Machine Learning Cheat Sheet - Python

Regression

```
X = df["features columns"]
y = df["label column"]
import statsmodels.formula.api as sm
model = sm.ols(formula="y ~ X1,X2", data=df)
fitted = model.fit()
fitted.summary2()
from sklearn.linear_model import LinearRegression
lr = LinearRegression()
lr.fit(X, y)
predictions = lr.predict(X)
```

Logistic Regression

```
from statsmodels.formula.api import ols
model = sm.ols(formula="Bought ~ Age", data=sales)
fitted = model.fit()
fitted.summary()
Other Option
from sklearn.linear_model import LogisticRegression
logistic = LogisticRegression()
logistic.fit(X,y)
predictions=logistic.predict(X_test)
```

Decision Trees

```
from sklearn import tree
clf = tree.DecisionTreeClassifier(
      criterion = "entropy", #splitting criterion
      splitter = "best", #split strategy
      max_depth = 5, #maximum depth of the tree
      min_samples_split = 10, #min samples to split node
      min_samples_leaf = 5, #min samples at leaf node
      max_leaf_nodes = 5, #max num of leaf node in a tree
      min_impurity_decrease:0.10) #split impurity threshold
clf.fit(x,y)
predictions=clf.predict(X_test)
```

Plotting the Trees

```
install pydot: conda install -c anaconda pydot
Also Install graphviz in the system
from IPython.display import Image
from sklearn.externals.six import StringIO
import pydot
dot_data = StringIO()
tree.export_graphviz(clf,out_file = dot_data,feature_names
= features,filled=True, rounded=True,impurity=False)
graph = pydot.graph_from_dot_data(dot_data.getvalue())
Image(graph.create_png())
```


GBM

```
from sklearn.ensemble import GradientBoostingClassifier
clf=GradientBoostingClassifier(
    loss ="exponential", #for AdaBoost : "exponential"
    learning_rate = 0.1, #shrinkage
    n_estimators = 100, #boosting stages to perform
    max_depth = 4, #number of nodes in tree *Important*
    criterion = "friedman_mse",#function: quality of a split
    min_samples_split : 3, #min samples for each split
    max_features =  "sqrt",#max feature in tree, sqrt of total
    verbose = 1, #to print progress, 0: don't print)
clf.fit(X,y)
y_pred = clf.predict(X_test)
accuracy = clf.score(X_test, y_test)
```

Model Validation Metrics

```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y, y_pred)
total1=sum(sum(cm))
from confusion matrix calculate accuracy
accuracy=(cm [0,0]+cm [1,1])/total1
Sensitivity=cm[0,0]/(cm[0,0]+cm[0,1])
specificity=cm[1,1]/(cm[1,0]+cm[1,1])
```

ROC Curve and AUC

```
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt
false_positive_rate, true_positive_rate, thresholds = roc_curve(y,
y_pred)
plt.title(Receiver Operating Characteristic)
plt.plot(false_positive_rate, true_positive_rate)
plt.plot([0,1],[0,1],r--)
plt.ylabel(True Positive Rate(Sensitivity))
plt.xlabel(False Positive Rate(Specificity))
plt.show()
```

Area Under Curve - AUC

```
roc_auc = auc(false_positive_rate, true_positive_rate)
roc_auc
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size =
0.2)
```

K-fold Cross Validation

```
from sklearn import cross_validation
kfold = cross_validation.KFold(len(X), n_folds=10)
score = cross_validation.cross_val_score(clf, X, y, cv=kfold)
Mean Kfold accuracy
score.mean()
```

SVM

```
from sklearn import svm
clf = svm.SVC(kernel="linear")
model =clf.fit(X,y)
y_pred = model.predict(X_test)
Accuracy = model.score(X_test, y_test)
```

Neural Network

```
pip install neurolab
import neurolab as nl
import numpy as np
#Defining Network: in the example below
1st argument is a list of min-max values of predictor variables: 2
input with range [0,1] each
2nd argument is list of num of nodes in each layer 4 noded hidden
layer and 1 noded outlayer
#Transf: list of transfer function applied in each layer in order
net = nl.net.newff([[0, 1],[0,1]],[4,1],transf=[nl.trans.LogSig()] * 2)
net.trainf = nl.train.train_rprop
#Training Network
error = []
error.append(net.train(X, y, epochs = 100, goal=0.001))
#Simulate Network (predicting)
predicted_values = net.sim(X_test)
```

Random Forest

```
from sklearn.ensemble import RandomForestClassifier
clf=RandomForestClassifier(
      n_estimators =100, #number of trees in forest
      criterion ="entropy", #tree splitting criterion
      max_features =  "sqrt", #max feature in tree, sqrt of total
      max_depth = 4, #number of nodes in the tree *Important*
      min_samples_split : 3,  #min samples for each split
      Bootstrap = True, #if samples are bootstrapped
      class_weight = "balanced") #to handle class imbalance
clf.fit(X,y)
y_pred = clf.predict(X_test)
accuracy = clf.score(X_test, y_test)
```