如何更稳健的计算组合最优权重（附代码）

Original 全网Quant都在看 量化投资与机器学习 2021-07-15 16:54

收录于合集
#深度研读系列 21个 >

量化投资与机器学习公众号独家解读

量化投资与机器学公众号 ***QIML Insight——深度研读系列*** 是公众号今年全力打造的一档**深度、前沿、高水准**栏目。

公众号**遴选**了各大期刊最新论文，按照理解和提炼的方式为读者呈现每篇论文最精华的部分。QIML希望大家能够读到可以成长的量化文章，愿与你共同进步！

第一期 | 第二期 | 第三期 | 第四期 | 第五期 | 第六期
第七期 | 第八期 | 第九期

> **本期遴选论文**
> **来源：** SSRN
> **标题：** A ROBUST ESTIMATOR OF THE EFFICIENT FRONTIER October 15, 2019
> **作者：** Marcos López de Prado

今天分享的论文是Marcos López de Prado 2019年的论文《A ROBUST ESTIMATOR OF THE EFFICIENT FRONTIER》本文主要有两个创新点。

首先，提出了一种新的解决方法，称为嵌套聚类优化（NCO），该方法解决凸优化问题中噪声及复杂的信号结构引起的不稳定性。其次，作者还采用了蒙特卡罗模拟方法（Monte Carlo Optimization Selection, 以下简称为MCOS）对多种最优化算法产生的误差进行了评估（包括NCO），这样就可以根据评估的结果选择最稳健的优化模型。

不是一般性，假设有个系统有N个随机变量，他们的期望用向量 μ 表示，协方差矩阵用 V 表示。目标是找到一个权重向量 ω使得系统的方差最小，即：

$$\begin{gathered}\min_{\omega} \tfrac{1}{2}\omega' V \omega \\ \text{s.t.: } \omega' a = 1\end{gathered}$$

在金融领域，这就是一个典型的组合优化问题，当a为向量1是最优组合就是minimum variance portfolio。而当a为向量u时，最优组合就是夏普最大的组合。其解析解为：

$$\omega^* = \frac{V^{-1}a}{a'V^{-1}a}$$

这类问题称为凸优化（CVO）,为了简单起见，后面的所有讨论都基于这个最基本的凸优化问题。但这并不是说明，本文提出的方法仅适用这个最简单的问题。

不稳定性的来源

上述问题的最优解中，V 和 a 都是未知的，一般会用估计值 $\hat{V}$ 和 $\hat{a}$。正是这些估计值会导致结果的不稳定性，他们细微的变化会极大的导致结果变化。这种不稳定性可以充以下两个方面说明。

噪音：假设一个TxN的矩阵X，由N的独立同分布的随机变量组成，它们的期望为0，方差为 σ^2。矩阵 $C=T^{-1}X'X$ 有特征值 λ，根据Marcenko-Pastur理论（该定理解释了独立同分布随机变量协方差矩阵的特征值分布情况，这些特征值反映的是各种噪音的波动性），当 $N\rightarrow+\infty$，$T\rightarrow+\infty$ 且 $1<T/N<+\infty$时，λ 的概率密度函数为：

$$f[\lambda]=\begin{cases}\frac{T}{N}\frac{\sqrt{(\lambda_{+}-\lambda)(\lambda-\lambda_{-})}}{2\pi\lambda\sigma^2} & \text{if } \lambda\in[\lambda_{-},\lambda_{+}]\\ 0 & \text{if } \lambda\notin[\lambda_{-},\lambda_{+}]\end{cases}$$

其中，λ 的最大取值 $\lambda_{+}=\sigma^2\left(1+\sqrt{\frac{N}{T}}\right)^2$，最小值 $\lambda_{-}=\sigma^2\left(1-\sqrt{\frac{N}{T}}\right)^2$。当 $\sigma^2=1$时，C 为 X的相关系数矩阵。

但是，实际情况中 $\frac{N}{T}\rightarrow 1$，这时 λ_{-} 趋近0，这就导致 $\hat{V}$ 的行列式接近0，$\hat{V}$ 的逆矩阵就不能很稳健的计算，那么由此得到的解就不稳定。

信号：当相关矩阵为单位矩阵时，特征值函数为一条水平线。 除了这种理想情况下，至少有一个变量子集显示出比其他变量子集更大的相关性，从而在相关矩阵中形成一个簇。当k个变量形成一个集群时，它们更容易暴露于一个共同的特征向量，这意味着相关的特征值解释了更大的数量的方差。但是由于相关性矩阵的迹恰好是N，这意味着一个特征值只能以牺牲该簇中其他K - 1个特征值为代价而增加，从而导致条件数大于1。对于相关性矩阵聚类的特性带来的不稳定性，作者提出了嵌套聚类优化（NCO）

蒙特卡罗模拟法MCOS

MCOS求解w的过程一共包含了五个步骤：

1、估计均值和方差：以 $\{\mu,V\}$ 为参数生成矩阵 $\hat{X}$,计算矩阵 $\hat{X}$ 的均值和方差$\left\{\hat{\mu},\hat{V}\right\}$

参考如下代码：

```
import numpy as np,pandas as pd
from sklearn.covariance import LedoitWolf

def simCovMu(mu0,cov0,nObs,shrink=False):
      x=np.random.multivariate_normal(mu0.flatten(),cov0,size=nObs)
      mu1=x.mean(axis=0).reshape(-1,1)
      if shrink:cov1=LedoitWolf().fit(x).covariance_
     else:cov1=np.cov(x,rowvar=0)
     return mu1,cov1
```

2、去噪音：这一步为了解决上文提到的由于噪音带来的不稳定性。首先用KDE算法，将特征值进行Marcenko-Pastur 分布拟合。这样就能够将噪音相关的特征值从信号相关的特征值分离出来。

参考以下代码：

```python
from sklearn.neighbors.kde import KernelDensity
from scipy.optimize import minimize

def fitKDE(obs, bWidth=.25, kernel='gaussian', x=None):
      # Fit kernel to a series of obs, and derive the prob of obs
      # x is the array of values on which the fit KDE will be evaluated
      if len(obs.shape)==1: obs=obs.reshape(-1,1)
      kde=KernelDensity(kernel=kernel, bandwidth=bWidth).fit(obs)
      if x is None: x=np.unique(obs).reshape(-1,1)
      if len(x.shape)==1:  x=x.reshape(-1,1)
      logProb=kde.score_samples(x) # log(density)
      pdf=pd.Series(np.exp(logProb), index=x.flatten())
      return pdf
#------------------------------------------------------------------------------
def mpPDF(var,q,pts):
      # Marcenko-Pastur pdf
      # q=T/N
      eMin,eMax=var*(1-(1./q)**.5)**2,var*(1+(1./q)**.5)**2
      eVal=np.linspace(eMin,eMax,pts)
      pdf=q/(2*np.pi*var*eVal)*((eMax-eVal)*(eVal-eMin))**.5
      pdf=pd.Series(pdf,index=eVal)
      return pdf
#------------------------------------------------------------------------------
def errPDFs(var,eVal,q,bWidth,pts=1000):
      # Fit error
      pdf0=mpPDF(var,q,pts) # theoretical pdf
      pdf1=fitKDE(eVal,bWidth,x=pdf0.index.values) # empirical pdf
      sse=np.sum((pdf1-pdf0)**2)
      return sse
#------------------------------------------------------------------------------
def findMaxEval(eVal,q,bWidth):
      # Find max random eVal by fitting Marcenko's dist to the empirical one
      out=minimize(lambda *x:errPDFs(*x),.5,args=(eVal,q,bWidth),
      bounds=((1E-5,1-1E-5),))
      if out['success']: var=out['x'][0]
      else: var=1
      eMax=var*(1+(1./q)**.5)**2
      return eMax,var

#------------------------------------------------------------------------------
def corr2cov(corr,std):
      cov=corr*np.outer(std,std)
      return cov
#------------------------------------------------------------------------------
def cov2corr(cov):
```

```
    # Derive the correlation matrix from a covariance matrix
    std=np.sqrt(np.diag(cov))
    corr=cov/np.outer(std,std)
    corr[corr<-1],corr[corr>1]=-1,1 # numerical error
    return corr
#---------------------------------------------------------------------------------
def getPCA(matrix):
    # Get eVal,eVec from a Hermitian matrix
    eVal,eVec=np.linalg.eigh(matrix)
    indices=eVal.argsort()[::-1] # arguments for sorting eVal desc
    eVal,eVec=eVal[indices],eVec[:,indices]
    eVal=np.diagflat(eVal)
    return eVal,eVec
#---------------------------------------------------------------------------------
def denoisedCorr(eVal,eVec,nFacts):
    # Remove noise from corr by fixing random eigenvalues
    eVal_=np.diag(eVal).copy()
    eVal_[nFacts:]=eVal_[nFacts:].sum()/float(eVal_.shape[0]-nFacts)
    eVal_=np.diag(eVal_)
    corr1=np.dot(eVec,eVal_).dot(eVec.T)
    corr1=cov2corr(corr1)
    return corr1
#---------------------------------------------------------------------------------
def deNoiseCov(cov0,q,bWidth):
    corr0=cov2corr(cov0)
    eVal0,eVec0=getPCA(corr0)
    eMax0,var0=findMaxEval(np.diag(eVal0),q,bWidth)
    nFacts0=eVal0.shape[0]-np.diag(eVal0)[::-1].searchsorted(eMax0)
    corr1=denoisedCorr(eVal0,eVec0,nFacts0)
    cov1=corr2cov(corr1,np.diag(cov0)**.5)
    return cov1
```

3、最优化：根据各种方法计算最优权重，比如CVO或者上文提到的NCO，NCO的代码如下。NCO的方法能够控制信号带来的不稳定性，具体步骤如下：

- 利用相关性矩阵对变量进行聚类；
- 对每个子簇进行最优权重计算，这样可以把每个子簇看成一个变量，各子簇之间的协方差矩阵称为简化版协方差矩阵（Reduced Covariance Matrix）；
- 计算各子簇之间的最优权重；
- 结合上述两个步骤就可以得出每个变量最终的最优权重。

详细代码如下：

```
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_samples
```

```python
#------------------------------------------------------------------------------
def clusterKMeansBase(corr0,maxNumClusters=None,n_init=10):
    dist, silh=((1-corr0.fillna(0))/2.)**.5,pd.Series() # distance matrix
    if maxNumClusters is None:
        maxNumClusters=corr0.shape[0]/2
    for init in range(n_init):
        for i in xrange(2,maxNumClusters+1): # find optimal num clusters
            kmeans_=KMeans(n_clusters=i,n_jobs=1,n_init=1)
            kmeans_=kmeans_.fit(dist)
            silh_=silhouette_samples(dist,kmeans_.labels_)
            stat=(silh_.mean()/silh_.std(),silh.mean()/silh.std())
            if np.isnan(stat[1]) or stat[0]>stat[1]:
                silh,kmeans=silh_,kmeans_
    newIdx=np.argsort(kmeans.labels_)
    corr1=corr0.iloc[newIdx] # reorder rows
    corr1=corr1.iloc[:,newIdx] # reorder columns
    clstrs={i:corr0.columns[np.where(kmeans.labels_==i)[0]].tolist() for \
        i in np.unique(kmeans.labels_)} # cluster members
    silh=pd.Series(silh,index=dist.index)
    return corr1,clstrs,silh
#------------------------------------------------------------------------------
def optPort(cov,mu=None):
    inv=np.linalg.inv(cov)
    ones=np.ones(shape=(inv.shape[0],1))
    if mu is None:mu=ones
    w=np.dot(inv,mu)
    w/=np.dot(ones.T,w)
    return w
#------------------------------------------------------------------------------
def optPort_nco(cov,mu=None,maxNumClusters=None):
    cov=pd.DataFrame(cov)
    if mu is not None:mu=pd.Series(mu[:,0])
    corr1=cov2corr(cov)
    corr1,clstrs,_=clusterKMeansBase(corr1,maxNumClusters,n_init=10)
    wIntra=pd.DataFrame(0,index=cov.index,columns=clstrs.keys())
    for i in clstrs:
        cov_=cov.loc[clstrs[i],clstrs[i]].values
        mu_=(None if mu is None else mu.loc[clstrs[i]].values.reshape(-1,1))
        wIntra.loc[clstrs[i],i]=optPort(cov_,mu_).flatten()
    cov_=wIntra.T.dot(np.dot(cov,wIntra)) # reduce covariance matrix
    mu_=(None if mu is None else wIntra.T.dot(mu))
    wInter=pd.Series(optPort(cov_,mu_).flatten(),index=cov_.index)
    nco=wIntra.mul(wInter,axis=1).sum(axis=1).values.reshape(-1,1)
    return nco
```

4、蒙特卡罗模拟：结合以上所有步骤，进行多次模拟计算，步骤1每次模拟的$\left\{\hat{\mu}, \hat{V}\right\}$，都计算出对应的最优解$\hat{w}^*$

```
def monteCarlo(mu0,cov0,nObs,nSims,bWidth,minVarPortf,shrink):
      w1=pd.DataFrame(columns=xrange(cov0.shape[0]),
      index=xrange(nSims),dtype=float)
      w1_d=w1.copy(deep=True)
      for i in range(nSims):
            mu1,cov1=simCovMu(mu0,cov0,nObs,shrink)
      if minVarPortf: mu1=None
      if bWidth>0: cov1=deNoiseCov(cov1,nObs*1./cov1.shape[1],bWidth)
      w1.loc[i]=optPort(cov1,mu1).flatten()
      w1_d.loc[i]=optPort_nco(cov1,mu1,int(cov1.shape[0]/2)).flatten()
      return
```

5、误差评估：把步骤4计算的 $\hat{w}^*$ 与使用原始均值方差 $\{\mu, V\}$ 计算出的最优权重 w^* 进行比较，计算误差，误差的定义可以是以下定义之一，或其他任何合理的定义：

a. 均值误差：$\left(w^* - \hat{w}^*\right)'\mu$

b. 方差误差：$\left(w^* - \hat{w}^*\right)'V\left(w^* - \hat{w}^*\right)$

c. 夏普误差：$\frac{\left(w^* - \hat{w}^*\right)'\mu}{\left(w^* - \hat{w}^*\right)'V\left(w^* - \hat{w}^*\right)}$

现成的工具包

上文给出的代码多以说明性为目的，在真实研究中应用还有所欠缺，Github上有一个开源的完善的针对本片论文的工具包：

> https://github.com/enjine-com/mcos

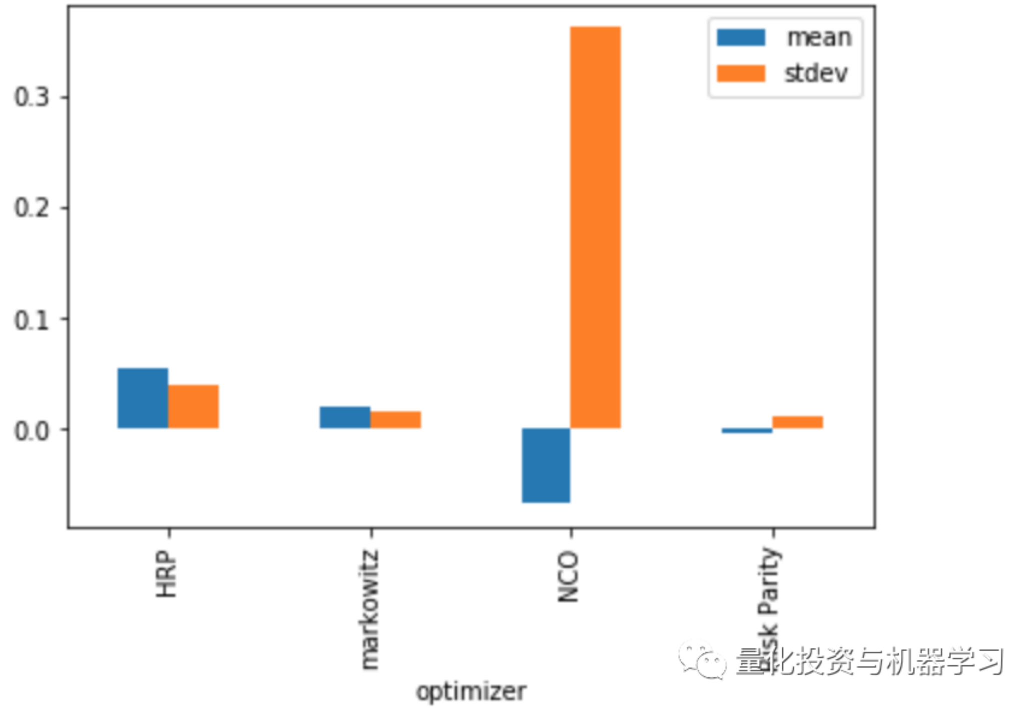
内部实现了多种最优化算法，包括Markowitz Optimization、 Nested Cluster Optimization、Risk Parity及Hierarchical Risk Parity。请看下面示例说明，针对近20只美股，对不同的权重优化算法进行比较，作者首先使用的ExpectedOutcomeErrorEstimator就是我们上文步骤5提到均值误差评估器。

```
import numpy as np
import pandas as pd
from mcos import optimizer
from mcos import observation_simulator
from mcos import mcos
from mcos.error_estimator import ExpectedOutcomeErrorEstimator, SharpeRatioErrorEstimator, \
    VarianceErrorEstimator
from mcos.covariance_transformer import DeNoiserCovarianceTransformer, AbstractCovarianceTransformer
```

```
from mcos.observation_simulator import AbstractObservationSimulator, MuCovLedoitWolfObservationSimulator, \
MuCovObservationSimulator
from pypfopt.expected_returns import mean_historical_return
from pypfopt.risk_models import sample_cov
import warnings
warnings.filterwarnings('ignore')
# Create dataframe of price history to use for expected returns and covariance
def prices_df() -> pd.DataFrame:
    tickers = ['goog','baba', 'amzn', 'wmt', 'glpi', 'bac', 'uaa', 'shld', 'jpm', 'sbux', 'amd', 'aapl','bby',
              'ge', 'rrc', 'ma','fb']
    total_df = pd.DataFrame()
    for id in tickers:
        temp = pd.read_csv( id + '.us.txt', parse_dates=True, index_col='Date')
        temp = pd.DataFrame(temp['Close']).rename(columns={"Close":id})
        if total_df.empty:
            total_df = temp
        else:
            total_df = total_df.join(temp)
    return total_df
# Choose the number of simulations to run
num_sims = 50
# Select the optimizers that you would like to compare
op = [optimizer.HRPOptimizer(), optimizer.MarkowitzOptimizer(),optimizer.NCOOptimizer(), optimizer.RiskParityO
# select the metric to use for comparison
ee = ExpectedOutcomeErrorEstimator()
# select your optional covariance transformer
cov_trans = DeNoiserCovarianceTransformer()
# convert price history to expected returns and covariance matrix
mu = mean_historical_return(prices_df()).values
cov = sample_cov(prices_df()).values
# select your observational simulator
obs_sim = MuCovObservationSimulator(mu, cov, num_sims)
# Run the simulation
results = mcos.simulate_optimizations(obs_sim, num_sims, op, ee, [cov_trans])
print(results)
results.plot.bar()
```

```
                   mean     stdev
optimizer
HRP            0.054100  0.039954
markowitz      0.020253  0.016700
NCO           -0.066727  0.360630
Risk Parity   -0.002904  0.011159
```

```
Out[4]: <matplotlib.axes._subplots.AxesSubplot at 0x2f4de6ac550>
```



上图为利用均值误差评估器，对各权重优化模型评估的结果，我们可以发现Risk Parity模型表现得最稳健。

该工具包还支持自定义优化器，并对其进行评估，有兴趣的Quant可以尝试~

收录于合集 #深度研读系列 21

< 上一篇
价值因子的改进：结合动量的思想

下一篇 >
从『Man VS AI』到『Man + AI』