Final Project

Nikesh Mishra
Spring 2021 ECON 46 - Jackson

Portfolio Management Through a Sector-Wise Analysis of Exchange-Traded Fund Centrality

I. Introduction

An exchange-traded fund (commonly abbreviated as "ETF") is a financial security that tracks a basket of related stocks but can be bought and sold on an exchange like any other asset (Chen 2021). The purpose of ETFs is to allow investors to purchase a security that performs like a given group of stocks (often referred to as the ETF's "holdings" or "basket") without needing to purchase any stocks in this group; this also lowers investor risk. In this project, we investigate the relationships between stocks held by ETFs, and see if any of the information gained can be helpful in trading securities on financial networks.

We will restrict our analysis to the study of centrality in the stock correlation networks derived from an ETF's holdings. In the study of networks, centrality is a characteristic of a vertex that denotes how "important" that vertex is in the network. Centrality is a highly relevant concept in the study of many different classes of data, including information about social relationships, epidemiological records, information packets flowing through the internet, and financial data. Stock correlation networks are, as their name might suggest, a type of financial network that derive their vertex set from a given collection of stocks, and their edge set from the correlation between the values of those stocks. This project will analyze how centrality of stocks within ETF stock correlation networks affects achieved returns.

II. Prior Work

There has been extensive study of stock correlation networks and their relationships to the performance of assets traded on public market exchanges. Some of this work has also considered how centrality affects general market performance – in 2002, Onnela and colleagues showed that a minimum spanning tree which originates from a central node and derives its edge set from a stock correlation network shrinks during a stock market crash (Onnela 2002). Onnela et al.'s

work is just one example of the existing work on this topic; a query to Google Scholar with the term “stock correlation network” yields more than 2,000,000 results.

An interesting extension of the proven relationships between centrality, stock correlation networks, and larger patterns in the market is the application of these networks to financial trading strategies. ETFs are an ideal target for comparing the returns of central stocks to parent assets, as they often focus on certain sectors or groups of companies that face similar market conditions. In this project, we will study the performance of central stocks relative to their parent ETFs over a one-year period, with the goal of determining if purchasing an ETF’s most central stock leads to better returns than purchasing a share of the ETF in the following year.

III. Methods

First, a group of ETFs was randomly selected. For each ETF, the stock correlation network was created, and the most central stock was identified. After this, the performance of each ETF and its most central stock was compared during the 2019 calendar year (1/1/19 - 12/31/19). Of the stock-ETF pairings where the stock yielded greater returns than the ETF during 2019, both the stock and ETF were simulated as being “purchased” on January 1st, 2020, and “sold” on December 31st, 2020. Ten ETFs were examined in this simulation. Due to limitations in data collection capabilities, only the American holdings of ETFs were considered (to account for this constraint, random selection of ETFs was from a database of US-centric funds). Furthermore, only assets that comprised more than 0.5% of an ETF’s holdings were considered – many funds trade miniscule amounts of interest owed to banks and other financial services companies. Since these are not stocks, they were discarded for the purposes of this experiment.

The code used to produce the simulation was written entirely in Python. As a result of the weak computing resources available to the author, all code in this project was written, deployed, and run on a Google Colab notebook. This service offers an exact clone of the interface provided in the very popular Jupyter notebook platform hosted on the internet; this leads to more efficient project management and offloads all computation onto Google’s servers, allowing for much faster computation times than could be achieved on the author’s personal computer. The entirety

of the simulation is automated; the only input necessary was a collection of ETF ticker symbols, which were provided in the form a Python list.

The Pandas software library was used to clean and manage data (McKinney 2010; Pandas Dev 2020). This object-oriented library offers a suite of tools for data manipulation and analysis and is widely used in academic and industrial data science projects. The NetworkX library was used to manipulate and manage the virtual representation of the stock correlation networks (Hagberg et al. 2008). Both Pandas and NetworkX are open-source software.

The ten randomly selected ETFs came from 3 sectors – energy, industrials, and financial services; three energy ETFs, three industrial ETFs, and three financial ETFs comprised the collection of examined funds. The following funds were examined: BlackRock's iShares Global Clean Energy ETF (ICLN), VanEck's Vectors Uranium + Nuclear Energy ETF (NLR), State Street's S&P Oil & Gas Exploration & Production ETF (XOP), State Street's Financial Select Sector SPDR Fund (XLF), Vanguard's Financials ETF (VFH), State Street's S&P Capital Markets ETF (KCE), Vanguard's Industrials ETF(VIS), BlackRock's iShares U.S. Industrials ETF (IYJ), BlackRock's iShares U.S. Aerospace & Defense ETF (ITA), and WBI's BullBear Value 3000 ETF (WBIF).

Once the ETFs were chosen, the first step of the simulation was to identify a given fund's basket of holdings. Unfortunately, this proved to be an extremely challenging process, as this information was not freely available anywhere. However, the author was able to identify an unsecured hyperlink to an ETF screener published by Fidelity Investments (Fidelity 2021). The site was constructed such that all the necessary data was stored in an HTML table element, a characteristic that lent itself to a Pandas-based web scraping procedure. As mentioned above, only American stocks that comprised more than 0.5% of a fund's portfolio were recorded as being components of an ETF's holdings. Once collected, this information was stored in a Python list.

After an ETF's basket of holdings was identified, the list of assets was passed to a function that identified each stock's adjusted closing price between two specified dates. The adjusted closing

price of a stock is the price of the asset when trading stops at the end of the business day, accounting for changes by the corporation selling that stock, such as “stock splits, dividends, and rights offerings” (Ganti 2020). Historical performance data was acquired with the Pandas DataReader library (PyData Dev 2020). Internally, this library utilizes the Yahoo Finance application programming interface (API), which allows users to query information about stocks listed on American exchanges (the geopolitical limitation of this API is the reason why the experiment is restricted to the American holdings of ETFs). Once the appropriate data was gathered, it was stored in a Pandas DataFrame, which is the standard two-dimensional data structure used for information storage in the Pandas library. A sample of the DataFrame of is shown in the image below. This data comes from the iShares Global Clean Energy ETF, offered by BlackRock.

```
                ENPH       XEL      PLUG  ...      NOVA       ALE      CSIQ
Date                                      ...
2020-01-02  0.001706 -0.004798  0.003091  ... -0.018430 -0.001985  0.032494
2020-01-03 -0.012553  0.001437 -0.167768  ...  0.006981  0.003476 -0.023573
2020-01-06 -0.011065  0.002079  0.002621  ... -0.008718  0.010880 -0.001791
2020-01-07 -0.033123  0.000961 -0.068468  ... -0.000868 -0.008763 -0.009348
2020-01-08 -0.018853 -0.002241  0.007380  ...  0.013974 -0.008934 -0.014079

[5 rows x 20 columns]
```

The DataFrame was then passed to a function which created a stock correlation network using the data structures available in NetworkX. The data were log-normalized (since we are interested in returns, and not absolute prices), and the correlation matrix was created with the Pandas function `.corr()`. The edge set of each network was created from the pairwise correlations between stocks, and insignificant correlations were dropped. After this, the network was adequately represented in NetworkX, and the library’s built-in centrality functions were used to find the most central node in the network. Betweenness centrality was the chosen centrality measure used to rank the vertices in the network; experimentation between different centrality measures showed that betweenness centrality yielded the best results.

Once the most central stock in the ETF’s stock correlation network was identified, the performance of both the most central stock and the fund was tested using data from calendar year

2019. Once this process had been completed, the ETFs that had underperformed their most central stocks were moved to next step of the simulation. Here, the same central stock and fund were evaluated for their performance in 2020. This data was collected and analyzed in the spreadsheet program Microsoft Excel.

In addition to the protocols and functions implemented above, another function was prepared outside of the primary codebase to visualize the stock correlation network for a given ETF. Each vertex in the generated graph represents a stock in ETF's basket of holdings; the size of each vertex in the network was scaled by its degree, which is a visual indication of the strength of a stock's correlation to other stocks in the basket. The networks created by this script were not used in further parts of the analysis, and only served to provide a visual understanding of the density of connections between different stock in each ETF's stock correlation network.

IV. Results

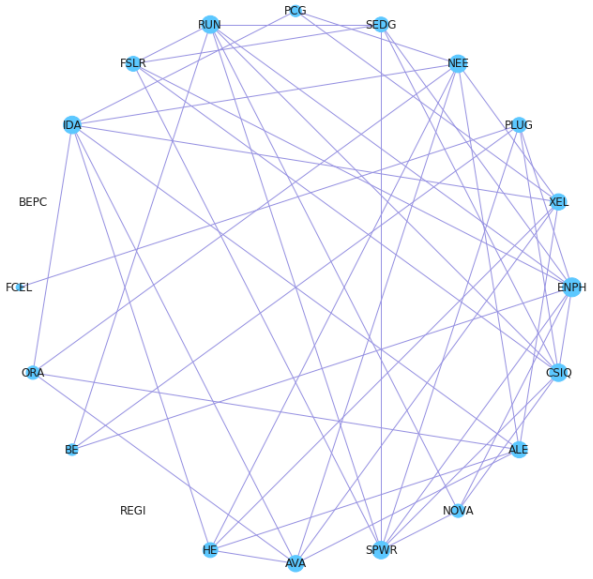
The results of the simulated asset performance during 2020 are shown below.

Initial Investment		**Fund**	**Percent Increase**	**2020 Year-End Total**		**Central Stock**	**Percent Increase**	**2020 Year-End Total**
$1,000		ICLN	138.18	$2,381.80		SPWR	400.03	$5,000.29
$1,000		NLR	3.23	$1,032.27		DUK	5.78	$1,057.80
$1,000		XOP	-36.29	$637.08		DVN	-34.89	$651.15
$1,000		XLF	-2.68	$973.15		AXP	-2.55	$974.52
$1,000		VFH	-2.79	$972.12		AXP	-2.55	$974.52
$1,000		KCE	29.04	$1,290.36		BLK	45.39	$1,453.89
$1,000		VIS	10.53	$1,105.32		ROP	18.72	$1,187.23
$1,000		IYJ	15.89	$1,158.94		AME	20.90	$1,209.05
$1,000		ITA	-15.59	$844.11		CW	-18.12	$818.81
$1,000		WBIF	-2.86	$971.39		RJF	7.12	$1,071.20
Total				$11,366.53				$14,398.46

In the calendar year 2019, all ten of the most central stocks yielded better returns than their parent ETFs. Furthermore, in the calendar year 2020, nine of the most central stocks continued to outperform their parent ETFs; the only stock-fund pair that failed this test was BlackRock's iShares U.S. Aerospace & Defense fund. This fund performed better than its most central stock, Curtiss-Wright.

The collection of ETFs increased in value by 13.67% over the course of 2020, while the collection of the central stocks of these ETFs increased in value by 43.98% during the same period. More concretely, if US$10,000 were equally invested in all 10 ETFs and all 10 most central stocks on January 1st, 2020, the money invested in the collection of ETFs would have appreciated to a value of US$11,366.53 by December 31st, 2020, while the money invested into the collection of central stocks would have appreciated to a value of US$14,398.46 by the same date. This is an absolute difference of $3,031.93, and a percent difference of 26.67%.

The results derived from this small sample of ETFs are positive and warrant further exploration of the available financial data to see how representative these results are of the larger market.



The image above shows the stock correlation network graph for the iShares Global Clean Energy ETF issued by BlackRock. As mentioned in the Methods section above, this image functions only as an aid to help visualize the stock correlation network.

V. Discussion

Perhaps the most important direction for future study is replication of these simulations with a larger and more accurate corpus of data. As mentioned in the Methods section above, the data sources that were available for use in this project were taken from non-official APIs and from scraping internet sites; resolving these issues and replicating the simulation on a larger set of ETFs would help generate data that would more confidently help corroborate (or potentially disprove) the preliminary results gathered here.

Another direction for future research might be to extend the range of prior data to a longer period when creating the stock correlation network, or accounting for market conditions and inflation rates when making the correlation matrix. In this project, we made the stock correlation matrix and tested asset performance over the two most recent calendar years; accounting for factors such as inflation or noting the state of the market (for instance, whether it is generally a "bull" or "bear" market during the testing period) might help regularize these results. Furthermore, the coronavirus pandemic has led to some strange effects on the stock market in the calendar year 2020 (and continues to do so as this report is being written in 2021) – testing this hypothesis in other years will yield a clearer insight into how central stocks perform against their parent ETFs over greater timespans.

After this experiment has been replicated and extended, we may also pursue a more granular analysis of the causal effects, both qualitative and quantitative, that lead to a certain central stock performing better than its parent ETF, or the opposite case where the parent ETF performs better than the most central stock. For instance, characteristics such as sector choice and distribution of stocks within a sector, company size of the most central stock relative to company size of other stocks in the ETF basket, or geographic region and distribution. This analysis will not only lead to a better understanding of the data but may also show some more important correlations – for instance, we might discover that certain characteristics of a given class of funds better predictors of success than centrality.

A similar evaluation can also be applied to different assets, such as index funds; we may also investigate trying different investment policies. There are some firms that focus almost

exclusively on trading ETFs, often buying & selling millions of dollars of ETF shares every single day. In this experiment, we simulated holding the chosen ETF and central stock for an entire calendar year; more active investment policies where we trade more frequently might lead to better results over a larger and more diverse collection of funds.

Following this analysis, it would be especially interesting to see the conclusions and data collected might be used as input to statistical learning techniques, which can help develop an automated classification scheme for ETFs. The input to a model generated from these statistical learning techniques would be a vector set $\boldsymbol{X}$ of ETF stock correlation networks and central stocks, each with a corresponding label; these labels would comprise a vector set $\boldsymbol{Y}$. The object of such a model, after being trained on the set ($\boldsymbol{X}$, $\boldsymbol{Y}$), would be to accurately predict the label for an unseen ETF correlation network $\boldsymbol{x}$. This may include models based on neural networks, such as deep learning classifiers, as well as non-neural network models, such as logistic regression for binary classification of the "goodness" of an ETF. Assuming that such a model can be developed and achieve a high degree of accuracy on a diverse test set, it could be deployed as part of an automated trading strategy, or as a tool that traders and investors use to inform their purchases and sales.

Bibliography

Chen, James. “Exchange Traded Fund – ETFs.” Investopedia. Dotdash, March 3, 2021. https://www.investopedia.com/terms/e/etf.asp.

Fidelity Investments. Fidelity ETF Screener. Fidelity Investments, 2021. https://research2.fidelity.com/fidelity/screeners/etf/public/etfholdings.asp.

Ganti, Akhilesh. “Adjusted Closing Price Definition.” Investopedia. Dotdash, December 28, 2020. https://www.investopedia.com/terms/a/adjusted_closing_price.asp.

Hagberg, Aric, Pieter Swart, and Daniel S Chult. Exploring network structure, dynamics, and function using NetworkX. No. LA-UR-08-05495; LA-UR-08-5495. Los Alamos National Lab. (LANL), Los Alamos, NM (United States), 2008.

McKinney, Wes. "Data structures for statistical computing in python." In Proceedings of the 9th Python in Science Conference, vol. 445, pp. 51-56. 2010.

Onnela, J-P., Anirban Chakraborti, Kimmo Kaski, and Janos Kertesz. "Dynamic asset trees and Black Monday." Physica A: Statistical Mechanics and its Applications 324, no. 1-2 (2003): 247-252.

Pandas Dev. Computer software. Pandas. NumFOCUS, February 2020. https://pandas.pydata.org/.

PyData Dev. “Pandas DataReader.” *Pandas-DataReader*. NumFOCUS, July 7, 2020. https://pandas-datareader.readthedocs.io/en/latest/.

Additional Material Begins Here

ECON 46 - Final Project - Nikesh Mishra

The code in this document was written as part of the final project for the Spring 2021 offering of ECON46.

We begin by importing all the requisite packages.

[1]:
```
from pandas_datareader._utils import RemoteDataError
import matplotlib.pyplot as plt
import pandas_datareader as dr
import pandas as pd
import networkx as nx
import seaborn as sns
import numpy as np
import datetime
import math
```

[2]:
```
globalList = []
```

The following function returns a list of ETF holdings given the name of a fund.

[3]:
```
def tickersOfETFHoldings(etf_ticker):
  url = 'http://research2.fidelity.com/fidelity/screeners/etf/public/
 ↪etfholdings.asp?symbol={}&view=Region'.format(etf_ticker)
  hd = pd.read_html(url)[0]
  hd = hd.query('Weight >= 0.5 & Geography == "United States"')
  initial = hd["Symbol"].tolist()
  result = [i for i in initial if (isinstance(i, str)) and i.isalpha()]
  return result
```

The following function produces a Pandas DataFrame of ticker prices between specified `start` and `end` dates when provided with a list of stock tickers.

[4]:
```
def makeDFfromTickers(tickers, start, end):
  result = pd.DataFrame()
  for ticker in tickers:
    try:
      history = dr.DataReader(ticker, 'yahoo', start, end)
      result[ticker] = pd.Series(history["Adj Close"])
    except (RemoteDataError, TypeError, NameError, KeyError):
      pass
  return result
```

The following function returns the vertex with highest betweenness centrality when given a NetworkX graph.

[5]:
```
def getBetweenness(graph):
  return max(nx.betweenness_centrality(graph), key=nx.
 ↪betweenness_centrality(graph).get)
```

The following function returns the difference in price of a stock between two dates `start` and `end`.

[6]:
```
def change(symbol, start, end):
  try:
    data = dr.DataReader(symbol, 'yahoo', start, end)
    change = ((float(data["Adj Close"][-1]) - float(data["Adj Close"][0]))/
 ↪float(data["Adj Close"][0])) * 100
  except (RemoteDataError, TypeError, NameError, KeyError):
    change = "Couldn't get the difference"
  return change
```

The following function almost replicates the functionality of the `change()` function written above, but is used as a utility to print the difference directly to the cell output.

[7]:
```
def price_between_dates(symbol, start, end):
  answer = ""
  try:
    data = dr.DataReader(symbol, 'yahoo', start, end)
    change = ((float(data["Adj Close"][-1]) - float(data["Adj Close"][0]))/
 ↪float(data["Adj Close"][0])) * 100
    answer = "Between {start} and {end}, the price of {symbol} changed by␣
 ↪{change}%.".format(start=start, end=end, symbol=symbol, change=change)
  except (RemoteDataError, TypeError, NameError, KeyError):
    pass
  return answer
```

This is the main simulation function that produces a stock correlation matrix and finds the vertex with highest betweenness centrality.

[8]:
```
def doThingETF(etfSymbol, start, end, normalized=pd.DataFrame()):
  tickers = tickersOfETFHoldings(etfSymbol)
  prices = makeDFfromTickers(tickers, start, end)

  for column in prices.columns:
      normalized[column] = np.log(prices[column]).diff(-1)

  stock_correlation_matrix = normalized.corr()

  edges = stock_correlation_matrix.stack()
  edges = edges.reset_index()
  edges.columns = ["first", "second", "corr"]
  edges = edges.loc[edges["first"] != edges["second"]]
  edges = edges.copy()
```

```
  threshold, under_thresh = 0.5, []
  G = nx.from_pandas_edgelist(edges, "first", "second", edge_attr=["corr"])
  for edge in G.edges():
      first, second = edge
      correlation = G[first][second]["corr"]
      if abs(correlation) < threshold:
          under_thresh.append((first, second))
  G.remove_edges_from(under_thresh)

  central = getBetweenness(G)

  try:
    if change(central, start, end) >= change(etfSymbol, start, end):
      globalList.append((etfSymbol, central))
  except (RemoteDataError, TypeError, NameError, KeyError):
    pass

  print("We are operating on {}.".format(etfSymbol))
  print("The most central stock is {central}, and the most heavily weighted␣
↪American stock is {big}.".format(central=central, big=tickers[0]))
  print(price_between_dates(central, start, end))
  print(price_between_dates(etfSymbol, start, end))
  print('\n')
```

The following cell contains the body of the simulation. The results are printed to the cell output.

[ ]:
```
etfs = ['ICLN', 'NLR', 'XOP', 'XLF', 'VFH', 'KCE', 'VIS', 'IYJ', 'ITA', 'WBIF']
start = datetime.datetime(2019, 1, 1)
end = datetime.datetime(2019, 12, 31)
for etf in etfs:
  doThingETF(etf, start, end)
print(globalList)

etf_symbols = [i[0] for i in globalList]
stock_symbols = [i[1] for i in globalList]
start = datetime.datetime(2020, 1, 1)
end = datetime.datetime(2020, 12, 31)
total = 0
for i in range(len(etf_symbols)):
  if change(stock_symbols[i], start, end) >= change(etf_symbols[i], start, end):
    print("{} satisfies the hypothesis.".format(etf_symbols[i]))
    print(price_between_dates(stock_symbols[i], start, end))
    print(price_between_dates(etf_symbols[i], start, end))
    total += 1
  else:
    print("{} doesn't satisfy the hypothesis".format(etf_symbols[i]))
  print('\n')
```

```
success = total/len(etfs)

print("Of the {numETFs} ETFs that you tested, {lenGlobList} were good. \
Of those, {total} satisfied the hypothesis. This is a success rate of \
{success}.".format(numETFs=len(etfs), lenGlobList=len(globalList), total=total,␣
 ↪success=success))
```

The following function visualizes a stock correlation network; much of the code is copied from `doThingETF()` above.

[10]:
```
def makeETFgraph(etfSymbol, start, end, normalized=pd.DataFrame()):
  tickers = tickersOfETFHoldings(etfSymbol)
  prices = makeDFfromTickers(tickers, start, end)

  for column in prices.columns:
      normalized[column] = np.log(prices[column]).diff(-1)
  stock_correlation_matrix = normalized.corr()

  edges = stock_correlation_matrix.stack()
  edges = edges.reset_index()
  edges.columns = ["first", "second", "corr"]
  edges = edges.loc[edges["first"] != edges["second"]]
  edges = edges.copy()

  threshold, under_thresh = 0.5, []
  G = nx.from_pandas_edgelist(edges, "first", "second", edge_attr=["corr"])
  for edge in G.edges():
      first, second = edge
      correlation = G[first][second]["corr"]
      if abs(correlation) < threshold:
          under_thresh.append((first, second))
  G.remove_edges_from(under_thresh)

  node_size = []

  for key, value in dict(G.degree).items():
      node_size.append((45 * value) + 1)

  sns.set(rc={"figure.figsize": (7, 7)})

  nx.draw(G, pos=nx.circular_layout(G), with_labels=True, node_size=node_size,␣
 ↪node_color="#5cc8ff", edge_color="#9792e3")
  plt.show()
```

Below is the command to produce a network visualization of the ICLN ETF based on stock correlation data during 2019.

[ ]:
```
start = datetime.datetime(2019, 1, 1)
end = datetime.datetime(2019, 12, 31)
makeETFgraph('ICLN', start, end)
```