```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import operator as opt
import inspect as i
import pdb
import pprint as pp

"""Globals
"""
eps = np.finfo(float).eps


class DecisionStump:
  def __init__(self):
    self.pol = 1
    self.feat_idx = None
    self.threshold = None
    self.alpha = None

  def predic(self, X):
    nSamps = X.shape[0]
    X_col = X[:,self.feat_idx]
    preds = np.ones(nSamps)
    if self.pol == 1:
      preds[X_col] < self.threshold = -1
    else:
      preds[X_col] > self.threshold = -1
    return preds

class adaBoost:
  def __init__(self, nClass=3):
    self.nCl = nClass

  def fit(self, X,y):
    nSamps, nFeat = X.shape
    # init w
    w = np.full(nSamps, (1/nSamps))
    self.classes = list()
    for _ in range(self.nCl):
      # greedy search
      classifier = DecisionStump()
      minErr = float('inf')
      for ft_i in range(nFeat):
        X_col = X[:, ft_i]
        thresholds = np.unique(X_col)
        for threshold in thresholds:
          p = 1
          predictions = np.ones(nSamps)
          predictions[X_col < threshold] = -1
          missClasd = X[y != predictions]
          err = sum(missClasd)
          if err > 0.5: # flip err if greater than .5, and sign
            err = 1-err
            p =-1
          if err < minErr:
            minErr = err
            classifier.pol = p
            classifier.threshold = threshold
```

```
            classifier.feat_idx = ft_i
        classifier.alpha  = (0.5)* np.log((1-err)/(err+eps))# calculate alpha
        predictions = classifier.predic(X)# get prediction
        w *= np.exp(-classifier.alpha * y * predictions)# update weights
        w /= np.sum(w)# normalize
        self.classes.append(classifier)
```

calculate alpha, update w and append it t

```
  def predict(self, X):
    classPred = [classifier.alpha * classifier.predict(X) for clf in self.classe
s]
    Yest = np.sum(classPred, axis=0)
    Yest = np.sign(Yest)
    return Yest

def getAcc(gndTruth, Est):
  correct = 0
  for i in range(len(gndTruth)):
    if gndTruth[i] == Est[i]:
      correct += 1
  return correct / float(len(gndTruth)) * 100.0

def getData(XY):
  X = XY.drop(XY.columns[-1], axis=1)
  Y = XY[XY.columns[-1]]
  return X, Y

"""Main
"""
if __name__ == "__main__":

  # import data
  XYtrain = pd.read_csv("./tic-tac-toe_train.csv")
  XYtrain = XYtrain.rename({'x':'p0','x.1':'p1','x.2':'p2','o':'p3','b':'p4', \
    'b.1':'p5','x.3':'p6','o.1':'p7','o.2':'p8'}, axis='columns')

  XYtest = pd.read_csv("./tic-tac-toe_test.csv")
  XYtest = XYtest.rename({'x':'p0','x.1':'p1','x.2':'p2','o':'p3','b':'p4', \
    'b.1':'p5','x.3':'p6','o.1':'p7','o.2':'p8'}, axis='columns')
  Xt, Yt = getData(XYtrain)
  Xtest, Ytest = getData(XYtest)
  Xt = pd.DataFrame(Xt).to_numpy()
  Yt = pd.DataFrame(Yt).to_numpy()
  Yt[Yt=='win'] = 1
  Yt[Yt=='no-win'] = -1
  Xtest = pd.DataFrame(Xtest).to_numpy()
  Ytest = pd.DataFrame(Ytest).to_numpy()
  Ytest[Ytest=='win'] = 1
  Ytest[Ytest=='no-win'] = -1
  boost = adaBoost(nClass=100)
  boost.fit(Xt, Yt)
  y_est = boost.predict(Xtest)
  acc = getAcc(Ytest, y_est)
  print('Acc:', acc)
```