import numpy as np  
import pandas as pd  
import pylab  
from sklearn.neural\_network import MLPClassifier  
from alpha\_vantage.timeseries import TimeSeries  
  
ts = TimeSeries(key='33VUO7M2H5J562FP',output\_format='pandas', indexing\_type='date')  
  
filename = ['vaw','vcr','vdc','vde','vfh','vgt','vht','vis','vnq','vox','vpu']  
datafile\_count = len(filename)  
sector\_rawdata = {}  
min\_length = 10000  
  
for i, file in enumerate(filename):  
 rawdata, meta\_data = ts.get\_monthly(symbol=filename[i])  
 sector\_rawdata[i] = rawdata.values  
 min\_length = min(min\_length,sector\_rawdata[i].shape[0])  
  
rawdata, meta\_data = ts.get\_monthly(symbol='spy')  
voo\_rawdata = rawdata.values  
print rawdata.head(5)  
  
  
# In[79]:  
  
print rawdata.tail(1)  
print voo\_rawdata[-1]  
  
  
# In[80]:  
  
sector\_data = np.empty([datafile\_count,min\_length,5])  
for k in range(datafile\_count):  
 offset = sector\_rawdata[k].shape[0] - min\_length  
 sector\_data[k] = sector\_rawdata[k][offset:]  
  
  
# In[81]:  
  
back\_months = 12  
volatility\_months = 3  
sector\_input = np.zeros([datafile\_count,min\_length-back\_months,back\_months\*2+volatility\_months])  
  
offset\_voo = voo\_rawdata.shape[0] - min\_length  
  
# Compute price change and volatility for last three months  
for k in range(datafile\_count):  
 offset = sector\_rawdata[k].shape[0] - min\_length  
 for i in range(min\_length-back\_months):  
 j = i + offset  
 volatility1 = [sector\_rawdata[k][j+1,4],0]  
 volatility2 = [sector\_rawdata[k][j+1,4],0]  
 volatility3 = [sector\_rawdata[k][j+1,4],0]  
 for l in range(back\_months):  
 # sector\_input[k][i,l] = 10 \* (sector\_rawdata[k][j+l,3] - sector\_rawdata[k][j,3]) / sector\_rawdata[k][j,3] / (back\_months +1)  
 sector\_input[k][i,l] = 10 \* (sector\_rawdata[k][j+l,3] - sector\_rawdata[k][j+l-1,3]) / sector\_rawdata[k][j+l-1,3]  
 # Compute volatility for last three months  
 volatility1[0] = min(volatility1[0],sector\_rawdata[k][j+1,4])  
 volatility1[1] = max(volatility1[1],sector\_rawdata[k][j+1,2])  
 volatility2[0] = min(volatility2[0],sector\_rawdata[k][j+1,4],sector\_rawdata[k][j+2,4])  
 volatility2[1] = max(volatility2[1],sector\_rawdata[k][j+1,2],sector\_rawdata[k][j+2,2])  
 volatility3[0] = min(volatility3[0],sector\_rawdata[k][j+1,4],sector\_rawdata[k][j+2,4],sector\_rawdata[k][j+3,4])  
 volatility3[1] = max(volatility3[1],sector\_rawdata[k][j+1,2],sector\_rawdata[k][j+2,2],sector\_rawdata[k][j+3,2])  
 sector\_input[k][i,l\*2+2] = (volatility1[1] - volatility1[0])/100  
 sector\_input[k][i,l\*2+1] = (volatility2[1] - volatility2[0])/100  
 sector\_input[k][i,l\*2] = (volatility3[1] - volatility3[0])/100  
  
 j = i + offset\_voo  
 for l in range(back\_months):  
 # sector\_input[k][i,back\_months+l] = 10 \* (voo\_rawdata[j+l,3] - voo\_rawdata[j,3]) / voo\_rawdata[j,3] / (back\_months +1)  
 sector\_input[k][i,back\_months+l] = 10 \* (voo\_rawdata[j+l,3] - voo\_rawdata[j+l-1,3]) / voo\_rawdata[j+l-1,3]  
  
  
  
  
# In[82]:  
  
sector\_classif = np.zeros([datafile\_count,min\_length-back\_months])  
  
price\_change = np.zeros([datafile\_count,min\_length-back\_months-1])  
voo\_price\_change = np.zeros(min\_length-back\_months-1)  
offset\_voo = voo\_rawdata.shape[0] - min\_length  
  
for i in range(min\_length-back\_months-1):  
  
 j = i + offset\_voo  
  
 voo\_price\_change[i] = (voo\_rawdata[j+back\_months+1,3] - voo\_rawdata[j+back\_months,3]) / voo\_rawdata[j+back\_months,3]  
  
 for k in range(datafile\_count):  
  
 offset = sector\_rawdata[k].shape[0] - min\_length  
 l = i + offset  
  
 price\_change[k][i] = (sector\_rawdata[k][l+back\_months+1,3] - sector\_rawdata[k][l+back\_months,3]) / sector\_rawdata[k][l+back\_months,3]  
  
 if (price\_change[k][i] >= voo\_price\_change[i]):  
 sector\_classif[k][i] = 1  
 else:  
 sector\_classif[k][i] = 0  
  
  
# In[83]:  
  
print sector\_input.shape  
xdata = sector\_input[k,:,0:back\_months-1]  
print xdata.shape  
  
  
# In[92]:  
  
min\_train = 30  
max\_train = 150  
offset = 0  
# Model error rate   
model\_error = np.zeros([datafile\_count,sector\_classif.shape[1],3])  
average\_error = np.zeros([datafile\_count,3])  
model\_loss = np.zeros([datafile\_count,sector\_classif.shape[1],3])  
model\_predict\_valid = np.zeros([datafile\_count,sector\_classif.shape[1]])  
  
#np.random.seed(222)  
np.random.seed()  
  
for k in range(datafile\_count):  
 #for k in range(1):  
  
  
 classif = sector\_classif[k]  
 #xdata = sector\_input[k,:,0:back\_months-1]  
 xdata = sector\_input[k]  
 for iter in range(min\_train+offset,min\_length-back\_months-1,1):  
  
 n\_train = iter  
 n\_valid = iter + 1  
  
 train\_inds = range(max(0,n\_train-max\_train),n\_train)  
 valid\_inds = [n\_valid]  
 train\_set = xdata[train\_inds]  
 valid\_set = xdata[valid\_inds]  
 train\_classif = classif[train\_inds]  
 valid\_classif = classif[valid\_inds]  
 clf = MLPClassifier(solver='lbfgs', alpha=1e-5, hidden\_layer\_sizes=(5,120),activation = 'tanh')  
 clf.fit(train\_set, train\_classif)  
 model\_predict\_valid[k][iter] = clf.predict(valid\_set)  
 model\_error[k][iter][0] = 100\*(1.0 - clf.score(train\_set, train\_classif))  
 model\_error[k][iter][1] = model\_predict\_valid[k][iter] != valid\_classif  
 unique,error = np.unique(model\_error[k][min\_train+offset:iter,1], return\_counts=True)  
 if error.shape[0] > 1:  
 model\_error[k][iter][2] = 100 \* error[1] / (error[0] + error[1])  
  
average\_error = np.sum(model\_error,axis = 1) / iter;  
total\_error = 0  
for k in range(datafile\_count):  
 print("Validation Error rate for ", filename[k],": ",average\_error[k][1])  
 total\_error += average\_error[k][1]  
print("Total Sample Validation Error rate: ", 100\*total\_error/datafile\_count)  
  
month\_axis = range(min\_train+offset,min\_length-back\_months,1)  
  
print "Training Error Rate for each Month"  
pylab.plot(month\_axis,np.swapaxes(model\_error[:,min\_train+offset:sector\_classif.shape[1],0],0,1))  
pylab.grid(True) # affiche un grille  
pylab.xlabel('Months')  
pylab.ylabel("Error Rate")  
#pylab.legend(('vaw','vcr','vdc','vde','vfh','vgt','vht','vis','vnq','vox','vpu'))  
pylab.legend(('vaw','vcr','vdc','vde','vfh','vgt','vht','vis','vnq','vox','vpu'),loc='upper center',  
 bbox\_to\_anchor=(0.5, -0.05),ncol=6, fancybox=True, shadow=True)  
pylab.title('Training Error Rate in time')  
pylab.show()  
  
print "Validation Error Rate for each Month"  
pylab.plot(month\_axis,np.swapaxes(model\_error[:,min\_train+offset:sector\_classif.shape[1],2],0,1))  
pylab.grid(True) # affiche un grille  
pylab.xlabel('Months')  
pylab.ylabel("Error Rate")  
#pylab.legend(('vaw','vcr','vdc','vde','vfh','vgt','vht','vis','vnq','vox','vpu'))  
pylab.legend(('vaw','vcr','vdc','vde','vfh','vgt','vht','vis','vnq','vox','vpu'),loc='upper center',  
 bbox\_to\_anchor=(0.5, -0.05),ncol=6, fancybox=True, shadow=True)  
pylab.title('Validation Error Rate in time')  
pylab.show()  
  
  
  
# In[ ]:  
  
# Do a classical training on two third of dataset and test on one third by concatening all the data  
  
  
xdata = sector\_input[0]  
classif = sector\_classif[0]  
  
for k in range(1,datafile\_count):  
 classif = np.hstack([sector\_classif[k],classif])  
 xdata = np.row\_stack([sector\_input[k],xdata])  
  
indices1 = np.arange(0,xdata.shape[0])  
np.random.shuffle(indices1)  
  
train\_set = xdata[indices1[0:1100]]  
valid\_set = xdata[indices1[1100:]]  
train\_classif = classif[indices1[0:1100]]  
valid\_classif = classif[indices1[1100:]]  
clf = MLPClassifier(solver='lbfgs', alpha=1e-5, hidden\_layer\_sizes=(5,120),activation = 'tanh')  
clf.fit(train\_set, train\_classif)  
model\_error\_train = 100\*(1.0 - clf.score(train\_set, train\_classif))  
model\_predict\_valid = clf.predict(valid\_set)  
model\_error\_valid = 100.0\*(np.sum(model\_predict\_valid != valid\_classif)/(valid\_classif.shape[0]\*1.0))  
print model\_error\_train,model\_error\_valid  
  
  
# In[72]:  
  
  
print np.sum(model\_predict\_valid != valid\_classif)/valid\_classif.shape[0]  
  
  
  
  
# In[93]:  
  
# Build and Modify Portfolio investment each month and check returns againt VOO (S&P 500)  
np.random.seed(111)  
# Investment per year from offset point as better results with shorter periods of training  
#print vis\_rawdata[:,1800]  
voo\_dollar = np.zeros(sector\_classif.shape[1]-back\_months)  
voo\_share = np.zeros(sector\_classif.shape[1]-back\_months)  
portf\_dollar = np.zeros(sector\_classif.shape[1]-back\_months)  
portf\_dollar\_gl = np.zeros(sector\_classif.shape[1]-back\_months)  
portf\_share = np.zeros([sector\_classif.shape[1]-back\_months,datafile\_count])  
portf\_dollar\_even = np.zeros(sector\_classif.shape[1]-back\_months)  
portf\_share\_even = np.zeros([sector\_classif.shape[1]-back\_months,datafile\_count])  
  
portf\_dollar\_random = np.zeros(sector\_classif.shape[1]-back\_months)  
portf\_share\_random = np.zeros([sector\_classif.shape[1]-back\_months,datafile\_count])  
  
  
hindsight\_dollar = np.zeros(sector\_classif.shape[1]-back\_months)  
hindsight\_share = np.zeros([sector\_classif.shape[1]-back\_months,datafile\_count])  
hindsight\_dollar[min\_train+offset] = 1000000.0  
  
voo\_dollar[min\_train+offset] = 1000000.0  
portf\_dollar[min\_train+offset] = 1000000.0  
portf\_dollar\_even[min\_train+offset] = 1000000.0  
portf\_dollar\_random[min\_train+offset] = 1000000.0  
  
  
  
portf\_dollar\_weight = np.zeros(sector\_classif.shape[1])  
portf\_share\_weight = np.zeros([sector\_classif.shape[1],datafile\_count])  
portf\_dollar\_weight[min\_train+offset] = 1000000.0  
  
offset\_voo = voo\_rawdata.shape[0] - min\_length  
  
for i in range(min\_train+offset,sector\_classif.shape[1]-back\_months-1,1):  
 weight = np.multiply(1.0 - model\_error[:,i,2]/100.0,model\_predict\_valid[:,i])  
 weight = weight/np.sum(weight)  
 divide = sector\_data[:,i+back\_months+1,3] \* np.count\_nonzero(model\_predict\_valid[:,i])  
 hindsight = sector\_data[:,i+back\_months+1,3] \* np.count\_nonzero(sector\_classif[:,i+1])  
 portf\_share[i] = portf\_dollar[i]\*model\_predict\_valid[:,i]/divide  
  
 hindsight\_share[i] = hindsight\_dollar[i]\*sector\_classif[:,i+1]/hindsight  
  
 divide2 = sector\_data[:,i+back\_months+1,3] \* np.ones(11) \* 11.0  
 if not np.any(divide2 == 0.0):  
 portf\_share\_even[i] = portf\_dollar\_even[i]/divide2  
 else:  
 portf\_share\_even[i] = portf\_share\_even[i-1]  
 # print "divide by zero pass"  
  
 random\_int = np.random.randint(2, size=11)  
 divide3 = sector\_data[:,i+back\_months+1,3] \* np.count\_nonzero(random\_int)  
 if not np.any(divide3 == 0.0):  
 portf\_share\_random[i] = portf\_dollar\_random[i]\*random\_int/divide3  
 else:  
 portf\_share\_random[i] = portf\_share\_random[i-1]  
 # print "divide by zero pass"  
  
 if not np.any(sector\_data[:,i+back\_months+1,3] == 0.0):  
 portf\_share\_weight[i] = portf\_dollar\_weight[i]\*(weight/sector\_data[:,i+4,3])  
 else:  
 portf\_share\_weight[i] = portf\_share\_weight[i-1]  
  
 portf\_dollar[i+1] = np.dot(portf\_share[i].T,sector\_data[:,i+back\_months+2,3])  
 portf\_dollar\_gl[i+1] = 100 \* (portf\_dollar[i+1] - portf\_dollar[i]) / portf\_dollar[i]  
  
 hindsight\_dollar[i+1] = np.dot(hindsight\_share[i].T,sector\_data[:,i+back\_months+2,3])  
  
 portf\_dollar\_weight[i+1] = np.dot(portf\_share\_weight[i].T,sector\_data[:,i+back\_months+2,3])  
  
 portf\_dollar\_even[i+1] = np.dot(portf\_share\_even[i].T,sector\_data[:,i+back\_months+2,3])  
  
 portf\_dollar\_random[i+1] = np.dot(portf\_share\_random[i].T,sector\_data[:,i+back\_months+2,3])  
  
 j = i + offset\_voo  
  
 if not np.any(voo\_rawdata[j+back\_months,3] == 0.0):  
 voo\_share[i] = voo\_dollar[i]/voo\_rawdata[j+back\_months,3]  
 else:  
 voo\_share[i] = voo\_share[i-1]  
 if not np.any(voo\_rawdata[i+back\_months+1,3] == 0.0):  
 voo\_dollar[i+1] = voo\_share[i]\*voo\_rawdata[j+back\_months+1,3]  
 else:  
 voo\_dollar[i+1] = voo\_dollar[i]  
  
#dollar = np.column\_stack([portf\_dollar,portf\_dollar\_random,portf\_dollar\_even,voo\_dollar,hindsight\_dollar])  
dollar = np.column\_stack([portf\_dollar,portf\_dollar\_random,portf\_dollar\_even,voo\_dollar])  
month\_axis = range(min\_train+offset,sector\_classif.shape[1]-back\_months,1)  
  
pylab.plot(month\_axis,dollar[min\_train+offset:])  
pylab.grid(True) # affiche un grille  
pylab.xlabel('Months')  
pylab.ylabel("Investment Results")  
pylab.legend(('Portfolio Neural Network','Portfolio Random','Portfolia Even','S&P500','Hindsight'))  
pylab.title('ML Sector Investment Result vs Random and Baselines each month')  
  
  
profit\_loss = np.column\_stack([portf\_dollar[min\_train+offset:]-portf\_dollar\_even[min\_train+offset:],  
 portf\_dollar\_random[min\_train+offset:]-portf\_dollar\_even[min\_train+offset:]])  
# hindsight\_dollar[min\_train+offset:]-portf\_dollar\_even[min\_train+offset:]])  
pylab.show()  
pylab.plot(profit\_loss)  
pylab.grid(True) # affiche un grille  
pylab.xlabel('Months')  
pylab.ylabel("Profit/Loss")  
pylab.legend(('Portfolio Neural Network','Portfolio Random','Hindsight'))  
  
pylab.title("ML Investment Portfolio Profit/Loss vs Even Baseline compared to Random")  
  
pylab.show()  
  
pylab.plot(portf\_dollar\_gl[min\_train+offset:])  
pylab.grid(True) # affiche un grille  
pylab.xlabel('Months')  
pylab.ylabel("Profit/Loss")  
pylab.legend(('Portfolio Neural Network'))  
  
pylab.title("ML Investment Portfolio Profit/Loss")  
  
pylab.show()  
  
print portf\_dollar\_even[-1]  
print portf\_dollar[-1]-portf\_dollar\_even[-1]  
  
  
# In[ ]:  
  
print sector\_input[0,156]  
print sector\_data[0,156:,3]  
print voo\_rawdata[-4:,3]  
  
  
# In[ ]:  
  
print model\_predict\_valid[:,154:].T  
print sector\_classif[:,156].T  
print sector\_data[:,159,3].T  
print sector\_data[:,158,3].T  
print portf\_dollar\_gl[153],portf\_dollar[153]  
print sector\_input[:,156]  
print sector\_classif[:,156]  
  
  
# In[ ]:  
  
print model\_predict\_valid[:,154:]  
print sector\_classif[:,155:]  
print sector\_data[:,159]  
print xdata[155:]  
print iter  
  
  
# In[ ]:  
  
# Run a complete random Portfolio investment to build a confidence interval and to compare with classifier results  
  
random\_results = np.zeros(1000)  
  
for k in range(0,1000,1):  
  
 portf\_dollar\_random = np.zeros(vis\_classif.shape[1])  
 portf\_share\_random = np.zeros([vis\_classif.shape[1],datafile\_count])  
 portf\_dollar\_random[min\_train+offset] = 1000000.0  
  
 for i in range(min\_train+offset,vis\_classif.shape[1]-1,1):  
  
 random\_int = np.random.randint(2, size=11)  
 divide3 = vis\_rawdata[:,(i+1)\*horizon+horizon\*3,3] \* np.count\_nonzero(random\_int)  
 if not np.any(divide3 == 0.0):  
 portf\_share\_random[i] = portf\_dollar\_random[i]\*random\_int/divide3  
 else:  
 portf\_share\_random[i] = portf\_share\_random[i-1]  
 # print "divide by zero pass"  
  
 portf\_dollar\_random[i+1] = np.dot(portf\_share\_random[i].T,vis\_rawdata[:,(i+1)\*horizon+horizon\*3+future,3])  
  
 random\_results[k] = portf\_dollar\_random[-1] - portf\_dollar\_even[-1]  
  
print np.mean(random\_results),np.std(random\_results),np.mean(random\_results)-1.64\*np.std(random\_results),np.mean(random\_results)+1.96\*np.std(random\_results)  
  
  
# the histogram of the data  
n, bins, patches = plt.hist(random\_results, 50, normed=1, facecolor='green', alpha=0.75)  
  
#l = plt.plot(bins, 'r--', linewidth=1)  
  
plt.xlabel('Profit/Loss')  
plt.ylabel('Probability')  
plt.title(r'Random investment, n = 1000')  
#plt.axis([40, 160, 0, 0.03])  
plt.grid(True)  
  
plt.show()  
  
  
# In[ ]:  
  
from sklearn.model\_selection import train\_test\_split  
  
from sklearn.tree import DecisionTreeClassifier  
from sklearn.metrics import accuracy\_score  
from sklearn import tree  
  
# generate train, validate and test set  
n\_train = 50  
n\_valid = 40  
n\_test = 40  
  
# create train, validate and test sets  
classif = sector\_classif[0]  
classif = sector\_classif[0].astype(int)  
xdata = sector\_input[0]  
  
#np.random.shuffle(inds)  
train\_inds = range(n\_train)  
test\_inds = range(n\_train+n\_valid,n\_train+n\_valid+n\_test,1)  
valid\_inds = range(n\_train,n\_train+n\_valid,1)  
  
train\_set = xdata[train\_inds]  
valid\_set = xdata[valid\_inds]  
test\_set = xdata[test\_inds]  
  
train\_classif = classif[train\_inds]  
valid\_classif = classif[valid\_inds]  
test\_classif = classif[test\_inds]  
  
train\_cols = [0,1,2,3,4,5,6,7,8] #seven traits  
  
class\_type = 'Gaussian'  
cov\_type = 'diagonale'  
  
  
#log\_prob\_valid = np.zeros([vis\_classif.shape[1],datafile\_count,2])  
#classesPred\_valid = np.zeros([vis\_classif.shape[1],datafile\_count])  
  
# Organise train sets per class for Bayesian classifier  
train1 = train\_set[train\_classif==0]  
train2 = train\_set[train\_classif==1]  
  
y = classif[train\_inds]  
  
  
model\_classe1=gauss\_mv(len(train\_cols),cov\_type=cov\_type)  
model\_classe2=gauss\_mv(len(train\_cols),cov\_type=cov\_type)  
model\_classe1.train(train1[:,train\_cols])  
model\_classe2.train(train2[:,train\_cols])  
modele\_mv=[model\_classe1,model\_classe2]  
priors=[0.5,0.5]  
  
  
classifieur=classif\_bayes(modele\_mv,priors)  
  
log\_prob\_train=classifieur.compute\_predictions(train\_set[0:n\_train,train\_cols])  
log\_prob\_valid=classifieur.compute\_predictions(valid\_set[0:n\_valid,train\_cols])  
log\_prob\_test=classifieur.compute\_predictions(test\_set[0:n\_test,train\_cols])  
  
  
classesPred\_train = log\_prob\_train.argmax(1)  
classesPred\_valid = log\_prob\_valid.argmax(1)  
classesPred\_test = log\_prob\_test.argmax(1)  
  
train\_error = (1-(classesPred\_train==train\_classif).mean())\*100.0  
valid\_error = (1-(classesPred\_valid==valid\_classif).mean())\*100.0  
test\_error = (1-(classesPred\_test==test\_classif).mean())\*100.0  
  
print "Bayesian Results"  
print "Training Error = ", train\_error  
print "Validation Error = ", valid\_error  
print "Testing Error = ", test\_error  
  
clf\_gini = DecisionTreeClassifier(criterion = "gini", max\_depth=15, max\_features=7)  
clf\_gini.fit(train\_set, train\_classif)  
  
train\_error = (1-(clf\_gini.predict(train\_set)==train\_classif).mean())\*100.0  
valid\_error = (1-(clf\_gini.predict(valid\_set)==valid\_classif).mean())\*100.0  
test\_error = (1-(clf\_gini.predict(test\_set)==test\_classif).mean())\*100.0  
  
print "Decision Tree Results"  
print "Training Error = ", train\_error  
print "Validation Error = ", valid\_error  
print "Testing Error = ", test\_error  
  
  
# In[ ]:  
  
train\_cols = [0,1,2,3,4,5,6] #seven traits  
  
class\_type = 'Gaussian'  
cov\_type = 'diagonale'  
  
min\_train = 80  
max\_train = 80  
offset = 0  
end\_of\_valid = 24  
start\_test = sector\_classif.shape[1]-1-end\_of\_valid-min\_train-offset  
  
log\_prob\_valid = np.zeros([sector\_classif.shape[1],datafile\_count,2])  
classesPred\_valid = np.zeros([sector\_classif.shape[1],datafile\_count])  
train\_error = np.zeros([sector\_classif.shape[1],datafile\_count])  
valid\_error = np.zeros([sector\_classif.shape[1],datafile\_count])  
model\_error = np.zeros([sector\_classif.shape[1],datafile\_count])  
  
for k in range(datafile\_count):  
 classif = sector\_classif[k]  
 classif = sector\_classif[k].astype(int)  
 xdata = sector\_input[k]  
  
 for iter in range(min\_train+offset,sector\_classif.shape[1]-1,1):  
 n\_train = iter  
 n\_valid = iter + 1  
 log\_prob\_train = np.zeros([sector\_classif.shape[1],datafile\_count,min(iter,max\_train),2])  
 classesPred\_train = np.zeros([sector\_classif.shape[1],datafile\_count,min(iter,max\_train)])  
  
 train\_inds = range(max(0,n\_train-max\_train),n\_train)  
 valid\_inds = [n\_valid]  
 train\_set = xdata[train\_inds]  
 valid\_set = xdata[valid\_inds]  
 train\_classif = classif[train\_inds]  
 valid\_classif = classif[valid\_inds]  
 t\_inds = range(train\_set.shape[0])  
  
 # Organise train sets per class for Bayesian classifier  
 train1 = train\_set[train\_classif==0]  
 train2 = train\_set[train\_classif==1]  
  
 y = classif[train\_inds]  
  
  
 model\_classe1=gauss\_mv(len(train\_cols),cov\_type=cov\_type)  
 model\_classe2=gauss\_mv(len(train\_cols),cov\_type=cov\_type)  
 # model\_classe3=gauss\_mv(len(train\_cols),cov\_type=cov\_type)  
 model\_classe1.train(train1[:,train\_cols])  
 model\_classe2.train(train2[:,train\_cols])  
 # model\_classe3.train(train3[:,train\_cols])  
 # modele\_mv=[model\_classe1,model\_classe2,model\_classe3]  
 modele\_mv=[model\_classe1,model\_classe2]  
 # priors=[0.3333,0.3333,0.3333]  
 priors=[0.5,0.5]  
  
  
 classifieur=classif\_bayes(modele\_mv,priors)  
  
 log\_prob\_train[iter,k,:,:]=classifieur.compute\_predictions(train\_set[0:n\_train,train\_cols])  
 log\_prob\_valid[iter,k]=classifieur.compute\_predictions(valid\_set[0:n\_valid,train\_cols])  
 # log\_prob\_test=classifieur.compute\_predictions(test\_set[0:n\_test,train\_cols])  
  
 classesPred\_train[iter,k] = log\_prob\_train[iter,k,:].argmax(1)  
 classesPred\_valid[iter,k] = log\_prob\_valid[iter,k].argmax(0)  
 # classesPred\_test = log\_prob\_test.argmax(1)  
  
 train\_error[iter,k] = (1-(classesPred\_train[iter,k]==train\_classif).mean())\*100.0  
 valid\_error[iter,k] = (1-(classesPred\_valid[iter,k]==valid\_classif).mean())\*100.0  
 unique,error = np.unique(valid\_error[min\_train+offset:iter,k], return\_counts=True)  
 if error.shape[0] > 1:  
 model\_error[iter][k] = 100 \* error[1] / (error[0] + error[1])  
  
 if k == 0:  
 bayes\_error0 = train\_error[:,k]  
 bayes\_error1 = model\_error[:,k]  
 else:  
 bayes\_error0 = np.column\_stack([train\_error[:,k],bayes\_error0])  
 bayes\_error1 = np.column\_stack([model\_error[:,k],bayes\_error1])  
  
month\_axis = range(min\_train+offset,sector\_classif.shape[1],1)  
  
pylab.plot(month\_axis,bayes\_error0[min\_train+offset:])  
pylab.grid(True) # affiche un grille  
pylab.xlabel('Months')  
pylab.ylabel("Bayesian training error rates")  
pylab.legend(('vaw','vcr','vdc','vde','vfh','vgt','vht','vis','vnq','vox','vpu'),loc='lower center',  
 ncol=5, fancybox=True, shadow=True)  
pylab.show()  
  
pylab.plot(month\_axis,bayes\_error1[min\_train+offset:])  
pylab.grid(True) # affiche un grille  
pylab.xlabel('Months')  
pylab.ylabel("Bayesian validation error rates")  
pylab.legend(('vaw','vcr','vdc','vde','vfh','vgt','vht','vis','vnq','vox','vpu'),loc='lower center',  
 ncol=5, fancybox=True, shadow=True)  
pylab.show()  
  
  
  
  
# In[ ]:  
  
# This is a special section to show that we can adjust the "hyper-parameters" for the Bayesian over validation data  
# and test independently for the following 24 months to prove that we can actually beat a random walk.   
# A Bayesian does not have "hyper-parameters" but we can still adjust the training set size and the number of   
# characteristics in the input vector.  
start\_test = 0  
import matplotlib.mlab as mlab  
import matplotlib.pyplot as plt  
# Build and Modify Portfolio investment each month and check returns againt VOO (S&P 500)  
  
# Investment per year from offset point as better results with shorter periods of training  
voo\_dollar = np.zeros(sector\_classif.shape[1]-3)  
voo\_share = np.zeros(sector\_classif.shape[1]-3)  
portf\_dollar = np.zeros(sector\_classif.shape[1]-3)  
portf\_share = np.zeros([sector\_classif.shape[1]-3,datafile\_count])  
portf\_dollar\_even = np.zeros(sector\_classif.shape[1]-3)  
portf\_share\_even = np.zeros([sector\_classif.shape[1]-3,datafile\_count])  
  
portf\_dollar\_random = np.zeros(sector\_classif.shape[1]-3)  
portf\_share\_random = np.zeros([sector\_classif.shape[1]-3,datafile\_count])  
  
hindsight\_dollar = np.zeros(sector\_classif.shape[1]-3)  
hindsight\_share = np.zeros([sector\_classif.shape[1]-3,datafile\_count])  
hindsight\_dollar[min\_train+offset] = 1000000.0  
  
voo\_dollar[min\_train+offset] = 1000000.0  
portf\_dollar[min\_train+offset] = 1000000.0  
portf\_dollar\_even[min\_train+offset] = 1000000.0  
portf\_dollar\_random[min\_train+offset] = 1000000.0  
portf\_dollar\_hindsight[min\_train+offset] = 1000000.0  
  
portf\_dollar\_weight = np.zeros(sector\_classif.shape[1])  
portf\_share\_weight = np.zeros([sector\_classif.shape[1],datafile\_count])  
portf\_dollar\_weight[min\_train+offset] = 1000000.0  
  
distribution\_results = np.zeros(sector\_classif.shape[1]-1-(min\_train+offset+start\_test))  
  
random\_results = np.zeros(sector\_classif.shape[1]-1-(min\_train+offset+start\_test))  
  
offset\_voo = voo\_rawdata.shape[0] - min\_length  
  
for i in range(min\_train+offset+start\_test,sector\_classif.shape[1]-4,1):  
 divide = sector\_data[:,i+3,3] \* np.count\_nonzero(classesPred\_valid[i,:])  
 hindsight = sector\_data[:,i+3,3] \* np.count\_nonzero(sector\_classif[:,i])  
 if not np.any(divide == 0.0):  
 portf\_share[i] = portf\_dollar[i]\*classesPred\_valid[i,:]/divide  
 else:  
 portf\_share[i] = portf\_share[i-1]  
 # print "divide by zero pass"  
  
 if not np.any(hindsight == 0.0):  
 hindsight\_share[i] = hindsight\_dollar[i]\*sector\_classif[:,i]/hindsight  
 else:  
 hindsight\_share[i] = hindsight\_share[i-1]  
  
 divide2 = sector\_data[:,i+3,3] \* np.ones(11) \* 11.0  
 if not np.any(divide2 == 0.0):  
 portf\_share\_even[i] = portf\_dollar\_even[i]/divide2  
 else:  
 portf\_share\_even[i] = portf\_share\_even[i-1]  
 # print "divide by zero pass"  
  
 random\_int = np.random.randint(2, size=11)  
 divide3 = sector\_data[:,i+3,3] \* np.count\_nonzero(random\_int)  
 if not np.any(divide3 == 0.0):  
 portf\_share\_random[i] = portf\_dollar\_random[i]\*random\_int/divide3  
 else:  
 portf\_share\_random[i] = portf\_share\_random[i-1]  
 # print "divide by zero pass"  
  
 if not np.any(sector\_data[:,i+3,3] == 0.0):  
 portf\_share\_weight[i] = portf\_dollar\_weight[i]\*(weight/sector\_data[:,i+3,3])  
 else:  
 portf\_share\_weight[i] = portf\_share\_weight[i-1]  
  
 portf\_dollar[i+1] = np.dot(portf\_share[i].T,sector\_data[:,i+4,3])  
  
 hindsight\_dollar[i+1] = np.dot(hindsight\_share[i].T,sector\_data[:,i+4,3])  
  
 portf\_dollar\_weight[i+1] = np.dot(portf\_share\_weight[i].T,sector\_data[:,i+4,3])  
  
 portf\_dollar\_even[i+1] = np.dot(portf\_share\_even[i].T,sector\_data[:,i+4,3])  
  
 portf\_dollar\_random[i+1] = np.dot(portf\_share\_random[i].T,sector\_data[:,i+4,3])  
  
 distribution\_results[i-(min\_train+offset+start\_test)] = (portf\_dollar[i+1] - portf\_dollar[i]) - (portf\_dollar\_even[i+1] - portf\_dollar\_even[i])  
  
 random\_results[i-(min\_train+offset+start\_test)] = (portf\_dollar\_random[i+1] - portf\_dollar\_random[i]) - (portf\_dollar\_even[i+1] - portf\_dollar\_even[i])  
  
 j = i + offset\_voo  
  
 if not np.any(voo\_rawdata[j+3,3] == 0.0):  
 voo\_share[i] = voo\_dollar[i]/voo\_rawdata[j+3,3]  
 else:  
 voo\_share[i] = voo\_share[i-1]  
 if not np.any(voo\_rawdata[i+4,3] == 0.0):  
 voo\_dollar[i+1] = voo\_share[i]\*voo\_rawdata[j+4,3]  
 else:  
 voo\_dollar[i+1] = voo\_dollar[i]  
  
dollar = np.column\_stack([portf\_dollar,hindsight\_dollar,portf\_dollar\_even,voo\_dollar])  
month\_axis = range(min\_train+offset,sector\_classif.shape[1]-3,1)  
  
pylab.plot(month\_axis[start\_test:],dollar[min\_train+offset+start\_test:])  
pylab.grid(True) # affiche un grille  
pylab.xlabel('Months')  
pylab.ylabel("Investment Results")  
pylab.legend(('Portfolio Bayesian','Portfolio Hindsight','Portfolia Even','S&P500'))  
pylab.title('ML Sector Investment Result vs Random and Baselines each month')  
  
  
profit\_loss = np.column\_stack([portf\_dollar[min\_train+offset+start\_test:]  
 -portf\_dollar\_even[min\_train+offset+start\_test:],hindsight\_dollar[min\_train+offset+start\_test:]  
 -portf\_dollar\_even[min\_train+offset+start\_test:]])  
pylab.show()  
pylab.plot(profit\_loss)  
pylab.grid(True) # affiche un grille  
pylab.xlabel('Months')  
pylab.ylabel("Profit/Loss")  
pylab.legend(('Portfolio Bayesian','Portfolio Hindsight'))  
  
pylab.title("ML Investment Portfolio Profit/Loss vs Even Baseline compared to Random")  
  
pylab.show()  
  
print (np.mean(distribution\_results),np.std(distribution\_results),np.mean(distribution\_results)-1.96\*np.std(distribution\_results),  
 np.mean(distribution\_results)+1.96\*np.std(distribution\_results))  
  
print (np.mean(random\_results),np.std(random\_results),np.mean(random\_results)-1.96\*np.std(random\_results),  
 np.mean(random\_results)+1.96\*np.std(random\_results))  
  
  
  
# the histogram of the data  
n, bins, patches = plt.hist(distribution\_results, 50, normed=1, facecolor='green', alpha=0.75)  
  
# add a the random distribution line  
y = mlab.normpdf( bins, np.mean(random\_results),np.std(random\_results))  
l = plt.plot(bins, y, 'r--', linewidth=1)  
  
plt.xlabel('Profit/Loss')  
plt.ylabel('Probability')  
plt.title(r'Bayesian Investment Return Distribution, n = 130')  
#plt.axis([-100000, 100000, 0, .00003])  
  
#plt.axis([-100, 100, 0, .00003])  
  
plt.grid(True)  
  
plt.show()  
  
  
# In[ ]:  
  
# Build and Modify Portfolio investment each month and check returns againt VOO (S&P 500)  
start\_test = 0  
# Investment per year from offset point as better results with shorter periods of training  
#print vis\_rawdata[:,1800]  
voo\_dollar = np.zeros(vis\_classif.shape[1])  
voo\_share = np.zeros(vis\_classif.shape[1])  
portf\_dollar = np.zeros(vis\_classif.shape[1])  
portf\_share = np.zeros([vis\_classif.shape[1],datafile\_count])  
portf\_dollar\_even = np.zeros(vis\_classif.shape[1])  
portf\_share\_even = np.zeros([vis\_classif.shape[1],datafile\_count])  
  
portf\_dollar\_random = np.zeros(vis\_classif.shape[1])  
portf\_share\_random = np.zeros([vis\_classif.shape[1],datafile\_count])  
  
  
hindsight\_dollar = np.zeros(vis\_classif.shape[1])  
hindsight\_share = np.zeros([vis\_classif.shape[1],datafile\_count])  
hindsight\_dollar[min\_train+offset+start\_test] = 1000000.0  
  
voo\_dollar[min\_train+offset+start\_test] = 1000000.0  
portf\_dollar[min\_train+offset+start\_test] = 1000000.0  
portf\_dollar\_even[min\_train+offset+start\_test] = 1000000.0  
portf\_dollar\_random[min\_train+offset+start\_test] = 1000000.0  
  
portf\_dollar\_weight = np.zeros(vis\_classif.shape[1])  
portf\_share\_weight = np.zeros([vis\_classif.shape[1],datafile\_count])  
portf\_dollar\_weight[min\_train+offset+start\_test] = 1000000.0  
  
distribution\_results = np.zeros(vis\_classif.shape[1]-1-(min\_train+offset+start\_test))  
  
random\_results = np.zeros(vis\_classif.shape[1]-1-(min\_train+offset+start\_test))  
  
  
for i in range(min\_train+offset+start\_test,vis\_classif.shape[1]-1,1):  
 divide = vis\_rawdata[:,(i+1)\*horizon+horizon\*3,3] \* np.count\_nonzero(classesPred\_valid[i,:])  
 hindsight = vis\_rawdata[:,(i+1)\*horizon+horizon\*3,3] \* np.count\_nonzero(classesPred\_valid[i,:])  
 if not np.any(divide == 0.0):  
 portf\_share[i] = portf\_dollar[i]\*classesPred\_valid[i,:]/divide  
 else:  
 portf\_share[i] = portf\_share[i-1]  
 print "divide by zero pass"  
  
 if not np.any(hindsight == 0.0):  
 hindsight\_share[i] = hindsight\_dollar[i]\*classesPred\_valid[i,:]/hindsight  
 else:  
 hindsight\_share[i] = hindsight\_share[i-1]  
  
 divide2 = vis\_rawdata[:,(i+1)\*horizon+horizon\*3,3] \* np.ones(11) \* 11.0  
 if not np.any(divide2 == 0.0):  
 portf\_share\_even[i] = portf\_dollar\_even[i]/divide2  
 else:  
 portf\_share\_even[i] = portf\_share\_even[i-1]  
 print "divide by zero pass"  
  
 for k in range(100):  
 random\_int = np.random.randint(2, size=11)  
 divide3 = vis\_rawdata[:,(i+1)\*horizon+horizon\*3,3] \* np.count\_nonzero(random\_int)  
 if not np.any(divide3 == 0.0):  
 portf\_share\_random[i] = portf\_dollar\_random[i]\*random\_int/divide3  
 else:  
 portf\_share\_random[i] = portf\_share\_random[i-1]  
 # print "divide by zero pass"  
 portf\_dollar\_random[i+1] += np.dot(portf\_share\_random[i].T,vis\_rawdata[:,(i+1)\*horizon+horizon\*3+future,3])  
  
 portf\_dollar\_random[i+1] /= 100  
  
 portf\_dollar[i+1] = np.dot(portf\_share[i].T,vis\_rawdata[:,(i+1)\*horizon+horizon\*3+future,3])  
  
 hindsight\_dollar[i+1] = np.dot(hindsight\_share[i].T,vis\_rawdata[:,(i+1)\*horizon+horizon\*3+future,3])  
  
 portf\_dollar\_even[i+1] = np.dot(portf\_share\_even[i].T,vis\_rawdata[:,(i+1)\*horizon+horizon\*3+future,3])  
  
  
 distribution\_results[i-(min\_train+offset+start\_test)] = (portf\_dollar[i+1] - portf\_dollar[i]) - (portf\_dollar\_even[i+1] - portf\_dollar\_even[i])  
  
 random\_results[i-(min\_train+offset+start\_test)] = (portf\_dollar\_random[i+1] - portf\_dollar\_random[i]) - (portf\_dollar\_even[i+1] - portf\_dollar\_even[i])  
  
  
  
 if not np.any(voo\_rawdata[(i+1)\*horizon+horizon\*3,3] == 0.0):  
 voo\_share[i] = voo\_dollar[i]/voo\_rawdata[(i+1)\*horizon+horizon\*3,3]  
 else:  
 voo\_share[i] = voo\_share[i-1]  
 if not np.any(voo\_rawdata[(i+1)\*horizon+horizon\*3+future,3] == 0.0):  
 voo\_dollar[i+1] = voo\_share[i]\*voo\_rawdata[(i+1)\*horizon+horizon\*3+future,3]  
 else:  
 voo\_dollar[i+1] = voo\_dollar[i]  
  
  
dollar = np.column\_stack([portf\_dollar,portf\_dollar\_random,portf\_dollar\_even,voo\_dollar])  
  
pylab.plot(month\_axis[start\_test:],dollar[min\_train+offset+start\_test:])  
pylab.grid(True) # affiche un grille  
pylab.xlabel('Months')  
pylab.ylabel("Investment Results")  
pylab.legend(('Portfolio Bayesian','Portfolio Random','Portfolia Even','S&P500'))  
pylab.title('ML Sector Investment Result vs Random and Baselines each month')  
  
  
profit\_loss = np.column\_stack([portf\_dollar[min\_train+offset+start\_test:]  
 -portf\_dollar\_even[min\_train+offset+start\_test:],portf\_dollar\_random[min\_train+offset+start\_test:]  
 -portf\_dollar\_even[min\_train+offset+start\_test:]])  
pylab.show()  
pylab.plot(profit\_loss)  
pylab.grid(True) # affiche un grille  
pylab.xlabel('Months')  
pylab.ylabel("Profit/Loss")  
pylab.legend(('Portfolio Bayesian','Portfolio Random'))  
  
pylab.title("ML Investment Portfolio Profit/Loss vs Even Baseline compared to Random")  
  
pylab.show()  
  
#print portf\_dollar\_even[-1]  
#print portf\_dollar[-1]-portf\_dollar\_even[-1]  
print (np.mean(distribution\_results),np.std(distribution\_results),np.mean(distribution\_results)-1.96\*np.std(distribution\_results),  
 np.mean(distribution\_results)+1.96\*np.std(distribution\_results))  
  
print (np.mean(random\_results),np.std(random\_results),np.mean(random\_results)-1.96\*np.std(random\_results),  
 np.mean(random\_results)+1.96\*np.std(random\_results))  
  
  
  
# the histogram of the data  
n, bins, patches = plt.hist(distribution\_results, 50, normed=1, facecolor='green', alpha=0.75)  
  
# add a the random distribution line  
y = mlab.normpdf( bins, np.mean(random\_results),np.std(random\_results))  
l = plt.plot(bins, y, 'r--', linewidth=1)  
  
plt.xlabel('Profit/Loss')  
plt.ylabel('Probability')  
plt.title(r'Bayesian Investment Return Distribution, n = 130')  
#plt.axis([-100000, 100000, 0, .00003])  
  
#plt.axis([-100, 100, 0, .00003])  
  
plt.grid(True)  
  
plt.show()  
  
  
# In[ ]:  
  
# Now we setup a decision tree classfier  
  
train\_cols = [0,1,2,3,4,5,6] #six traits  
  
min\_train = 40  
max\_train = 50  
offset = 0  
  
train\_error = np.zeros([vis\_classif.shape[1],datafile\_count])  
valid\_error = np.zeros([vis\_classif.shape[1],datafile\_count])  
model\_error = np.zeros([vis\_classif.shape[1],datafile\_count])  
treePred\_valid = np.zeros([vis\_classif.shape[1],datafile\_count])  
  
  
for k in range(datafile\_count):  
 classif = vis\_classif[k]  
 classif = vis\_classif[k].astype(int)  
 xdata = vis\_input[k]  
  
 for iter in range(min\_train+offset,vis\_classif.shape[1]-1,1):  
 n\_train = iter  
 n\_valid = iter + 1  
 log\_prob\_train = np.zeros([vis\_classif.shape[1],datafile\_count,min(iter,max\_train),2])  
 classesPred\_train = np.zeros([vis\_classif.shape[1],datafile\_count,min(iter,max\_train)])  
  
 train\_inds = range(max(0,n\_train-max\_train),n\_train)  
 valid\_inds = [n\_valid]  
 train\_set = xdata[train\_inds]  
 valid\_set = xdata[valid\_inds]  
 train\_classif = classif[train\_inds]  
 valid\_classif = classif[valid\_inds]  
 t\_inds = range(train\_set.shape[0])  
  
  
 y = classif[train\_inds]  
  
 clf\_gini = DecisionTreeClassifier(criterion = "gini", random\_state = 100,  
 max\_depth=15, min\_samples\_leaf=5, max\_features=7)  
 # clf\_gini = DecisionTreeClassifier(criterion = "gini", max\_depth=15, max\_features=9)  
 clf\_gini.fit(train\_set, train\_classif)  
  
  
 treePred\_valid[iter,k] = clf\_gini.predict(valid\_set)  
  
 train\_error[iter,k] = (1-(clf\_gini.predict(train\_set)==train\_classif).mean())\*100.0  
 valid\_error[iter,k] = (1-(clf\_gini.predict(valid\_set)==valid\_classif).mean())\*100.0  
 unique,error = np.unique(valid\_error[min\_train+offset:iter,k], return\_counts=True)  
 if error.shape[0] > 1:  
 model\_error[iter][k] = 100 \* error[1] / (error[0] + error[1])  
  
 if k == 0:  
 tree\_error0 = train\_error[:,k]  
 tree\_error1 = model\_error[:,k]  
 else:  
 tree\_error0 = np.column\_stack([train\_error[:,k],tree\_error0])  
 tree\_error1 = np.column\_stack([model\_error[:,k],tree\_error1])  
  
month\_axis = range(min\_train+offset,vis\_classif.shape[1],1)  
  
pylab.plot(month\_axis,tree\_error0[min\_train+offset:])  
pylab.grid(True) # affiche un grille  
pylab.xlabel('Months')  
pylab.ylabel("Decision Tree training error rates")  
pylab.legend(('vaw','vcr','vdc','vde','vfh','vgt','vht','vis','vnq','vox','vpu'),loc='lower center',  
 ncol=5, fancybox=True, shadow=True)  
pylab.show()  
  
pylab.plot(month\_axis,tree\_error1[min\_train+offset:])  
pylab.grid(True) # affiche un grille  
pylab.xlabel('Months')  
pylab.ylabel("Decision Tree validation error rates")  
pylab.legend(('vaw','vcr','vdc','vde','vfh','vgt','vht','vis','vnq','vox','vpu'),loc='lower center',  
 ncol=5, fancybox=True, shadow=True)  
pylab.show()  
  
  
  
  
  
  
# In[ ]:  
  
# Build and Modify Portfolio investment each month and check returns againt VOO (S&P 500)  
  
# Investment per year from offset point as better results with shorter periods of training  
#print vis\_rawdata[:,1800]  
voo\_dollar = np.zeros(vis\_classif.shape[1])  
voo\_share = np.zeros(vis\_classif.shape[1])  
portf\_dollar = np.zeros(vis\_classif.shape[1])  
portf\_share = np.zeros([vis\_classif.shape[1],datafile\_count])  
portf\_dollar\_even = np.zeros(vis\_classif.shape[1])  
portf\_share\_even = np.zeros([vis\_classif.shape[1],datafile\_count])  
  
portf\_dollar\_random = np.zeros(vis\_classif.shape[1])  
portf\_share\_random = np.zeros([vis\_classif.shape[1],datafile\_count])  
  
  
hindsight\_dollar = np.zeros(vis\_classif.shape[1])  
hindsight\_share = np.zeros([vis\_classif.shape[1],datafile\_count])  
hindsight\_dollar[min\_train+offset] = 1000000.0  
  
voo\_dollar[min\_train+offset] = 1000000.0  
portf\_dollar[min\_train+offset] = 1000000.0  
portf\_dollar\_even[min\_train+offset] = 1000000.0  
portf\_dollar\_random[min\_train+offset] = 1000000.0  
  
portf\_dollar\_weight = np.zeros(vis\_classif.shape[1])  
portf\_share\_weight = np.zeros([vis\_classif.shape[1],datafile\_count])  
portf\_dollar\_weight[min\_train+offset] = 1000000.0  
  
distribution\_results = np.zeros(vis\_classif.shape[1]-1-(min\_train+offset))  
  
random\_results = np.zeros(vis\_classif.shape[1]-1-(min\_train+offset))  
  
  
for i in range(min\_train+offset,vis\_classif.shape[1]-1,1):  
 divide = vis\_rawdata[:,(i+1)\*horizon+horizon\*3,3] \* np.count\_nonzero(treePred\_valid[i,:])  
 hindsight = vis\_rawdata[:,(i+1)\*horizon+horizon\*3,3] \* np.count\_nonzero(treePred\_valid[i,:])  
 if not np.any(divide == 0.0):  
 portf\_share[i] = portf\_dollar[i]\*treePred\_valid[i,:]/divide  
 else:  
 portf\_share[i] = portf\_share[i-1]  
 # print "divide by zero pass"  
  
 if not np.any(hindsight == 0.0):  
 hindsight\_share[i] = hindsight\_dollar[i]\*treePred\_valid[i,:]/hindsight  
 else:  
 hindsight\_share[i] = hindsight\_share[i-1]  
  
 divide2 = vis\_rawdata[:,(i+1)\*horizon+horizon\*3,3] \* np.ones(11) \* 11.0  
 if not np.any(divide2 == 0.0):  
 portf\_share\_even[i] = portf\_dollar\_even[i]/divide2  
 else:  
 portf\_share\_even[i] = portf\_share\_even[i-1]  
 # print "divide by zero pass"  
  
 random\_int = np.random.randint(2, size=11)  
 divide3 = vis\_rawdata[:,(i+1)\*horizon+horizon\*3,3] \* np.count\_nonzero(random\_int)  
 if not np.any(divide3 == 0.0):  
 portf\_share\_random[i] = portf\_dollar\_random[i]\*random\_int/divide3  
 else:  
 portf\_share\_random[i] = portf\_share\_random[i-1]  
 # print "divide by zero pass"  
  
 portf\_dollar[i+1] = np.dot(portf\_share[i].T,vis\_rawdata[:,(i+1)\*horizon+horizon\*3+future,3])  
  
 hindsight\_dollar[i+1] = np.dot(hindsight\_share[i].T,vis\_rawdata[:,(i+1)\*horizon+horizon\*3+future,3])  
  
 portf\_dollar\_even[i+1] = np.dot(portf\_share\_even[i].T,vis\_rawdata[:,(i+1)\*horizon+horizon\*3+future,3])  
  
 portf\_dollar\_random[i+1] = np.dot(portf\_share\_random[i].T,vis\_rawdata[:,(i+1)\*horizon+horizon\*3+future,3])  
  
  
 distribution\_results[i-(min\_train+offset)] = (portf\_dollar[i+1] - portf\_dollar[i])  
  
 random\_results[i-(min\_train+offset)] = (portf\_dollar\_random[i+1] - portf\_dollar\_random[i])  
  
  
  
 if not np.any(voo\_rawdata[(i+1)\*horizon+horizon\*3,3] == 0.0):  
 voo\_share[i] = voo\_dollar[i]/voo\_rawdata[(i+1)\*horizon+horizon\*3,3]  
 else:  
 voo\_share[i] = voo\_share[i-1]  
 if not np.any(voo\_rawdata[(i+1)\*horizon+horizon\*3+future,3] == 0.0):  
 voo\_dollar[i+1] = voo\_share[i]\*voo\_rawdata[(i+1)\*horizon+horizon\*3+future,3]  
 else:  
 voo\_dollar[i+1] = voo\_dollar[i]  
  
  
dollar = np.column\_stack([portf\_dollar,portf\_dollar\_random,portf\_dollar\_even,voo\_dollar])  
  
pylab.plot(month\_axis,dollar[min\_train+offset:])  
pylab.grid(True) # affiche un grille  
pylab.xlabel('Months')  
pylab.ylabel("Investment Results")  
pylab.legend(('Portfolio Decision Tree','Portfolio Random','Portfolia Even','S&P500'))  
pylab.title('ML Sector Investment Result vs Random and Baselines each month')  
  
  
profit\_loss = np.column\_stack([portf\_dollar[min\_train+offset:]-portf\_dollar\_even[min\_train+offset:],portf\_dollar\_random[min\_train+offset:]-portf\_dollar\_even[min\_train+offset:]])  
pylab.show()  
pylab.plot(profit\_loss)  
pylab.grid(True) # affiche un grille  
pylab.xlabel('Months')  
pylab.ylabel("Profit/Loss")  
pylab.legend(('Portfolio Decision Tree','Portfolio Random'))  
  
pylab.title("ML Investment Portfolio Profit/Loss vs Even Baseline compared to Random")  
  
pylab.show()  
  
  
  
print (np.mean(distribution\_results),np.std(distribution\_results),np.mean(distribution\_results)-1.64\*np.std(distribution\_results),  
 np.mean(distribution\_results)+1.96\*np.std(distribution\_results))  
  
  
# the histogram of the data  
n, bins, patches = plt.hist(distribution\_results, 50, normed=1, facecolor='green', alpha=0.75)  
  
# add a 'best fit' line  
y = mlab.normpdf( bins, np.mean(random\_results),np.std(random\_results))  
l = plt.plot(bins, y, 'r--', linewidth=1)  
  
plt.xlabel('Profit/Loss')  
plt.ylabel('Probability')  
plt.title(r'AI Investment Return Distribution, n = 130')  
plt.axis([-100000, 100000, 0, .00003])  
plt.grid(True)  
  
plt.show()