

# Machine Learning Algorithms



## Types

### Supervised Learning

- Decision Tree
- Random Forest
- kNN
- Logistic Regression

### Unsupervised Learning

- Apriori algorithm
- k-means
- Hierarchical Clustering

### Reinforcement Learning

- Markov Decision Process
- Q Learning



### Python Code

```
#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)
```

### Linear Regression

## Logistic Regression

```
#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)
```

## Decision Tree

```
#Import Library
#Import 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)
```

## SVM (Support Vector Machine)

```
#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 are various options associated
with it, this is simple for classification.
#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)
```

## Naive Bayes

```
#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
#Train the model using the training sets and check
#score
model.fit(X, y)
#Predict Output
predicted= model.predict(x_test)
```

## kNN (k- Nearest Neighbors)

```
#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)
```

## k-Means

```
#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)
```

## Random Forest

```
#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)
```

## Dimensionality Reduction Algorithms

```
#Import Library
from sklearn import decomposition
#Assumed you have training and test data set as train and
#test
#Create PCA object 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)
```

## Gradient Boosting & AdaBoost

```
#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)
```