1. 算法导论：
   1. KNN是通过已经标记样本来预测新样本的类别；
   2. Kmeans物以类聚，人以群分，自然天成；根据样本特征自身特性进行分类，离得近的被分为一组；如果知道两个中心，然后计算每个样本距离每个中心的距离（欧氏距离），距离哪个中心近，就分到哪一组中；如果我们不知道两个中心，如果知道两组的点，可以通过两组数据点计算出两个中心；但是我们现在拿到的数据，没有中心，也不知道两组的数据；how？两个中心随机分配如何？计算每个点属于哪个中心，然后通过点来更新中心，再次计算每个点的归属，依次循环，直到中心不再变化；
   3. PCA横看成岭侧成峰，远近高低各不同；横看的时候，样本分散情况，侧看样本分布情况，垂直看的情况，样本分散的情况，然后对三者按照分散情况进行排序，去掉分散值最小的，就将三维数据降低为二维数据；联想numpy时候的线性代数，当时提到过PCA，在那一部分？特征值特征向量，那么横看，侧看和垂直看三个维度，相当于特征值的?特征向量，特征值就是分散程度；求特征值和特征向量即可实现；
   4. NN天下合久必分，分久必合；提取特征进行多维复杂预测分类；ppt实例直观解析；x1+x2=1;x1x2=1;他们彼此组合，可以进行更为复杂的划分，如果要穷举所有x1和x2的组合，其增长级别是指数增长的；但是用NN却可以综合考虑多个特征值，并自定义特征的扩增和收缩；
   5. KNN和NN是监督式学习，Kmeans和PCA是非监督式学习；没有严格的界定，大致这样区分，通过是否需要对样本进行标记来区分，是否需要参考“标准答案”来预测或者训练；
   6. KNN,Kmean和PCA，就是一招，而NN更像是组合拳，可变性很强，可以调节重构的环节很多，柔性很强，如CNN，RNN等都是它的衍生及变异；
   7. KNN,Kmean和PCA都是严格定义的求解的，是有标准答案的，但是NN不是的，模糊的计算，在模糊中寻找准确；也正因为如此，所以才有了更好的延展性，而由它衍生出来的网络也都继承了它的这一特性；

Kmean方法：

**import** numpy **as** np  
**import** matplotlib.pyplot **as** plt  
**from** sklearn.datasets **import** make\_blobs  
**import** pdb  
  
**def** kmeanDIY(Samples, k):  
 Samples=np.array(Samples) *#生成numpy数组；* nSamples=Samples.shape[0]  
 pointSelected = np.random.choice(nSamples, k, replace=**False**)  
 centroids = Samples[pointSelected, :].copy() *#生成随机聚类中心；  
  
 # Kmeans++  
 # centroids=Samples[np.random.randint(0,nSamples,1),:] #随机选择一个样本作为聚类中心；  
 # tmp=np.argmax(np.sum(np.square(Samples-centroids),axis=1)) #选择第二个样本作为聚类中心  
 # centroids=np.row\_stack([Samples[tmp],centroids]) #合并聚类中心  
 # if k>2: #如果聚类中心大于2个的话；  
 # for i2 in range(2,k): #遍历剩余聚类中心  
 # distances=[]  
 # for i in range(nSamples): #遍历所有样本点  
 # distances.append(min(np.sum(np.square(Samples[i] - centroids), axis=1)))  
 # tmp=np.argmax(distances)  
 # centroids = np.row\_stack([Samples[tmp], centroids])* clusterAssment=np.zeros([nSamples,2]) *#创建[nSamples X k ]二维数组记录聚类详情,第1列记录该样本属于哪个聚类中心，第二列记录该点距离聚类中心的距离；* clusterChanged = **True  
 while** clusterChanged:  
 clusterChanged = **False** *# Dtmp=np.sum(np.square(Samples[:,np.newaxis,:]-centroids),axis=2) #代替两个for循环  
 # IndexT=np.argmin(Dtmp,axis=1)  
 # if (~np.equal(clusterAssment[:,0],IndexT)).sum():  
 # clusterChanged=True  
 # tmp=[Dtmp[i,IndexT[i]] for i in range(len(IndexT))]  
 # clusterAssment=np.column\_stack([IndexT,tmp])* **for** i **in** range(nSamples): *#遍历每一个样本；* minIndex = 0 *#记录距离哪个聚类中心最近；* minDist = np.inf *#记录距离最近聚类中心的距离；  
 # distances=np.square(np.sum(np.square(Samples[i]-centroids),axis=1))#代替一个for循环  
 # minIndex=np.argmin(distances)  
 # minDist=distances[minIndex]* **for** j **in** range(k): *#遍历每个聚类中心；* distance=sum(np.square(Samples[i]-centroids[j]))*#计算距离；* **if** distance < minDist: *#判断是否有更近的聚类中心点出现；* minDist = distance  
 minIndex = j  
 **if** clusterAssment[i, 0] != minIndex: *#判断该点是否需要更新聚类中心；* clusterChanged = **True** clusterAssment[i, :] = minIndex, minDist  
  
 **for** j **in** range(k): *#遍历每个聚类中心* pointsInCluster = Samples[clusterAssment[:,0]==j,:]  
 centroids[j, :] = np.mean(pointsInCluster, axis = 0)  
  
 *# for i in range(k): # 遍历所有聚类中心  
 # tmp = clusterAssment[:, 0] == i # 选择属于i个聚类中心点的index;  
 # tmp = Samples[tmp, :] # 选出这些点  
 # plt.scatter(tmp[:, 0], tmp[:, 1], c=colors[i], s=50)  
 # plt.show()* **return** centroids, clusterAssment  
  
X,y=make\_blobs(n\_samples=150,n\_features=2,centers=3,cluster\_std=0.5,random\_state=0) *#生成随机点；  
#plt.scatter(X[:,0],X[:,1],c='k',marker='o',s=50)*nClusters=3  
colors=[**'k'**,**'g'**,**'r'**,**'y'**,**'c'**,**'tan'**,**'gold'**,**'gray'**,**'steelblue'**,**'darkred'**]  
  
centroids,clusterAssment=kmeanDIY(X,nClusters)  
**for** i **in** range(nClusters): *#遍历所有聚类中心* tmp=clusterAssment[:,0]==i *#选择属于第i个聚类中心点的index;* tmp=X[tmp,:] *#选出这些点* plt.scatter(tmp[:,0],tmp[:,1],c=colors[i],s=50)  
plt.show()  
  
*# loss=[] #做肘形图；  
# for i in range(2,7):  
# \_, clusterAssment = kmeanDIY(X, i) #返回距离值  
# loss.append(sum(clusterAssment[:,1]))  
# plt.plot(loss)  
# plt.show()*

1. 看一下每次更新打印出来的图；
2. 肘型图确定需要几个中心点；
3. Kmean++和Kmean||都是微调优化；

PCA方法：

**import** numpy **as** np  
**import** matplotlib.pyplot **as** plt  
**from** sklearn.preprocessing **import** StandardScaler  
**import** pdb  
 **def** PCAdiy(Data,n):  
 meanV = np.mean(Data, axis=0) *# 取均值* dataNew=Data-meanV *#减去均值，归零化* covMat=np.cov(dataNew,rowvar=0) *# 0 代表一行为一个样本，非零代表一列为一个样本* eigVals,eigVects=np.linalg.eig(covMat) *#求特征值特征向量* eigVindex = np.argsort(eigVals)[::-1] *#对特征值排序并返回排序后的索引值（降序）；* **if** n<1:  
 eigTmp = eigVals[eigVindex]  
 Values = eigTmp.cumsum()  
 ValuesPercents = Values / sum(eigTmp)  
 n=sum(ValuesPercents<n)+1  
 eigVindexSelect = eigVindex[:n] *#选取eigVals的Top n 的索引；* eigVectSelect=eigVects[:,eigVindexSelect] *#选取eigVals的Top n对应的eigVectors；* **return** eigVectSelect  
  
Data=np.loadtxt(**'wine.txt'**)  
*# X = (X - X.mean(axis=0)) / X.std(axis=0)*sc=StandardScaler() *#标准化*Data=sc.fit\_transform(Data) *#标准化*vectors=PCAdiy(Data,0.6)  
dataNew=np.dot(Data,vectors)  
plt.scatter(dataNew[:,0],dataNew[:,1])  
plt.show()

Notes：特征值为负数，比如如果生成随机数，就会产生特征值为负数，说明原始数据太乱，很可能是错误数据；

NN算法

1. 更新参数θ时候应该从输入端开始更新，才能实现同时更新；
2. 经济是一个动态的充满生机的网络，网络中各个节点会维持持续动态更新、删减和添加；一种新产品的上市，是不是能经受住市场的考验，是不是能被消费认可；一种游戏，上市能不能流行起来，都是要经过现实数据的训练筛选，没有历史数据，只能实时产生，并迅速反馈，每一次运行都即是train也是test；消费反馈修改生产权重，更新经济结构，这也是历史见证行业企业兴衰交替，更迭发展过程的原因；
3. 比如我们最近提到的一路一带，这会在全球经济网络产生总体的影响，加入很多新节点，形成很多新路径，并且对原有路径节点的权重形成影响并更新；然后这些节点也会接受全球经济环境的“经济数据”训练，进行资源配置的寻优；寻优的目标函数：人尽其才，无尽其用；从而使得世界更快更好的发展；全球经济网中，著名的波罗的海指数（BDI）
4. Bug也是很明显的，现实运行中调节和反馈系统相当复杂，我们所能接收到的反馈有限的，还有很多隐含的或者我们不可见的反馈在运行，我们仅仅对我们可见的反馈做出回应，这种回应的不全面性，将会产生错误，当这种错误增大到一定程度时，就会产生一定程度的崩溃，就是所谓的经济危机；经济危机将是对网络硬性强力修改；
5. 交易：一级市场和二级市场；
   1. 一级市场：我们最熟悉的“打新”应该属于参与一级市场行为，是一级市场的最后环节，和IPO对应（即上市公司通过券商发起IPO首次公开发行卖出股票，股民通过打新股买入这些股票）；
   2. 其它的一级市场行为：一般包括天使投资，VC/PE等风险股权投资。每次投资都会增加公司的资本且一般发生在ipo前。
   3. 个人也参与一级市场，但是需要的资金量大，有较好的风险承担能力和金融知识；
   4. 二级市场：由持股股东自由交易转让股票的交易属于二级市场。卖家是投资者，买家也是投资者。一般的股票交易就在此列。实际上，二级市场的交易无法对上市公司本身的资本造成任何影响，是一种权益转让，影响的只有持股人；一般发生在ipo后。
   5. 增发可算作一级半市场把，是公司发起IPO后增加资本的主要手段。

从生产到消费过程每一级的彼此连接，都是通过交易完成，并且由此建立反馈修正基础（类似正向传播中的a(1),a(2),...）；接下来，我们进入经济的微观结构，交易；什么是交易？

价差：商品质量和价值预期；商品质量是基于商品的实用性进行的评估，从而影响价差；当我们讨论股票等投资标的的时候，它们并不具有这种实在的实用性，所以影响价差的因素仅仅是价值预期；另外投资标的的买方和卖方对买卖具有更高的统一性，即一个市场上的交易者即是买方，也是卖方；所以买卖报价也都是对等的；

1. 1. 交易基本概念，摆单拍单；冰山一角
   1. 买和卖 是相当模糊的概念，也仅仅对我们大陆A股市场适应；实际“买”和“卖”代表的是方向，即做多标的和做空标的；
2. 2. 投资交易市场
3. 3. 量化基金概况
4. 4. 市场分析--交易优势，T+0，开盘时卖掉，收盘时买回，ai增加交易优势；联系赌场；交易优势，阈值，达到一定程度激活；

胜率：并不是寻找一种方法 以至于百发百中；而是寻求一个相对稳定的概率，在相当长的时间里，赢面可以持续稳定超越亏损；

1. 数据获取
   1. 打开交易软件解释一下；
   2. 上证指数获取作图

**import** tushare **as** ts  
**import** matplotlib.pyplot **as** plt  
**import** numpy **as** np  
*#from pylab import \*  
#mpl.rcParams['font.sans-serif']=['SimHei']  
#mpl.rcParams['axes.unicode\_minus']=False*stock=**'600000'**data=ts.get\_k\_data(stock,ktype=**'D'**,autype=**None**,index=**False**,start=**'2016-01-01'**,end=**'2017-12-10'**)  
fig=plt.figure()  
ax=fig.add\_subplot(1,1,1)  
ax.plot(data[**'close'**],**'r'**,label=**'None'**)  
data=ts.get\_k\_data(stock,ktype=**'D'**,autype=**'qfq'**,index=**False**,start=**'2016-01-01'**,end=**'2017-12-10'**)  
ax.plot(data[**'close'**],**'g'**,label=**'qfq'**)  
dates=data[**'date'**]  
L=len(dates)  
xindex=list(range(0,L,L//12))  
ax.set\_xticks(xindex)  
ax.set\_xticklabels(dates[xindex],rotation=45)  
ax.grid()  
ax.set\_title(stock)  
ax.set\_xlabel(**'Date'**)  
ax.set\_ylabel(**'Price'**)  
plt.legend()  
plt.show()

练习：

1. 读取00300近一年数据，并作折线图，标注日期；
2. 读取000651/600519从2010年01月01日到现在的，不复权，前复权，和后复权数据，做一张图；比对同一时间601857/600519

Ktype 设置成5分钟等分钟线；读取600188数据，绘制[2018-01-09,2018-01-12]期间收盘价折线图；选择日期：[**True if** i[:10] **in** [**'2018-01-09'**,**'2018-01-10'**,**'2018-01-11'**,**'2018-01-12'**,**'2018-01-13'**] **else False for** i **in** dates]

Note:

1. index 000001 上证综指和平安银行都是这个代码；
2. Ktype 5 15 30 60四种，其它的不能设置，10分钟的可以合成；wind几乎可以给出任意任重数据；获取分钟数据的时候，时间限制不可用；
3. 复权一般选择前复权即默认值即可，极少数情况会用到不复权或者后复权；

df = ts.get\_stock\_basics()

df[df[‘name’]==’贵州茅台’].index[0]

Note: 没有市净率，市盈率，总市值的历史值，如何获取？

市盈/净率：每股股价/每股收益（每股净资产） ts.get\_k\_data, ts.get\_report\_data(2014,3)中的esp和bvps

总市值：每股股价\*总股本;ts.get\_k\_data,ts.get\_stock\_basics()中的totals，单位是（亿股）

df = ts.get\_realtime\_quotes('000581')

实时监测某个/几个指标是否到达某个值；

**import** tushare **as** ts  
**import** time  
**import** win32api  
  
stock=**'600000'**targetPrice=12.71  
**while** 1:  
 time.sleep(1)  
 priceCurrent=float(ts.get\_realtime\_quotes(stock)[**'price'**])  
 timeStr=time.strftime(**'%H:%M:%S'**, time.localtime())  
 **if** targetPrice>0:  
 **if** priceCurrent>targetPrice:  
 tmp=timeStr+**' stock '**+stock+**' 已大于目标点位。'** print(tmp)  
 tmp=win32api.MessageBox(0,tmp,**'Alert!'**,1)  
 **if** tmp==1:  
 print(**'Monitor continues!'**)  
 **else**:  
 print(**'Monitor exits!'**)  
 **break  
 else**:  
 print(timeStr+ **'stock '**+stock+**' 还差'**+str(round(targetPrice-priceCurrent,3))+**'个点，才能到达目标点位。'**)  
 **else**:  
 **if** priceCurrent<-targetPrice:  
 tmp = timeStr + **' stock '** + stock + **' 已小于目标点位。'** print(tmp)  
 tmp = win32api.MessageBox(0, tmp, **'Alert!'**, 1)  
 **if** tmp == 1:  
 print(**'Monitor continues!'**)  
 **else**:  
 print(**'Monitor exits!'**)  
 **break  
 else**:  
 print(timeStr+**' stock '**+stock+**' 还差'**+str(round(priceCurrent+targetPrice,3))+**'个点，才能到达目标点位。'**)

让学生自己完成：同时监测多个指标：

stock=[**'600000'**,**'300001'**,**'000001'**,**'000002'**,**'000004'**,**'000005'**]  
targetPrice=[12.71,0.3,0.3,0.3,0.3,0.3]  
**while** 1:  
 time.sleep(1)  
 priceCurrent=list(map(float,ts.get\_realtime\_quotes(stock)[**'price'**]))  
 timeStr=time.strftime(**'%H:%M:%S'**, time.localtime())  
 msgRecord=[]  
 **for** i **in** range(len(stock)):  
 **if** targetPrice[i]>0:  
 **if** priceCurrent[i]>targetPrice[i]:  
 msgRecord.append(timeStr+**' stock '**+stock[i]+**' 已大于目标点位。'**)  
 **else**:  
 print(timeStr+ **'stock '**+stock[i]+**' 还差'**+str(round(targetPrice[i]-priceCurrent[i],3))+**'个点，才能到达目标点位。'**)  
 **else**:  
 **if** priceCurrent[i]<-targetPrice[i]:  
 msgRecord.append(timeStr + **' stock '** + stock[i] + **' 已小于目标点位。'**)  
 **else**:  
 print(timeStr+**' stock '**+stock[i]+**' 还差'**+str(round(targetPrice[i]+priceCurrent[i],3))+**'个点，才能到达目标点位。'**)  
 **if** len(msgRecord):  
 tmp=**'\n'**.join(msgRecord)  
 tmp = win32api.MessageBox(0, tmp, **'Alert!'**, 1)  
 **if** tmp==1:  
 print(**'Monitor continues!'**)  
 **else**:  
 print(**'Monitor exits!'**)  
 **break**

1. 数据可视化

Matplotlib.finance标准格式

**import** tushare **as** ts  
**import** matplotlib.pyplot **as** plt  
**import** matplotlib.finance **as** mpf  
**from** matplotlib.pylab **import** date2num  
**import** numpy **as** np  
**import** datetime  
  
data=ts.get\_k\_data(**'600519'**,ktype=**'D'**,autype=**'qfq'**,start=**'2017-09-17'**,end=**'2017-12-11'**)  
prices=data[[**'open'**,**'high'**,**'low'**,**'close'**]]  
dates=data[**'date'**]  
dates=[date2num(datetime.datetime.strptime(x,**'%Y-%m-%d'**)) **for** x **in** dates ]  
candleData=np.column\_stack([dates,prices])  
plt.figure(figsize=(12,8))  
ax=plt.subplot()  
mpf.candlestick\_ohlc(ax,candleData,width=0.5,colorup=**'r'**,colordown=**'g'**)  
ax.xaxis\_date()  
ax.grid()  
plt.xticks(rotation=45)  
plt.show()

修改一下，自定义x轴

**import** tushare **as** ts  
**import** matplotlib.pyplot **as** plt  
**import** matplotlib.finance **as** mpf  
**from** matplotlib.pylab **import** date2num  
**import** numpy **as** np  
**import** datetime  
  
data=ts.get\_k\_data(**'600519'**,ktype=**'D'**,autype=**'qfq'**,start=**'2017-09-17'**,end=**'2017-12-11'**)  
prices=data[[**'open'**,**'high'**,**'low'**,**'close'**]]  
dates=data[**'date'**]  
*#dates=[date2num(datetime.datetime.strptime(x,'%Y-%m-%d')) for x in dates ]*candleData=np.column\_stack([list(range(len(dates))),prices])  
plt.figure(figsize=(12,8))  
ax=plt.subplot()  
mpf.candlestick\_ohlc(ax,candleData,width=0.5,colorup=**'r'**,colordown=**'g'**)  
dates=dates.values  
L=len(dates)  
xindex=list(range(0,L,L//12))  
ax.set\_xticks(xindex)  
ax.set\_xticklabels(dates[xindex],rotation=45)  
ax.grid()  
plt.show()

添加成交量及均线并标记点

**import** tushare **as** ts  
**import** matplotlib.pyplot **as** plt  
**import** matplotlib.finance **as** mpf  
**from** matplotlib.pylab **import** date2num  
**import** numpy **as** np  
**import** datetime  
  
data=ts.get\_k\_data(**'600519'**,ktype=**'D'**,autype=**'qfq'**,start=**'2017-09-17'**,end=**'2017-12-11'**)  
prices=data[[**'open'**,**'high'**,**'low'**,**'close'**]]  
dates=data[**'date'**]  
*#dates=[date2num(datetime.datetime.strptime(x,'%Y-%m-%d')) for x in dates ]*candleData=np.column\_stack([list(range(len(dates))),prices])  
fig=plt.figure(figsize=(12,8))  
ax=fig.add\_axes([0.1,0.3,0.8,0.6])  
mpf.candlestick\_ohlc(ax,candleData,width=0.5,colorup=**'r'**,colordown=**'g'**)  
dates=dates.values  
L=len(dates)  
closes=data[**'close'**]  
ma5=[];ma10=[]  
**for** i **in** range(4,L):  
 ma5.append(closes[i-4:i+1].mean())  
 **if** i>=9:  
 ma10.append(closes[i-9:i+1].mean())  
ax.plot(range(4,L),ma5)  
ax.plot(range(9,L),ma10)  
xindex=list(range(0,L,L//12))  
ax.set\_xticks(xindex)  
ax.grid()  
ax1=fig.add\_axes([0.1,0.1,0.8,0.2])  
ax1.bar(range(L),data[**'volume'**])  
ax1.set\_xticks(xindex)  
ax1.set\_xticklabels(dates[xindex],rotation=45)  
ax1.grid()  
plt.show()

**课堂练习：让学生自己写一下获取其它股票的数据，生成K线图；**

作出【2018-01-11，2018-01-12】5分钟k线图和成交量Bar图；bar图上加入ma5；

1. 量化策略
   1. 收盘卖掉，开盘买入；
      1. 考虑涨跌停；
      2. 比对矩阵运算和for循环，他们效果相当的时候，用矩阵运算；
      3. 添加条件，比如去掉涨停；
      4. 只有涨的时候才抛出；
      5. 还可以做更为细致的优化，比如测试涨跌幅区间，配合成交量等；
      6. 每个股票的优化参数是不一样的，跟他们的波动率相关；

T+1

**import** tushare **as** ts  
**import** matplotlib.pyplot **as** plt  
**import** matplotlib.finance **as** mpf  
**import** numpy **as** np  
  
stock=**'000001'**data=ts.get\_k\_data(stock,ktype=**'D'**,autype=**'qfq'**,start=**'2010-01-01'**,end=**'2018-01-01'**)  
closes=np.array(data[**'close'**])  
opens=np.array(data[**'open'**])  
  
*#diff=1-opens[2:]/closes[1:-1]  
#tem=(closes[1:-1]/closes[:-2]>=1.094 )|(1-closes[1:-1]/closes[:-2]>=0.094)  
#diff[tem]=0.0*diff=[]  
**for** i **in** range(2,len(closes)):  
 **if** closes[i-1]/closes[i-2]<1.094 **and** 1-closes[i-1]/closes[i-2]<0.094: #**and** closes[i-1]/closes[i-2]>1:  
 diff.append(1-opens[i]/closes[i-1])  
 *#else:  
 #diff.append(0.0)*print(np.mean(diff))  
diff=np.array(diff)  
  
plt.figure(figsize=(10,6))  
ax=plt.subplot()  
ax.plot(diff.cumsum(),label=**'Strategy'**)  
diff1=closes[1:]/closes[0]-1  
ax.plot(diff1,label=**'Common'**)  
dates=data[**'date'**].reset\_index(drop=**True**)  
L=len(dates)  
xindex=list(range(0,L,L//12))  
ax.set\_xticks(xindex)  
ax.set\_xticklabels(dates[xindex],rotation=30)  
plt.title(stock)  
plt.grid()  
plt.legend()  
plt.show()

策略二股指期货 T+0

将数据IFT\_0推送给学生；

**import** matplotlib.pyplot **as** plt  
**import** numpy **as** np  
**import** pandas **as** pd  
**import** tushare **as** ts  
  
x=pd.read\_csv(**'test.csv'**,index\_col=**'datetime'**)  
lows=x[**'low'**].values  
highs=x[**'high'**].values  
closes=x[**'close'**].values  
  
Ndays=len(closes)//48  
Re=[]  
holds=15  
**for** i **in** range(Ndays):  
 lowT=lows[i\*48:(i+1)\*48]  
 highT=highs[i\*48:(i+1)\*48]  
 closeT=closes[i\*48:(i+1)\*48]  
 startBar=holds//5  
 upline=max(highT[:startBar])  
 downline=min(lowT[:startBar])  
 **for** i2 **in** range(startBar,len(lowT)):  
 **if** lowT[i2]<downline:  
 Re.append(1-closeT[-1]/downline)  
 **break  
 elif** highT[i2]>upline:  
 Re.append(closeT[-1]/upline-1)  
 **break  
 if** i2>len(lowT)-2:  
 Re.append(0.0)  
  
Re=np.array(Re)  
plt.plot(Re.cumsum(),label=**'Strategy'**)  
reRaw=closes[47::48]/closes[::48]-1  
plt.plot(reRaw.cumsum(),label=**'HS300'**)  
plt.legend()  
plt.show()

1. 没有考虑手续费，T+0现在手续费已经很高了；
2. 这是股指期货的策略，现在使用的是沪深300指数；

Question1: 真的这么好吗？

Answer1：需要去除手续费；

这只是一个交易优势，而且要应用是有条件的，这一个手续费可以由多个交易优势分摊，或者筛选对个交易优势大的个股；

特征：5日收益，近5日波动率（求标准差），成交量变化，5日成交量方差；最高价和最低价比值；监测不同个股以及不同方法交叉验证；经典指标：KDJ，RSI，MACD，布林带等；

在T+1策略上使用Kmean方法和knn方法进行识别；基本面信息，通过tushare获取，pe,pb,市值等；自定义：比如通过复权分红波动率来评估；

Knn

1. 使用的数据为2005年到2012年，然后预测2012年到2017年
2. 直接使用数据，效果不好，然后使用比例；
3. 效果并不好，我们调整一下时间2016-7年（即2009年）到2016年训练，预测2016年到现在的情况
4. 循环递归训练，每三年数据训练接下来一年的数据；

开放式作业：考虑时间权重，考虑Re权重即改变0和1的值为具体值（扣除平均涨跌幅，参考大盘等），测试其它股票或者指数；

**from** sklearn.neighbors **import** KNeighborsClassifier  
**import** tushare **as** ts  
**import** numpy **as** np  
**import** matplotlib.pyplot **as** plt  
**import** pdb  
  
objectTrade=**'000001'  
def** Get\_XY(startDate,endDate): *#获取交易数据及其收益* data=ts.get\_k\_data(objectTrade,start=startDate,end=endDate,index=**True**)  
 closes=np.array(data[**'close'**])  
 vols=np.array(data[**'volume'**])  
 opens=np.array(data[**'open'**])  
 highs=np.array(data[**'high'**])  
 lows=np.array(data[**'low'**])  
  
 ReX=[];ReY=[]  
 **for** i **in** range(1,len(closes)-1):  
 tmp=closes[i-1]  
 ReX.append([opens[i]/tmp,closes[i]/tmp,highs[i]/tmp,lows[i]/tmp])  
 ReY.append(closes[i+1]/closes[i]-1)  
  
 X=np.array(ReX)  
 ReY=np.array(ReY)  
 **return** X,ReY  
  
**'''  
 tmp=closes[:-2]  
 openNew=opens[1:-1]/tmp  
 closeNew=closes[1:-1]/tmp  
 highNew=highs[1:-1]/tmp  
 lowNew=lows[1:-1]/tmp   
 ReY=closes[2:]/closes[1:-1]-1  
 X=np.column\_stack([openNew,closeNew,highNew,lowNew])  
 return X,ReY'''  
  
def** Fig(labels,labelsU,ReY): *#针对不同分类，作图显示其走势差异* Rlist=[];titles=[];  
 **for** i **in** range(len(labelsU)):  
 tem=labels==labelsU[i]  
 ReTem=ReY[tem]  
 Rlist.append(ReTem)  
 titles.append(**'Label:'**+str(labelsU[i]))  
 plt.figure(figsize=(15,8))  
 **for** i **in** range(len(titles)):  
 plt.plot(Rlist[i].cumsum(),label=titles[i])  
 plt.title(**'SZIndex:'**+objectTrade+**'; year:'**+str(years))  
 plt.legend()  
 plt.grid()  
 plt.show()  
  
years=2014  
X,ReY=Get\_XY(str(years-3)+**'-01-01'**,str(years)+**'-01-01'**)  
  
knn=KNeighborsClassifier(n\_neighbors=5)  
knn.fit(X,ReY>0)  
  
X,ReY=Get\_XY(str(years)+**'-01-01'**,str(years)+**'-12-01'**)  
labels=knn.predict(X)  
labelsU=np.unique(labels)  
Fig(labels,labelsU,ReY)

Kmean

1. n\_clusters=2, 分类并不合适；
2. 调整n=3（涨，跌，震荡）,效果不错，还试了一下n=4,n=5；(当N=5的时候，出现两个小的反向，样本太少，不能作为参考)；那么按照分类，我们应该怎么做，才好；用loss肘部图来确定；当n=6的时候，有两个比较好，测试也是这两个比较好；
3. 用preprocessing.StandarScaler()处理后，分类更好，不容易纠缠
4. PCA编写代码，返回的值是np.array格式，然后运算的时候用np.dot或者np.mat转换；
5. 加入PCA，0.9效果不明显，再降低一个维度到0.8的时候，输出维度降低到2维，有除去噪音的效果；

**from** sklearn.cluster **import** KMeans  
**import** tushare **as** ts  
**import** numpy **as** np  
**import** matplotlib.pyplot **as** plt  
**from** sklearn **import** preprocessing  
**from** sklearn.decomposition **import** PCA  
  
objectTrade=**'000001'  
  
def** GetXY(startDate,endDate):  
 **global** objectTrade  
 data=ts.get\_k\_data(objectTrade,start=startDate,end=endDate,index=**True**)  
 closes=np.array(data[**'close'**])  
 vols=np.array(data[**'volume'**])  
 opens=np.array(data[**'open'**])  
 highs=np.array(data[**'high'**])  
 lows=np.array(data[**'low'**])  
  
 tmp=closes[:-2]  
 openNew=opens[1:-1]/tmp  
 closeNew=closes[1:-1]/tmp  
 highNew=highs[1:-1]/tmp  
 lowNew=lows[1:-1]/tmp  
 *#volNew=vols[1:-1]/vols[:-2]* ReY=closes[2:]/closes[1:-1]-1  
 X=np.column\_stack([openNew,closeNew,highNew,lowNew])  
  
 **return** X,ReY  
  
**def** Fig(labels,labelsU,ReY):  
 Rlist=[];titles=[]  
 **for** i **in** range(len(labelsU)):  
 tem=labels==labelsU[i]  
 Rlist.append(ReY[tem])  
 titles.append(**'Label:'**+str(labelsU[i]))  
 plt.figure(figsize=(15,8))  
 **for** i **in** range(len(titles)):  
 plt.plot(Rlist[i].cumsum(),label=titles[i])  
 plt.title(objectTrade)  
 plt.legend()  
 plt.grid()  
 plt.show()  
  
X,ReY=GetXY(**'2000-01-01'**,**'2013-01-01'**)  
*# X=(X-X.mean(axis=0))/X.std(axis=0)*scaler=preprocessing.StandardScaler()  
X=scaler.fit\_transform(X)  
pca=PCA(0.8)  
X=pca.fit\_transform(X)  
kmean=KMeans(n\_clusters=6)  
kmean.fit(X)  
labels=kmean.labels\_  
labelsU=np.unique(labels)  
Fig(labels,labelsU,ReY)  
  
*# loss=[] #通过肘型图测试K值选取问题；  
# for i in range(2,25):  
# kmean=KMeans(n\_clusters=i)  
# kmean.fit(X)  
# loss.append(kmean.inertia\_)  
# plt.plot(loss)  
# plt.show()*X,ReY=GetXY(**'2013-01-01'**,**'2017-12-01'**)  
X=scaler.transform(X)  
X=pca.transform(X)  
labels=kmean.predict(X)  
Fig(labels,labelsU,ReY)

MLPnn

1. 数据不做任何处理，无法实现分类；
2. 查看原因，分的三类里面各自的个数，都差不多；
3. 考虑是因为各个数据样本太过相似，区分度小，所以引入标准化处理，减去均值除以方差;
4. 加入PCA效果通常是有的，但是因为每次MLPnn都会进行不同的初始化值，所以每次结果并不一样；造成的差异；0.9，在调整到0.85，去除噪音后效果还是不错的；
5. 调整激活函数，’identity’，’logistic’，’tanh’，’relu’ 切换；
   1. ‘identity’, no-op activation, useful to implement linear bottleneck, returns f(x) = x
   2. ‘logistic’, the logistic sigmoid function, returns f(x) = 1 / (1 + exp(-x)).
   3. ‘tanh’, the hyperbolic tan function, returns f(x) = tanh(x).
   4. ‘relu’, the rectified linear unit function, returns f(x) = max(0, x)
6. 调整solver：’lbfgs’，’sgd’，’adam’；
   1. Sgd:Stochastic gradient descent: Use 1 example in each iteration；
   2. Adam:This method change the learning rate to over time according to gradients before;
   3. Adam 适合大样本，lbfgs适合小样本；
7. 调整网络结构或者增加分类（注意每个分类，数量相当）；

**import** tushare **as** ts  
**import** numpy **as** np  
**import** matplotlib.pyplot **as** plt  
**from** sklearn.neural\_network **import** MLPClassifier  
**from** sklearn **import** preprocessing  
**from** sklearn.decomposition **import** PCA  
  
objectTrade=**'000001'  
def** GetXY(startDate,endDate): *#获取交易数据及其收益；* data=ts.get\_k\_data(objectTrade,start=startDate,end=endDate,index=**True**)  
 closes=np.array(data[**'close'**])  
 vols=np.array(data[**'volume'**])  
 opens=np.array(data[**'open'**])  
 highs=np.array(data[**'high'**])  
 lows=np.array(data[**'low'**])  
  
 tmp=closes[:-2]  
 openNew=opens[1:-1]/tmp  
 closeNew=closes[1:-1]/tmp  
 highNew=highs[1:-1]/tmp  
 lowNew=lows[1:-1]/tmp  
 ReY=closes[2:]/closes[1:-1]-1  
 X=np.column\_stack([openNew,closeNew,highNew,lowNew])  
 **return** X,ReY  
  
**def** Fig(labels,labelsU,ReY): *#对不同分类作图* Rlist=[];titles=[]  
 **for** i **in** range(len(labelsU)):  
 tem=labels==labelsU[i]  
 Rlist.append(ReY[tem])  
 titles.append(**'Label:'**+str(labelsU[i]))  
 plt.figure(figsize=(15,8))  
 **for** i **in** range(len(titles)):  
 plt.plot(Rlist[i].cumsum(),label=titles[i])  
 plt.title(objectTrade)  
 plt.legend()  
 plt.grid()  
 plt.show()  
  
X,ReY=GetXY(**'2000-01-01'**,**'2012-01-01'**)  
scaler=preprocessing.StandardScaler() *#数据标准化，等同于Z-score*X=scaler.fit\_transform(X)  
pca=PCA(2)  
X=pca.fit\_transform(X)  
clf=MLPClassifier(solver=**'adam'**,max\_iter=2000,hidden\_layer\_sizes=(10,5),activation=**'tanh'**)  
tmp=np.ones(len(ReY))  
tmp[ReY>0.005]=2  
tmp[ReY<-0.005]=0  
*#print((tmp==0).sum())  
#print((tmp==1).sum())  
#print((tmp==2).sum())*kk=clf.fit(X,tmp)  
  
labels=clf.predict(X)  
labelsU=np.unique(labels)  
Fig(labels,labelsU,ReY)  
  
X,ReY=GetXY(**'2012-01-01'**,**'2017-12-01'**)  
X=scaler.transform(X)  
X=pca.transform(X)  
labels=clf.predict(X)  
Fig(labels,labelsU,ReY)

**import** numpy **as** np  
**import** pdb  
  
x=np.random.randn(5,5)  
y = np.array([[1, 0, 0, 0],[0,1,0,0],[0,0,0,1],[0,1,0,0],[0,0,0,1]]).T  
theta1 = np.random.randn(10,5)  
theta2=np.random.randn(10,10)  
theta3=np.random.randn(4,10)  
  
**for** i **in** range(1000):  
 a1=x  
 a2 = 1/(1+np.exp(-np.dot(theta1, a1)))  
 a3= 1/(1+np.exp(-np.dot(theta2, a2)))  
 a4= 1/(1+np.exp(-np.dot(theta3, a3)))  
 delta4=a4-y  
 *# pdb.set\_trace()* delta3=np.dot(theta3.T,delta4)\*(a3\*(1-a3))  
 delta2=np.dot(theta2.T,delta3)\*(a2\*(1-a2))  
 theta1 -= np.dot(delta2,a1.T)  
 theta2 -= np.dot(delta3,a2.T)  
 theta3 -= np.dot(delta4,a3.T)  
  
print(**"the training result is:"**)  
print(a4)

**import** numpy **as** np  
**import** pdb  
  
**def** NN(x,y,hiddenLayers):  
 netWork=np.concatenate([[x.shape[0]],hiddenLayers,[y.shape[0]]])  
 Lnet=len(netWork)  
 theta=[]  
 **for** i **in** range(Lnet-1):  
 theta.append(np.random.randn(netWork[i+1],netWork[i]))  
  
 **for** i **in** range(1000):  
 a=[x]  
 **for** i2 **in** range(1,Lnet):  
 a.append(1 / (1 + np.exp(-np.dot(theta[i2-1], a[i2-1]))))  
  
 delta=[a[-1]-y]  
 **for** i2 **in** range(1,Lnet):  
 delta.insert(0,np.dot(theta[-i2].T, delta[-i2]) \* (a[-i2-1] \* (1 - a[-i2-1])))  
 **for** i2 **in** range(0,Lnet-1):  
 theta[i2]-=np.dot(delta[i2+1],a[i2].T)  
  
 print(**"the training result is:"**)  
 print(a[-1])  
  
x=np.random.randn(5,5)  
y = np.array([[1, 0, 0, 0],[0,1,0,0],[0,0,0,1],[0,1,0,0],[0,0,0,1]]).T  
NN(x,y,[10,20,12])

SVM 使用了tensorflow，题目：1. 添加核函数，2.使用one-vs-all来实现三分类；

**import** numpy **as** np  
**import** pandas **as** pd  
**import** tensorflow **as** tf  
  
*# get samples and their labels*iris = pd.read\_csv(**'C:\\Users\\Administrator\\Desktop\\iris.txt'**,header=**None**)  
x=iris.iloc[:,:4].values  
y=iris.iloc[:,4].values  
y = np.array([1 **if** i==**'Iris-setosa' else** -1 **for** i **in** y])  
  
*# use Gaussian Kernal to transform*xGaussian=[]  
**for** i **in** range(x.shape[0]):  
 xGaussian.append(np.exp(-np.sum((x[i]-x)\*\*2,axis=-1)/3))  
x=np.array(xGaussian)  
  
*# define placeholder for samples*Lx=x.shape[0]  
xHolder = tf.placeholder(shape=[**None**, Lx], dtype=tf.float32)  
yHolder = tf.placeholder(shape=[**None**, 1], dtype=tf.float32)  
  
*# define variables for thetas*w = tf.Variable(tf.random\_normal(shape=[Lx, 1]))  
b = tf.Variable(tf.random\_normal(shape=[1, 1]))  
  
*# get one part of Loss caused by thetas*lossTheta = tf.reduce\_sum(tf.square(w))  
*# get one part of Loss caused by wrong labels*xw\_b = tf.add(tf.matmul(xHolder, w), b)  
lossLabels = tf.reduce\_mean(tf.maximum(0., tf.subtract(1., tf.multiply(xw\_b, yHolder))))  
*# Loss together*lbd= tf.constant([0.01])  
loss = tf.add(lossLabels, tf.multiply(lbd, lossTheta))  
  
*# create optimizer*opt = tf.train.GradientDescentOptimizer(0.01)  
train = opt.minimize(loss)  
  
*# train*init = tf.global\_variables\_initializer()  
sess = tf.Session()  
sess.run(init)  
**for** i **in** range(1000):  
 yT = np.transpose([y])  
 sess.run(train, feed\_dict={xHolder: x, yHolder: yT})  
  
*# predict*tmp=sess.run(w)  
[[b]] = sess.run(b)  
  
theta=np.array(tmp)  
y\_predict=np.dot(x,theta)+b  
y\_predict=y\_predict.flatten()>=0  
print(**'the predict accuracy:{}%'**.format(sum(y\_predict==(y>0))/len(y)\*100))