Cheatsheet - Python & R codes for common Machine Learning Algorithms

BEGINNER BUSINESS ANALYTICS CHEATSHEET INFOGRAPHICS MACHINE LEARNING PYTHON R

Introduction

In his famous book – Think and Grow Rich, Napolean Hill narrates story of Darby, who after digging for a gold vein for a few years walks away from it when he was three feet away from it.

Now, I don't know whether the story is true or false. But, I surely know of a few Data Darby around me. These people understand the purpose of machine learning, its execution and use just a set 2 – 3 algorithms on whatever problem they are working on. They don't update themselves with better algorithms or techniques, because they are too tough or they are time consuming.

Like Darby, they are surely missing from a lot of action after reaching this close! In the end, they give up on machine learning by saying it is very computation heavy or it is very difficult or I can't improve my models above a threshold – what's the point? Have you heard them?

Today's cheat sheet aims to change a few Data Darby's to machine learning advocates. Here's a collection of 10 most commonly used machine learning algorithms with their codes in Python and R. Considering the rising usage of machine learning in building models, this cheat sheet is good to act as a code guide to help you bring these machine learning algorithms to use. Good Luck!

For the super lazy Data Darbies, we will make your life even easier. You can download the PDF Version of the cheat sheet here and copy paste the codes from it directly.

CHEATSHEET

Machine Learning Algorithms

(Python and R Codes)

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	R Code

Linear Regression

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)
```

R Code

```
#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
x <- cbind(x_train,y_train)
#Train the model using the training sets and
#check score
linear <- lm(y_train ~ ., data = x)
summary(linear)
#Predict Output
predicted= predict(linear,x_test)
```

Logistic Regression

Python Code

```
#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)
```

R Code

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

```python
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

```python
#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)
```

```r
#Import Library
library(rpart)
x <- cbind(x_train,y_train)
#grow tree
fit <- rpart(y_train ~ ., data = x,method="class")
summary(fit)
#Predict Output
predicted= predict(fit,x_test)
```

SVM (Support Vector Machine)

```python
#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)
```

```r
#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

```python
#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)
```

```r
#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)
```

```python
#Import Library
```

```r
#Import Library
```

kNN (k- Nearest Neighb

```python
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)
```

```r
library(knn)
x <- cbind(x_train,y_train)
#Fitting model
fit <-knn(y_train ~ ., data = x,k=5)
summary(fit)
#Predict Output
predicted= predict(fit,x_test)
```

k-Means

```python
#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)
```

```r
#Import Library
library(cluster)
fit <- kmeans(X, 3)
#5 cluster solution
```

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)
x <- cbind(x_train,y_train)
#Fitting model
fit <- randomForest(Species ~ ., x,ntree=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 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)
```

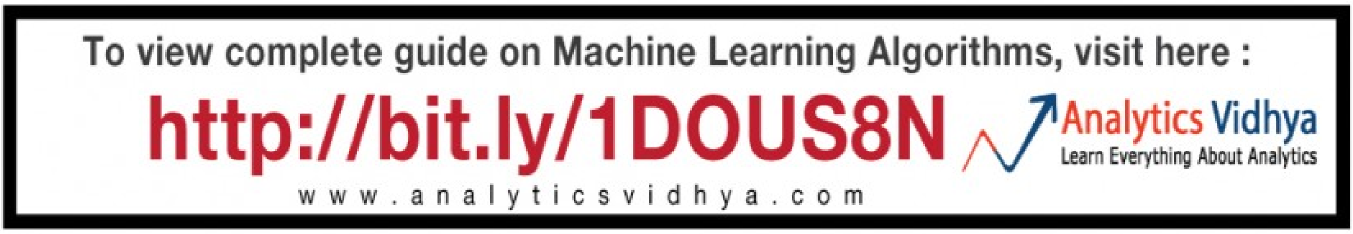
```r
#Import Library
library(stats)
pca <- princomp(train, cor = TRUE)
train_reduced  <- predict(pca,train)
test_reduced  <- predict(pca,test)
```

ient Boosting & AdaBoost

```python
#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
```

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

```
predicted= model.predict(x_test)
```



Keep this cheat sheet handy when you work on data sets. download the complete cheat sheet here: [PDF Version]

If you like what you just read & want to continue your analytics learning, [subscribe to our emails], [follow us on twitter] or join our [Facebook Group]

Article Url - https://www.analyticsvidhya.com/blog/2015/09/full-cheatsheet-machine-learning-algorithms/

Analytics Vidhya

This is the official account of the Analytics Vidhya team.