1. **Prediction**
2. **Linear Regression and Linear Classifier**
   * 1. **Topics**

Multiple linear regression two or more independent variables are used to predict the value of a dependent variable.

Relationship between a scalar dependent variable y and one or more explanatory variables.

* + 1. **Types**
    2. **Assumptions**
    3. **Output Interpretation**
    4. **Tabulation**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **I/P** | **O/P** | **Parametric/**  **Non Parametric** | **Supervised/**  **UnSupervised** | **Black Box** | **Statistical Method** |
| Continuous or dummy for categorical | Continuous | Parametric(Assumptions) | Supervised | No | Least squares, Gaussian, LASSO |

* + 1. **Package, Function & Sample code**

linear = linear\_model.LinearRegression()

# Train the model using the training sets and check score

linear.fit(x\_train, y\_train)

linear.score(x\_train, y\_train)

#Equation coefficient and Intercept

print('Coefficient: \n', linear.coef\_)

print('Intercept: \n', linear.intercept\_)

#Predict Output

predicted= linear.predict(x\_test)

* + 1. **Hyper parameter tuning**
    2. **Performance Metrics**

SSE

1. **Logistic regression**
   * 1. **Topics**

It is a classification not a regression algorithm. It is used to estimate discrete values ( Binary values like 0/1, yes/no, true/false ) based on given set of independent variable(s). In simple words, it predicts the probability of occurrence of an event by fitting data to a logit function. Hence, it is also known as logit regression. Since, it predicts the probability, its output values lies between 0 and 1 (as expected).

* + 1. **Types**

Binomial, Multinomial, Ordinal

* + 1. **Assumptions**
    2. **Output Interpretation**
    3. **Tabulation**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **I/P** | **O/P** | **Parametric/**  **Non Parametric** | **Supervised/**  **UnSupervised** | **Black Box** | **Statistical Method** |
| 1. All are categlorical use logistic  2. Continuous or dummy for categorical | Probability(0 to 1)/Categorical(binary) | parametric | Supervised |  | Linear regression |

* + 1. **Package, Function & Sample code**

from sklearn.linear\_model import LogisticRegression

# Create logistic regression object

model = LogisticRegression()

# Train the model using the training sets and check score

model.fit(X, y)

model.score(X, y)

#Equation coefficient and Intercept

print('Coefficient: \n', model.coef\_)

print('Intercept: \n', model.intercept\_)

#Predict Output

predicted= model.predict(x\_test)

* + 1. **Hyper parameter tuning**
    2. **Performance Metrics**

Types of logistic regression model (Binomial, multinomial, ordinal)

Logistic regression model is evaluated using some of the following:

AIC

Deviance (Null and Residual)

ROC curve

Hosmer Lemeshow test

Psedu R-squared (McFadden R-Squared, Likelihood ration R-squared, Cox and snell R-squared etc)

Lower the value of AIC, better fitted is the model

1. **GradientBoosting**
2. **Topics**
3. **Types**
4. **Assumptions**
5. **Output Interpretation**
6. **Tabulation**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **I/P** | **O/P** | **Parametric/**  **Non Parametric** | **Supervised/**  **UnSupervised** | **Black Box** | **Statistical Method** |
|  |  |  | **Supervised** |  |  |

1. **Package, Function & Sample code**

from sklearn.ensemble import GradientBoostingClassifier #For Classification

from sklearn.ensemble import GradientBoostingRegressor #For Regression

clf = GradientBoostingClassifier(n\_estimators=100, learning\_rate=1.0, max\_depth=1)

clf.fit(X\_train, y\_train)

1. **Hyper parameter tuning**

n\_estimators: It controls the number of weak learners.

learning\_rate:Controls the contribution of weak learners in the final combination. There is a trade-off between learning\_rate and n\_estimators.

max\_depth: maximum depth of the individual regression estimators. The maximum depth limits the number of nodes in the tree. Tune this parameter for best performance; the best value depends on the interaction of the input variables.

1. **Performance Metrics**
2. **AdaBoost**
3. **Topics**
4. **Types**
5. **Assumptions**
6. **Output Interpretation**
7. **Tabulation**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **I/P** | **O/P** | **Parametric/**  **Non Parametric** | **Supervised/**  **UnSupervised** | **Black Box** | **Statistical Method** |
|  |  |  | **Supervised** | **No** |  |

1. **Package, Function & Sample code**

from sklearn.ensemble import AdaBoostClassifier #For Classification

from sklearn.ensemble import AdaBoostRegressor #For Regression

from sklearn.tree import DecisionTreeClassifier

dt = DecisionTreeClassifier()

clf = AdaBoostClassifier(n\_estimators=100, base\_estimator=dt,learning\_rate=1)

#Above I have used decision tree as a base estimator, you can use any ML learner

#as base estimator if it accepts sample weight

clf.fit(x\_train,y\_train)

1. **Hyper parameter tuning**

n\_estimators: It controls the number of weak learners.

learning\_rate:Controls the contribution of weak learners in the final combination. There is a trade-off between learning\_rate and n\_estimators.

base\_estimators: It helps to specify different ML algorithm.

**Note:- You can also tune the parameters of base learners to optimize its performance.**

1. **Performance Metrics**
2. **XGBoost**
3. **Topics**

Another classic gradient boosting algorithm that’s known to be the decisive choice between winning and losing in some Kaggle competitions.

The XGBoost has an immensely high predictive power which makes it the best choice for accuracy in events as it possesses both linear model and the tree learning algorithm, making the algorithm almost 10x faster than existing gradient booster techniques.

1. **Types**

regression, classification and ranking

1. **Assumptions**
2. **Output Interpretation**
3. **Tabulation**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **I/P** | **O/P** | **Parametric/**  **Non Parametric** | **Supervised/**  **UnSupervised** | **Black Box** | **Statistical Method** |
|  |  |  |  |  |  |

1. **Package, Function & Sample code**

from xgboost import XGBClassifier

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import accuracy\_score

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, Y, test\_size=0.33, random\_state=seed)

model = XGBClassifier()

model.fit(X\_train, y\_train)

#Make predictions for test data

y\_pred = model.predict(X\_test)

--------------------------------------------------------------------------

from sklearn.ensemble import GradientBoostingClassifier #For Classification

from sklearn.ensemble import GradientBoostingRegressor #For Regression

clf = GradientBoostingClassifier(n\_estimators=100, learning\_rate=1.0, max\_depth=1)

clf.fit(X\_train, y\_train)

1. **Hyper parameter tuning**

n\_estimators: It controls the number of weak learners.

learning\_rate:Controls the contribution of weak learners in the final combination. There is a trade-off between learning\_rate and n\_estimators.

max\_depth: maximum depth of the individual regression estimators. The maximum depth limits the number of nodes in the tree. Tune this parameter for best performance; the best value depends on the interaction of the input variables.

1. **Performance Metrics**
2. **LightGBM**
3. **Topics**
4. **Types**
5. **Assumptions**
6. **Output Interpretation**
7. **Tabulation**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **I/P** | **O/P** | **Parametric/**  **Non Parametric** | **Supervised/**  **UnSupervised** | **Black Box** | **Statistical Method** |
|  |  |  |  |  |  |

1. **Package, Function & Sample code**

data = np.random.rand(500, 10) # 500 entities, each contains 10 features

label = np.random.randint(2, size=500) # binary target

train\_data = lgb.Dataset(data, label=label)

test\_data = train\_data.create\_valid('test.svm')

param = {'num\_leaves':31, 'num\_trees':100, 'objective':'binary'}

param['metric'] = 'auc'

num\_round = 10

bst = lgb.train(param, train\_data, num\_round, valid\_sets=[test\_data])

bst.save\_model('model.txt')

# 7 entities, each contains 10 features

data = np.random.rand(7, 10)

ypred = bst.predict(data)

1. **Hyper parameter tuning**
2. **Performance Metrics**
3. **Factor Analysis**
4. **Topics**
5. **Types**
6. **Assumptions**
7. **Output Interpretation**
8. **Tabulation**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **I/P** | **O/P** | **Parametric/**  **Non Parametric** | **Supervised/**  **UnSupervised** | **Black Box** | **Statistical Method** |
|  |  |  |  |  |  |

1. **Package, Function & Sample code**
2. **Hyper parameter tuning**
3. **Performance Metrics**
4. **CART**
5. **Topics**
6. **Types**
7. **Assumptions**
8. **Output Interpretation**
9. **Tabulation**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **I/P** | **O/P** | **Parametric/**  **Non Parametric** | **Supervised/**  **UnSupervised** | **Black Box** | **Statistical Method** |
|  |  |  |  |  |  |

1. **Package, Function & Sample code**
2. **Hyper parameter tuning**
3. **Performance Metrics**
4. **CHAID**
5. **Topics**
6. **Types**
7. **Assumptions**
8. **Output Interpretation**
9. **Tabulation**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **I/P** | **O/P** | **Parametric/**  **Non Parametric** | **Supervised/**  **UnSupervised** | **Black Box** | **Statistical Method** |
|  |  |  |  |  |  |

1. **Package, Function & Sample code**
2. **Hyper parameter tuning**
3. **Performance Metrics**
4. **RandomForest**
5. **Topics**

classification and regression

The random forest algorithm is actually a bagging algorithm: also here, we draw random bootstrap samples from our training set. However, in addition to the bootstrap samples, we also draw random subsets of features for training the individual trees; in bagging, we provide each tree with the full set of features. Due to the random feature selection, the trees are more independent of each other compared to regular bagging, which often results in better predictive performance (due to better variance-bias trade-offs), and I'd say that it's also faster than bagging, because each tree learns only from a subset of features.

1. **Types**
2. **Assumptions**
3. **Output Interpretation**
4. **Tabulation**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **I/P** | **O/P** | **Parametric/**  **Non Parametric** | **Supervised/**  **UnSupervised** | **Black Box** | **Statistical Method** |
|  | **discrete variables** | **nonparametric** | **Supervised** |  |  |

1. **Package, Function & Sample code**

from sklearn.ensemble import RandomForestClassifier

# Create Random Forest object

model= RandomForestClassifier()

# Train the model using the training sets and check score

model.fit(X, y)

#Predict Output

predicted= model.predict(x\_test)

1. **Hyper parameter tuning**

**n\_estimators:**

It defines the number of decision trees to be created in a random forest.

Generally, a higher number makes the predictions stronger and more stable, but a very large number can result in higher training time.

**criterion:**

It defines the function that is to be used for splitting.

The function measures the quality of a split for each feature and chooses the best split.

**max\_features :**

It defines the maximum number of features allowed for the split in each decision tree.

Increasing max features usually improve performance but a very high number can decrease the diversity of each tree.

**max\_depth:**

Random forest has multiple decision trees. This parameter defines the maximum depth of the trees.

**min\_samples\_split:**

Used to define the minimum number of samples required in a leaf node before a split is attempted.

If the number of samples is less than the required number, the node is not split.

**min\_samples\_leaf:**

This defines the minimum number of samples required to be at a leaf node.

Smaller leaf size makes the model more prone to capturing noise in train data.

**min\_samples\_leaf:**

This defines the minimum number of samples required to be at a leaf node.

Smaller leaf size makes the model more prone to capturing noise in train data.

**max\_leaf\_nodes:**

This parameter specifies the maximum number of leaf nodes for each tree.

The tree stops splitting when the number of leaf nodes becomes equal to the max leaf node.

**n\_jobs:**

This indicates the number of jobs to run in parallel.

Set value to -1 if you want it to run on all cores in the system.

**random\_state:**

This parameter is used to define the random selection.

It is used for comparison between various models.

1. **Performance Metrics**
2. **Timeseries**
3. **Topics**
4. **Types**
5. **Assumptions**
6. **Output Interpretation**
7. **Tabulation**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **I/P** | **O/P** | **Parametric/**  **Non Parametric** | **Supervised/**  **UnSupervised** | **Black Box** | **Statistical Method** |
|  |  |  |  |  |  |

1. **Package, Function & Sample code**
2. **Hyper parameter tuning**
3. **Performance Metrics**
4. **Adaptive filter**
5. **Topics**
6. **Types**
7. **Assumptions**
8. **Output Interpretation**
9. **Tabulation**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **I/P** | **O/P** | **Parametric/**  **Non Parametric** | **Supervised/**  **UnSupervised** | **Black Box** | **Statistical Method** |
|  |  |  |  |  |  |

1. **Package, Function & Sample code**
2. **Hyper parameter tuning**
3. **Performance Metrics**
4. **Classification**
5. **Decision trees**
6. **Topics**

It is a type of supervised learning algorithm that is mostly used for classification problems. Surprisingly, it works for both categorical and continuous dependent variables. In this algorithm, we split the population into two or more homogeneous sets. This is done based on most significant attributes/ independent variables to make as distinct groups as possible.

1. **Types**

non-linear

* CART (Classification and Regression Trees) → uses Gini Index(Classification) as metric.
* ID3 (Iterative Dichotomiser 3) → uses Entropy function and Information gain as metrics.

1. **Assumptions**
2. **Output Interpretation**
3. **Tabulation**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **I/P** | **O/P** | **Parametric/**  **Non Parametric** | **Supervised/**  **UnSupervised** | **Black Box** | **Statistical Method** |
| categorical and continuous | categorical and continuous | nonparametric | Supervised |  |  |

1. **Package, Function & Sample code**

from sklearn import tree

# Create tree object

model = tree.DecisionTreeClassifier(criterion='gini') # for classification, here you can change the algorithm as gini or entropy (information gain) by default it is gini

# model = tree.DecisionTreeRegressor() for regression

# Train the model using the training sets and check score

model.fit(X, y)

model.score(X, y)

#Predict Output

predicted= model.predict(x\_test)

1. **Hyper parameter tuning**
2. **Performance Metrics**
3. **K-means**
4. **Topics**
5. **Types**
6. **Assumptions**
7. **Output Interpretation**
8. **Tabulation**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **I/P** | **O/P** | **Parametric/**  **Non Parametric** | **Supervised/**  **UnSupervised** | **Black Box** | **Statistical Method** |
|  | **groupings of data** | **non-parametric** | **Unsupervised** |  |  |

1. **Package, Function & Sample code**

from sklearn.cluster import Kmeans

# Create KNeighbors classifier object model

k\_means = KMeans(n\_clusters=3, random\_state=0)

# Train the model using the training sets and check score

model.fit(X)

#Predict Output

predicted= model.predict(x\_test)

1. **Hyper parameter tuning**
2. **Performance Metrics**
3. **Support Vector Method**
4. **Topics**

we plot each data item as a point in n-dimensional space (where n is number of features you have) with the value of each feature being the value of a particular coordinate.

Now, we will find some line that splits the data between the two differently classified groups of data. This will be the line such that the distances from the closest point in each of the two groups will be farthest away.

For example, if we only had two features like Height and Hair length of an individual, we’d first plot these two variables in two dimensional space where each point has two co-ordinates (these co-ordinates are known as Support Vectors)

1. **Types**
2. **Assumptions**
3. **Output Interpretation**
4. **Tabulation**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **I/P** | **O/P** | **Parametric/**  **Non Parametric** | **Supervised/**  **UnSupervised** | **Black Box** | **Statistical Method** |
|  | **discrete variables** | **nonparametric** |  | **Yes** |  |

1. **Package, Function & Sample code**

from sklearn import svm

# Create SVM classification object

model = svm.svc() # there is various option associated with it, this is simple for classification. You can refer link, for mo# re detail.

# Train the model using the training sets and check score

model.fit(X, y)

model.score(X, y)

#Predict Output

predicted= model.predict(x\_test)

1. **Hyper parameter tuning**
2. **Performance Metrics**
3. **DISCRIMINANT Analysis**
4. **Topics**
5. **Types**
6. **Assumptions**

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

That each attribute has the same variance, that values of each variable vary around the mean by the same amount on average.

1. **Output Interpretation**
2. **Tabulation**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **I/P** | **O/P** | **Parametric/**  **Non Parametric** | **Supervised/**  **UnSupervised** | **Black Box** | **Statistical Method** |
| **1.if all are continuous discriminant preferred.**  **2.if 100% categorical then no Discriminant.**  **3. if 1 or 2 are category rest are continuous use discriminant or logistic** | **Multiple Categorical or multiple discrete** | **parametric(Assumptions)** | **Supervised** | **No** | **Gaussian** |

1. **Package, Function & Sample code**
2. **Hyper parameter tuning**
3. **Performance Metrics**
4. **Hierarchical clustering**
5. **Topics**
6. **Types**
7. **Assumptions**
8. **Output Interpretation**
9. **Tabulation**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **I/P** | **O/P** | **Parametric/**  **Non Parametric** | **Supervised/**  **UnSupervised** | **Black Box** | **Statistical Method** |
|  | **groupings of data** | **non-parametric** | **Unsupervised** |  |  |

1. **Package, Function & Sample code**
2. **Hyper parameter tuning**
3. **Performance Metrics**
4. **Apriori**
5. **Topics**
6. **Types**
7. **Assumptions**
8. **Output Interpretation**
9. **Tabulation**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **I/P** | **O/P** | **Parametric/**  **Non Parametric** | **Supervised/**  **UnSupervised** | **Black Box** | **Statistical Method** |
|  |  |  | **Unsupervised** |  |  |

1. **Package, Function & Sample code**
2. **Hyper parameter tuning**
3. **Performance Metrics**
4. **Principal component analysis (PCA)**
5. **Topics**

PCA is used to decompose a multivariate dataset in a set of successive orthogonal components that explain a maximum amount of the variance. In scikit-learn, PCA is implemented as a transformer object that learns n components in its fit method, and can be used on new data to project it on these components.

1. **Types**
2. **Assumptions**
3. **Output Interpretation**
4. **Tabulation**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **I/P** | **O/P** | **Parametric/**  **Non Parametric** | **Supervised/**  **UnSupervised** | **Black Box** | **Statistical Method** |
|  | Dimension reduction |  | UnSupervised |  |  |

1. **Package, Function & Sample code**

from sklearn import decomposition

#Import Library

from sklearn import decomposition

#Assumed you have training and test data set as train and test

# Create PCA obeject pca= decomposition.PCA(n\_components=k) #default value of k =min(n\_sample, n\_features)

# For Factor analysis

#fa= decomposition.FactorAnalysis()

# Reduced the dimension of training dataset using PCA

train\_reduced = pca.fit\_transform(train)

#Reduced the dimension of test dataset

test\_reduced = pca.transform(test)

1. **Hyper parameter tuning**
2. **Performance Metrics**
3. **KNN**
4. **Topics**
5. **Types**
6. **Assumptions**
7. **Output Interpretation**
8. **Tabulation**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **I/P** | **O/P** | **Parametric/**  **Non Parametric** | **Supervised/**  **UnSupervised** | **Black Box** | **Statistical Method** |
|  | **discrete variables** | **non-parametric** | **Supervised** |  |  |

1. **Package, Function & Sample code**

from sklearn.neighbors import KNeighborsClassifier

# Create KNeighbors classifier object model

KNeighborsClassifier(n\_neighbors=6) # default value for n\_neighbors is 5

# Train the model using the training sets and check score

model.fit(X, y)

#Predict Output

predicted= model.predict(x\_test)

1. **Hyper parameter tuning**
2. **Performance Metrics**
3. **Naive Bayes**
4. **Reference URL:-**

http://dataaspirant.com/2017/02/06/naive-bayes-classifier-machine-learning/

https://dzone.com/articles/naive-bayes-tutorial-naive-bayes-classifier-in-pyt

https://machinelearningmastery.com/naive-bayes-for-machine-learning/

1. **Topics**

Naive Bayes is a kind of classifier which uses the Bayes Theorem. It predicts membership probabilities for each class such as the probability that given record or data point belongs to a particular class. The class with the highest probability is considered as the most likely class.

"The posterior probability equals the prior probability times the likelihood ratio."

Bayes' theorem (alternatively Bayes' law or Bayes' rule) describes the probability of an event, based on prior knowledge of conditions that might be related to the event.

**Advantages:-**

Naive Bayes Algorithm is a fast, highly scalable algorithm.

Naive Bayes can be use for Binary and Multiclass classification

Great choice for Text Classification problems. It’s a popular choice for spam email classification.

It can be easily train on small dataset

**Disadvantages:-**

Naive Bayes can learn individual features importance but can’t determine the relationship among features.

1. **Types**
2. Gaussian Naive Bayes
3. MultiNomial Naive Bayes
4. Bernoulli Naive Bayes Gaussian Naive Bayes
5. MultiNomial Naive Bayes
6. Bernoulli Naive Bayes
7. **Assumptions**

Assumption of independence among predictors.

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

1. **Output Interpretation**
2. **Tabulation**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **I/P** | **O/P** | **Parametric/**  **Non Parametric** | **Supervised/**  **UnSupervised** | **Black Box** | **Statistical Method** |
|  |  |  |  |  |  |

1. **Package, Function & Sample code**

from sklearn.naive\_bayes import GaussianNB

classifier = GaussianNB()

classifier.fit(X\_train, y\_train)

1. **Hyper parameter tuning**
2. **Performance Metrics**
3. **Neural networks**
4. **Topics**
5. **Types**
6. **Assumptions**
7. **Output Interpretation**
8. **Tabulation**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **I/P** | **O/P** | **Parametric/**  **Non Parametric** | **Supervised/**  **UnSupervised** | **Black Box** | **Statistical Method** |
|  |  |  |  |  |  |

1. **Package, Function & Sample code**
2. **Hyper parameter tuning**
3. **Performance Metrics**

**Cross Validation Sample**

**Statistical Test**

|  |  |  |
| --- | --- | --- |
| **Sno** | **Statistical test** | **Purpose** |
| 1 | Durban watson test |  |
| 2 | Anderson darling test | Testing Normality,recommended when N is large |
| 3 | Shapiro wilk test | Testing Normality,N is less than 2,000 |
| 4 | Shapiro-Francia | Testing Normality,N is less than 5,000 |
| 5 | Kolmogorov Smirnov test | Testing Normality, recommended when N is large |
| 6 | Jarque-Bera test | Good alternative for normality testing. |
| 7 | Park test | The Park Test is a test for heteroscedasticity. Heteroscedasticity means that the variances of the errors are not the same across a set of independent (predictor) variables. Use the Park test for heteroscedasticity if you have some variable Z that you think might explain the different variances of the residuals. |
| 8 | White test | White’s test is used to test for heteroscedastic (“differently dispersed”) errors in regression analysis. It is a special case of the (simpler) Breusch-Pagan test. |
| 9 | Multiple correspondence analysis |  |
| 10 | Yule and Kendal | The Yule coefficient is used to measure the skewness of a frequencydistribution. |
| 11 | William fellar |  |
| 12 | Naïve Bayes theorm |  |
| 13 | Naïve rule | Predicts the new record as similar to majority class |
| 14 | Levings test |  |
| 15 | Parsimony principle |  |
| 16 | Skewness | skewness of a variable is larger than 0, the variable is skewed to the right with many observations on the left of the distribution |
| 17 | Kurtosis | kurtosis is greater than 3), the distribution has a high peak and flat tails |
| 22 | Correlational | These tests look for an association between variables |
| 23 | Pearson correlation | Tests for the strength of the association between two continuous variables |
| 24 | Spearman correlation | Tests for the strength of the association between two ordinal variables (does not rely on the assumption of normal distributed data) |
| 25 | Chi-square | Tests for the strength of the association between two categorical variables.Chi-square test is used to compare categorical variables |
| 26 | Chi-square-Goodness of fit test | Goodness of fit test, which determines if a sample matches the population |
| 27 | Chi-square-independent test | test for two independent variables is used to compare  two variables in a contingency table to check if the data fits. |
| 28 |  |  |
| 29 | Comparison of Means: | look for the difference between the means of variables |
| 30 | Paired T-test | Tests for difference between two related variables |
| 31 | Independent T-test | Tests for difference between two independent variables |
| 32 | ANOVA | Tests the difference between group means after any other variance in the outcome variable is accounted for. used to compare multiple (three or more) samples with a single test |
| 34 | T-Test | A t-test is used to compare the mean of two given samples.  Like a z-test, a t-test also assumes a normal distribution of the sample. A t-test is used when the population parameters (mean and standard deviation) are not known |
| 35 | Z-test | used to validate a hypothesis that the sample drawn belongs to the same population. |
| 36 | Null Hypothesis and Testing | A null hypothesis, proposes that no significant difference exists in a set of given observations |
| 37 | Regression: | assess if change in one variable predicts change in another variable |
| 38 | Simple regression | Tests how change in the predictor variable predicts the level of change in the outcome variable |
| 39 | Multiple regression | Tests how change in the combination of two or more predictor variables predict the level of change in the outcome variable |
| 43 | Non-parametric: | are used when the data does not meet assumptions required for parametric tests |
| 44 | Wilcoxon rank-sum test | Tests for difference between two independent variables - takes into account magnitude and direction of difference |
| 45 | Wilcoxon sign-rank test | Tests for difference between two related variables - takes into account magnitude and direction of difference |
| 46 | Sign test | Tests if two related variables are different – ignores magnitude of change, only takes into account direction |

**Comparison of Algorithm**

URL:- https://www.analyticsvidhya.com/blog/2018/09/an-end-to-end-guide-to-understand-the-math-behind-xgboost/

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No** | **Random Forest** | **GradientBoosting** | **XGBoosting** | **Decision Tree** |
|  | Reduce overfitting | Reduce Bias & variance |  | High variance |
|  | Bagging or boosting aggregation helps to reduce the variance in any learner. | Reduce bias and variance |  | High Bias, Overfitting |
|  |  | large number of trees might lead to overfitting | XGBoost has an option to penalize complex models through both L1 and L2 regularization |  |
|  |  |  | XGBoost incorporates a sparsity-aware split finding algorithm to handle different types of sparsity patterns in the data |  |
|  |  |  | For faster computing, XGBoost can make use of multiple cores on the CPU |  |
|  |  |  |  |  |

**Over-fitting and Regularization**

**URL:-**

https://towardsdatascience.com/over-fitting-and-regularization-64d16100f45c

https://towardsdatascience.com/l1-and-l2-regularization-methods-ce25e7fc831c

**Overfitting** happens when model learns signal as well as noise in the training data and wouldn’t perform well on new data on which model wasn’t trained on.

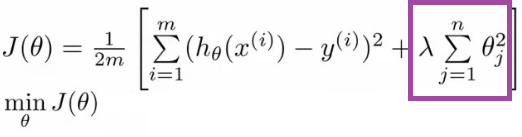
1. Regularization technique
2. Cross Validation
3. Drop out

**Avoid overfitting** your model on training data like cross-validation sampling, reducing number of features, pruning, regularization etc.

**Regularization** basically adds the penalty as model complexity increases. Regularization parameter (lambda) penalizes all the parameters except intercept so that model generalizes the data and won’t overfit.

**Regularization** is a technique to discourage the complexity of the model. It does this by penalizing the loss function. This helps to solve the overfitting problem. Loss function is the sum of squared difference between the actual value and the predicted value.

**Regularization** will add the penalty for higher terms. This will decrease the importance given to higher terms and will bring the model towards less complex equation.



**L1 & L2 Regularization**

A regression model that uses **L1 regularization technique is called Lasso Regression** and

model which uses **L2 is called Ridge Regression.**

**L2 Ridge regression** adds “squared magnitude” of coefficient as penalty term

**L1 Lasso Regression** (Least Absolute Shrinkage and Selection Operator) adds “absolute value of magnitude” of coefficient as penalty term.

The key difference between these techniques is that Lasso shrinks the less important feature’s coefficient to zero thus, removing some feature altogether. So, this works well for feature selection in case we have a huge number of features.

**Comparison:-**

URL:- https://medium.com/datadriveninvestor/l1-l2-regularization-7f1b4fe948f2

|  |  |  |
| --- | --- | --- |
| **Sno.** | **L1 Lasso** | **L2 Ridge** |
| 1 | L1 penalizes sum of absolute value of weights. | L2 regularization penalizes sum of square weights. |
| 2 | L1 has a sparse solution | L2 has a non sparse solution |
| 3 | L1 has multiple solutions | L2 has one solution |
| 4 | L1 has built in feature selection | L2 has no feature selection |
| 5 | L1 is robust to outliers | L2 is not robust to outliers |
| 6 | L1 generates model that are simple and interpretable but cannot learn complex patterns | L2 gives better prediction when output variable is a function of all input features |
| 6 |  | L2 regularization is able to learn complex data patterns |

**Entropy**

URL:-

https://medium.com/deep-math-machine-learning-ai/chapter-4-decision-trees-algorithms-b93975f7a1f1

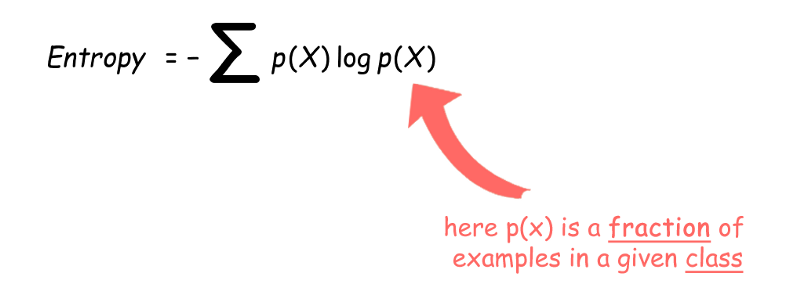
https://medium.com/coinmonks/what-is-entropy-and-why-information-gain-is-matter-4e85d46d2f01

Entropy refers to disorder or uncertainty.

**Definition:** Entropy is the measures of impurity, disorder or uncertainty in a bunch of examples.

**What an Entropy basically does?**

Entropy controls how a Decision Tree decides to split the data. It actually effects how a Decision Tree draws its boundaries.



**What is Information gain and why it is matter in Decision Tree?**

Definition: Information gain (IG) measures how much “information” a feature gives us about the class.

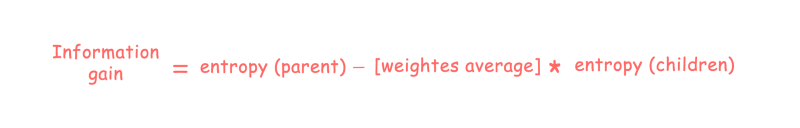
**Why it matter ?**

Information gain is the main key that is used by Decision Tree Algorithms to construct a Decision Tree.

Decision Trees algorithm will always tries to maximize Information gain.

An attribute with highest Information gain will tested/split first.

The Equation of Information gain:



1. CART (Classification and Regression Trees) → uses Gini Index(Classification) as metric.
2. ID3 (Iterative Dichotomiser 3) → uses Entropy function and Information gain as metrics.

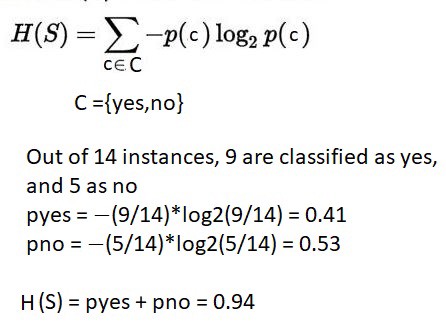
In Decision tree algorithm ID3, entropy characterizes the dataset. The attribute with smaller entropy is used to split the dataset is used to split the dataset on the iteration.

For a binary classification problem

1. If all examples are positive or all are negative then entropy will be zero i.e, low.
2. If half of the examples are of positive class and half are of negative class then entropy is one i.e., high.

Information gain is the measure of the difference in entropy from before to after the dataset is split on an attribute. In other words how many uncertainty was reduced after splitting dataset on attribute A.

**E.G for calculating the entropy interview question**



-(9/14) \* log2(9/14=> -0.642 \* - 0.637 = 0.41

**GINI**

**Classification with using the CART algorithm.**

CART we use Gini index as a metric.

1.compute the gini index for data-set

2.for every attribute/feature:

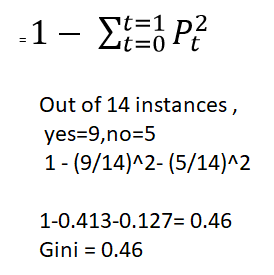
1.calculate gini index for all categorical values

2.take average information entropy for the current attribute

3.calculate the gini gain

3. pick the best gini gain attribute.

4. Repeat until we get the tree we desired.



**Maximum value of Gini Index could be 1** when all target values are equally distributed.

**Minimum value of Gini Index will be 0** when all observations belong to one label.

**Other Concepts**

**URL Reference:-**

https://scikit-learn.org/stable/modules/feature\_extraction.html#the-bag-of-words-representation

https://towardsdatascience.com/beyond-accuracy-precision-and-recall-3da06bea9f6c

https://towardsdatascience.com/data-science-performance-metrics-for-everyone-4d68f4859eef

**FAQ:-**

https://github.com/rasbt/python-machine-learning-book/tree/master/faq

**The Bag of Words representation:-**

Text Analysis is a major application field for machine learning algorithms. However the raw data, a sequence of symbols cannot be fed directly to the algorithms themselves as most of them expect numerical feature vectors with a fixed size rather than the raw text documents with variable length.

In order to address this, scikit-learn provides utilities for the most common ways to extract numerical features from text content, namely:

tokenizing strings and giving an integer id for each possible token, for instance by using white-spaces and punctuation as token separators.

counting the occurrences of tokens in each document.

normalizing and weighting with diminishing importance tokens that occur in the majority of samples / documents.

In this scheme, features and samples are defined as follows:

each individual token occurrence frequency (normalized or not) is treated as a feature.

the vector of all the token frequencies for a given document is considered a multivariate sample.

A corpus of documents can thus be represented by a matrix with one row per document and one column per token (e.g. word) occurring in the corpus.

We call vectorization the general process of turning a collection of text documents into numerical feature vectors. This specific strategy (tokenization, counting and normalization) is called the Bag of Words or “Bag of n-grams” representation. Documents are described by word occurrences while completely ignoring the relative position information of the words in the document.

**Sparsity:-**

As most documents will typically use a very small subset of the words used in the corpus, the resulting matrix will have many feature values that are zeros (typically more than 99% of them).

For instance a collection of 10,000 short text documents (such as emails) will use a vocabulary with a size in the order of 100,000 unique words in total while each document will use 100 to 1000 unique words individually.

In order to be able to store such a matrix in memory but also to speed up algebraic operations matrix / vector, implementations will typically use a sparse representation such as the implementations available in the scipy.sparse package.

**How can I avoid overfitting?**

1. collect more data
2. use ensembling methods that "average" models
3. choose simpler models / penalize complexity

**Remedies against overfitting include**

1.Choose a simpler model by adding bias and/or reducing the number of parameters

2. Adding regularization penalties

3. Reducing the dimensionality of the feature space

4.Collecting more training data

**Parametric Vs Non Parametric**

You "assume" your data follows, and this distribution comes with the finite number of parameters (for example, the mean and standard deviation of a normal distribution); you don't make/have these assumptions in non-parametric models.

**Modelling**

URL:-

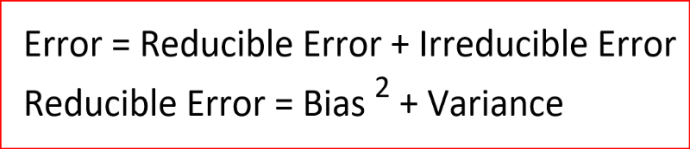
https://medium.com/mlreview/gradient-boosting-from-scratch-1e317ae4587d

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)

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

**Loss Function/Cost Function**

The objective of any supervised learning algorithm is to define a loss function and minimize it. Let’s see how maths work out for Gradient Boosting algorithm.



**Reducible Error** has two components — bias and variance.

Presence of bias or variance causes overfitting or underfitting of data.

**PARAMETRIC/NON-PARAMETRIC**

URL:- https://sebastianraschka.com/faq/docs/parametric\_vs\_nonparametric.html

**"A statistical procedure is of a nonparametric type if it has properties which are satisfied to a reasonable approximation when some assumptions that are at least of a moderately general nature hold.”**

The term parametric is also associated with a specified probability distribution that you “assume” your data follows, and this distribution comes with the finite number of parameters (for example, the mean and standard deviation of a normal distribution); you don’t make/have these assumptions in non-parametric models. So, in intuitive terms, we can think of a non-parametric model as a “distribution” or (quasi) assumption-free model.

**BIAS and VARIANCE:-**

Low Bias Low Variance: Models are accurate and consistent on averages and this what we want to strive for in our model.

**Bias**

URL:- https://medium.com/datadriveninvestor/bias-and-variance-in-machine-learning-51fdd38d1f86

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

Bias is how far are the predicted values from the actual values. If the average predicted values are far off from the actual values then the bias is high.

High bias causes algorithm to miss relevant relationship between input and output variable. **When a model has a high bias then it implies that the model is too simple and does not capture the complexity of data thus underfitting the data.**

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

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

**Variance**

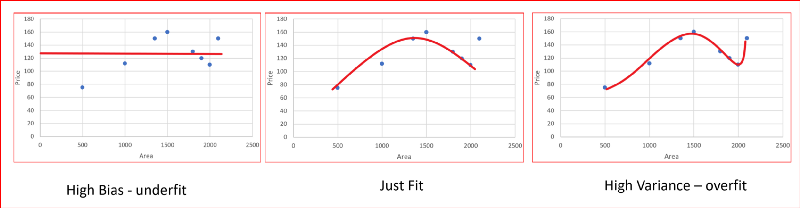
**Variance is the amount that the estimate of the target function will change if different training data was used.**

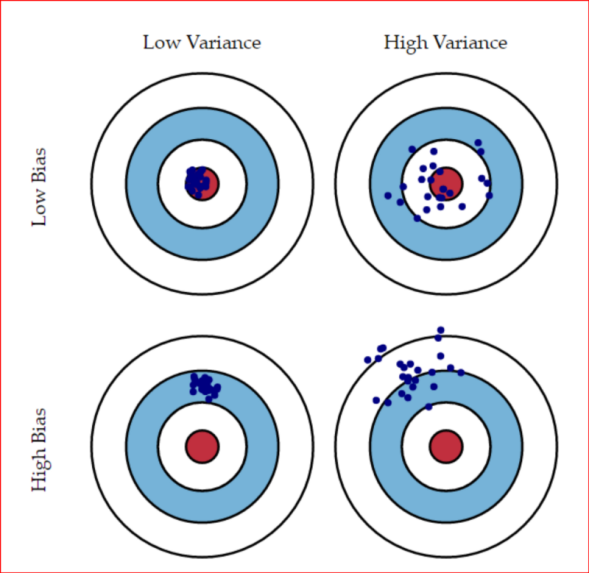
Variance occurs when the model performs good on the trained dataset but does not do well on a dataset that it is not trained on, like a test dataset or validation dataset. Variance tells us how scattered are the predicted value from the actual value.

High variance causes overfitting that implies that the algorithm models random noise present in the training data. when a model has a high variance then the model becomes very flexible and tune itself to the data points of the training set. when a high variance model encounters a different data point that it has not learnt then it cannot make right prediction.

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

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





URL:- https://machinelearningmastery.com/gentle-introduction-to-the-bias-variance-trade-off-in-machine-learning/

|  |  |  |
| --- | --- | --- |
| Sno. | Bias | Variance |
|  | high bias making them fast to learn |  |
|  | easier to understand but generally less flexible | nonparametric machine learning algorithms that have a lot of flexibility have a high variance. For example, decision trees have a high variance, that is even higher if the trees are not pruned before use. |
|  | Low Bias: Suggests less assumptions about the form of the target function | Low Variance: Suggests small changes to the estimate of the target function with changes to the training dataset. |
|  | High-Bias: Suggests more assumptions about the form of the target function. | High Variance: Suggests large changes to the estimate of the target function with changes to the training dataset. |
|  | low-bias machine learning algorithms include: Decision Trees, k-Nearest Neighbors and Support Vector Machines. | low-variance machine learning algorithms include: Linear Regression, Linear Discriminant Analysis and Logistic Regression. |
|  | high-bias machine learning algorithms include: Linear Regression, Linear Discriminant Analysis and Logistic Regression. | high-variance machine learning algorithms include: Decision Trees, k-Nearest Neighbors and Support Vector Machines. |
|  |  |  |
|  |  |  |

**Ensemble**

URL:- https://www.analyticsvidhya.com/blog/2018/06/comprehensive-guide-for-ensemble-models/

An ensemble is just a collection of predictors which come together (e.g. mean of all predictions) to give a final prediction. The reason we use ensembles is that many different predictors trying to predict same target variable will perform a better job than any single predictor alone. Ensembling techniques are further classified into

|  |  |  |
| --- | --- | --- |
| **Sno.** | **Basic Ensemble Techniques** | **Advanced Ensemble Techniques** |
|  | Voting | Stacking |
|  | Averaging | Blending |
|  | Weighted Average | Bagging |
|  |  | Boosting |

|  |  |  |
| --- | --- | --- |
| **Sno.** | **Bagging Techniques** | **Boosting Techniques** |
|  | Bagging meta-estimator | AdaBoost |
|  | Random Forest | GBM |
|  |  | XGB |
|  |  | Light GBM |
|  |  | CatBoost |

**Stacking**

Stacking is an ensemble learning technique that uses predictions from multiple models (for example decision tree, knn or svm) to build a new model. This model is used for making predictions on the test set.

**Blending**

Blending follows the same approach as stacking but uses only a holdout (validation) set from the train set to make predictions. In other words, unlike stacking, the predictions are made on the holdout set only. The holdout set and the predictions are used to build a model which is run on the test set.

**Bagging meta-estimator**

Bagging meta-estimator is an ensembling algorithm that can be used for both classification (BaggingClassifier) and regression (BaggingRegressor) problems. It follows the typical bagging technique to make predictions. Following are the steps for the bagging meta-estimator algorithm:

* Random subsets are created from the original dataset (Bootstrapping).
* The subset of the dataset includes all features.
* A user-specified base estimator is fitted on each of these smaller sets.
* Predictions from each model are combined to get the final result.

**Code Sample**

from sklearn.ensemble import BaggingClassifier

from sklearn import tree

model = BaggingClassifier(tree.DecisionTreeClassifier(random\_state=1))

model.fit(x\_train, y\_train)

model.score(x\_test,y\_test)

0.75135135135135134

Sample code for regression problem:

from sklearn.ensemble import BaggingRegressor

model = BaggingRegressor(tree.DecisionTreeRegressor(random\_state=1))

model.fit(x\_train, y\_train)

model.score(x\_test,y\_test)

**Bootstrap(Sub Sample)**

Bootstrap sampling means drawing random samples from our training set with replacement. E.g., if our training set consists of 7 training samples, our bootstrap samples (here: n=7) can look as follows, where C1, C2, ... Cm shall symbolize the decision tree classifiers.

URL Reference:-

https://medium.com/mlreview/gradient-boosting-from-scratch-1e317ae4587d

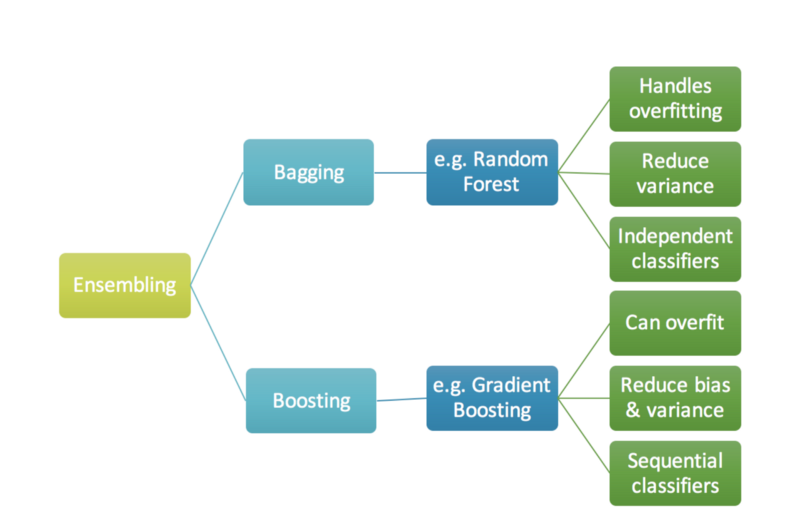
**Bagging**

Bagging classifier would have a lower variance (less overfitting) than an individual decision tree.

Bagging: It is an approach where you take random samples of data, build learning algorithms and take simple means to find bagging probabilities.

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 sub-sample/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. **Example of bagging ensemble is Random Forest models.**



**Boosting**

**URL:-**

https://nbviewer.jupyter.org/github/groverpr/Machine-Learning/blob/master/notebooks/01\_Gradient\_Boosting\_Scratch.ipynb

**Boosting is an ensemble technique in which the predictors are not made independently, but sequentially.**

**The term ‘Boosting’ refers to a family of algorithms which converts weak learner to strong learners.**

**Boosting: Boosting is similar, however the selection of sample is made more intelligently. We subsequently give more and more weight to hard to classify observations.**

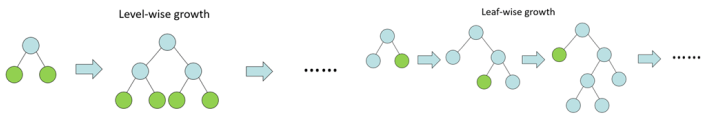
This technique employs the logic in which the subsequent predictors learn from the mistakes of the previous predictors. Therefore, the observations have an unequal probability of appearing in subsequent models and ones with the highest error appear most. (So the observations are not chosen based on the 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. Gradient Boosting is an example of boosting algorithm.

Bagging classifier would have a lower variance (less overfitting) than an individual decision tree.

use very simple classifiers as base classifiers, so-called "weak learners." Picture these weak learners as "decision tree stumps" -- decision trees with only 1 splitting rule. Below, we will refer to the probably most popular example of boosting, AdaBoost. Here, we start with one decision tree stump (1) and "focus" on the samples it got wrong. In the next round, we train another decision tree stump that attempts to get these samples right (2); we achieve this by putting a larger weight on these training samples. Again, this 2nd classifier will likely get some other samples wrong, so we'd re-adjust the weights ...

In a nutshell, we can summarize "Adaboost" as "adaptive" or "incremental" learning from mistakes. Eventually, we will come up with a model that has a lower bias than an individual decision tree (thus, it is less likely to underfit the training data).

The intuition behind gradient boosting algorithm is to leverage the pattern in residuals and strenghten a weak prediction model, until our residuals become randomly (maybe random normal too) distributed. Once we reach a stage that residuals do not have any pattern that can be modeled, we can stop modeling residuals (as it might lead to overfitting).



**IMPURITY METRICS**

* maximize the posterior probabilities (e.g., naive Bayes)
* maximize a fitness function (genetic programming)
* maximize the total reward/value function (reinforcement learning)
* maximize information gain/minimize child node impurities (CART decision tree classification)
* minimize a mean squared error cost (or loss) function (CART, decision tree regression, linear regression, adaptive linear neurons, ...
* maximize log-likelihood or minimize cross-entropy loss (or cost) function
* minimize hinge loss (support vector machine) ...

**Metrics to Evaluate your Machine Learning Algorithm**

URL:- https://towardsdatascience.com/metrics-to-evaluate-your-machine-learning-algorithm-f10ba6e38234

1. **Classification Accuracy**
2. **Logarithmic Loss**
3. **Confusion Matrix**
4. **Area under Curve**
5. **F1 Score**
6. **Mean Absolute Error**
7. **Mean Squared Error**
8. **ROC curve**

**1.Classification Accuracy:-**

https://cdn-images-1.medium.com/max/800/1*yRa2inzTnyASJOre93ep3g.gif

It works well only if there are equal number of samples belonging to each class. For example, consider that there are 98% samples of class A and 2% samples of class B in our training set. Then our model can easily get 98% training accuracy by simply predicting every training sample belonging to class A.

The cost of failing to diagnose the disease of a sick person is much higher than the cost of sending a healthy person to more tests.

**2.Logarithmic Loss:-**

Logarithmic Loss or Log Loss, works by penalising the false classifications. It works well for multi-class classification. When working with Log Loss, the classifier must assign probability to each class for all the samples.

Log Loss nearer to 0 indicates higher accuracy, whereas if the Log Loss is away from 0 then it indicates lower accuracy.

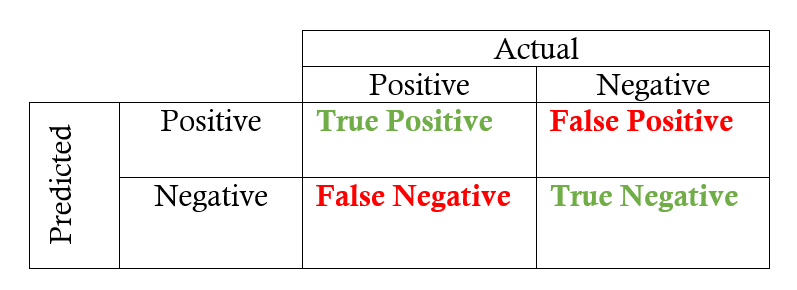
In general, minimising Log Loss gives greater accuracy for the classifier.

Confusion Matrix

Confusion Matrix as the name suggests gives us a matrix as output and describes the complete performance of the model.

**3.Confusion Matrix:-**

Confusion Matrix as the name suggests gives us a matrix as output and describes the complete performance of the model.



* **True positives:**data points labeled as positive that are actually positive
* **False positives:**data points labeled as positive that are actually negative
* **True negatives:**data points labeled as negative that are actually negative
* **False negatives:**data points labeled as negative that are actually positive

**4.Area Under Curve**

Area Under Curve(AUC) is one of the most widely used metrics for evaluation. It is used for binary classification problem. AUC of a classifier is equal to the probability that the classifier will rank a randomly chosen positive example higher than a randomly chosen negative example. Before defining AUC, let us understand two basic terms :

**True Positive Rate/Sensitivity(Recall)**

Recall (also called Sensitivity) and Specificity. For our example, recall would be what fraction did you predict would be hospitalized out of all the hospitalizations.

**False Positive Rate/Specificity**

Specificity would be what fraction did you predict would not be hospitalized out all the people who were not hospitalized.

**Accuracy**

Accuracy what fraction were not hospitalized out of all the people you predicted to be not hospitalized.

**5.F1 Score:-**

we want to find an optimal blend of precision and recall we can combine the two metrics using what is called the F1 score. The F1 score is the harmonic mean of precision and recall taking both metrics into account.

F1 score is single metric that combines recall and precision using the harmonic mean.

**Precision**

Precision in our example, this would be what fraction were actually hospitalized out of all the people you predicted to be hospitalized.

https://cdn-images-1.medium.com/max/800/1*wMIDfGwT9bA6HezvYhdbpQ.gif

**Recall/Sensitivity**

Recall (also called Sensitivity) and Specificity. For our example, recall would be what fraction did you predict would be hospitalized out of all the hospitalizations.

https://cdn-images-1.medium.com/max/800/1*gIlQMZBPjtUHWkwOhRG3KA.gif

**6.Mean Absolute Error:-**

Mean Absolute Error is the average of the difference between the Original Values and the Predicted Values. It gives us the measure of how far the predictions were from the actual output.

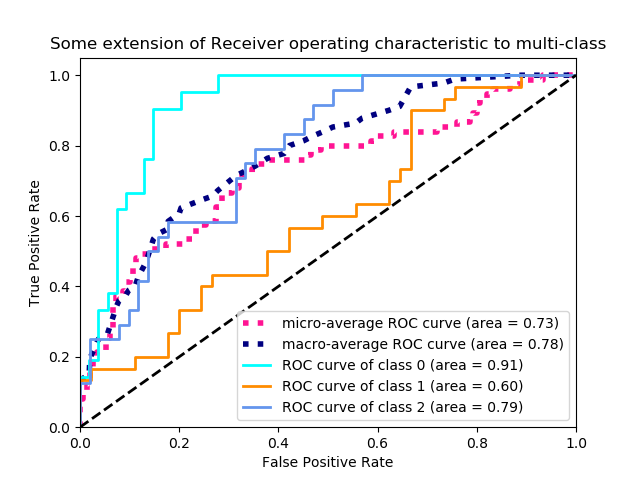
**7.Mean Squared Error:-**

Mean Squared Error(MSE) is quite similar to Mean Absolute Error, the only difference being that MSE takes the average of the square of the difference between the original values and the predicted values. The advantage of MSE being that it is easier to compute the gradient, whereas Mean Absolute Error requires complicated linear programming tools to compute the gradient.

**8.ROC Curve:-**

ROC curves typically feature true positive rate on the Y axis, and false positive rate on the X axis. This means that the top left corner of the plot is the “ideal” point - a false positive rate of zero, and a true positive rate of one. This is not very realistic, but it does mean that a larger area under the curve (AUC) is usually better.

The “steepness” of ROC curves is also important, since it is ideal to maximize the true positive rate while minimizing the false positive rate.



For different values of threshold TPR and FPR are calculated and plotted to find maximum AUC from ROC.

**Bias-Variance Trade-Off:-**

URL:- https://machinelearningmastery.com/gentle-introduction-to-the-bias-variance-trade-off-in-machine-learning/

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

The k-nearest neighbours algorithm has low bias and high variance, but the trade-off can be changed by increasing the value of k which increases the number of neighbours that contribute t the prediction and in turn increases the bias of the model.

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

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

Increasing the bias will decrease the variance.

Increasing the variance will decrease the bias.

**How to handle Imbalanced Classification**

**URL Reference:-**

https://www.analyticsvidhya.com/blog/2017/03/imbalanced-classification-problem/

**Challenges with standard Machine learning techniques**

The conventional model evaluation methods do not accurately measure model performance when faced with imbalanced datasets.

Standard classifier algorithms like Decision Tree and Logistic Regression have a bias towards classes which have number of instances. They tend to only predict the majority class data. The features of the minority class are treated as noise and are often ignored.

while working in an imbalanced domain accuracy is not an appropriate measure to evaluate model performance. For eg: A classifier which achieves an accuracy of 98 % with an event rate of 2 % is not accurate, if it classifies all instances as the majority class. And eliminates the 2 % minority class observations as noise.

**Approach to handling Imbalanced Datasets**

Data Level approach: Resampling Techniques

Random Under-Sampling

Random Over-Sampling

Cluster-Based Over Sampling

Informed Over Sampling: Synthetic Minority Over-sampling Technique

Modified synthetic minority oversampling technique (MSMOTE)

**Algorithmic Ensemble Techniques for Imbalanced dataset**

Bagging Based

Boosting-Based

Adaptive Boosting- Ada Boost

Gradient Tree Boosting

XG Boost

**GBM VS XGBoost(Regularized gradient boosting)**

URL Reference:-

https://xgboost.readthedocs.io/en/latest/tutorials/model.html

**Decision Trees Pruning**

**Cheat Sheet for Python**

|  |  |  |
| --- | --- | --- |
| **Sno.** | **Function** | **Description** |
|  | lower variance | less overfitting |
|  | high variance | will be overfitting |
|  |  |  |