Cognixia

A Collabera LEARNING SOLUTIONS COMPANY

# Introduction to Dimensionality Reduction

# Agenda

1. Dimensionality Reduction
2. Principal component analysis (PCA)
3. Linear discriminant analysis (LDA)

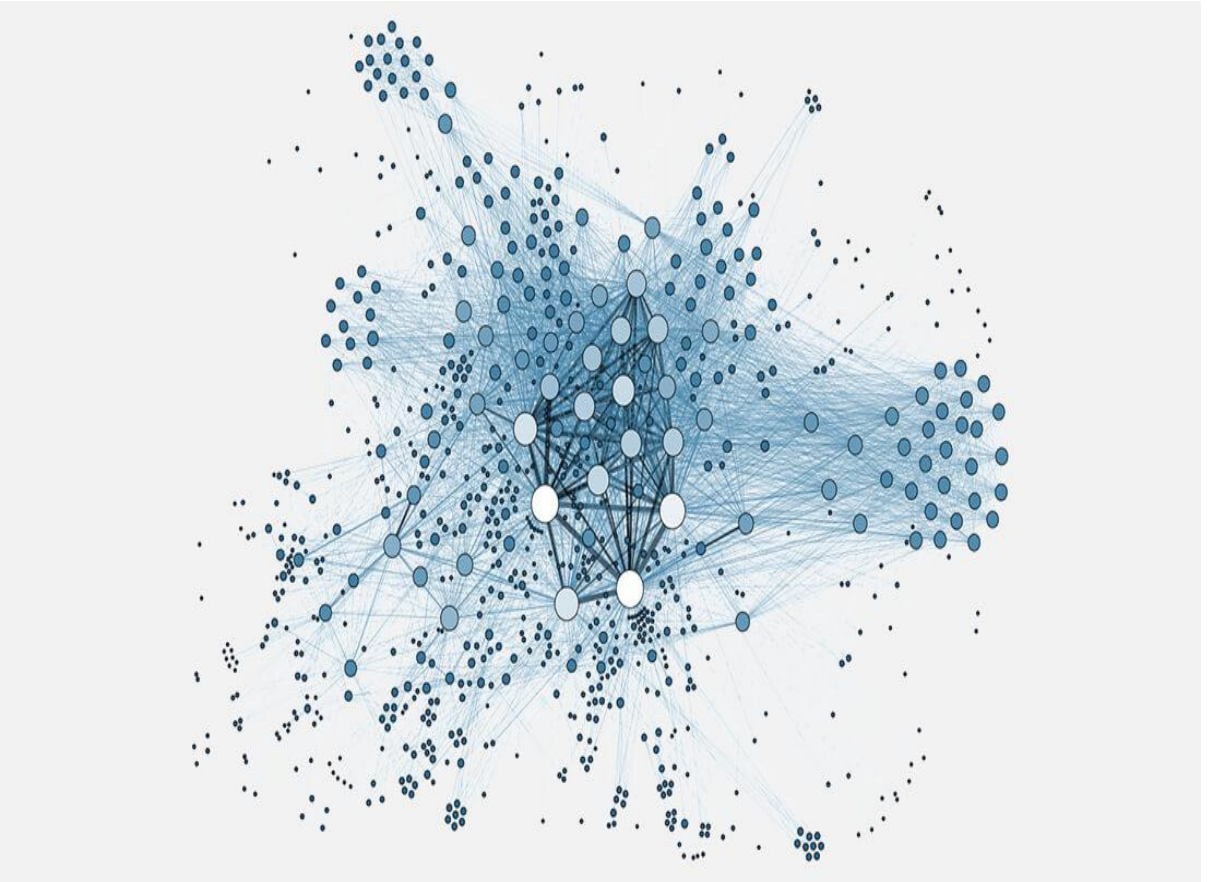
# Dimensionality Reduction

# Why it is needed?

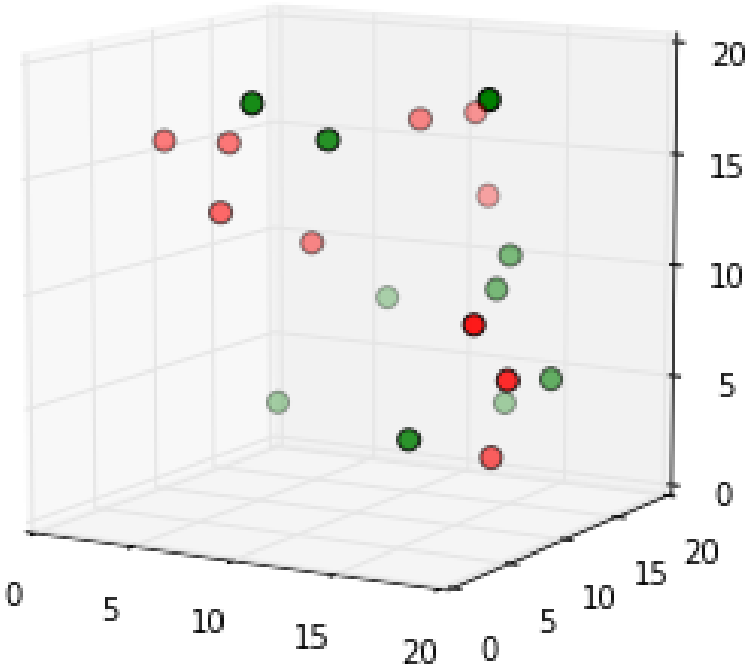
Dataset is very large and there are many redundant fields?

## Points of Action:

- Remove redundant data to make data classification easier.
- Improve accuracy of the prediction algorithm.
- Use components with high variance to make right predictions.
- From a theoretical point of view, increasing the number of features should lead to better performance.
- In practice, the inclusion of more features leads to worse performance (i.e., curse of dimensionality).
- The number of training examples required increases exponentially with dimensionality.



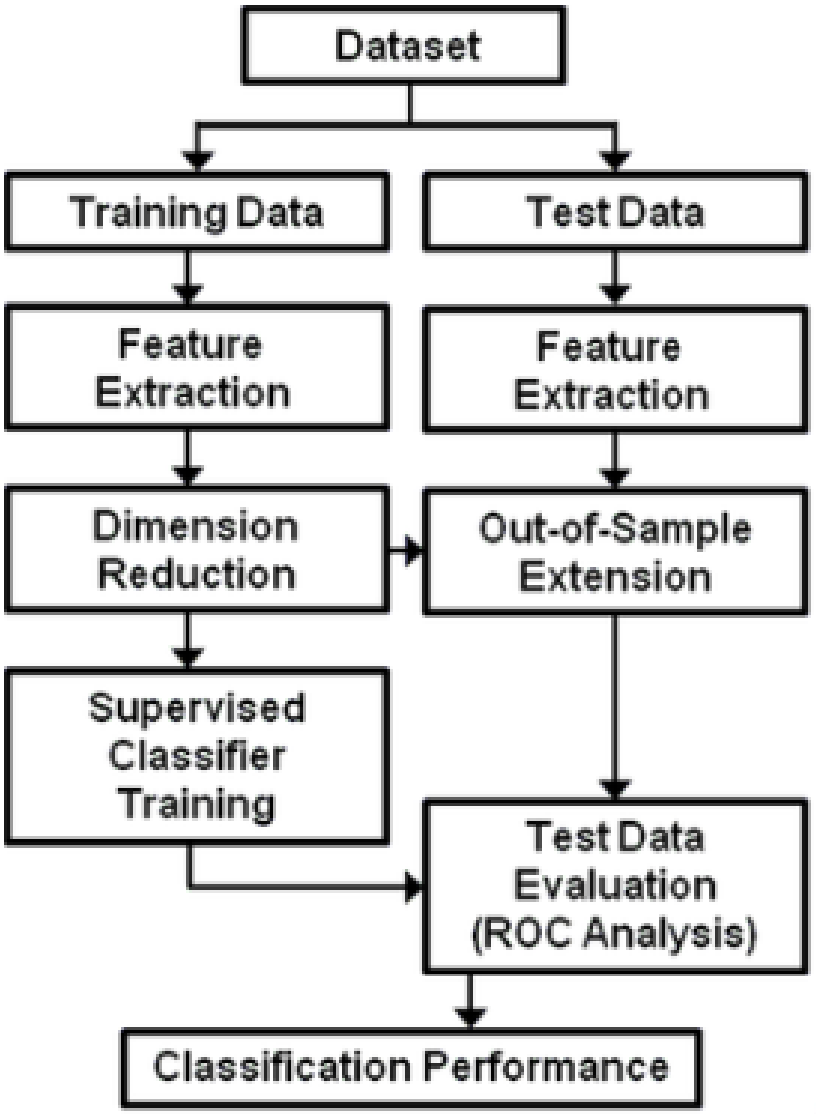
# What is Dimensionality Reduction?




Dimensionality Reduction

- *Dimension reduction is the process of reducing the number of random variables under consideration by obtaining a set of principal variables.*
- *Helps in data compressing and reducing the storage space required.*
- *It fastens the time required for performing same computations.*
- *Reducing the dimensions of data to 2D or 3D may allow us to plot and visualize it precisely.*

# Dimensionality Reduction in Classification



# Dimensionality Reduction in Classification

- Significant improvements can be achieved by first mapping the data into a *lower-dimensional* space.

$$x = \begin{bmatrix} a_1 \\ a_2 \\ \dots \\ a_N \end{bmatrix} --> reduce\ dimensionality --> y = \begin{bmatrix} b_1 \\ b_2 \\ \dots \\ b_K \end{bmatrix} \quad (K << N)$$

- Dimensionality can be reduced by:
  - Combining features using a linear or non-linear transformations.
  - Selecting a subset of features (i.e., feature selection).
- Linear combinations are particularly attractive because they are simple to compute and analytically tractable.
- Given $x \in R^N$, the goal is to find an N x K matrix U such that:

$$y = U^T x \in R^K \text{ where } K<<N$$

# Dimensionality Reduction in Classification

Represent data in terms of basis vectors in a lower dimensional space (embedded within the original space).

(1) Higher-dimensional space representation:

$$x = a_1 v_1 + a_2 v_2 + \cdots + a_N v_N$$

$v_1, v_2, ..., v_N$ is a basis of the $N$-dimensional space

$$x = \begin{bmatrix} a_1 \\ a_2 \\ \cdots \\ a_N \end{bmatrix}$$

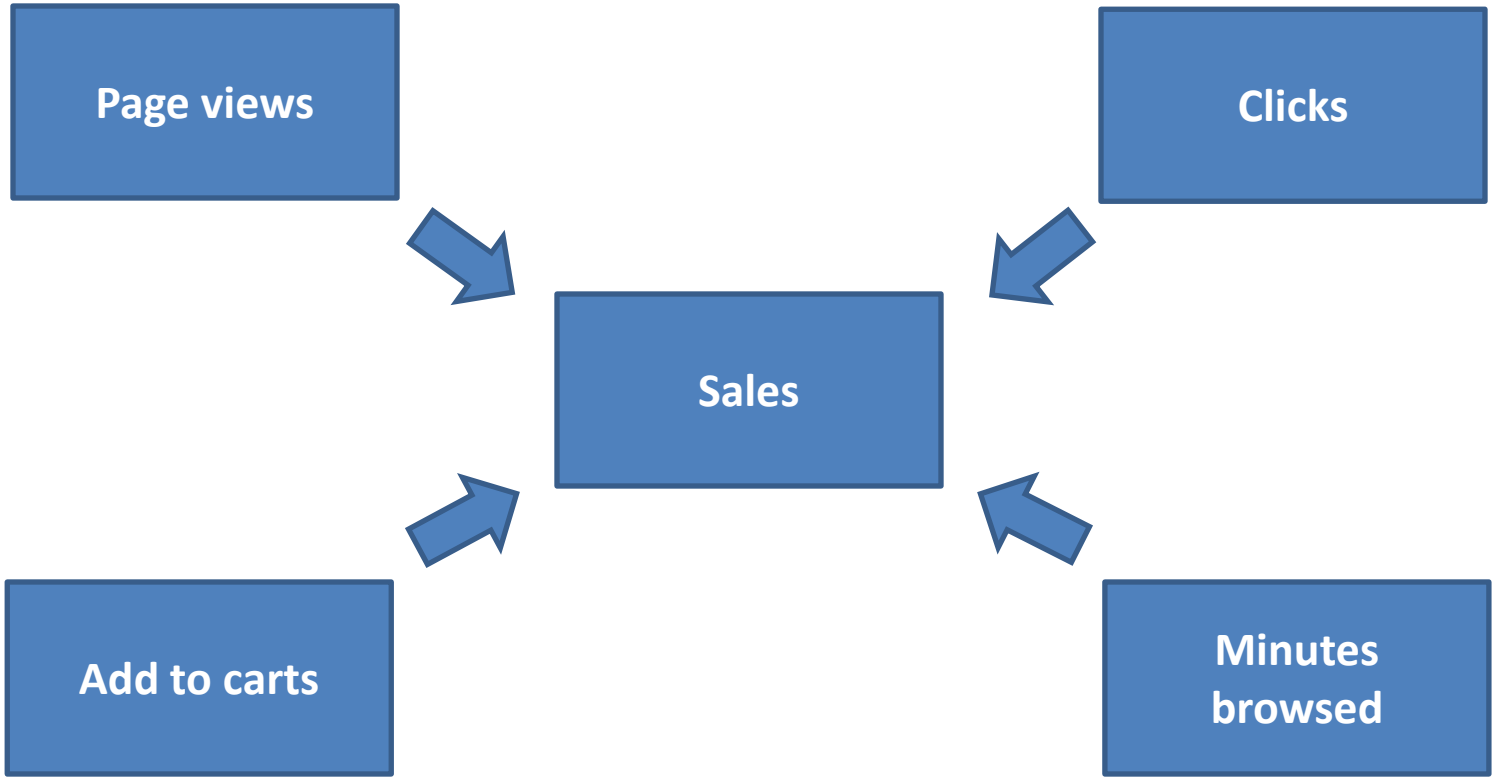
(2) Lower-dimensional sub-space representation:

$$\hat{x} = b_1 u_1 + b_2 u_2 + \cdots + b_K u_K$$

$u_1, u_2, ..., u_K$ is a basis of the $K$-dimensional space

$$y = \begin{bmatrix} b_1 \\ b_2 \\ \cdots \\ b_K \end{bmatrix}$$

# An Example

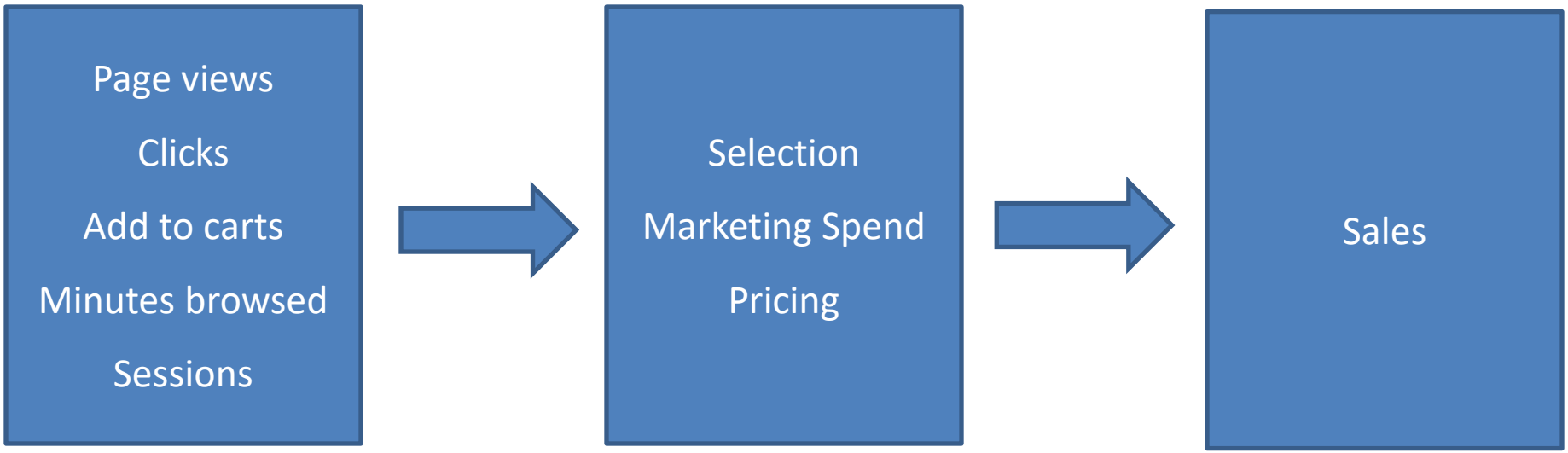



Consider an example, where you want to figure out some relationship between some observable variables and another observable outcome like Sales

# Few underlying causes

Selection, Marketing spend, pricing may be some of the drivers leading to observable behaviour from the users.

Ultimately, converging to sales




Many observed causes

Few underlying causes

Effect

# Problem of Multicollinearity

It might be possible that various independent variables are correlated to each other, containing the same information and affecting the accuracy of the model.

Solution: By factor analysis and extract underlying causes leading to this behaviour will be beneficial and leads to the high R square value.

# Types of Dimensionality Reduction

**Principal Component Analysis (PCA):**

Principal component analysis, performs a linear mapping of the data to a lower-dimensional space in such a way that the variance of the data in the low-dimensional representation is maximized. PCA performs a linear transformation moving the original set of features to a new space composed by principal component.


**Linear Discriminant Analysis (LDA):**

Linear discriminant analysis (LDA) is a generalization of Fisher's linear discriminant, a method used in statistics, pattern recognition and machine learning to find a linear combination of features that characterizes or separates two or more classes of objects or events. The goal is to project a dataset onto a lower-dimensional space with good class-separability in order avoid over-fitting and also reduce computational costs.

# Principal Component Analysis

Step 1: Normalize the data

First step is to normalize the data that we have so that PCA works properly. This is done by subtracting the respective means from the numbers in the respective column. So if we have two dimensions X and Y, all X become $\boldsymbol{x}$- and all Y become $\boldsymbol{y}$-. This produces a dataset whose mean is zero.

Step 2: Calculate the covariance matrix

For a 2-dimensional dataset, the result is a 2x2 Covariance matrix.

$$Matrix(Covariance) = \begin{bmatrix} Var[X_1] & Cov[X_1, X_2] \\ Cov[X_2, X_1] & Var[X_2] \end{bmatrix}$$

Please note that $Var[X_1]$= $Cov[X_1, X_1]$ and $Var[X_2]$= $Cov[X_2, X_2]$.

Step 3: Calculate the eigenvalues and eigenvectors

Next step is to calculate the eigenvalues and eigenvectors for the covariance matrix. The same is possible because it is a square matrix. ƛ is an eigenvalue for a matrix A if it is a solution of the characteristic equation:

$$\det( ƛI - A ) = 0$$

Where, I is the identity matrix of the same dimension as A which is a required condition for the matrix subtraction as well in this case and 'det' is the determinant of the matrix. For each eigenvalue ƛ, a corresponding eigen-vector v, can be found by solving:

$$( ƛI - A )v = 0$$

# Principal Component Analysis

Step 4: Choosing components and forming a feature vector:

We order the eigenvalues from largest to smallest so that it gives us the components in order or significance. Here comes the dimensionality reduction part.

If we have a dataset with n variables, then we have the corresponding n eigenvalues and eigenvectors. It turns out that the eigenvector corresponding to the highest eigenvalue is the principal component of the dataset and it is our call as to how many eigenvalues we choose to proceed our analysis with. To reduce the dimensions, we choose the first p eigenvalues and ignore the rest. We do lose out some information in the process, but if the eigenvalues are small, we do not lose much.

Next we form a feature vector which is a matrix of vectors, in our case, the eigenvectors.

**Feature Vector = (eig1, eig2)**

Step 5: Forming Principal Components:

This is the final step where we actually form the principal components using all the math we did till here.

For the same, we take the transpose of the feature vector and left-multiply it with the transpose of scaled version of original dataset.

NewData = FeatureVector$^T$ X ScaledData$^T$

# Principal Component Analysis

Here,

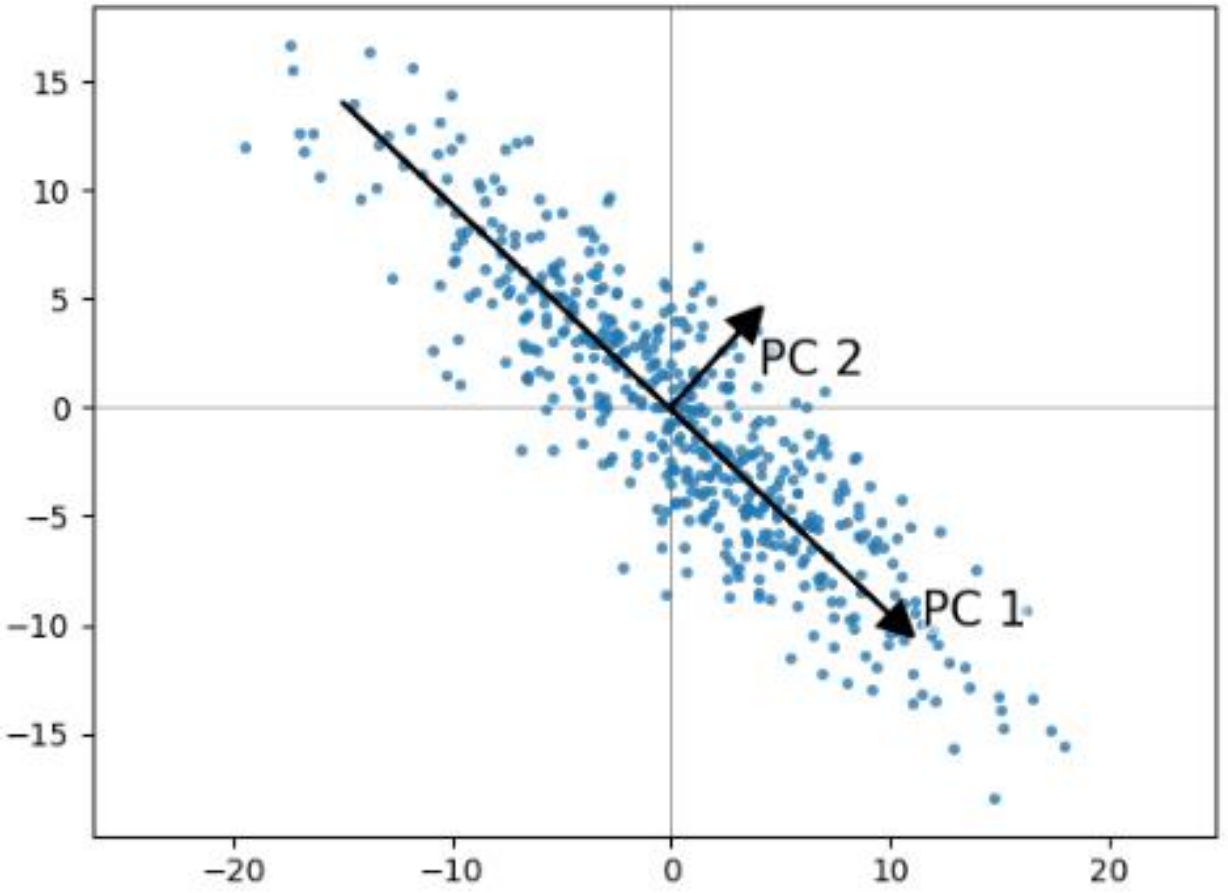
NewData: is the Matrix consisting of the principal components,

FeatureVector: is the matrix we formed using the eigenvectors we chose to keep, and

ScaledData: is the scaled version of original dataset

('T' in the superscript denotes transpose of a matrix which is formed by interchanging the rows to columns and vice versa. In particular, a 2x3 matrix has a transpose of size 3x2)

# Identifying two principal components in a set of random points



# Limitations of PCA

- Background changes cause problems
- Light changes degrade performance
- Performance decreases quickly with changes to face size
- Performance decreases with changes to face orientation (but not as fast as with scale changes)
- PCA is not always an optimal dimensionality-reduction technique for classification purposes.

# Linear Discriminant Analysis (LDA)

**The LDA makes some simplifying assumptions about your data:**

- That your data is Gaussian, that each variable is shaped like a bell curve when plotted.
- That each attribute has the same variance, that values of each variable vary around the mean by the same amount on average.

With these assumptions, the LDA model estimates the mean and variance from your data for each class. Take example of univariate (single input variable) case with two classes.

The mean (mu) value of each input (x) for each class (k) can be estimated in the normal way by dividing the sum of values by the total number of values.

$$mu^k = \frac{1}{n^k} * \text{sum(x)}$$

# Linear Discriminant Analysis (LDA)

Where $mu^k$ is the mean value of x for the class k, $n^k$ is the number of instances with class k. The variance is calculated across all classes as the average squared difference of each value from the mean.

$$sigma^2 = \frac{1}{n-k} * \text{sum}((x - mu)^2)$$

Where:

- $sigma^2$ is the variance across all inputs (x).
- n is the number of instances.
- K is the number of classes.
- And mu is the mean for input x.

# Linear Discriminant Analysis (LDA)

**Making predictions with LDA**

LDA makes predictions by estimating the probability that a new set of inputs belongs to each class. The class that gets the highest probability is the output class and a prediction is made.

The model uses Bayes Theorem to estimate the probabilities. Briefly Bayes' Theorem can be used to estimate the probability of the output class (k) given the input (x) using the probability of each class and the probability of the data belonging to each class:

$$P(Y=x|X=x) = (PIk * fk(x)) / sum(PIl * fl(x))$$

Where PIk refers to the base probability of each class (k) observed in your training data (e.g. 0.5 for a 50-50 split in a two class problem.

# Linear Discriminant Analysis (LDA)

In Bayes' Theorem this is called the prior probability.

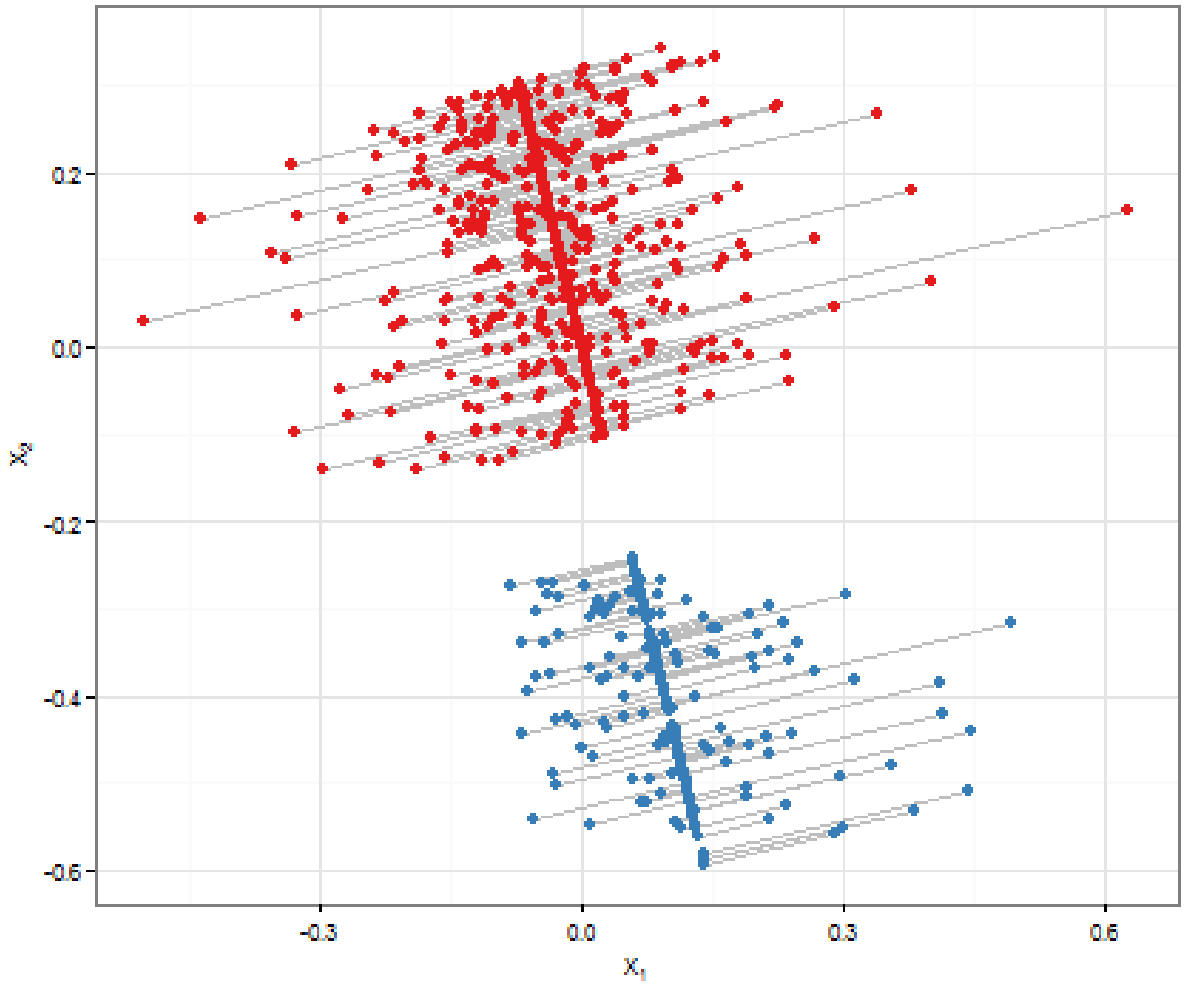
PIk = nk/n

The f(x) above is the estimated probability of x belonging to the class. A Gaussian distribution function is used for f(x). Plugging the Gaussian into the above equation and simplifying we end up with the equation below. This is called a discriminate function and the class is calculated as having the largest value will be the output classification (y):

Dk(x) = x * (muk/siga^2) – (muk^2/(2*sigma^2)) + ln(PIk)

Dk(x) is the discriminate function for class k given input x, the muk, sigma^2 and PIk are all estimated from your data.

# Mapping of 2-D data points onto a 1-D line



# PCA Code

```
# PCA

# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

import os
os.chdir('Give path')
# Importing the dataset
dataset = pd.read_csv('Wine.csv')
X = dataset.iloc[:, 0:13].values
y = dataset.iloc[:, 13].values

# Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 0)
```

# PCA Code

```
# Feature Scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

# Applying PCA
from sklearn.decomposition import PCA
pca = PCA(n_components = 2)
X_train = pca.fit_transform(X_train)
X_test = pca.transform(X_test)
explained_variance = pca.explained_variance_ratio_

# Fitting Logistic Regression to the Training set
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression(random_state = 0)
classifier.fit(X_train, y_train)
```

# PCA Code

```
# Predicting the Test set results
y_pred = classifier.predict(X_test)

# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
```

# LDA Code

```
#LDA

# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

import os
os.chdir('Give Path')
# Importing the dataset
dataset = pd.read_csv('Wine.csv')
X = dataset.iloc[:, 0:13].values
y = dataset.iloc[:, 13].values

# Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 0)
```

# LDA Code

```
# Feature Scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

# Applying LDA
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
lda = LDA(n_components = 2)
X_train = lda.fit_transform(X_train, y_train)
X_test = lda.transform(X_test)

# Fitting Logistic Regression to the Training set
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression(random_state = 0)
classifier.fit(X_train, y_train)
```

# LDA Code

```
# Predicting the Test set results
y_pred = classifier.predict(X_test)

# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
```


Cognixia
A Collabera LEARNING SOLUTIONS COMPANY
THANK YOU
www.cognixia.com