Backtesting a Random Forest

MGMT 638: Data-Driven Investments: Equity

Kerry Back, Rice University

Outline

- Read data saved in 05a-fundamentals.ipynb
- Loop over specified training dates (e.g., once per year)
    - At each training date, train the random forest on prior data
    - Use the trained model to make predictions until the next training date
- Use the predictions to form portfolios
    - Best and worst stocks each week
    - Best and worst stocks in each sector each week
- Compare returns of equally weighted portfolios (long and short)

- This is what we did last week, except
- Instead of combining value and momentum ranks, we make predictions based on
    - the model trained on prior data
    - more characteristics: marketcap, volume, volatility, ...

Read data

In [44]:
```
import pandas as pd

# change path_to_file to "./" if the file is in your working directory
path_to_file = "../../"

df = pd.read_csv(path_to_file + "data-2023-11-08.csv")
df.head()
```

Out[44]:

	ticker	date	marketcap	pb	ret	mom	volume	volatility
0	AACC	2011-01-14	188.3	1.4	-0.014634	-0.184615	2.078000e+04	0.071498
1	AAI	2011-01-14	1012.1	2.0	0.002677	0.438224	2.775580e+06	0.128450
2	AAIC	2011-01-14	189.3	1.0	-0.010119	0.684547	3.466000e+04	0.048505
3	AAON	2011-01-14	479.4	4.2	0.007778	0.528685	2.817291e+05	0.044912
4	AATC	2011-01-14	63.3	1.4	-0.013960	0.008216	6.800000e+03	0.049756

Define model and target

In [45]:

```
from sklearn.ensemble import RandomForestRegressor
forest = RandomForestRegressor(max_depth=3)

df["target"] = df.groupby("date", group_keys=False).ret.apply(
    lambda x: x - x.median()
)
```

Define predictors (features)

In [46]:

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

Define training dates and training windows

- For this example, I am going to train once per year using the prior three years of data.
- Obviously, other choices are possible.
- The reason for not using all past data is to capture any changes in the market.

In [47]:

```
dates = list(df.date.unique())
dates.sort()
train_dates = dates[156::52]  # once per year starting after three years

past_dates = {}               # dates on which to train for each training dat
future_dates = {}             # dates for which to predict for each training
for date in train_dates:
    past_dates[date] = dates[(dates.index(date)-156):dates.index(date)]
    if date < train_dates[-1]:
        future_dates[date] = dates[dates.index(date):(dates.index(date)+52)]
    else:
        future_dates[date] = dates[dates.index(date):]
```

Run the loop

In [48]:

```
new_data = None
for date in train_dates:
    past = past_dates[date]
    past = df[df.date.isin(past)]
    future = future_dates[date]
    future = df[df.date.isin(future)]
    forest.fit(X=past[features], y=past.target)
    predictions = forest.predict(X=future[features])
    predictions = pd.DataFrame(predictions)
    predictions.columns = ["predict"]
    for col in ["ticker", "date"]:
        predictions[col] = future[col].to_list()
    new_data = pd.concat((new_data, predictions))

df = df.merge(new_data, on=["ticker", "date"], how="inner")
```

In [49]:
```
df.tail()
```

Out[49]:

	ticker	date	marketcap	pb	ret	mom	volume	volatility
1010376	ZNTL	2023-11-06	1262.5	2.4	-0.302013	-0.174662	743655.8	0.086553
1010377	ZUMZ	2023-11-06	335.0	0.9	-0.023063	-0.245402	201904.4	0.053633
1010378	ZUO	2023-11-06	1088.9	9.7	-0.011613	0.080163	662494.2	0.070317
1010379	ZYME	2023-11-06	504.0	1.1	-0.020188	-0.215539	435386.8	0.062766
1010380	ZYXI	2023-11-06	310.4	5.3	0.014746	-0.356304	379338.0	0.066363

Form portfolios and compute returns

In [50]:

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

In [51]:
```
long_ret = longs.groupby("date").ret.mean()
short_ret = shorts.groupby("date").ret.mean()
print(f"mean annualized long return is {52*long_ret.mean():.2%}")
print(f"mean annualized short return is {52*short_ret.mean():.2%}")
```

```
mean annualized long return is 35.70%
mean annualized short return is -12.52%
```

Try sector-neutral strategy

In [52]:

```
df["rnk_long"] = df.groupby(["date", "sector"], group_keys=False).predict.ran
    ascending=False,
    method="first"
)
df["rnk_short"] = df.groupby(["date", "sector"], group_keys=False).predict.ra
    ascending=True,
    method="first"
)
longs = df[df.rnk_long<=4]
shorts = df[df.rnk_short<=4]
```

In [53]:
```
long_neutral_ret = longs.groupby("date").ret.mean()
short_neutral_ret = shorts.groupby("date").ret.mean()
print(f"mean annualized long sector-neutral return is {52*long_neutral_ret.me
print(f"mean annualized short sector-neutral return is {52*short_neutral_ret.
```

```
mean annualized long sector-neutral return is 33.90%
mean annualized short sector-neutral return is -7.34%
```

Plot long-minus-short returns

In [54]:

```
lms = long_ret - short_ret
lms_neutral = long_neutral_ret - short_neutral_ret

lms.index = pd.to_datetime(lms.index)
lms_neutral.index = pd.to_datetime(lms_neutral.index)

import matplotlib.pyplot as plt
import seaborn as sns
sns.set_style("whitegrid")

(1+lms).cumprod().plot(logy=True, label="long minus short")
(1+lms_neutral).cumprod().plot(logy=True, label="neutral long minus short")
plt.legend()
plt.show()
```