DS-GA 3001:
Advanced Python For Data Science

Stock Portfolio Selection --- CORN

Group 07:

Wang, Yakun: yw3918@nyu.edu
Wang, Ziwei: zw1365@nyu.edu
Shen, Yulin: ys2542@nyu.edu
Zhou, Yihong: yz4552@nyu.edu
Zhou, Yi: yz4525@nyu.edu

CORN-Algorithm

- An agorithm uses **statistcal correlations** between **market windows** in the historical stock market to train **multiple experts**.

$$C_t(w,\rho) = \{w < i < t-1, \frac{Cov(\boldsymbol{X}_{\boldsymbol{i-w}}^{\boldsymbol{i-1}}, \boldsymbol{X}_{\boldsymbol{t-w}}^{\boldsymbol{t-1}})}{std(\boldsymbol{X}_{\boldsymbol{i-w}}^{\boldsymbol{i-1}})std(\boldsymbol{X}_{\boldsymbol{t-w}}^{\boldsymbol{t-1}})} >= \rho\}$$

- Each expert will output a **portfolio** for trading each day by sloving a **optimization problem.**

$$b_t(w,\rho) = \underset{b \in \triangle_m}{\operatorname{argmax}} \prod_{i \in C_t(w,\rho)} (b \cdot x_i)$$

- Final portfolio for each day is achieved by **combining** all “suggesions” from expert.

General Program Structure

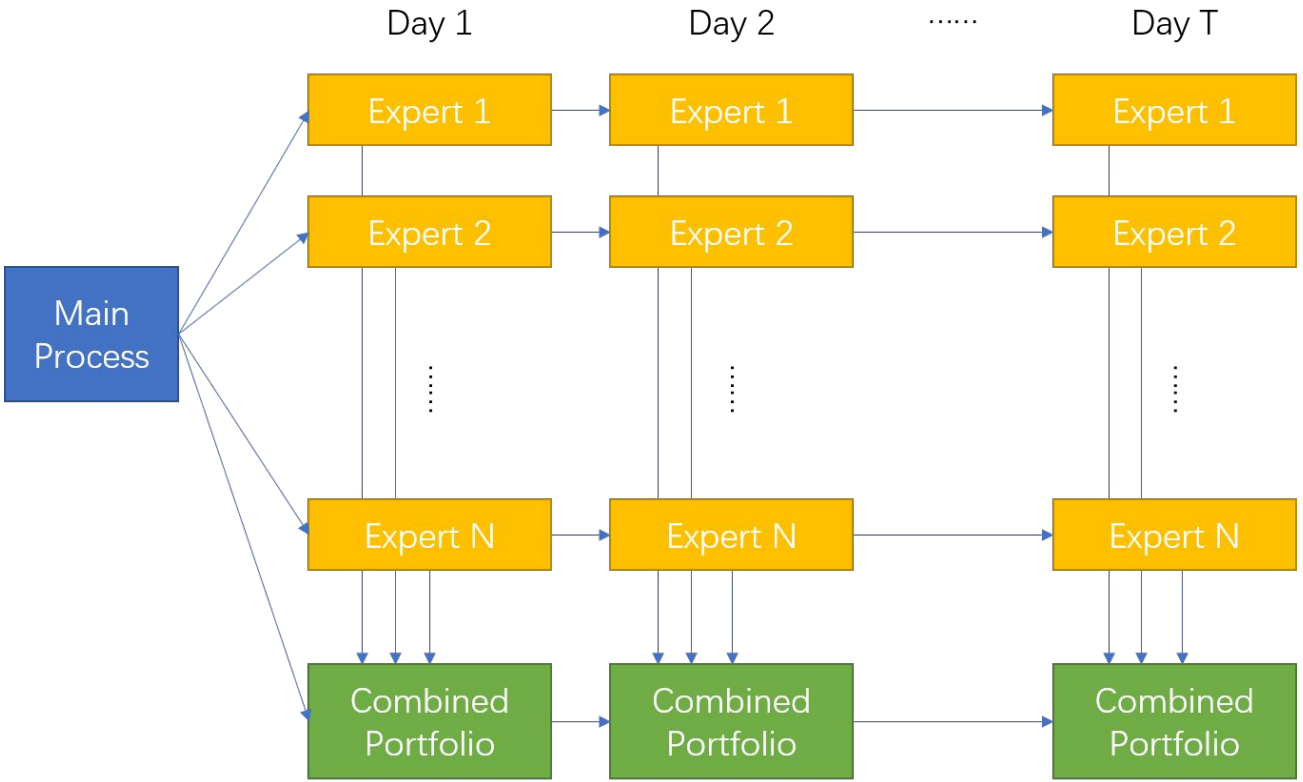
- Experts learn independently
- Experts specified by ε(w,ρ)

- Encapsulate learning and updating process in Class expert

- CORNK needs to select best K experts every day

- Sortable expert objects through __eq__ and __lt__ methods

General Program Structure



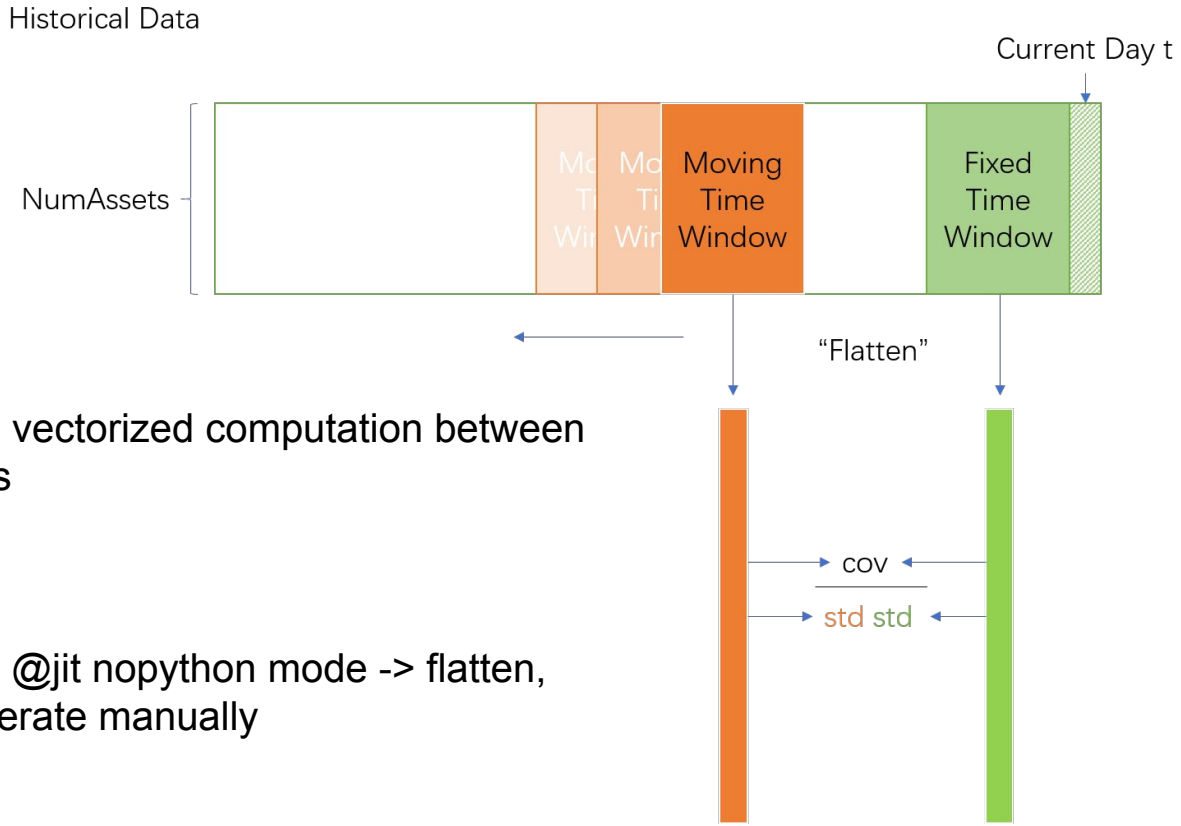
Line Profilie Benchmark

- Total Running Time: **636** seconds (~ 10 mins) for 1000 days of 36 stocks using Cython.
- Details: **99%** of time was spent in experts' learning progress:
  - ➢ 75% in computing correlation set + 20% in solving optimization problems

Optimization Big Picture:

- Individual Expert's Optimal Portfolio → **Log Transform**
- Correlation Matrix → **Numba**
- Multiple Experts' learning progress → **Multiprocessing**

Difficulty 1: Effiency of Correlation Matrix Computation



- numpy.corrcoef() -> vectorized computation between **every two** elements
- ~30.1ms

- Fit well with Numba @jit nopython mode -> flatten, vectorize and accelerate manually
- Finally ~1.08ms

Difficulty 2: Optimization Solvers

- General optimization solver: Scipy with SLSQP method
- Improvement: Reformulate problem
  - From: An ill-conditioned convex optimization problem

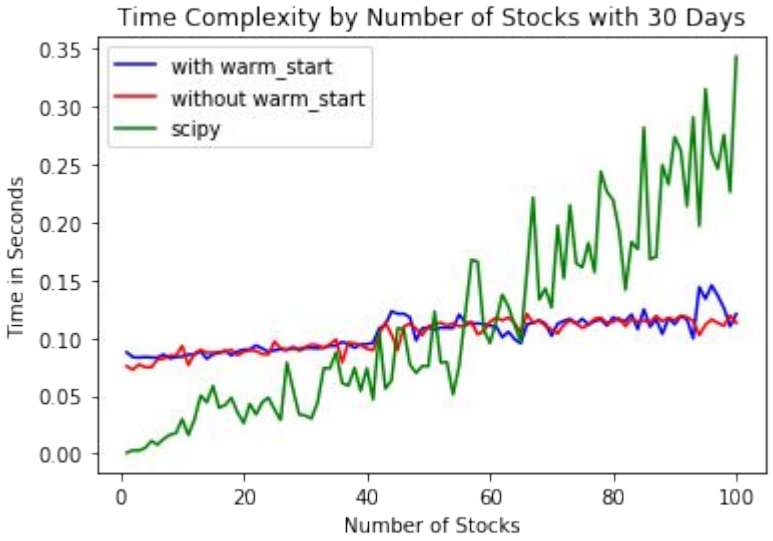
$$b_t(w, \rho) = \underset{b \in \triangle_m}{\operatorname{argmax}} \prod_{i \in C_t(w,\rho)} (b \cdot x_i)$$

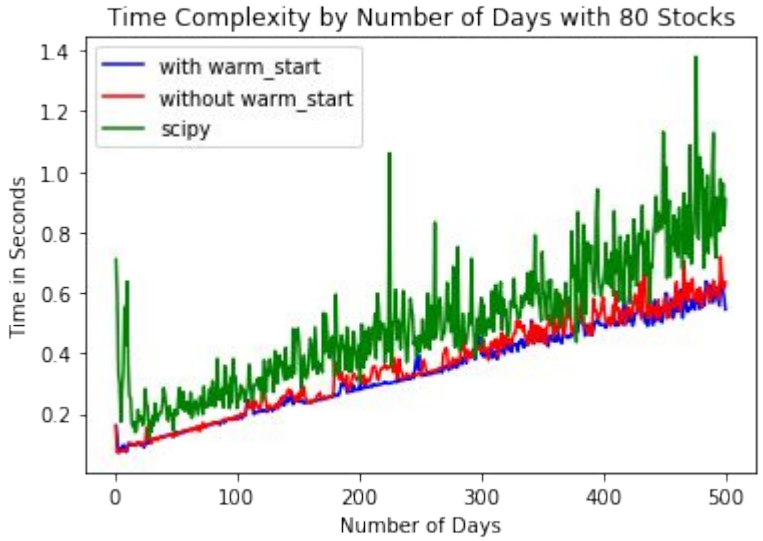
  - To: Disciplined convex programming (DCP) problem

$$\log(b_t(w, \rho)) = \underset{b \in \triangle_m}{\operatorname{argmax}} \sum_{i \in C_t(w,\rho)} [\log(b \cdot x_i)]$$

- Convex optimization solver: Cvxpy with SCS method

Solvers Performance Comparison





- Scipy: both O(n) and unstable in solving
- Cvxpy: almost O(1) and O(n)
    - warm_start option in Cvxpy help little in the problem

Difficulty 3: Parallel programming

- Multiprocessing:

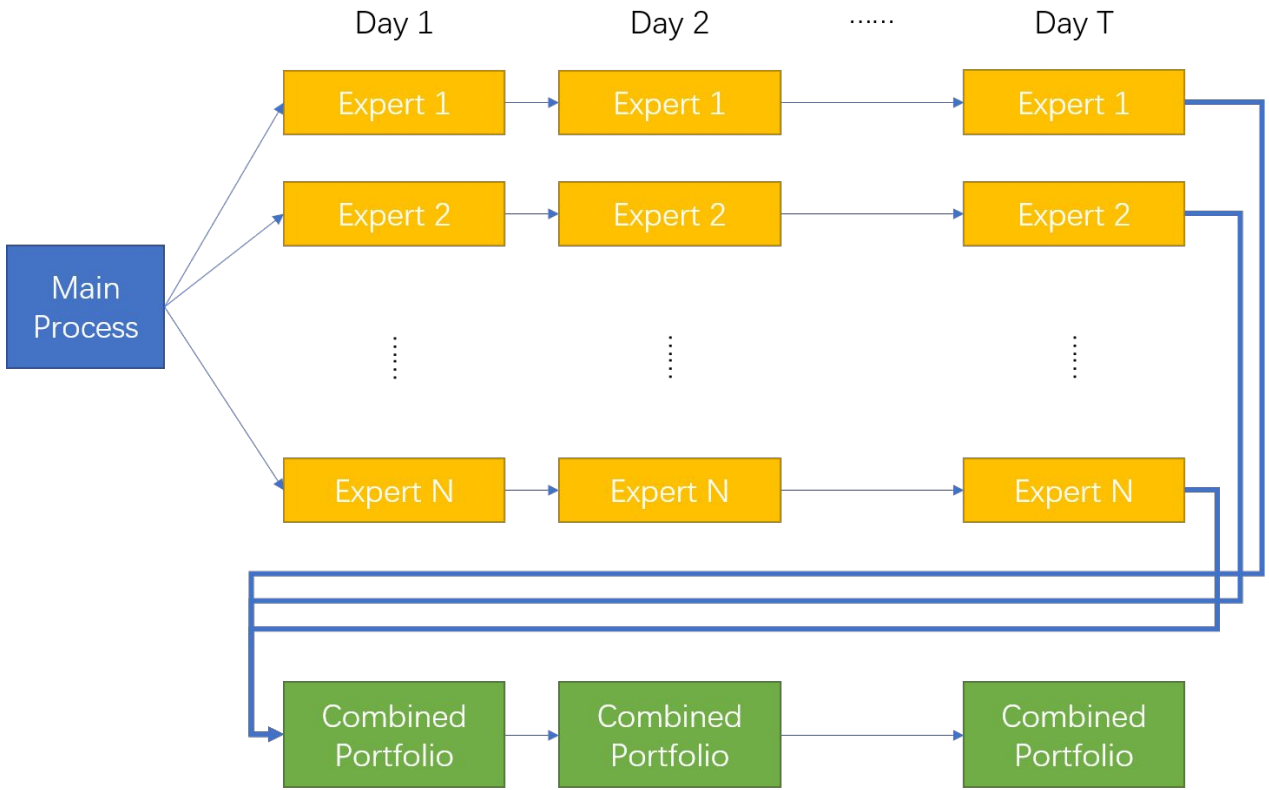
  1. parallel the step of updating experts' portfolio in each day

     heavy overhead of distributing Pool

  2. parallel computing each experts' portfolio from time 1 to T

     If T is large, seperate T into chunks

Difficulty 3: Parallel programming



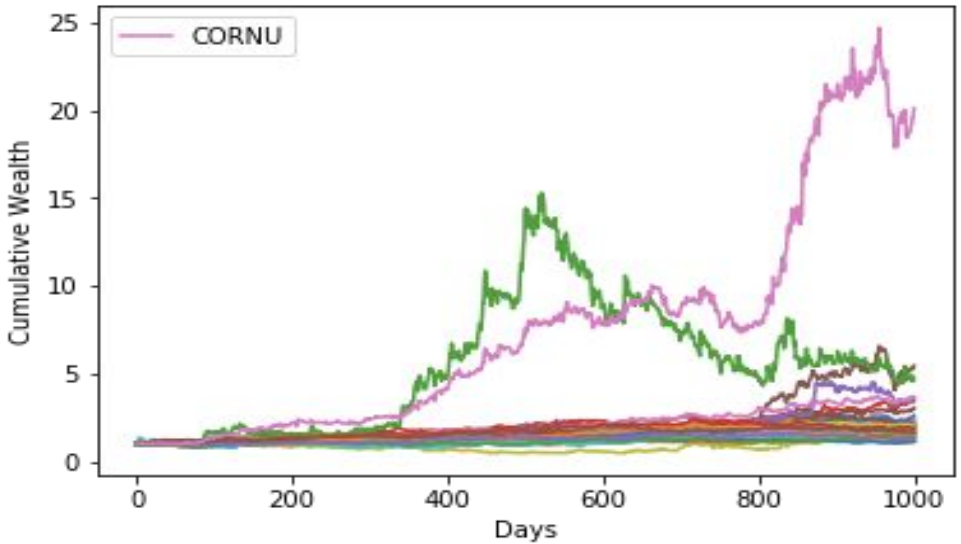
Multiprocessing’ s result (w = 5, rho = 0.4, T = 500)

Approach Used	Running time
Without acceleration	82.2s
with expensive multiprocessing	1737.3s
with MPI	49.8 s
with multiprocessing one time	20.0s

Conclusion:

After 3 major optimizations, a larger dataset with T = 1000:

- New program takes **55.8** seconds.
  - ➢ > **10** times shorter compared with the **636.0** seconds.



Thank you! :)