images/image021.gif

where m is any real number greater than 1, uij is the degree of membership of xi in the cluster j, xi is the ith of d-dimensional measured data, cj is the d-dimension center of the cluster, and ||\*|| is any norm expressing the similarity between any measured data and the center.  
Fuzzy partitioning is carried out through an iterative optimization of the objective function shown above, with the update of membership uij and the cluster centers cj by:

     ,     

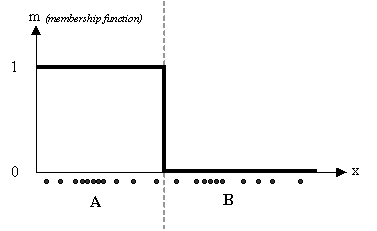
This iteration will stop when https://home.deib.polimi.it/matteucc/Clustering/tutorial_html/images/image027.gif, where https://home.deib.polimi.it/matteucc/Clustering/tutorial_html/images/image002.gif is a termination criterion between 0 and 1, whereas k are the iteration steps. This procedure converges to a local minimum or a saddle point of Jm.  
The algorithm is composed of the following steps:

|  |
| --- |
| 1. Initialize U=[uij] matrix, U(0) 2. At k-step: calculate the centers vectors C(k)=[cj] with U(k)*https://home.deib.polimi.it/matteucc/Clustering/tutorial_html/images/image025.gif* 3. Update U(k) , U(k+1)*https://home.deib.polimi.it/matteucc/Clustering/tutorial_html/images/image023.gif* 4. If || U(k+1) - U(k)||<*https://home.deib.polimi.it/matteucc/Clustering/tutorial_html/images/image002.gif* then STOP; otherwise return to step 2. |

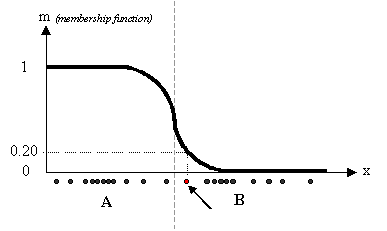
**Remarks**  
As already told, data are bound to each cluster by means of a Membership Function, which represents the fuzzy behavior of this algorithm. To do that, we simply have to build an appropriate matrix named U whose factors are numbers between 0 and 1, and represent the degree of membership between data and centers of clusters.  
For a better understanding, we may consider this simple mono-dimensional example. Given a certain data set, suppose to represent it as distributed on an axis. The figure below shows this:

https://home.deib.polimi.it/matteucc/Clustering/tutorial_html/images/image031.gif

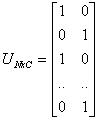
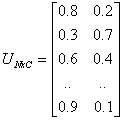
Looking at the picture, we may identify two clusters in proximity of the two data concentrations. We will refer to them using ‘A’ and ‘B’. In the k-means algorithm - each datum is associated to a specific centroid; therefore, this membership function looks like this:



In the FCM approach, instead, the same given datum does not belong exclusively to a well defined cluster, but it can be placed in a middle way. In this case, the membership function follows a smoother line to indicate that every datum may belong to several clusters with different values of the membership coefficient.



In the figure above, the datum shown as a red marked spot belongs more to the B cluster rather than the A cluster. The value 0.2 of ‘m’ indicates the degree of membership to A for such datum. Now, instead of using a graphical representation, we introduce a matrix U whose factors are the ones taken from the membership functions:

(a)                                  (b)

The number of rows and columns depends on how many data and clusters we are considering. More exactly we have C = 2 columns (C = 2 clusters) and N rows, where C is the total number of clusters and N is the total number of data. The generic element is so indicated: uij.  
In the examples above we have considered the k-means (a) and FCM (b) cases. We can notice that in the first case (a) the coefficients are always unitary. It is so to indicate the fact that each datum can belong only to one cluster. Other properties are shown below:

* https://home.deib.polimi.it/matteucc/Clustering/tutorial_html/images/image041.gif
* https://home.deib.polimi.it/matteucc/Clustering/tutorial_html/images/image043.gif
* https://home.deib.polimi.it/matteucc/Clustering/tutorial_html/images/image045.gif

**Python Implementation**

Fuzzy c-means clustering is accomplished via skfuzzy.cmeans, and the output from this function can be repurposed to classify new data according to the calculated clusters (also known as *prediction*) viaskfuzzy.cmeans\_predict

from \_\_future\_\_ import division, print\_function

import numpy as np

import matplotlib.pyplot as plt

import skfuzzy as fuzz

colors = ['b', 'orange', 'g', 'r', 'c', 'm', 'y', 'k', 'Brown', 'ForestGreen']

# Define three cluster centers

centers = [[4, 2],

[1, 7],

[5, 6]]

# Define three cluster sigmas in x and y, respectively

sigmas = [[0.8, 0.3],

[0.3, 0.5],

[1.1, 0.7]]

# Generate test data

np.random.seed(42) # Set seed for reproducibility

xpts = np.zeros(1)

ypts = np.zeros(1)

labels = np.zeros(1)

for i, ((xmu, ymu), (xsigma, ysigma)) in enumerate(zip(centers, sigmas)):

xpts = np.hstack((xpts, np.random.standard\_normal(200) \* xsigma + xmu))

ypts = np.hstack((ypts, np.random.standard\_normal(200) \* ysigma + ymu))

labels = np.hstack((labels, np.ones(200) \* i))

# Visualize the test data

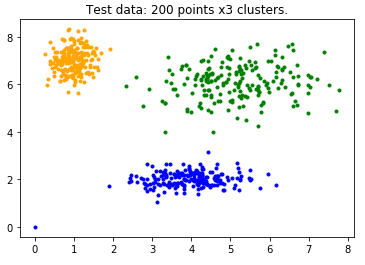
fig0, ax0 = plt.subplots()

for label in range(3):

ax0.plot(xpts[labels == label], ypts[labels == label], '.',

color=colors[label])

ax0.set\_title('Test data: 200 points x3 clusters.')

****

Assume we didn’t know how many clusters we should expect or the data were not so clearly clustered. Let’s try clustering data several times, with between 2 and 9 clusters.

# Set up the loop and plot

fig1, axes1 = plt.subplots(3, 3, figsize=(8, 8))

alldata = np.vstack((xpts, ypts))

fpcs = []

for ncenters, ax in enumerate(axes1.reshape(-1), 2):

cntr, u, u0, d, jm, p, fpc = fuzz.cluster.cmeans(

alldata, ncenters, 2, error=0.005, maxiter=1000, init=None)

# Store fpc values for later

fpcs.append(fpc)

# Plot assigned clusters, for each data point in training set

cluster\_membership = np.argmax(u, axis=0)

for j in range(ncenters):

ax.plot(xpts[cluster\_membership == j],

ypts[cluster\_membership == j], '.', color=colors[j])

# Mark the center of each fuzzy cluster

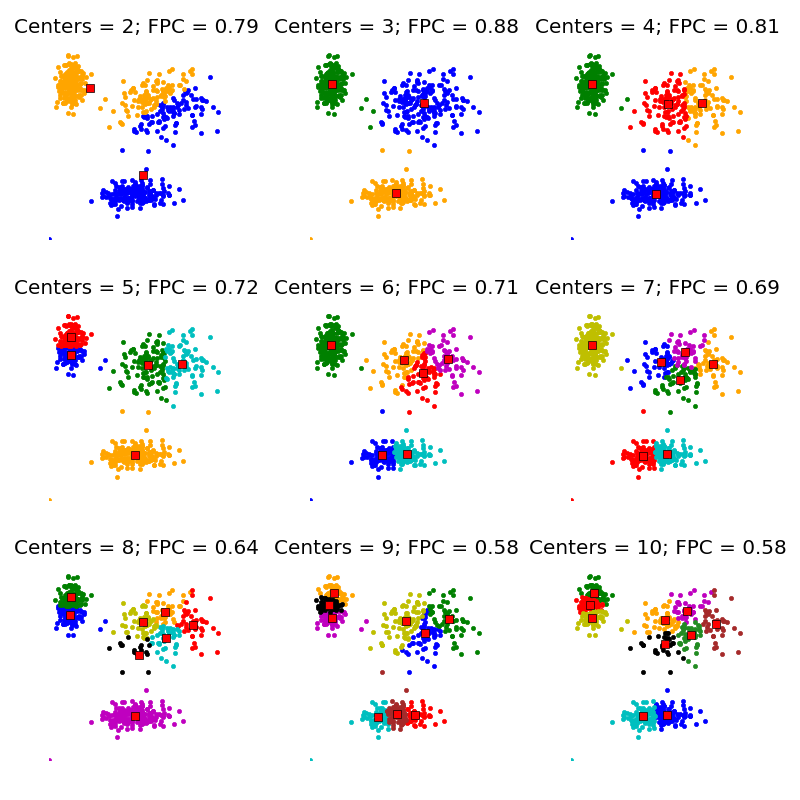
for pt in cntr:

ax.plot(pt[0], pt[1], 'rs')

ax.set\_title('Centers = {0}; FPC = {1:.2f}'.format(ncenters, fpc))

ax.axis('off')

fig1.tight\_layout()



**Building FCM model**

*# Regenerate fuzzy model with 3 cluster centers - note that center ordering*

*# is random in this clustering algorithm, so the centers may change places*

cntr, u\_orig, \_, \_, \_, \_, \_ = fuzz.cluster.cmeans(

alldata, 3, 2, error=0.005, maxiter=1000)

*# Show 3-cluster model*

fig2, ax2 = plt.subplots()

ax2.set\_title('Trained model')

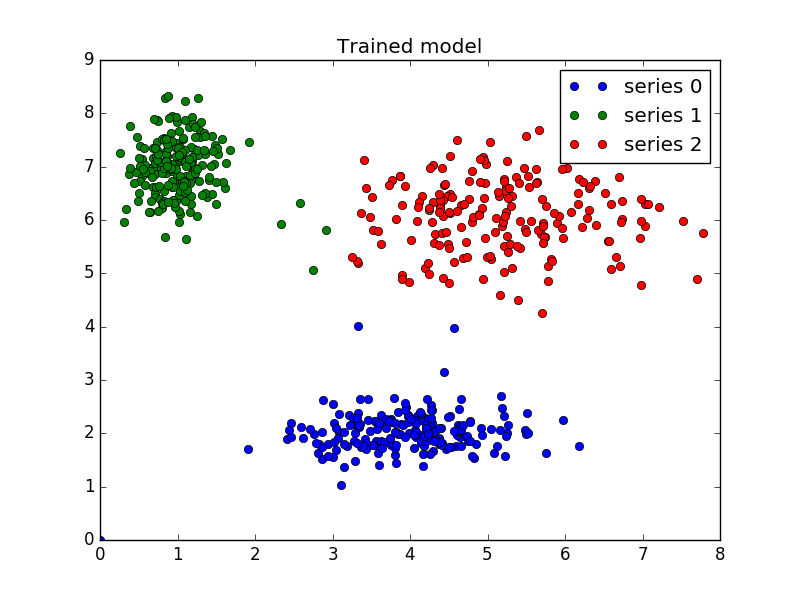
for j in range(3):

ax2.plot(alldata[0, u\_orig.argmax(axis=0) == j],

alldata[1, u\_orig.argmax(axis=0) == j], 'o',

label='series ' + str(j))

ax2.legend()



**Prediction:**

# *Generate uniformly sampled data spread across the range [0, 10] in x and y*

newdata = np.random.uniform(0, 1, (1100, 2)) \* 10

*# Predict new cluster membership with `cmeans\_predict` as well as*

*# `cntr` from the 3-cluster model*

u, u0, d, jm, p, fpc = fuzz.cluster.cmeans\_predict(

newdata.T, cntr, 2, error=0.005, maxiter=1000)

*# Plot the classified uniform data. Note for visualization the maximum*

*# membership value has been taken at each point (i.e. these are hardened,*

*# not fuzzy results visualized) but the full fuzzy result is the output*

*# from cmeans\_predict.*

cluster\_membership = np.argmax(u, axis=0) *# Hardening for visualization*

fig3, ax3 = plt.subplots()

ax3.set\_title('Random points classifed according to known centers')

for j in range(3):

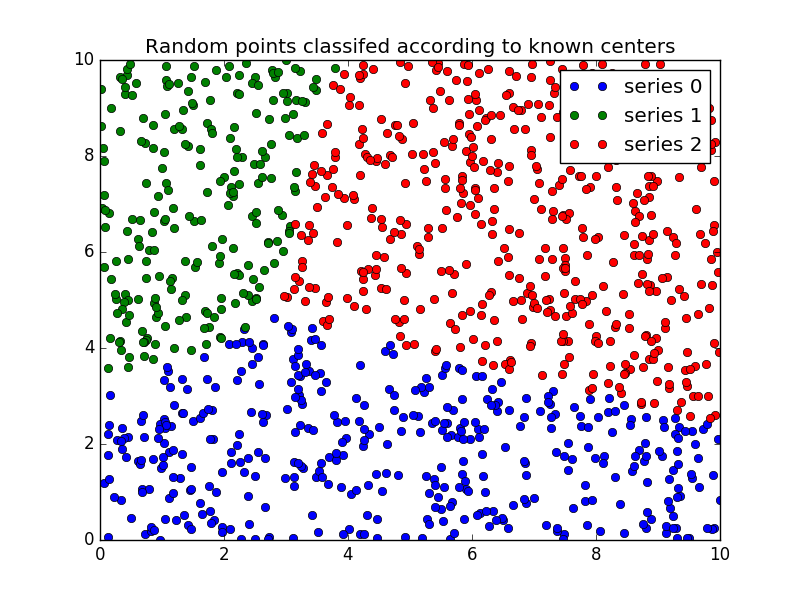
ax3.plot(newdata[cluster\_membership == j, 0],

newdata[cluster\_membership == j, 1], 'o',

label='series ' + str(j))

ax3.legend()

plt.show()

****

**Books:**

**Earl Gose, Richard Johnsonbaugh, “Pattern Recognition and Image Analysis”, 1st Edition, Prentice Hall of India Private limited, 2009.**