Certification Training Courses for Graduates and Professionals

www.signitivetech.com

Classification
Part 2

Naive Bayes Classifier

Naive Bayes Classifier

- It is a classification technique based on Bayes' Theorem with an assumption of independence among predictors. In simple terms, a Naive Bayes classifier assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature.
- Naive Bayes model is easy to build and particularly useful for very large data sets. Along with simplicity, Naive Bayes is known to outperform even highly sophisticated classification methods.
- Naive Bayes classifier gives great results when we use it for textual data analysis. Such as Natural Language Processing.
- Bayes theorem named after Rev. Thomas Bayes. It works on conditional probability. Conditional probability is the probability that something will happen, given that something else has already occurred. Using the conditional probability, we can calculate the probability of an event using its prior knowledge.

Naive Bayes Classifier

- Naive Bayes is a family of algorithms based on applying Bayes theorem with a strong(naive) assumption, that every feature is independent of the others, in order to predict the category of a given sample.
- They are probabilistic classifiers, therefore will calculate the probability of each category using Bayes theorem, and the category with the highest probability will be output.
- Naive Bayes classifier is a straightforward and powerful algorithm for the classification task. Even if we are working on a data set with millions of records with some attributes, it is suggested to try Naive Bayes approach.

Naive Bayes Classifier

Bayes' Theorem:

$$P(A \mid B) = \frac{P(B \mid A) * P(A)}{P(B)}$$

It tells us how often A happens given that B happens, written P(A|B), when we know how often B happens given that A happens, written P(B|A), and how likely A and B are on their own.

- P(A|B) is "Probability of A given B", the probability of A given that B happens. This is the conditional probability that event A occurs, given that B has occurred.
- P(A) is Probability of A
- P(B|A) is "Probability of B given A", the probability of B given that A happens. This is the conditional probability that event B occurs, given that A has occurred.
- P(B) is Probability of B

Naive Bayes Classifier

- P(A|B) : This is the conditional probability of response variable belonging to a particular value, given the input attributes. This is also known as the posterior probability.
- P(A) : This is the prior probability of the response variable.
- P(B) : This is the probability of training data or the evidence.
- P(B|A) : This is known as the likelihood of the training data.

Therefore above equation can be written as:

$$\text{posterior} = \frac{\text{prior} * \text{likelihood}}{\text{evidence}}$$

Naive Bayes Classifier

- 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. Bayes' Theorem is stated as:

$$P(h|d) = (P(d|h) * P(h)) / P(d)$$

- **P(h|d)** is the probability of hypothesis h given the data d.
- **P(d|h)** is the probability of data d given that the hypothesis h was true.
- **P(h)** is the probability of hypothesis h being true (regardless of the data).
- **P(d)** is the probability of the data (regardless of the hypothesis).

Naive Bayes Classifier

- We can see that we are interested in calculating the posterior probability of P(h|d) from the prior probability p(h) with P(D) and P(d|h).
- After calculating the posterior probability for a number of different hypotheses, we can select the hypothesis with the highest probability. This is the maximum probable hypothesis and may formally be called the maximum a posteriori (MAP) hypothesis. This can be written as:

$$MAP(h) = max(P(h|d))$$

Or

$$MAP(h) = max((P(d|h) * P(h)) / P(d))$$

Or

$$MAP(h) = max(P(d|h) * P(h))$$

Naive Bayes Classifier

- The P(d) is a normalizing term which allows us to calculate the probability. We can drop it when we are interested in the most probable hypothesis as it is constant and only used to normalize.
- Back to classification, if we have an even number of instances in each class in our training data, then the probability of each class (e.g. P(h)) will be equal. Again, this would be a constant term in our equation and we could drop it so that we end up with:

$$MAP(h) = max(P(d|h))$$

Naive Bayes Classifier

- Naive Bayes classifier assumes that all the features are **unrelated** to each other. Presence or absence of a feature does not influence the presence or absence of any other feature. Wikipedia example for explaining the logic:
- **"A fruit may be considered to be an apple if it is red, round, and about 4" in diameter. Even if these features depend on each other or upon the existence of the other features, a naive Bayes classifier considers all of these properties to independently contribute to the probability that this fruit is an apple."**
- In real datasets, we test a hypothesis given multiple evidence (features). So, calculations become complicated. To simplify the work, the feature independence approach is used to 'uncouple' multiple evidence and treat each as an independent one.

$$P(H \mid \text{Multiple Evidences}) = P(E1|H) * P(E2|H) *\ldots\ldots * P(En|H) * P(H) / P(\text{Multiple Evidences})$$

Naive Bayes Classifier

- Consider a dataset of weather and corresponding variable play. Task is to classify whether players will play or not based on weather conditions.

Weather	Play		
Sunny	No	Rainy	No
Overcast	Yes	Sunny	Yes
Rainy	Yes	Rainy	Yes
Sunny	Yes	Sunny	No
Sunny	Yes	Overcast	Yes
Overcast	Yes	Overcast	Yes
Rainy	No	Rainy	No

Naive Bayes Classifier

- Problem Statement: Players will Play if weather is Sunny → Is this statement is correct or not?
- To solve this problem we use Naive Bayes Theorem. First we have to draw two tables, one is frequency table and second is likelihood table before applying Bayes theorem as follows:

Frequency Table			
Weather | Play	No	Yes	Total
Overcast	0	4	4
Rainy	3	2	5
Sunny	2	3	5
Total	5	9	14

Naive Bayes Classifier

Likelihood Table				
Weather | Play	No	Yes	Total	
Overcast	0/5 = 0.00 (likelihood of Overcast | No)	4/9 = 0.44 (likelihood of Overcast | Yes)	4	= 4/14 = 0.29 (likelihood of Overcast)
Rainy	3/5 = 0.60 (likelihood of Rainy | No)	2/9 = 0.22 (likelihood of Rainy | Yes)	5	= 5/14 = 0.36 (likelihood of Rainy)
Sunny	2/5 = 0.40 (likelihood of Sunny | No)	3/9 = 0.33 (likelihood of Sunny | Yes)	5	= 5/14 = 0.36 (likelihood of Sunny)
Total	5	9		
	= 5/14 = 0.36 (likelihood of No)	= 9/14 = 0.64 (likelihood of Yes)		

Naive Bayes Classifier

- Here, problem statement can be written as what is the probability that Players will Play if weather is Sunny i.e. we have to calculate P(Yes | Sunny)
- However, we will have to calculate Probabilities for all Hypothesis, i.e. probability of players will play and probability of players will not play when weather is sunny.
- We will first calculate Probability of Yes when weather is Sunny:
- So applying Bayes theorem, equation will look like:

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

P(Sunny | Yes) = 3/9 = 0.33

P(Yes) = 9/14 = 0.64

P(Sunny) = 5/14 = 0.36

P(Yes | Sunny) = 0.33 * 0.64 /0.36 = 0.60

Naive Bayes Classifier

Similarly:

- We will calculate Probability of No when weather is Sunny:
- So applying Bayes theorem, equation will look like:

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

P(Sunny | No) = 2/5 = 0.40

P(No) = 5/14 = 0.36

P(Sunny) = 5/14 = 0.36

P(No | Sunny) = 0.40 * 0.36 / 0.36 = 0.40

- Now, as P(Yes | Sunny) i.e. 0.60 is greater than P(No | Sunny) i.e. 0.40, the answer is Players will Play when Weather is Sunny is Correct Statement.

Naive Bayes Classifier

Types of Naive Bayes Classifier:

- **Gaussian Naive Bayes:** When the predictors take up a continuous value and are not discrete, we assume that these values are sampled from a gaussian distribution i.e. when attribute values are continuous, an assumption is made that the values associated with each class are distributed according to Gaussian i.e. Normal Distribution.
- **Multinomial Naive Bayes:** This is mostly used for document classification problem, i.e whether a document belongs to the category of sports, politics, technology etc. The features/predictors used by the classifier are the frequency of the words present in the document. MultiNomial Naive Bayes is preferred to use on data that is multinomial distribution. It is one of the standard classic algorithms which is used in text categorization (classification). Each event in text classification represents the occurrence of a word in a document.

Naive Bayes Classifier

Types of Naive Bayes Classifier:

- **Bernoulli Naive Bayes:** Bernoulli Naive Bayes is used on the data that is distributed according to multivariate Bernoulli distributions .i.e. multiple features can be there, but each one is assumed to be a binary-valued (Bernoulli / boolean values) variable. So, it requires features to be binary valued.

Naive Bayes Classifier

Advantages:

- Naive Bayes Algorithm is a fast, highly scalable algorithm.
- Naive Bayes can be use for Binary and Multiclass classification.
- It is a simple algorithm that depends on doing a bunch of counts.
- Great choice for Text Classification and Spam Email Classification.
- It can be easily train on small dataset.

Disadvantages:

- It considers all the features to be unrelated, so it cannot learn the relationship between features. Naive Bayes can learn individual features importance but can’t determine the relationship among features.

Naive Bayes Classifier

Steps to Perform Naive Bayes Classification using Scikit-Learn:

- Import required libraries
- Load the Dataset
- Do the Data preprocessing
- Separate features and target
- Feature Scaling using Standard Scalar
- Split the data into train and test part

Naive Bayes Classifier

Steps to Perform Naive Bayes Classification using Scikit-Learn:

- Train the Algorithm: Import the GaussianNB class, instantiate it, and call the fit() method along with our training data, when we want to use Gaussian Naive Bayes Classifier.

```
from sklearn.naive_bayes import GaussianNB
model = GaussianNB()
model.fit(X_train, Y_train)
```

- Make the Predictions: To make predictions, use testing data.

```
y_predicted = model.predict(x_test)
```

- Evaluate the Model by calculating Confusion Matrix.

Naive Bayes Classifier

Steps to Perform Naive Bayes Classification using Scikit-Learn:

- Train the Algorithm: Import the MultinomialNB class, instantiate it, and call the fit() method along with our training data, when we want to use Multinomial Naive Bayes Classifier

```
from sklearn.naive_bayes import MultinomialNB
model = MultinomialNB()
model.fit(X_train, Y_train)
```

- Make the Predictions: To make predictions, use testing data.

```
y_predicted = model.predict(x_test)
```

- Evaluate the Model by calculating Confusion Matrix.

Naive Bayes Classifier

Steps to Perform Naive Bayes Classification using Scikit-Learn:

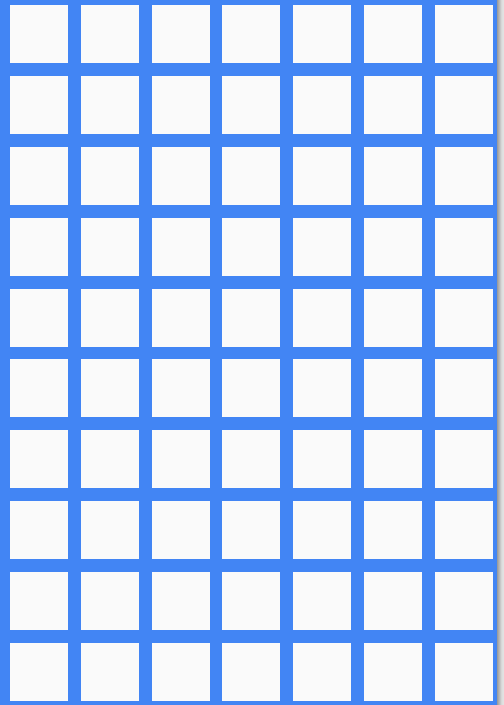
- Train the Algorithm: Import the BernoulliNB class, instantiate it, and call the fit() method along with our training data, when we want to use Bernoulli Naive Bayes Classifier

```
from sklearn.naive_bayes import BernoulliNB
model = BernoulliNB()
model.fit(X_train, Y_train)
```

- Make the Predictions: To make predictions, use testing data.

```
y_predicted = model.predict(x_test)
```

- Evaluate the Model by calculating Confusion Matrix.



Bagging and Boosting

Bootstrap Method:

- The bootstrap is a powerful statistical method for estimating a quantity from a data sample. This is easiest to understand if the quantity is a descriptive statistic such as a mean or a standard deviation.
- Let's assume we have a sample of 100 values (x) and we'd like to get an estimate of the mean of the sample. We can calculate the mean directly from the sample as: mean(x) = 1/100 * sum(x)
- We know that our sample is small and that our mean has error in it. We can improve the estimate of our mean using the bootstrap procedure explained on the next slide.

Bootstrap Method:

- Bootstrap procedure:

1. Create many (e.g. 1000) random sub-samples of our dataset with replacement (meaning we can select the same value multiple times).
2. Calculate the mean of each sub-sample.
3. Calculate the average of all of our collected means and use that as our estimated mean for the data.

- For example, let's say we used 3 resamples and got the mean values 2.3, 4.5 and 3.3. Taking the average of these we could take the estimated mean of the data to be 3.367.

Bootstrap Method:

- Bootstrap procedure:

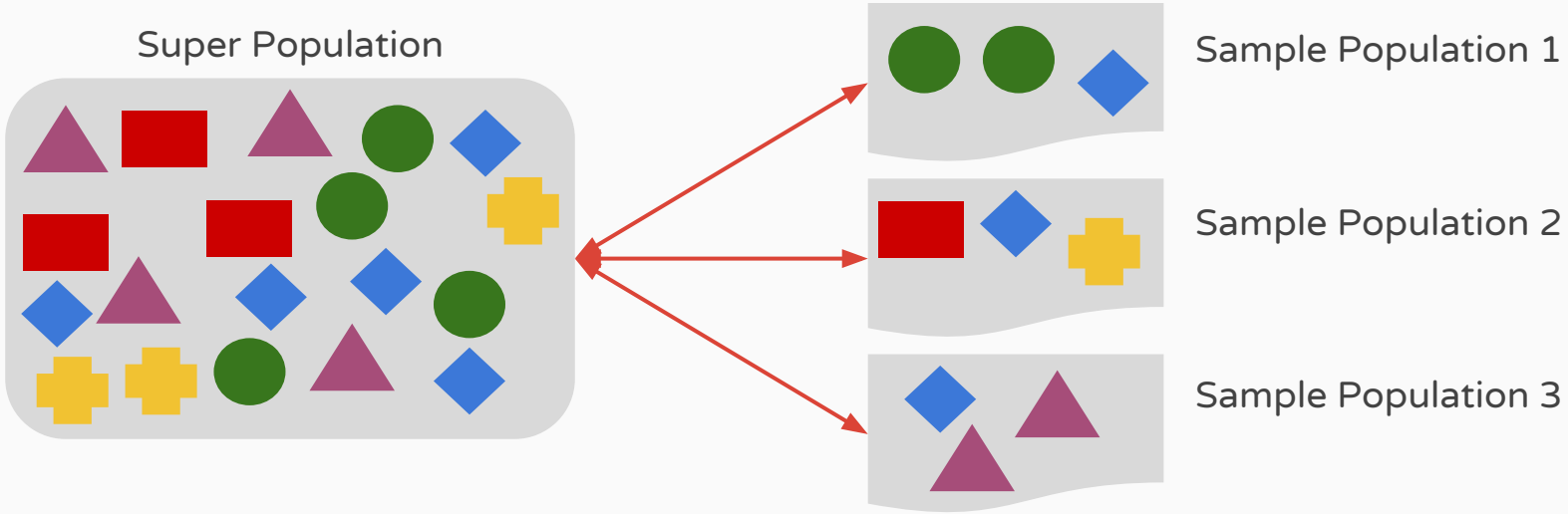
1. Create many (e.g. 1000) random sub-samples of our dataset with replacement (meaning we can select the same value multiple times).
2. Calculate the mean of each sub-sample.
3. Calculate the average of all of our collected means and use that as our estimated mean for the data.

- For example, let's say we used 3 resamples and got the mean values 2.3, 4.5 and 3.3. Taking the average of these we could take the estimated mean of the data to be 3.367.

Bagging and Boosting

Bootstrap Method:

- Bootstrap refers to random sampling with replacement.
- Bootstrap allows us to better understand the bias and the variance.
- Bootstrap involves random sampling of small subset of data from the dataset. This subset can be replace.
- The selection of all the example in the dataset has equal probability.



Bagging and Boosting

Ensembling:

- Ensemble methods are meta-algorithms that combine several machine learning techniques into one predictive model in order to decrease variance(bagging), bias (boosting), or improve predictions (stacking).
- An ensemble method is a technique that combines the predictions from multiple machine learning algorithms together to make more accurate predictions than any individual model.
- Ensemble methods combine several decision trees classifiers to produce better predictive performance than a single decision tree classifier.
- The main principle behind the ensemble model is that a group of weak learners come together to form a strong learner, thus increasing the accuracy of the model. When we try to predict the target variable using any machine learning technique, the main causes of difference in actual and predicted values are noise, variance, and bias. Ensemble helps to reduce these factors (except noise, which is irreducible error).

Bagging and Boosting

Ensembling:

- Bagging and Boosting are similar in that they are both ensemble techniques, where a set of weak learners are combined to create a strong learner that obtains better performance than a single one.
- Using techniques like Bagging and Boosting helps to decrease the variance and increased the robustness of the model. Combinations of multiple classifiers decrease variance, especially in the case of unstable classifiers, and may produce a more reliable classification than a single classifier.
- Ensembling techniques are further classified into Bagging and Boosting.

Bagging and Boosting

Ensembling: Ensemble methods can be divided into two groups:

- Sequential ensemble methods where the base learners are generated sequentially (e.g. AdaBoost).
- The basic motivation of sequential methods is to exploit the dependence between the base learners. The overall performance can be boosted by weighing previously mislabeled examples with higher weight.
- Parallel ensemble methods where the base learners are generated in parallel (e.g. Random Forest).
- The basic motivation of parallel methods is to exploit independence between the base learners since the error can be reduced dramatically by averaging.
- Most ensemble methods use a single base learning algorithm to produce homogeneous base learners, i.e. learners of the same type, leading to homogeneous ensembles.

Bagging and Boosting

Bagging:

- **Bagging** is a simple ensembling technique in which we build many independent predictors/models/learners and combine them using some model averaging techniques. (e.g. weighted average, majority vote or normal average)
- We typically take random subsample/bootstrap of data for each model, so that all the models are little different from each other.
- Each observation is chosen with replacement to be used as input for each of the model.
- So, each model will have different observations based on the bootstrap process.
- Because this technique takes many uncorrelated learners to make a final model, it reduces error by reducing variance.

Bagging and Boosting

Bagging:

- Bagging (Bootstrap Aggregation) is used when our goal is to reduce the variance of a decision tree. Here idea is to create several subsets of data from training sample chosen randomly with replacement.
- Now, each collection of subset data is used to train their decision trees. As a result, we end up with an ensemble of different models.
- Average of all the predictions from different trees are used which is more robust than a single decision tree.
- Bagging uses bootstrap sampling to obtain the data subsets for training the base learners.
- For aggregating the outputs of base learners, bagging uses voting for classification and averaging for regression.

Bagging and Boosting

Bagging:

- When bagging with decision trees, we are less concerned about individual trees overfitting the training data. For this reason and for efficiency, the individual decision trees are grown deep (e.g. few training samples at each leaf-node of the tree) and the trees are not pruned. These trees will have both high variance and low bias. These are important characterize of sub-models when combining predictions using bagging.
- The only parameters when bagging decision trees is the number of samples and hence the number of trees to include. This can be chosen by increasing the number of trees on run after run until the accuracy begins to stop showing improvement.
- Bootstrap Aggregation (or Bagging for short), is a simple and very powerful ensemble method. Bagging is the application of the Bootstrap procedure to a high-variance machine learning algorithm, typically decision trees.

Boosting:

- Boosting refers to a family of algorithms that are able to convert weak learners to strong learners. The main principle of boosting is to fit a sequence of weak learners– models that are only slightly better than random guessing, such as small decision trees– to weighted versions of the data. More weight is given to examples that were misclassified by earlier rounds.
- The predictions are then combined through a weighted majority vote (classification) or a weighted sum (regression) to produce the final prediction. The principal difference between boosting and the committee methods, such as bagging, is that base learners are trained in sequence on a weighted version of the data.

Boosting:

- **Boosting** is another ensemble technique to create a collection of predictors. In this technique, learners are learned sequentially with early learners fitting simple models to the data and then analyzing data for errors. In other words, we fit consecutive trees (random sample) and at every step, the goal is to solve for net error from the prior tree.
- When an input is misclassified by a hypothesis, its weight is increased so that next hypothesis is more likely to classify it correctly. By combining the whole set at the end converts weak learners into better performing model.
- Boosting refers to a group of algorithms that utilize weighted averages to make weak learners into stronger learners. Unlike bagging that had each model run independently and then aggregate the outputs at the end without preference to any model. Boosting is all about “teamwork”. Each model that runs, dictates what features the next model will focus on.

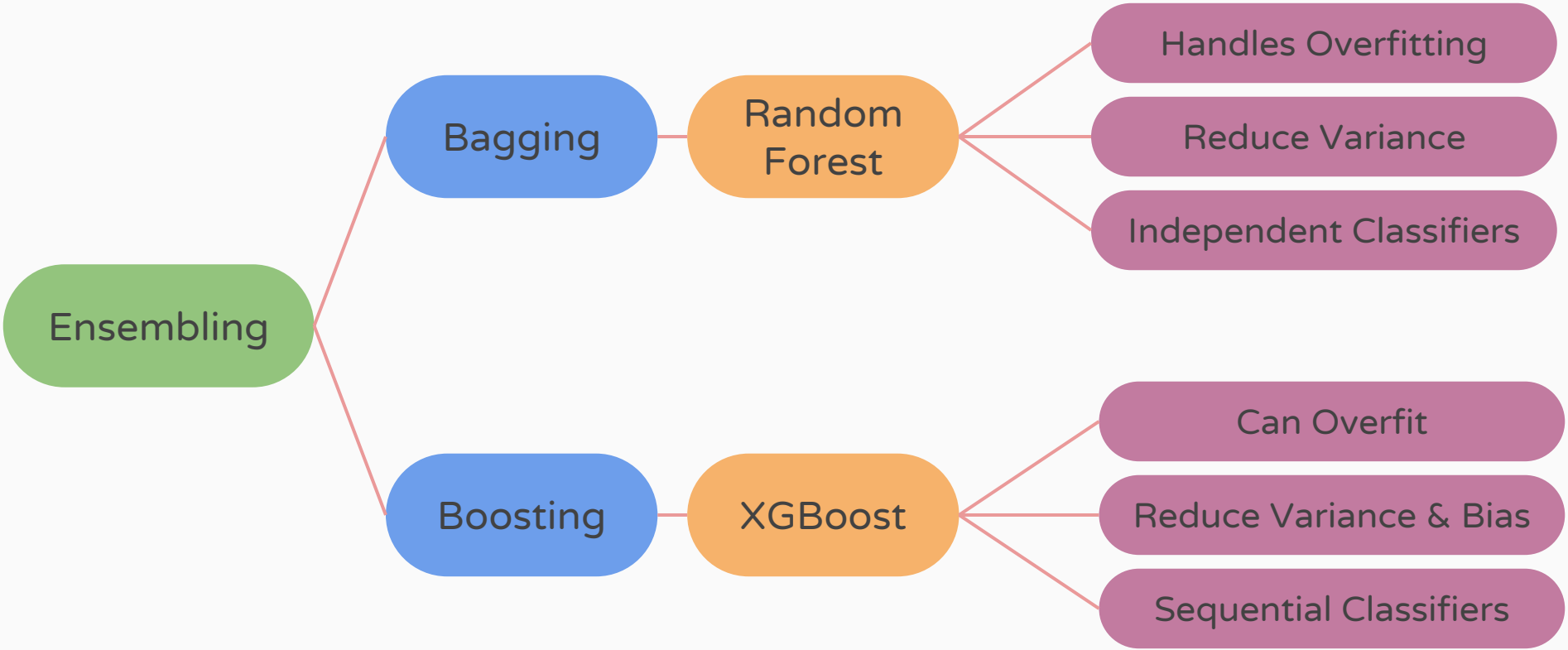
Boosting:

- Boosting also requires bootstrapping. However, there is another difference here. Unlike in bagging, boosting weights each sample of data. This means some samples will be run more often than others.
- When boosting runs each model, it tracks which data samples are the most successful and which are not. The data sets with the most misclassified outputs are given heavier weights. These are considered to be data that have more complexity and requires more iterations to properly train the model.
- During the actual classification stage, there is also a difference in how boosting treats the models. In boosting, the model's error rates are kept track of because better models are given better weights.

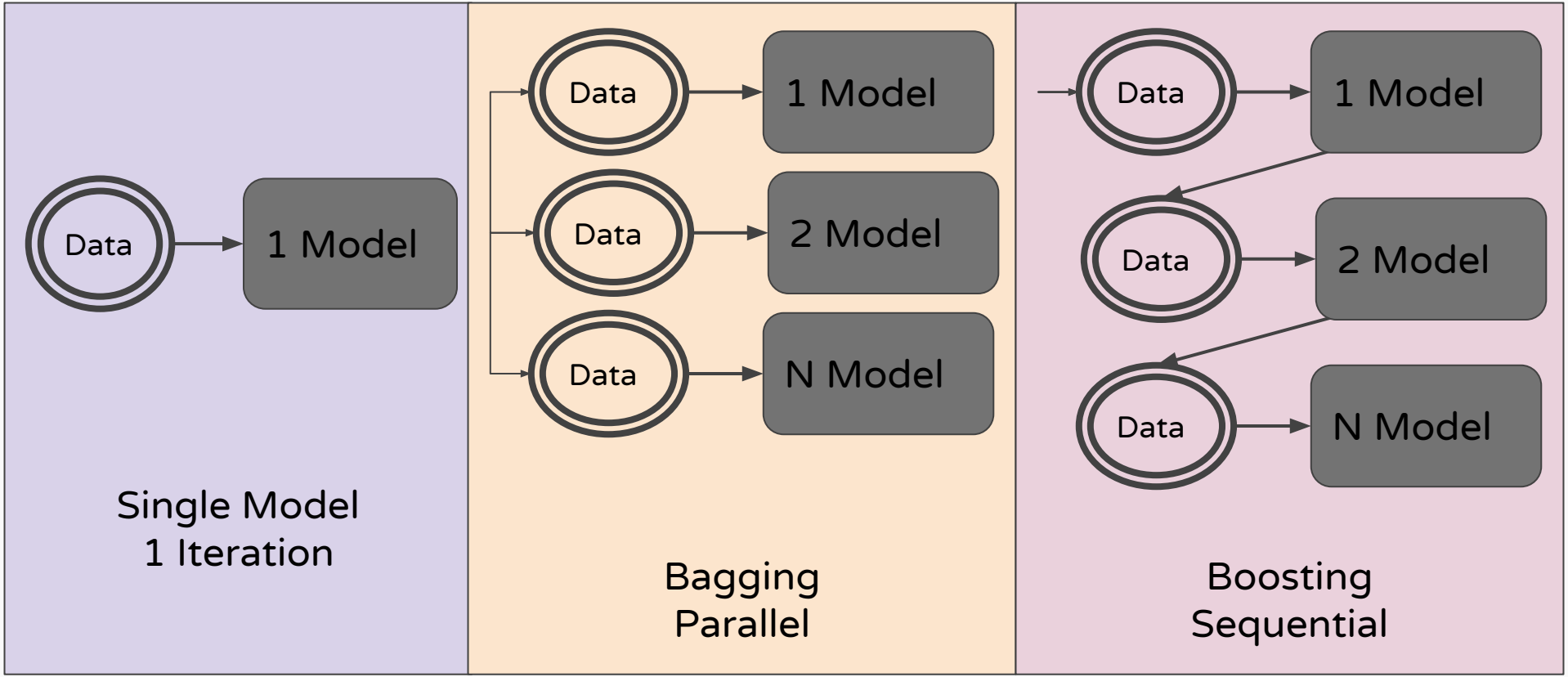
Boosting:

- **Boosting** is an ensemble technique in which the predictors are not made independently, but sequentially.
- This technique employs the logic in which the subsequent predictors learn from the mistakes of the previous predictors. Therefore, the observations have an unequal probability of appearing in subsequent models and ones with the highest error appear most. (So the observations are not chosen based on the bootstrap process, but based on the error).
- The predictors can be chosen from a range of models like decision trees, regressors, classifiers etc. Because new predictors are learning from mistakes committed by previous predictors, it takes less time/iterations to reach close to actual predictions.
- But we have to choose the stopping criteria carefully or it could lead to overfitting on training data.

Bagging and Boosting

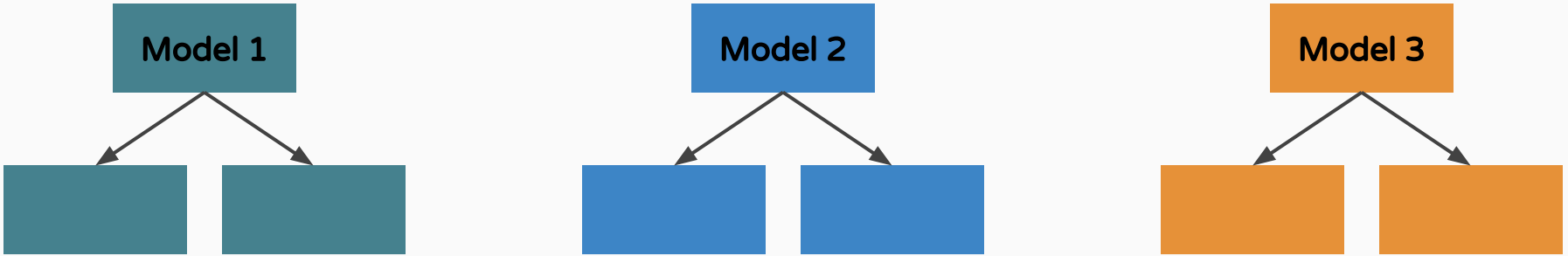


Bagging and Boosting



Working of Bagging Technique:

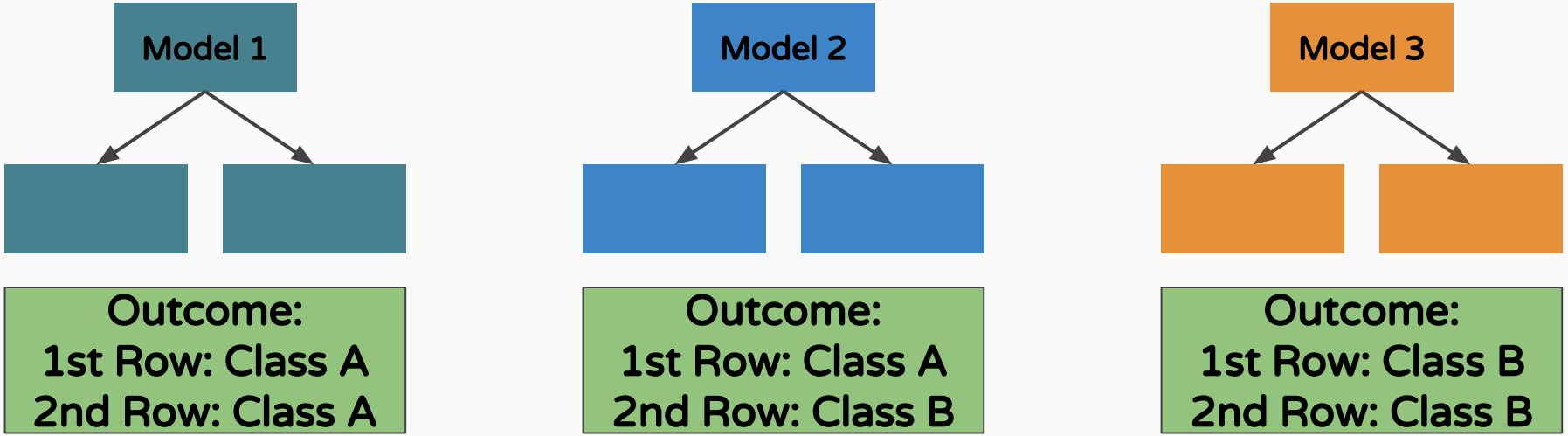
Suppose we have the Dataset containing M rows and K features, in Bagging, first we will select 1 Bootstrap Sample and we will draw 1 Decision Tree on this sample by selecting feature node on Best Split Methods. Using this 1 Model we will predict our Target. The process will go on upto number of Decision Trees or Bootstrap Samples we want to draw from Dataset. Consider we will create 3 Samples and thus will create 3 Decision Trees:



Now we have testing set of two observations. After creating Bootstrap models on training set, we will test our testing rows on these models one by one. We will record the final outcome i.e. our target given by each bootstrap or decision tree model.

Working of Bagging Technique:

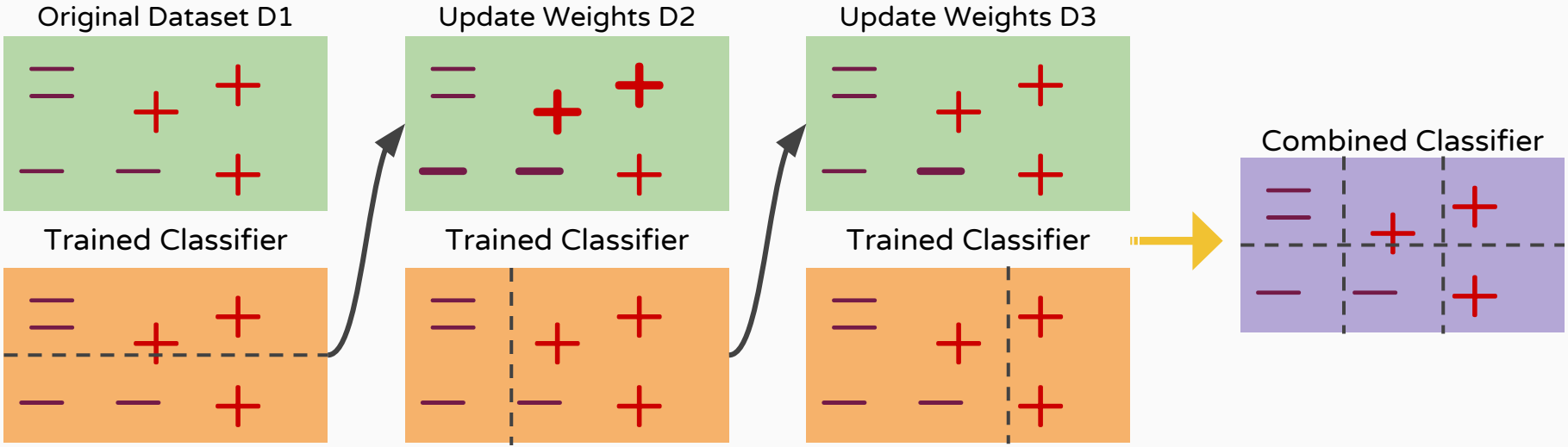
Suppose for Classification we have 2 Classes: A and B
Now Model 1 is giving class A for 1st Row and Class A for 2nd Row.
Model 2 is giving Class A for 1st Row and Class B for 2nd Row.
And Model 3 is giving Class B for 1st Row and Class B for 2nd Row.
By Majority Voting we will classify ROW 1 in CLASS A and ROW 2 in CLASS B



Bagging and Boosting

Working of Boosting Technique:

Suppose we have the Dataset, in Boosting, first we will create a Decision Tree and and Initialize weights (any random numbers) to each Record. Then we will record which Rows get correctly classified and which Rows get misclassified. For those Rows which get misclassified we will Increase the Weights and again create the Decision Tree. The Process go on until the Misclassification Rate or Error Rate gets minimized or untill number of Decision Trees we want to Calculate and Draw.



Random Forest

Random Forest

- Random forest builds multiple decision trees and merges them together to get a more accurate and stable prediction.
- The “Forest” it builds, is an ensemble of Decision Trees, trained with the “bagging” method. The general idea of the bagging method is that a combination of learning models increases the overall result.
- Random forest can be used for both classification and regression problems, which form the majority of current machine learning systems.
- Random Forest adds additional randomness to the model, while growing the trees. Instead of searching for the most important feature while splitting a node, it searches for the best feature among a random subset of features. This results in a wide diversity that results in a better model.
- Therefore, in Random Forest, only a random subset of the features is taken into consideration by the algorithm for splitting a node.

Random Forest

- The random forest is a model made up of many decision trees. Rather than just simply averaging the prediction of trees (forest), this model uses two key concepts that gives it the name **random**:

➔ Random sampling of training data points when building trees

➔ Random subsets of features considered when splitting nodes

In Short: The random forest combines hundreds or thousands of decision trees, trains each one on a slightly different set of the observations, splitting nodes in each tree considering a limited number of the features. The final predictions of the random forest are made by averaging the predictions of each individual tree.

Random Forest

Random sampling of training observations:

- When training, each tree in a random forest learns from a random sample of the data points.
- The samples are drawn with replacement, known as bootstrapping, which means that some samples will be used multiple times in a single tree.
- The idea is that by training each tree on different samples, although each tree might have high variance with respect to a particular set of the training data, overall, the entire forest will have lower variance but not at the cost of increasing the bias.
- At test time, predictions are made by averaging the predictions of each decision tree.
- This procedure of training each individual learner on different bootstrapped subsets of the data and then averaging the predictions is known as bagging, short for bootstrap aggregating.

Random Forest

Random Subsets of features for splitting nodes:

- The other main concept in the random forest is that only a subset of all the features are considered for splitting each node in each decision tree.
- The number of features that can be searched at each split point (m) must be specified as a parameter to the algorithm.

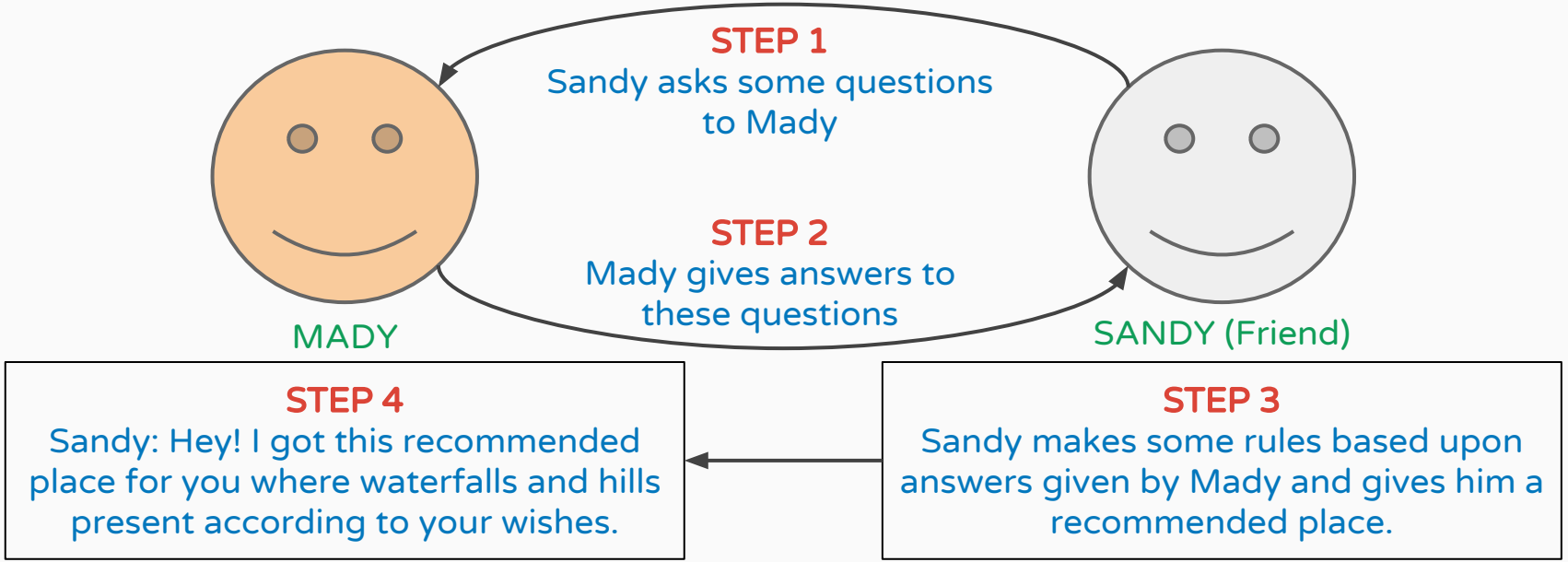
For classification a good default is: m = sqrt(p)

For regression a good default is: m = p/3

Where m is the number of randomly selected features that can be searched at a split point and p is the number of input variables.

Random Forest

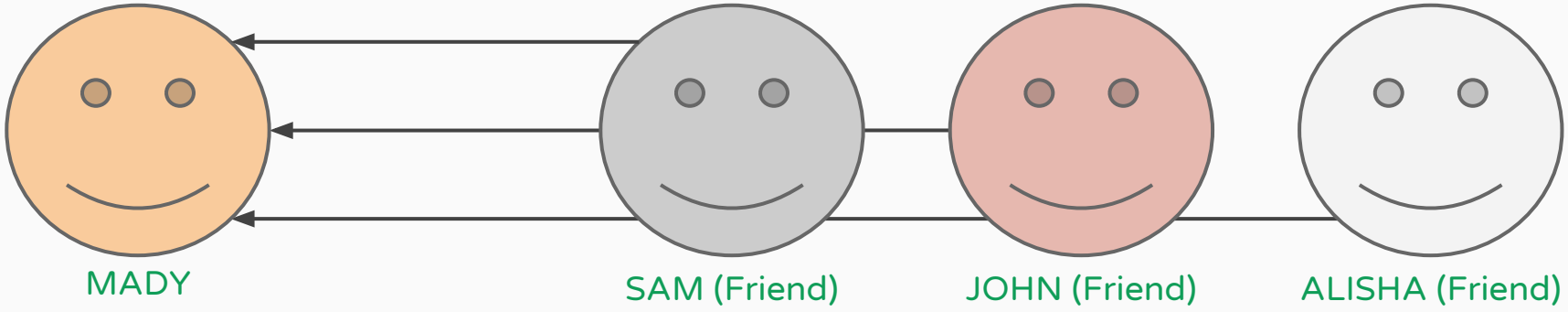
How Decision Tree Algorithm works: Recommend Best Place for MADY



Sandy makes Decision Tree. The Recommended Place is the leaf of the decision tree i.e. Target Class and this target is finalized by a single person i.e. Only One Decision Tree.

Random Forest

How Random Forest Algorithm works: Recommend Best Place for MADY



- Each friend asks Mady different questions and Mady gives answers to them.
- Later Mady consider all these recommendations and calclutes votes i.e. number of count or popular places from these recommendations.

Here, the recommendation is the Target Prediction. Each friend is a tree and combined all friends i.e. trees will form the forest. Random Tree will be selected by Mady based upon highest count of Votes (Majority).

Random Forest

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 Random Forest creation pseudocode:

- Randomly select "**K**" features from total "**m**" features where **k << m**
- Among the "**K**" features, calculate the node "**d**" using the best split point
- Split the node into **daughter nodes** using the **best split**
- Repeat the **a to c** steps until "l" number of nodes has been reached
- Build forest by repeating steps **a to d** for "n" number times to create **"n" number of trees**

Random Forest

How Random Forest algorithm works?

- In the next stage, with the random forest classifier created, we will make the prediction.
- The Random Forest prediction pseudocode:

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

Random Forest

Variable Importance:

- As the Bagged decision trees are constructed, we can calculate how much the error function drops for a variable at each split point.
- In regression problems this may be the drop in sum squared error and in classification this might be the Gini score.
- These drops in error can be averaged across all decision trees and output to provide an estimate of the importance of each input variable. The greater the drop when the variable was chosen, the greater the importance.
- These outputs can help identify subsets of input variables that may be most or least relevant to the problem and suggest at possible feature selection experiments you could perform where some features are removed from the dataset.

Random Forest

Out Of Bag Error:

- For each bootstrap sample taken from the training data, there will be samples left behind that were not included. These samples are called Out-Of-Bag samples or OOB.
- The performance of each model on its left out samples when averaged can provide an estimated accuracy of the bagged models. This estimated performance is often called the OOB estimate of performance.
- Random trees are constructed using different bootstrap samples of the original dataset. Approximately 37% of inputs are left out of a particular bootstrap sample and are not used in the construction of the **K**-th tree.
- To sum up, each base algorithm is trained on ~ 63% of the original examples. It can be validated on the remaining ~37%. The Out-of-Bag estimate is nothing more than the mean estimate of the base algorithms on those ~37% of inputs that were left out of training.

Random Forest

Applications of Random Forest:

- **Banking:** Random Forest algorithm is used to find loyal customers, which means customers who can take out plenty of loans and pay interest to the bank properly, and fraud customers, which means customers who have bad records like failure to pay back a loan on time or have dangerous actions.
- **Medicine:** Random Forest algorithm can be used to both identify the correct combination of components in medicine, and to identify diseases by analyzing the patient's medical records.
- **Stock Market:** Random Forest algorithm can be used to identify a stock's behavior and the expected loss or profit.
- **E-Commerce:** Random Forest algorithm can be used for predicting whether the customer will like the recommend products, based on the experience of similar customers.

Random Forest

Steps to Perform Random Forest Classification using Scikit-Learn:

- Import required libraries
- Load the Dataset
- Do the Data preprocessing
- Separate features and target
- Feature Scaling using Standard Scalar
- Split the data into train and test part

Random Forest

Steps to Perform Random Forest Classification using Scikit-Learn:

- Train the Algorithm: Import the RandomForestClassifier class, instantiate it, and call the fit() method along with our training data.

```
from sklearn.ensemble import RandomForestClassifier
model = RandomForestClassifier()
model.fit(X_train, Y_train)
```

- Make the Predictions: To make predictions, use testing data.

```
y_predicted = model.predict(x_test)
```

- Evaluate the Model by calculating Confusion Matrix.

Random Forest

Steps to Perform Random Forest Regression using Scikit-Learn:

- Import required libraries
- Load the Dataset
- Check the relationship between Independent and Dependent Variables
- Do the Data preprocessing
- Separate features and target
- Feature Scaling using Standard Scalar
- Split the data into train and test part

Random Forest

Steps to Perform Random Forest Regression using Scikit-Learn:

- Train the Algorithm: Import the RandomForestRegressor class, instantiate it, and call the fit() method along with our training data.

```
from sklearn.ensemble import RandomForestRegressor
model = RandomForestRegressor()
model.fit(X_train, Y_train)
```

- Make the Predictions: To make predictions, use testing data.

```
y_predicted = model.predict(x_test)
```

- Evaluate the Model by calculating MSE, MAE and R2 Score

Random Forest

Parameters:

- n_estimators: The number of trees in the forest (default=10)
- criterion: The function to measure the quality of a split. Supported criteria are “gini” for the Gini impurity and “entropy” for the information gain (default = “gini”) for Classifier and “mse” for mean squared error, which is equal to variance reduction as feature selection criterion and “mae” for mean absolute error (default = “mse”) for Regressor.
- max_features: The number of features to consider when looking for the best split (default="auto")
  - If int, then consider ‘max_features’ features at each split
  - If float, then ‘max_features’ is a percentage and ‘int(max_features * n_features)’ features are considered at each split
  - If "auto", then ‘max_features=sqrt(n_features)’
  - If "sqrt", then ‘max_features=sqrt(n_features)’ (same as "auto")
  - If "log2", then ‘max_features=log2(n_features)’
  - If None, then ‘max_features=n_features’

Random Forest

Parameters:

- max_depth: The maximum depth of the tree. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples (default=None)
- min_samples_split: The minimum number of samples required to split an internal node (default=2)
- min_samples_leaf: The minimum number of samples required to be at a leaf node (default=1)
- bootstrap: Whether bootstrap samples are used when building trees (default=True)
- oob_score: Whether to use out-of-bag samples to estimate the generalization accuracy (default=False)

Extra Trees

Extra Trees

- ExtraTreesClassifier is an ensemble learning method fundamentally based on decision trees. ExtraTreesClassifier, like RandomForest, randomizes certain decisions and subsets of data to minimize over-learning from the data and overfitting.
- Let's look at some ensemble methods ordered from high to low variance, ending in ExtraTreesClassifier.

Decision Tree (High Variance)

- A single decision tree is usually overfits the data it is learning from because it learn from only one pathway of decisions.
- Predictions from a single decision tree usually don't make accurate predictions on new data.

Random Forest (Medium Variance)

- Random forest models reduce the risk of overfitting by introducing randomness by
  - building multiple trees (n_estimators)
  - drawing observations with replacement (a bootstrapped sample)
  - splitting nodes on the best split among a **random subset** of the features selected at every node

Extra Trees

Extra Trees (Low Variance)

- Extra Trees is like Random Forest, in that it builds multiple trees and splits nodes using random subsets of features, but with two key differences:
  - It does not bootstrap observations (samples without replacement).
  - Nodes are split on random splits, not best splits.

So, in summary, Extra Trees:

- Extra Trees build multiple trees with **bootstrap** = **False** by default, which means it samples without replacement.
- In Extra Trees nodes are split based on **random** splits among a **random subset** of the features selected at every node.

In Extra Trees, randomness doesn't come from bootstrapping of data, but rather comes from the random splits of all observations. ExtraTrees is named for **Extremely Randomized Trees.**

Extra Trees

Steps to Perform Extra Trees Classification using Scikit-Learn:

- Import required libraries
- Load the Dataset
- Do the Data preprocessing
- Separate features and target
- Feature Scaling using Standard Scalar
- Split the data into train and test part

Extra Trees

Steps to Perform Extra Trees Classification using Scikit-Learn:

- Train the Algorithm: Import the ExtraTreesClassifier class, instantiate it, and call the fit() method along with our training data.

```
from sklearn.ensemble import ExtraTreesClassifier
model = ExtraTreesClassifier(bootstrap = False)
model.fit(X_train, Y_train)
```

- Make the Predictions: To make predictions, use testing data.

```
y_predicted = model.predict(x_test)
```

- Evaluate the Model by calculating Confusion Matrix.

Extra Trees

Steps to Perform Extra Trees Regression using Scikit-Learn:

- Import required libraries
- Load the Dataset
- Check the relationship between Independent and Dependent Variables
- Do the Data preprocessing
- Separate features and target
- Feature Scaling using Standard Scalar
- Split the data into train and test part

Extra Trees

Steps to Perform Extra Trees Regression using Scikit-Learn:

- Train the Algorithm: Import the ExtraTreesRegressor class, instantiate it, and call the fit() method along with our training data.

```
from sklearn.ensemble import ExtraTreesRegressor
model = ExtraTreesRegressor(bootstrap = False)
model.fit(X_train, Y_train)
```

- Make the Predictions: To make predictions, use testing data.

```
y_predicted = model.predict(x_test)
```

- Evaluate the Model by calculating MSE, MAE and R2 Score

Adaboost

Adaboost

- 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 is best used to boost the performance of decision trees on binary classification problems.
- Simply put, the idea is to set weights to both classifiers and data points (samples) in a way that forces classifiers to concentrate on observations that are difficult to correctly classify.
- This process is done sequentially in that the two weights are adjusted at each step as iterations of the algorithm proceed. This is why Adaboost is referred to as a sequential ensemble method

Adaboost

- To build a AdaBoost classifier, imagine that as a first base classifier we train a Decision Tree algorithm to make predictions on our training data.
- Now, following the methodology of AdaBoost, the weight of the misclassified training instances is increased.
- The second classifier is trained and acknowledges the updated weights and it repeats the procedure over and over again.
- At the end of every model prediction we end up boosting the weights of the misclassified instances so that the next model does a better job on them, and so on.

Adaboost

Below are the steps for performing the AdaBoost algorithm:

1. Initially, all observations are given equal weights.
2. A model is built on a subset of data.
3. Using this model, predictions are made on the whole dataset.
4. Errors are calculated by comparing the predictions and actual values.
5. While creating the next model, higher weights are given to the data points which were predicted incorrectly.
6. Weights can be determined using the error value. For instance, the higher the error the more is the weight assigned to the observation.
7. This process is repeated until the error function does not change, or the maximum limit of the number of estimators is reached.

Adaboost

Steps to Perform Adaboost Binary Classification using Scikit-Learn:

- Import required libraries
- Load the Dataset
- Do the Data preprocessing
- Separate features and target
- Feature Scaling using Standard Scalar
- Split the data into train and test part

Adaboost

Steps to Perform Adaboost Binary Classification using Scikit-Learn:

- Train the Algorithm: Import the AdaboostClassifier class, instantiate it, and call the fit() method along with our training data.

```
from sklearn.ensemble import AdaboostClassifier
model = AdaboostClassifier(n_estimators=50, learning_rate=1.0)
model.fit(X_train, Y_train)
```

- Make the Predictions: To make predictions, use testing data.

```
y_predicted = model.predict(x_test)
```

- Evaluate the Model by calculating Confusion Matrix.

Adaboost

Steps to Perform Adaboost Regression using Scikit-Learn:

- Import required libraries
- Load the Dataset
- Check the relationship between Independent and Dependent Variables
- Do the Data preprocessing
- Separate features and target
- Feature Scaling using Standard Scalar
- Split the data into train and test part

Adaboost

Steps to Perform Adaboost Regression using Scikit-Learn:

- Train the Algorithm: Import the AdaboostRegressor class, instantiate it, and call the fit() method along with our training data.

```
from sklearn.ensemble import AdaboostRegressor
model = AdaboostRegressor(n_estimators=50, learning_rate=1.0)
model.fit(X_train, Y_train)
```

- Make the Predictions: To make predictions, use testing data.

```
y_predicted = model.predict(x_test)
```

- Evaluate the Model by calculating MSE, MAE and R2 Score

XGBoost

XGBoost

- XGBoost (eXtreme Gradient Boosting) can be used for supervised learning tasks such as Regression, Classification, and Ranking.
- It is built on the principles of gradient boosting framework and designed to "push the extreme of the computation limits of machines to provide a scalable, portable and accurate library."
- Gradient boosting is a machine learning technique for regression and classification problems, which produces a prediction model in the form of an ensemble of weak prediction models, typically decision trees.
- It builds the model in a stage-wise fashion like other boosting methods do, and it generalizes them by allowing optimization of an arbitrary differentiable loss function.
- XGBoost is one of the implementations of Gradient Boosting concept, but what makes XGBoost unique is that it uses "a more regularized model formalization to control over-fitting, which gives it better performance." Therefore, it helps to reduce overfitting.

XGBoost

- Gradient boosting is an ensemble method that sequentially adds our trained predictors and assigns them a weight.
- However, instead of assigning different weights to the classifiers after every iteration, this method fits the new model to new residuals of the previous prediction and then minimizes the loss when adding the latest prediction.
- XGBoost specifically, implements this algorithm for decision tree boosting with an additional custom regularization term in the objective function.
- The algorithm is implemented using the Objective function(Loss Function + Penalty) with an important regularization term added to it(the Penalty, which puts the ‘Boosting’ in Gradient Boosting).

XGBoost

To implement XGBoost first we have to install this package. For this, Open Anaconda Prompt and type following command which will install xgboost.

```
pip install xgboost
```

Steps to Perform XGBoost Classification using Scikit-Learn:

- Import required libraries
- Load the Dataset
- Do the Data preprocessing
- Separate features and target
- Feature Scaling using Standard Scalar
- Split the data into train and test part

XGBoost

Steps to Perform XGBoost Classification using Scikit-Learn:

- Train the Algorithm: Import the XGBClassifier class from xgboost package, instantiate it, and call the fit() method along with our training data.

```
from xgboost import XGBClassifier
model = XGBClassifier()
model.fit(X_train, Y_train)
```

- Make the Predictions: To make predictions, use testing data.

```
y_predicted = model.predict(x_test)
```

- Evaluate the Model by calculating Confusion Matrix.

XGBoost

Steps to Perform XGBoost Regression using Scikit-Learn:

- Import required libraries
- Load the Dataset
- Check the relationship between Independent and Dependent Variables
- Do the Data preprocessing
- Separate features and target
- Feature Scaling using Standard Scalar
- Split the data into train and test part

XGBoost

Steps to Perform XGBoost Regression using Scikit-Learn:

- Train the Algorithm: Import the XGBRegressor class from xgboost package, instantiate it, and call the fit() method along with our training data.

```
from xgboost import XGBRegressor
model = XGBRegressor()
model.fit(X_train, Y_train)
```

- Make the Predictions: To make predictions, use testing data.

```
y_predicted = model.predict(x_test)
```

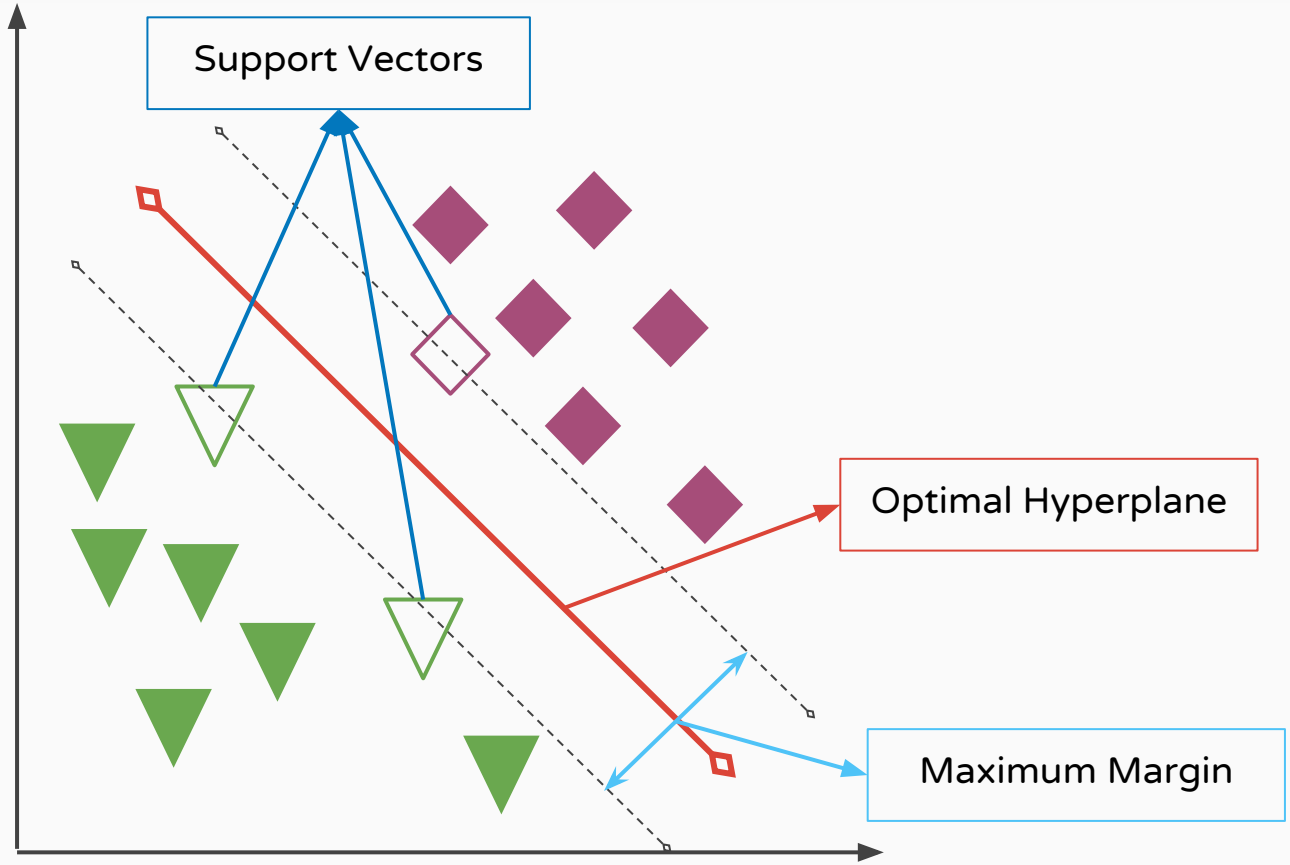
- Evaluate the Model by calculating MSE, MAE and R2 Score

Support Vector Machines

Support Vector Machines

- Support Vector Machine, abbreviated as SVM can be used for both regression and classification tasks. But, it is widely used in classification objectives.
- The objective of the support vector machine algorithm is to find a hyperplane in an N-dimensional space(N — the number of features) that distinctly classifies the data points.
- To separate the two classes of data points, there are many possible hyperplanes that could be chosen. Our objective is to find a plane that has the maximum margin, i.e the maximum distance between data points of both classes. Maximizing the margin distance provides some reinforcement so that future data points can be classified with more confidence.
- Support vector machines attempt to pass a linearly separable hyperplane through a dataset in order to classify the data into two or more groups. This hyperplane is a linear separator for any dimension; it could be a line (2D), plane (3D), and hyperplane (4D+).

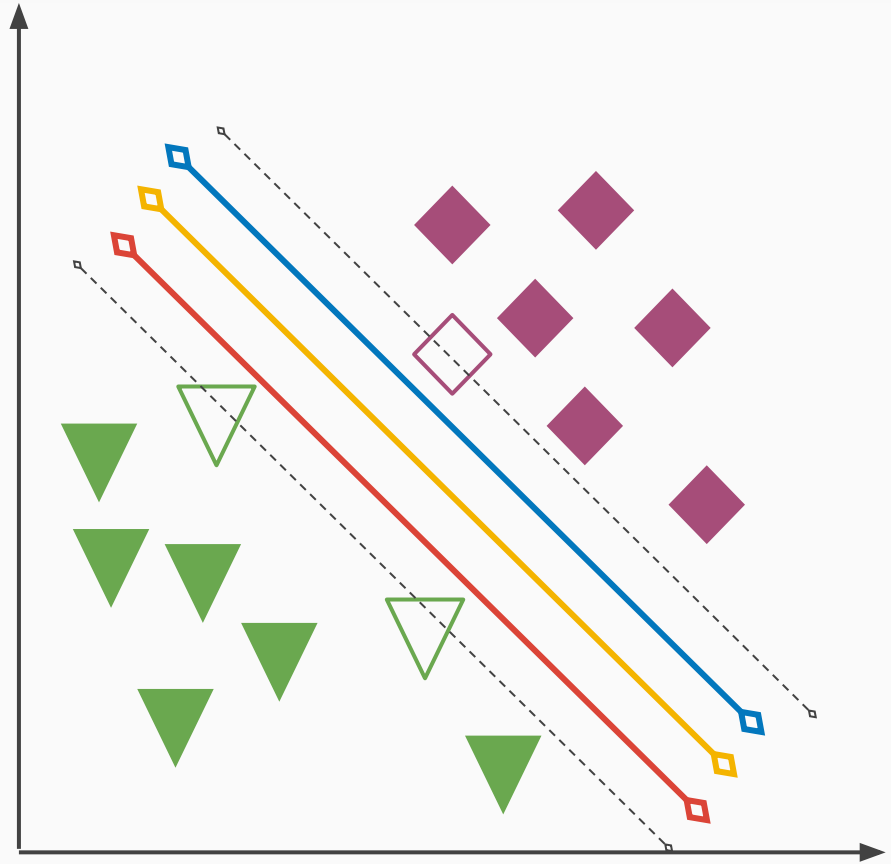
Support Vector Machines



Support Vector Machines

- Hyperplane: It is basically a generalization of plane. 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.
  - In one dimension, an hyperplane is called a point.
  - In two dimensions, it is a line.
  - In three dimensions, it is a plane.
  - In more dimensions we can call it an hyperplane.
- Margin: It is the distance between the hyperplane and the closest data point. If we double it, it would be equal to the margin.
- Support Vectors: 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.

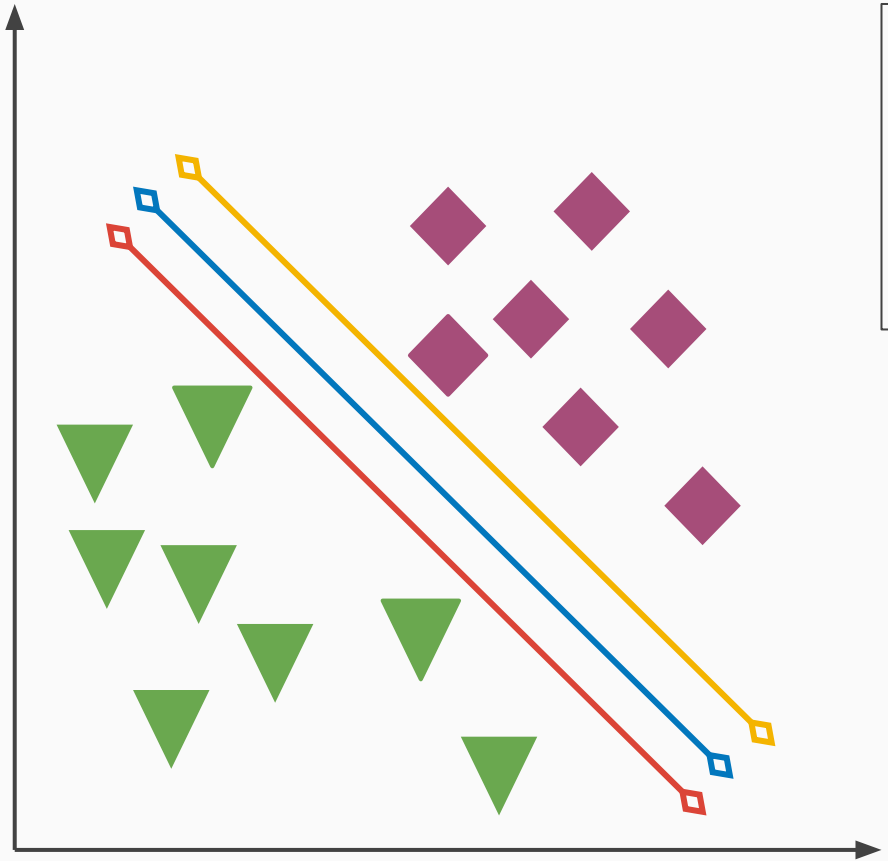
Support Vector Machines



Support Vector Machines

- Consider this scenario, in this to classify or separate data points we can draw many hyperplanes.
- We can separate the red, blue and yellow objects with an infinite number of hyperplanes. Which hyperplane is the best? Well, the best hyperplane is the one that maximizes the margin. The margin is the distance between the hyperplane and a few close points. These close points are the support vectors because they control the hyperplane.
- This is the Maximum Margin Classifier. It maximizes the margin of the hyperplane. This is the best hyperplane because it reduces the generalization error the most. If we add new data, the Maximum Margin Classifier is the best hyperplane to correctly classify the new data.

Support Vector Machines

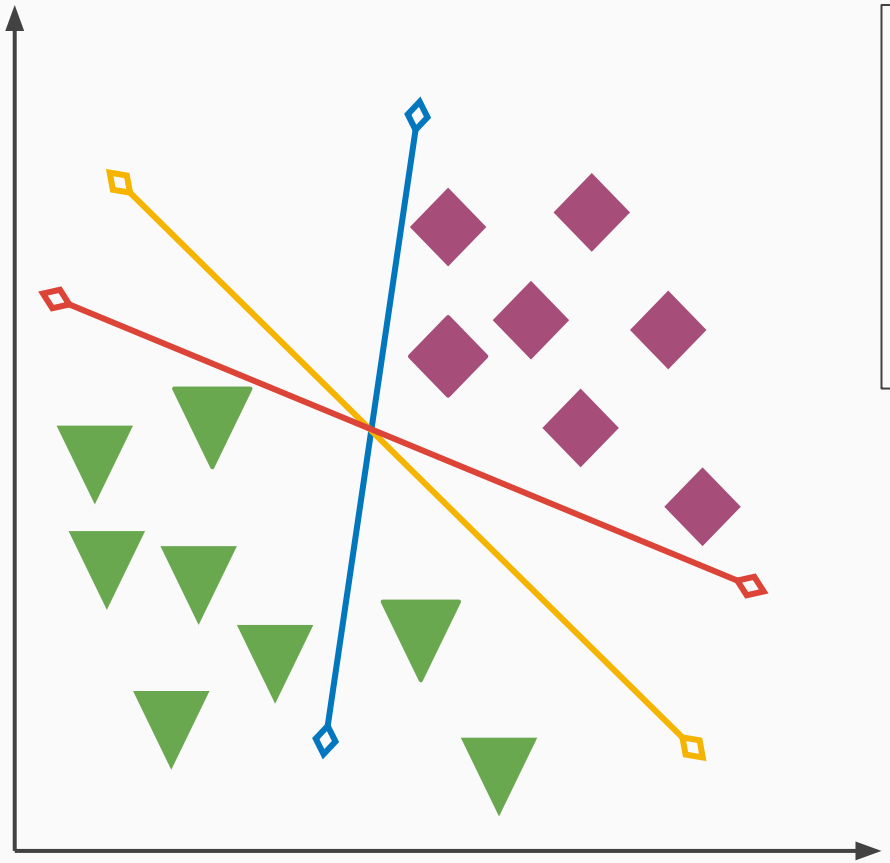


Scenario 1:

Select Blue Line:

As Blue line maximizes the distances between support vectors and hyperplane.

Support Vector Machines

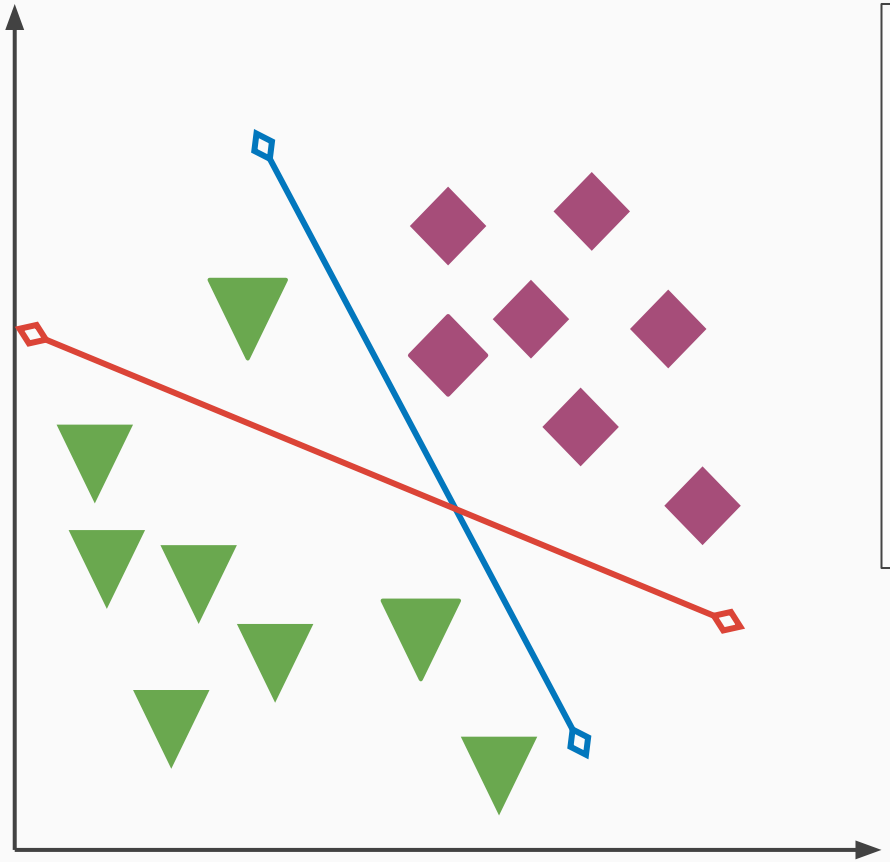


Scenario 2:

Select Yellow Line:

As Yellow line maximizes the distances between support vectors and hyperplane as well as segregates the two classes better.

Support Vector Machines

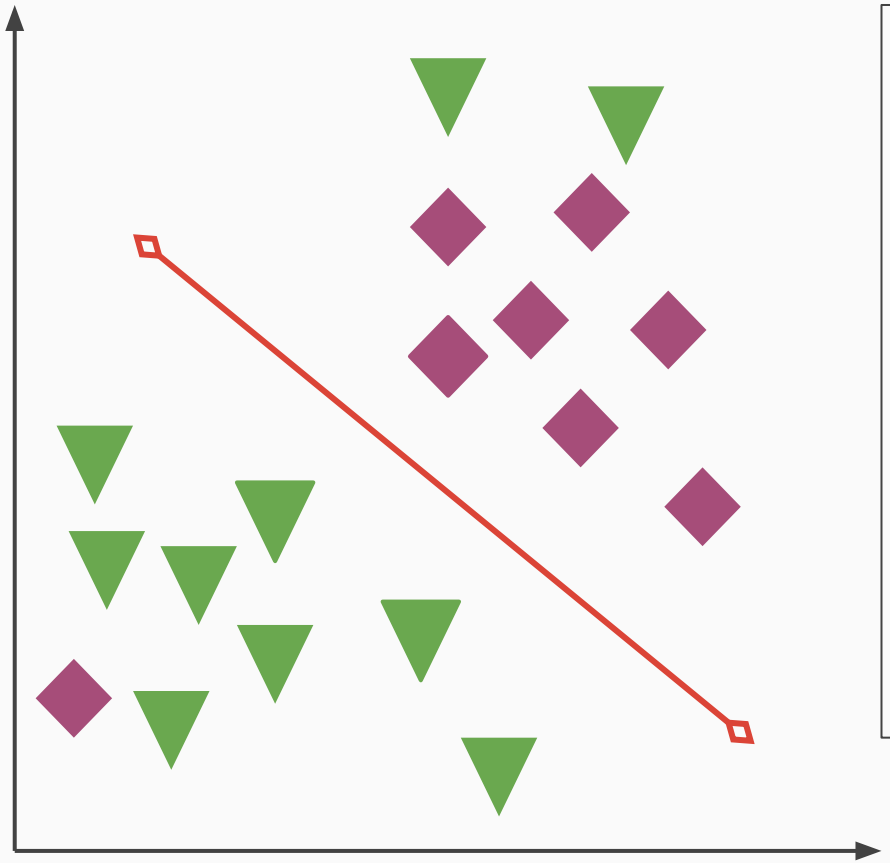


Scenario 3:

Select Blue Line:

Although Red line maximizes the distances between support vectors and hyperplane but does not segregate the two classes better. SVM selects the hyperplane which classifies accurately prior to maximizing margin.

Support Vector Machines



Scenario 4:

Select Red Line:

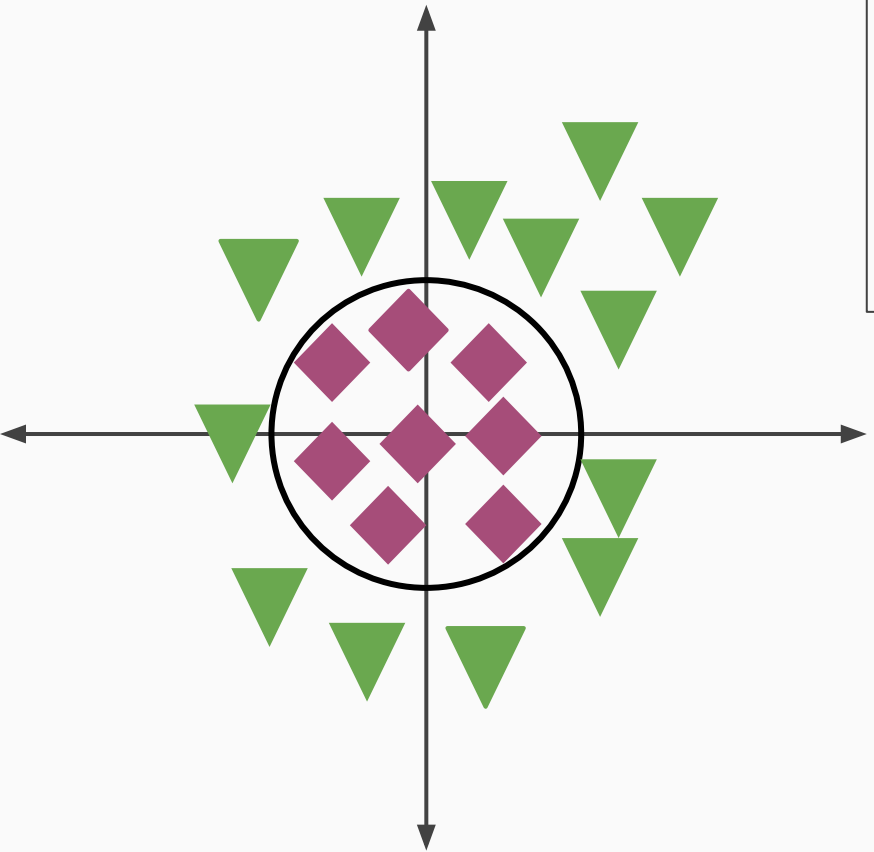
SVM has a feature to ignore Outliers and find best hyperplane that has maximum margin.

In this case, the Maximum Margin Classifier would not work. Vapnik developed a soft margin that would allow for some misclassification of data. This is known as a Soft Margin Classifier.

Support Vector Machines

- The support vector classifier contains a tuning parameter in order to control how much misclassification it will allow. This tuning parameter is important when looking to minimize error. As with all supervised learning, there is a bias-variance tradeoff.
- When the tuning parameter (often denoted as C) is small, the classifier allows only a small bit of misclassification. The support vector classifier will have low bias but may not generalize well and have high variance. We may overfit the training data if our tuning parameter is too small.
- If C is large, the number of misclassifications allowed has been increased. This classifier would generalize better but may have a high amount of bias.
- When the tuning parameter is zero, there can be no misclassification and we have the maximum margin classifier.

Support Vector Machines



Scenario 5:
Non-Linear Data:
Here hyperplane will be 2-D circle. SVM handles this case by using Kernel function.

Support Vector Machines

- The support vector classifier can fail if the data is not linearly separable. SVM uses the kernel trick to separate these data points.
- Kernels are functions that quantify similarities between observations. Simply, these kernels transform our data in order to pass a linear hyperplane and thus classify our data. The kernel function transforms the data into a higher dimensional feature space to make it possible to perform the linear separation.
- Now data points are separated by a simple plane. It is possible by **kernel function**. A nonlinear function is learned by a support vector (linear learning) machine in a high-dimensional feature space.
- Different Kernels:
  - Linear
  - Polynomial
  - Gaussian (RBF → Radial Basis)
  - Sigmoid

Support Vector Machines

Steps to Perform SVM Classification using Scikit-Learn:

- Import required libraries
- Load the Dataset
- Do the Data preprocessing
- Separate features and target
- Feature Scaling using Standard Scalar
- Split the data into train and test part

Support Vector Machines

Steps to Perform SVM Classification using Scikit-Learn:

- Train the Algorithm: Import the SVC class, instantiate it, and call the fit() method along with our training data.

```
from sklearn.svm import SVC
model = SVC()
model.fit(X_train, Y_train)
```

- Make the Predictions: To make predictions, use testing data.

```
y_predicted = model.predict(x_test)
```

- Evaluate the Model by calculating Confusion Matrix.

Support Vector Machines

Steps to Perform SVM Regression using Scikit-Learn:

- Import required libraries
- Load the Dataset
- Check the relationship between Independent and Dependent Variables
- Do the Data preprocessing
- Separate features and target
- Feature Scaling using Standard Scalar
- Split the data into train and test part

Support Vector Machines

Steps to Perform SVM Regression using Scikit-Learn:

- Train the Algorithm: Import the SVR class, instantiate it, and call the fit() method along with our training data.

```
from sklearn.svm import SVR
model = SVR()
model.fit(X_train, Y_train)
```

- Make the Predictions: To make predictions, use testing data.

```
y_predicted = model.predict(x_test)
```

- Evaluate the Model by calculating MSE, MAE and R2 Score

Thanks!

Signitive Technologies

Sneh Nagar, Behind ICICI Bank,
Chhatrapati Square, Nagpur 15

Landmark:
Bharat Petrol Pump
Chhatrapati Square

Contact: 9011033776
www.signitivetech.com

“Keep Learning, Happy Learning”

Best Luck!

Have a Happy Future