**Quantitative Trading Project Report**

1. **课题选择**

考虑到数据的可获得性，和结果的可验证性，本小组选用了多因子量化模型作为主要的研究手段来寻找到有效的因子，从而可以根据股票的因子值进行选股。在大类因子的选择上，本小组尽量覆盖课上介绍到的所有类型的因子，以期对多因子策略有初步的了解。

1. **模型构建流程**
2. 数据预处理

在构建模型之前，我们需要获取大量的原始股票数据，并对它们进行初步且有效的整理。

1.1数据来源及时间长度

所有的原始股票数据的来源都是基于python的财经数据接口包Tushare及其升级版Tushare Pro。

为确保数据的可获得，以及回测时间的长度，我们提取了从2011年01月到2019年06月的数据（由于有些数据为季度数据，数据分析时还没能拿到2019年第三季度的数据，所以以2019年06月为结束点）

1.3 研究股票

我们以沪深300成份股作为股票池。由于金融行业财务指标的衡量标准与其他行业相差较大，因此为了简化模型，我们剔除了金融行业的股票。

由于每年沪深300的成分股都有变动，初期为了简化模型，我们以最新的沪深300股票列表为标准，选取其中2011年已经上市，财务数据可得的93支股票。

（以下附上93支股票的股票代码）

|  |
| --- |
| '600004.SH', '600009.SH', '600011.SH', '600027.SH', '600028.SH', '600029.SH', '600031.SH', '600048.SH', '600066.SH', '600068.SH', '600085.SH', '600089.SH', '600111.SH', '600115.SH', '600118.SH', '600170.SH', '600176.SH', '600177.SH', '600188.SH', '600196.SH', '600208.SH', '600276.SH', '600340.SH', '600352.SH', '600362.SH', '600372.SH', '600383.SH', '600415.SH', '600436.SH', '600489.SH', '600519.SH', '600535.SH', '600570.SH', '600583.SH', '600585.SH', '600588.SH', '600660.SH', '600674.SH', '600703.SH', '600809.SH', '600867.SH', '600887.SH', '600998.SH', '601006.SH', '601111.SH', '601117.SH', '601186.SH', '601607.SH', '601668.SH', '601808.SH', '601857.SH', '601898.SH', '601899.SH', '601933.SH', '000157.SZ', '000338.SZ', '000402.SZ', '000423.SZ', '000425.SZ', '000568.SZ', '000596.SZ', '000661.SZ', '000671.SZ', '000725.SZ', '000768.SZ', '000895.SZ', '000898.SZ', '000961.SZ', '002001.SZ', '002007.SZ', '002008.SZ', '002024.SZ', '002032.SZ', '002081.SZ', '002146.SZ', '002179.SZ', '002202.SZ', '002236.SZ', '002241.SZ', '002271.SZ', '002294.SZ', '002304.SZ', '002311.SZ', '002352.SZ', '002422.SZ', '002456.SZ', '002475.SZ', '002493.SZ', '300015.SZ', '300017.SZ', '300024.SZ', '300033.SZ', '300124.SZ' |

1.4因子池

根据课上介绍到的因子种类，我们挑选了能够获得数据的大类因子，包括估值、成长、反转、技术、财务质量、杠杆、规模以及波动率因子。在每类因子下挑选出1-3个进行单因子分析，再挑选出RankIC表现较好的因子进入最终策略。

（以下表格为因子池初始数据）

|  |  |  |  |
| --- | --- | --- | --- |
| 大类因子 | 因子简称 | 内容 | 频度 |
| 估值因子 | pe | 市盈率 | 月度 |
|  | pb | 市净率 | 月度 |
|  | ps | 市销率 | 月度 |
| 成长因子 | mbrg | 主营业务收入增长率(%) | 季度 |
| 反转因子 | ccp ratio | 一个月前复权收盘价之比 | 月度 |
| 财务质量 | roe | ROE | 季度 |
|  | roa | ROA | 季度 |
| 杠杆因子 | cashratio | 现金比率 | 季度 |
|  | currentratio | 流动比率 | 季度 |
| 规模因子 | lgmv | 市值对数 | 月度 |
| 技术因子 | tr | 换手率(%) | 月度 |
|  | tr\_f | 换手率(自由流通股) | 月度 |
| 波动率因子 | mmr | 每月最高价/最低价 | 月度 |

1. 单因子数据处理及分析

2.1因子预处理

我们在处理因子数据前明确了使用因子的秩即因子的排名值（rank）来进行进一步的单因子分析和多因子模型构建。

2.2异常值处理

由于在获取因子数据时会有数据缺失，我们根据每个因子的数据情况进行缺失值填充（填大或者填小），为不影响到最后的因子有效性和模型回测，在后面会根据具体处理情况分析各个因子异常值填充的合理性。

2.3因子数据处理具体过程

从Tushare中获取月度或季度的股票数据，利用python中的pandas处理数据。首先筛选出93支所要研究的股票的因子数据，并根据所研究日期进行数据对齐。若获取数据即为因子数据，则直接对因子值进行由小到大的排序；若因子值不能直接获取，则通过获取的多个相关参数的数据，进行计算最终得到因子值并排序。然后以股票代码作为index，研究日期点作为columns，制成每个因子的秩的DataFrame。

（以下附上一类因子的获取及处理过程的代码）

|  |
| --- |
| data\_ps=pd.DataFrame() # 获取2011-2019每月度的ps的rank  for i in range(104):  date=datelist[i]  basic=pro.daily\_basic(ts\_code='',  trade\_date=date,  fields='ts\_code,trade\_date,ps')  basic.index=basic["ts\_code"]  ps=basic.loc[codelist]["ps"].astype("float")  ps1=ps.fillna(1000).astype("float")  rankps=ps1.rank().astype("int")  rank\_ps=rankps.tolist()  data\_ps[date]=rank\_ps  print(data\_ps) |

2.4处理季度数据

财务指标相关的因子只能获取季度数据，我们将这些因子每个月的数据设置成等于当个季度的数据。

（以下附上处理季度数据的代码）

|  |
| --- |
| def get\_df():  '''  创建一个dataframe用于存储因子数据    输入：无    输出：dataframe对象，共93行，每一行的index都是一只股票  '''  all\_code\_factor = pd.DataFrame()  a = ['1' for i in range(93)]    all\_code\_factor = all\_code\_factor.append(a)    all\_code\_factor.index = have\_code  return all\_code\_factor |

|  |
| --- |
| def process\_sdata(indicator):  file = 'f-' + indicator + '.xlsx'  df = pd.read\_excel(file, index\_col=0)  df.columns = strtimes    newdf = get\_df()    i = 0    for season in strtimes:  for j in range(3):  month = strtime[i]  newdf[month] = list(df[season])  i += 1    newfile = 'f-' + indicator + 'm.csv'  newdf.to\_excel(newfile)    return newdf  # 处理季度数据  process\_sdata('roe')  process\_sdata('roa')  process\_sdata('cashratio')  process\_sdata('currentratio')  # 已运行 |

2.5收益率排名

由于后面分析时会使用到，所以我们计算了每只股票在2011/01-2019/06每个月的收益率，并制成表格，在此基础之上我们对收益率进行排名。

（以下附上收益率排名的代码）

|  |
| --- |
| # 获取收益率  file = 'historical returns from 2011 to 2019.xlsx'  df = pd.read\_excel(file, index\_col=0)  # 获取收益率排名  columns = list(df.columns)  for col in columns:  df[col] = df[col].rank()    # 输出收益率排名  newfile = 'f-rr.xlsx'  df.to\_excel(newfile) |

2.6处理因子排名数据中出现的问题

由于mbrg这个因子数据在获取中有多处重复以及缺失，数据清洗过后仅剩下78支股票，为了便于之后的数据对齐、模型构建及回测，所以我们剔除了mbrg这个因子。

2.7单因子分析

方法：用spearman秩相关系数（RankIC）来分析因子的各种特性，以筛选有效因子。（RankIC反映了因子横截面收益预测能力，风险调整后的IC是衡量因子预测能力的重要标准）

公式：因子排名A，收益排名B

RankIC=corr(A,B)

根据之前的数据处理结果可以获得每一个因子在每个研究时间点的rankic值，以及rankic值的均值、标准差、t值，从而作为因子筛选的重要指标。

（以下表格为最终每个因子的rankic值的具体情况描述）

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 因子名称 | rankic\_mean  (均值) | rankic\_std  (标准差) | rankic\_t  (t值) | 相关性  (+/-) |
| ccp ratio | 0.019499337 | 0.16465546 | 1.196034727 | + |
| lgmv | 0.028665983 | 0.193241479 | 1.498188892 | + |
| mmr | 0.04353553 | 0.228079179 | 1.927783591 | + |
| roe | -0.043772898 | 0.179377121 | -2.428040068 | - |
| roa | -0.046387214 | 0.211373153 | -2.183564686 | - |
| cashratio | -0.021706217 | 0.16198598 | -1.353339624 | - |
| currentratio | -0.025173834 | 0.165325317 | -1.537836225 | - |
| pe | -0.03353855 | 0.24718587 | -1.383685257 | - |
| pb | -0.045400191 | 0.285560053 | -1.621350446 | - |
| ps | -0.044456724 | 0.239005094 | -1.896911075 | - |
| tr | -0.000651555 | 0.185852675 | -0.035751868 | - |
| tr\_f | 0.008437245 | 0.186798563 | 0.460621046 | + |

(以下附上rankic值处理代码)

|  |
| --- |
| def describe\_rankic(indicator):  file = 'rankic ' + indicator + '.xlsx'  df = pd.read\_excel(file, index\_col=0)  return df.describe()  def get\_d(indicator):  data=describe\_rankic(indicator)  count=data.iloc[0][0]  mean=data.iloc[1][0]  std=data.iloc[2][0]  t=(mean/std)\*np.sqrt(count)  return mean,std,t  mean=[]  std=[]  tval=[]  df=pd.DataFrame()  for indicator in indicators:  m=get\_d(indicator)[0]  s=get\_d(indicator)[1]  t=get\_d(indicator)[2]  mean.append(m)  std.append(s)  tval.append(t)  df["mean"]=mean  df["std"]=std  df["tval"]=tval  df.index=indicators  df.to\_excel("rankic.xlsx") |

2.8筛选因子标准以及结果

(1)t值大小

可从上述表格中看到，tr因子的t值明显较小，说明因子显著性太低，所以我们剔除了tr因子。

(2)缺失值填充对于因子有效性的影响

由于之前的所有rank的排名都是从小到大的，若是因子rankic值的均值是正数，则认为因子与股票收益率呈正相关性，反之则呈负相关性。所以我们为了寻找收益率高的股票，即寻找因子排名大（正相关性）、因子排名小（负相关性）。易知只有两种填充值的情况对之后的分析影响较小：一种是缺失值填大数，但因子与收益率呈负相关性；另一种是缺失值填小数，但因子与收益率呈正相关性。

（以下表格为因子缺失值详细情况）

|  |  |  |  |
| --- | --- | --- | --- |
| 因子名称 | 缺失值(大/小) | 相关性(+/-) | 结果(有/无影响) |
| roe | 填大 | - | 无 |
| roa | 填大 | - | 无 |
| cashratio | 填大 | - | 无 |
| currentratio | 填大 | - | 无 |
| pb | 填大 | - | 无 |
| pe | 填大 | - | 无 |
| ps | 填大 | + | 无 |
| tr\_f | 填大 | + | 有 |

所以从上述表格中，我们可以剔除tr\_f因子。

(3)利用PCA(主成分分析)降维，合并相似因子

根据因子的种类，我们将cashratio，currentratio通过PCA降维成一维，降维后的因子名称为cashratio\_currentratio；将pe，pb通过PCA降维成一维，降维后的因子名称为pe\_pb。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 因子名称 | rankic\_mean | rankic\_std | rankic\_t | 相关性 |
| cashratio\_currentratio | -0.021723344 | 0.1671982021 | -1.354440555 | - |
| pe\_pb | -0.03084655 | 0.247379842 | -1.259338179 | - |

(以下为PCA处理过程代码)

|  |
| --- |
| from sklearn.decomposition import PCA  data=pd.DataFrame()  for date in strtime[:-2]:  list\_pca=[]  rank1=get\_fr(date, "cashratio\_rawm").tolist()  rank2=get\_fr(date, "currentratio\_rawm").tolist()  arr=np.array([rank1,rank2]).T  pca = PCA(n\_components=1) #降到1维  pca.fit(arr) #训练  new\_arr=pca.fit\_transform(arr) #降维后的数据  matrix=new\_arr.reshape(93,1)  for i in range(93):  list\_pca.append(matrix[i,0])  obj=pd.Series(list\_pca)  listrank=obj.rank().astype("int")  data[date]=listrank  data.index=have\_code  list\_rank=[]  pca\_cashratio\_currentratio=pca\_cashratio\_currentratio.astype("int")  for i in range(102):  r=pca\_cashratio\_currentratio.iloc[:,i].corr(data\_return.iloc[:,i])  list\_rank.append(r)  cashratio\_currentratio=pd.DataFrame(list\_rank)  cashratio\_currentratio.columns=["cashratio\_currentratio"]  cashratio\_currentratiodata=cashratio\_currentratio.describe() |

(4)最终选定因子

模型一：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 因子名称 | rankic\_mean  (均值) | rankic\_std  (标准差) | rankic\_t  (t值) | 相关性  (+/-) |
| ccp ratio | 0.019499337 | 0.16465546 | 1.196034727 | + |
| lgmv | 0.028665983 | 0.193241479 | 1.498188892 | + |
| mmr | 0.04353553 | 0.228079179 | 1.927783591 | + |
| roe | -0.043772898 | 0.179377121 | -2.428040068 | - |
| roa | -0.046387214 | 0.211373153 | -2.183564686 | - |
| cashratio | -0.021706217 | 0.16198598 | -1.353339624 | - |
| currentratio | -0.025173834 | 0.165325317 | -1.537836225 | - |
| pe | -0.03353855 | 0.24718587 | -1.383685257 | - |
| pb | -0.045400191 | 0.285560053 | -1.621350446 | - |
| ps | -0.044456724 | 0.239005094 | -1.896911075 | - |

模型二：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 因子名称 | rankic\_mean | rankic\_std | rankic\_t | 相关性 |
| ccp ratio | 0.019499337 | 0.16465546 | 1.196034727 | + |
| lgmv | 0.028665983 | 0.193241479 | 1.498188892 | + |
| mmr | 0.04353553 | 0.228079179 | 1.927783591 | + |
| roe | -0.043772898 | 0.179377121 | -2.428040068 | - |
| roa | -0.046387214 | 0.211373153 | -2.183564686 | - |
| ps | -0.044456724 | 0.239005094 | -1.896911075 | - |
| cashratio\_currentratio | -0.021723344 | 0.1671982021 | -1.354440555 | - |
| pe\_pb | -0.03084655 | 0.247379842 | -1.259338179 | - |

(三)回测模型

3.1构建回测模型

使用得分法选股。从2011年4月到2019年6月开始每月月末，以月末收盘价卖出持有的十只股票，并挑选因子等权排名前十的十只股票，以月末收盘价买入，考虑到排名靠后的股票含有当期数据缺失的股票，且沪深股票比较难卖空，因此不考虑卖空的情况。

3.2关键假设

（1）从下一季度首月月末开始即可得到上一季度的财务数据

（2）每月月末可以得到当月的股票数据，例如月最高收盘价、最低收盘价、换手率等

（3）月末选股之后，可以以当月月末收盘价买卖所需数量的股票

3.3代码实现

以模型一为例

|  |
| --- |
| # 获取所有股票在输入日期的特定因子排名  def get\_fr(date, indicator):  file = 'f-' + indicator + 'r.xlsx'  df = pd.read\_excel(file, index\_col=0)  rank\_list = df[date]  return rank\_list |
| # 获取股票在输入日期的所有因子的等权排名(由小到大，越大越好)  def get\_wr(date):  indicators=['ccp ratio', 'lgmv', 'mmr', 'roem', 'roam', 'cashratiom', 'currentratiom', 'pe', 'pb', 'ps']    rank\_all\_indicator = get\_df()    for indicator in indicators:  if indicator in ['roem', 'roam', 'cashratiom', 'currentratiom', 'cashratio\_currentratio']:  index = strtime.index(date)  sdate = strtime[index-3] # 财务数据当月无法获得，只能获取上一季度的  rank\_list = 94-get\_fr(sdate, indicator) # 负相关（rankic均值为负数）  elif indicator in ['ps','pe','pb']:  rank\_list = 94-get\_fr(date, indicator) # 负相关（rankic均值为负数）  else:  rank\_list = get\_fr(date, indicator)  rank\_all\_indicator[indicator] = rank\_list    rank\_all\_indicator['all'] = 0    for indicator in indicators:  rank\_all\_indicator['all'] = rank\_all\_indicator['all'] + rank\_all\_indicator[indicator]    return rank\_all\_indicator['all'].rank() |
| # 获取输入日期的选出的十支股票的代码  def get\_code(date):  codes = []  wr = get\_wr(date)  temp = 0  for i in range(93,73,-1):  code = list(wr[wr.values==i].index)  if code:  for c in code:  codes.append(c)  else:  temp += 1  if temp == 2:  code = list(wr[wr.values==i+0.5].index)  for c in code:  codes.append(c)    return codes |
| # 获取输入日期所有股票的收盘价  def get\_cp(date):  df = pro.monthly(trade\_date=date, fields='ts\_code,trade\_date,close')  return df |
| # 获取指定几种股票的收盘价  def get\_scp(codes, df):  cps = []  for code in codes:  ccps = list(df[df['ts\_code']==code].loc[:,'close'])    # 若数据不为空，则在list中添加数据  if ccps:  ccp = ccps[0]  cps.append(ccp)    # 若数据为空，则添加100作为数据缺失的标志（roe不会到100）  else:  print('empty error')  cps.append(0)    return cps |
| # 在输入日期买入股票，每个100手  def buy(date, money):  codes = get\_code(date)    df = get\_cp(date)  cps = get\_scp(codes, df)    for cp in cps:  money = money - cp \* 10000    return money, cp\*10000 |
| # 在输入日期卖出持有股票，每个100手  def sell(date, money, codes):    df = get\_cp(date)  cps = get\_scp(codes, df)    for cp in cps:  money = money + cp \* 10000    return money, cp\*10000 |
| # 交易  hold\_codes = []  money = 0  r = [] # 每月的资金余额  cost\_list = []  gain\_list = []  i = 0  for date in strtime[3:-2]:  codes = get\_code(date)  print(i, end=' ')  i += 1    money, cost = buy(date, money)  r.append(money)  cost\_list.append(cost)    if hold\_codes:  money, gain = sell(date, money, codes)  gain\_list.append(gain)    hold\_codes = codes |
| # 月收益率  mr\_code = []  for i, m in enumerate(gain\_list):  cmr = m / cost\_list[i] - 1  mr\_code.append(cmr)  fig, ax = plt.subplots()  x = np.linspace(0, len(mr\_code)-1, len(mr\_code))  y = np.array(mr\_code)  ax.plot(x, y)  plt.show() |
|  |
| # 从2011年4月开始至今的收益率  # 至今的收益率  ror\_code = []  cror = 1  for cr in mr\_code:  cror = cror \* (1+cr)  ror\_code.append(cror)  fig, ax = plt.subplots()  x = np.linspace(0, len(ror\_code)-1, len(ror\_code))  y = np.array(ror\_code)  ax.plot(x, y)  plt.show() |
|  |

3.4结果分析

由于股票池的挑选具有前视偏差，2011年初并不知道哪些沪深300成分股会一直在列表中直到今天，所以为了更加准确的检测模型的效果，我们还计算了从2011年初持有全部93支股票至今的收益率，并与策略结果进行比较。

|  |
| --- |
| # 股票池的收益率  sd = strtime[3]  ed = strtime[-1]  df1 = pro.monthly(trade\_date=sd, fields='ts\_code,trade\_date,close')  df2 = pro.monthly(trade\_date=ed, fields='ts\_code,trade\_date,close')  cps1 = get\_scp(have\_code, df1)  cps2 = get\_scp(have\_code, df2) |
| sum(cps2) / sum(cps1) - 1 |
|  |
| money = 0  r\_all = []  for date in strtime[3:-2]:  df = pro.monthly(trade\_date=date, fields='ts\_code,trade\_date,close')  cps = get\_scp(have\_code, df)    money = sum(cps) \* 10000  r\_all.append(money) |
| # 月收益率  mr\_all = []  for i, m in enumerate(r\_all):  if i >= 1:  cmr = m / r\_all[i-1] - 1  mr\_all.append(cmr) |
| # 至今的收益率  ror\_all = []  cror = 1  for cr in mr\_all:  cror = cror \* (1 + cr)  ror\_all.append(cror)  fig, ax = plt.subplots()  x = np.linspace(0, len(ror\_all)-1, len(ror\_all))  y = np.array(ror\_all)  ax.plot(x, y)  plt.show() |
|  |
| # 模型一，模型二，与股票池的总体收益率比较  fig, ax = plt.subplots()  x = np.linspace(0, len(ror\_all)-1, len(ror\_all))  y = np.array(ror\_all) # 股票池  z = np.array(ror\_code) # 模型一  h = np.array(ror\_pca) # 模型二  ax.plot(x, y, 'b-')  ax.plot(x, z, 'r-')  ax.plot(x, h)  plt.show() |
|  |

|  |  |  |  |
| --- | --- | --- | --- |
|  | 模型一 | 模型二  （PCA） | 参考标准  （整体股票池） |
| 年化收益率 | 13.23% | -0.07% | 2.34% |
| 月收益率标准差 | 0.95 | 0.83 | 0.073 |
| 信息比率 | 0.08 | 0.08 | - |
| 最大回撤 | 0.96 | 0.92 | 0.45 |

（四）总结

4.1模型的优化空间

（1）动态股票池，股票池应该随沪深300成份的变化而变化

（2）财务数据并非下一季度立刻可以得到上一季度的数据，要根据各个公司财报发布时间进行回测

（3）有些因子的RankIC出现一段时间为正，一段时间为负的情况，是否能设置一个合适的周期，提升策略的效果

4.2问题与反馈

（1）使用PCA合成因子之后，年化收益反而变成负值，是否说明同类因子的共性不是赚取收益的原因，因子的特性才是