

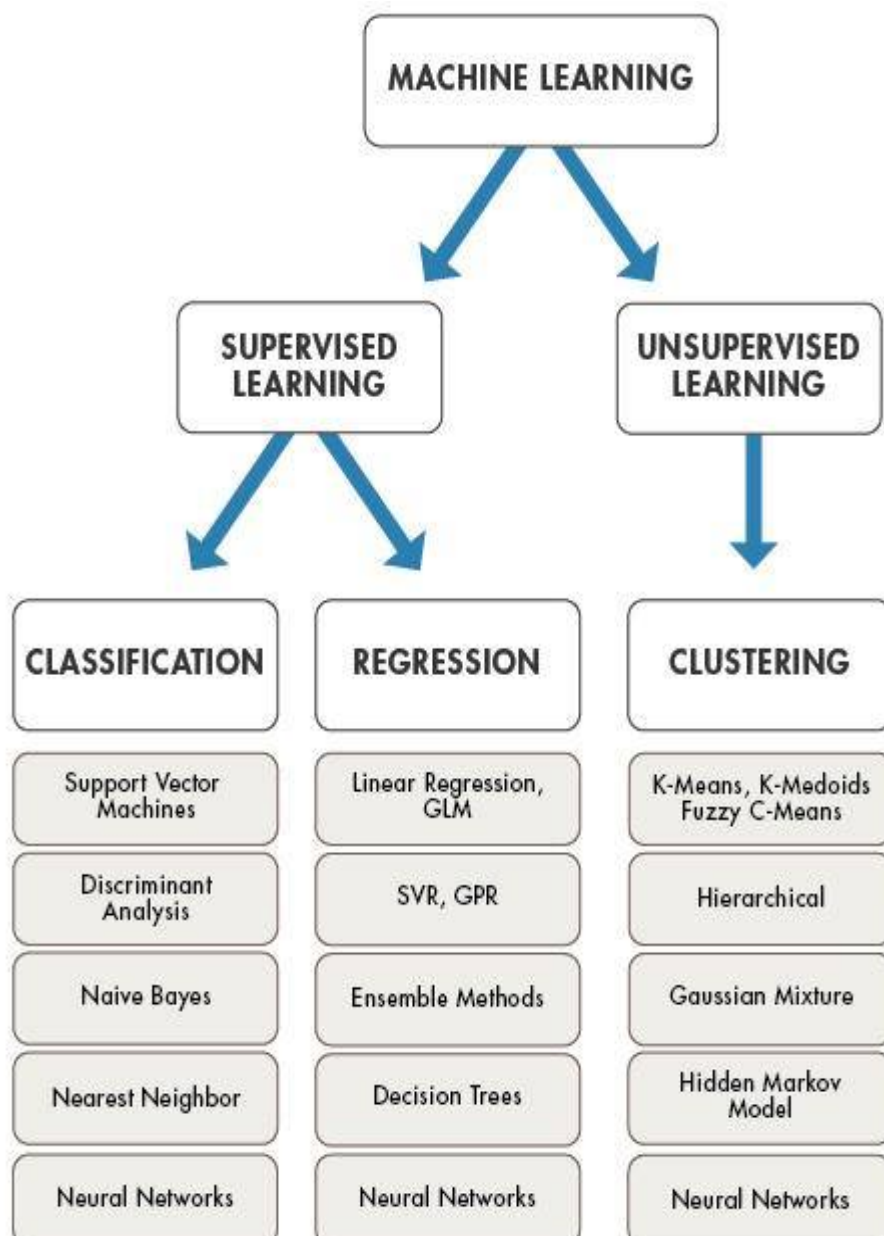
# Machine Learning Tutorial

Kunal Bhashkar  
[kbhashkar@rediffmail.com](mailto:kbhashkar@rediffmail.com)  
+91-8527927040

## 1. INTRODUCTION

Machine learning, a type of artificial intelligence that "learns" as it identifies new patterns in data. It is a computer program which said to learn from experience  $E$  with respect to some class of tasks  $T$  and performance measure  $P$ , if its performance at tasks in  $T$ , as measured by  $P$ , improves with experience  $E$ .

**Categorization :**



## 2. METHODS

### 2.1 Supervised Learning

Supervised learning is where you have input variables (x) and an output variable (Y) and you use an algorithm to learn the mapping function from the input to the output.

$$Y = f(X)$$

The goal is to approximate the mapping function so well that when you have new input data (x) that you can predict the output variables (Y) for that data.

It is called supervised learning because the process of an algorithm learning from the training dataset can be thought of as a teacher supervising the learning process. We know the correct answers, the algorithm iteratively makes predictions on the training data and is corrected by the teacher. Learning stops when the algorithm achieves an acceptable level of performance.

#### Example

##### Supervised Learning with Real Life Example

Suppose you had a basket and filled it with different kinds of fruits.

- Your task is to arrange them into groups.
- For understanding let consider four types of fruits in our basket.

**Apple, Banana, Grape, Cherries**

##### Supervised Learning Algorithm steps

- You already learn from your previous work about the physical characters of fruits
- So arranging the same type of fruits at one place is easy now
- In data mining terminology the earlier work is called as training the data
- You already learn the things from your train data. This is because of response variable
- Response variable means just a decision variable
- You can observe response variable below (FRUIT NAME)

No.	SIZE	COLOR	SHAPE	FRUIT NAME
1	Big	Red	Rounded shape with depression at the top	Apple
2	Small	Red	Heart-shaped to nearly globular	Cherry
3	Big	Green	Long curving cylinder	Banana
4	Small	Green	Round to oval,Bunch shape Cylindrical	Grape

- Suppose you have taken a new fruit from the basket then you will see the size, color, and shape of that particular fruit.

- If size is Big, color is Red, the shape is rounded shape with a depression at the top, you will confirm the fruit name as apple and you will put in apple group.
- Likewise for other fruits also.
- The job of grouping fruits was done and the happy ending.
- You can observe in the table that a column was labeled as “FRUIT NAME“. This is called as a response variable.
- If you learn the thing before from training data and then applying that knowledge to the test data(for new fruit), This type of learning is called as Supervised Learning.

Supervised learning problems can be further grouped into regression and classification problems.

- **Classification:** A classification problem is when the output variable is a category, such as “red” or “blue” or “disease” and “no disease”.
- **Regression:** A regression problem is when the output variable is a real value, such as “dollars” or “weight”.

### 2.1.1 Classification

In machine learning and statistics, **classification** is the problem of identifying to which of a set of categories (sub-populations) a new observation belongs, on the basis of a training set of data containing observations (or instances) whose category membership is known. Classification is an example of pattern recognition.

#### Example

Computer vision (Medical imaging and medical image analysis, Optical character recognition, Video tracking), Drug discovery and development (Toxicogenomics, Quantitative structure-activity relationship), Geostatistics, Speech recognition, Handwriting recognition, Biometric identification, Biological classification, Statistical natural language processing, Document classification, Internet search engines, Credit scoring, Pattern recognition, Micro-array classification

### Classification Algorithm

#### 2.1.1.1. Support Vector Machine

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 which categorizes new examples. In two dimensional space this hyperplane is a line dividing a plane in two parts where in each class lay in either side.

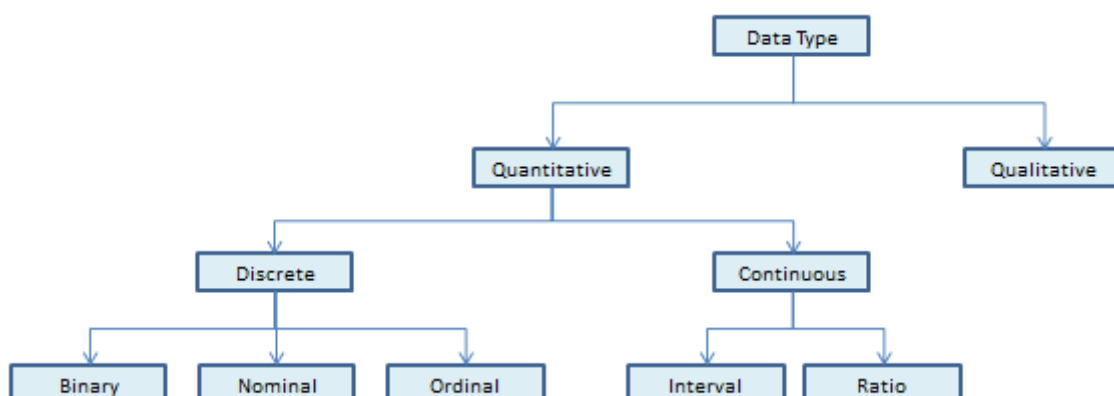
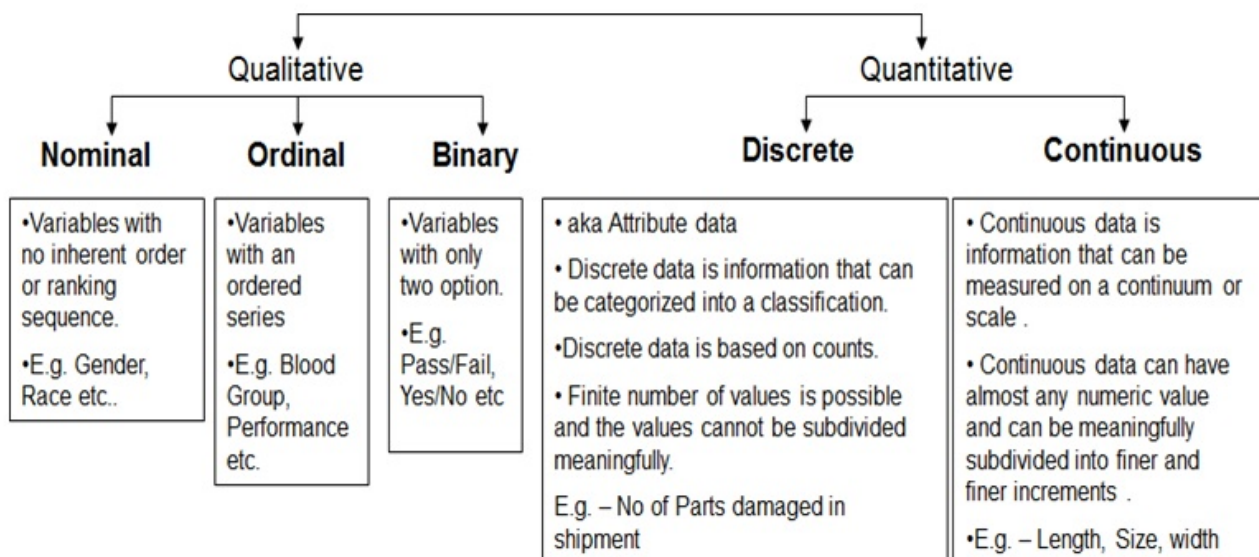
**Note:** Labeled data is a group of samples that have been tagged with one or more labels. For example, labels might indicate whether a photo contains a horse or a cow, which words were uttered in an audio recording, what type of action is being performed in a video, what the topic of a news article is, what the overall sentiment of a tweet is, whether the dot in an x-ray is a tumor, etc.

## Maximal-Margin Classifier

The Maximal-Margin Classifier is a hypothetical classifier in which the numeric input variables ( $x$ ) in your data (the columns) form an  $n$ -dimensional space. For example, if you had two input variables, this would form a two-dimensional space. It works with numeric values, nominal values (zip code, eye colors, ID numbers)

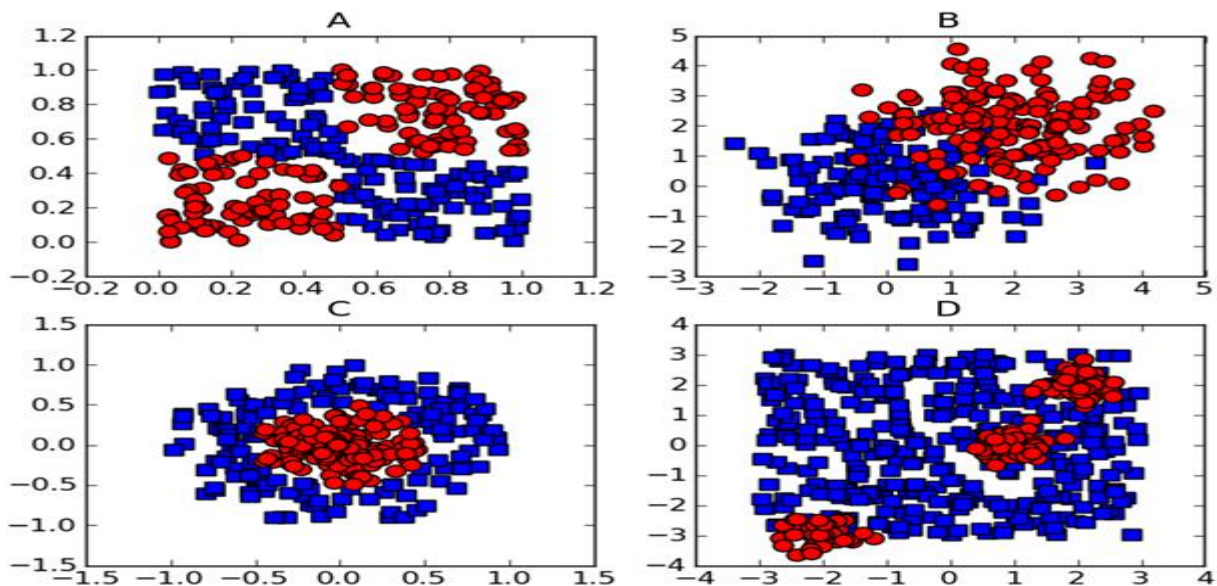
The distance between the line and the closest data points is referred to as the margin. The best or optimal line that can separate the two classes is the line that has the largest margin. This is called the Maximal-Margin hyperplane.

## Data Types



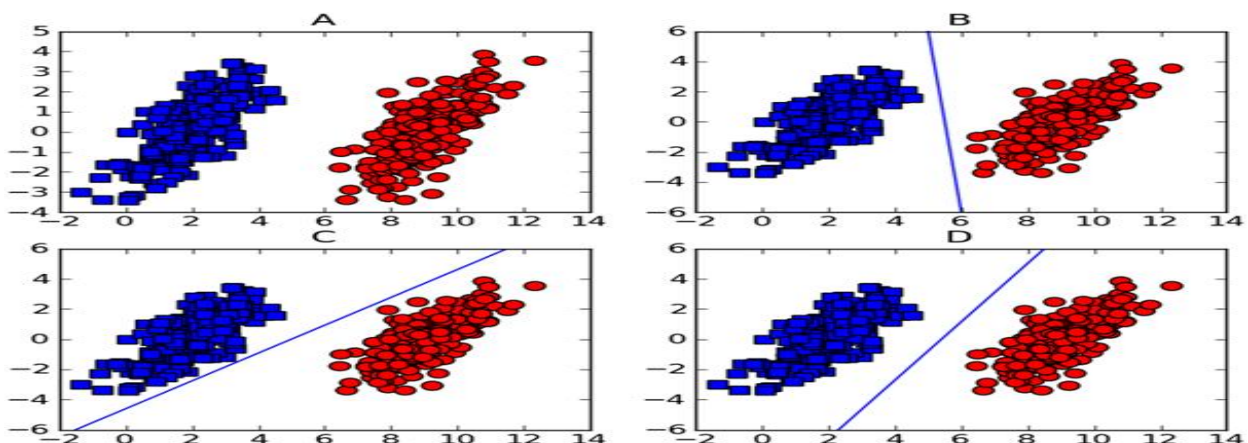
A hyperplane is a line that splits the input variable space. In SVM, a hyperplane is selected to best separate the points in the input variable space by their class, either class 0 or class 1. In two-dimensions you can visualize this as a line.

Consider an examples of datasets that aren't linearly separable



The line used to separate the dataset is called a separating hyperplane and the points closest to the separating hyperplane are known as support vectors. Now that we know that we're trying to maximize the distance from the separating line to the support vectors, we need to find a way to optimize this problem.

Linearly separable data is shown in frame A. Frames B, C, and D show possible valid lines separating the two classes of data



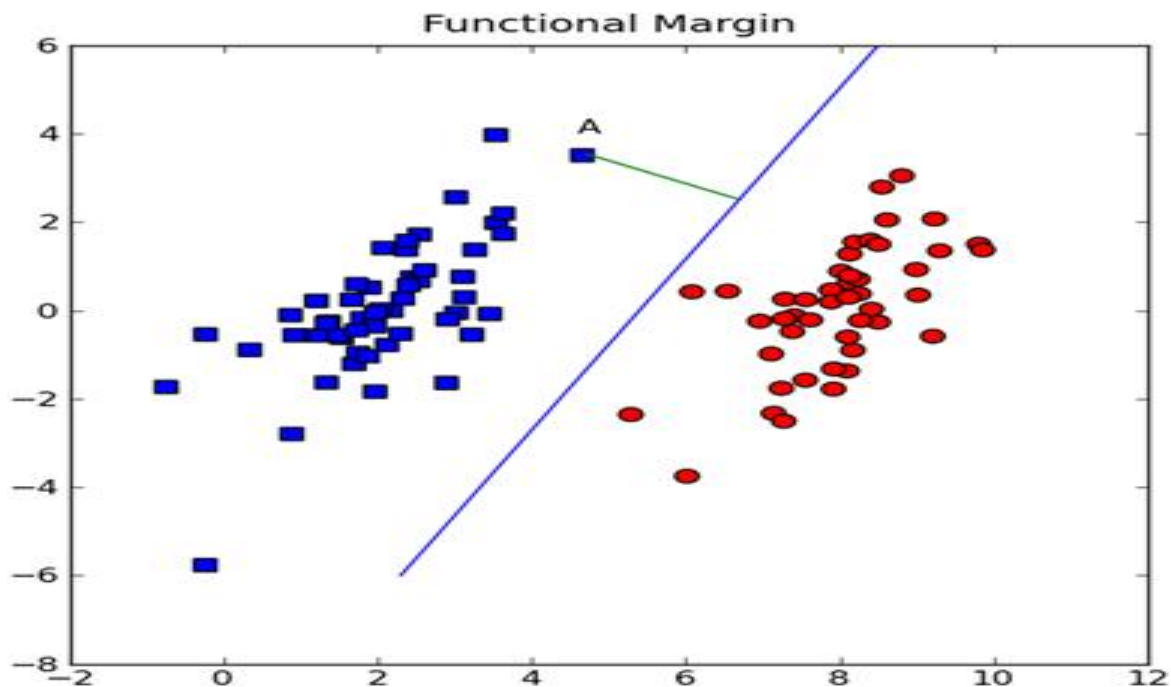
### Finding the maximum margin

The margin is calculated as the perpendicular distance from the line to only the closest points. Only these points are relevant in defining the line and in the construction of the classifier. These points are called the support vectors. They support or define the hyperplane.

The hyperplane is learned from training data using an optimization procedure that maximizes the margin.

Our separating hyperplane has the form  $w^T x + b$ . If we want to find the distance from A to the separating plane, we must measure normal or perpendicular to the line. This is given by  $|w^T x + b| / \|w\|$ . The constant b is just an offset like  $w_0$  in logistic regression. All this w and b stuff describes the separating line, or hyperplane, for our data.

The distance from point A to the separating plane is measured by a line normal to the separating plane.



### General Approach to SVMs

#### General approach to SVMs

1. Collect: Any method.
2. Prepare: Numeric values are needed.
3. Analyze: It helps to visualize the separating hyperplane.
4. Train: The majority of the time will be spent here. Two parameters can be adjusted during this phase.
5. Test: Very simple calculation.
6. Use: You can use an SVM in almost any classification problem. One thing to note is that SVMs are binary classifiers. You'll need to write a little more code to use an SVM on a problem with more than two classes.

### Soft Margin Classifier

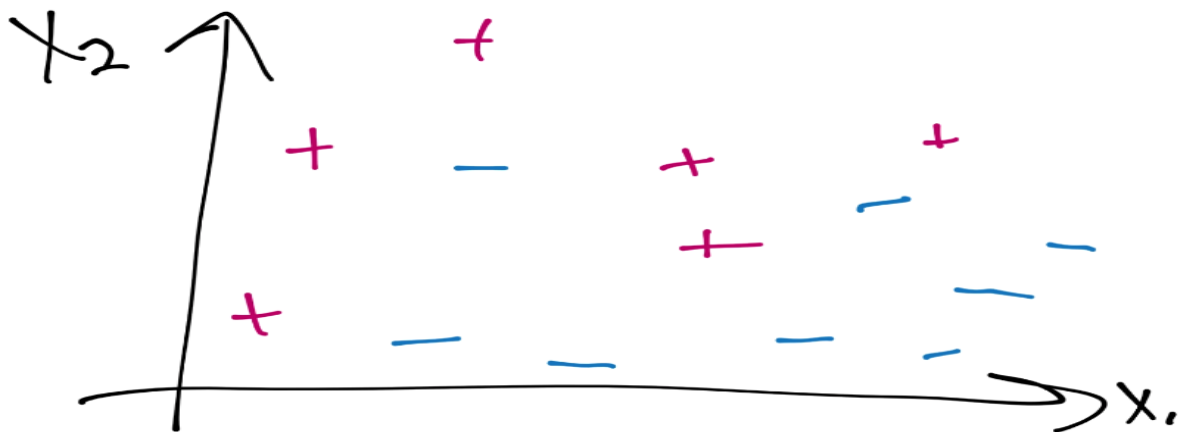
In practice, real data is messy and cannot be separated perfectly with a hyperplane.



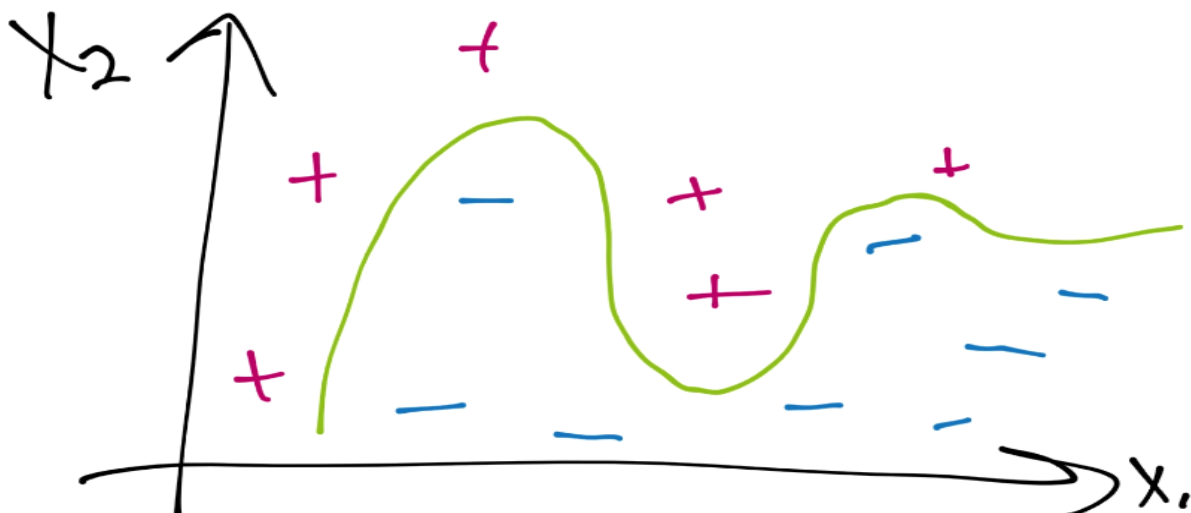
The constraint of maximizing the margin of the line that separates the classes must be relaxed. This is often called the soft margin classifier. This change allows some points in the training data to violate the separating line.

First, there are two major reasons why the soft-margin classifier might be superior. One reason is your data is not perfectly linearly separable, but is very close and it makes more sense to continue using the default linearly kernel. The other reason is, even if you are using a kernel, you may wind up with significant overfitting if you want to use a hard-margin.

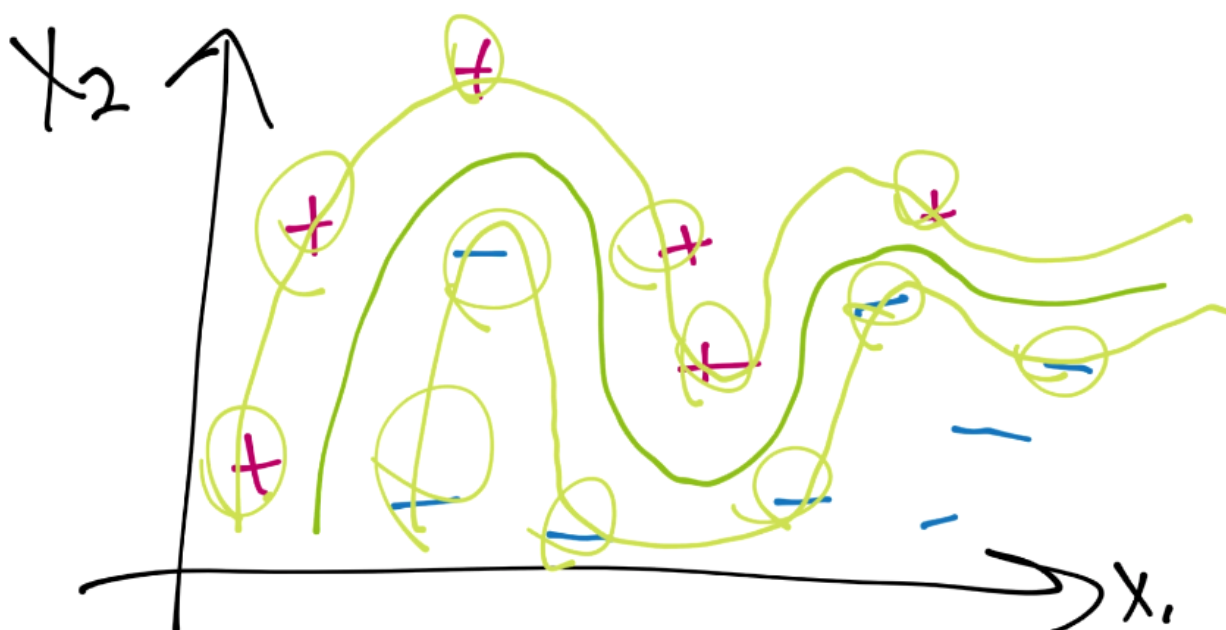
Consider an example



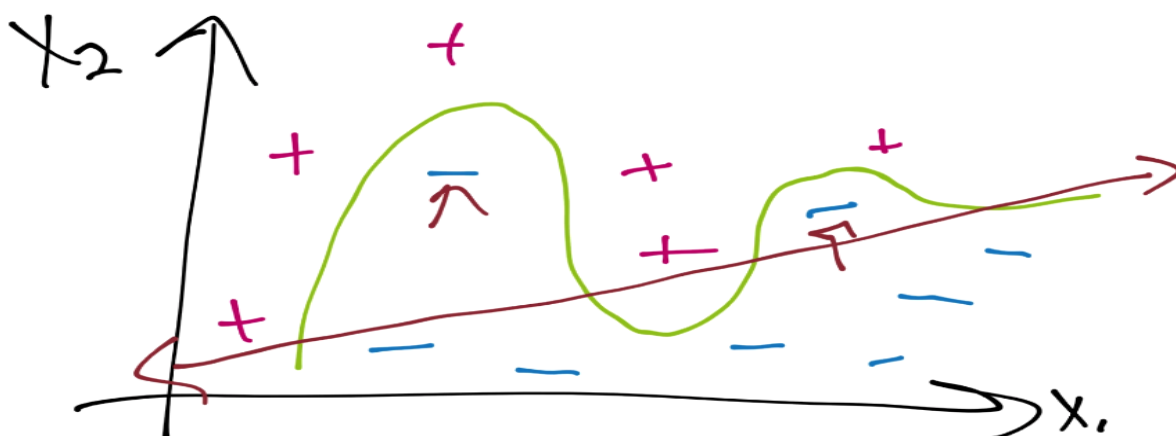
Here's a case of data that isn't currently linearly separable. Assuming a hard-margin, we might use a kernel to achieve a decision boundary of:



draw the support vector hyperplanes, and circle the support vectors:



In this case, every single data sample for the positive class is a support vector, and only two of the negative class aren't support vectors. This signals to use a high chance of overfitting having happened. That's something we want to avoid, since, as we move forward to classify future points,



### Support Vector Machines (Kernels)

Kernel filters the raw data and presents its features to the machine in a way that makes the learning task as simple as possible.

In machine learning, **kernel methods** are a class of algorithms for pattern analysis, whose best known member is the support vector machine (SVM). The general task of pattern analysis is to find and study general types of relations in datasets.



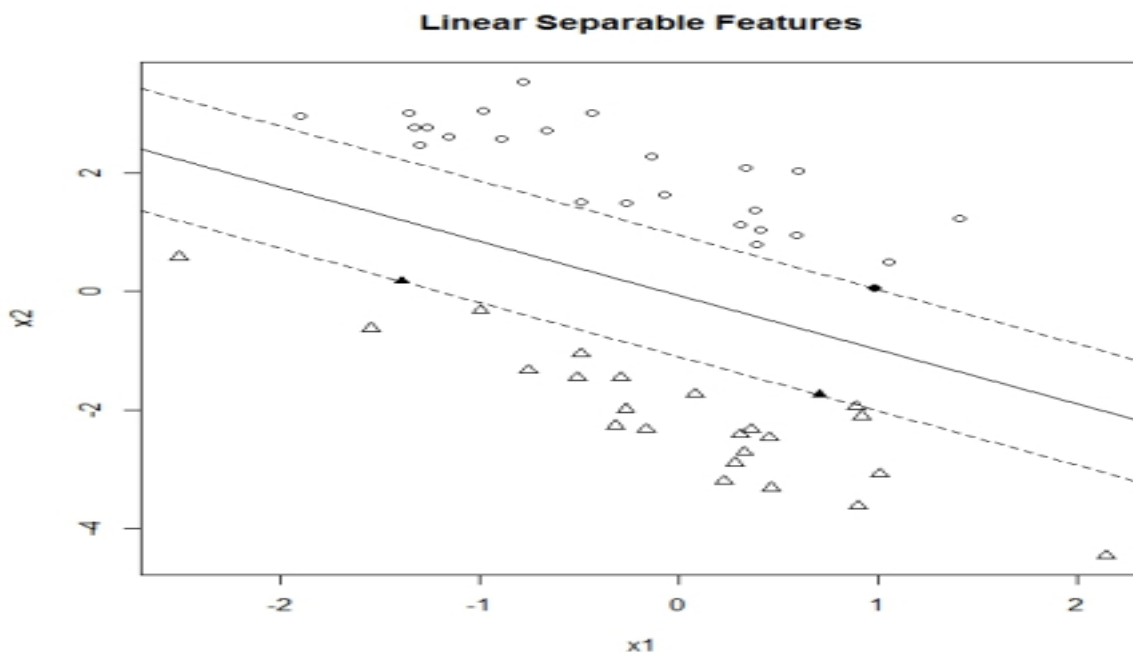
## Linear Kernel SVM

The dot-product is called the kernel and can be re-written as:

$$K(x, x_i) = \text{sum}(x * x_i)$$

Here  $x$  is input and  $x_i$  is support vector

The kernel defines the similarity or a distance measure between new data and the support vectors. The dot product is the similarity measure used for linear SVM or a linear kernel because the distance is a linear combination of the inputs.



## Polynomial Kernel SVM

Instead of the dot-product, we can use a polynomial kernel, for example:

$$K(x, x_i) = 1 + \text{sum}(x * x_i)^d$$

Here  $x$  is input and  $x_i$  is support vector,

Where the degree of the polynomial must be specified by hand to the learning algorithm. When  $d=1$  this is the same as the linear kernel. The polynomial kernel allows for curved lines in the input space.

## Radial Kernel SVM

We can also have a more complex radial kernel. For example:

$$K(x, x_i) = \exp(-\gamma * \text{sum}((x - x_i)^2))$$

Here  $x$  is input and  $x_i$  is support vector,

Where  $\gamma$  is a parameter that must be specified to the learning algorithm. A good default value

for gamma is 0.1, where gamma is often  $0 < \gamma < 1$ . The radial kernel is very local and can create complex regions within the feature space, like closed polygons in two-dimensional space.

### How to Learn a SVM Model

The SVM model needs to be solved using an optimization procedure.

You can use a numerical optimization procedure to search for the coefficients of the hyperplane. This is inefficient and is not the approach used in widely used SVM implementations like LIBSVM. If implementing the algorithm as an exercise, you could use *stochastic gradient descent*.

There are specialized optimization procedures that re-formulate the optimization problem to be a Quadratic Programming problem. The most popular method for fitting SVM is the Sequential Minimal Optimization (SMO) method that is very efficient. It breaks the problem down into sub-problems that can be solved analytically (by calculating) rather than numerically (by searching or optimizing).

### Sequential Minimal Optimization (SMO) method

Sequential minimal optimization (SMO) is an algorithm for solving the quadratic programming (QP) problem that arises during the training of support vector machines.

The simplified SMO works OK on small datasets with a few hundred points but slows down on larger datasets.

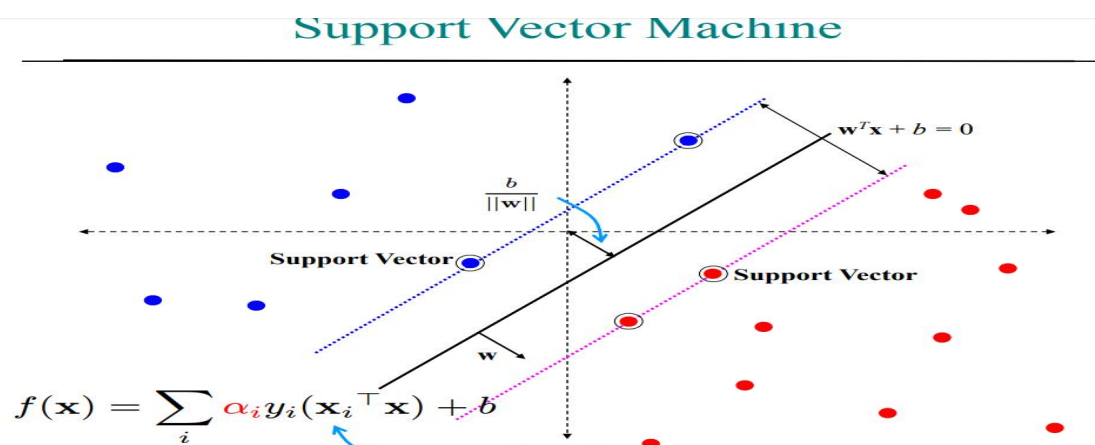
### Data Preparation for SVM

This section lists some suggestions for how to best prepare your training data when learning an SVM model.

**Numerical Inputs:** SVM assumes that your inputs are numeric. If you have categorical inputs you may need to convert them to binary dummy variables (one variable for each category).

**Binary Classification:** Given a collection of objects let us say we have the task to classify the objects into two groups based on some feature.

**Note:** Support vectors are the data points nearest to the hyperplane, the points of a data set that, if removed, would alter the position of the dividing hyperplane. Because of this, they can be considered the critical elements of a data set.



## Example

Steps required for digit recognition with SVMs

1. Collect: Text file provided.
2. Prepare: Create vectors from the binary images.
3. Analyze: Visually inspect the image vectors.
4. Train: Run the SMO algorithm with two different kernels and different settings for the radial bias kernel.
5. Test: Write a function to test the different kernels and calculate the error rate.
6. Use: A full application of image recognition requires some image processing, which we won't get into.

## Multi-class classification

In machine learning, **multiclass** or **multinomial classification** is the problem of classifying instances into one of three or more classes.

### 2.1.1.2 Discriminant Analysis

#### i.Linear Discriminant Analysis for Machine Learning

Linear Discriminant Analysis (LDA) is most commonly used as dimensionality reduction technique in the pre-processing step for pattern-classification and machine learning applications. The goal is to project a dataset onto a lower-dimensional space with good class-separability in order avoid overfitting (“curse of dimensionality”) and also reduce computational costs.

#### Learning LDA Models

LDA makes some simplifying assumptions about your data:

1. That your data is Gaussian, that each variable is shaped like a bell curve when plotted.
2. 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.

#### 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.

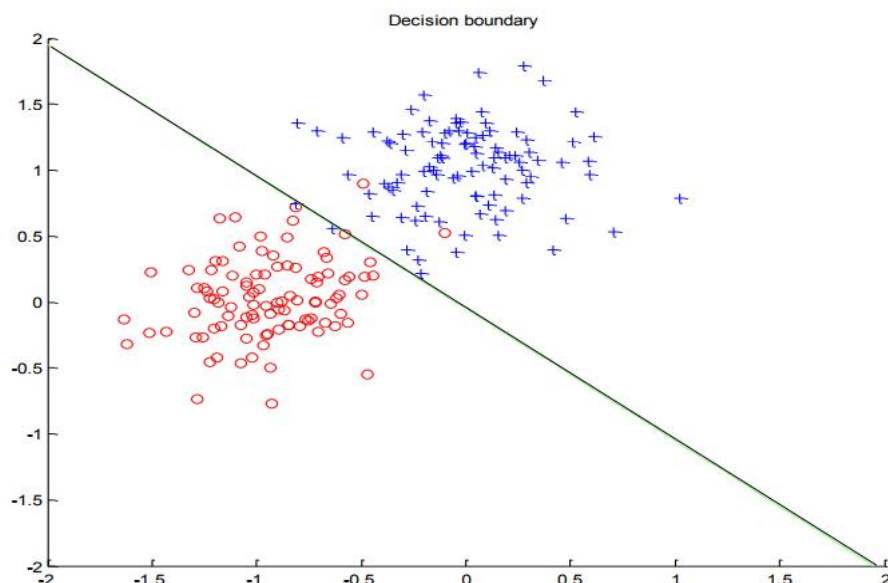
#### How to Prepare Data for LDA

This section lists some suggestions you may consider when preparing your data for use with LDA.

- **Classification Problems.** This might go without saying, but LDA is intended for classification problems where the output variable is categorical. LDA supports both binary and multi-class classification.

- **Gaussian Distribution.** The standard implementation of the model assumes a Gaussian distribution of the input variables. Consider reviewing the univariate distributions of each attribute and using transforms to make them more Gaussian-looking (e.g. log and root for exponential distributions and Box-Cox for skewed distributions).
- **Remove Outliers.** Consider removing outliers from your data. These can skew the basic statistics used to separate classes in LDA such the mean and the standard deviation.
- **Same Variance.** LDA assumes that each input variable has the same variance. It is almost always a good idea to standardize your data before using LDA so that it has a mean of 0 and a standard deviation of 1.

## LDA: linear decision boundary



### Extensions to LDA

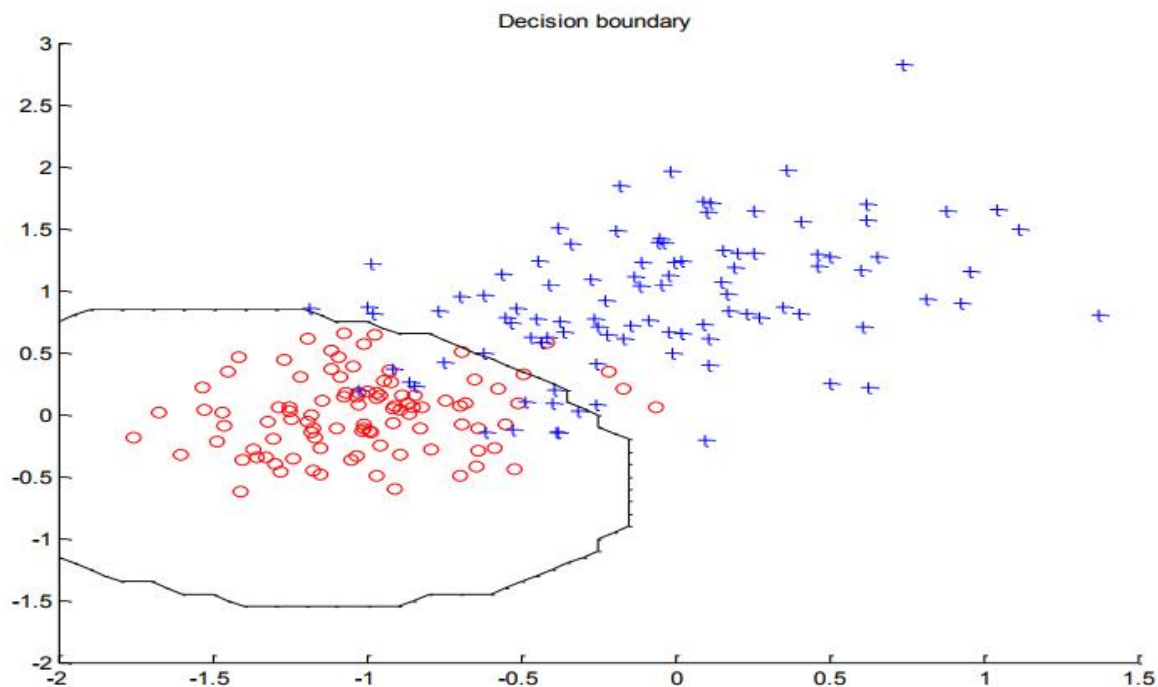
Linear Discriminant Analysis is a simple and effective method for classification. Because it is simple and so well understood, there are many extensions and variations to the method. Some popular extensions include:

- **Quadratic Discriminant Analysis (QDA):** Each class uses its own estimate of variance (or covariance when there are multiple input variables). A **quadratic classifier** is used in machine learning and statistical classification to separate measurements of two or more classes of objects or events by a quadric surface. It is a more general version of the linear classifier.
- **Flexible Discriminant Analysis (FDA):** Where non-linear combinations of inputs is used such as splines.

- **Regularized Discriminant Analysis (RDA):** Introduces regularization into the estimate of the variance (actually covariance), moderating the influence of different variables on LDA.

The original development was called the Linear Discriminant or Fisher's Discriminant Analysis. The multi-class version was referred to Multiple Discriminant Analysis. These are all simply referred to as Linear Discriminant Analysis now.

## QDA: Quadratic decision boundary



### Example

#### Face recognition

In computerised face recognition, each face is represented by a large number of pixel values. Linear discriminant analysis is primarily used here to reduce the number of features to a more manageable number before classification. Each of the new dimensions is a linear combination of pixel values, which form a template. The linear combinations obtained using Fisher's linear discriminant are called *Fisher faces*, while those obtained using the related principal component analysis are called *eigenfaces*.

#### Marketing

In marketing, discriminant analysis was once often used to determine the factors which distinguish different types of customers and/or products on the basis of surveys or other forms of collected data.

Logistic regression or other methods are now more commonly used. The use of discriminant analysis in marketing can be described by the following steps:

- i. Formulate the problem and gather data
- ii. Estimate the Discriminant Function Coefficients and determine the statistical significance and validity
- iii. Plot the results on a two dimensional map, define the dimensions, and interpret the results.

### **Biomedical studies**

The main application of discriminant analysis in medicine is the assessment of severity state of a patient and prognosis of disease outcome. For example, during retrospective analysis, patients are divided into groups according to severity of disease – mild, moderate and severe form. Then results of clinical and laboratory analyses are studied in order to reveal variables which are statistically different in studied groups. Using these variables, discriminant functions are built which help to objectively classify disease in a future patient into mild, moderate or severe form.

### **Earth science**

This method can be used to separate the alteration zones. For example, when different data from various zones are available, discriminant analysis can find the pattern within the data and classify it effectively.

#### **2.1.1.3 Naive Bays**

In machine learning we are often interested in selecting the best hypothesis (h) given data (d).

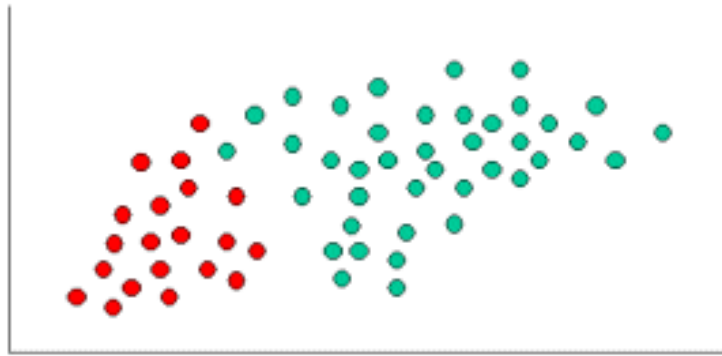
In a classification problem, our hypothesis (h) may be the class to assign for a new data instance (d).

One of the easiest ways of selecting the most probable hypothesis given the data that we have that we can use as our prior knowledge about the problem. Bayes' Theorem provides a way that we can calculate the probability of a hypothesis given our prior knowledge.

#### **Naive Bayes Classifier**

Naive Bayes is a classification algorithm for binary (two-class) and multi-class classification problems. The technique is easiest to understand when described using binary or categorical input values.

Naive Bayes is a kind of classifier which uses the Bayes Theorem. It predicts membership probabilities for each class such as the probability that given record or data point belongs to a particular class. The class with the highest probability is considered as the most likely class. This is also known as **Maximum A Posteriori (MAP)**.



To demonstrate the concept of Naïve Bayes Classification, consider the example displayed in the illustration above. As indicated, the objects can be classified as either GREEN or RED. Our task is to classify new cases as they arrive, i.e., decide to which class label they belong, based on the currently existing objects.

Since there are twice as many GREEN objects as RED, it is reasonable to believe that a new case (which hasn't been observed yet) is twice as likely to have membership GREEN rather than RED. In the Bayesian analysis, this belief is known as the prior probability. Prior probabilities are based on previous experience, in this case the percentage of GREEN and RED objects, and often used to predict outcomes before they actually happen.

### **Representation Used By Naive Bayes Models**

The representation for naive Bayes is probabilities.

A list of probabilities are stored to file for a learned naive Bayes model. This includes:

- **Class Probabilities:** The probabilities of each class in the training dataset.
- **Conditional Probabilities:** The conditional probabilities of each input value given each class value.

### **Learn a Naive Bayes Model From Data**

Learning a naive Bayes model from your training data is fast.

Training is fast because only the probability of each class and the probability of each class given different input (x) values need to be calculated. No coefficients need to be fitted by optimization procedures.

### **Make Predictions With a Naive Bayes Model**

Given a naive Bayes model, you can make predictions for new data using Bayes theorem.

### **Gaussian Naive Bayes**

Naive Bayes can be extended to real-valued attributes, most commonly by assuming a Gaussian distribution.



This extension of naive Bayes is called Gaussian Naive Bayes. Other functions can be used to estimate the distribution of the data, but the Gaussian (or Normal distribution) is the easiest to work with because you only need to estimate the mean and the standard deviation from your training data.

### **Best Prepare Your Data For Naive Bayes**

- **Categorical Inputs:** Naive Bayes assumes label attributes such as binary, categorical or nominal.
- **Gaussian Inputs:** If the input variables are real-valued, a Gaussian distribution is assumed. In which case the algorithm will perform better if the univariate distributions of your data are Gaussian or near-Gaussian.
- **Classification Problems:** Naive Bayes is a classification algorithm suitable for binary and multiclass classification.
- **Log Probabilities:** The calculation of the likelihood of different class values involves multiplying a lot of small numbers together. This can lead to an underflow of numerical precision. As such it is good practice to use a log transform of the probabilities to avoid this underflow.
- **Kernel Functions:** Rather than assuming a Gaussian distribution for numerical input values, more complex distributions can be used such as a variety of kernel density functions.
- **Update Probabilities:** When new data becomes available, you can simply update the probabilities of your model. This can be helpful if the data changes frequently.

### **General approach to naïve Bayes**

1. Collect: Any method. We'll use RSS feeds in this chapter.
2. Prepare: Numeric or Boolean values are needed.
3. Analyze: With many features, plotting features isn't helpful. Looking at histograms is a better idea.
4. Train: Calculate the conditional probabilities of the independent features.
5. Test: Calculate the error rate.
6. Use: One common application of naïve Bayes is document classification. You can use naïve Bayes in any classification setting. It doesn't have to be text.

### **Example:**

#### **A. Using naïve Bayes to classify email**

- i. Collect: Text files provided.
- ii. Prepare: Parse text into token vectors.
- iii. Analyze: Inspect the tokens to make sure parsing was done correctly.
- iv. Train: Use `trainNB0()` that we created earlier.
- v. Test: Use `classifyNB()` and create a new testing function to calculate the error rate over a set of documents.

vi. Use: Build a complete program that will classify a group of documents and print misclassified documents to the screen.

### **B. Using naïve Bayes to find locally used words**

i. Collect: Collect from RSS feeds. We'll need to build an interface to the RSS feeds.

ii. Prepare: Parse text into token vectors.

iii. Analyze: Inspect the tokens to make sure parsing was done correctly.

iv. Train: Use `trainNB0()` that we created earlier.

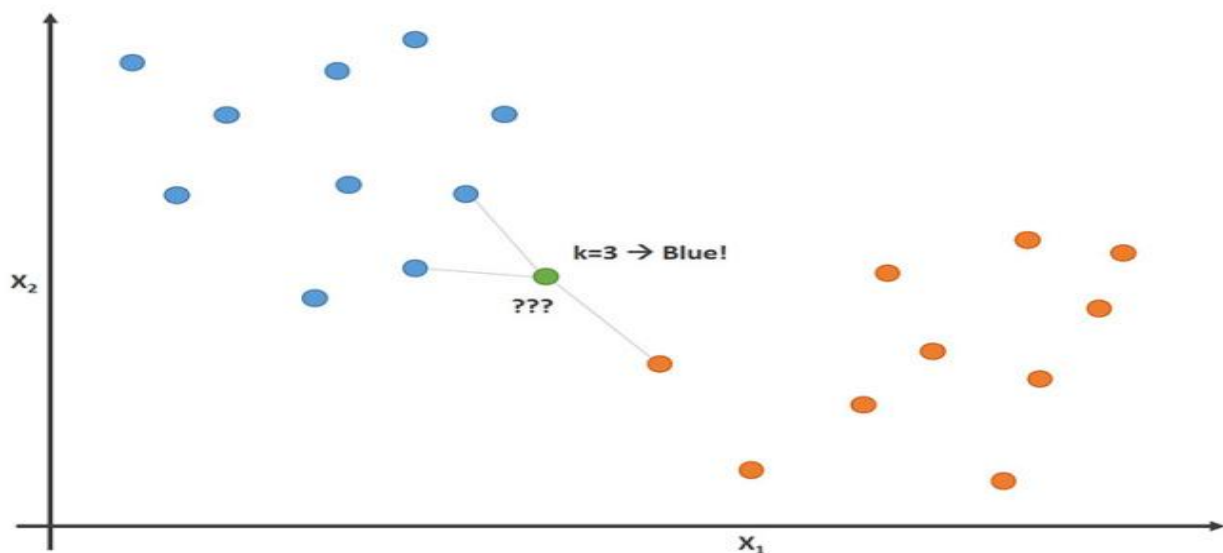
v. Test: We'll look at the error rate to make sure this is actually working. We can make modifications to the tokenizer to improve the error rate and results.

vi. Use: We'll build a complete program to wrap everything together. It will display the most common words given in two RSS feeds.

#### **2.1.1.4 K-Nearest Neighbors for Machine Learning**

K-Nearest Neighbors (K-NN) is one of the simplest machine learning algorithms. When a new situation occurs, it scans through all past experiences and looks up the  $k$  closest experiences. Those experiences (or: data points) are what we call the  $k$  nearest neighbors.

For example, when something significant happens in your life, you memorize that experience and use it as a guideline for future decisions.



To determine which of the  $K$  instances in the training dataset are most similar to a new input a distance measure is used. For real-valued input variables, the most popular distance measure is Euclidean distance.

Other popular distance measures include:

- **Hamming Distance:** Calculate the distance between binary vectors .
- **Manhattan Distance:** Calculate the distance between real vectors using the sum of their absolute difference. Also called City Block Distance .

- **Minkowski Distance:** Generalization of Euclidean and Manhattan distance .

### **KNN for Regression**

When KNN is used for regression problems the prediction is based on the mean or the median of the K-most similar instances.

### **KNN for Classification**

When KNN is used for classification, the output can be calculated as the class with the highest frequency from the K-most similar instances. Each instance in essence votes for their class and the class with the most votes is taken as the prediction.

### **Best Prepare Data for KNN**

- **Rescale Data:** KNN performs much better if all of the data has the same scale. Normalizing your data to the range [0, 1] is a good idea. It may also be a good idea to standardize your data if it has a Gaussian distribution.
- **Address Missing Data:** Missing data will mean that the distance between samples can not be calculated. These samples could be excluded or the missing values could be imputed.
- **Lower Dimensionality:** KNN is suited for lower dimensional data. You can try it on high dimensional data (hundreds or thousands of input variables) but be aware that it may not perform as well as other techniques. KNN can benefit from feature selection that reduces the dimensionality of the input feature space.

### **General approach to kNN**

1. Collect: Any method.
2. Prepare: Numeric values are needed for a distance calculation. A structured data format is best.
3. Analyze: Any method.
4. Train: Does not apply to the kNN algorithm.
5. Test: Calculate the error rate.
6. Use: This application needs to get some input data and output structured numeric values. Next, the application runs the kNN algorithm on this input data and determines which class the input data should belong to. The application then takes some action on the calculated class.

### **Algorithm**

For every point in our dataset:

- i) calculate the distance between inX and the current point
- ii) sort the distances in increasing order
- iii) take k items with lowest distances to inX
- iv) find the majority class among these items return the majority class as our prediction for the class of inX

## **Example:**

### **A) Improving matches from a dating site with kNN**

My friend Hellen has been using some online dating sites to find different people to go out with. She realized that despite the site's recommendations, she didn't like everyone she was matched with. After some introspection, she realized there were three types of people she went out with:

- i. People she didn't like
- ii. People she liked in small doses
- iii. People she liked in large doses

#### **Steps**

1. Collect: Text file provided.
2. Prepare: Parse a text file in Python.
3. Analyze: Use Matplotlib to make 2D plots of our data.
4. Train: Doesn't apply to the kNN algorithm.
5. Test: Write a function to use some portion of the data Hellen gave us as test examples. The test examples are classified against the non-test examples. If the predicted class doesn't match the real class, we'll count that as an error.
6. Use: Build a simple command-line program Hellen can use to predict whether she'll like someone based on a few inputs.

### **B. Using kNN on a handwriting recognition system**

1. Collect: Text file provided.
2. Prepare: Write a function to convert from the image format to the list format used in our classifier, `classify0()`.
3. Analyze: We'll look at the prepared data in the Python shell to make sure it's correct.
4. Train: Doesn't apply to the kNN algorithm.
5. Test: Write a function to use some portion of the data as test examples. The test examples are classified against the non-test examples. If the predicted class doesn't match the real class, you'll count that as an error.
6. Use: Not performed in this example. You could build a complete program to extract digits from an image, such a system used to sort the mail in the United States.

#### **2.1.2 Regression**

Regression is the task of predicting a continuous quantity. Regression predictive modeling is the task of approximating a mapping function ( $f$ ) from input variables ( $X$ ) to a continuous output variable ( $y$ ). A continuous output variable is a real-value, such as an integer or floating point value. These are often quantities, such as amounts and sizes.

### 2.1.2.1 Logistic Regression for Machine Learning

The logistic function, also called the sigmoid function was developed by statisticians to describe properties of population growth in ecology, rising quickly and maxing out at the carrying capacity of the environment. It's an S-shaped curve that can take any real-valued number and map it into a value between 0 and 1, but never exactly at those limits.

$$1 / (1 + e^{-\text{value}})$$

#### Logistic Regression Predicts Probabilities (Technical Interlude)

Logistic regression models the probability of the default class (e.g. the first class).

For example, if we are modeling people's sex as male or female from their height, then the first class could be male and the logistic regression model could be written as the probability of male given a person's height, or more formally:

$$P(\text{sex}=\text{male}|\text{height})$$

Written another way, we are modeling the probability that an input (X) belongs to the default class (Y=1), we can write this formally as:

$$P(X) = P(Y=1|X)$$

#### Learning the Logistic Regression Model

The coefficients (Beta values b) of the logistic regression algorithm must be estimated from your training data. This is done using maximum-likelihood estimation.

Maximum-likelihood estimation is a common learning algorithm used by a variety of machine learning algorithms, although it does make assumptions about the distribution of your data (more on this when we talk about preparing your data).

**Note:** *Maximum-likelihood estimation:* In statistics, **maximum likelihood estimation (MLE)** is a method of estimating the parameters of a statistical model given observations, by finding the parameter values that maximize the likelihood of making the observations given the parameters.

#### Prepare Data for Logistic Regression

The assumptions made by logistic regression about the distribution and relationships in your data are much the same as the assumptions made in linear regression.

Ultimately in predictive modeling machine learning projects you are laser focused on making accurate predictions rather than interpreting the results. As such, you can break some assumptions as long as the model is robust and performs well.

- **Binary Output Variable:** This might be obvious as we have already mentioned it, but logistic regression is intended for binary (two-class) classification problems. It will predict the probability of an instance belonging to the default class, which can be snapped into a 0 or 1 classification.
- **Remove Noise:** Logistic regression assumes no error in the output variable (y), consider removing outliers and possibly misclassified instances from your training data.

- **Gaussian Distribution:** Logistic regression is a linear algorithm (with a non-linear transform on output). It does assume a linear relationship between the input variables with the output. Data transforms of your input variables that better expose this linear relationship can result in a more accurate model. For example, you can use log, root, Box-Cox and other univariate transforms to better expose this relationship.
- **Remove Correlated Inputs:** Like linear regression, the model can overfit if you have multiple highly-correlated inputs. Consider calculating the pairwise correlations between all inputs and removing highly correlated inputs.
- **Fail to Converge:** It is possible for the expected likelihood estimation process that learns the coefficients to fail to converge. This can happen if there are many highly correlated inputs in your data or the data is very sparse (e.g. lots of zeros in your input data).

### Logistic Regression by Stochastic Gradient Descent

We can estimate the values of the coefficients using stochastic gradient descent.

This is a simple procedure that can be used by many algorithms in machine learning. It works by using the model to calculate a prediction for each instance in the training set and calculating the error for each prediction.

We can apply stochastic gradient descent to the problem of finding the coefficients for the logistic regression model as follows:

Given each training instance:

1. Calculate a prediction using the current values of the coefficients.
2. Calculate new coefficient values based on the error in the prediction.

The way this optimization algorithm works is that each training instance is shown to the model one at a time. The model makes a prediction for a training instance, the error is calculated and the model is updated in order to reduce the error for the next prediction.

### Estimating Coefficients

We can estimate the coefficient values for our training data using stochastic gradient descent.

Stochastic gradient descent requires two parameters:

- **Learning Rate:** Used to limit the amount each coefficient is corrected each time it is updated.
- **Epochs:** The number of times to run through the training data while updating the coefficients.

### General approach to logistic regression

1. Collect: Any method.
2. Prepare: Numeric values are needed for a distance calculation. A structured data format is best.
3. Analyze: Any method.
4. Train: We'll spend most of the time training, where we try to find optimal coefficients to classify our data.

5. Test: Classification is quick and easy once the training step is done.

6. Use: This application needs to get some input data and output structured numeric values. Next, the application applies the simple regression calculation on this input data and determines which class the input data should belong to. The application then takes some action on the calculated class.

### **Classification with logistic regression and the sigmoid function: a tractable step function**

i. Pros: Computationally inexpensive, easy to implement, knowledge representation easy to interpret

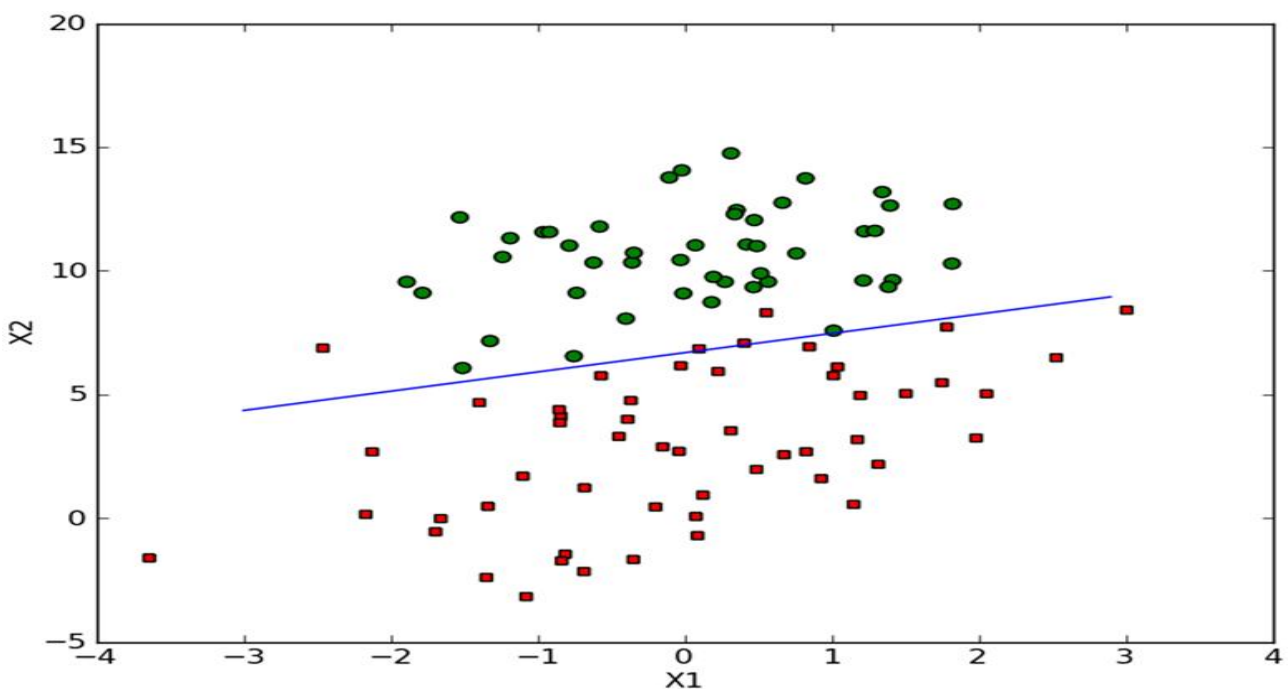
ii. Cons: Prone to underfitting, may have low accuracy

iii. Works with: Numeric values, nominal values

#### **Note:**

**Gradient Descent:** With gradient descent we're trying to minimize some function rather than maximize it.

#### **The logistic regression best-fit line after 500 cycles of gradient ascent**



### **Applications of Logistic Regression**

1. Image Segmentation and Categorization

2. Geographic Image Processing

3. Handwriting recognition

4. Healthcare : Analyzing a group of over million people for myocardial infarction within a period of 10 years is an application area of logistic regression.

5. Prediction whether a person is depressed or not based on bag of words from the corpus seems to be conveniently solvable using logistic regression and SVM.



## Real World Example

The dependent and the independent variables are the same which we were discussed in the building simple linear regression model. Just to give you a glance. The dependent variable is the target class variable we are going to predict. However, the independent variables are the features or attributes we are going to use to predict the target class.

Suppose the shop owner would like to predict the customer who entered into the shop will buy the Macbook or Not. To predict whether the customer will buy the MacBook or not. The shop owner will observe the customer features like.

- Gender:
  - Probabilities wise male will high chances of purchasing a MacBook than females.
- Age:
  - Kids won't purchase MacBook.

The shop owner will use the above, similar kind of features to predict the likelihood occurrence of the event (Will buy the Macbook or not.)

In the mathematical side, the logistic regression model will pass the likelihood occurrences through the logistic function to predict the corresponding target class. This popular logistic function is the Softmax function.



### 2.1.2.2 Linear Regression for Machine Learning

Linear regression is a **linear model**, e.g. a model that assumes a linear relationship between the input variables ( $x$ ) and the single output variable ( $y$ ). More specifically, that  $y$  can be calculated from a linear combination of the input variables ( $x$ ).

When there is a single input variable (x), the method is referred to as **simple linear regression**. When there are **multiple input variables**, literature from statistics often refers to the method as multiple linear regression.

Different techniques can be used to prepare or train the linear regression equation from data, the most common of which is called **Ordinary Least Squares**. It is common to therefore refer to a model prepared this way as Ordinary Least Squares Linear Regression or just Least Squares Regression.

### **Simple Linear Regression**

With simple linear regression when we have a single input, we can use statistics to estimate the coefficients.

For example, in a simple regression problem (a single x and a single y), the form of the model would be:

$$y = B_0 + B_1 * x$$

### **Ordinary Least Squares**

When we have more than one input we can use Ordinary Least Squares to estimate the values of the coefficients.

### **Regularization**

There are extensions of the training of the linear model called regularization methods. These seek to both minimize the sum of the squared error of the model on the training data (using ordinary least squares) but also to reduce the complexity of the model (like the number or absolute size of the sum of all coefficients in the model).

Two popular examples of regularization procedures for linear regression are:

- **Lasso Regression:** where Ordinary Least Squares is modified to also minimize the absolute sum of the coefficients (called L1 regularization).
- **Ridge Regression:** where Ordinary Least Squares is modified to also minimize the squared absolute sum of the coefficients (called L2 regularization).

These methods are effective to use when there is collinearity in your input values and ordinary least squares would overfit the training data.

### **Making Predictions with Linear Regression**

Given the representation is a linear equation, making predictions is as simple as solving the equation for a specific set of inputs.

Let's make this concrete with an example. Imagine we are predicting weight (y) from height (x). Our linear regression model representation for this problem would be:

$$y = B_0 + B_1 * x_1$$

or

$$\text{weight} = B_0 + B_1 * \text{height}$$

Where  $B_0$  is the bias coefficient and  $B_1$  is the coefficient for the height column. We use a learning technique to find a good set of coefficient values. Once found, we can plug in different height values to predict the weight.

### Preparing Data For Linear Regression

Try different preparations of your data using these heuristics and see what works best for your problem.

- **Linear Assumption.** Linear regression assumes that the relationship between your input and output is linear. It does not support anything else. This may be obvious, but it is good to remember when you have a lot of attributes. You may need to transform data to make the relationship linear (e.g. log transform for an exponential relationship).
- **Remove Noise.** Linear regression assumes that your input and output variables are not noisy. Consider using data cleaning operations that let you better expose and clarify the signal in your data. This is most important for the output variable and you want to remove outliers in the output variable ( $y$ ) if possible.
- **Remove Collinearity.** Linear regression will over-fit your data when you have highly correlated input variables. Consider calculating pairwise correlations for your input data and removing the most correlated.
- **Gaussian Distributions.** Linear regression will make more reliable predictions if your input and output variables have a Gaussian distribution. You may get some benefit using transforms (e.g. log or BoxCox) on your variables to make their distribution more Gaussian looking.
- **Rescale Inputs:** Linear regression will often make more reliable predictions if you rescale input variables using standardization or normalization.

### Example

#### Problem Setting

Sales (in thousands of units) for a particular product as a function of advertising budgets (in thousands of dollars) for TV, radio, and newspaper media. Suppose that in our role as statistical consultants we are asked to suggest.

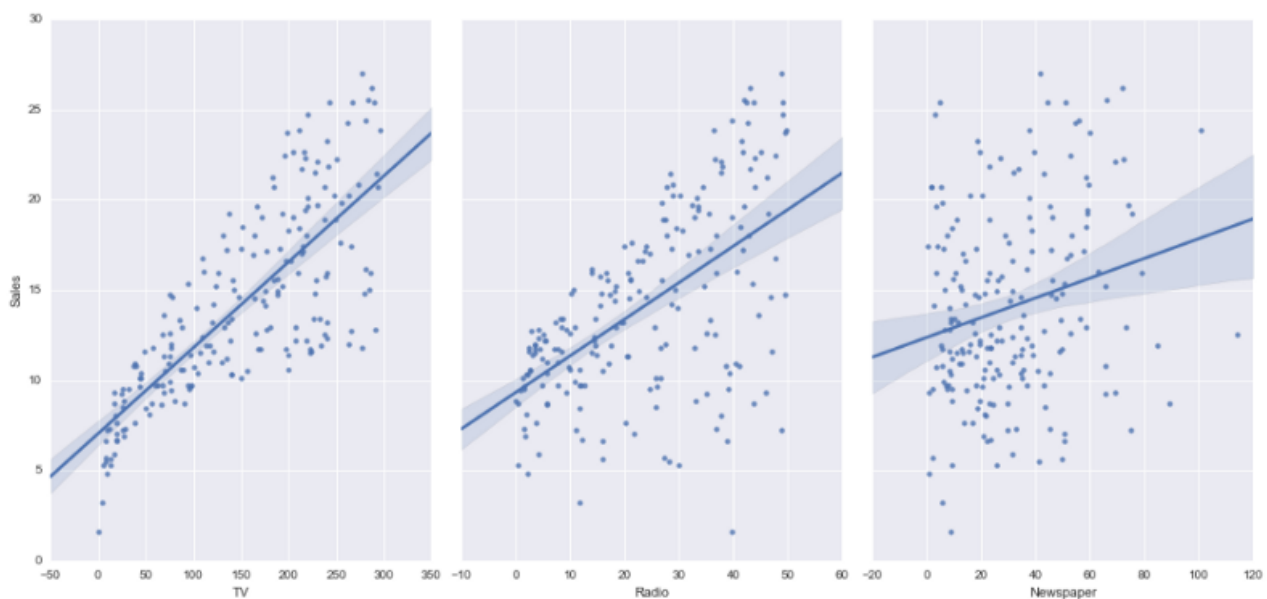
- (1) We want to find a function that given input budgets for TV, radio and newspaper predicts the output sales.
- (2) Which media contribute to sales?
- (3) Visualize the relationship between the features and the response using scatter plots.

## Advertising Data

	TV	Radio	Newspaper	Sales
1	230.1	37.8	69.2	22.1
2	44.5	39.3	45.1	10.4
3	17.2	45.9	69.3	9.3
4	151.5	41.3	58.5	18.5
5	180.8	10.8	58.4	12.9

Here you can check this plots drawn from this data to inspect the relationship between TV, Radio and Newspaper with Sales.

<seaborn.axisgrid.PairGrid at 0xaaf8fd0>



$Y = \text{Sales}$ ,  $X = [\text{TV}, \text{Radio}, \text{Newspaper}]$

## Feature selection

- Does **Newspaper** improve the quality of our predictions?
- **Hypothesis:** Newspaper does not improve model predictions.
- **Hypothesis Testing Procedure:** Let's remove Newspaper from the model and check the **RMSE** (Root Mean Squared Error).

## TIP:

- Error is something we want to minimize, so a lower number for RMSE is better.

- If we wanted to make changes and improvements to a model, the RMSE should be lower if the model is getting better.

After applying algorithm First we got RMSE of 1.40 but after removing Newspaper we got 1.38.

### 2.1.2.3 Decision Tree

Decision Trees are a type of Supervised Machine Learning (that is you explain what the input is and what the corresponding output is in the training data) where the data is continuously split according to a certain parameter. The tree can be explained by two entities, namely decision nodes and leaves. The leaves are the decisions or the final outcomes. And the decision nodes are where the data is split.

An example of a decision tree can be explained using above binary tree. Let's say you want to predict whether a person is fit given their information like age, eating habit, and physical activity, etc. The decision nodes here are questions like 'What's the age?', 'Does he exercise?', 'Does he eat a lot of pizzas'? And the leaves, which are outcomes like either 'fit', or 'unfit'. In this case this was a binary classification problem (a yes no type problem).

There are two main types of Decision Trees:

#### 1. **Classification trees** (Yes/No types)

What we've seen above is an example of classification tree, where the outcome was a variable like 'fit' or 'unfit'. Here the decision variable is **Categorical**.

#### 2. **Regression trees** (Continuous data types)

Here the decision or the outcome variable is **Continuous**, e.g. a number like 123.

### **Learn a CART Model From Data**

Creating a CART model involves selecting input variables and split points on those variables until a suitable tree is constructed.

The selection of which input variable to use and the specific split or cut-point is chosen using a greedy algorithm to minimize a cost function. Tree construction ends using a predefined stopping criterion, such as a minimum number of training instances assigned to each leaf node of the tree.

### **Data Preparation for CART**

CART does not require any special data preparation other than a good representation of the problem.

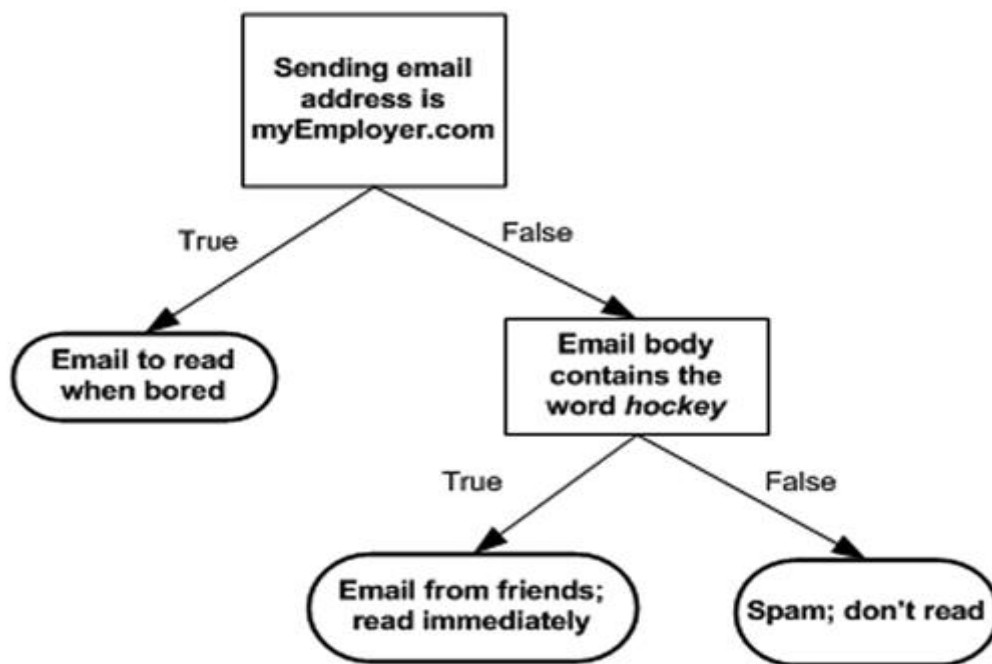
**Note:**

## Pruning

The performance of a tree can be further increased by **pruning**. It involves **removing the branches that make use of features having low importance**. This way, we reduce the complexity of tree, and thus increasing its predictive power by reducing overfitting.

## Decision Tree Construction

- i. Pros: Computationally cheap to use, easy for humans to understand learned results, missing values OK, can deal with irrelevant features
- ii. Cons: Prone to overfitting
- iii. Works with: Numeric values, nominal values



A decision tree in flowchart form

## General approach to decision trees

1. Collect: Any method.
2. Prepare: This tree-building algorithm works only on nominal values, so any continuous values will need to be quantized.
3. Analyze: Any method. You should visually inspect the tree after it is built.
4. Train: Construct a tree data structure.
5. Test: Calculate the error rate with the learned tree.
6. Use: This can be used in any

## Example:

### Using decision trees to predict contact lens type

1. Collect: Text file provided.

2. Prepare: Parse tab-delimited lines.
3. Analyze: Quickly review data visually to make sure it was parsed properly. The final tree will be plotted with `createPlot()`.
4. Train: Use `createTree()`
5. Test: Write a function to descend the tree for a given instance.
6. Use: Persist the tree data structure so it can be recalled without building the tree; then use it in any application.

#### 2.1.2.4 Support Vector Regressor(SVR)

The SVR uses the same principles as the SVM for classification, with only a few minor differences. First of all, because output is a real number it becomes very difficult to predict the information at hand, which has infinite possibilities.

#### Example

##### Stock Price predictions

##### 1) Training phase

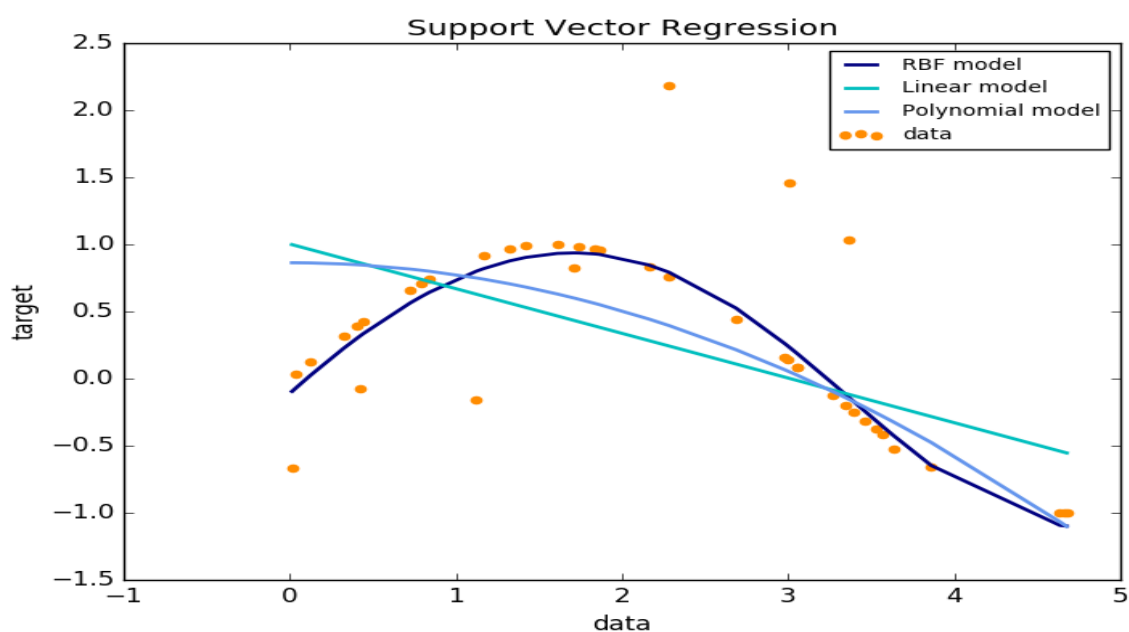
Step 1: Read the training dataset from local repository.

Step 2: Apply windowing operator to transform the time series data into a generic dataset. This step will convert the last row of a window within the time series into a label or target variable. Last variable is treated as label.

Step 3: Accomplish a cross validation process of the produced label from windowing operator in order to feed them as inputs into SVR model.

Step 4: Select kernel types and select special parameters of SVR ( C , f , g etc).

Step 5: Run the model and observe the performance (accuracy).





## Concluding Remarks

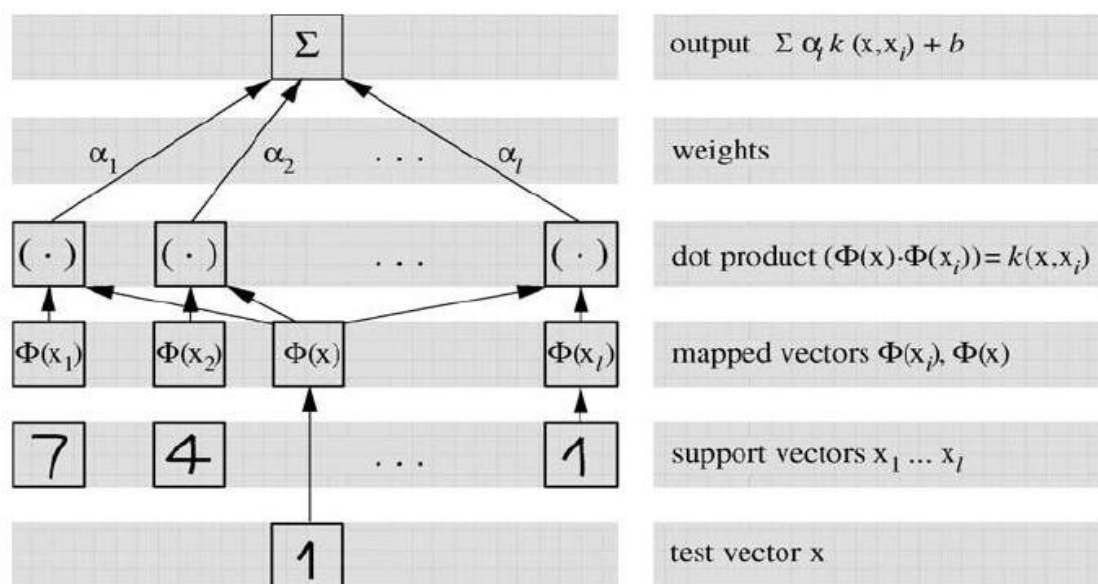
- i. Small to medium data sets only. Training becomes extremely slow in the case of larger datasets.
- ii. Data sets with low noise. When the data set has more noise i.e. target classes are overlapping, SVM perform very poorly.
- iii. When feature dimensions are very large. SVMs are extremely helpful specially when no. of features is larger than no. of samples.
- iv. Since only a subset of training points are used in the decision function (called support vectors), it is quite memory efficient. This also leads to extremely fast prediction.

An important point to note is that the SVM doesn't directly provide probability estimates, these are calculated using an expensive cross-validation in scikit-learn implementation.

## Another Example

# SVR Applications

## Optical Character Recognition (OCR)



---

- Stock price prediction



#### 2.1.2.5 Generalized linear models

Generalized linear models are a class of linear models which unify several widely used models, including linear regression and logistic regression. The distribution over each output is assumed to be an exponential family distribution whose natural parameters are a linear function of the inputs.

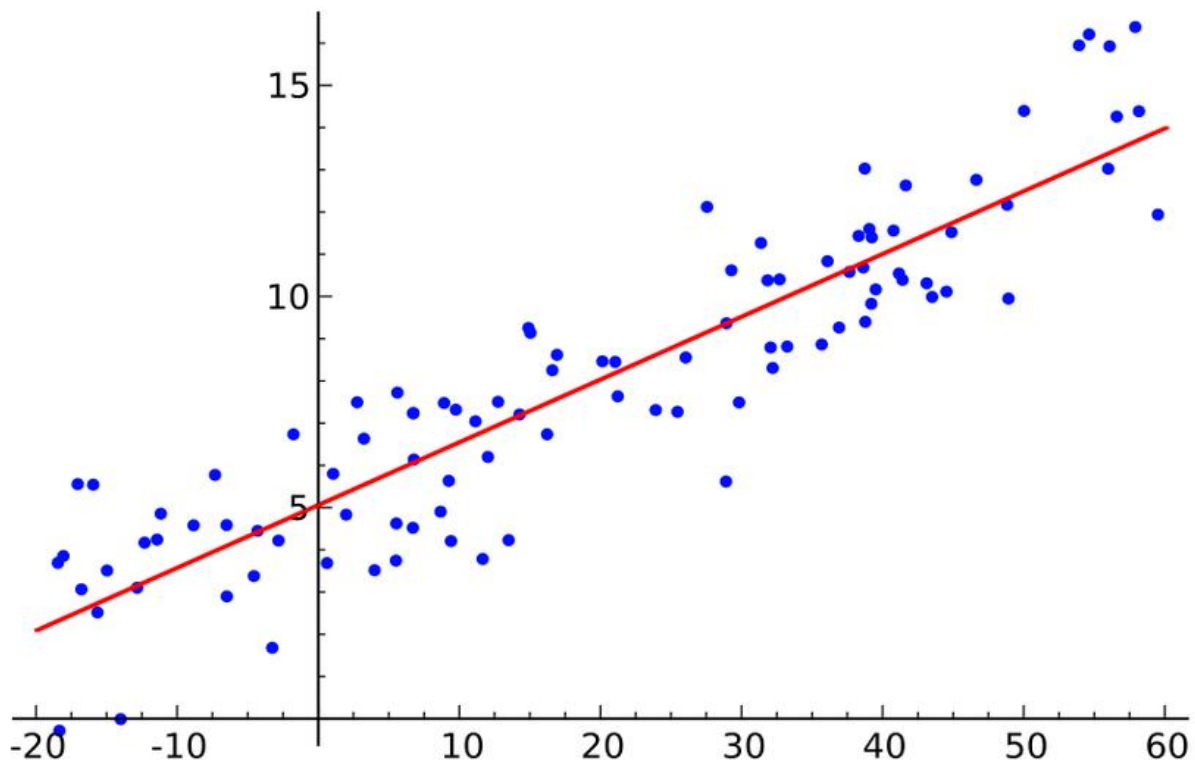
In statistics, the **generalized linear model (GLM)** is a flexible generalization of ordinary linear regression that allows for response variables that have error distribution models other than a normal distribution. The GLM generalizes linear regression by allowing the linear model to be related to the response variable via a *link function* and by allowing the magnitude of the variance of each measurement to be a function of its predicted value.

GLMs allow for response variables that have error distribution models other than a normal distribution. Some common examples of GLMs are:

- i. Poisson regression for count data.
- ii. Logistic regression and probit regression for binary data.
- iii. Multinomial logistic regression and multinomial probit regression for categorical data.
- iv. Ordered probit regression for ordinal data.

# Generalized Linear Models

---



## 2.1.2.6 Ensemble Method

In statistics and machine learning, **ensemble methods** use multiple learning algorithms to obtain better predictive performance than could be obtained from any of the constituent learning algorithms alone.

Ensemble learning is just a way of combining the output of multiple models in order to get a better result. This is a cheap way of improving your model.

**Example:** I want to invest in a company XYZ. I am not sure about its performance though. So, I look for advice on whether the stock price will increase more than 6% per annum or not? I decide to approach various experts having diverse domain experience:

1. Employee of Company XYZ: This person knows the internal functionality of the company and have the insider information about the functionality of the firm. But he lacks a broader perspective on how are competitors innovating, how is the technology evolving and what will be the impact of this evolution on Company XYZ's product. **In the past, he has been right 70% times.**

2. Financial Advisor of Company XYZ: This person has a broader perspective on how companies strategy will fair of in this competitive environment. However, he lacks a view on how the company's internal policies are fairing off. **In the past, he has been right 75% times.**

3. *Stock Market Trader*: This person has observed the company's stock price over past 3 years. He knows the seasonality trends and how the overall market is performing. He also has developed a strong intuition on how stocks might vary over time. **In the past, he has been right 70% of times.**

4. Employee of a competitor: This person knows the internal functionality of the competitor firms and is aware of certain changes which are yet to be brought. He lacks a sight of company in focus and the external factors which can relate the growth of competitor with the company of subject. **In the past, he has been right 60% of times.**

5. Market Research team in same segment: This team analyzes the customer preference of company XYZ's product over others and how is this changing with time. Because he deals with customer side, he is unaware of the changes company XYZ will bring because of alignment to its own goals. **In the past, they have been right 75% of times.**

6. Social Media Expert: This person can help us understand how has company XYZ has positioned its products in the market. And how are the sentiment of customers changing over time towards company. He is unaware of any kind of details beyond digital marketing. **In the past, he has been right 65% of times.**

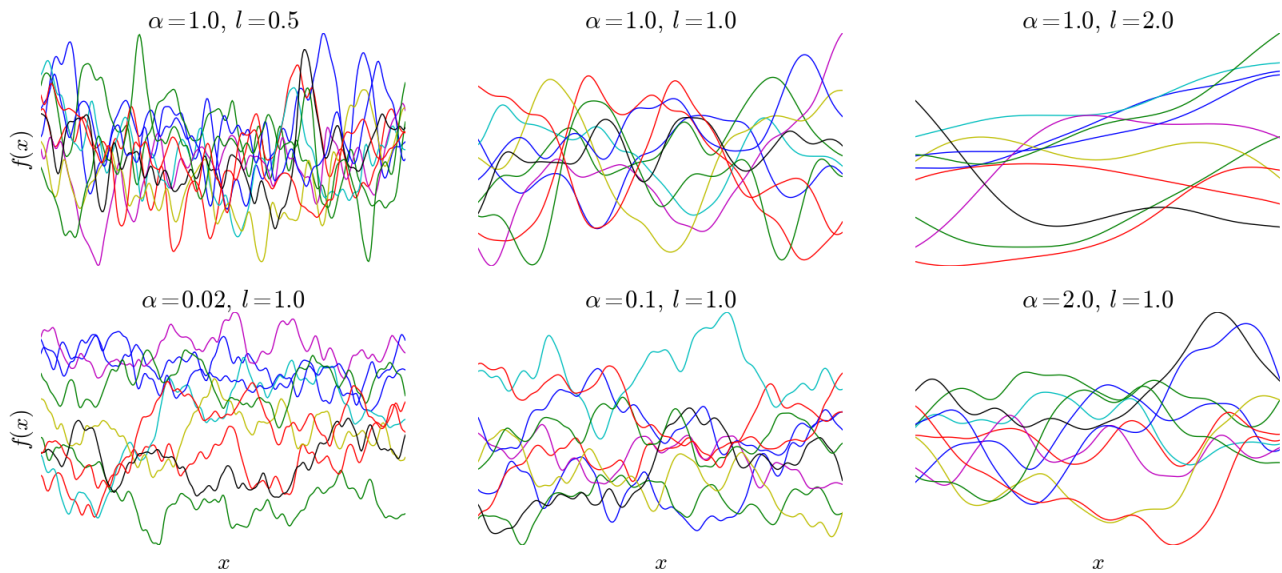
#### 2.1.2.7 Gaussian Processes Regression

A GPR model is defined primarily by the selection of a covariance function, which defines how the expected value of the target variable changes as values change across the input space.

A Gaussian process is a generalization of the Gaussian probability distribution. Whereas a probability distribution describes random variables which are scalars or vectors (for multivariate distributions), a stochastic process governs the properties of functions.

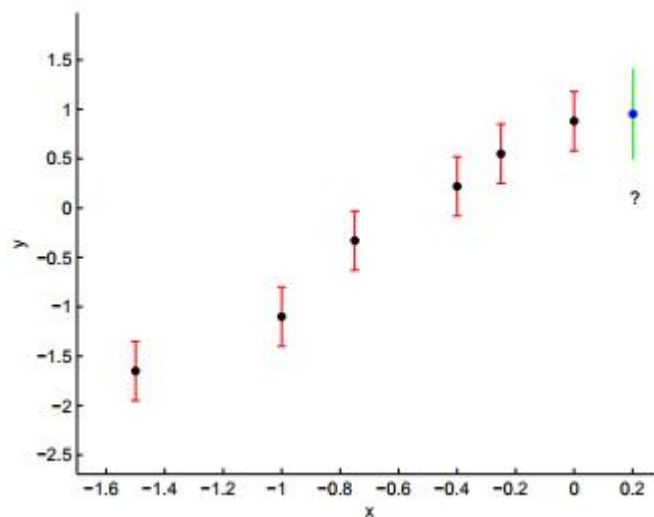
**For example**, suppose we measure the temperature every day of the year at noon, resulting in a 365-dimensional vector. In reality, temperature is a continuous process, and the choice to take a measurement every day at noon is arbitrary. What would happen if we took the temperature in the evening instead? What if we took measurements every hour or every week? If we model the data with a GP, we are assuming that each of these possible data collection schemes would yield data from a multivariate normal distribution.

Below are samples drawn from a GP with a rational quadratic kernel and various kernel parameters



### Another Example

Given six noisy data points (error bars are indicated with vertical lines), we are interested in estimating a seventh at  $x^* = 0.2$ .



**Note:**

#### 1. Variance Covariance Matrix

**Variance** is a measure of the variability or spread in a set of data. Mathematically, it is the average squared deviation from the mean score. We use the following formula to compute variance.

$$\text{Var}(X) = \frac{\sum (X_i - \bar{X})^2}{N} = \frac{\sum X_i^2}{N} - \bar{X}^2$$

**Covariance** is a measure of the extent to which corresponding elements from two sets of ordered data move in the same direction.

Covariance calculates for every combination of variables who are more similar (on a scale of 0 to any number). Then you can choose the redundant variables from there to make your model more effective and accurate.

### Definition

Covariance is a measure of the extent to which corresponding elements from two sets of ordered data move in the same direction. We use the following formula to compute covariance.

$$\text{Cov}(X, Y) = \Sigma (X_i - X_m) (Y_i - Y_m) / N$$

$$\text{Cov}(X) = E (E(X - X_m)' * E(X - X_m))$$

here X is the dataset, X<sub>m</sub> is the mean of corresponding columns or variables, similarly for y . E is the mean.

You have a fleet of workers performing some tasks under you. Some of them are doing same tasks everyday and therefore redundant. What will you do then ?... Continue to pay them as usual. Sounds like a bad idea, as your profit would be sacrificed. Then what's the solution.. Find who are the ones doing the same activity everyday, same actions, same time of entry and exit etc ? After identification remove either one, or remove all and appoint 1 person instead who can perform the task collectively on their behalf.

Now bringing some maths here, workers are the columns (variables) of data. Your profit is your accuracy, and your payment is overfitting. To remove the problem of overfitting you need to remove some extra variables or instead appoint one variable which can be treated as a collection of 2 or more old variables.

Covariance matrix is the solution to your problem, which calculates for every combination of variables who are more similar (on a scale of 0 to any number). Then you can choose the redundant variables from there to make your model more effective and accurate.

Take an example:

Student	Math	English	Art
1	90	60	90
2	90	90	30
3	60	60	60
4	60	60	90
5	30	30	30

⇒

90	60	90
90	90	30
60	60	60
60	60	90
30	30	30

**A**

Covariance matrix is of dimension #cols \* #cols, diagonal represents the variance of each variable (obviously as it gets calculated with itself)

$$\mathbf{V} = \mathbf{a}'\mathbf{a} / n = \begin{bmatrix} 2520/5 & 1800/5 & 900/5 \\ 1800/5 & 1800/5 & 0/5 \\ 900/5 & 0/5 & 3600/5 \end{bmatrix} = \begin{bmatrix} 504 & 360 & 180 \\ 360 & 360 & 0 \\ 180 & 0 & 720 \end{bmatrix}$$

We can interpret the variance and covariance statistics in matrix  $\mathbf{V}$  to understand how the various test scores vary and covary.

Shown in red along the diagonal, we see the variance of scores for each test. The art test has the biggest variance (720); and the English test, the smallest (360). So we can say that art test scores are more variable than English test scores.

- i. The covariance is displayed in black in the off-diagonal elements of matrix  $\mathbf{V}$ .
- ii. The covariance between math and English is positive (360), and the covariance between math and art is positive (180). This means the scores tend to covary in a positive way. As scores on math go up, scores on art and English also tend to go up; and vice versa.
- iii. The covariance between English and art, however, is zero. This means there tends to be no predictable relationship between the movement of English and art scores.

If the covariance between any tests had been negative, it would have meant that the test scores on those tests tend to move in opposite directions. That is, students with relatively high scores on the first test would tend to have relatively low scores on the second test.

### Covariance Matrix

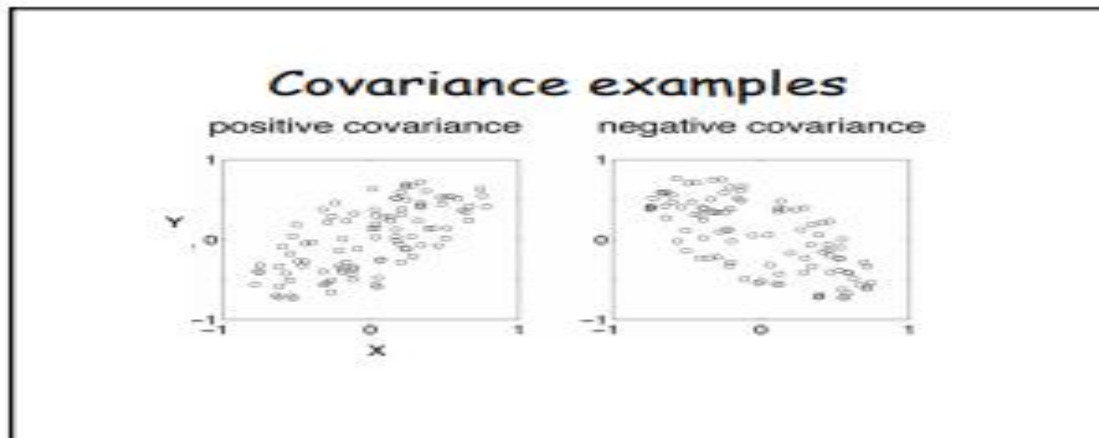
- Representing Covariance between dimensions as a matrix e.g. for 3 dimensions:

$$C = \begin{bmatrix} \text{cov}(x,x) & \text{cov}(x,y) & \text{cov}(x,z) \\ \text{cov}(y,x) & \text{cov}(y,y) & \text{cov}(y,z) \\ \text{cov}(z,x) & \text{cov}(z,y) & \text{cov}(z,z) \end{bmatrix}$$

Variances

- Diagonal is the variances of x, y and z
- $\text{cov}(x,y) = \text{cov}(y,x)$  hence matrix is symmetrical about the diagonal
- N-dimensional data will result in NxN covariance matrix





The covariance between two variables is positive when they tend to move in the same direction and negative if they tend to move in opposite directions.

To summarize whether the two variables move together, we can look at the product:

- i. When both variables have the same sign, it's positive;
- ii. When they have opposite signs, it's negative;
- iii. When they are both large and have the same sign, you get a big positive number;
- iii. When they are both large and have opposite signs, you get a big negative number;
- iv. And so on.

## 2. AdaBoost Algorithm in Machine Learning (Classification)

### AdaBoost

- i. Pros: Low generalization error, easy to code, works with most classifiers, no parameters to adjust
- ii. Cons: Sensitive to outliers
- iii. Works with: Numeric values, nominal values

Boosting is a general ensemble method that creates a strong classifier from a number of weak classifiers.

This is done by building a model from the training data, then creating a second model that attempts to correct the errors from the first model. Models are added until the training set is predicted perfectly or a maximum number of models are added.

AdaBoost was the first really successful boosting algorithm developed for binary classification. It is the best starting point for understanding boosting.

Modern boosting methods build on AdaBoost, most notably stochastic gradient boosting machines.

### Learning An AdaBoost Model From Data

AdaBoost is best used to boost the performance of decision trees on binary classification problems.

### How To Train One Model

A weak classifier (decision stump) is prepared on the training data using the weighted samples. Only binary (two-class) classification problems are supported, so each decision stump makes one decision on one input variable and outputs a +1.0 or -1.0 value for the first or second class value.

### **Making Predictions with AdaBoost**

Predictions are made by calculating the weighted average of the weak classifiers.

### **Data Preparation for AdaBoost**

This section lists some heuristics for best preparing your data for AdaBoost.

- **Quality Data:** Because the ensemble method continues to attempt to correct misclassifications in the training data, you need to be careful that the training data is of a high-quality.
- **Outliers:** Outliers will force the ensemble down the rabbit hole of working hard to correct for cases that are unrealistic. These could be removed from the training dataset.
- **Noisy Data:** Noisy data, specifically noise in the output variable can be problematic. If possible, attempt to isolate and clean these from your training dataset.

### **General approach to AdaBoost**

1. Collect: Any method.
2. Prepare: It depends on which type of weak learner you're going to use. In this chapter, we'll use decision stumps, which can take any type of data. You could use any classifier, so any of the classifiers from chapters 2–6 would work. Simple classifiers work better for a weak learner.
3. Analyze: Any method.
4. Train: The majority of the time will be spent here. The classifier will train the weak learner multiple times over the same dataset.
5. Test: Calculate the error rate.
6. Use: Like support vector machines, AdaBoost predicts one of two classes. If you want to use it for classification involving more than two classes, then you'll need to apply some of the same methods as for support vector machines.

### **Example: using AdaBoost on a difficult dataset**

1. Collect: Text file provided.
2. Prepare: We need to make sure the class labels are +1 and -1, not 1 and 0.
3. Analyze: Manually inspect the data.
4. Train: We'll train a series of classifiers on the data using the `adaBoost- TrainDS()` function.
5. Test: We have two datasets. With no randomization, we can have an apples-to-apples comparison of the AdaBoost results versus the logistic regression results.
6. Use: We'll look at the error rates in this example. But you could create a web-site that asks a trainer for the horse's symptoms and then predicts whether the horse will live or die.

### **Another Example**

i. A Medical-Diagnosis Example

ii. Tiny toy learning problem

## **2.2 Unsupervised Learning**

Suppose you have a basket and it is filled with some different types of fruits and your task is to arrange them as groups.

i. This time, you don't know anything about the fruits, honestly saying this is the first time you have seen them. You have no clue about those.

ii. So, how will you arrange them?

iii. What will you do first???

iv. You will take a fruit and you will arrange them by considering the physical character of that particular fruit.

v. Suppose you have considered color.

a. Then you will arrange them on considering base condition as color.

Then the groups will be something like this.

RED COLOR GROUP: apples & cherry fruits.

GREEN COLOR GROUP: bananas & grapes.

vi. So now you will take another physical character such as size.

RED COLOR AND BIG SIZE: apple.

RED COLOR AND SMALL SIZE: cherry fruits.

GREEN COLOR AND BIG SIZE: bananas.

GREEN COLOR AND SMALL SIZE: grapes.

vii. The job has done, the happy ending.

viii. Here you did not learn anything before ,means no train data and no response variable.

ix. In data mining or machine learning, this kind of learning is known as unsupervised learning.

### **2.2.1 K-Means Clustering Algorithm**

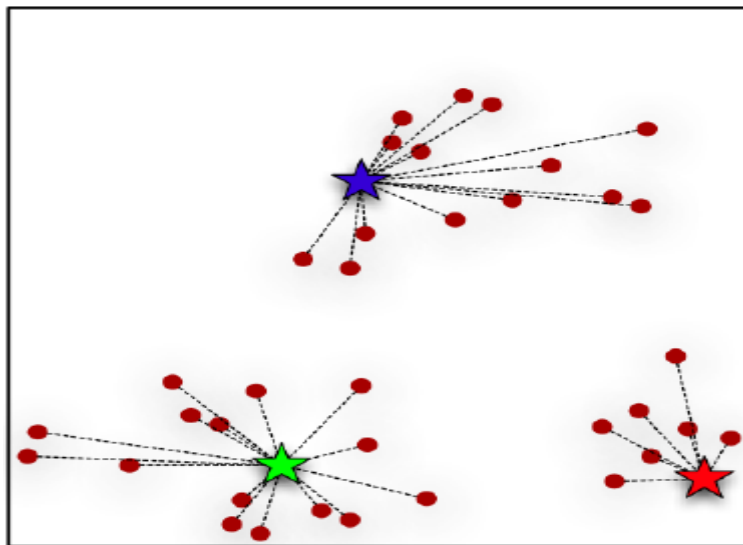
K-means is one of the simplest unsupervised learning algorithms that solve the well known clustering problem. The procedure follows a simple and easy way to classify a given data set through a certain number of clusters (assume k clusters) fixed a priori. The main idea is to define k centroids, one for each cluster. These centroids should be placed in a cunning way because of different location causes different result. So, the better choice is to place them as much as possible far away from each other. The next step is to take each point belonging to a given data set and associate it to the nearest centroid. When no point is pending, the first step is completed and an early groupage is done. At this point we need to re-calculate k new centroids as barycenters of the clusters resulting from the previous step. After we have these k new centroids, a new binding has to be done between the same data set points and the nearest new centroid. A loop has been generated.

As a result of this loop we may notice that the  $k$  centroids change their location step by step until no more changes are done. In other words centroids do not move any more. Finally, this algorithm aims at minimizing an *objective function*, in this case a squared error function.

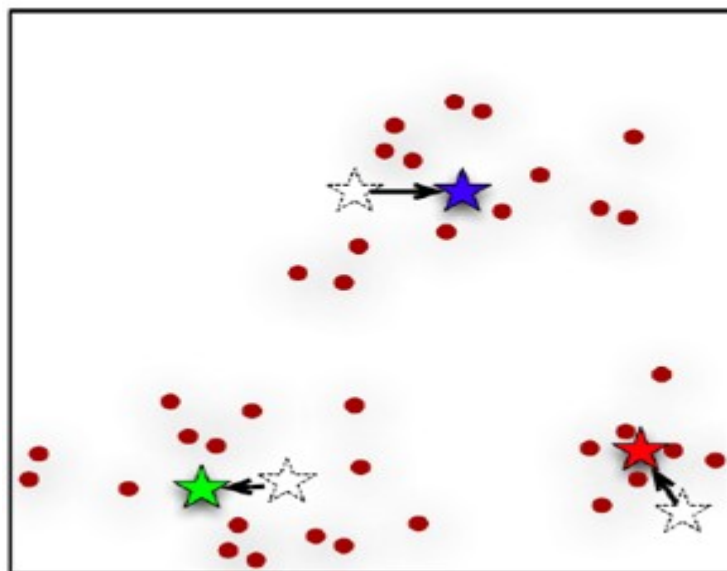
**The algorithm is composed of the following steps**

- i. Place  $K$  points into the space represented by the objects that are being clustered. These points represent initial group centroids.
- ii. Assign each object to the group that has the closest centroid.
- iii. When all objects have been assigned, recalculate the positions of the  $K$  centroids.
- iv. Repeat Steps 2 and 3 until the centroids no longer move. This produces a separation of the objects into groups from which the metric to be minimized can be calculated.

**Example:**



**Fig. 1**



**Fig 2**

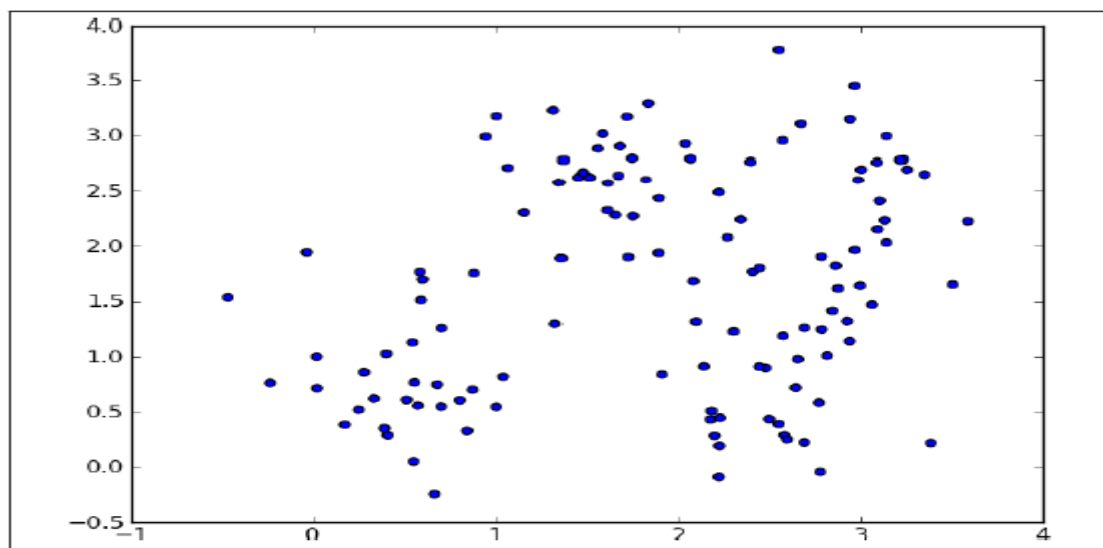
To run a k-means algorithm, you have to randomly initialize three points (See the figures 1 and 2) called the cluster centroids. I have three cluster centroids, because I want to group my data into three clusters. K-means is an iterative algorithm and it does two steps: 1. Cluster assignment step 2. Move centroid step.

In Cluster assignment step, the algorithm goes through each of the data points and depending on which cluster is closer, whether the red cluster centroid or the blue cluster centroid or the green; It assigns the data points to one of the three cluster centroids.

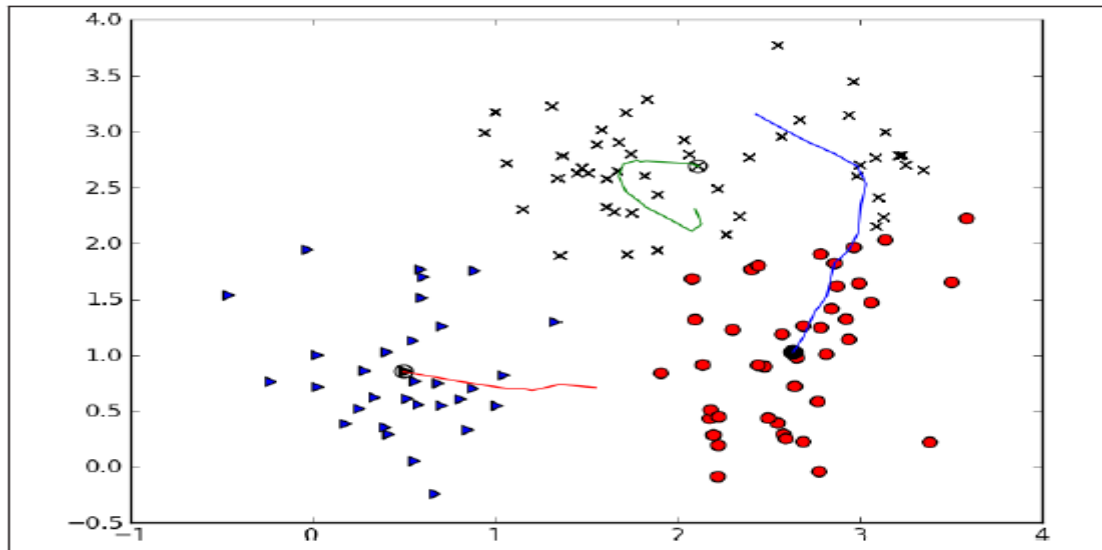
In move centroid step, K-means moves the centroids to the average of the points in a cluster. In other words, the algorithm calculates the average of all the points in a cluster and moves the centroid to that average location.

This process is repeated until there is no change in the clusters (or possibly until some other stopping condition is met). K is chosen randomly or by giving specific initial starting points by the user.

Now, check out the figures 3 and 4 below. They are the examples of K-means being run on 90 data points (with  $k=3$ ). The data does not have well defined clusters as in the previous examples. Figure 3 shows the initial data points before clustering and figure 4 shows the result after 16 iterations. The three lines in figure 4 shows the path from each centroid's initial location to its final location.



**Fig 3**



**Fig 4**

K-means is usually run many times, starting with different random centroids each time. The results can be compared by examining the clusters or by a numeric measure such as the clusters' distortion, which is the sum of the squared differences between each data point and its corresponding centroid. In cluster distortion case, the clustering with lowest distortion value can be chosen as the best clustering.

For choosing an appropriate value for K, just run the experiment using different values of K and see which ones generate good results. Since, K-means is used for exploratory data mining, you must examine the clustering results anyways to determine which clusters make sense. The value for k can be decreased if some clusters are too small, and increased if the clusters are too broad.

### **Example: using bisecting k-means on geographic data**

1. Collect: Use the Yahoo! PlaceFinder API to collect data.
2. Prepare: Remove all data except latitude and longitude.
3. Analyze: Use Matplotlib to make 2D plots of our data, with clusters and map.
4. Train: Doesn't apply to unsupervised learning.
5. Test: Use biKmeans(), developed in section 10.4.
6. Use: The final product will be your map with the clusters and cluster centers.

**This is a versatile algorithm that can be used for any type of grouping. Some examples of use cases are:**

Behavioral segmentation: Segment by purchase history

- Segment by activities on application, website, or platform
- Define personas based on interests
- Create profiles based on activity monitoring

Inventory categorization:

- Group inventory by sales activity
- Group inventory by manufacturing metrics

Sorting sensor measurements:

- Detect activity types in motion sensors
- Group images
- Separate audio
- Identify groups in health monitoring

Detecting bots or anomalies:

- Separate valid activity groups from bots
- Group valid activity to clean up outlier detection

In addition, monitoring if a tracked data point switches between groups over time can be used to detect meaningful changes in the data.

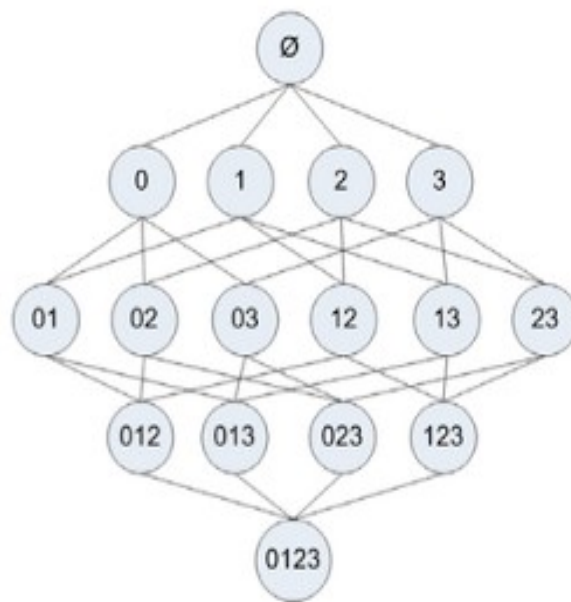
### **General approach to k-means clustering**

1. Collect: Any method.
2. Prepare: Numeric values are needed for a distance calculation, and nominal values can be mapped into binary values for distance calculations.
3. Analyze: Any method.
4. Train: Doesn't apply to unsupervised learning.
5. Test: Apply the clustering algorithm and inspect the results. Quantitative error measurements such as sum of squared error (introduced later) can be used.
6. Use: Anything you wish. Often, the clusters centers can be treated as representative data of the whole cluster to make decisions.

### 2.2.2 Apriori Algorithm in Machine Learning

**Apriori** is an algorithm for frequent item set mining and association rule learning over transactional databases. It proceeds by identifying the frequent individual items in the database and extending them to larger and larger item sets as long as those item sets appear sufficiently often in the database. The frequent item sets determined by Apriori can be used to determine association rules which highlight general trends in the database: this has applications in domains such as market basket analysis.

Let's assume that we are running a market store with a very limited selection. We are interested in finding out which items were purchased together. We have only four items: item0, item1, item2 and item3. What are all the possible combinations that can be purchased ? We can have one item, say item0 alone, or two items, or three items, or all of the items together. If someone purchased two of item0 and four of item2, we don't care. We are concerned only that they purchased one or more of an item.



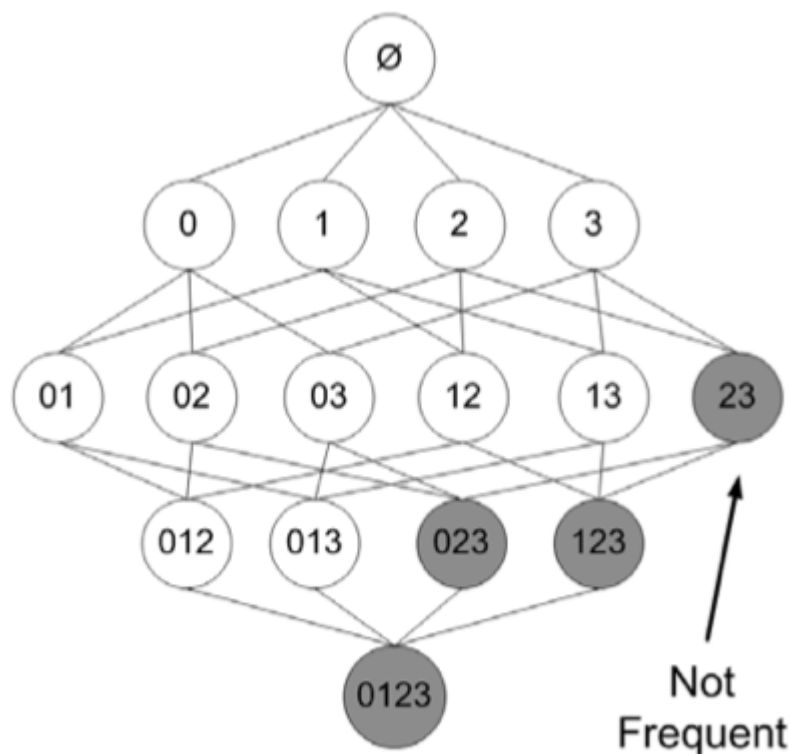
A diagram showing all possible combinations of the items is shown in the figure above. To make easier to interpret, we only use the item number such as 0 instead of item0. The first set is a big  $\emptyset$ , which means the null set or a set containing no items. Lines connecting item sets indicate that two or more sets can be combined to form a larger set.

Remember that our goal is to find sets of items that are purchased together frequently. The support of a set counted the percentage of transactions that contained that set. How do we calculate this support for a given set, say  $\{0,3\}$ ? Well. we go through every transaction and ask, "Did this transaction contain 0 and 3?" If the transaction did contain both those items, we increment the total. After scanning all of our data, we divide the total by the number of transactions and we have our



support. This result is for only one set:  $\{0,3\}$ . We will have to do this many times to get the support for every possible set. We can count the sets in Figure below and see that for four items, we have to go over the data 15 times. This number gets large quickly. A data set that contains  $N$  possible items can generate  $2^N - 1$  possible itemsets. Stores selling 10,000 or more items are not uncommon. Even a store selling 100 items can generate  $1.26 \times 10^{30}$  possible itemsets. This would take a very, very long time to compute on a modern computer.

To reduce the time needed to compute this value, researchers identified something called the **Apriori principle**. The Apriori principle helps us reduce the number of possible interesting itemsets. The Apriori principle is: if an itemset is frequent, then all of its subsets are frequent. In Figure below this means that if  $\{0,1\}$  is frequent then  $\{0\}$  and  $\{1\}$  have to be frequent. This rule as it is doesn't really help us, but if we turn it inside out it will help us. The rule turned around reads: if an itemset is infrequent then its supersets are also infrequent, as shown in Figure below.



As you can observe, the shaded itemset  $\{2,3\}$  is known to be infrequent. From this knowledge, we know that itemsets  $\{0,2,3\}$ ,  $\{1,2,3\}$ , and  $\{0,1,2,3\}$  are also infrequent. This tells us that once we have computed the support of  $\{2,3\}$ , we don't have to compute the support of  $\{0,2,3\}$ ,  $\{1,2,3\}$ , and  $\{0,1,2,3\}$  because we know they won't meet our requirements. Using this principle, we can halt the exponential growth of itemsets and in a reasonable amount of time compute a list of frequent item sets.

#### **Example: finding association rules in congressional voting records**

1. Collect: Use the votesmart module to access voting records.

2. Prepare: Write a function to process votes into a series of transaction records.
3. Analyze: We'll look at the prepared data in the Python shell to make sure it's correct.
4. Train: We'll use the `apriori()` and `generateRules()` functions written earlier in this chapter to find the interesting information in the voting records.
5. Test: Doesn't apply.
6. Use: For entertainment purposes, but you could use the results for a political campaign or to forecast how elected officials will vote.

### **Another Example: finding similar features in poisonous mushrooms**

#### **2.2.3 Hidden Markov Model**

The transition to the Hidden Markov Model consists mainly of generating another output from each state; each output having its own probability of occurring. For example, let us consider the speech recognition problem, for which HMMs have been extensively used for several decades. In speech recognition, we are interested in predicting the uttered word from a recorded speech signal. For this purpose, the speech recognizer tries to find the sequence of phonemes (states) that gave rise to the actual uttered sound (observations). Since there can be a large variation in the actual pronunciation, the original phonemes (and ultimately, the uttered word) cannot be directly observed, and need to be predicted.

#### **Applications of HMMs**

- i. Speech recognition.
- ii. Language modeling
- iii. Motion video analysis/tracking.
- iv. Protein sequence and genetic sequence alignment and analysis.
- v. Financial time series prediction.
- vi. handwriting recognition
- vii. Target tracking and localization
- viii. Natural language processing and part-of-speech recognition

#### **Note:**

Hidden Markov Models can be either supervised or unsupervised and also are called *Markovian* due to their reliance on a Markov Model. They work well where there doesn't need to be a lot of historical information built into the model. They also work well for adding localized context to a classification.

#### **2.2.4 Gaussian mixture models**

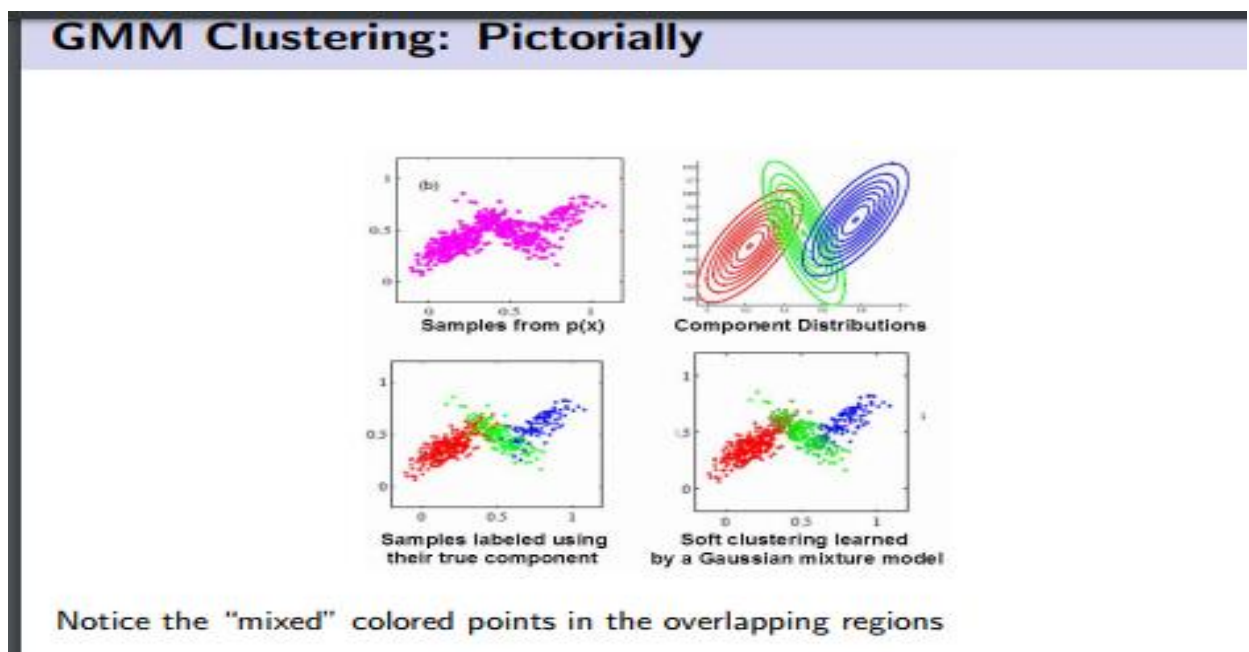
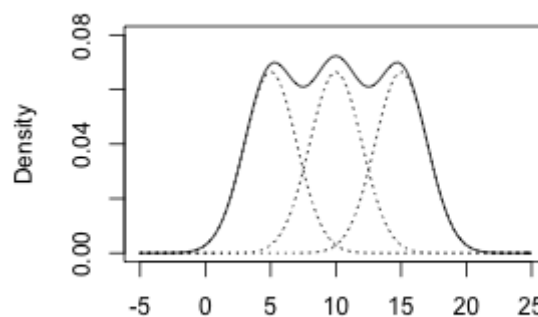
**Gaussian mixture models** are a probabilistic model for representing normally distributed subpopulations within an overall population. Mixture models in general don't require knowing which subpopulation a data point belongs to, allowing the model to learn the subpopulations

automatically. Since subpopulation assignment is not known, this constitutes a form of unsupervised learning.

For example, in modeling human height data, height is typically modeled as a normal distribution for each gender with a mean of approximately 5'10" for males and 5'5" for females. Given only the height data and not the gender assignments for each data point, the distribution of all heights would follow the sum of two scaled (different variance) and shifted (different mean) normal distributions. A model making this assumption is an example of a Gaussian mixture model (**GMM**), though in general, a GMM may have more than two components. Estimating the parameters of the individual normal distribution components is a canonical problem in modeling data with GMMs.

GMMs have been used for feature extraction from speech data, and have also been used extensively in object tracking of multiple objects, where the number of mixture components and their means predict object locations at each frame in a video sequence.

### **A Gaussian mixture of three normal distributions**



# Applications of GMM in computer vision

## 2- Object tracking:

Knowing the moving object distribution in the first frame, we can localize the object in the next frames by tracking its distribution.

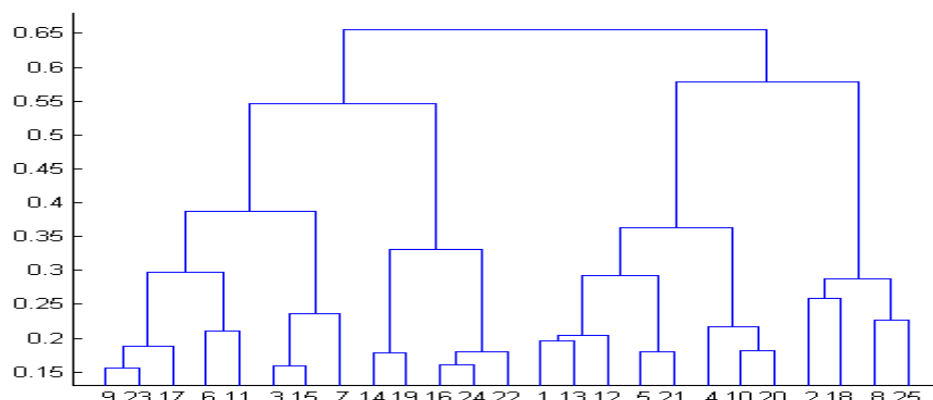


### 2.2.4 Hierarchical Clustering

Create a hierarchical decomposition of the set of objects using some criterion.

Hierarchical clustering, as the name suggests is an algorithm that builds hierarchy of clusters. This algorithm starts with all the data points assigned to a cluster of their own. Then two nearest clusters are merged into the same cluster. In the end, this algorithm terminates when there is only a single cluster left.

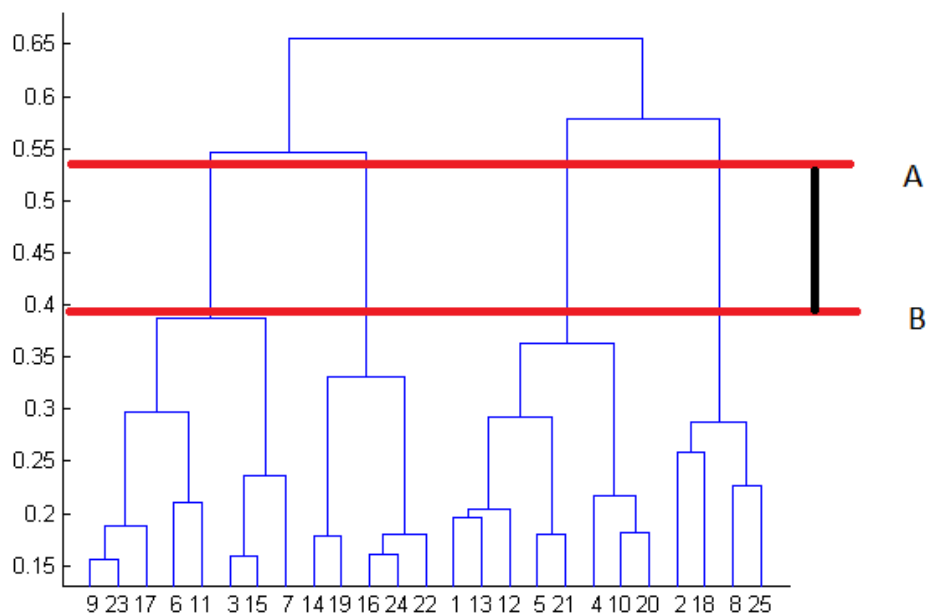
The results of hierarchical clustering can be shown using dendrogram. The dendrogram can be interpreted as:



At the bottom, we start with 25 data points, each assigned to separate clusters. Two closest clusters are then merged till we have just one cluster at the top. The height in the dendrogram at which two clusters are merged represents the distance between two clusters in the data space.

The decision of the no. of clusters that can best depict different groups can be chosen by observing the dendrogram. The best choice of the no. of clusters is the no. of vertical lines in the dendrogram cut by a horizontal line that can transverse the maximum distance vertically without intersecting a cluster.

In the above example, the best choice of no. of clusters will be 4 as the red horizontal line in the dendrogram below covers maximum vertical distance AB.



Two important things that you should know about hierarchical clustering are:

- i. This algorithm has been implemented above using bottom up approach. It is also possible to follow top-down approach starting with all data points assigned in the same cluster and recursively performing splits till each data point is assigned a separate cluster.
- ii. The decision of merging two clusters is taken on the basis of closeness of these clusters. There are multiple metrics for deciding the closeness of two clusters :

- Euclidean distance:  $\|a-b\|_2 = \sqrt{\sum(a_i-b_i)}$
- Squared Euclidean distance:  $\|a-b\|_2^2 = \sum(a_i-b_i)^2$

- Manhattan distance:  $\|a-b\|_1 = \sum |a_i - b_i|$
- Maximum distance:  $\|a-b\|_{\text{INFINITY}} = \max_i |a_i - b_i|$
- Mahalanobis distance:  $\sqrt{(a-b)^T S^{-1} (a-b)}$  {where, s : covariance matrix}

## Application of Hierarchical Clustering

Evaluate student performance of institute

Data Mining

Pattern recognition

Image analysis

Bioinformatics

Voice minig

Image processing

Text mining

Web cluster engines

Whether report analysis

## 2.4 Semi-supervised machine learning algorithms

**Semi-supervised machine learning algorithms** fall somewhere in between supervised and unsupervised learning, since they use both labeled and unlabeled data for training – typically a small amount of labeled data and a large amount of unlabeled data. The systems that use this method are able to considerably improve learning accuracy. Usually, semi-supervised learning is chosen when the acquired labeled data requires skilled and relevant resources in order to train it / learn from it. Otherwise, acquiring unlabeled data generally doesn't require additional resources.

## 2.5 Reinforcement learning

In reinforcement learning, the algorithm gets to choose an action in response to each data point. The learning algorithm also receives a reward signal a short time later, indicating how good the decision was. Based on this, the algorithm modifies its strategy in order to achieve the highest reward. Currently there are no reinforcement learning algorithm modules in Azure Machine Learning. Reinforcement learning is common in robotics, where the set of sensor readings at one point in time is a data point, and the algorithm must choose the robot's next action. It is also a natural fit for Internet of Things applications.

### Note:

Typically, **unlabeled** data consists of samples of natural or human-created artifacts that you can obtain relatively easily from the world. Some examples of unlabeled data might include photos, audio recordings, videos, news articles, tweets, x-rays (if you were working on a medical application), etc. There is no "explanation" for each piece of unlabeled data -- it just contains the data, and nothing else.

**Labeled** data typically takes a set of unlabeled data and augments each piece of that unlabeled data with some sort of meaningful "tag," "label," or "class" that is somehow informative or desirable to know. For example, labels for the above types of unlabeled data might be whether this photo contains a horse or a cow, which words were uttered in this audio recording, what type of action is being performed in this video, what the topic of this news article is, what the overall sentiment of this tweet is, whether the dot in this x-ray is a tumor, etc.

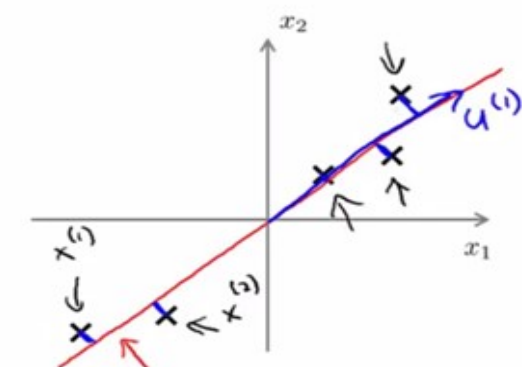
## 2.6 Principle component analysis

The main idea of principal component analysis (PCA) is to reduce the dimensionality of a data set consisting of many variables correlated with each other, either heavily or lightly, while retaining the variation present in the dataset, up to the maximum extent. The same is done by transforming the variables to a new set of variables, which are known as the principal components (or simply, the PCs) and are orthogonal, ordered such that the retention of variation present in the original variables decreases as we move down in the order. So, in this way, the 1st principal component retains maximum variation that was present in the original components. The principal components are the eigenvectors of a covariance matrix, and hence they are orthogonal.

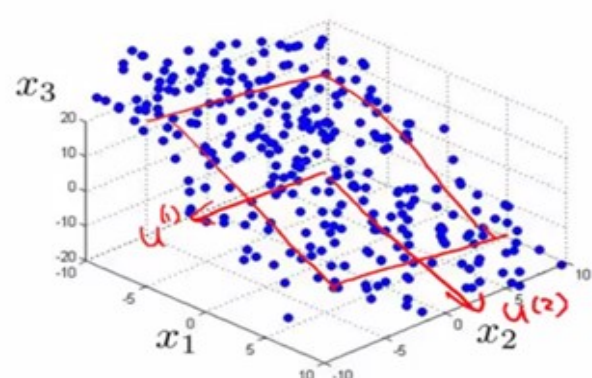
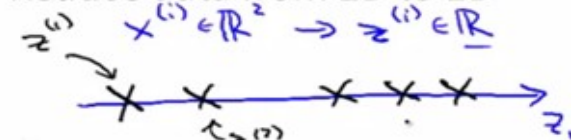
Importantly, the dataset on which PCA technique is to be used must be scaled. The results are also sensitive to the relative scaling. As a layman, it is a method of summarizing data. Imagine some wine bottles on a dining table. Each wine is described by its attributes like colour, strength, age, etc. But redundancy will arise because many of them will measure related properties. So what PCA will do in this case is summarize each wine in the stock with less characteristics.

Intuitively, Principal Component Analysis can supply the user with a lower-dimensional picture, a projection or "shadow" of this object when viewed from its most informative viewpoint.

### Principal Component Analysis (PCA) algorithm



Reduce data from 2D to 1D



Reduce data from 3D to 2D

## Properties of Principal Component

Technically, a principal component can be defined as a linear combination of optimally-weighted observed variables. The output of PCA are these principal components, the number of which is less than or equal to the number of original variables. Less, in case when we wish to discard or reduce the dimensions in our dataset. The PCs possess some useful properties which are listed below:

1. The PCs are essentially the linear combinations of the original variables, the weights vector in this combination is actually the eigenvector found which in turn satisfies the principle of least squares.
2. The PCs are orthogonal, as already discussed.
3. The variation present in the PCs decrease as we move from the 1st PC to the last one, hence the importance.

The least important PCs are also sometimes useful in regression, outlier detection, etc.

## Implementing PCA on a 2-D Dataset

### 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  $\bar{x}$  and all Y become  $\bar{y}$ . This produces a dataset whose mean is zero.

### Step 2: Calculate the covariance matrix

Since the dataset we took is 2-dimensional, this will result in a 2x2 Covariance matrix.

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

Please note that  $\text{Var}[X_1] = \text{Cov}[X_1, X_1]$  and  $\text{Var}[X_2] = \text{Cov}[X_2, X_2]$ .

### 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.  $\lambda$  is an eigenvalue for a matrix  $A$  if it is a solution of the characteristic equation:

$$\det(\lambda I - A) = 0$$

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  $\lambda$ , a corresponding eigen-vector  $v$ , can be found by solving:

$$(\lambda I - A)v = 0$$

### 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 of 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. In fact, only those eigenvectors which we want to proceed with. Since we just have 2 dimensions in the running example, we can either choose the one corresponding to the greater eigenvalue or simply take both.

*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.

$NewData = FeatureVector^T \times ScaledData^T$

Here,

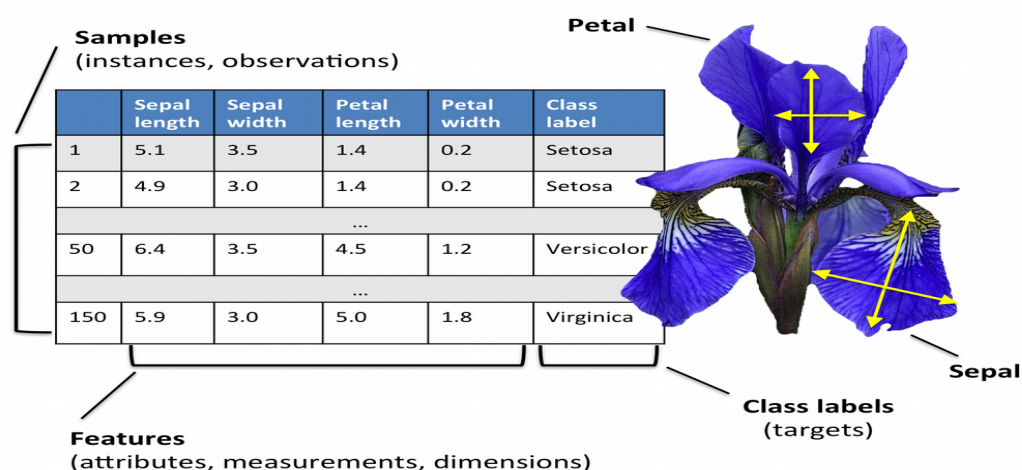
*NewData* is the Matrix consisting of the principal components,

*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  $2 \times 3$  matrix has a transpose of size  $3 \times 2$ )

### **Example**

#### **Preparing the Iris Dataset**



The diagram illustrates the structure of the Iris dataset. On the left, a table represents the dataset with 150 samples (rows) and 6 features (columns). The first two columns are 'Sepal length' and 'Sepal width', the next two are 'Petal length' and 'Petal width', and the last column is 'Class label'. The samples are grouped into three species: Setosa (samples 1-50), Versicolor (samples 51-100), and Virginica (samples 101-150). On the right, an image of an iris flower is shown with yellow arrows indicating the measurements for 'Petal' (length and width) and 'Sepal' (length and width). The 'Class labels (targets)' are also indicated.

Samples (instances, observations)	Sepal length	Sepal width	Petal length	Petal width	Class label
1	5.1	3.5	1.4	0.2	Setosa
2	4.9	3.0	1.4	0.2	Setosa
...	...	...	...	...	...
50	6.4	3.5	4.5	1.2	Versicolor
...	...	...	...	...	...
150	5.9	3.0	5.0	1.8	Virginica

**Features**  
(attributes, measurements, dimensions)

**Class labels**  
(targets)

**Petal**

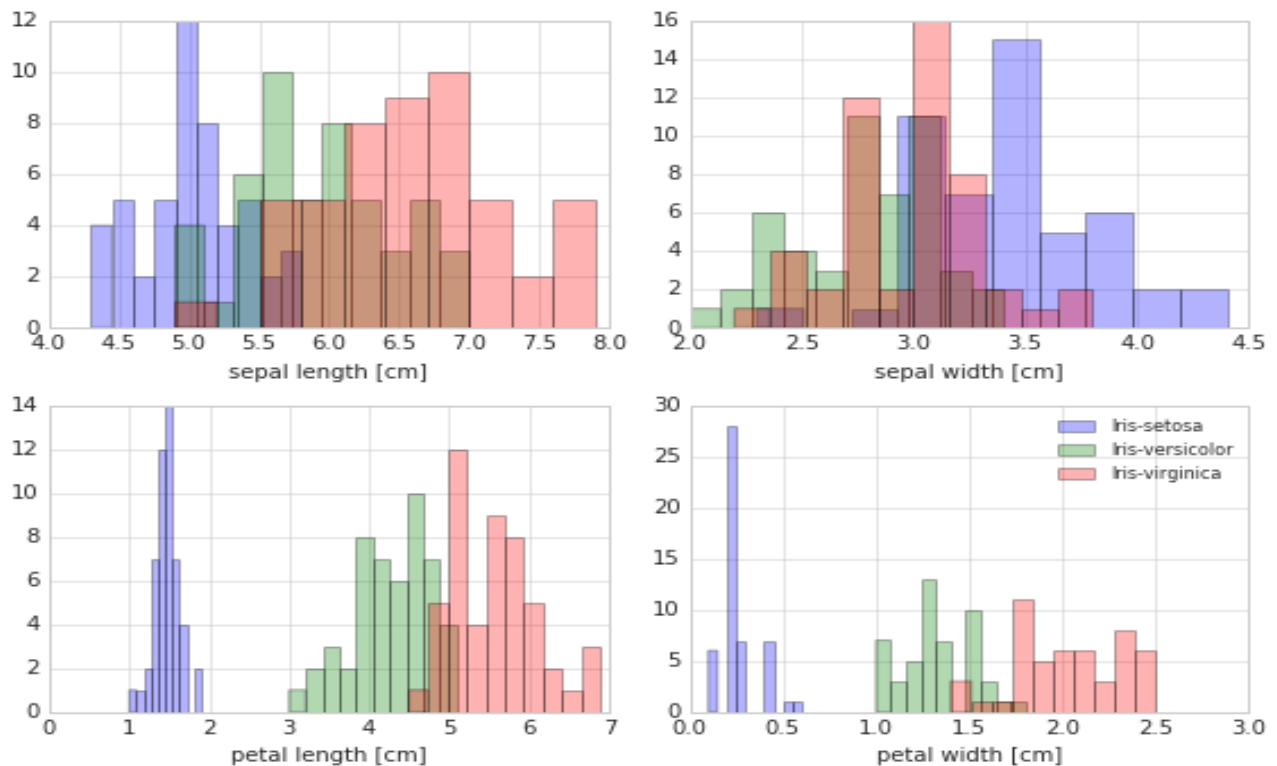
**Sepal**

### Step 01: Loading the Dataset

Our iris dataset is now stored in form of a  $150 \times 4$  matrix where the columns are the different features, and every row represents a separate flower sample. Each sample row  $\mathbf{x}$  can be pictured as a 4-dimensional vector

### Step 02: Exploratory Visualization

To get a feeling for how the 3 different flower classes are distributed along the 4 different features, let us visualize them via histograms.



### Step 03: Standardizing

Whether to standardize the data prior to a PCA on the covariance matrix depends on the measurement scales of the original features. Since PCA yields a feature subspace that maximizes the variance along the axes, it makes sense to standardize the data, especially, if it was measured on different scales. Although, all features in the Iris dataset were measured in centimeters, let us continue with the transformation of the data onto unit scale (mean=0 and variance=1), which is a requirement for the optimal performance of many machine learning algorithms.

```
from sklearn.preprocessing import StandardScaler
```

```
X_std = StandardScaler().fit_transform(X)
```

#### Step 04: Eigendecomposition - Computing Eigenvectors and Eigenvalues

The eigenvectors and eigenvalues of a covariance (or correlation) matrix represent the “core” of a PCA: The eigenvectors (principal components) determine the directions of the new feature space, and the eigenvalues determine their magnitude. In other words, the eigenvalues explain the variance of the data along the new feature axes.

#### Step 05: Finding covariance and correlation matrix

We can clearly see that all three approaches yield the same eigenvectors and eigenvalue pairs:

- Eigendecomposition of the covariance matrix after standardizing the data.
- Eigendecomposition of the correlation matrix.
- Eigendecomposition of the correlation matrix after standardizing the data.

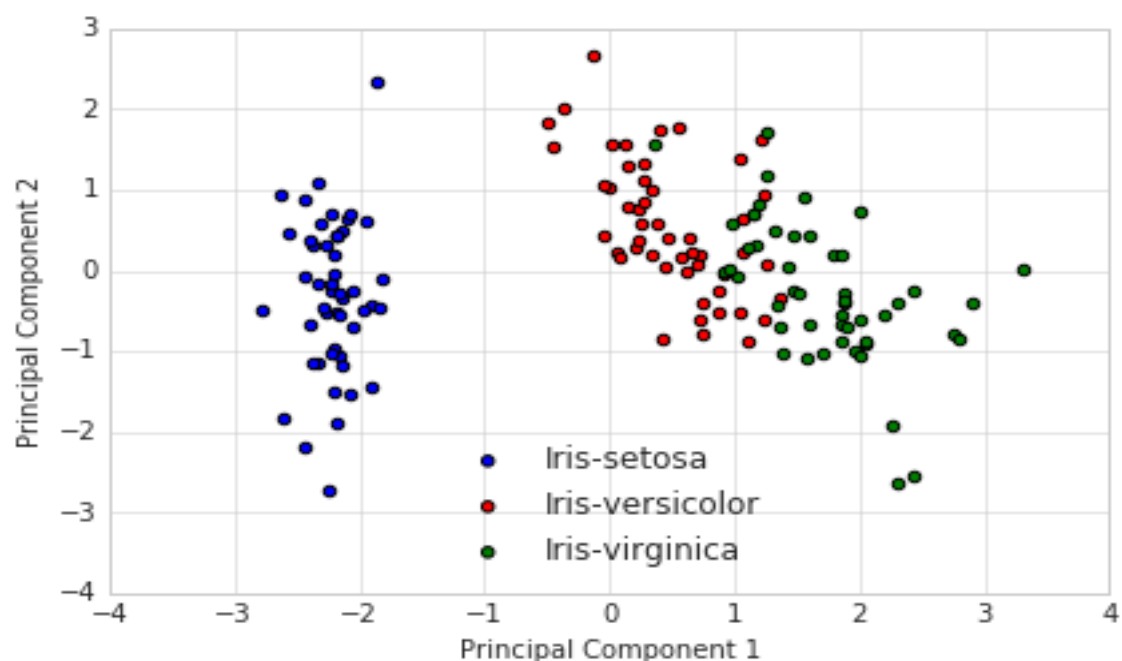
#### Step 06: Singular Vector Decomposition

While the eigendecomposition of the covariance or correlation matrix may be more intuitive, most PCA implementations perform a Singular Vector Decomposition (SVD) to improve the computational efficiency. So, let us perform an SVD to confirm that the result are indeed the same:

#### Step 07: Selecting Principal Components

- i. Sorting Eigenpairs
- ii. Explained Variance
- iii. Projection Matrix

#### Step 08: Projection Onto the New Feature Space



**Note:****PCA Vs. LDA**

Both Linear Discriminant Analysis (LDA) and PCA are linear transformation methods. PCA yields the directions (principal components) that maximize the variance of the data, whereas LDA also aims to find the directions that maximize the separation (or discrimination) between different classes, which can be useful in pattern classification problem (PCA “ignores” class labels). In other words, PCA projects the entire dataset onto a different feature (sub)space, and LDA tries to determine a suitable feature (sub)space in order to distinguish between patterns that belong to different classes.

**2.7 Random Forest Algorithm in Machine Learning**

*First, Random Forest algorithm is a supervised classification algorithm. We can see it from its name, which is to create a forest by some way and make it random. There is a direct relationship between the number of trees in the forest and the results it can get: the larger the number of trees, the more accurate the result. But one thing to note is that creating the forest is not the same as constructing the decision with information gain or gain index approach.*

**Random forests** or **random decision forests** are an ensemble learning method for classification, regression and other tasks, that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees.

The author gives 4 links to help people who are working with decision trees for the first time to learn it, and understand it well. The decision tree is a decision support tool. It uses a tree-like graph to show the possible consequences. If you input a training dataset with targets and features into the decision tree, it will formulate some set of rules. These rules can be used to perform predictions. The author uses one example to illustrate this point: suppose you want to predict whether your daughter will like an animated movie, you should collect the past animated movies she likes, and take some features as the input. Then, through the decision tree algorithm, you can generate the rules. You can then input the features of this movie and see whether it will be liked by your daughter. The process of calculating these nodes and forming the rules is using information gain and Gini index calculations.

The difference between Random Forest algorithm and the decision tree algorithm is that in Random Forest, the process of finding the root node and splitting the feature nodes will run randomly.

### **Real World Example**

In this section, the author gives us a real-life example to make the Random Forest algorithm easy to understand. Suppose Mady wants to go to different places that he may like for his two-week vacation, and he asks his friend for advice. His friend will ask where he has been to already, and whether he likes the places that he's visited. Based on Mady's answers, his friend starts to give the recommendation. Here, his friend forms the decision tree.

Mady wants to ask more friends for advice because he thinks only one friend cannot help him make an accurate decision. So his other friends also ask him random questions, and finally, provides an answer. He considers the place with the most votes as his vacation decision. Here, the author provides an analysis for this example.

His one friend asked him some questions and gave the recommendation of the best place based on the answers. This is a typical decision tree algorithm approach. The friend created the rules based on the answers and used the rules to find the answer that matched the rules.

Mady's friends also randomly asked him different questions and gave answers, which for Mady are the votes for the place. At the end, the place with the highest votes is the one Mady will select to go. This is the typical Random Forest algorithm approach.

### **How Random Forest algorithm works?**

There are two stages in Random Forest algorithm, one is random forest creation, the other is to make a prediction from the random forest classifier created in the first stage. The whole process is shown below, and it's easy to understand using the figure.

Here the author firstly shows the Random Forest creation pseudocode:

1. Randomly select "K" features from total "m" features where  $k \ll m$
2. Among the "K" features, calculate the node "d" using the best split point
3. Split the node into daughter nodes using the best split
4. Repeat the a to c steps until "l" number of nodes has been reached
5. Build forest by repeating steps a to d for "n" number times to create "n" number of trees

## Prediction in Random Forest

In the next stage, with the random forest classifier created, we will make the prediction. The random forest prediction pseudocode is shown below:

1. Takes the **test features** and use the rules of each randomly created decision tree to predict the outcome and stores the predicted outcome (target)
2. Calculate the **votes** for each predicted target
3. Consider the **high voted** predicted target as the **final prediction** from the random forest algorithm

## Application of Random Forest

Below are some the application where random forest algorithm is widely used.

1. Banking
2. Medicine
3. Stock Market
4. E-commerce

Let's begin with the banking sector.

### 1. Banking:

In the banking sector, random forest algorithm widely used in two main application. These are for finding the loyal customer and finding the fraud customers.

The loyal customer means not the customer who pays well, but also the customer whom can take the huge amount as loan and pays the loan interest properly to the bank. As the growth of the bank purely depends on the loyal customers. The bank customers data highly analyzed to find the pattern for the loyal customer based the customer details.

In the same way, there is need to identify the customer who are not profitable for the bank, like taking the loan and paying the loan interest properly or find the outlier customers. If the bank can identify theses kind of customer before giving the loan the customer. Bank will get a chance to not approve the loan to these kinds of customers. In this case, also random forest algorithm is used to identify the customers who are not profitable for the bank.

## **2.Medicine**

In medicine field, random forest algorithm is used identify the correct combination of the components to validate the medicine. Random forest algorithm also helpful for identifying the disease by analyzing the patient's medical records.

## **3.Stock Market**

In the stock market, random forest algorithm used to identify the stock behavior as well as the expected loss or profit by purchasing the particular stock.

## **4.E-commerce**

In e-commerce, the random forest used only in the small segment of the recommendation engine for identifying the likely hood of customer liking the recommend products base on the similar kinds of customers.

Running random forest algorithm on very large dataset requires high-end GPU systems. If you are not having any GPU system. You can always run the machine learning models in cloud hosted desktop. You can use clouddesktoponline platform to run high-end machine learning models from sitting any corner of the world.

## **Features of Random Forests**

It is unexcelled in accuracy among current algorithms.

It runs efficiently on large data bases.

It can handle thousands of input variables without variable deletion.

It gives estimates of what variables are important in the classification.

It generates an internal unbiased estimate of the generalization error as the forest building progresses.

It has an effective method for estimating missing data and maintains accuracy when a large proportion of the data are missing.

It has methods for balancing error in class population unbalanced data sets.

Generated forests can be saved for future use on other data.

Prototypes are computed that give information about the relation between the variables and the classification.

It computes proximities between pairs of cases that can be used in clustering, locating outliers, or (by scaling) give interesting views of the data.

The capabilities of the above can be extended to unlabeled data, leading to unsupervised clustering, data views and outlier detection.

It offers an experimental method for detecting variable interactions.

## 2.8 Efficiently Finding Frequent with FP-Growth

The Frequent Pattern (FP)-Growth method is used with databases and not with streams. The Apriori algorithm needs  $n+1$  scans if a database is used, where  $n$  is the length of the longest pattern. By using the FP-Growth method, the number of scans of the entire database can be reduced to two. The algorithm extracts frequent item sets that can be used to extract association rules. This is done using the *support* of an item set.

The FP -growth algorithm scans the dataset only twice. The basic approach to finding frequent itemsets using the FP -growth algorithm is as follows:

1. Build the FP -tree.
2. Mine frequent itemsets from the FP -tree.

TID	Items
1	{a,b}
2	{b,c,d}
3	{a,c,d,e}
4	{a,d,e}
5	{a,b,c}
6	{a,b,c,d}
7	{a}
8	{a,b,c}
9	{a,b,d}
10	{b,c,e}

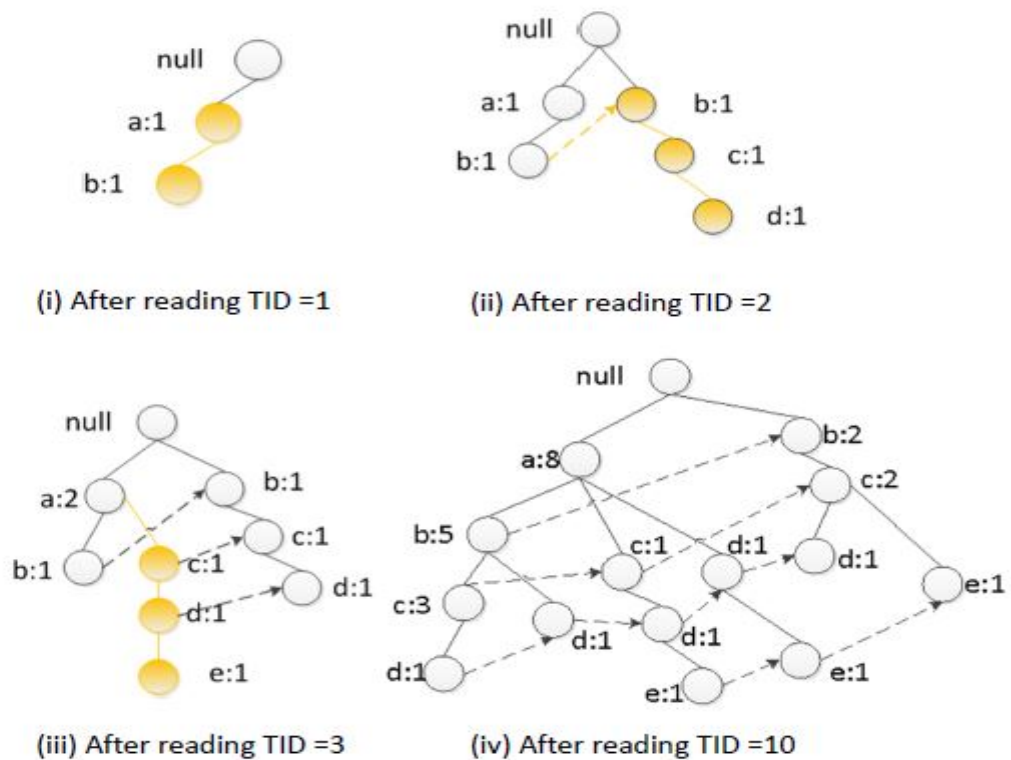


Fig. 7. Construction of an FP-tree - Based on [28]



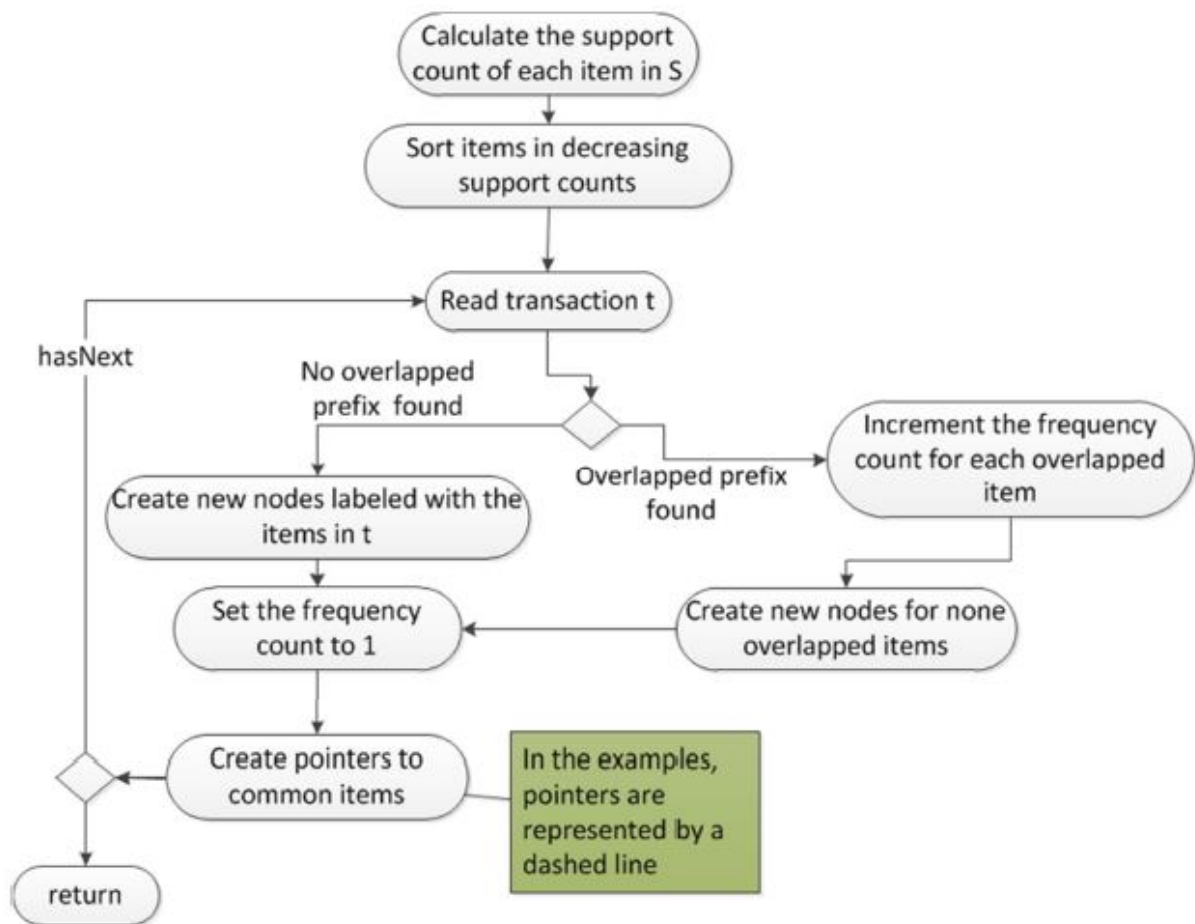


Fig. 8. Activity diagram - Construction of the FP-Tree - Notation: [27]

### Example: finding co-occurring words in a Twitter feed

1. Collect: Use the python-twitter module to access tweets.
2. Prepare: Write a function to remove URLs, remove punctuation, convert to lower-case, and create a set of words from a string.
3. Analyze: We'll look at the prepared data in the Python shell to make sure it's correct.
4. Train: We'll use createTree() and mineTree(), developed earlier in this chapter, to perform the FP-growth algorithm.
5. Test: Doesn't apply.
6. Use: Not performed in this example. You could do sentiment analysis or provide search query suggestion.

### 2.8 Some Random Topics

## 1. Confusion Matrix

In the field of machine learning and specifically the problem of statistical classification, a confusion matrix, also known as an error matrix.

A confusion matrix is a table that is often used to describe the performance of a classification model (or “classifier”) on a set of test data for which the true values are known. It allows the visualization of the performance of an algorithm.

It allows easy identification of confusion between classes e.g. one class is commonly mislabeled as the other. Most performance measures are computed from the confusion matrix.

Positives and Negatives

In a Confusion Matrix we have things called False Positives and False Negatives. These are defined as:

False Positive: Falsely Predicting an event (or saying a Cat is a Dog)

False Negative: Missing and incoming event (or saying a Dog is a Cat)

As well as these, we also have True Positives and True Negatives. Remembering that a Cat=0 and Dog=1, a Confusion Matrix would look like this:

		Prediction	
		0	1
Actual	0	TN	FP
	1	FN	TP

So using this, lets use the example that we have 50 Cats and 50 Dogs. We see that we have predicted only 30% of Cats as Cats and 20% of Dogs as Dogs. In a Confusion Matrix this would look like:

		Prediction	
		Cat	Dog
Actual	Cat	15	35
	Dog	40	10

So here we can see that we predicted 15 Cats as Cats, yet we predicted 35 Cats and Dogs. We can also see that we predicted 40 Dogs as Cats and only 10 Dogs as Dogs. (in reality these numbers are low... i think...)

15 Cats as Cats (TN) – the N because Cat=0

35 Cats as Dogs (FP)

40 Dogs as Cats (FN)

10 Dogs as Dogs (TP)

Sensitivity and Specificity

Now we have 2 equations to note

The first one is Sensitivity which is:

$TP / (TP+FN)$

and then Specificity which is:

$TN / (TN+FP)$

In general here, Sensitivity means the accuracy on the class Dog, and Specificity means the accuracy on the class Cat.

So using this, what is the accuracy on Dogs and Cats? Well:

Sensitivity =  $TP / (TP+FN) = 10/50 = 0.2 = 20\%$  (which is correct)

Specificity =  $TN / (TN+FP) = 15/50 = 0.3 = 30\%$  (which is also correct!)

Let's start with an **example confusion matrix for a binary classifier** (though it can easily be extended to the case of more than two classes):

n=165	Predicted: NO	Predicted: YES
	Actual: NO	Actual: YES
	50	10
	5	100

What can we learn from this matrix?

- There are two possible predicted classes: "yes" and "no". If we were predicting the presence of a disease, for example, "yes" would mean they have the disease, and "no" would mean they don't have the disease.
- The classifier made a total of 165 predictions (e.g., 165 patients were being tested for the presence of that disease).
- Out of those 165 cases, the classifier predicted "yes" 110 times, and "no" 55 times.
- In reality, 105 patients in the sample have the disease, and 60 patients do not.

Let's now define the most basic terms, which are whole numbers (not rates):

- **true positives (TP):** These are cases in which we predicted yes (they have the disease), and they do have the disease.
- **true negatives (TN):** We predicted no, and they don't have the disease.

- false positives (FP):** We predicted yes, but they don't actually have the disease. (Also known as a "Type I error.")
- false negatives (FN):** We predicted no, but they actually do have the disease. (Also known as a "Type II error.")

I've added these terms to the confusion matrix, and also added the row and column totals:

n=165		Predicted: NO	Predicted: YES	
Actual: NO		TN = 50	FP = 10	60
Actual: YES		FN = 5	TP = 100	105
		55	110	

This is a list of rates that are often computed from a confusion matrix for a binary classifier:

- Accuracy:** Overall, how often is the classifier correct?
- $(TP+TN)/total = (100+50)/165 = 0.91$
- Misclassification Rate:** Overall, how often is it wrong?
- $(FP+FN)/total = (10+5)/165 = 0.09$
- equivalent to 1 minus Accuracy
- also known as "Error Rate"
- True Positive Rate:** When it's actually yes, how often does it predict yes?
- $TP/actual\ yes = 100/105 = 0.95$
- also known as "Sensitivity" or "Recall"
- False Positive Rate:** When it's actually no, how often does it predict yes?
- $FP/actual\ no = 10/60 = 0.17$
- Specificity:** When it's actually no, how often does it predict no?
- $TN/actual\ no = 50/60 = 0.83$
- equivalent to 1 minus False Positive Rate
- Precision:** When it predicts yes, how often is it correct?
- $TP/predicted\ yes = 100/110 = 0.91$
- Prevalence:** How often does the yes condition actually occur in our sample?
- $actual\ yes/total = 105/165 = 0.64$

A couple other terms are also worth mentioning:

•**Positive Predictive Value:** This is very similar to precision, except that it takes prevalence into account. In the case where the classes are perfectly balanced (meaning the prevalence is 50%), the positive predictive value (PPV) is equivalent to precision. (**More details about PPV.**)

•**Null Error Rate:** This is how often you would be wrong if you always predicted the majority class. (In our example, the null error rate would be  $60/165=0.36$  because if you always predicted yes, you would only be wrong for the 60 "no" cases.) This can be a useful baseline metric to compare your classifier against. However, the best classifier for a particular application will sometimes have a higher error rate than the null error rate, as demonstrated by the **Accuracy Paradox**.

•**Cohen's Kappa:** This is essentially a measure of how well the classifier performed as compared to how well it would have performed simply by chance. In other words, a model will have a high Kappa score if there is a big difference between the accuracy and the null error rate. (**More details about Cohen's Kappa.**)

•**F Score:** This is a weighted average of the true positive rate (recall) and precision. (**More details about the F Score.**)

•**ROC Curve:** This is a commonly used graph that summarizes the performance of a classifier over all possible thresholds. It is generated by plotting the True Positive Rate (y-axis) against the False Positive Rate (x-axis) as you vary the threshold for assigning observations to a given class.

High recall, low precision: This means that most of the positive examples are correctly recognized (low FN) but there are a lot of false positives.

Low recall, high precision: This shows that we miss a lot of positive examples (high FN) but those we predict as positive are indeed positive (low FP)

F-measure:

Since we have two measures (Precision and Recall) it helps to have a measurement that represents both of them. We calculate an F-measure which uses Harmonic Mean in place of Arithmetic Mean as it punishes the extreme values more. The F-Measure will always be nearer to the smaller value of Precision or Recall.

$$F - measure = \frac{2 * Recall * Precision}{Recall + Precision}$$

## 2. Receiver Operating Characteristic Curve

ROC curves assess predictive behavior independent of error costs or class distributions. It has following characteristics.

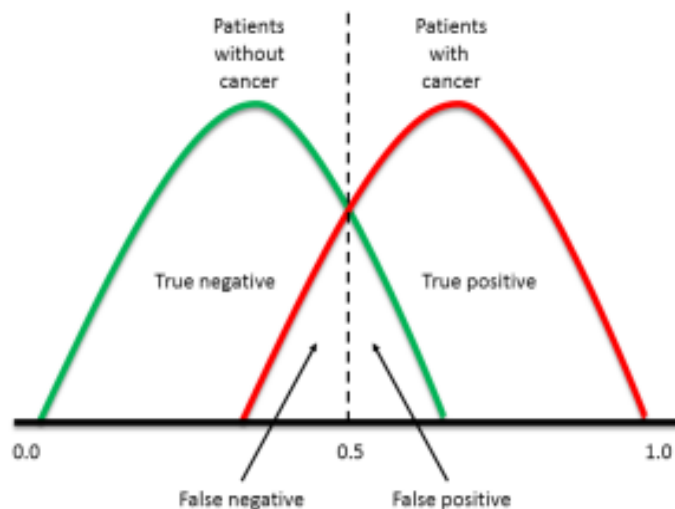
- i. Originated from signal detection theory
- ii. Common in medical diagnosis
- iii. Becoming common in ML evaluations

### Some Performance Metrics

In a previous blog post we discussed some of the other performance metrics which can be applied to the assessment of a classifier. To review:

Most classifiers produce a score, which is then thresholded to decide the classification. If a classifier produces a score between 0.0 (definitely negative) and 1.0 (definitely positive), it is common to consider anything over 0.5 as positive.

However, any threshold applied to a dataset (in which PP is the positive population and NP is the negative population) is going to produce true positives (TP), false positives (FP), true negatives (TN) and false negatives (FN) (Figure 1). We need a method which will take into account all of these numbers.



Once you have numbers for all of these measures, some useful metrics can be calculated.

• **Accuracy** =  $(1 - \text{Error}) = (TP + TN)/(PP + NP) = \text{Pr}(C)$ , the probability of a correct classification.

• **Sensitivity** =  $TP/(TP + FN) = TP/PP$  = the ability of the test to detect disease in a population of diseased individuals.

• **Specificity** =  $TN/(TN + FP) = TN / NP$  = the ability of the test to correctly rule out the disease in a disease-free population.

Let's calculate these metrics for some reasonable real-world numbers. If we have 100,000 patients, of which 200 (20%) actually have cancer, we might see the following test results (Table 1):

	Test Positive	Test Negative	Total
<b>Patient Diseased</b>	160	40	200
<b>Patient Healthy</b>	29940	69860	99800
<b>Total</b>	30100	69900	100000

Table 1. Illustration of diagnostic test performance for "reasonable" values for Pap smear screening

For this data:

• **Sensitivity** =  $TP / (TP + FN) = 160 / (160 + 40) = 80.0\%$

**Specificity** =  $TN / (TN + FP) = 69,860 / (69,860 + 29,940) = 70.0\%$

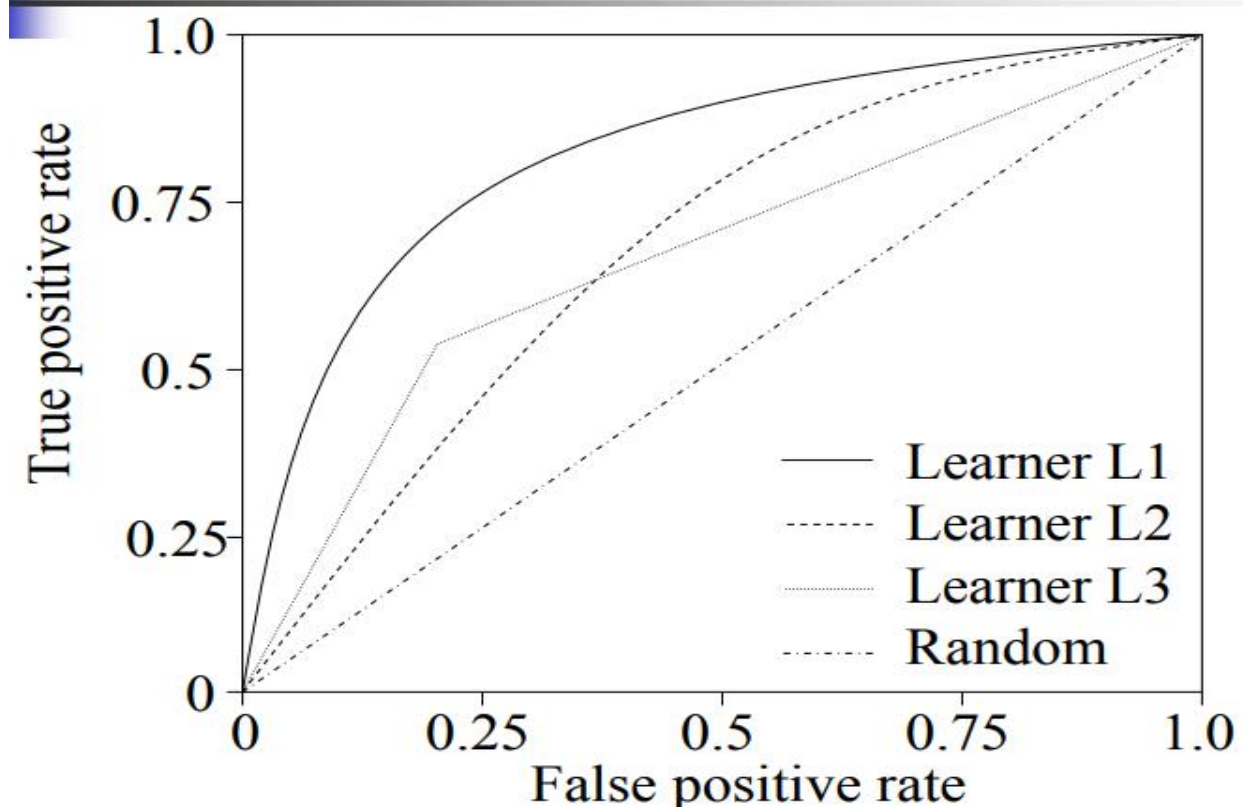
In other words, our test will correctly identify 80% of people with the disease, but 30% of healthy people will incorrectly test positive. By only considering the sensitivity (or accuracy) of the test, potentially important information is lost.

### ROC Curve

ROC space \_\_ False positive (FP) rate on X axis \_\_ True positive (TP) rate on Y axis \_\_

Each classifier represented by a point in ROC space corresponding to its (FP,TP) pair \_\_ For continuous-output models, classifiers defined based on varying thresholds on output.

## Example ROC Curve



### Domination in ROC Space

- Learner L1 dominates L2 is L2's ROC curve is beneath L1's curve
- If L1 dominates L2, then L1 better than L2 for all possible costs and class distributions
- If neither dominates (L2 and L3), then there are times when L2 maximizes accuracy, but does not minimize cost.

### 3. One Hot Encoding

A one hot encoding is a representation of categorical variables as binary vectors.

What is Categorical Data?

Categorical data are variables that contain label values rather than numeric values.

The number of possible values is often limited to a fixed set.

Categorical variables are often called nominal.

Some examples include:

- A “pet” variable with the values: “dog” and “cat”.
- A “color” variable with the values: “red”, “green” and “blue”.
- A “place” variable with the values: “first”, “second” and “third”.

Each value represents a different category.

Some categories may have a natural relationship to each other, such as a natural ordering.

The “place” variable above does have a natural ordering of values. This type of categorical variable is called an ordinal variable.

### **What is the Problem with Categorical Data?**

Some algorithms can work with categorical data directly.

For example, a decision tree can be learned directly from categorical data with no data transform required (this depends on the specific implementation).

Many machine learning algorithms cannot operate on label data directly. They require all input variables and output variables to be numeric.

In general, this is mostly a constraint of the efficient implementation of machine learning algorithms rather than hard limitations on the algorithms themselves.

This means that categorical data must be converted to a numerical form. If the categorical variable is an output variable, you may also want to convert predictions by the model back into a categorical form in order to present them or use them in some application.

### **How to Convert Categorical Data to Numerical Data?**

This involves two steps:

1. Integer Encoding
2. One-Hot Encoding

#### **1. Integer Encoding**

As a first step, each unique category value is assigned an integer value.

For example, “red” is 1, “green” is 2, and “blue” is 3.

This is called a label encoding or an integer encoding and is easily reversible.

For some variables, this may be enough.

The integer values have a natural ordered relationship between each other and machine learning algorithms may be able to understand and harness this relationship.



For example, ordinal variables like the “place” example above would be a good example where a label encoding would be sufficient.

## 2. One-Hot Encoding

For categorical variables where no such ordinal relationship exists, the integer encoding is not enough.

In fact, using this encoding and allowing the model to assume a natural ordering between categories may result in poor performance or unexpected results (predictions halfway between categories).

In this case, a one-hot encoding can be applied to the integer representation. This is where the integer encoded variable is removed and a new binary variable is added for each unique integer value.

In the “color” variable example, there are 3 categories and therefore 3 binary variables are needed. A “1” value is placed in the binary variable for the color and “0” values for the other colors.

For example:

	red, green, blue
1	1, 0, 0
2	0, 1, 0
3	0, 0, 1

The binary variables are often called “dummy variables” in other fields, such as statistics.

## 4. Cross Validation

### What is Cross Validation?

Cross Validation is a technique which involves reserving a particular sample of a data set on which you do not train the model. Later, you test the model on this sample before finalizing the model.

Here are the steps involved in cross validation:

1. You *reserve* a sample data set.
2. Train the model using the remaining part of the data set.
3. Use the reserve sample of the data set test (validation) set. This will help you to know the effectiveness of model performance. If your model delivers a positive result on validation data, go ahead with current model. It rocks!

What are common methods used for Cross Validation ?

There are various methods of cross validation. I’ve discussed few of them below:

### i. The Validation set Approach

In this approach, we reserve 50% of dataset for validation and rest 50% for model training. After testing the model performance on validation data set. However, a major disadvantage of this approach is that we train a model on 50% of the data set only, whereas, it may be possible that we are leaving some interesting information about data i.e. higher bias.

## ii. Leave one out cross validation (LOOCV)

In this approach, we reserve only one data-point of the available data set. And, train model on the rest of data set. This process iterates for each data point. It is also known for its advantages and disadvantages. Let's look at them:

- We make use of all data points, hence low bias.
- We repeat the cross validation process iterates  $n$  times (where  $n$  is number of data points) which results in higher execution time
- This approach leads to higher variation in testing model effectiveness because we test against one data point. So, our estimation gets highly influenced by the data point. If the data point turns out to be an outlier, it can lead to higher variation.

## iii. k-fold cross validation

From the above two validation methods, we've learnt:

1. We should train model on large portion of data set. Else, we'd fail every time to read the underlying trend of data sets. Eventually, resulting in higher bias.
2. We also need a good ratio testing data points. As, we have seen that lower data points can lead to variance error while testing the effectiveness of model.
3. We should iterate on training and testing process multiple times. We should change the train and test data set distribution. This helps to validate the model effectiveness well.

Do we have a method which takes care of all these 3 requirements ?

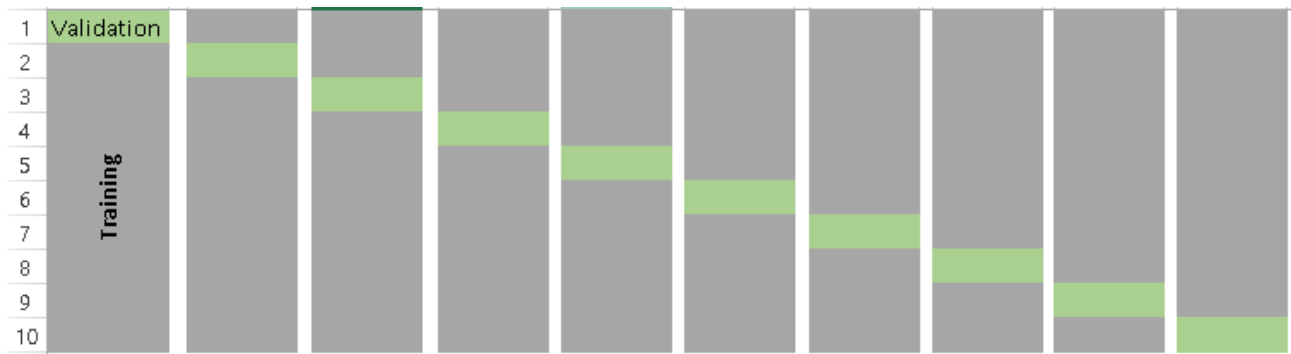
Yes! That method is known as "**k- fold cross validation**". It's easy to follow and implement. Here are the quick steps:

1. Randomly split your entire dataset into  $k$  "folds".
2. For each  $k$  folds in your dataset, build your model on  $k - 1$  folds of the data set. Then, test the model to check the effectiveness for  $k$ th fold.
3. Record the error you see on each of the predictions.
4. Repeat this until each of the  $k$  folds has served as the test set.
5. The average of your  $k$  recorded errors is called the cross-validation error and will serve as your performance metric for the model.

Below is the visualization of how does a  $k$ -fold validation work for  $k=10$ .

Now, one of most commonly asked question is, "**How to choose right value of  $k$ ?**"

Always remember, lower value of  $K$  is more biased and hence undesirable. On the other hand, higher value of  $K$  is less biased, but can suffer from large variability. It is good to know that, smaller value of  $k$  always takes us towards validation set approach, where as higher value of  $k$  leads to LOOCV approach. Hence, it is often suggested to use  $k=10$ .



### How to measure the model's bias-variance?

After k-fold cross validation, we'll get k different model estimation errors ( $e_1, e_2, \dots, e_k$ ). In ideal scenario, these error values should add to zero. To return the model's bias, we take the average of all the errors. Lower the average value, better the model.

Similarly for calculating model's variance, we take standard deviation of all errors. Lower value of standard deviation suggests our model does not vary a lot with different subset of training data.

We should focus on achieving a balance between bias and variance. This can be done by reducing the variance and controlling bias to an extent. It'll result in better predictive model. This trade-off usually leads to building less complex predictive models.

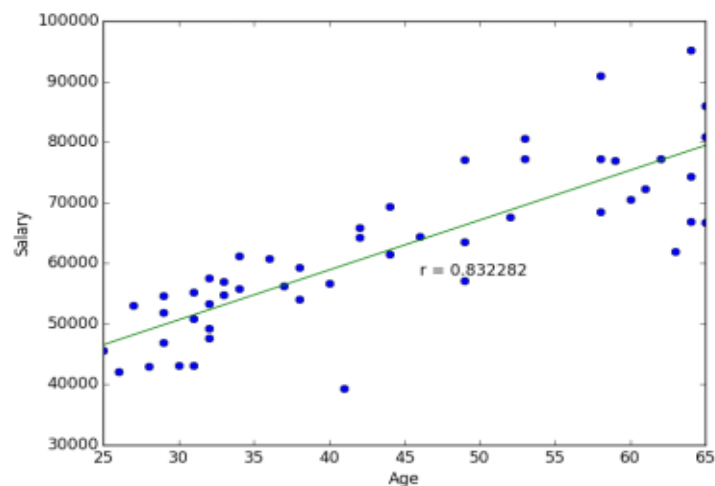
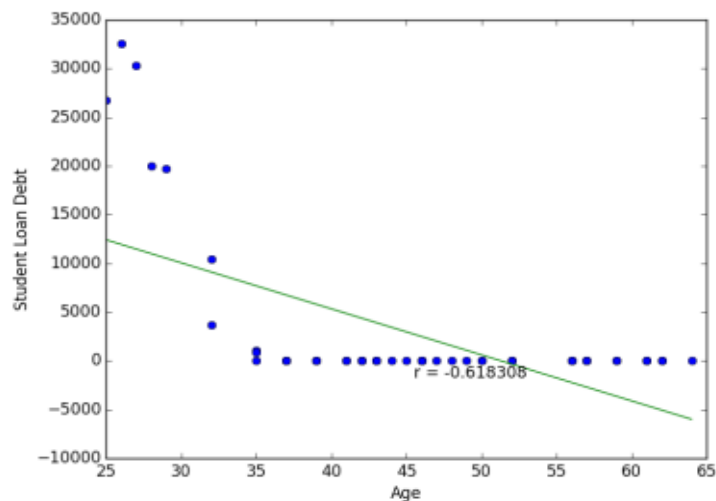
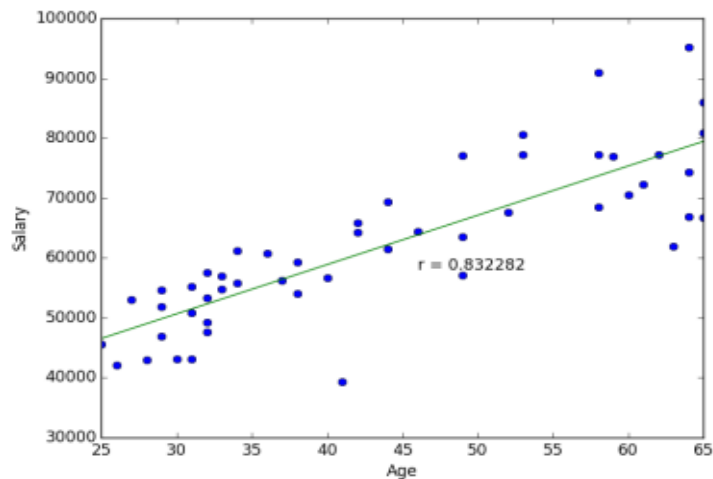
### 5. Correlation

correlation is a measure of how strongly one variable depends on another. Consider a hypothetical dataset containing information about professionals in the software industry. We might expect a strong relationship between age and salary, since senior project managers will tend to be paid better than young pup engineers. On the other hand, there is probably a very weak, if any, relationship between shoe size and salary. Correlations can be positive or negative. Our age and salary example is a case of positive correlation. Individuals with a higher age would also tend to have a higher salary. An example of negative correlation might be age compared to outstanding student loan debt: typically older people will have more of their student loans paid off.

Correlation can be an important tool for **feature engineering** in building machine learning models. Predictors which are uncorrelated with the objective variable are probably good candidates to trim from the model (shoe size is not a useful predictor for salary). In addition, if two predictors are strongly correlated to each other, then we only need to use one of them (in predicting salary, there is no need to use both age in years, and age in months). Taking these steps means that the resulting model will be simpler, and simpler models are easier to interpret.

There are many measures for correlation, but by far the most widely used one is **Pearson's Product-Moment coefficient**, or Pearson's r. Given a collection of paired (x,y) values, Pearson's

coefficient produces a value between -1 and +1 to quantify the strength of dependence between the variables  $x$  and  $y$ . A value of +1 means that all the  $(x,y)$  points lie exactly on a line with positive slope, and inversely, a value of -1 means that all of the points lie exactly on a line with negative slope. A Pearson's coefficient of 0 means that there is no relationship between the two variables. To see this visually, we can look at plots of our hypothetical data, and the Pearson's coefficient computed from them.



## 6. Eigendecomposition of a covariance matrix

In the next section, we will discuss how the covariance matrix can be interpreted as a linear operator that transforms white data into the data we observed. However, before diving into the technical details, it is important to gain an intuitive understanding of how eigenvectors and eigenvalues uniquely define the covariance matrix, and therefore the shape of our data.

As we saw in figure 3, the covariance matrix defines both the spread (variance), and the orientation (covariance) of our data. So, if we would like to represent the covariance matrix with a vector and its magnitude, we should simply try to find the vector that points into the direction of the largest spread of the data, and whose magnitude equals the spread (variance) in this direction.

**If we define this vector as  $\vec{v}$ , then the projection of our data  $D$  onto this vector is obtained as  $\vec{v}^T D$ , and the variance of the projected data is  $\vec{v}^T \Sigma \vec{v}$ . Since we are looking for the vector  $\vec{v}$  that points into the direction of the largest variance, we should choose its components such that the covariance matrix  $\vec{v}^T \Sigma \vec{v}$  of the projected data is as large as possible. Maximizing any function of the form  $\vec{v}^T \Sigma \vec{v}$  with respect to  $\vec{v}$ , where  $\vec{v}$  is a normalized unit vector, can be formulated as a so called Rayleigh Quotient. The maximum of such a Rayleigh Quotient is obtained by setting  $\vec{v}$  equal to the largest eigenvector of matrix  $\Sigma$ .**

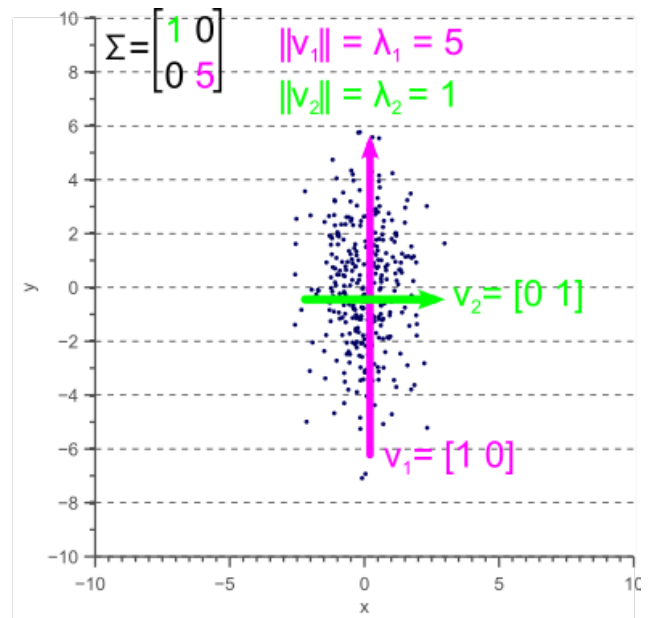
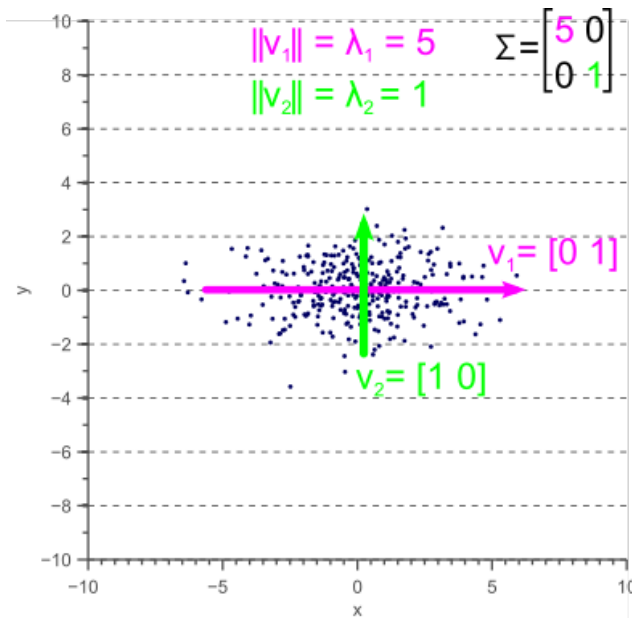
In other words, the largest eigenvector of the covariance matrix always points into the direction of the largest variance of the data, and the magnitude of this vector equals the corresponding eigenvalue. The second largest eigenvector is always orthogonal to the largest eigenvector, and points into the direction of the second largest spread of the data.

**Now let's have a look at some examples. In an earlier article we saw that a linear transformation matrix  $T$  is completely defined by its eigenvectors and eigenvalues. Applied to the covariance matrix, this means that:**

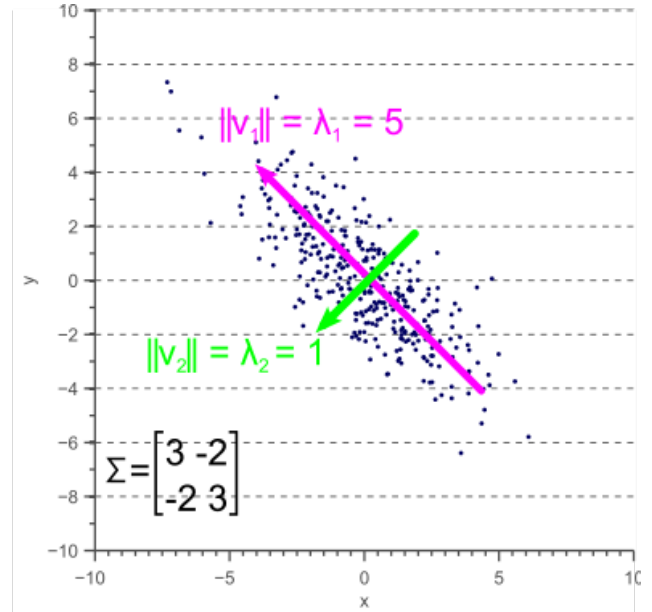
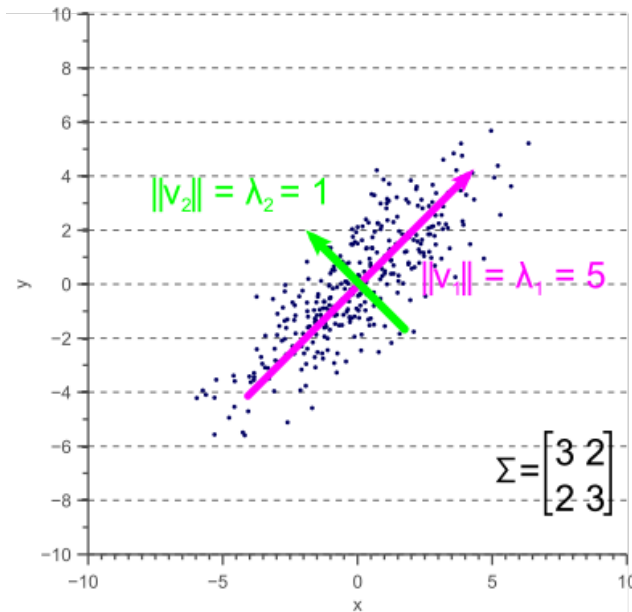
$$(4) \Sigma \vec{v} = \lambda \vec{v}$$

where  $\vec{v}$  is an eigenvector of  $\Sigma$ , and  $\lambda$  is the corresponding eigenvalue.

If the covariance matrix of our data is a diagonal matrix, such that the covariances are zero, then this means that the variances must be equal to the eigenvalues  $\lambda$ . This is illustrated by figure 4, where the eigenvectors are shown in green and magenta, and where the eigenvalues clearly equal the variance components of the covariance matrix.



However, if the covariance matrix is not diagonal, such that the covariances are not zero, then the situation is a little more complicated. The eigenvalues still represent the variance magnitude in the direction of the largest spread of the data, and the variance components of the covariance matrix still represent the variance magnitude in the direction of the x-axis and y-axis. But since the data is not axis aligned, these values are not the same anymore as shown by figure 5.



By comparing figure 5 with figure 4, it becomes clear that the eigenvalues represent the variance of the data along the eigenvector directions, whereas the variance components of the covariance matrix represent the spread along the axes. If there are no covariances, then both values are equal.

## 7. Underfitting and Overfitting in Machine Learning

Let us consider that we are designing a machine learning model. A model is said to be a good machine learning model, if it generalizes any new input data from the problem domain in a proper way. This helps us to make predictions in the future data, that data model has never seen.

Now, suppose we want to check how well our machine learning model learns and generalizes to the new data. For that we have overfitting and underfitting, which are majorly responsible for the poor performances of the machine learning algorithms.

### Underfitting:

A statistical model or a machine learning algorithm is said to have underfitting when it cannot capture the underlying trend of the data. (*It's just like trying to fit undersized pants!*) Underfitting destroys the accuracy of our machine learning model. Its occurrence simply means that our model or the algorithm does not fit the data well enough. It usually happens when we have less data to build an accurate model and also when we try to build a linear model with a non-linear data. In such cases the rules of the machine learning model are too easy and flexible to be applied on such a minimal data and therefore the model will probably make a lot of wrong predictions. Underfitting can be avoided by using more data and also reducing the features by feature selection.

### Overfitting:

A statistical model is said to be overfitted, when we train it with a lot of data (*just like fitting ourselves in an oversized pants!*). When a model gets trained with so much of data, it starts learning from the noise and inaccurate data entries in our data set. Then the model does not categorize the data correctly, because of too much of details and noise. The causes of overfitting are the non-parametric and non-linear methods because these types of machine learning algorithms have more freedom in building the model based on the dataset and therefore they can really build unrealistic models. A solution to avoid overfitting is using a linear algorithm if we have linear data or using the parameters like the maximal depth if we are using decision trees.

### **If our model does much better on the training set than on the test set, then we're likely overfitting.**

For example, it would be a big red flag if our model saw 99% accuracy on the training set but only 55% accuracy on the test set.

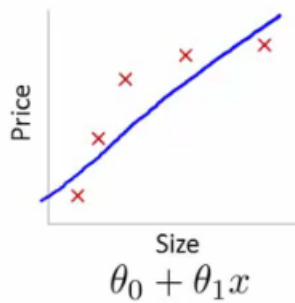
Training Set

Test Set

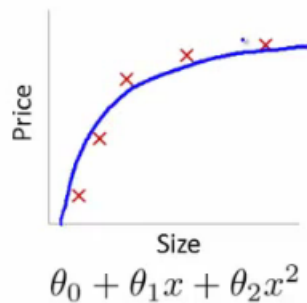
Train and tune your models  
(using cross-validation)

Don't touch this until the very end.

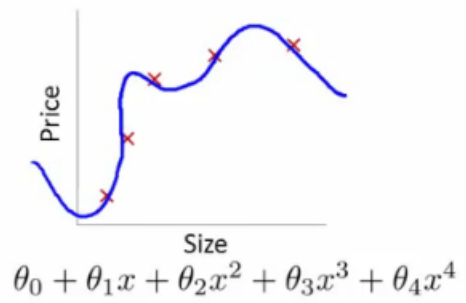
Examples:



High bias  
(underfit)

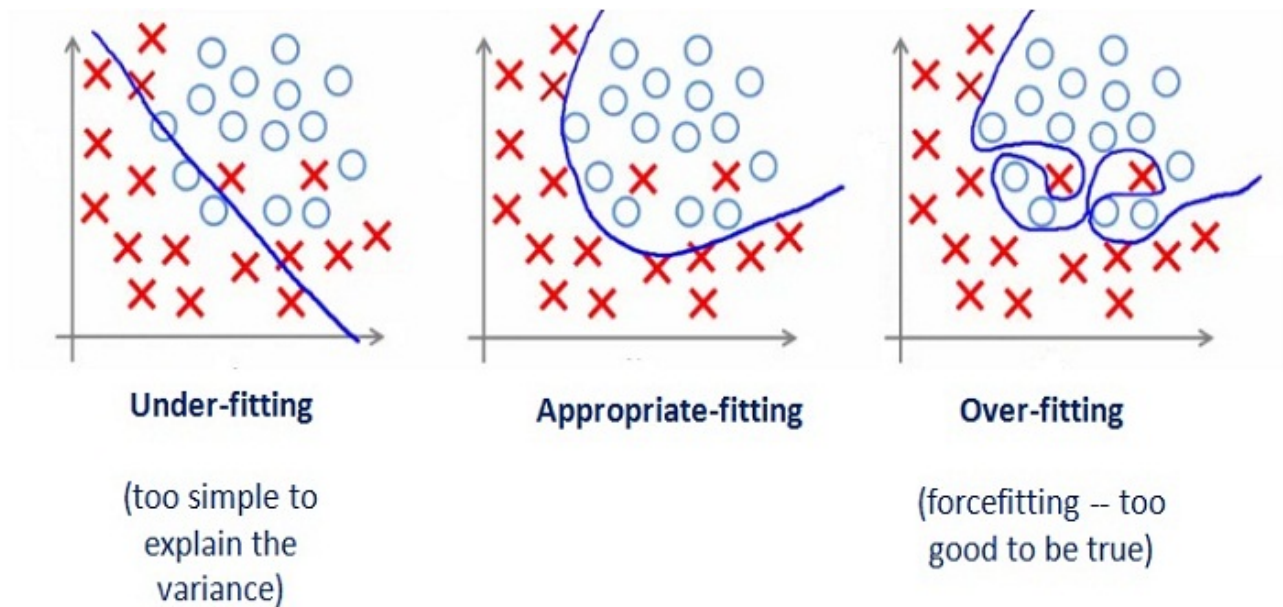


"Just right"



High variance  
(overfit)





### How to avoid Overfitting:

The commonly used methodologies are:

- Cross- Validation: A standard way to find out-of-sample prediction error is to use 5-fold cross validation.
- Early Stopping: Its rules provide us the guidance as to how many iterations can be run before learner begins to over-fit.
- Pruning: Pruning is extensively used while building related models. It simply removes the nodes which add little predictive power for the problem in hand.
- Regularization: It introduces a cost term for bringing in more features with the objective function. Hence it tries to push the coefficients for many variables to zero and hence reduce cost term.

### Good Fit in a Statistical Model:

Ideally, the case when the model makes the predictions with 0 error, is said to have a good fit on the data. This situation is achievable at a spot between overfitting and underfitting. In order to understand it we will have to look at the performance of our model with the passage of time, while it is learning from training dataset.

With the passage of time, our model will keep on learning and thus the error for the model on the training and testing data will keep on decreasing. If it will learn for too long, the model will become more prone to overfitting due to presence of noise and less useful details. Hence the performance of our model will decrease. In order to get a good fit, we will stop at a point just before where the error

starts increasing. At this point the model is said to have good skills on training dataset as well our unseen testing dataset.

## 8. Singular vector Decomposition

The singular value decomposition ( SVD ) is a powerful tool for dimensionality reduction. You can use the SVD to approximate a matrix and get out the important features. By taking only the top 80% or 90% of the energy in the matrix, you get the important features and throw out the noise. The SVD is employed in a number of applications today. One successful application is in recommendation engines.

The singular value decomposition (SVD)

- Pros: Simplifies data, removes noise, may improve algorithm results.
- Cons: Transformed data may be difficult to understand.
- Works with: Numeric values.

### Example: image compression with the SVD

## 9.Overview of Bias and Variance

The prediction error for any machine learning algorithm can be broken down into three parts:

- Bias Error
- Variance Error
- Irreducible Error

The irreducible error cannot be reduced regardless of what algorithm is used. It is the error introduced from the chosen framing of the problem and may be caused by factors like unknown variables that influence the mapping of the input variables to the output variable.

Bias Error

Bias are the simplifying assumptions made by a model to make the target function easier to learn.

Generally, parametric algorithms have a high bias making them fast to learn and easier to understand but generally less flexible. In turn, they have lower predictive performance on complex problems that fail to meet the simplifying assumptions of the algorithms bias.

- Low Bias:** Suggests less assumptions about the form of the target function.
- High-Bias:** Suggests more assumptions about the form of the target function.

Examples of low-bias machine learning algorithms include: Decision Trees, k-Nearest Neighbors and Support Vector Machines.

Examples of high-bias machine learning algorithms include: Linear Regression, Linear Discriminant Analysis and Logistic Regression.

For example, if the relationship between Y(your output) and X(your input) is actually a degree-4 equation, and you use simple linear regression for modeling, your bias will be pretty high.

Variance Error

Variance is the amount that the estimate of the target function will change if different training data was used. The target function is estimated from the training data by a

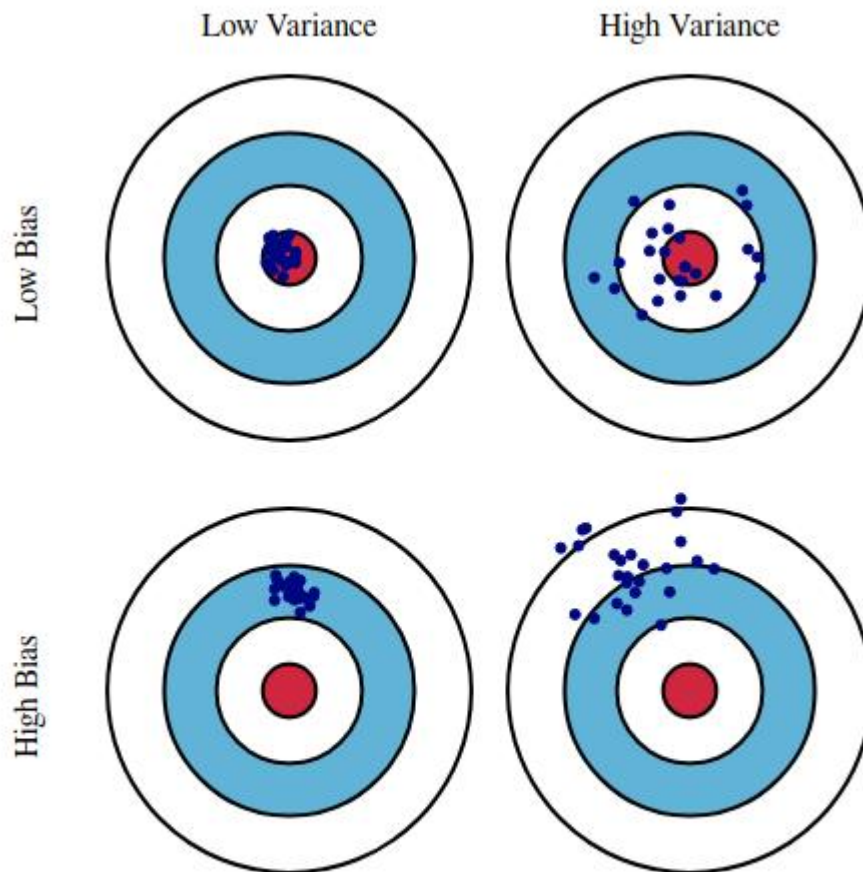


Fig. 1 Graphical illustration of bias and variance.

machine learning algorithm, so we should expect the algorithm to have some variance. Ideally, it should not change too much from one training dataset to the next, meaning that the algorithm is good at picking out the hidden underlying mapping between the inputs and the output variables. Machine learning algorithms that have a high variance are strongly influenced by the specifics of the training data. This means that the specifics of the training have influences the number and types of parameters used to characterize the mapping function.

•**Low Variance:** Suggests small changes to the estimate of the target function with changes to the training dataset.

•**High Variance:** Suggests large changes to the estimate of the target function with changes to the training dataset.

Generally, nonparametric machine learning algorithms that have a lot of flexibility have a high variance. For example, decision trees have a high variance, that is even higher if the trees are not pruned before use.

Examples of low-variance machine learning algorithms include: Linear Regression, Linear Discriminant Analysis and Logistic Regression.

Examples of high-variance machine learning algorithms include: Decision Trees, k-Nearest Neighbors and Support Vector Machines.

### Bias-Variance Trade-Off

The goal of any supervised machine learning algorithm is to achieve low bias and low variance. In turn the algorithm should achieve good prediction performance.

You can see a general trend in the examples above:

- Parametric or linear machine learning algorithms often have a high bias but a low variance.
- Non-parametric or non-linear machine learning algorithms often have a low bias but a high variance.

The parameterization of machine learning algorithms is often a battle to balance out bias and variance.

Below are two examples of configuring the bias-variance trade-off for specific algorithms:

- The k-nearest neighbors algorithm has low bias and high variance, but the trade-off can be changed by increasing the value of k which increases the number of neighbors that contribute to the prediction and in turn increases the bias of the model.
- The support vector machine algorithm has low bias and high variance, but the trade-off can be changed by increasing the C parameter that influences the number of violations of the margin allowed in the training data which increases the bias but decreases the variance.

There is no escaping the relationship between bias and variance in machine learning.

- Increasing the bias will decrease the variance.
- Increasing the variance will decrease the bias.

There is a trade-off at play between these two concerns and the algorithms you choose and the way you choose to configure them are finding different balances in this trade-off for your problem

In reality, we cannot calculate the real bias and variance error terms because we do not know the actual underlying target function. Nevertheless, as a framework, bias and variance provide the tools to understand the behavior of machine learning algorithms in the pursuit of predictive performance.

### Special cases

Some learning algorithms make particular assumptions about the structure of the data or the desired results. If you can find one that fits your needs, it can give you more useful results, more accurate predictions, or faster training times.

Algorithm	Accuracy	Training time	Linearity	Parameters	Notes
<b>Two-class classification</b>					
logistic regression		●	●	5	
decision forest	●	○		6	
decision jungle	●	○		6	Low memory footprint
boosted decision tree	●	○		6	Large memory footprint
neural network	●			9	Additional customization is possible
averaged perceptron	○	○	●	4	
support vector machine		○	●	5	Good for large feature sets
locally deep support vector machine	○			8	Good for large feature sets
Bayes' point machine		○	●	3	
<b>Multi-class classification</b>					
logistic regression		●	●	5	
decision forest	●	○		6	
decision jungle	●	○		6	Low memory footprint
neural network	●			9	Additional customization is possible
one-v-all	-	-	-	-	See properties of the two-class method selected
<b>Regression</b>					

Algorithm	Accuracy	Training time	Linearity	Parameters	Notes
<a href="#">linear</a>		●	●	4	
<a href="#">Bayesian linear</a>		○	●	2	
<a href="#">decision forest</a>	●	○		6	
<a href="#">boosted decision tree</a>	●	○		5	Large memory footprint
<a href="#">fast forest quantile</a>	●	○		9	Distributions rather than point predictions
<a href="#">neural network</a>	●			9	<a href="#">Additional customization is possible</a>
<a href="#">Poisson</a>			●	5	Technically log-linear. For predicting counts
<a href="#">ordinal</a>				0	For predicting rank-ordering
<b>Anomaly detection</b>					
<a href="#">support vector machine</a>	○	○		2	Especially good for large feature sets
<a href="#">PCA-based anomaly detection</a>		○	●	3	
<a href="#">K-means</a>		○	●	4	A clustering algorithm

