Project Summary

In this project, I perfrom the Upper Confidence Bound RL Algorithm to optimally pick one equity from each listed S&P 500 sector. Initialize a certain amount of money (initialized by parameter "start_money") into each sector. Then, on each trading day, the model will allocate all of its money in each sector to a single equity picked out by the UCB algorithm. While the money initialized to each sector at the beginning of the process is the same, it will not stay that way as each individual sector realizes varying returns. This dynamic distribution of funds to multiple sectors ensures portfolio diversification and risk mitigation, as multiple stocks across multiple sectors will always be invested in to reduce unsystematic risk, but the worse performing sectors have less money to invest. At the beginning of the next trading day, the position on each equity is sold, and the returns from each individual stock are reinvested in the stock picked by the algorithm for the next day. The UCB algorithm picks the optimal stock by the follwing formula:

For each day t ans sector s:

$$stock_{t,s} = argmax_i(\hat{\mu}_i + UCB_i)$$

Where
$$UCB_i = \left\{egin{array}{ll} \sqrt{rac{lpha ln(t)}{2N_t(i)}} & ext{if } N_i(t) > 0 \ 1 & ext{else} \end{array}
ight.$$

and $\hat{\mu}_i$ is the sample mean of the return of each stock, α is a hyperparemter responsible for exploration(higher α leads to greater exploration), and $N_t(i)$ represents the number of times equity i has been selected to be invested in by time t.

The percent retuens of each sector are aggregated to demonstrate a total annualized return.

This specific backtest returns 14.4% annually, and is tested over the period of 2010-01-01 through 2019-12-31. It is one of many backtests conducted in this project by the club.

Prepare Data

```
import pandas as pd
import numpy as np
import yfinance
import matplotlib.pyplot as plt
from matplotlib.ticker import PercentFormatter
from sec import constants, stock, lookups, processor
```

```
In [ ]: # Get SP500 tickers
        ## This is a library created by the GTSF Investments Committee, can be found on the committee GitHub
        sp500 = lookups.get sp500 tickers()
In [ ]: #Download data from Yfinance
        all data = pd.DataFrame()
        for ticker in sp500:
             data = yfinance.download(ticker, interval = '1d', period='15y')
            if ticker == sp500[0]:
                 all_data.index = data.index
             ## Drop all stocks that were not in S&P before 2010 (beginning of this backtest)
             if len(data.index) < 3774:</pre>
                 continue
             all_data[ticker] = data.iloc[:, 0].values
In [ ]: # Get data in 2010s
         all_data.index = pd.to_datetime(all_data.index)
         all_data = all_data[(all_data.index.year > 2009) & (all_data.index.year < 2020)]
         #Reset data index - useful for UCB time steps
         all_data.index = range(len(all_data.index))
In [ ]: #Change columns from $ return to pct return
        for col in all_data.columns:
             all_data[col] = all_data[col].pct_change()
        #Drop first row (no pct return)
         all data = all data.drop(0, axis=0)
In [ ]: #Scrape data containing ticker sectors
        wiki = pd.read html('https://en.wikipedia.org/wiki/List of S%26P 500 companies')[0]
        wiki = wiki.set_index('Symbol')
In [ ]: #combine data sources
         sectors = {sector: [] for sector in wiki['GICS Sector'].value_counts().index}
        for ticker in all data.columns:
            try:
                 sector = wiki.loc[ticker, 'GICS Sector']
                 sectors[sector].append(ticker)
             except KeyError:
                 continue
         sector_dfs = {sector: all_data[sectors[sector]] for sector in sectors.keys()}
```

Implement Algorithm

```
In [ ]: def get_UCB_arm(Q: np.array, N: np.array, t: int, alpha: float)-> int:
             '''Calculates the decision at time t.
               Q: Array of sample means.
               N: Array containing number of times each arm has been pulled.
               t: Current Time Step.
                alpha: Value of hyperparameter alpha of UCB algorithm. Higher alpha -> more exploration.
               Return: Index of stock to invest in on day t '''
            #Calculate exploration bonuses
            for arm in range(len(Q)):
                if N[arm] == 0:
                  explo_bonus = 1
               else:
                  explo bonus = np.sqrt(-0.5*alpha*np.log(t)/N[arm])
           # Choose argmax(sample mean + explo bonus)
            arm_vals = [Q[num] + explo_bonus for num in range(len(Q))]
            return np.argmax(arm_vals)
        def UCB(alpha:float, sector name: str, start money: float, time horizon = len(all data.index)) -> float:
          ''' Implements UCB Algorithm.
              alpha: Value of hyperparameter alpha of UCB algorithm. Higher alpha -> more exploration.
              sector name: Name of sector to implement algorithm on.
              start money: Amount of money to invest in each sector.
              time horizon: If you only want to implement the algo up to a certain time t.
              Return: Total money returned over duration of investment period '''
          ## Initializations
          sector_data = sector_dfs[sector_name]
          num_stocks_in_sector = len(sector_data.columns)
          N = np.zeros(num stocks in sector)
          Q = np.ones(num stocks in sector)
          starting_money = start_money*np.ones(num_stocks_in_sector)
          ## Get sector index to adjust money allocated to sector
          sector index = sector data.keys().index(sector name)
          for t in range(1, time horizon):
              arm = get_UCB_arm(Q, N, alpha, t)
              #Increment by 1 if arm pulled
              N[arm] += 1
              # Get reward
              reward = sector_data.iloc[t, arm]
              #Re-invest all sector money, calculate next day money
```

```
#Subtract money if negative return
              if reward < 0:</pre>
                 start_money[sector_index] = start_money[sector_index]*(1-abs(reward))
                 #Do nothing if 0 pct change
              elif reward - 1 == 0:
                 start_money[sector_index] = start_money[sector_index]
                 #calculate new money if positive chnq
              else:
                start money[sector index] = start money[sector index]*(reward + 1)
              # Update sample mean of the pulled arm based on reward
              Q[arm] = (1-1/N[arm])*Q[arm] + reward/N[arm]
          return start_money
In [ ]:
        #Calculate cumulative return
        money = []
        starting_money = 1000
        for sector, df in list(sector_dfs.items()):
            port_val = UCB(1, sector, starting_money)
            money.append(port_val)
        pnl = (sum(money) - starting_money*11)
        print(f'Cumulative Return: {(pnl)/(11*starting_money)}%')
        print('----')
        print(f'Annualized Return: {(pnl)/(11*starting_money*10)}%')
        Cumulative Return: 143.553248647227%
        Annualized Return: 14.355324864722702%
```

Visualize Returns

```
In []: ## Run algorithm for al time horizons < T (10 years)
money_by_time = []
for t in range(1, len(all_data.index), 14):
    money = []
    for sector, df in list(sector_dfs.items()):
        pnl = UCB(0.01, sector, starting_money, time_horizon=t)
        money.append(pnl)
    money_by_time.append(20*sum(money)/(11*starting_money))</pre>
```

```
In [ ]: # Plot results
fig, ax = plt.subplots()
plt.plot(list(range(1, len(all_data.index), 14)), money_by_time)
```

```
plt.title('UCB Trading Strategy % Return vs. Trading Days')
plt.xlabel('Trading Days')
plt.ylabel('Portfolio Value')
ax.yaxis.set_major_formatter(PercentFormatter())
plt.show()
```

