Applying a Random Forest

MGMT 638: Data-Driven Investments: Equity

Kerry Back, Rice University

Outline

- Create current features:
    - Get data from SQL library.
    - We only want most recent data, but go back a couple of years to compute momentum, growth rates, etc.
    - Follow same procedure as in 5a-fundamentals.ipynb, but do not shift momentum, volatility, etc. forward.
    - And do not keep return (return for prior week is not useful)
- Apply saved random forest model to current data to form future predictions.
- Use predictions to identify best and worst stocks today (maybe sector neutral).

Create connection

In [18]:

```python
import pandas as pd

from sqlalchemy import create_engine
import pymssql
server = 'fs.rice.edu'
database = 'stocks'
username = 'stocks'
password = '6LAZH1'
string = "mssql+pymssql://" + username + ":" + password + "@" + server + "/"
conn = create_engine(string).connect()
```

```
Exception during reset or similar
Traceback (most recent call last):
  File "c:\Users\kerry\AppData\Local\Programs\Python\Python310\lib\si
te-packages\sqlalchemy\pool\base.py", line 753, in _finalize_fairy
    fairy._reset(pool)
  File "c:\Users\kerry\AppData\Local\Programs\Python\Python310\lib\si
te-packages\sqlalchemy\pool\base.py", line 1004, in _reset
    pool._dialect.do_rollback(self)
  File "c:\Users\kerry\AppData\Local\Programs\Python\Python310\lib\si
te-packages\sqlalchemy\dialects\mssql\base.py", line 2792, in do_roll
back
    super(MSDialect, self).do_rollback(dbapi_connection)
  File "c:\Users\kerry\AppData\Local\Programs\Python\Python310\lib\si
te-packages\sqlalchemy\engine\default.py", line 683, in do_rollback
    dbapi_connection.rollback()
  File "src\pymssql\_pymssql.pyx", line 316, in pymssql._pymssql.Conn
ection.rollback
  File "src\pymssql\_pymssql.pyx", line 300, in pymssql._pymssql.Conn
```

Calculate financial ratios and growth rates

Data from SF1

```
In [19]: sf1 = pd.read_sql(
             """
             select ticker, datekey, lastupdated, netinc, ncfo, equity, assets
             from sf1
             where dimension='ARQ' and datekey>='2021-01-01' and equity>0 and assets>0
             order by ticker, datekey
             """,
             conn,
             parse_dates=["datekey"]
         )
         sf1 = sf1.groupby(["ticker", "datekey", "lastupdated"]).last()
         sf1 = sf1.droplevel("lastupdated")
         sf1 = sf1.reset_index()
```

```
In [ ]: for col in ["netinc", "ncfo"]:
            sf1[col] = sf1.groupby("ticker", group_keys=False)[col].apply(
                lambda x: x.rolling(4).sum()
            )
        for col in ["equity", "assets"]:
            sf1[col] = sf1.groupby("ticker", group_keys=False)[col].apply(
                lambda x: x.rolling(4).mean()
            )
        sf1["roe"] = sf1.netinc / sf1.equity
        sf1["accruals"] = (sf1.netinc - sf1.ncfo) / sf1.equity
        sf1["agr"] = sf1.groupby("ticker", group_keys=False)["assets"].pct_change()
        sf1 = sf1[["ticker", "datekey", "roe", "accruals", "agr"]].dropna()
```

Returns, volume, momentum, volatility

Data from sep_weekly

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

In [ ]:

```
sep_weekly["ret"] = sep_weekly.groupby("ticker", group_keys=False).closeadj.p
sep_weekly["annual"] = sep_weekly.groupby("ticker", group_keys=False).closead
sep_weekly["monthly"] = sep_weekly.groupby("ticker", group_keys=False).closea
sep_weekly["mom"] = sep_weekly.groupby("ticker", group_keys=False).apply(
    lambda d: (1+d.annual)/(1+d.monthly) - 1
)
sep_weekly["volatility"] = sep_weekly.groupby("ticker", group_keys=False).ret
    lambda x: x.rolling(26).std()
)
sep_weekly = sep_weekly[["mom", "volume", "volatility", "closeunadj"]]
sep_weekly = sep_weekly.reset_index()
```

Get marketcap and pb

Data from weekly

```
In [ ]: weekly = pd.read_sql(
            """
            select ticker, date, marketcap, pb, lastupdated
            from weekly
            where date>='2022-01-01' and marketcap>0 and pb>0
            order by ticker, date, lastupdated
            """,
            conn,
            parse_dates=["date"]
        )
        weekly = weekly.groupby(["ticker", "date", "lastupdated"]).last()
        weekly = weekly.droplevel("lastupdated")
        weekly = weekly.reset_index()
```

Merge

```
In [ ]: df = weekly.merge(sep_weekly, on=["ticker", "date"], how="inner")
        df["year"] = df.date.apply(lambda x: x.isocalendar()[0])
        df["week"] = df.date.apply(lambda x: x.isocalendar()[1])
        sf1["year"] = sf1.datekey.apply(lambda x: x.isocalendar()[0])
        sf1["week"] = sf1.datekey.apply(lambda x: x.isocalendar()[1])
        df = df.merge(sf1, on=["ticker", "year", "week"], how="left")
        df = df.drop(columns=["year", "week", "datekey"])
```

Fill ratios and growth rates forward

```
In [ ]: for col in ["roe", "accruals", "agr"]:
            df[col] = df.groupby("ticker", group_keys=False)[col].apply(
                lambda x: x.ffill()
            )
```

Add sector data

```
In [ ]: tickers = pd.read_sql(
            """
            select ticker, sector from tickers
            """,
            conn
        )
        df = df.merge(tickers, on="ticker")
```

Filter to today’s data

```
In [ ]: df = df[df.date==df.date.max()].copy()
```

Filter to small caps and exclude penny stocks

In [ ]:

```
df = df[df.closeunadj>5]
df = df.dropna()
df["rnk"] = df.marketcap.rank(
    ascending=False,
    method="first"
)
df = df[(df.rnk>1000) & (df.rnk<=3000)]
df = df.drop(columns=["closeunadj", "rnk"])
```

Define features

In [ ]:

```python
features = [
    "marketcap",
    "pb",
    "mom",
    "volume",
    "volatility",
    "roe",
    "accruals"
]
```

Make predictions

```
In [ ]: # change this to "./" if forest.joblib is in your working directory
        path_to_file = "../../"

        from joblib import load
        forest = load(path_to_file + "forest.joblib")
        df["predict"] = forest.predict(X=df[features])
```

Find best and worst stocks

```
In [ ]: df["rnk_long"] = df.predict.rank(
            ascending=False,
            method="first"
        )
        df["rnk_short"] = df.predict.rank(
            ascending=True,
            method="first"
        )
        longs = df[df.rnk_long<=44]
        shorts = df[df.rnk_short<=44]
```

Sector-neutral version

In [ ]:

```
df["rnk_long"] = df.groupby("sector", group_keys=False).predict.rank(
    ascending=False,
    method="first"
)
df["rnk_short"] = df.groupby("sector", group_keys=False).predict.rank(
    ascending=True,
    method="first"
)
longs_neutral = df[df.rnk_long<=4]
shorts_neutral = df[df.rnk_short<=4]
```

Save results

In [ ]:

```
with pd.ExcelWriter("portfolios 2023-11-08.xlsx") as writer:
    longs.to_excel(writer, "long", index=False)
    shorts.to_excel(writer, "short", index=False)
    longs_neutral.to_excel(writer, "long neutral", index=False)
    shorts_neutral.to_excel(writer, "short neutral", index=False)
    df.to_excel(writer, "today", index=False)
```