Rank and Trade

MGMT 767, Quantitative Investments Lab

Kerry Back & Kevin Crotty, Rice University

Outline

1. Build feature dataset with today's features
2. Load and apply model to predict
3. Trade to 140/40 portfolio with 100% in SPY (a) Close unwanted positions (b) Rebalance SPY (c) Open/rebalance long positions (d) Open/rebalance short positions
4. Save account equity and positions

In [1]:

```
import numpy as np
import pandas as pd
from sqlalchemy import create_engine
from joblib import load
import yfinance as yf
from datetime import datetime
import os.path

from alpaca.trading.client import TradingClient
from alpaca.trading.requests import MarketOrderRequest, GetAssetsRequest, Ass
from alpaca.trading.enums import OrderSide, TimeInForce
```

Build Feature Dataset

- Don't need much history. Start here in 2022.
- And don't need weekly returns (after computing momentum).

In [2]:

```
server = 'fs.rice.edu'
database = 'stocks'
username = 'stocks'
password = '6LAZH1'
driver = 'SQL+Server'
string = f"mssql+pyodbc://{username}:{password}@{server}/{database}"
try:
    conn = create_engine(string + "?driver='SQL+Server'").connect()
except:
    try:
        conn = create_engine(string + "?driver='ODBC+Driver+18+for+SQL+Server
    except:
        import pymssql
        string = f"mssql+pymssql://{username}:{password}@{server}/{database}"
        conn = create_engine(string).connect()
```

In [3]:

```
sep_weekly = pd.read_sql(
    """
    select date, ticker, closeadj, closeunadj, volume, lastupdated from sep_we
    where date >= '2022-01-01'
    order by ticker, date, lastupdated
    """,
    conn,
)
sep_weekly = sep_weekly.groupby(["ticker", "date"]).last()
sep_weekly = sep_weekly.drop(columns=["lastupdated"])

ret = sep_weekly.groupby("ticker", group_keys=False).closeadj.pct_change()
ret.name = "ret"

price = sep_weekly.closeunadj
price.name = "price"

volume = sep_weekly.volume
volume.name = "volume"
```

In [4]:

```
ret_annual = sep_weekly.groupby("ticker", group_keys=False).closeadj.pct_chan
ret_monthly = sep_weekly.groupby("ticker", group_keys=False).closeadj.pct_cha
mom = (1 + ret_annual) / (1 + ret_monthly) - 1
mom.name = "mom"
```

In [5]:

```
weekly = pd.read_sql(
    """
    select date, ticker, pb, marketcap, lastupdated from weekly
    where date>='2022-01-01'
    order by ticker, date, lastupdated
    """,
    conn,
)
weekly = weekly.groupby(["ticker", "date"]).last()
weekly = weekly.drop(columns=["lastupdated"])

pb = weekly.pb
pb.name = "pb"
marketcap = weekly.marketcap
marketcap.name = "marketcap"
```

In [6]:

```
sf1 = pd.read_sql(
    """
    select datekey as date, ticker, assets, netinc, equity, lastupdated from
    where datekey>='2022-01-01' and dimension='ARY' and assets>0 and equity>0
    order by ticker, datekey, lastupdated
    """,
    conn,
)
sf1 = sf1.groupby(["ticker", "date"]).last()
sf1 = sf1.drop(columns=["lastupdated"])

# change dates to Fridays
from datetime import timedelta
sf1 = sf1.reset_index()
sf1.date =sf1.date.map(
    lambda x: x + timedelta(4 - x.weekday())
)
sf1 = sf1.set_index(["ticker", "date"])
sf1 = sf1[~sf1.index.duplicated()]

assets = sf1.assets
assets.name = "assets"
netinc = sf1.netinc
netinc.name = "netinc"
equity = sf1.equity
equity.name = "equity"

equity = equity.groupby("ticker", group_keys=False).shift()
roe = netinc / equity
```

In [7]:

```
df = pd.concat(
    (
        mom,
        volume,
        price,
        pb,
        marketcap,
        roe,
        assetgr
        ),
        axis=1
    )
df["roe"] = df.groupby("ticker", group_keys=False).roe.ffill()
df["assetgr"] = df.groupby("ticker", group_keys=False).assetgr.ffill()

df = df.reset_index()
df.date = df.date.astype(str)
df = df[df.date==df.date.max()]
df = df[df.price >= 5]
df = df.dropna()

features = [
    "mom",
    "volume",
    "pb",
    "marketcap",
    "roe",
    "assetgr"
]
```

In [8]:

```
industries = pd.read_sql(
    """
    select ticker, famaindustry as industry from tickers
    """,
    conn,
)
industries["industry"] = industries.industry.fillna("Almost Nothing")
df = df.merge(industries, on="ticker", how="left")
df = df.dropna()
```

In [9]:

```
for x in features:
    df[f"{x}_industry"] = df.groupby(
        ["industry"],
        group_keys=False
    )[x].apply(
        lambda x: x - x.median()
    )

features += [f"{x}_industry" for x in features]
```

In [10]:

```
for f in features:
    df[f] = df[f].rank(pct=True)
```

Load Model and Predict

In [11]:

```
model = load("mymodel.joblib")
df["predict"] = model.predict(df[features])
```

Best and worst stocks

- Best stocks must be tradable
- Worst stocks must be tradable and shortable

In [12]:

```
with open("keys.txt", "r") as f:
    keys = f.readlines()

key, secret_key = [x.strip() for x in keys]
trading_client = TradingClient(key, secret_key, paper=True)

search_params = GetAssetsRequest(asset_class=AssetClass.US_EQUITY)
assets = trading_client.get_all_assets(search_params)
tradable = [x.symbol for x in assets if x.tradable]
shortable = [x.symbol for x in assets if x.shortable]
```

In [13]:

```
numstocks = 50

df = df.sort_values(by="predict", ascending=False)
best = df[["ticker", "predict"]].copy().reset_index(drop=True)
best = best[best.ticker.isin(tradable)].iloc[:numstocks]

df = df.sort_values(by="predict", ascending=True)
worst = df[["ticker", "predict"]].copy().reset_index(drop=True)
worst = worst[worst.ticker.isin(shortable)].iloc[:numstocks]
```

In [14]:
```
best
```

Out[14]:

	ticker	predict
0	SMCI	52.086654
1	COST	51.263532
2	ODFL	51.263532
3	CDNS	51.263532
4	VRSK	51.263532
5	CLX	51.263532
6	FAST	51.263532
7	SNPS	51.263532
8	ROL	51.263532
9	BAH	51.263532
10	AMT	51.263532
11	LULU	51.263532
12	CMG	51.257535
13	BX	51.256013
14	MPWR	51.255101
15	ANET	51.255101

In [15]:
```
worst
```

Out[15]:

	ticker	predict
2	EIGR	38.363996
6	PRPO	41.206560
12	BODY	43.961997
15	KPLT	44.359595
18	XOS	44.683329
19	CALC	44.738121
20	SKLZ	44.743086
22	ONCT	44.778882
27	AIRT	45.248374
29	TSE	45.379756
32	ECOR	45.510872
33	IPWR	45.621705
40	CRVO	45.809763
41	AEYE	45.823377
42	LEE	45.849742
44	CRIS	45.917552

Close unwanted positions

In [16]:

```
positions = trading_client.get_all_positions()
positions = {x.symbol: float(x.qty) for x in positions}
positions_to_close = [
    symbol for symbol in positions
    if (symbol not in best.ticker.to_list())
    and (symbol not in worst.ticker.to_list())
    and (symbol != "SPY")
]

for symbol in positions_to_close:
    qty = positions[symbol]
    order=MarketOrderRequest(
        symbol=symbol,
        qty=abs(qty),
        side=OrderSide.BUY if qty<0 else OrderSide.SELL,
        time_in_force=TimeInForce.DAY
    )
    _ = trading_client.submit_order(order)
```

Rebalance SPY

In [17]:

```
price = yf.download("SPY", start=2024, progress=False)["Close"].iloc[-1].item

account = trading_client.get_account()
equity = float(account.equity)
qty = int(equity / price)
qty -= positions["SPY"] if "SPY" in positions else 0

if qty != 0:
    order = MarketOrderRequest(
        symbol="SPY",
        qty=abs(qty),
        side=OrderSide.BUY if qty>0 else OrderSide.SELL,
        time_in_force=TimeInForce.DAY
    )
    _ = trading_client.submit_order(order)
```

Trade best stocks

In [18]:

```
symbols = best.ticker.to_list()
prices = yf.download(symbols, start=2024)["Close"].iloc[-1]
symbols = [s for s in symbols if not np.isnan(prices[s])]
dollars = 0.4 * equity / numstocks
for symbol in symbols:
    price = prices[symbol]
    qty = int(dollars / price)
    qty -= positions[symbol] if symbol in positions else 0
    if qty != 0:
        try:
            order = MarketOrderRequest(
                symbol=symbol,
                qty=abs(qty),
                side=OrderSide.BUY if qty>0 else OrderSide.SELL,
                time_in_force=TimeInForce.DAY
            )
            _ = trading_client.submit_order(order)
        except Exception as error:
            print("An error occurred:", error)
```

```
[**********100%%**********]  50 of 50 complete
d
An error occurred: {"buying_power":"60.71","code":40310000,"cost_basi
s":"794.16","message":"insufficient buying power"}
An error occurred: {"buying_power":"60.71","code":40310000,"cost_basi
s":"762","message":"insufficient buying power"}
An error occurred: {"buying_power":"60.71","code":40310000,"cost_basi
s":"787","message":"insufficient buying power"}
An error occurred: {"buying_power":"60.71","code":40310000,"cost_basi
```

Trade worst stocks

In [19]:

```
symbols = worst.ticker.to_list()
prices = yf.download(symbols, start=2024)["Close"].iloc[-1]
symbols = [s for s in symbols if not np.isnan(prices[s])]
for symbol in symbols:
    price = prices[symbol]
    qty = - int(dollars / price)
    qty -= positions[symbol] if symbol in positions else 0
    if qty != 0:
        try:
            order = MarketOrderRequest(
                symbol=symbol,
                qty=abs(qty),
                side=OrderSide.BUY if qty>0 else OrderSide.SELL,
                time_in_force=TimeInForce.DAY
            )
            _ = trading_client.submit_order(order)
        except Exception as error:
            print("An error occurred:", error)
```

```
[**********************100%%**********************]  50 of 50 complete
d
An error occurred: {"buying_power":"405.43","code":40310000,"cost_bas
is":"819.18","message":"insufficient buying power"}
```

Save data

In [20]:

```
today = datetime.strftime(datetime.today(), "%Y-%m-%d")
account = trading_client.get_account()
equity = float(account.equity)
if os.path.isfile("equity.csv"):
    d = pd.read_csv("equity.csv", index_col="date")
    d.loc[today] = equity
else:
    d = pd.Series({today: equity})
    d.name = "equity"
    d.index.name = "date"
d.to_csv("equity.csv")
```

In [21]:

```
positions = trading_client.get_all_positions()
d = pd.DataFrame([x.qty for x in positions], index=[x.symbol for x in positio
d["date"] = today
d.index.name = "symbol"
d = d.reset_index()
if os.path.isfile("positions.csv"):
    d0 = pd.read_csv("positions.csv")
    d = pd.concat((d0, d))
d.to_csv("positions.csv", index=False)
```