Machine Learning Algorithms Classifiers and Models

Dr. Tassadaq Hussain

Assistant Professor Riphah International University

Collaborations:

Microsoft Research and Barcelona Supercomputing Center

Barcelona, Spain

UCERD Pvt Ltd Islamabad

- Linear Regression
- Logistic Regression
- Decision Tree
- SVM
- Naive Bayes
- kNN
- K-Means
- Random Forest
- Dimensionality Reduction Algorithms
- Gradient Boosting algorithms
    - GBM
    - XGBoost
    - LightGBM
    - CatBoost

UCERD

Linear Regression

- It is used to estimate real values (cost of houses, number of calls, total sales etc.) based on continuous variable(s). Here, we establish relationship between independent and dependent variables by fitting a best line. This best fit line is known as regression line and represented by a linear equation Y= a *X + b.

UCERD

- #Import Library
- #Import other necessary libraries like pandas, numpy...
- from sklearn import linear_model
- #Load Train and Test datasets
- #Identify feature and response variable(s) and values must be numeric and numpy arrays
- x_train=input_variables_values_training_datasets
- y_train=target_variables_values_training_datasets
- x_test=input_variables_values_test_datasets
- # Create linear regression object
- 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)

Logistic Regression

- 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).

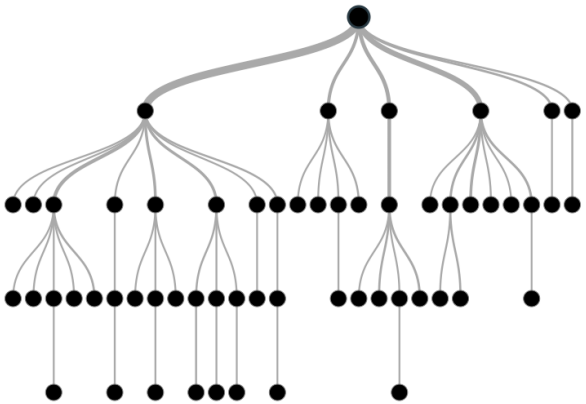
UCERD

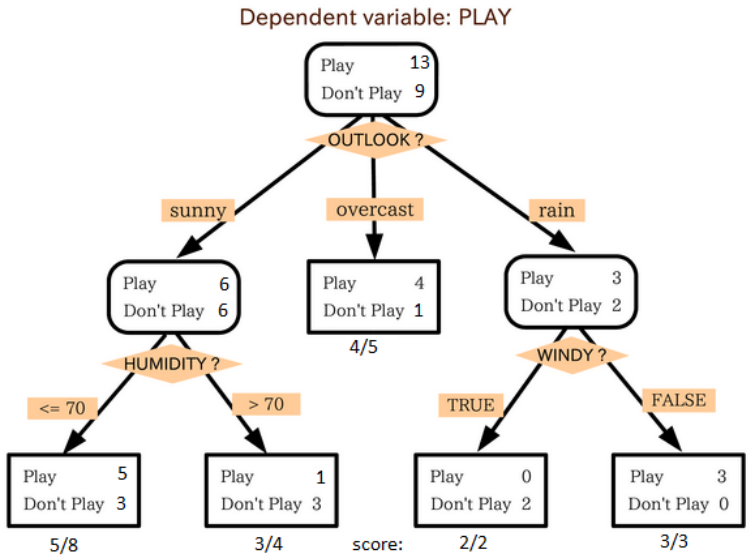
```
#Import Library
from sklearn.linear_model import LogisticRegression
#Assumed you have, X (predictor) and Y (target) for training data set and x_test(predictor) of test_dataset
# Create logistic regression object
model = LogisticRegression()
# Train the model using the training sets and check score
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)
```

UCERD

Decision Tree

- 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.
- Decision Tree 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.





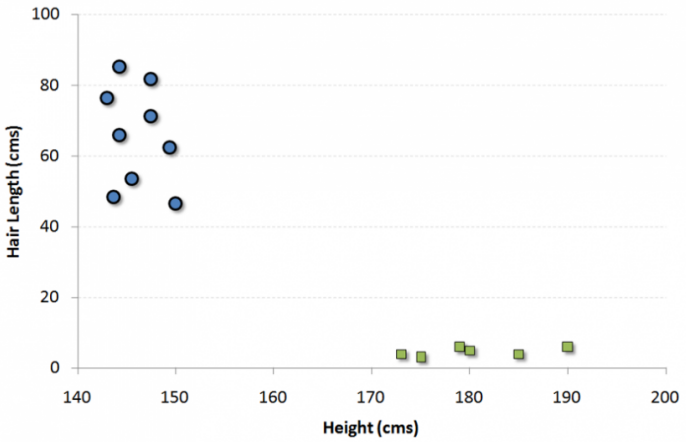
UCERD

```
# Import Library
#I mport other necessary libraries like pandas, numpy...
from sklearn import tree
#Assumed you have, X (predictor) and Y (target) for training data set and
x_test(predictor) of test_dataset
# 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)
```

UCERD

SVM (Support Vector Machine)

It is a classification method. The algorithm is used to 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.



UCERD

- #Import Library
- from sklearn import svm
- #Assumed you have, X (predictor) and Y (target) for training data set and x_test(predictor) of test_dataset
- # 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)

UCERD

Naive Bayes

It is a classification technique based on Bayes' theorem with an assumption of independence between 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. For example, a fruit may be considered to be an apple if it is red, round, and about 3 inches in diameter. Even if these features depend on each other or upon the existence of the other features, a naive Bayes classifier would consider all of these properties to independently contribute to the probability that this fruit is an apple.

UCERD

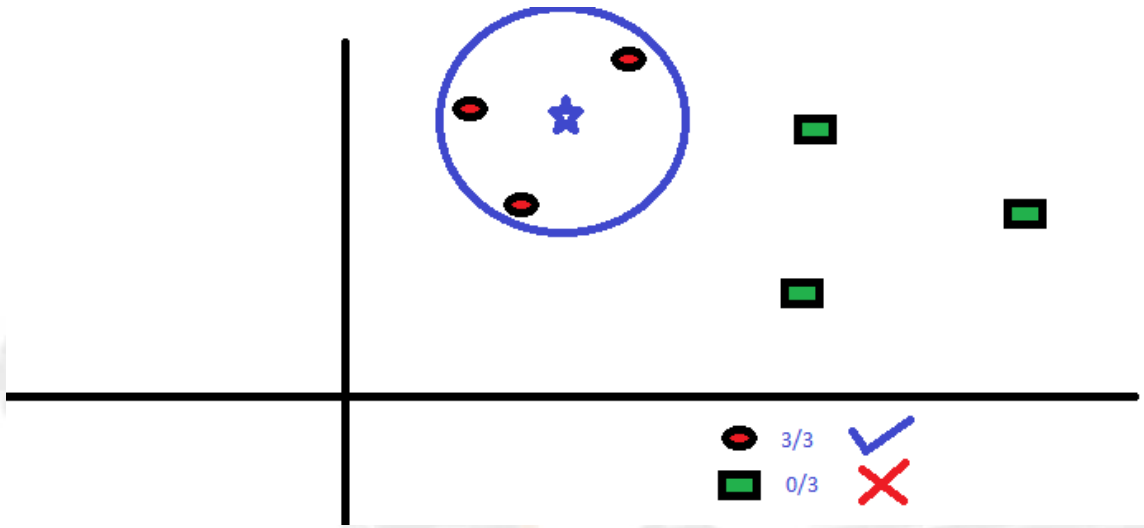
```
#Import Library

from sklearn.naive_bayes import GaussianNB

#Assumed you have, X (predictor) and Y (target) for training data set and x_test(predictor) of test_dataset

# Create SVM classification object model = GaussianNB() # there is other distribution for multinomial classes like Bernoulli Naive Bayes, Refer link

# Train the model using the training sets and check score

model.fit(X, y)

#Predict Output

predicted= model.predict(x_test)
```

UCERD

kNN (k- Nearest Neighbors)

It can be used for both classification and regression problems.

- However, it is more widely used in classification problems in the industry. K nearest neighbors is a simple algorithm that stores all available cases and classifies new cases by a majority vote of its k neighbors. The case being assigned to the class is most common amongst its K nearest neighbors measured by a distance function.
  - KNN is computationally expensive
  - Variables should be normalized else higher range variables can bias it
  - Works on pre-processing stage more before going for kNN like outlier, noise removal



UCERD

```
#Import Library
from sklearn.neighbors import KNeighborsClassifier
#Assumed you have, X (predictor) and Y (target) for training data set and x_test(predictor) of test_dataset
# 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)
```

UCERD

K-Means

- t is a type of unsupervised algorithm which solves the clustering problem. Its procedure follows a simple and easy way to classify a given data set through a certain number of clusters (assume k clusters). Data points inside a cluster are homogeneous and heterogeneous to peer groups.
- How K-means forms cluster:
  - K-means picks k number of points for each cluster known as centroids.
  - Each data point forms a cluster with the closest centroids i.e. k clusters.
  - Finds the centroid of each cluster based on existing cluster members. Here we have new centroids.
  - As we have new centroids, repeat step 2 and 3. Find the closest distance for each data point from new centroids and get associated with new k-clusters. Repeat this process until convergence occurs i.e. centroids does not change.

UCERD

- How to determine value of K:

  In K-means, we have clusters and each cluster has its own centroid. Sum of square of difference between centroid and the data points within a cluster constitutes within sum of square value for that cluster. Also, when the sum of square values for all the clusters are added, it becomes total within sum of square value for the cluster solution.

  We know that as the number of cluster increases, this value keeps on decreasing but if you plot the result you may see that the sum of squared distance decreases sharply up to some value of k, and then much more slowly after that. Here, we can find the optimum number of cluster.

UCERD

```
#Import Library
from sklearn.cluster import KMeans
#Assumed you have, X (attributes) for training data set and
x_test(attributes) of test_dataset
# 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)
```

UCERD

Random Forest

- Random Forest is a trademark term for an ensemble of decision trees. In Random Forest, we've collection of decision trees (so known as "Forest"). To classify a new object based on attributes, each tree gives a classification and we say the tree "votes" for that class. The forest chooses the classification having the most votes (over all the trees in the forest).
- Each tree is planted & grown as follows:
  - If the number of cases in the training set is N, then sample of N cases is taken at random but with replacement. This sample will be the training set for growing the tree.
  - If there are M input variables, a number m<<M is specified such that at each node, m variables are selected at random out of the M and the best split on these m is used to split the node. The value of m is held constant during the forest growing.
  - Each tree is grown to the largest extent possible. There is no pruning.

UCERD

- #Import Library
- from sklearn.ensemble import RandomForestClassifier
- #Assumed you have, X (predictor) and Y (target) for training data set and x_test(predictor) of test_dataset
- # 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)

UCERD

Dimensionality Reduction Algorithms

- Dimension Reduction refers to the process of converting a set of data having vast dimensions into data with lesser dimensions ensuring that it conveys similar information concisely.
- These techniques are typically used while solving machine learning problems to obtain better features for a classification or regression task.

UCERD

Importance

- It helps in data compressing and reducing the storage space required
- It fastens the time required for performing same computations. Less dimensions leads to less computing, also less dimensions can allow usage of algorithms unfit for a large number of dimensions
- It takes care of multi-collinearity that improves the model performance. It removes redundant features. For example: there is no point in storing a value in two different units (meters and inches).
- Reducing the dimensions of data to 2D or 3D may allow us to plot and visualize it precisely. You can then observe patterns more clearly. Below you can see that, how a 3D data is converted into 2D. First it has identified the 2D plane then represented the points on these two new axis z1 and z2.

UCERD

- #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)

UCERD

Gradient Boosting Algorithms

- GBM is a boosting algorithm used when we deal with plenty of data to make a prediction with high prediction power. Boosting is actually an ensemble of learning algorithms which combines the prediction of several base estimators in order to improve robustness over a single estimator. It combines multiple weak or average predictors to a build strong predictor. These boosting algorithms always work well in data science competitions like Kaggle, AV Hackathon, CrowdAnalytix.

UCERD

```
#Import Library

from sklearn.ensemble import GradientBoostingClassifier

#Assumed you have, X (predictor) and Y (target) for training data set and x_test(predictor) of test_dataset

# Create Gradient Boosting Classifier object

model= GradientBoostingClassifier(n_estimators=100, learning_rate=1.0, max_depth=1, random_state=0)

# Train the model using the training sets and check score

model.fit(X, y)

#Predict Output

predicted= model.predict(x_test)
```

UCERD