CHEATSHEET

# MACHINE LEARNING ALGORITHMS

(Python and R Codes)

## TYPES

### Supervised Learning
- Decision Tree
- k-NN
- Random Forest
- Logistic Regression

### Unsupervised Learning
- Apriori algorithm
- K - means
- Hierarchical Clustering

### Reinforcement Learning
- Markov Decision Process
- Q Learning

## LINEAR REGRESSION

python

```
#Import Library
#Import other necessary libraries like pandas, numpy
from sklearn import linear_model
#Load Train and Test datasets
#Identify features and response variable(s) and
#values must be numeric and numpy arrays
x_train = input_variables_training_datasets
y_train = input_variables_test_datasets
#Create linear regression object
linear = linear_model.LinearRegression()
#Train the model using training sets and check score
linear.fit(x_train,y_train)
linear.score(x_train,y_train)
#Equation coefficient and intercept
print('Coefiicient: \n ',linear.coef_)
print('Intercept: \n',linear.intercept_)
#Predict Output
predicted = linear.predict(x_test)
```

R

```
#Load Train and test datasets
#Identify feature and response variable(s) and
#values must be numerics and numpy arrays
x_train <- input_variables_training_datasets
y_train <- input_variables_test_datasets
x <- cbind(x_train,y_train)
#Train the model using training sets and check score
linear <- ln(y_train ~ . , data = x)
summary(linear)
#Predict output
predicted = predict(linear, x_test)
```

## LOGISTIC REGRESSION

python

```
#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('Coefiicient: \n ',model.coef_)
print('Intercept: \n',model.intercept_)
#Predict Output
predicted = model.predict(x_test)
```

R

```
x <- cbind(x_train,y_train)
#Train the model using training sets and check score
logistic <- glm(y_train ~ . , data = x,family = 'binomial)
summary(logistic)
#Predict output
predicted = predict(logistic, x_test)
```

## DECISION TREE

python

```
#Import Library
#Import other necessay 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(criteria = '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)
```

R

```
#Import Library
library(rpart)
x <- cbind(x_train,y_train)
#grow tree
fit <- rpart(y_train ~ . , data = x,family = 'class)
summary(fit)
#Predict output
predicted = predict(fit, 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)
```

```
#Import Library
library(e1071)
x <- cbind(x_train,y_train)
#Fitting model
fit <- svm(y_train ~ . , data = x)
summary(fit)
#Predict output
predicted = predict(fit, 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)
```

```
#Import Library
library(e1071)
x <- cbind(x_train,y_train)
#Fitting model
fit <- naiveBayes(y_train ~ . , data = x)
summary(fit)
#Predict output
predicted = predict(fit, x_test)
```

# K- NN ( K - NEAREST NEIGHBOR )


```
#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)
```

```
#Import Library
library(knn)
x <- cbind(x_train,y_train)
#Fitting model
fit <- knn(y_train ~ . , data = x)
summary(fit)
#Predict output
predicted = predict(fit, x_test)
```

# K - MEANS


```
#Import Library
from sklearn.cluster import KMeans
#Assumed you have, X (attributes)  for training
#dataset  and x_test(attributes) for test_dataset
#for training data set and x_test(predictor) 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
k_means.fit(X,y)
#Predict Output
predicted = k_means.predict(x_test)
```

```
#Import Library
library(cluster)
#Fitting model
fit <- kmeans(x,3)
# 5 cluster silution
```

# RANDOM FOREST

python

```
#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)
```

R

```
#Import Library
library(randomForest)
c <- cbind(x_train, y_train)
#Fitting model
fit <- randomForest(Species ~ . , x , ntreee = 500)
summary(fit)
#Predict Output
predicted = predict(fit, x_test)
```

# DIMENSIONALITY REDUCTION ALGORITHMS

python

```
#Import Library
from sklearn import decomposition
#Assumed you have training and test data set as train
#and test
#Create PCA object
pcs = decomposition.PCS(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 = pcs.fir_transform(train)
#Reduced the dimension of test dataset
test_reduced = pca.transform(test)
```

R

```
#Import Library
library(stats)
pca <- princomp(train, cor = TRUE)
train_redcuted <- predict(pcs, train)
test_reduced <- predict(pca,test)
```

# GRADIENT DESCENT & ADABOOST

python

```
#Import Library
from sklearn import GradientBoostingClassifier
#Assumed you have, X (predictor) and Y (target) for
#training dataset 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)
```

R

```
#Import Library
library(caret)
x <- cbind(x_train, y_train)
#Fitting model
fitControl <- trainControl(method = 'repeatedcv', +
number = 4 , repeats = 4)
fit < - train (y ~. , data = x, method = "gbm", + trControl
= fit Control, verbose = FALSE )
predicted = predict (fit, x_test, type = "prob")[ , 2 ]
```

# FREE COURSES ON MACHINE LEARNING ALGORITHS

All the courses listed below are available on **https://courses.analyticsvidhya.com**

- INTRODUCTION TO PYTHON
- REGRESSION ANALYSIS
- K - NEAREST NEIGHBORS
- DECISION TREE
- SUPPORT VECTOR MACHINE
- ENSEMBLE LEARNING

**FOR AMAZING INFOGRAPHICS, VISIT**

**www.analyticsvidhya.com** Analytics Vidhya