





Decision Tree · Random Forest Logistic Regression • kNN

Supervised Learning

Unsupervised Learning Reinforcement Learning · Apriori algorithm · k-means

- · Hierarchical Clustering

Markov Decision Process

- Q Learning

#Import other necessary libraries like pandas,

Python

Code Code

from sklearn import linear_model

#numpy...

#Import Library

#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)

#Import Library from sklearn.linear_model import LogisticRegression | #Assumed you have, X (predictor) and Y (target)

#Identify feature and response variable(s) and #values must be numeric and numpy arrays

x_train <- input_variables_values_training_datasets</pre> y_train <- target_variables_values_training_datasets

#Load Train and Test datasets

x_test <- input_variables_values_test_datasets</pre> x <- cbind(x_train,y_train)</pre>

#Train the model using the training sets and #check score

#Train the model using the training sets and check

logistic <- glm(y_train ~ ., data = x,family='binomial')</pre>

linear <- $lm(y_train \sim ., data = x)$ summary(linear)

#Predict Output predicted= predict(linear,x_test)

#of test_dataset #Create logistic regression object

model = LogisticRegression()

#and check score model.fit(X, y)

print('Coefficient: \n', model.coef_)

#Train the model using the training sets

#for training data set and x_test(predictor)

model.score(X, y) #Equation coefficient and Intercept

print('Intercept: \n', model.intercept) #Predict Output predicted= model.predict(x_test)

#Import Library

from sklearn import tree #Assumed you have, X (predictor) and Y (target) for

predicted= predict(logistic,x_test)

summary(logistic)

#Predict Output

x <- cbind(x_train,y_train)</pre>

#score

#Import other necessary libraries like pandas, numpy... library(rpart)

#test_dataset #Create tree object model = tree.DecisionTreeClassifier(criterion='gini')

#training data set and x_test(predictor) of

#for classification, here you can change the #algorithm as gini or entropy (information gain) by

#default it is gini #model = tree.DecisionTreeRegressor() for #regression

#score model.fit(X, y)

#Train the model using the training sets and check

predicted= model.predict(x_test)

model.score(X, y)

#Predict Output

#Import Library

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

#there are various options associated

#Predict Output predicted= predict(fit,x_test)

summary(fit)

#grow tree

#Import Library

x <- cbind(x_train,y_train)</pre>

fit <- rpart(y_train ~ ., data = x,method="class")</pre>

Support Vector Machine) from sklearn import svm #Create SVM classification object

#score

Decision Tree

with it, this is simple for classification. #Train the model using the training sets and check

model = svm.svc()

model.fit(X, y)

#Predict Output

#Import Library

#score

model.fit(X, y)

#Predict Output

model.fit(X, y)

#Predict Output

#Import Library

Naive Bayes

kNN (k- Nearest Neighbors)

model.score(X, y)

predicted= model.predict(x_test)

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

#Train the model using the training sets and check

#training data set and x_test(predictor) of test_dataset

#Train the model using the training sets and check score

#Create KNeighbors classifier object model

KNeighborsClassifier(n_neighbors=6)

#default value for n neighbors is 5

predicted= model.predict(x_test)

#Predict Output predicted= predict(fit,x_test)

summary(fit)

#Import Library

library(e1071)

#Fitting model

x <- cbind(x_train,y_train)</pre>

fit $<-svm(y_train ~ ., data = x)$

#Import Library

x <- cbind(x_train,y_train)</pre> #Fitting model

summary(fit)

#Predict Output

fit <-naiveBayes(y_train ~ ., data = x)</pre>

predicted= predict(fit,x_test)

library(e1071)

#Import Library from sklearn.neighbors import KNeighborsClassifier #Assumed you have, X (predictor) and Y (target) for

like Bernoulli Naive Bayes

predicted= model.predict(x_test)

x <- cbind(x train,y train)</pre> #Fitting model fit $<-knn(y_train \sim ., data = x,k=5)$

summary(fit)

#Predict Output

library(knn)

#Import Library

#Import Library

predicted= predict(fit,x_test)

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)

predicted= model.predict(x_test)

#Predict Output

#Import Library

model.fit(X, y)

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

#Predict Output predicted= model.predict(x_test)

#Import Library from sklearn import decomposition #Assumed you have training and test data set as train and

#default value of k =min(n_sample, n_features) #For Factor analysis

#Create PCA object pca= decomposition.PCA(n_components=k)

train_reduced = pca.fit_transform(train) #Reduced the dimension of test dataset

test_reduced = pca.transform(test)

#Import Library

#Create Gradient Boosting Classifier object model= GradientBoostingClassifier(n_estimators=100, \

library(cluster) fit <- kmeans(X, 3)</pre>

#5 cluster solution

summary(fit) #Predict Output predicted= predict(fit,x_test)

#Import Library

#Fitting model

library(randomForest)

x <- cbind(x_train,y_train)</pre>

fit <- randomForest(Species ~ ., x,ntree=500)</pre>

#Import Library library(stats)

test_reduced <- predict(pca,test)</pre>

pca <- princomp(train, cor = TRUE)</pre>

train_reduced <- predict(pca,train)</pre>

from sklearn.ensemble import GradientBoostingClassifier library(caret) #training data set and x_test(predictor) of test_dataset

learning_rate=1.0, max_depth=1, random_state=0)

To view complete guide on Machine Learning Algorithms, visit here:

nality Reduction Algorithms

Gradient Boosting & AdaBoost

Random Forest

#test

#fa= decomposition.FactorAnalysis() #Reduced the dimension of training dataset using PCA

#Assumed you have, X (predictor) and Y (target) for

#Train the model using the training sets and check score model.fit(X, y)

#Predict Output predicted= model.predict(x_test)

#Import Library

x <- cbind(x train,y train)</pre> #Fitting model

fitControl <- trainControl(method = "repeatedcv",

+ number = 4, repeats = 4) fit <- train(y ~ ., data = x, method = "gbm",

+ trControl = fitControl, verbose = FALSE) predicted= predict(fit,x_test,type= "prob")[,2]

www.analyticsvidhya.com