Jntuk R20 ML UNIT-III - This lecture notes are helpful to all btech students who are in r20 regulation.This

machine learning (Jawaharlal Nehru Technological University, Kakinada)

Unit III

Ensemble Learning and Random Forests: Introduction, Voting Classifiers, Bagging and Pasting, Random Forests, Boosting, Stacking.

Support Vector Machine: Linear SVM Classification, Nonlinear SVM Classification, SVM Regression, Naïve Bayes Classifiers.

....

Ensemble learning is a machine learning paradigm where multiple models (often called "weak learners") are trained to solve the same problem and combined to get better results. The main hypothesis is that when weak models are correctly combined we can obtain more accurate and/or robust models.

Single weak learner

In machine learning, no matter if we are facing a classification or a regression problem, the choice of the model is extremely important to have any chance to obtain good results. This choice can depend on many variables of the problem: quantity of data, dimensionality of the space, distribution hypothesis…

A low bias and a low variance, although they most often vary in opposite directions, are the two most fundamental features expected for a model. Indeed, to be able to "solve" a problem, we want our model to have enough degrees of freedom to resolve the underlying complexity of the data we are working with, but we also want it to have not too much degrees of freedom to avoid high variance and be more robust. This is the well known **bias-variance tradeoff**.

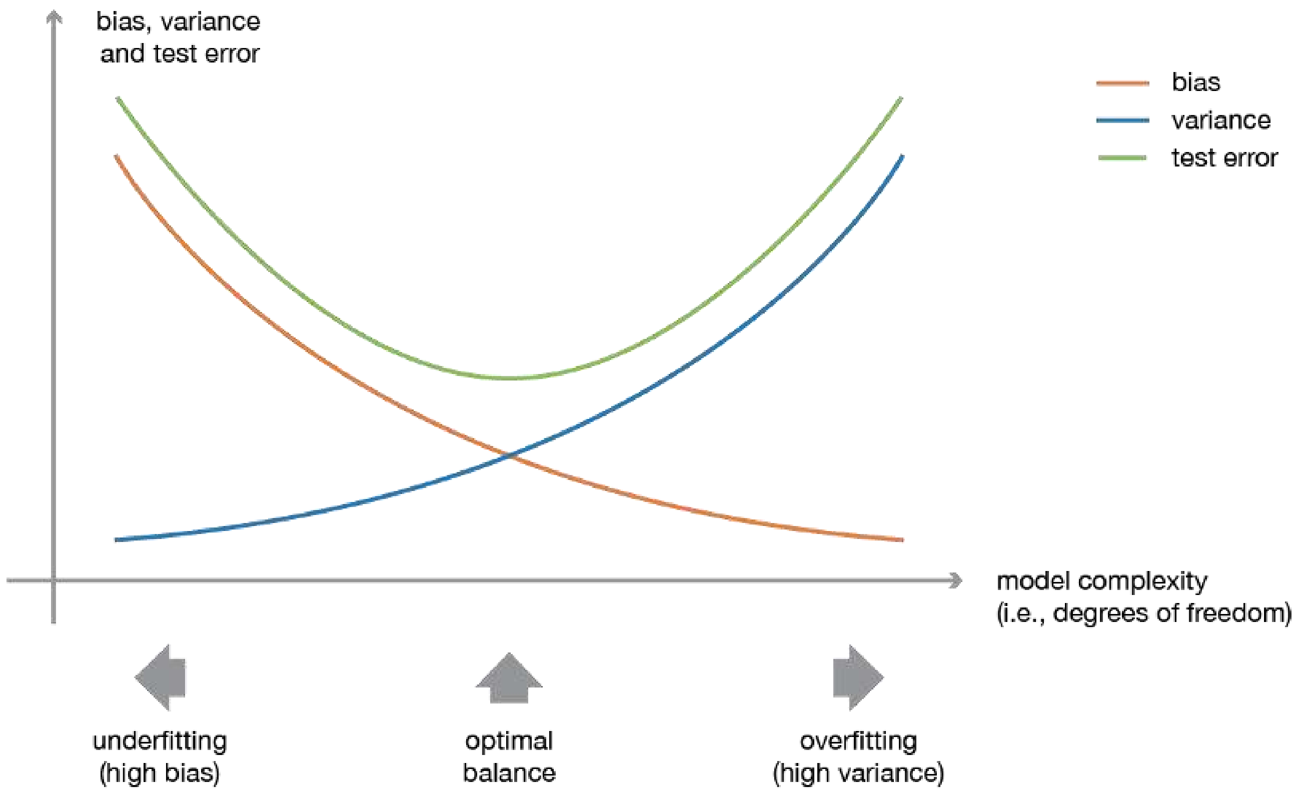
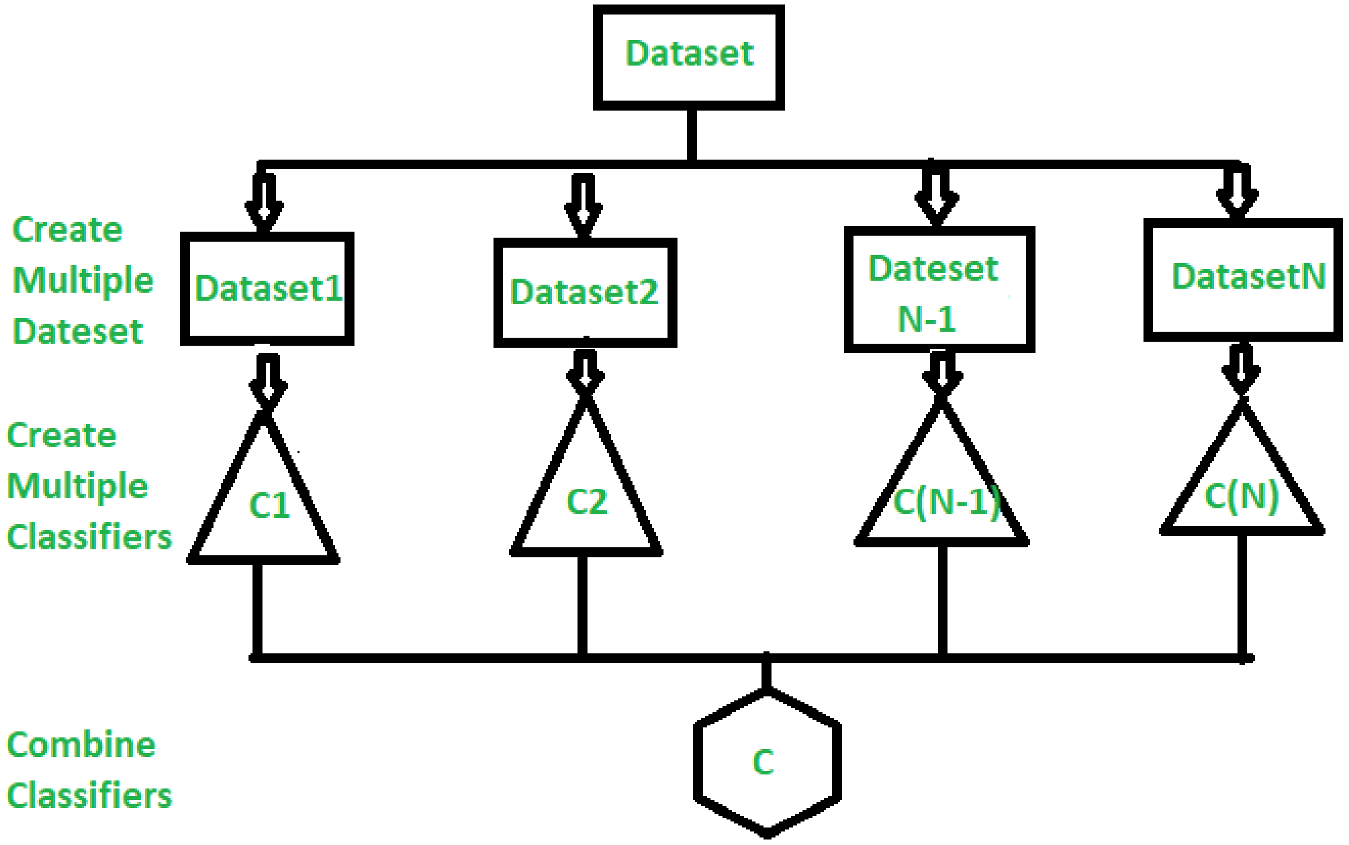


Fig: Illustration of the bias-variance tradeoff.

In ensemble learning theory, we call **weak learners** (or **base models**) models that can be used as building blocks for designing more complex models by combining several of them. Most of the time, these basics models perform not so well by themselves either because they have a high bias (low degree of freedom models, for example) or because they have too much variance to be robust (high degree of freedom models, for example). Then, the idea of ensemble methods is to try reducing bias and/or variance of such weak learners by combining several of them together in order to create a **strong learner** (or **ensemble model**) that achieves better performances.

Ensemble learning helps improve machine learning results by combining several models. This approach allows the production of better predictive performance compared to a single model. Basic idea is to learn a set of classifiers (experts) and to allow them to vote.

Advantage : Improvement in predictive accuracy.
Disadvantage : It is difficult to understand an ensemble of classifiers.



Why do ensembles work?

Dietterich(2002) showed that ensembles overcome three problems –

- **Statistical Problem –**
The Statistical Problem arises when the hypothesis space is too large for the amount of available data. Hence, there are many hypotheses with the same accuracy on the data and the learning algorithm chooses only one of them! There is a risk that the accuracy of the chosen hypothesis is low on unseen data!
- **Computational Problem –**
The Computational Problem arises when the learning algorithm cannot guarantees finding the best hypothesis.
- **Representational Problem –**
The Representational Problem arises when the hypothesis space does not contain any good approximation of the target class(es).

Main Challenge for Developing Ensemble Models?

The main challenge is not to obtain highly accurate base models, but rather to obtain base models which make different kinds of errors. For example, if ensembles are used for classification, high accuracies can be accomplished if different base models misclassify different training examples, even if the base classifier accuracy is low.

Methods for Independently Constructing Ensembles –

- Majority Vote
- Bagging and Random Forest
- Randomness Injection
- Feature-Selection Ensembles
- Error-Correcting Output Coding

Methods for Coordinated Construction of Ensembles –

- Boosting
- Stacking

Reliable Classification: Meta-Classifier Approach, Co-Training and Self-Training

This brings us to the question of how to combine these models. We can mention three major kinds of meta-algorithms that aims at combining weak learners:

- **bagging**, that often considers homogeneous weak learners, learns them independently from each other in parallel and combines them following some kind of deterministic averaging process
- **boosting**, that often considers homogeneous weak learners, learns them sequentially in a very adaptative way (a base model depends on the previous ones) and combines them following a deterministic strategy
- **stacking**, that often considers heterogeneous weak learners, learns them in parallel and combines them by training a meta-model to output a prediction based on the different weak models predictions

1. Voting Classifiers

A Voting Classifier is a machine learning model that trains on an ensemble of numerous models and predicts an output (class) based on their highest probability of chosen class as the output.
It simply aggregates the findings of each classifier passed into Voting Classifier and predicts the output class based on the highest majority of voting. The idea is instead of creating separate dedicated models and finding the accuracy for each them, we create a single model which trains by these models and predicts output based on their combined majority of voting for each output class.

Voting Classifier supports two types of votings.

1. **Hard Voting:** In hard voting, the predicted output class is a class with the highest majority of votes i.e the class which had the highest probability of being predicted by each of the classifiers. Suppose three classifiers predicted the *output class(A, A, B)*, so here the majority predicted *A* as output. Hence *A* will be the final prediction.
2. **Soft Voting:** In soft voting, the output class is the prediction based on the average of probability given to that class. Suppose given some input to three models, the prediction probability for class *A = (0.30, 0.47, 0.53)* and *B = (0.20, 0.32, 0.40)*. So the average for class *A is 0.4333* and *B is 0.3067*, the winner is clearly class *A* because it had the highest probability averaged by each classifier.

Note: Make sure to include a variety of models to feed a Voting Classifier to be sure that the error made by one might be resolved by the other.

```
# importing libraries
from sklearn.ensemble import VotingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.datasets import load_iris
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
# loading iris dataset
iris = load_iris()
X = iris.data[:, :4]
Y = iris.target
# train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,Y,test_size = 0.20, random_state = 42)
# group / ensemble of models
estimator = [ ]
estimator.append(('LR', LogisticRegression(solver ='lbfgs', multi_class ='multinomial',max_iter = 200)))
estimator.append(('SVC', SVC(gamma ='auto', probability = True)))
estimator.append(('DTC', DecisionTreeClassifier()))
# Voting Classifier with hard voting
vot_hard = VotingClassifier(estimators = estimator, voting ='hard')
vot_hard.fit(X_train, y_train)
y_pred = vot_hard.predict(X_test)
# using accuracy_score metric to predict accuracy
score = accuracy_score(y_test, y_pred)
print("Hard Voting Score % d" % score)
# Voting Classifier with soft voting
vot_soft = VotingClassifier(estimators = estimator, voting ='soft')
vot_soft.fit(X_train, y_train)
y_pred = vot_soft.predict(X_test)
# using accuracy_score
score = accuracy_score(y_test, y_pred)
print("Soft Voting Score % d" % score)
```

2. Bagging and Pasting

In machine learning, sometimes multiple predictors grouped together have a better predictive performance than anyone of the group alone. These techniques are very popular in competitions and in production. They are called Ensemble Learning.

There are several ways to group models. They differ in the training algorithm and data used in each one of them and also how they are grouped. We will be talking about two methods called **Bagging** and **Pasting** here.

But before we begin talking about Bagging and Pasting, we have to know what is **Bootstrapping**.

Bootstrapping

In statistics, bootstrapping refers to a resample method that consists of repeatedly drawn, with replacement, samples from data to form other smaller datasets, called bootstrapping samples. It's as if the bootstrapping method is making a bunch of simulations to our original dataset so in some cases we can generalize the mean and the standard deviation.

For example, let's say we have a set of observations: [2, 4, 32, 8, 16]. If we want each bootstrap sample containing *n* observations, the following are valid samples:

- *n*=3: [32, 4, 4], [8, 16, 2], [2, 2, 2]…
- *n*=4: [2, 32, 4, 16], [2, 4, 2, 8], [8, 32, 4, 2]…

Since we drawn data with replacement, the observations can appear more than one time in a single sample.

Bagging means bootstrap+aggregating and it is a ensemble method in which we first bootstrap our data and for each bootstrap sample we train one model. After that, we aggregate them with equal weights. When it's not used replacement, the method is called pasting.

Out-of-Bag Scoring

If we are using bagging, there's a chance that a sample would never be selected, while anothers may be selected multiple time. The probability of not selecting a specific sample is (1–1/n), where n is the number of samples. Therefore, the probability of not picking n samples in n draws is (1–1/n)^n. When the value of n is big, we can approximate this probability to 1/e, which is approximately 0.3678. This means that when the dataset is big enough, 37% of its samples are never selected and we could use it to test our model. This is called Out-of-Bag scoring, or OOB Scoring.

3. Random Forests

Random forest is a Supervised Machine Learning Algorithm that is used widely in Classification and Regression problems.

It builds decision trees on different samples and takes their majority vote for classification and average in case of regression.

One of the most important features of the Random Forest Algorithm is that it can handle the data set containing continuous variables, as in the case of regression, and categorical variables, as in the case of classification. It performs better for classification and regression tasks.

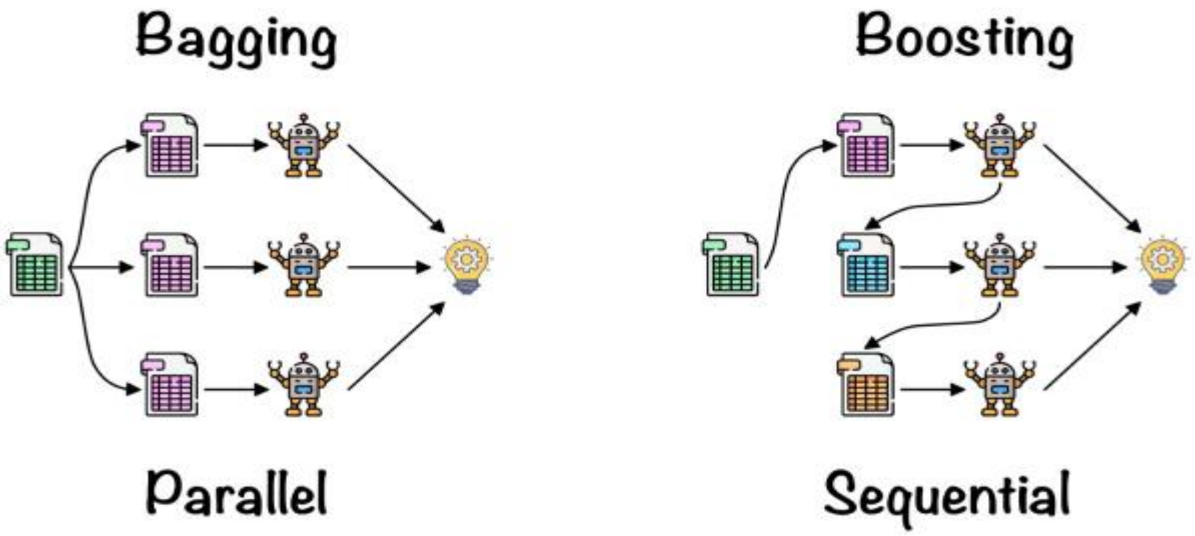
Working of Random Forest Algorithm

Before understanding the working of the random forest algorithm in machine learning, we must look into the ensemble learning technique. ***Ensemble*** simply means combining multiple models. Thus a collection of models is used to make predictions rather than an individual model.

Ensemble uses two types of methods:

1. **Bagging**– It creates a different training subset from sample training data with replacement & the final output is based on majority voting. For example, Random Forest.

2. **Boosting**– It combines weak learners into strong learners by creating sequential models such that the final model has the highest accuracy. For example, ADA BOOST, XG BOOST.



Steps Involved in Random Forest Algorithm

Step 1: In the Random forest model, a subset of data points and a subset of features is selected for constructing each decision tree. Simply put, n random records and m features are taken from the data set having k number of records.

Step 2: Individual decision trees are constructed for each sample.

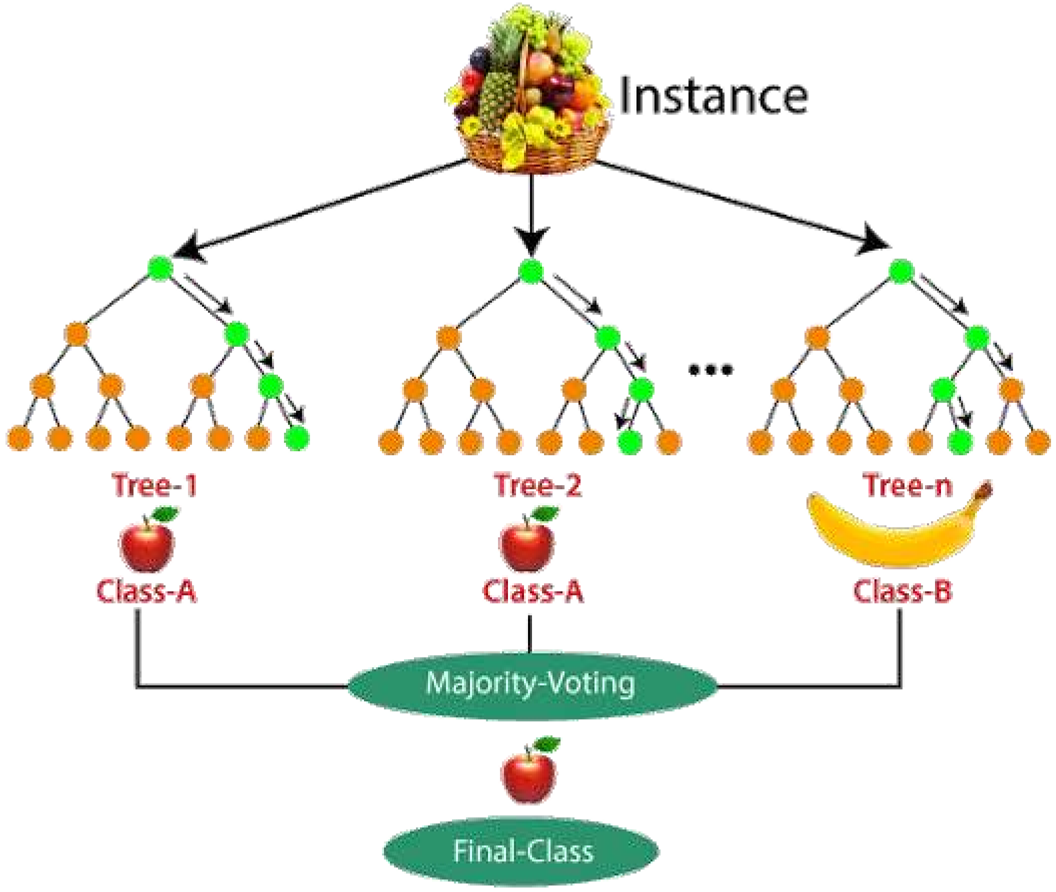
Step 3: Each decision tree will generate an output.

Step 4: Final output is considered based on ***Majority Voting or Averaging*** for Classification and regression, respectively.

Random Forest Algorithm Example:

For example: consider the fruit basket as the data as shown in the figure below. Now n number of samples are taken from the fruit basket, and an individual decision tree is constructed for each sample. Each decision tree will generate an output, as shown in the figure. The final output is considered based on majority voting.

In the below figure, you can see that the majority decision tree gives output as an apple when compared to a banana, so the final output is taken as an apple.



Important Features of Random Forest

- **Diversity:** Not all attributes/variables/features are considered while making an individual tree; each tree is different.
- **Immune to the curse of dimensionality:** Since each tree does not consider all the features, the feature space is reduced.
- **Parallelization:** Each tree is created independently out of different data and attributes. This means we can fully use the CPU to build random forests.
- **Train-Test split:** In a random forest, we don't have to segregate the data for train and test as there will always be 30% of the data which is not seen by the decision tree.
- **Stability:** Stability arises because the result is based on majority voting/ averaging.

For more watch the video: https://youtu.be/3LQI-w7-FuE

Advantages and Disadvantages of Random Forest Algorithm

Advantages

1. It can be used in classification and regression problems.

2. It solves the problem of overfitting as output is based on majority voting or averaging.

3. It performs well even if the data contains null/missing values.

4. Each decision tree created is independent of the other; thus, it shows the property of parallelization.

5. It is highly stable as the average answers given by a large number of trees are taken.

6. It maintains diversity as all the attributes are not considered while making each decision tree though it is not true in all cases.

7. It is immune to the curse of dimensionality. Since each tree does not consider all the attributes, feature space is reduced.

8. We don't have to segregate data into train and test as there will always be 30% of the data, which is not seen by the decision tree made out of bootstrap.

Disadvantages

1. Random forest is highly complex compared to decision trees, where decisions can be made by following the path of the tree.

2. Training time is more than other models due to its complexity. Whenever it has to make a prediction, each decision tree has to generate output for the given input data.

Difference Between Decision Tree and Random Forest

Random forest is a collection of decision trees; still, there are a lot of differences in their behavior.

Decision trees	Random Forest
1. Decision trees normally suffer from the problem of overfitting if it's allowed to grow without any control.	1. Random forests are created from subsets of data, and the final output is based on average or majority ranking; hence the problem of overfitting is taken care of.
2. A single decision tree is faster in computation.	2. It is comparatively slower.
3. When a data set with features is taken as input by a decision tree, it will formulate some rules to make predictions.	3. Random forest randomly selects observations, builds a decision tree, and takes the average result. It doesn't use any set of formulas.

Thus random forests are much more successful than decision trees only if the trees are diverse and acceptable.

4. Boosting

Boosting is an ensemble modeling technique that attempts to build a strong classifier from the number of weak classifiers. It is done by building a model by using weak models in series. Firstly, a model is built from the training data. Then the second model is built which tries to correct the errors present in the first model. This procedure is continued and models are added until either the complete training data set is predicted correctly or the maximum number of models are added.

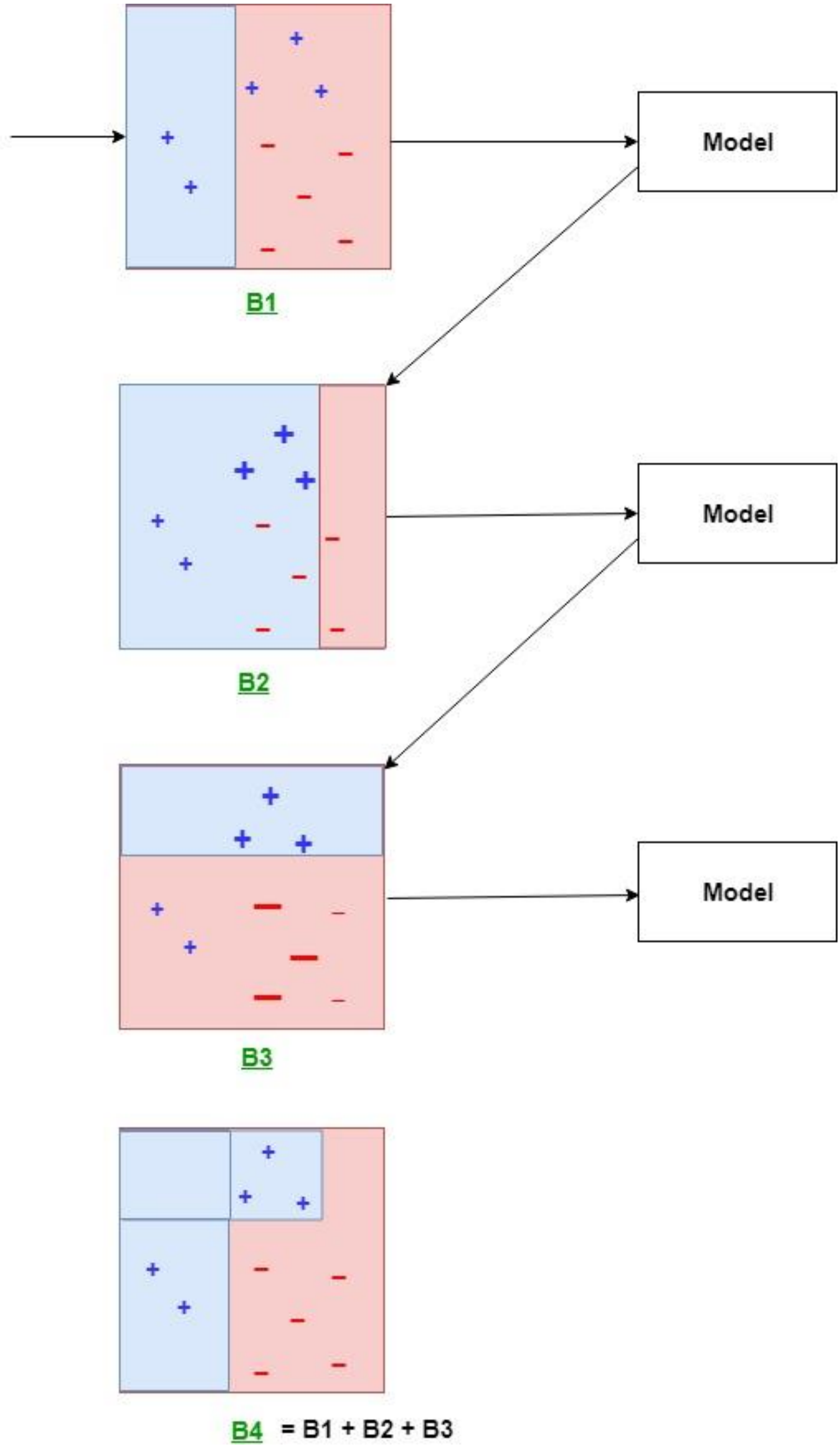
AdaBoost was the first really successful boosting algorithm developed for the purpose of binary classification. *AdaBoost* is short for *Adaptive Boosting* and is a very popular boosting technique that combines multiple "weak classifiers" into a single "strong classifier". It was formulated by Yoav Freund and Robert Schapire. They also won the 2003 Gödel Prize for their work.

Algorithm:

```
1. Initialise the dataset and assign equal weight to each of the data point.
2. Provide this as input to the model and identify the wrongly classified data points.
3. Increase the weight of the wrongly classified data points.
4. if (got required results)
     Goto step 5
   else
```

Goto step 2

5. End



Explanation:

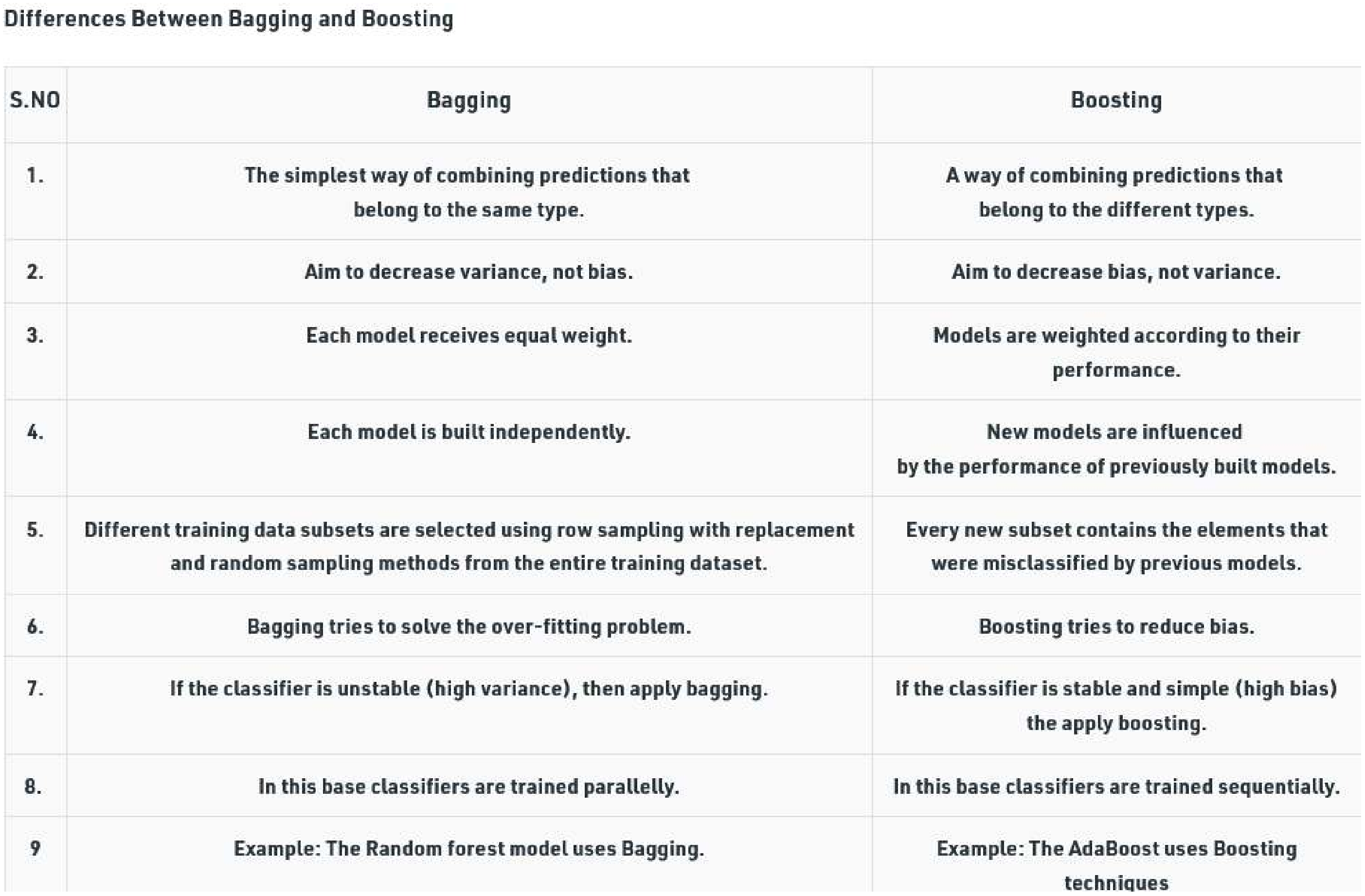
The above diagram explains the AdaBoost algorithm in a very simple way. Let's try to understand it in a stepwise process:

- **B1** consists of 10 data points which consist of two types namely plus(+) and minus(-) and 5 of which are plus(+) and the other 5 are minus(-) and each one has been assigned equal weight initially. The first model tries to classify the data points and generates a vertical separator line but it wrongly classifies 3 plus(+) as minus(-).
- **B2** consists of the 10 data points from the previous model in which the 3 wrongly classified plus(+) are weighted more so that the current model tries more to classify these pluses(+) correctly. This model generates a vertical separator line that correctly classifies the previously wrongly classified pluses(+) but in this attempt, it wrongly classifies three minuses(-).
- **B3** consists of the 10 data points from the previous model in which the 3 wrongly classified minus(-) are weighted more so that the current model tries more to classify these minuses(-) correctly. This model generates a horizontal separator line that correctly classifies the previously wrongly classified minuses(-).
- **B4** combines together B1, B2, and B3 in order to build a strong prediction model which is much better than any individual model used.

Similarities Between Bagging and Boosting

Bagging and Boosting, both being the commonly used methods, have a universal similarity of being classified as ensemble methods. Here we will explain the similarities between them.

1. Both are ensemble methods to get N learners from 1 learner.
2. Both generate several training data sets by random sampling.
3. Both make the final decision by averaging the N learners (or taking the majority of them i.e Majority Voting).
4. Both are good at reducing variance and provide higher stability.



Differences Between Bagging and Boosting

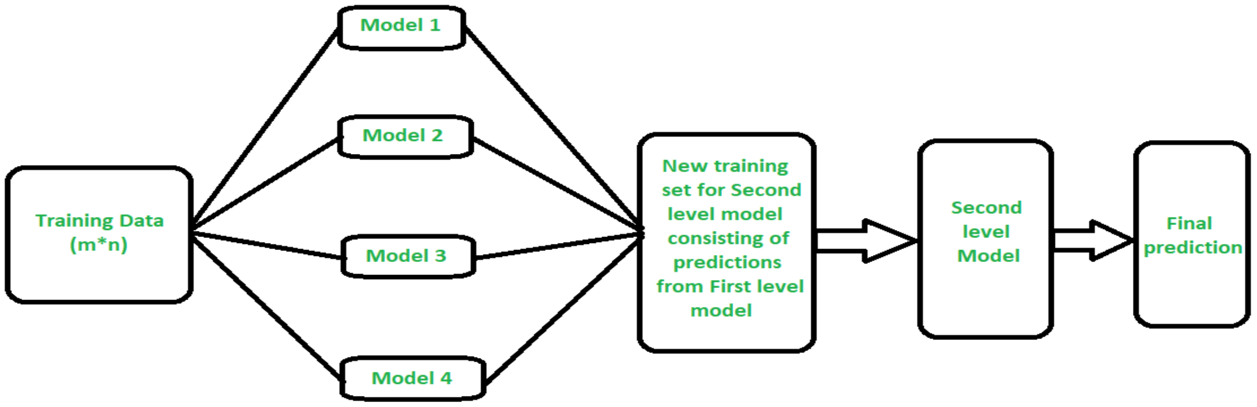
S.NO	Bagging	Boosting
1.	The simplest way of combining predictions that belong to the same type.	A way of combining predictions that belong to the different types.
2.	Aim to decrease variance, not bias.	Aim to decrease bias, not variance.
3.	Each model receives equal weight.	Models are weighted according to their performance.
4.	Each model is built independently.	New models are influenced by the performance of previously built models.
5.	Different training data subsets are selected using row sampling with replacement and random sampling methods from the entire training dataset.	Every new subset contains the elements that were misclassified by previous models.
6.	Bagging tries to solve the over-fitting problem.	Boosting tries to reduce bias.
7.	If the classifier is unstable (high variance), then apply bagging.	If the classifier is stable and simple (high bias) the apply boosting.
8.	In this base classifiers are trained parallelly.	In this base classifiers are trained sequentially.
9	Example: The Random forest model uses Bagging.	Example: The AdaBoost uses Boosting techniques

5. Stacking

Stacking is a way to ensemble multiple classifications or regression model. There are many ways to ensemble models, the widely known models are ***Bagging*** or ***Boosting***. Bagging allows multiple similar models with high variance are averaged to decrease variance. Boosting builds multiple incremental models to decrease the bias, while keeping variance small.

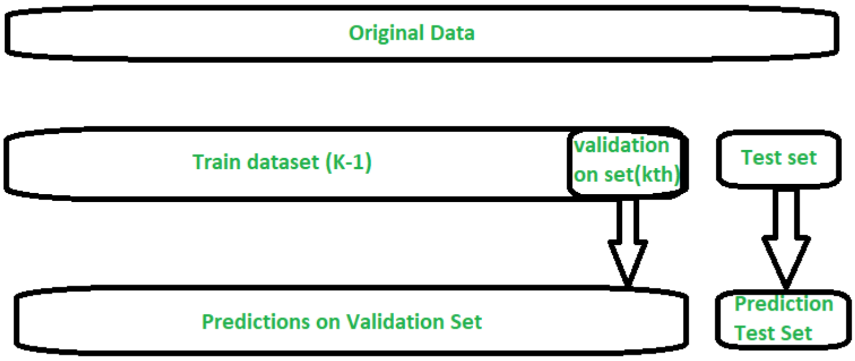
Stacking (sometimes called *Stacked Generalization*) is a different paradigm. The point of stacking is to explore a space of different models for the same problem. The idea is that you can attack a learning problem with different types of models which are capable to learn some part of the problem, but not the whole space of the problem. So, you can build multiple different learners and you use them to build an intermediate prediction, one prediction for each learned model. Then you add a new model which learns from the intermediate predictions the same target.

This final model is said to be stacked on the top of the others, hence the name. Thus, you might improve your overall performance, and often you end up with a model which is better than any individual intermediate model. Notice however, that it does not give you any guarantee, as is often the case with any machine learning technique.


Training Data (m*n)
Model 1
Model 2
Model 3
Model 4
New training set for Second level model consisting of predictions from First level model
Second level Model
Final prediction

How stacking works?

1. We split the training data into K-folds just like K-fold cross-validation.
2. A base model is fitted on the K-1 parts and predictions are made for Kth part.
3. We do for each part of the training data.
4. The base model is then fitted on the whole train data set to calculate its performance on the test set.
5. We repeat the last 3 steps for other base models.
6. Predictions from the train set are used as features for the second level model.
7. Second level model is used to make a prediction on the test set.

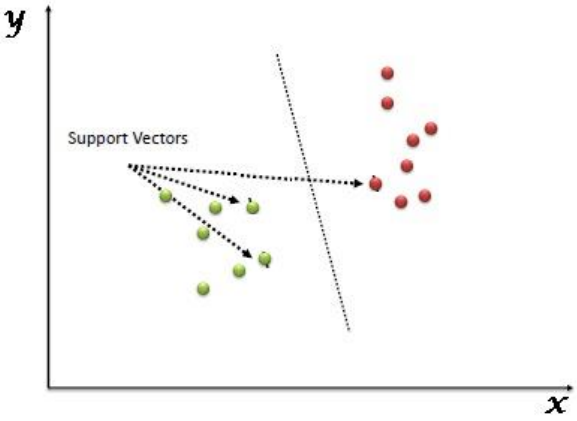


CHAPTER-II

Support Vector Machine: Linear SVM Classification, Nonlinear SVM Classification, SVM Regression, Naïve Bayes Classifiers.

……

"Support Vector Machine" (SVM) is a supervised learning machine learning algorithm that can be used for both classification or regression challenges. However, it is mostly used in classification problems, such as text classification. In the SVM algorithm, we plot each data item as a point in n-dimensional space (where n is the number of features you have), with the value of each feature being the value of a particular coordinate. Then, we perform classification by finding the optimal hyper-plane that differentiates the two classes very well (look at the below snapshot).

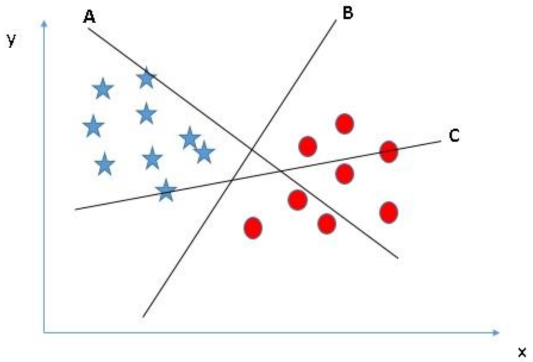


Support Vectors are simply the coordinates of individual observation, and a hyper-plane is a form of SVM visualization. The SVM classifier is a frontier that best segregates the two classes (hyper-plane/line).

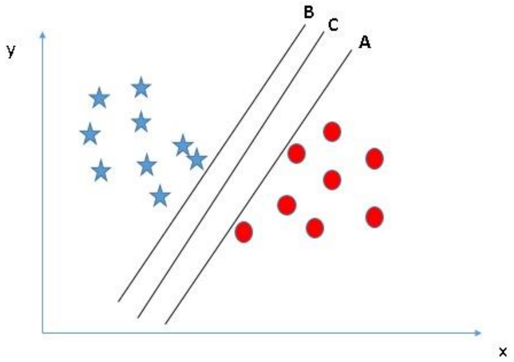
1. Linear SVM Classification

Let's understand:

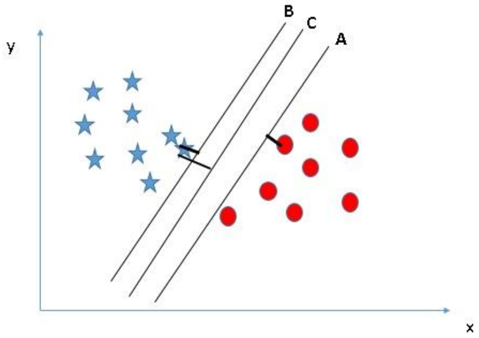
- **Identify the right hyper-plane (Scenario-1):** Here, we have three hyper-planes (A, B, and C). Now, identify the right hyper-plane to classify stars and circles.



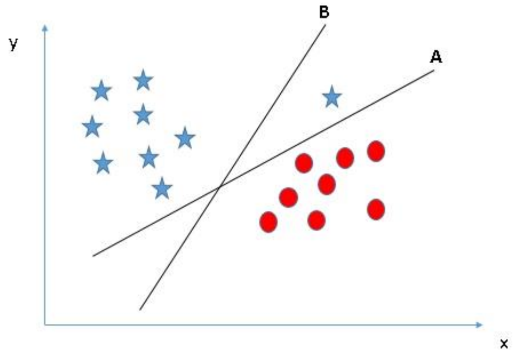
- You need to remember a thumb rule to identify the right hyper-plane: "Select the hyper-plane which segregates the two classes better." In this scenario, hyper-plane "B" has excellently performed this job.
- **Identify the right hyper-plane (Scenario-2):** Here, we have three hyper-planes (A, B, and C), and all segregate the classes well. Now, How can we identify the right hyper-plane?



- Here, maximizing the distances between the nearest data point (either class) and the hyper-plane will help us to decide the right hyper-plane. This distance is called a **Margin**. Let's look at the below snapshot:

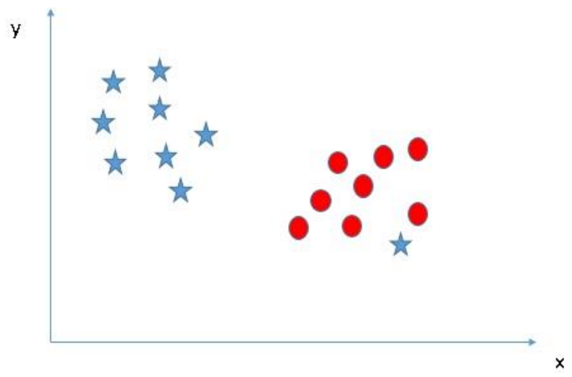


- Above, you can see that the margin for hyper-plane C is high as compared to both A and B. Hence, we name the right hyper-plane as C. Another lightning reason for selecting the hyper-plane with a higher margin is robustness. If we select a hyper-plane having a low margin, then there is a high chance of misclassification.
- **Identify the right hyper-plane (Scenario-3):** Hint: Use the rules as discussed in the previous section to identify the right hyper-plane.

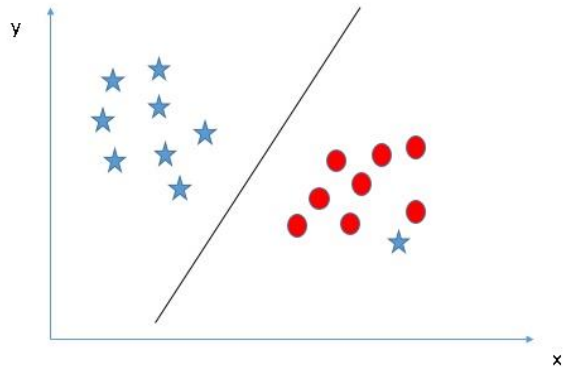


Some of you may have selected hyper-plane B as it has a higher margin compared to A. But, here is the catch, SVM selects the hyper-plane which classifies the classes accurately prior to maximizing the margin. Here, hyper-plane B has a classification error, and A has classified all correctly. Therefore, the right hyper-plane is A.

- **Can we classify two classes (Scenario-4)?:** Below, I am unable to segregate the two classes using a straight line, as one of the stars lies in the territory of the other (circle) class as an outlier.

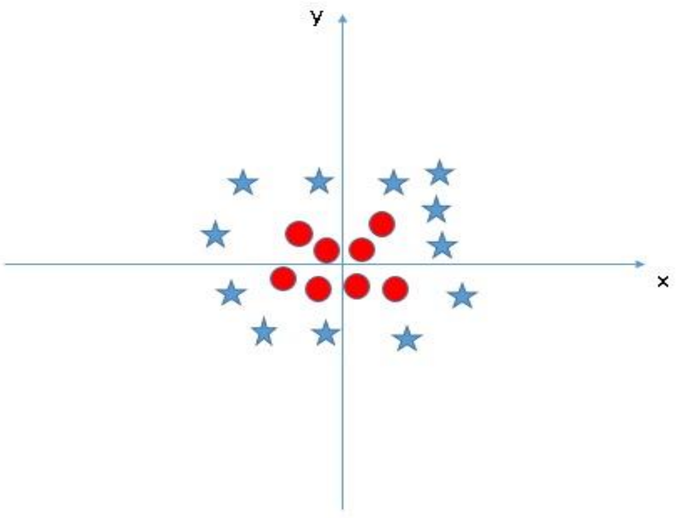


- As I have already mentioned, one star at the other end is like an outlier for the star class. The SVM algorithm has a feature to ignore outliers and find the hyper-plane that has the maximum margin. Hence, we can say SVM classification is robust to outliers.

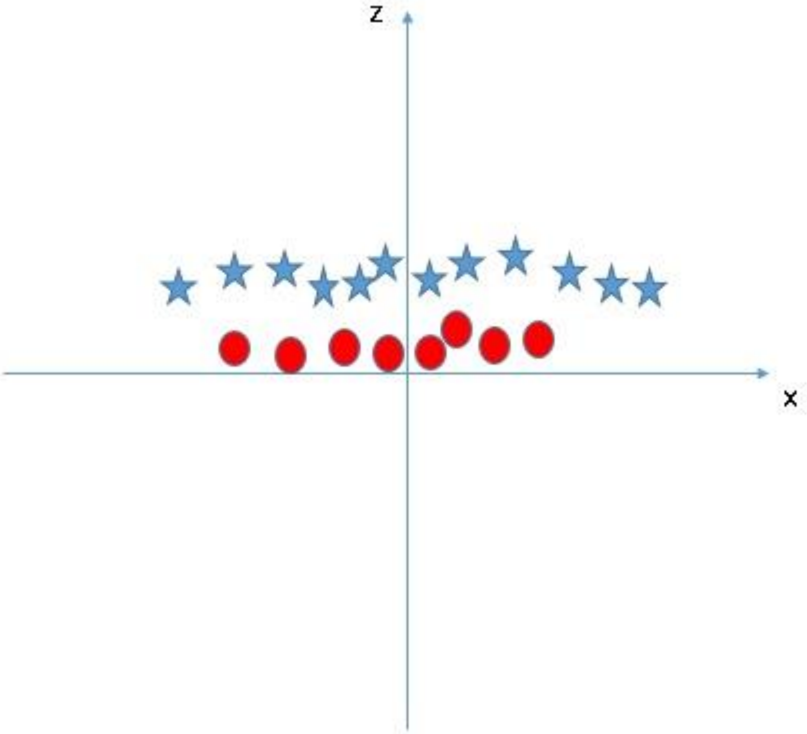


2. Non-Linear SVM Classification

- **Find the hyper-plane to segregate to classes (Scenario-5):** In the scenario below, we can't have a linear hyper-plane between the two classes, so how does SVM classify these two classes? Till now, we have only looked at the linear hyper-plane.



- SVM can solve this problem. Easily! It solves this problem by introducing additional features. Here, we will add a new feature, z=x^2+y^2. Now, let's plot the data points on axis x and z:



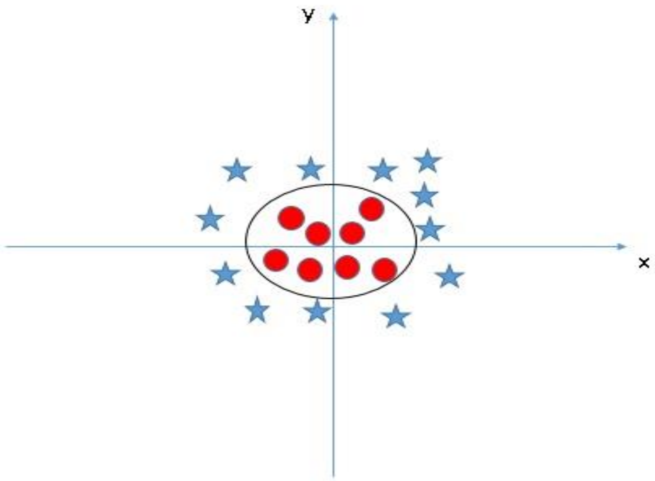
In the above plot, points to consider are:

- All values for z would always be positive because z is the squared sum of both x and y
- In the original plot, red circles appear close to the origin of the x and y axes, leading to a lower value of z. The star is relatively away from the original results due to the higher value of z.

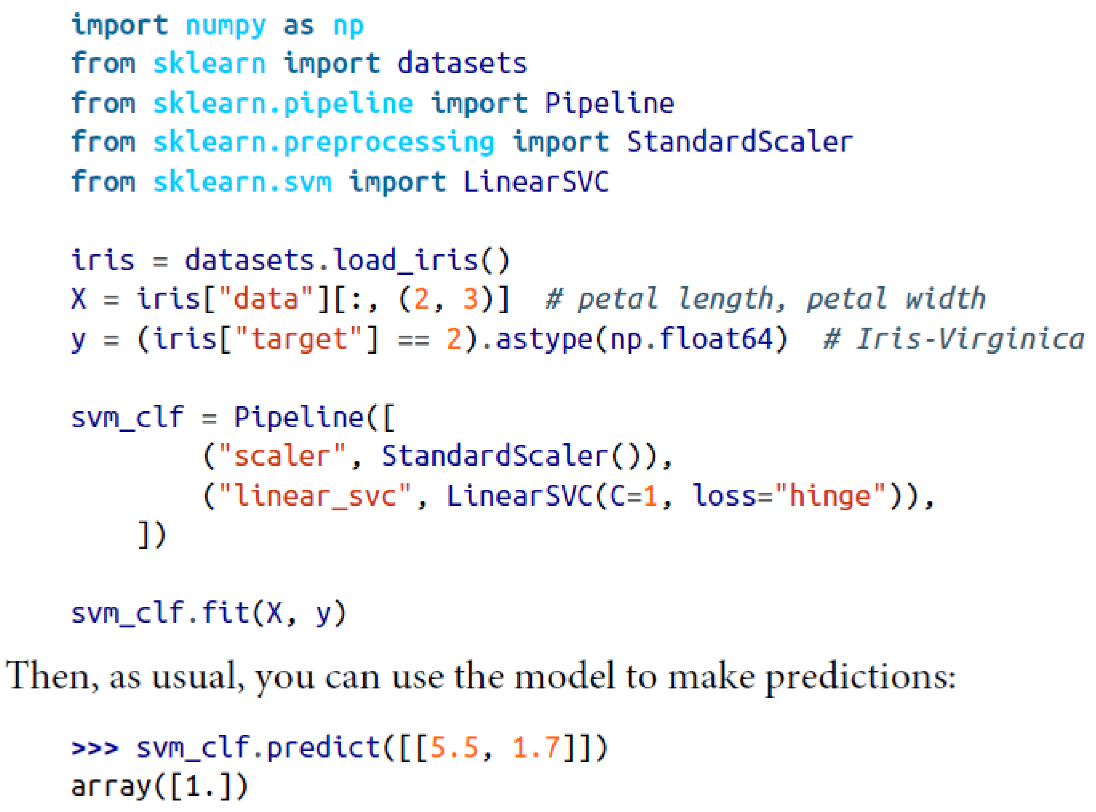
In the SVM classifier, having a linear hyper-plane between these two classes is easy. But, another burning question that arises is if we should we need to add this feature manually to have a hyper-plane. No, the SVM algorithm has a technique called the **kernel trick**.

The SVM kernel is a function that takes low dimensional input space and transforms it to a higher dimensional space, i.e., it converts not separable problem to a separable problem. It is mostly useful in non-linear data separation problems. Simply put, it does some extremely complex data transformations, then finds out the process to separate the data based on the labels or outputs you've defined.

When we look at the hyper-plane in the original input space, it looks like a circle:



Linear classification implementation



```
import numpy as np
from sklearn import datasets
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.svm import LinearSVC

iris = datasets.load_iris()
X = iris["data"][:, (2, 3)]  # petal length, petal width
y = (iris["target"] == 2).astype(np.float64)  # Iris-Virginica

svm_clf = Pipeline([
        ("scaler", StandardScaler()),
        ("linear_svc", LinearSVC(C=1, loss="hinge")),
    ])

svm_clf.fit(X, y)
```

Then, as usual, you can use the model to make predictions:

```
>>> svm_clf.predict([[5.5, 1.7]])
array([1.])
```

3. SVM Regression

The SVM algorithm is quite versatile: not only does it support linear and nonlinear classification, but it also supports linear and nonlinear regression. The trick is to reverse the objective: instead of trying to fit the largest possible street between two classes while limiting margin violations, SVM Regression tries to fit as many instances as possible *on* the street while limiting margin violations (i.e., instances *off* the street). The width of the street is controlled by a hyperparameter ϵ.

Some of the key parameters used are as mentioned below:

1. Hyperplane:

Hyperplanes are decision boundaries that is used to predict the continuous output. The data points on either side of the hyperplane that are closest to the hyperplane are called Support Vectors. These are used to plot the required line that shows the predicted output of the algorithm.

2. Kernel:

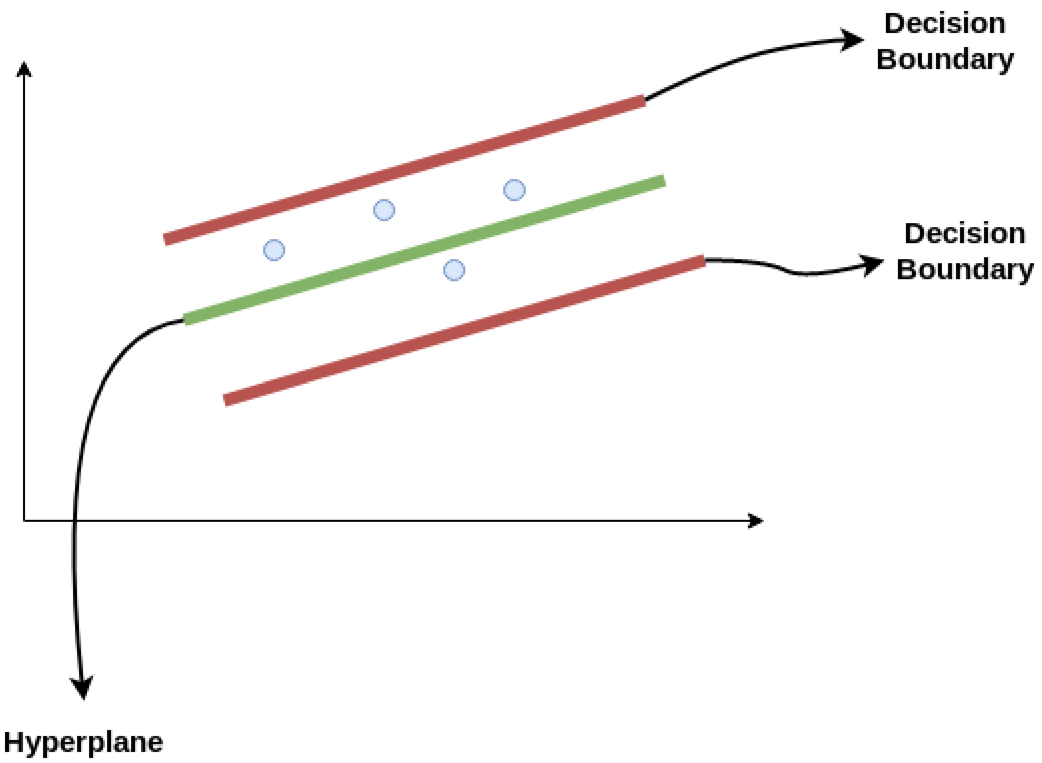
A kernel is a set of mathematical functions that takes data as input and transform it into the required form. These are generally used for finding a hyperplane in the higher dimensional space. The most widely used kernels include **Linear**, **Non-Linear**, **Polynomial**, **Radial Basis Function (RBF)** and **Sigmoid**. By default, RBF is used as the kernel. Each of these kernels are used depending on the dataset.

3. Boundary Lines:

These are the two lines that are drawn around the hyperplane at a distance of **ε (epsilon)**. It is used to create a margin between the data points.

The Idea Behind Support Vector Regression

The problem of regression is to find a function that approximates mapping from an input domain to real numbers on the basis of a training sample. So let's now dive deep and understand how SVR works actually.



Consider these two red lines as the decision boundary and the green line as the hyperplane. **Our objective, when we are moving on with SVR, is to basically consider the points that are within the decision boundary line.** Our best fit line is the hyperplane that has a maximum number of points.

The first thing that we'll understand is what is the decision boundary (the danger red line above!). Consider these lines as being at any distance, say 'a', from the hyperplane. So, these are the lines that we draw at distance '+a' and '-a' from the hyperplane. This 'a' in the text is basically referred to as epsilon.

Assuming that the equation of the hyperplane is as follows:

$Y = wx+b$ (equation of hyperplane)

Then the equations of decision boundary become:

$wx+b= +a$

$wx+b= -a$

Thus, any hyperplane that satisfies our SVR should satisfy:

$\mathbf{-a < Y- wx+b < +a}$

Our main aim here is to decide a decision boundary at 'a' distance from the original hyperplane such that data points closest to the hyperplane or the support vectors are within that boundary line.

Hence, we are going to take only those points that are within the decision boundary and have the least error rate, or are within the Margin of Tolerance. This gives us a better fitting model.

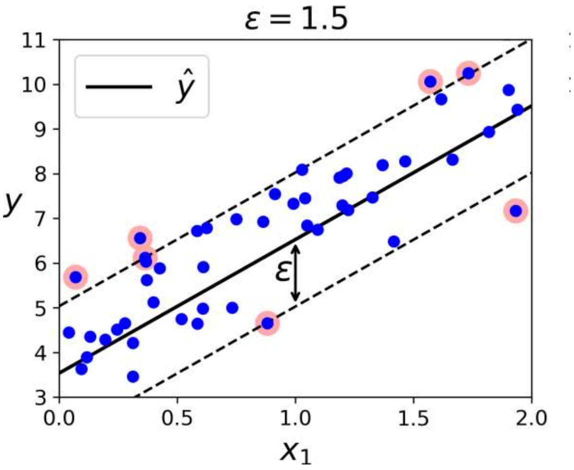
You can use Scikit-Learn's LinearSVR class to perform linear SVM Regression

```
from sklearn.svm import LinearSVR

svm_reg = LinearSVR(epsilon=1.5)

svm_reg.fit(X, y)
```

Output:



4. Naïve Bayes Classifiers

- Naïve Bayes algorithm is a supervised learning algorithm, which is based on **Bayes theorem** and used for solving classification problems.
- It is mainly used in *text classification* that includes a high-dimensional training dataset.
- Naïve Bayes Classifier is one of the simple and most effective Classification algorithms which helps in building the fast machine learning models that can make quick predictions.
- **It is a probabilistic classifier, which means it predicts on the basis of the probability of an object**.
- Some popular examples of Naïve Bayes Algorithm are **spam filtration, Sentimental analysis, and classifying articles**.

Why is it called Naïve Bayes?

The Naïve Bayes algorithm is comprised of two words Naïve and Bayes, Which can be described as:

- **Naïve**: It is called Naïve because it assumes that the occurrence of a certain feature is independent of the occurrence of other features. Such as if the fruit is identified on the bases of color, shape, and taste, then red, spherical, and sweet fruit is recognized as an apple. Hence each feature individually contributes to identify that it is an apple without depending on each other.
- **Bayes**: It is called Bayes because it depends on the principle of Bayes' Theorem.

Bayes' Theorem:

- Bayes' theorem is also known as **Bayes' Rule** or **Bayes' law**, which is used to determine the probability of a hypothesis with prior knowledge. It depends on the conditional probability.
- The formula for Bayes' theorem is given as:

$$P(A|B)=\frac{P(B|A)P(A)}{P(B)}$$

Where,

P(A|B) is Posterior probability: Probability of hypothesis A on the observed event B.

P(B|A) is Likelihood probability: Probability of the evidence given that the probability of a hypothesis is true.

P(A) is Prior Probability: Probability of hypothesis before observing the evidence.

P(B) is Marginal Probability: Probability of Evidence.

Working of Naïve Bayes' Classifier:

Working of Naïve Bayes' Classifier can be understood with the help of the below example:

Suppose we have a dataset of **weather conditions** and corresponding target variable "**Play**". So using this dataset we need to decide that whether we should play or not on a particular day according to the weather conditions. So to solve this problem, we need to follow the below steps:

1. Convert the given dataset into frequency tables.
2. Generate Likelihood table by finding the probabilities of given features.
3. Now, use Bayes theorem to calculate the posterior probability.

Problem: If the weather is sunny, then the Player should play or not?

Solution: To solve this, first consider the below dataset:

	Outlook	**Play**
0	Rainy	Yes
1	Sunny	Yes
2	Overcast	Yes
3	Overcast	Yes
4	Sunny	No
5	Rainy	Yes
6	Sunny	Yes
7	Overcast	Yes
8	Rainy	No
9	Sunny	No
10	Sunny	Yes
11	Rainy	No
12	Overcast	Yes
13	Overcast	Yes

Frequency table for the Weather Conditions:

Weather	Yes	No
Overcast	5	0
Rainy	2	2
Sunny	3	2
Total	10	5

Likelihood table weather condition:

Weather	Yes	No
Overcast	5	0
Rainy	2	2
Sunny	3	2
Total	10	5

Applying Bayes' theorem:

P(Yes|Sunny)= P(Sunny|Yes)*P(Yes)/P(Sunny)

P(Sunny|Yes)= 3/10= 0.3

P(Sunny)= 0.35

P(Yes)=0.71

Learn more

So P(Yes|Sunny) = 0.3*0.71/0.35= **0.60**

P(No|Sunny)= P(Sunny|No)*P(No)/P(Sunny)

P(Sunny|NO)= 2/4=0.5

P(No)= 0.29

P(Sunny)= 0.35

So P(No|Sunny)= 0.5*0.29/0.35 = **0.41**

So as we can see from the above calculation that **P(Yes|Sunny)>P(No|Sunny)**

Hence on a Sunny day, Player can play the game.

Advantages of Naïve Bayes Classifier:

- Naïve Bayes is one of the fast and easy ML algorithms to predict a class of datasets.
- It can be used for Binary as well as Multi-class Classifications.
- It performs well in Multi-class predictions as compared to the other Algorithms.
- It is the most popular choice for **text classification problems**.

Disadvantages of Naïve Bayes Classifier:

- Naive Bayes assumes that all features are independent or unrelated, so it cannot learn the relationship between features.

Applications of Naïve Bayes Classifier:

- It is used for **Credit Scoring**.
- It is used in **medical data classification**.
- It can be used in **real-time predictions** because Naïve Bayes Classifier is an eager learner.
- It is used in Text classification such as **Spam filtering** and **Sentiment analysis**.