1. **What metrics do you use to evaluate a model?**

we have different evaluation metrics for each type of supervised learning. Here is the list of the evaluation metrics we will be covering in this course:

**Evaluation metrics for Classification:**

1. Confusion Matrix
2. Accuracy
3. Alternatives to Accuracy
4. Precision and Recall
5. F-Score
6. AUC-ROC
7. Log Loss
8. Gini Coefficient

**Evaluation metrics for Regression**

1. MAE(Mean Absolute Error)
2. MSE(Mean Square Error)
3. RMSE(Root Mean Square Error)
4. RMSLE(Root Mean Square Log Error)
5. R2 and Adjusted R2
6. **What is the difference between supervised and unsupervised learning?**

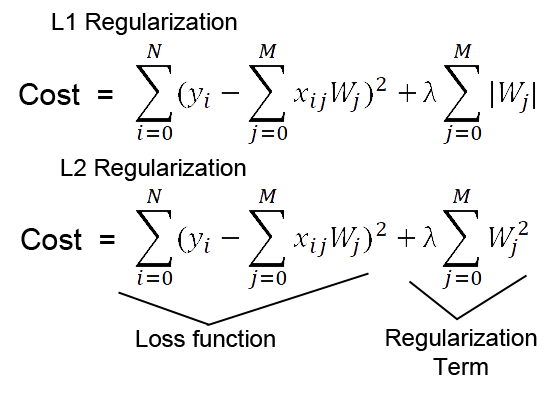
**Supervised Machine Learning:** When you have past data with outcomes (labels in machine learning terminology) and you want to predict the outcomes for the future – you would use Supervised Machine Learning algorithms. Supervised Machine Learning problems can again be divided into 2 kinds of problems:

* 1. **Classification Problems:** When you want to classify outcomes into different classes. For example – whether the floor needs cleaning/mopping is a classification problem. The outcome can fall into one of the classes – Yes or No. Similarly, whether a customer would default on their loan or not is a classification problem which is of high interest to any Bank
  2. **Regression Problem:** When you are interested in answering how much – these problems would fall under the Regression umbrella. For example – how much cleaning needs to be done is a Regression problem. Or what is the expected amount of default from a customer is a Regression problem

**Unsupervised Machine Learning:** There are times when you don’t want to exactly predict an Outcome. You just want to perform a segmentation or clustering. For example – a bank would want to have a segmentation of its customers to understand their behaviour. This is an Unsupervised Machine Learning problem as we are not predicting any outcomes here.

1. **Describe the difference between L1 and L2 regulations, specifically with regards to the difference in their impact on the model training process.**

The main intuitive difference between the L1 and L2 regularization is that **L1 regularization tries to estimate the median of the data** while the L2 regularization tries to estimate the mean of the data to avoid overfitting.



As we can see from the formula of L1 and L2 regularization, L1 regularization adds the penalty term in cost function by adding the absolute value of weight(Wj) parameters, while L2 regularization adds the squared value of weights(Wj) in the cost function.

While taking derivative of the cost function, in **L1 regularization** it will estimate around the median of the data. Let me explain it in this way — Suppose you take an arbitrary value from the data (assume data is spread along a horizontal line). If you then move in one direction to some distance **d,**suppose in the backward direction, then while calculating loss, the values to the one side (let say left side) of the chosen point will have a lesser loss value while on another side will contribute more in the loss function calculation.

Therefore, to minimize the loss function, we should try to estimate a value that should lie at the mid of the data distribution. That value will also be the **median** of the data distribution mathematically.

While in **L2 regularization,**while calculating the loss function in the gradient calculation step, the loss function tries to minimize the loss by subtracting it from the average of the data distribution.

That’s the main intuitive difference between the L1 (Lasso) and L2 (Ridge) regularization technique.

1. **How does XGBoost handle the bias-variance trade-off?**

Bagging algorithms control for high variance in a model. However, boosting algorithms are considered more effective as they deal with both bias as well as variance (the bias-variance trade-off).

XGBoost is an implementation of Gradient Boosting Machines (GBM) and is used for supervised learning.

First lets talk about pseudo residuals:

* In gradient boosting we have k number of base models with high bias which means they are shallow, can be called decision stumps if we talk about tree based models.
* Now we combine these base learners using pseudo residuals and they way we do pseudo residuals is by using the negative of the derivative of the loss function with respect to the function at stage k-1

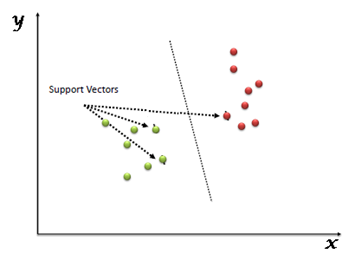
The main idea of gradient boosting is to have any loss function which is differentiable.



This is the formula we use to update the model, where m is the number of base models which is a hyperparameter. As number of base models increase we have a higher change of overfitting, but our bias is reducing. Optimal number of base models can be found using cross validation.

1. **Explain the SVM machine learning algorithm.**

Support Vector Machine” (SVM) is a supervised [machine learning algorithm](https://courses.analyticsvidhya.com/courses/introduction-to-data-science-2?utm_source=blog&utm_medium=understandingsupportvectormachinearticle) which can be used for both classification or regression challenges. However, it is mostly used in classification problems. In the SVM algorithm, we plot each data item as a point in an n-dimensional space (where n is the number of features you have). The value of each feature here is the value of a particular coordinate. Then, we perform classification by finding the plane that differentiates the two classes very well

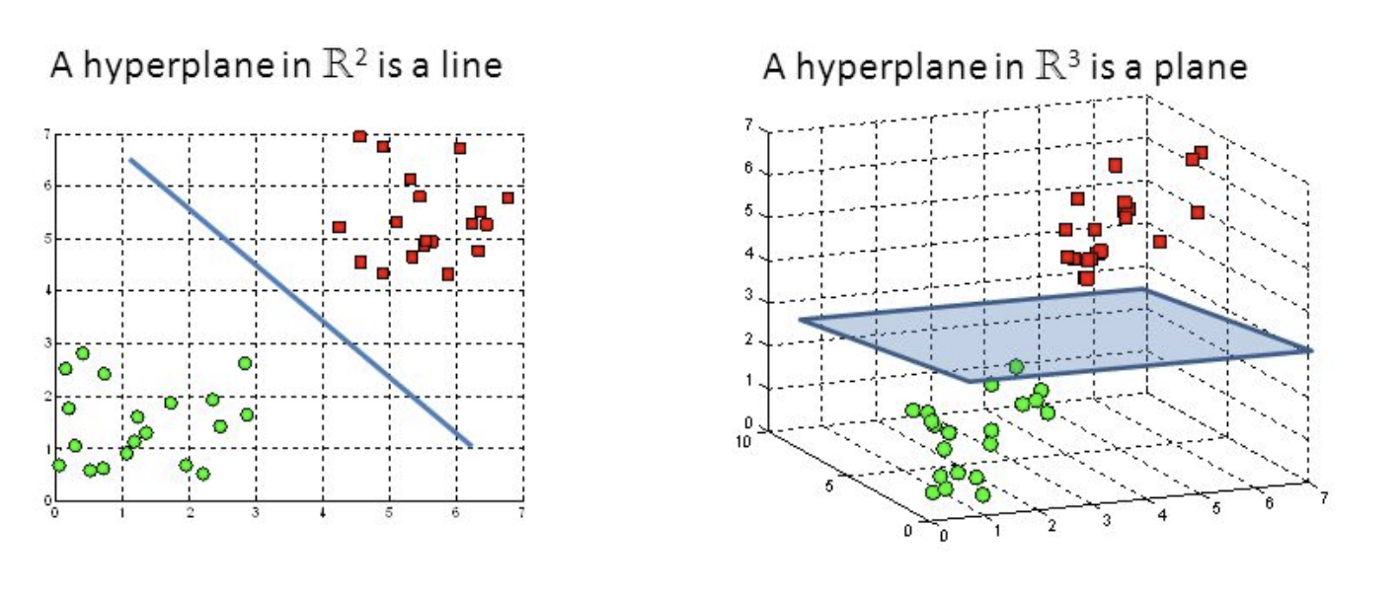


Think of machine learning algorithms as an armoury packed with axes, sword, blades, bow, dagger, etc. You have various tools, but you ought to learn to use them at the right time. As an analogy, think of ‘Regression’ as a sword capable of slicing and dicing data efficiently, but incapable of dealing with highly complex data. On the contrary, the ‘Support Vector Machine’ is like a sharp knife – it works on smaller datasets, but on the complex ones, it can be much stronger and powerful in building machine learning models.

The main advantages of using the SVM algorithm are:

* It works really well with a clear margin of separation
* It is effective in high dimensional spaces.
* It is effective in cases where the number of dimensions is greater than the number of samples.
* It uses a subset of training points in the decision function (called support vectors), so it is also memory efficient.

## Hyperplanes and Support Vectors



Hyperplanes are decision boundaries that help classify the data points. Data points falling on either side of the hyperplane can be attributed to different classes. Also, the dimension of the hyperplane depends upon the number of features. If the number of input features is 2, then the hyperplane is just a line. If the number of input features is 3, then the hyperplane becomes a two-dimensional plane. It becomes difficult to imagine when the number of features exceeds 3.



Support vectors are data points that are closer to the hyperplane and influence the position and orientation of the hyperplane. Using these support vectors, we maximize the margin of the classifier. Deleting the support vectors will change the position of the hyperplane. These are the points that help us build our SVM.

1. **Write a function to detect if a binary tree is a mirror image on both the left and right sub-trees.**

class Node:

# Utility function to create new node

def \_\_init\_\_(self, key):

self.key = key

self.left = None

self.right = None

# Returns True if trees

#with roots as root1 and root 2 are mirror

def isMirror(root1, root2):

# If both trees are empty, then they are mirror images

if root1 is None and root2 is None:

return True

""" For two trees to be mirror images,

the following three conditions must be true

1 - Their root node's key must be same

2 - left subtree of left tree and right subtree

of the right tree have to be mirror images

3 - right subtree of left tree and left subtree

of right tree have to be mirror images

"""

if (root1 is not None and root2 is not None):

if root1.key == root2.key:

return (isMirror(root1.left, root2.right)and

isMirror(root1.right, root2.left))

# If none of the above conditions is true then root1

# and root2 are not mirror images

return False

def isSymmetric(root):

# Check if tree is mirror of itself

return isMirror(root, root)

# Driver Code

# Let's construct the tree show in the above figure

root = Node(1)

root.left = Node(2)

root.right = Node(2)

root.left.left = Node(3)

root.left.right = Node(4)

root.right.left = Node(4)

root.right.right = Node(3)

print "Symmetric" if isSymmetric(root) == True else "Not symmetric"