XGBoost (Extreme Gradient Boosting)

It is a supervised learning algorithm. Extreme Gradient Boosting (xgboost) is similar to gradient boosting framework but more efficient. So, what makes it fast is its capacity to do parallel computation on a single machine. This makes xgboost at least **10 times faster** than existing gradient boosting implementations or any algorithm.

Since it is very high in predictive power but relatively slow with implementation, "xgboost" becomes an ideal fit for many competitions. It also has additional features for doing cross validation and finding important variables. There are many parameters which needs to be controlled to optimize the model.

Like any other ML algorithm, this too has its own pros and cons. But fortunately pros easily outweigh cons given we have an astute understanding of the algorithm and an intuition for proper parameter tuning.

How to use XGBoost?

There are library implementations of XGBoost in all major data analysis languages. We just have to train the model and tune its parameters. Below, is the series of steps to follow :

- Load your dataset.
- Prepare your data to contain only numeric features (yes, XGBoost works only with numeric features).
- Split the data into train and test sets.
- Train the model and tune the parameters.
- Deploy your model on test data.

Pros:

- Extremely fast (parallel computation).
- Highly efficient.
- Versatile (Can be used for classification, regression or ranking).
- Can be used to extract variable importance.
- Do not require feature engineering (missing values imputation, scaling and normalization)

Cons:

- Only work with numeric features.
- Leads to overfitting if hyper-parameters are not tuned properly.

Naïve Bayes

Naive Bayes is a machine learning algorithm for classification problems. It is based on Bayes' probability theorem. It is primarily used for text classification which involves high dimensional training data sets. A few examples are spam filtration, sentimental analysis, and classifying news articles.

It is not only known for its simplicity, but also for its effectiveness. It is fast to build models and make predictions with Naive Bayes algorithm. Naive Bayes is the first algorithm that should be considered for solving text classification problem.

What is Naive Bayes algorithm?

Naive Bayes algorithm is the algorithm that learns the probability of an object with certain features belonging to a particular group/class. In short, it is a probabilistic classifier. You must be wondering why is it called so?

The Naive Bayes algorithm is called "naive" because it makes the assumption that the occurrence of a certain feature is independent of the occurrence of other features.

For instance, if you are trying to identify a fruit based on its color, shape, and taste, then an orange colored, spherical, and tangy fruit would most likely be an orange. Even if these features depend on each other or on the presence of the other features, all of these properties individually contribute to the probability that this fruit is an orange and that is why it is known as "naive."

As for the "Bayes" part, it refers to the statistician and philosopher, Thomas Bayes and the theorem named after him, Bayes' theorem, which is the base for Naive Bayes Algorithm.

Naive Bayes' Classification

Below is the Naive Bayes' Theorem:

$$P(A \mid B) = P(A) * P(B \mid A) / P(B)$$

P(outcome | evidence) = P(outcome) * P(evidence | outcome) / P(evidence)

The components of the above statement are:-

- P(A|B): Probability (conditional probability) of occurrence of event A given the event B is true
- P(A) and P(B): Probabilities of the occurrence of event A and B respectively
- P(B|A): Probability of the occurrence of event B given the event A is true

The terminology in the Bayesian method of probability (more commonly used) is as follows:

- A is called the proposition and B is called the evidence.
- P(A) is called the prior probability of proposition and P(B) is called the prior probability of evidence.
- P(A|B) is called the posterior.
- P(B|A) is the likelihood.

It is with this formula that the Naive Bayes classifier calculates conditional probabilities for a class outcome given prior information or evidence (our attributes in this case).

Pros and Cons of Naive Bayes algorithm

Pros

- It is a relatively easy algorithm to build and understand.
- It is faster to predict classes using this algorithm than many other classification algorithms.
- It can be easily trained using a small data set.

Cons

- If a given class and a feature have 0 frequency, then the conditional probability estimate for that category will come out as 0. This problem is known as the "Zero Conditional Probability Problem." This is a problem because it wipes out all the information in other probabilities too. There are several sample correction techniques to fix this problem such as "Laplacian Correction."
- Another disadvantage is the very strong assumption of independence class features that it makes. It is near to impossible to find such data sets in real life.

Applications of Naive Bayes Algorithms

Real time Prediction: Naive Bayes is an eager learning classifier and it is sure fast. Thus, it could be used for making predictions in real time.

Multi class Prediction: This algorithm is also well known for multi class prediction feature. Here we can predict the probability of multiple classes of target variable.

Text classification/ Spam Filtering/ Sentiment Analysis: Naive Bayes classifiers mostly used in text classification (due to better result in multi class problems and independence rule) have higher success rate as compared to other algorithms. As a result, it is widely used in Spam filtering (identify spam e-mail) and Sentiment Analysis (in social media analysis, to identify positive and negative customer sentiments)

Recommendation System: Naive Bayes Classifier and Collaborative Filtering together builds a Recommendation System that uses machine learning and data mining techniques to filter unseen information and predict whether a user would like a given resource or not

Recommendations:-

You can build Xgboost Model with MlR package .

So you can tune hyper parameters. And feed the best values (eg. nrounds, max_depth, gamma, min_child_weight, subsample, colsample_bytree ,eta etc) to Model instead of putting different values manually in model.

Eg:-

```
#define Parameters for tuning
xg_ps<- makeParamSet(
  makeIntegerParam("nrounds",lower = 200, upper = 500),
  makeIntegerParam("max_depth",lower = 3,upper = 5),
  makeNumericParam("gamma",lower=0.8,upper= 1),
  makeNumericParam("min_child_weight",lower = 1,upper = 3),
  makeNumericParam("subsample",lower = .5,upper=1),
  makeNumericParam("colsample_bytree",lower = .5 ,upper = 1)
)
```