In [1]:
```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg

%matplotlib notebook
```

PART A

In [2]:
```
#     @staticmethod
#     def information_gain(X, y, thresh):
#         X0,y0,X1,y1 = split(X,y, thresh)
#         def g(y):
#             pyk_2 = 0
#             for i in range(2):
#                 try:
#                     pyeqk = len(y[y == i])/float(len(y))
#                     pyk_2 = pyeqk**2
#                 except ZeroDivisionError:
#                     pyeqk = 0
#             return 1 - pyk_2

#         return len(X0)*g(y0)/float(len(X)) + len(X1)*g(y1)/float(le
n(X))

#     @staticmethod
#     def gini_impurity(X, y, thresh):
#         X0,y0,X1,y1 = split(X,y, thresh)

#         def h(y) :
#             hy = 0
#             for i in range(2) :
#                 try :
#                     pyeqk = len(y[y == i])/float(len(y))
#                     hy = hy - pyeqk*np.log(pyeqk)
#                 except ZeroDivisionError :
#                     pyeqk = 0

#             return hy

#         return len(X0)*h(y0)/float(len(X)) + len(X1)*h(y1)/float(le
n(X))
```

PART C

PART D (Bagged Trees)

In [ ]:
```
# class BaggedTrees(BaseEstimator, ClassifierMixin):
#     def __init__(self, params=None, n=200):

#         if params is None:
#             params = {}
#         self.params = params
#         self.n = n
#         self.decision_trees = [
#             sklearn.tree.DecisionTreeClassifier(random_state=i, **s
elf.params)
#             for i in range(self.n)
#         ]

#     def fit(self, X, y):
#       self.mask = []
#       for tree in self.decision_trees:
#               mask = np.random.randint(0, high = len(X), size= len
(X))
#               Xsampling = X[mask,:]
#               ysampling = y[mask]
#               tree.fit(Xsampling,ysampling)
#               self.mask.append(mask)



#     def predict(self, X):
#       preds = []
#       for tree in self.decision_trees:
#               preds.append(tree.predict(X))
#       return stats.mode(np.array(preds), axis = 0 )[0].reshape(len
(X))
```

PART F (Random Forest)

In [ ]:
```
# class RandomForest(BaggedTrees):
#     def __init__(self, params=None, n=200, m=2):
#       super().__init__(params = params , n = n )
#       self.m = m

#     def fit(self, X,y):
#       self.mask = []
#       self.features = []
#       for tree in self.decision_trees:
#               mask = np.random.randint(0, high = len(X), size= len
(X))
#               features = np.random.choice( X.shape[1], size = self.
m )
#               Xsampling = X[mask,:]
#               Xsampling = Xsampling[:,features]
#               ysampling = y[mask]
#               tree.fit(Xsampling,ysampling)
#               self.mask.append(mask)
#               self.features.append(features)
#     def predict(self,X):
#       preds = []
#       k = 0
#       for tree in self.decision_trees:
#               preds.append(tree.predict(X[:,self.features[k]]))
#       return stats.mode(np.array(preds), axis = 0 )[0].reshape(len
(X))
```

PART H (AdaBoost)

In [ ]:
```
# class BoostedRandomForest(RandomForest):
#     def fit(self, X, y ):
#         self.w = np.ones(X.shape[0]) / X.shape[0]  # Weights on dat
a
#         self.a = np.zeros(self.n)  # Weights on decision trees
#         k = 0
#         self.features = []
#         for tree in self.decision_trees:
#               mask = np.random.randint(0, high = len(X), size= len
(X))
#               features = np.random.choice(X.shape[1], size = self.m
 )
#               Xsampling = X[mask,:]
#               Xsampling = Xsampling[:,features]
#               ysampling = y[mask]
#               tree.fit(Xsampling,ysampling)
#               self.features.append(features)

#               ej = 0
#               for j in range(len(Xsampling)):
#                       ej = checkXY(Xsampling[j,:], ysampling[j], tr
ee )*self.w[j] + ej

#               ej = ej/float(sum(self.w))

#               self.a[k] = 0.5*np.log((1-ej)/float(ej))

#               for i in range(len(Xsampling)):
#                       if checkXY(Xsampling[i,:],ysampling[i], tree)
 > 0.5 :
#                               self.w[i] = self.w[i]*np.exp(self.a
[k])
#                       else :
#                               self.w[i] = self.w[i]*np.exp(-self.a
[k])
#               k = k + 1


#     def predict(self, X):
#         classes = list(set(y))
#         preds_tot = []
#         for i in range(len(X)):
#               preds = []
#               for c in classes :
#                       zj = 0
#                       k = 0
#                       for tree in self.decision_trees:
#                               Xcheck = X[:,self.features[k]]
#                               Xcheck = Xcheck[i,:]
#                               zj = zj + self.a[k]*checkXY(Xcheck,
 c, tree)
#                               k = k + 1
#                       preds.append(zj)
#                       preds_tot.append(classes[np.argmax(preds)])
#               return preds_tot
```

Part J (Results)

In [ ]:
```
#TITANIC :
# accuracy = [kfold1, kfold2, kfold3, avg_training]

# DecisionTree
# [0.5993975903614458, 0.63855421686746983, 0.60240963855421692, 0.61
369315342328834]
# BaggedTrees
# accuracy = [kfold1, kfold2, kfold3, avg_training]
# [0.79518072289156627, 0.76204819277108438, 0.77710843373493976, 0.9
7801099450274853]
# RandomForest
# accuracy = [kfold1, kfold2, kfold3, avg_training]
# [0.5993975903614458, 0.4006024096385542, 0.60240963855421692, 0.580
209895052473781
# Adaboost
# [0.21686746987951808, 0.63855421686746983, 0.21385542168674698, 0.2
4837581209395301]




# SPAM

# DecisionTree
# accuracy = [kfold1, kfold2, kfold3, avg_training]
# [0.5993975903614458, 0.63855421686746983, 0.60240963855421692, 0.61
369315342328834]
# BaggedTrees
# accuracy = [kfold1, kfold2, kfold3, avg_training]
# [0.78915662650602414, 0.76204819277108438, 0.77409638554216864, 0.9
7801099450274853]
# RandomForest
# accuracy = [kfold1, kfold2, kfold3, avg_training]
# [0.5993975903614458, 0.6506024096385542, 0.61144578313253017, 0.628
68565717141422]
# Adaboost
# accuracy = [kfold1, kfold2, kfold3, avg_training]
# [0.5993975903614458, 0.22289156626506024, 0.21987951807228914, 0.24
887556221889059]
```