Forecasting Returns Using Earnings Call Transcripts

June 12, 2017

Justin Kahl
Stanford University: Management Science and Engineering
jkahl@stanford.edu

Jonathan Khalfayan
Stanford University: Economics
jkhal@stanford.edu

Santiago Rodriguez
Stanford University: Chemical Engineering
santiago.rodriguez@stanford.edu

Matthias Schmitz
Stanford University: Mathematical and Computational Science
schmitzm@stanford.edu

Sam Sklar
Stanford University: Mathematical and Computational Science
ssklar2@stanford.edu

and

Juan Pablo Villarreal
Stanford University: Management Science and Engineering
jvillarr@stanford.edu

Abstract

After obtaining roughly 120,000 quarterly earnings call transcripts from 2007-2017 for U.S. equities, we ran these transcripts through a language processing algorithm, which identified certain corporate initiatives in the transcripts through word and phrase frequencies. Taking our most robust data set (2012-2015), we chose a universe of small, cheap, and highly leveraged equities. For each transcript corresponding to a certain release date and equity, we regressed the corporate initiative variables against 1 year forward adjusted returns, normalized to the Russell 2000 ETF (RUT). After finding the most statistically significant variables, we formed a scoring algorithm based on the sum product of the variable frequencies weighted by t-stat contributions of those respective variables. Based on these scores, we formed portfolios, which we tested on our out-of-sample data from 2007-2012 and 2016-2017.

Contents

1 Introduction

In the past 50 years, the two most influential inventions of asset management are John Bogle's index fund and Henry Kravis's Leveraged Buy-Out. Kravis demonstrated that using debt to purchase cheap companies, and then using cash flows from said company to deleverage, could generate annual returns greater than 20%. Bogle recognized an asymmetry between active management performance and active management fees and thus invented the index fund.

Our project is an extension of two papers by Daniel Rasmussen and Brian Chingono: "Leveraged Small Value Equities" and "Forecasting Debt Paydown

Among Leveraged Equities". Their research shows that from 1965 to 2013, on a ranking method devised by the researchers, annual portfolios of the top 25 small value stocks produced returns of 25.1% annually, outperforming private equity (Rasmussen, Chingono, 2015). These portfolios outperformed the market by 11.7% on average (Rasmussen, Chingono, 2015). Our strategy, while more contingent upon the markets than private equity, has some significant advantages over private equity. The first is the transparent nature of public markets, followed by the lack of control premiums–which allows us to purchase equities at a lower price–and the third, like Bogle's index fund, is the ability to charge lower fees because we do not pay investment banks outrageous fees for financial transactions, nor do we have a team of analysts staffed on each company.

The foundation of our strategy is the use of earnings call transcripts as indicators of future performance. We start by identifying a universe of stocks which contain the characteristics of a successful LBO candidate. First we filtered our universe to between the 25th and 75th percentile by market capitalization. We intended to utilize the size premium while excluding micro-caps too small for institutional investors. Next, we took the 50% cheapest companies by EV/EBITDA. This filter is meant to exploit the value premium—"The EBITDA/EV metric can plausibly provide a more current measure of valuation than price/book because book value is generally a stale accounting measure, whereas enterprise value and EBITDA both incorporate the most recently available fundamental information" (Rasmussen, Chingono, 2015). The third criterion we used is above mean long-term debt/EV. This filter ensures our companies are leveraged and able to pay down debt.

While free cash flow allows a company to deleverage, it is ultimately contingent upon interest rates, capital intensity, and tax rates, so we use EBITDA as a substitute for free cash flow. For obvious reasons, a firm's ability to deleverage is largely determined by the free cash flow yield, defined as free cash flow / market cap. Valuation is far more meaningful in a leveraged universe. Consider a company with no debt: if one wants to double the free cash flow yield of that company, the only way to do so would be to pay half the purchase price. Leverage, however, allows an investor to reduce the amount of equity needed to purchase the business, reducing market cap, but leverage also adds interest payments, reducing free cash flow (Rasmussen, Chingono, 2015). And so, when considering a constant leverage percentage, increasing the purchase price of an equity has a twofold negative effect: increasing the interest payments and increasing the amount of equity needed to purchase.

Having illustrated the fundamental relationship between leverage and value, we will explore the following market-driven relationship: leverage aversion. Leverage has long been a topic of contention among institutional investors. Throughout the 1970's the financial world was brought to a frenzy over the new LBO craze. High-Yield bonds, known as junk bonds, have been a controversial instrument since their increased use in the 1980's by Michael Milken at Drexel Burnham Lambert (Burrough, Helyar). For better or worse, junk bonds sparked the boom that created the private equity industry we know today (Burrough, Helyar). Big name institutional investors have tended to avoid

leveraged equities. Benjamin Graham once described debt as "adverse economic factor of some magnitude and a real problem for many individual enterprises", and Warren Buffett once said "I do not like debt and do not like to invest in companies that have too much debt, particularly long-term debt". At the crux of Rasmussen and Chingono's research is the idea that "Investors have a well-documented pattern of leverage aversion, which we believe contributes to the excess returns to be found in the leveraged small-value stocks" (Rasmussen, Chingono, 2015).

And so, our strategy seeks to find these excess returns through the analysis of earnings call transcripts. Through processing and analyzing transcripts of the companies in our universe over a 4 year period, we identified certain corporate sentiments that correlate with higher returns.

2 Data

The data necessary to train the model includes the following two components: quarterly earnings call transcripts analyzed through natural language processing techniques and adjusted forward returns normalized against the Russell 2000 for each equity following the date of the earnings call.

In order to create an exhaustive data set capable of accommodating a shifting universe of equities year-over-year, we acquired 120,856 earnings call transcripts of companies listed on both the NYSE and NASDAQ, primarily from 2012 to 2015. As the files were initially compiled in HTML format, we were forced to first go through the monotonous process of converting them into TXT files. We then chose to analyze these files by creating a Python script to walk through each file, counting the frequency of desired words and two-word phrases as well as the total number of words contained within the file. In order to account for subtle differences in word appearance, such as between "there," and "There", we accounted for both case and punctuation. In a more substantial manner, we also accounted for tense and part of speech, as to obtain an accurate representation of the frequency of the desired base word or phrase. As we parsed through the multitude of TXT files, we wrote our results to a CSV file, from which we could run regression analysis. (See Appendix for Code)

Our initial analysis examined a variety of keywords and phrases that could potentially signal equity performance growth for our universe of small, cheap, and highly-levered stocks. These words were chosen from our business knowledge and fell within our general hypothesis derived from the aforementioned papers. After initial regression analysis of these terms, we found that the data was too sparse to obtain a strong correlation to stock performance. So in order to proceed, we drew inspiration from the work of Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Their efforts have worked to create Global Vectors for Word Representation (GloVe), which uses an "unsupervised learning algorithm for obtaining vector representations for words" (Pennington, Socher, and Manning). With the creation of these vector representations, they're able to find the Euclidean distance between two words, and gauge "the linguistic

or semantic similarity of the corresponding words." Our inspiration drew from the notion of finding a word's nearest neighbors, which allows you to, with reasonable certainty, believe that a group of words (located close to each other in word-vector space) have similar meaning. For example, "Toad" and "Frog" appear as each other's closest neighbor, insinuating their similarity. This is analogous to searching for "Deleverage" and "Debt Paydown," both of which point to the same procedure.

To this vain, we decided to create groups of words and two-word phrases which correspond to similar business initiatives. The four initiatives chosen were the following: operational improvement, deleverage, dividends and repurchases, and expansion. In lieu of using an algorithm to obtain the similarity of words/phrases, we used qualitative business knowledge to determine the implication of different terms and grouped them accordingly. Thus, in creating groups of words corresponding to different initiatives, we replicated the concept of nearest neighbors. After the creation of these groups, we counted the number of times any of these words or phrases appeared in an earnings call transcript, ending with a sum total for each group of words. After obtaining the word and phrase frequency for each transcript, we needed to find each equity's normalized performance over the following year to be able to analyze our data through regression analysis. We decided to find each equity's adjusted closing price the day of the earnings call release and one year later because the adjusted closing price accounts for corporate actions such as dividend payments and stock splits, while the closing price does not. In order to adjust for general market trends, we found the difference between the equity's performance and the performance of the Russell 2000 over the same period. The Russell 2000 was chosen to normalize the returns because it is a small-cap market index that most closely resembles our universe of stocks. Using data purchased through Intrinio's Microsoft Excel Add-in, we obtained the one year forward returns of each company.

After using our Natural Language Processing algorithm to identify the frequency of groups of words and obtaining the normalized forward returns for each equity, we eliminated all equities not contained within our tradeable universe for each calendar year through a simple Python script. After this final refinement, the data was ready to be put through a regression analysis. In review, each earnings call had been parsed into the following data: Ticker, Date of Earnings Call, Sum of Group 1 Words and Phrases, Sum of Group 2, Sum of Group 3, Sum of Group 4.

3 Analysis

After processing the earnings call transcripts in search of certain key words and phrases, we regressed the frequency of keywords vs. 1 year forward returns normalized to the Russell 2000. As we mentioned, due to scarcity of frequencies of individual words/phrases and low correlation to stock performance, we then ran the same regressions on our four corporate initiative variables: Operational Improvement (coefficient = .0059, t stat = 1.53), Deleverage (coeffi-

cient = 0.005642, t stat = 1.058302), Dividends and Repurchases (coefficient = 0.005147, t stat = 2.954937), and Expansion (coefficient = 0.001621, t stat = 1.79641). To clarify, the coefficients are in terms of percentage returns above the Russell 2000. For example, a company whose earnings call transcript mentions dividends and repurchases 10 times will, on average, be expected to outperform the Russell 2000 by 5 percentage points the following year, all else equal. These data sets were particularly noisy, and we used the regressions to gain hints into which groups provided value to our strategy. Given the nature of our universe, we omitted the expansion group variable in our scoring calculations.

We scored the transcripts by first ranking each transcript by frequency of counts of each variable in ascending order (with a rank of 1 corresponding to the lowest frequency of mentions and so on). We then weighted by statistical significance by calculating a sum product of each variable ranking with its t stat contribution. The t stat contribution was calculated by taking the t stat of each variable and dividing it by the sum of all three t stats. The equations below illustrate how the scoring for transcript i occurred.

$$Score_i = rank_{i1} * t_1 + rank_{i2} * t_2 + rank_{i3} * t_3$$

$$t_1 = \frac{tstat_1}{tstat_1 + tstat_2 + tstat_3}$$

4 Execution

After scoring and dating all of our transcripts, we began to formulate our plan for executing trades. We chose to use the Quantopian platform to build our algorithm, and had to format our material to fit into the available API. To better explain our strategy and thought process, we will give a brief explanation of how the system works. In Quantopian, each stock is assigned a unique ID which maps to all of the fundamental data associated with that stock. One can query for the current price, open volume, as well as a myriad of other information available through the service. If one is in a trading back test and the current day in simulation is 5/10/12, a query for the enterprise value of AAPL would return the most recent update of the enterprise value of AAPL on or before 5/10/12. Thus, while lookup is easy, it is impossible to look back at previous values or look ahead to upcoming values. The same would be true for data that we import.

For our first test, we took the earnings call transcripts that were in the top quintile of all scores over all years, and assigned those to be "buys." To signify this, within the Quantopian framework, we would give this stock a score of 4 on that given date. For simplicity, we execute all trades at market open, one day after the public release of the transcript to prevent look-ahead bias. Each subsequent quarter that the earnings call transcript did not meet our criteria, its score would be decremented by 1. Thus a stock that had a score of 3 at a given time would be held for at least 3 subsequent quarters, and all tradable stocks would be assigned equal weights. Anything with a 0 would not be a part

of our current portfolio. We reshuffled our portfolio once per quarter, and tested it on our in sample data with the result in Figure 1 (below).

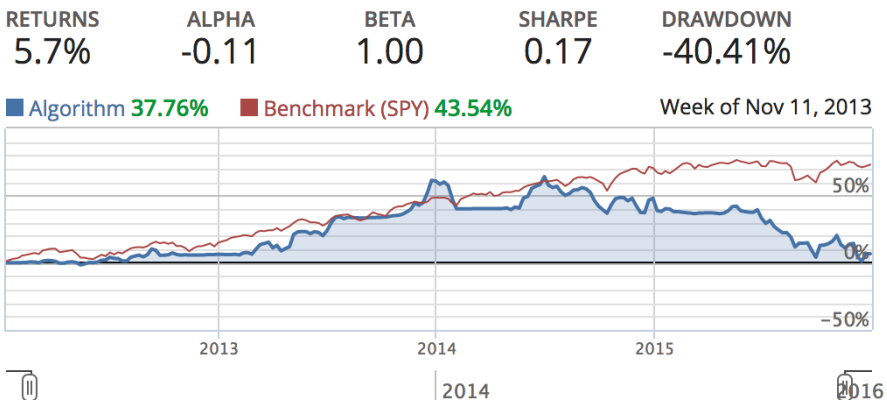

Figure 1: Performance of Reactionary Trading Strategy,
One Year Holding Period

Clearly, we did not receive promising results, and checked the internal logs to find out why. It was clear that many of our orders were going unfilled especially when we attempted to terminate a position that no longer fit our criteria. Thus, as our portfolio became overloaded with undesirable stocks, our returns suffered. To account for the relatively low trading volume present among our target stocks, we reduced our book size from 1,000,000 to 100,000 USD. We ran our algorithm again and obtained the following results in Figure 2 (below).

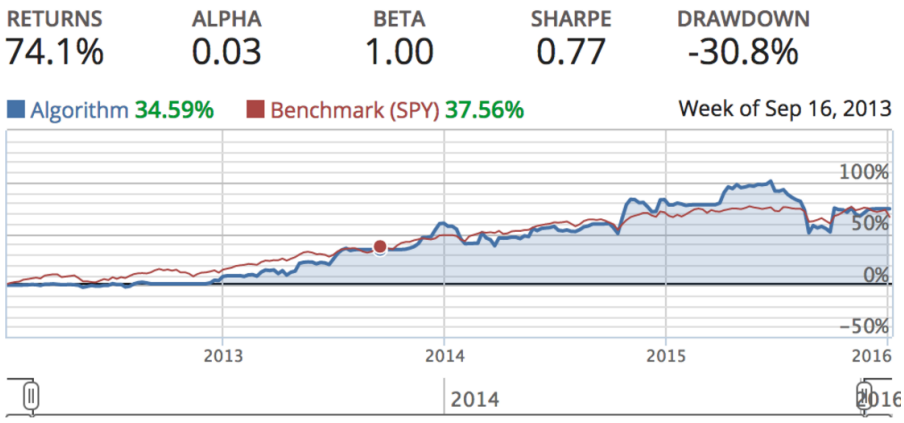

Figure 2: Performance of Reactionary Algorithm with Reduced Portfolio Size,
One Year Holding Period

While a step in the right direction, we knew that there was a serious flaw in our trading algorithm because the results of our regressions confirmed that we should be beating not only the SP, but also the Russell 2000 which itself outperformed the SP over this period. At this point, we realized that we needed to be shuffling our portfolio more frequently, as well as prioritizing stocks that had recently received a buy signal. Without changing our scored data, we shifted to one quarter holding periods with an algorithm that reshuffled every month, buying only those stocks that held a 4 in the current period. The results of that back test are shown in Figure 3 (below).

Figure 3: Performance of Reactionary Strategy, One Month Holding Period

At this point, we confirmed the value of increasing the frequency in which our portfolios are reshuffled, as our scoring algorithm is most effective at predicting price increase directly after an earnings call transcript is released. We looked closer at our data set, and found that there was a small subset of earnings call transcripts at the top of each scoring category that explained the majority of the excess returns. For example, in 2013, our tradeable universe averaged a net return of 15.74 percentage points in excess of the Russell 2000. Meanwhile, the top 10 stocks in terms of the deleverage criterion had returns of 36.44 percent in excess of the Russell 2000. We proceeded to import the score ranking directly into Quantopian, and reshuffled our portfolio daily to contain only the top performing stocks. We implemented a top 30 basket, and a top 10 basket, shown below in Figures 4 and 5, respectively.

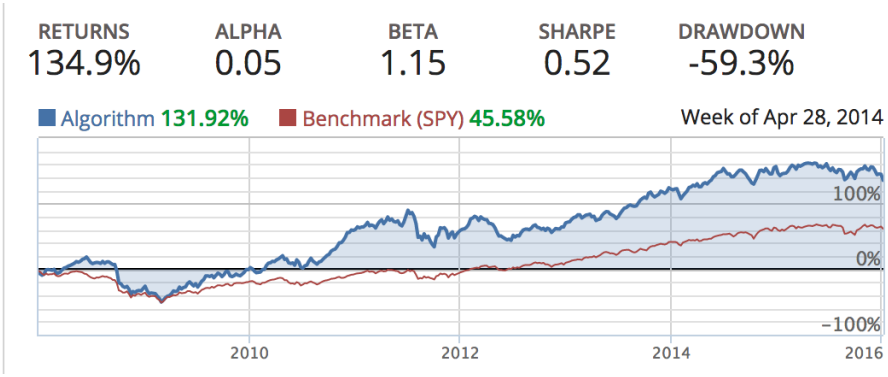

Figure 4: Performance of Daily Reshuffle Strategy, Top 30 Stock Basket

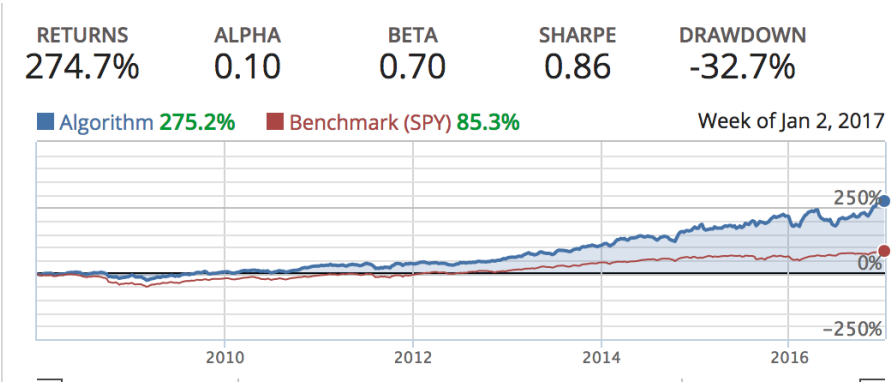

Figure 5: Performance of Daily Reshuffle Strategy, Top 10 Stock Basket

5 Results

Having decided on the top 10 basket daily shuffle as the optimal strategy, we proceeded to test it on the remaining out of sample data that we had. Having used the beginning of 2012 through the end of 2015 as training periods, we were left with 2008 – 2012 and 2016 – 2017. Below is the result of the first back test:

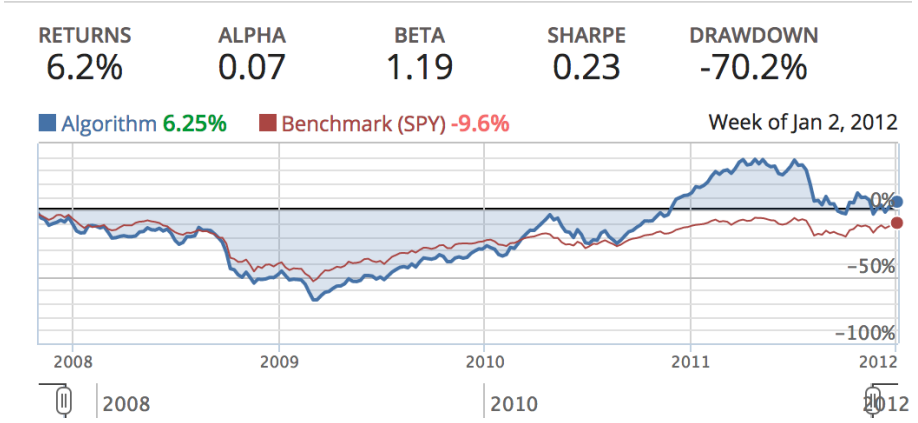

Figure 6: Daily Reshuffle Strategy, Top 10 Stock Basket (2008-2012)

While our 6.2 percent returns far outperformed the benchmark's 9.6 percent loss, our Sharpe ratio was depressed by our relatively high max drawdown. Considering our strategy depends upon highly leveraged companies attempting to pay down debt, the rise in high yield bond rates during the beginning of the crisis of 2008 were not conducive to the success of our strategy. In our 2016 - 2017 back test, our algorithm fared much better, far outperforming the benchmark while maintaining a Sharpe ratio of 2.14 (Figure 7).

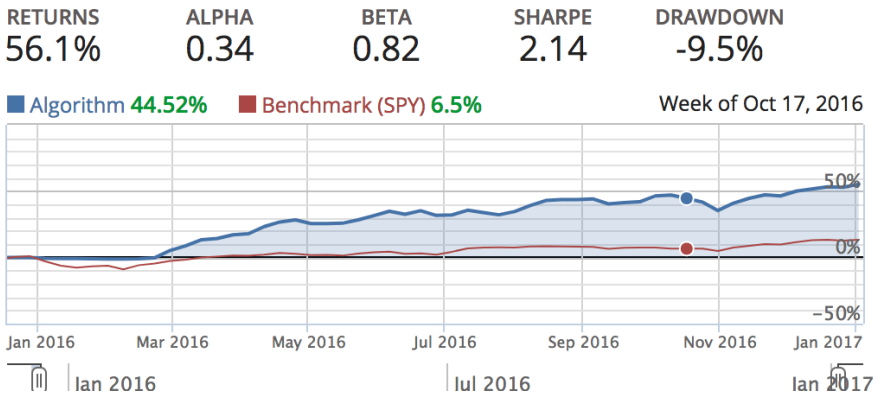

Figure 7: Daily Reshuffle Strategy, Top 10 Stock Basket (2016)

6 Discussion

Moving forward, we have identified several strategies to further improve our algorithm. First, our strategy is particularly effective at identifying effective long positions, but we fail to hedge our risk with shorts. Thus, we are highly market correlated and susceptible to market risk. We intend to do further fundamental analysis, followed by similar regressions, to identify corporate strategies, or lack thereof, that indicate a short candidate. Second, within our long candidates, we can implement stop loss purchases to minimize the large max drawdown that diminishes our Sharpe ratio. As our data set is limited, we intend to use bootstrap techniques to simulate a data set of stock returns similar to our portfolio. Moreover, our current algorithm is strictly alpha seeking, as we terminate and take new positions completely based on our scoring algorithm, without consideration of incurring greater transaction costs or tax concerns. Finally, there is a great potential for growth within our NLP algorithm. While it was created with tenses in mind, we can better train it with more sophisticated word vectors to better implement our scoring with future data sets. To prepare our algorithm for live trading, we will need to scrape the web for new Earnings Call Transcripts daily. After market close, all new transcripts will be processed and scored, further training our scoring system while simultaneously preparing us for new trades at market open the next day.

7 Conclusion

Theoretically, our strategy should be inherently more volatile than the market, producing greater returns but also greater max drawdown than the market at large, for two principle reasons: the first is the nature of our universe. Intuitively, "publically-traded companies that show high levels of leverage and low valuations are typically facing some risky or problematic situation, declining revenues or profitability, or other unfavorable circumstance." (Rasmussen, Chingono, 2015). As a result, we would expect a portfolio of 10 of these equities to be highly volatile. The backtests failed to demonstrate the effect of diversification on the volatility between our 10 equity portfolio and our 30 equity portfolio, likely due to sample size. A second reason for the volatility of our fund is the dependence of our portfolio companies on the high yield bond market. Because the majority of the companies in our universe are highly leveraged, with most companies issuing debt in the high-yield market, the success of our strategy depends on the high yield spread. Every three to six years, investors demand higher premiums for taking credit risks, and as the high yield spread increases, the equities in our portfolio find it increasingly difficult to refinance their debt, often leading to bankruptcy. Figure 8 (below) illustrates the movement of our strategy with respect to the high yield spread.

Figure 8: (top) Top 10 Stock Basket (2016),
(bottom) BofA Merrill Lynch High Yield Spread

8 Acknowledgements

With regards to acknowledgements, we would like to sincerely thank Dr. Lisa Borland and Enguerrand Horel for their invaluable guidance throughout the past ten weeks, especially in helping us obtain the earnings call transcripts. Nicholas Schmitz and Daniel Rasmussen from Verdad Advisors provided the original inspiration for our project and were exceptional resources, generously offering their knowledge, critiques, and support throughout the process. Finally, congratulations to our fellow MSE 448 classmates this quarter whose strategies and insights never failed to impress us.

9 Bibliography

Pennington, Jeffrey, Richard Socher, and Christopher D. Manning. GloVe: Global Vectors for Word Representation. Stanford University. Available at: https://nlp.stanford.edu/pubs/glove.pdf

Burrough, Bryan, and John Heylar. Barbarians at the Gate: The Fall of RJR Nabisco. London: Arrow, 2010. Print.

Chingono, Brian Kundai and Rasmussen, Daniel, Leveraged Small Value Equities (August 1, 2015). Available at SSRN: https://ssrn.com/abstract=2639647

10 Appendix: Code

NLP Algorithm

```
import os
import math

WordsOfInterest = ["cost-
cut","deleverage","deleveraging","deleveraged","debt-
reduction","synergy","synergies","acquisition","acquisitions","acquire","acqu
iring","merger","mergers","paydown","pay-down","buyback","buy-
back","repurchase","repurchased","repurchasing","tender-
offer","dividends","dividend"]

PhrasesOfInterest = ["improved margins","improve margins", "margin
improvement", "margin expansion", "eliminate costs", "cost cutting", "cut
cost", "reduced costs", "reducing costs", "pay down","debt
reduction","reducing debt","reduced debt","reduce debt","buy back","bought
back","tender offer", "revenue growth","debt restructuring","decrease
leverage","decreasing leverage","pay off"]
import csv
listOfPhrases = []
listOfWords = []
listOfMonths =
["Jan.","Feb.","Mar.","Apr.","May.","Jun.","Jul.","Aug.","Sep.","Oct.","Nov."
,"Dec."]
previous = ""
ticker = ""
finalDate = ""
from datetime import datetime
directory =
'/Users/Santiago/Documents/Stanford/Year_2/MSE_448/EarningsTranscripts'

with open('NLP.csv','wb') as file:
    file.write("Date,")
    file.write("Ticker,")
    for word in WordsOfInterest:
        file.write(word)
        file.write(",")
    for phrase in PhrasesOfInterest:
        file.write(phrase)
        file.write(",")
    file.write('\n')
# Iterate through all txt files in the current directory.
    for filename in os.listdir(directory):
        currpath = os.path.join(directory, filename)
        with open(currpath,'r') as f:
                if 1 == 1:
                    ticker = ""
                    finalDate = ""
                    listOfPhrases = []
                    listOfWords = []
                    previous = ""
```

```
                    # Iterate through every line in current text file.
                    for line in f:
                        # Obtain ticker symbol
                        if "NYSE:" in line:
                            i=0
                            while i < len(line):
                                if line[i-5:i] == "NYSE:":
                                    while i < len(line):
                                        if line[i] == ")":
                                            break
                                        ticker += line[i]
                                        i += 1
                                    break
                                i += 1
                        if "NASDAQ:" in line:
                            i=0
                            while i < len(line):
                                if line[i-7:i] == "NASDAQ:":
                                    while i < len(line):
                                        if line[i] == ")":
                                            break
                                        ticker += line[i]
                                        i += 1
                                    break
                                i += 1
                        # Obtain date of earnings call transcript.
                        for month in listOfMonths:
                         if month in line and finalDate == "" and line[6] == '.':
                                date = line[:9]
                                date =
str((datetime.strptime(date,'%b.%d.%y')))
                                finalDate = date[:10]
                        # Parse line into words
                        for word in line.split():
                            word = word.lower()
                         if word[len(word)-1] == ',' or word[len(word)-1] == '.':
                                word = word[:len(word)-1]
                            listOfWords.append(word)
                            phrase = previous+" "+word
                            listOfPhrases.append(phrase)
                            previous = word

                    totalWords = len(listOfWords)
    # Confirms text file is valid for a company and contained ticker symbol.
                    if ticker != "":
                        file.write(finalDate)
                        file.write(",")
                        file.write(ticker)
                        file.write(",")
                        # Determine word frequency for all keywords.
                        for word in WordsOfInterest:
                            value = listOfWords.count(word)
                            if value == 0:
```

```
            file.write("0")
        if value != 0:
            file.write(str(value))
        file.write(",")
        value = 0
    for phrase in PhrasesOfInterest:
        value = listOfPhrases.count(phrase)
        if value == 0:
            file.write("0")
        if value != 0:
            file.write(str(value))
        file.write(",")
        value = 0
    # Write number of total words in transcript.
    file.write(str(totalWords))
    file.write('\n')
```

Quantopian Code
(Year Long Holding Period):

```
from quantopian.algorithm import attach_pipeline, pipeline_output
from quantopian.pipeline import Pipeline
from quantopian.pipeline.data.builtin import USEquityPricing
from quantopian.pipeline.factors import AverageDollarVolume
from quantopian.pipeline.data import morningstar
from quantopian.pipeline.filters.morningstar import Q1500US
from dateutil.relativedelta import relativedelta
from datetime import datetime
import pandas as pd
tickerCollection = []
monthCounter = 0


def preview(df):
    log.info(df.head())
    return df

def initialize(context):
    """
    Called once at the start of the algorithm.
    """


fetch_csv('https://www.dl.dropboxusercontent.com/s/xa9ewlazvj1s7s3/ScoringCSV
2.csv?dl=0', date_column='Date', date_format='%m/%d/%y', pre_func=preview)

    context.stocks = symbols('AAPL', 'MSFT')

    # Rebalance every day, 1 hour after market open.
    schedule_function(my_monthly_trade, date_rules.month_start(),
time_rules.market_open(hours=1))

    # Record tracking variables at the end of each day.
    schedule_function(my_record_vars, date_rules.every_day(),
time_rules.market_close())

    # Create our middle cap, cheap, highly levered universe.
    pipe = attach_pipeline(make_pipeline(), 'pipe')

    ebitda = morningstar.income_statement.ebitda.latest
    long_term_debt = morningstar.balance_sheet.long_term_debt.latest
    #long_term_debty2 = morningstar.balance_sheet.long_term_debt.one_year_ago
    enterprise_value = morningstar.valuation.enterprise_value.latest
    market_cap = morningstar.valuation.market_cap.latest
```

```
    pipe.add(ebitda, 'ebitda')
    pipe.add(long_term_debt, 'long_term_debt')
    pipe.add(enterprise_value, 'enterprise_value')
    pipe.add(market_cap, 'market_cap')

    nonzero = (ebitda != 0)
    ev_eb = enterprise_value/ebitda
    ev_eb_ratio = ev_eb.percentile_between(0,50)
    ltd_eb = long_term_debt/ebitda
    ltd_eb_ratio = ltd_eb.percentile_between(50,100)
    percentile = market_cap.percentile_between(25,75)

    pipe.set_screen(nonzero & ev_eb_ratio & ltd_eb_ratio & percentile,
overwrite=True)

def make_pipeline():
    """
    A function to create our dynamic stock selector (pipeline). Documentation
on
    pipeline can be found here: https://www.quantopian.com/help#pipeline-
title
    """

    # Base universe set to the Q500US
    base_universe = Q1500US()

    # Factor of yesterday's close price.
    yesterday_close = USEquityPricing.close.latest

    pipe = Pipeline(
        screen = base_universe,
        columns = {
            'close': yesterday_close,
        }
    )
    return pipe



def my_monthly_trade(context,data):
    """
    Execute orders according to our schedule_function() timing.
    """
    context.output = pipeline_output('pipe')

    context.security_list = context.output.index

    global monthCounter
```

```
for stock in context.portfolio.positions:
    order_target_percent(stock, 0)



for stock in context.security_list:
    if data.can_trade(stock):
        if data.current(stock, 'indicator') > 0:

            last_date_str = str(data.current(stock, 'indicatordate'))
            if(last_date_str != 'nan'):

             last_date = datetime.strptime(last_date_str, "%m/%d/%y")
                today = str(get_datetime(None))
              today_date = datetime.strptime(today[0:10], "%Y-%m-%d")
                difference = (today_date - last_date).days

                score = data.current(stock, 'indicator')
                if score == 4 and difference < 295:
                    order_target_percent(stock, .05)
                elif score == 3 and difference < 190:
                    order_target_percent(stock, .05)
                elif score == 2 and difference < 100:
                    order_target_percent(stock, .05)
                elif score == 1 and difference < 10:
                    order_target_percent(stock, .05)
```

Quantopian Code
(Daily Shuffle, Top Ten Basket):

```
from quantopian.algorithm import attach_pipeline, pipeline_output
from quantopian.pipeline import Pipeline
from quantopian.pipeline.data.builtin import USEquityPricing
from quantopian.pipeline.factors import AverageDollarVolume
from quantopian.pipeline.data import morningstar
from quantopian.pipeline.filters.morningstar import Q1500US
from dateutil.relativedelta import relativedelta
from datetime import datetime
import pandas as pd
tickerCollection = []
dayCounter = 0

def preview(df):
    log.info(df.head())
    return df

def initialize(context):

    context.stock_list = [sid(24)]
    context.min_stock = sid(24

    """
    Called once at the start of the algorithm.
    """

fetch_csv('https://www.dl.dropboxusercontent.com/s/jtxuk0152xp7m6o/NEWSCORES.
csv?dl=0', date_column='Date', date_format='%m/%d/%y', pre_func=preview)

    # Rebalance every day, 1 hour after market open.
    schedule_function(my_daily_trade, date_rules.every_day(),
time_rules.market_open(hours=1))

    # Record tracking variables at the end of each day.
    schedule_function(my_record_vars, date_rules.every_day(),
time_rules.market_close())

    # Create our dynamic stock selector.
    pipe = attach_pipeline(make_pipeline(), 'pipe')

    ebitda = morningstar.income_statement.ebitda.latest
    long_term_debt = morningstar.balance_sheet.long_term_debt.latest
    #long_term_debty2 = morningstar.balance_sheet.long_term_debt.one_year_ago
    enterprise_value = morningstar.valuation.enterprise_value.latest
    market_cap = morningstar.valuation.market_cap.latest
```

```
    pipe.add(ebitda, 'ebitda')
    pipe.add(long_term_debt, 'long_term_debt')
    pipe.add(enterprise_value, 'enterprise_value')
    pipe.add(market_cap, 'market_cap')

    nonzero = (ebitda != 0)
    ev_eb = enterprise_value/ebitda
    ev_eb_ratio = ev_eb.percentile_between(0,50)
    ltd_eb = long_term_debt/ebitda
    ltd_eb_ratio = ltd_eb.percentile_between(50,100)
    percentile = market_cap.percentile_between(25,75)

    pipe.set_screen(nonzero & ev_eb_ratio & ltd_eb_ratio & percentile,
overwrite=True)

def make_pipeline():
    """
    A function to create our dynamic stock selector (pipeline). Documentation
on
    pipeline can be found here: https://www.quantopian.com/help#pipeline-
title
    """

    # Base universe set to the Q500US
    base_universe = Q1500US()

    # Factor of yesterday's close price.
    yesterday_close = USEquityPricing.close.latest

    pipe = Pipeline(
        screen = base_universe,
        columns = {
            'close': yesterday_close,
        }
    )
    return pipe


def my_daily_trade(context,data):
    """
    Execute orders according to our schedule_function() timing.
    """
    context.output = pipeline_output('pipe')
    context.security_list = context.output.index

    global dayCounter

    print len(context.security_list)
    #buy top 10 stocks to start
    if dayCounter == 0 or len(context.stock_list) < 10:
        minval = -1
        for stock in context.security_list:
            if len(context.stock_list) < 10:
```

```
                context.stock_list.append(stock)
                if data.current(stock, 'indicator') >= minval:
                   context.min_stock = stock
                   minval = data.current(stock, 'indicator')
            else:
                   if data.current(stock, 'indicator') >= minval:
                       context.stock_list.append(stock)
                       context.stock_list.remove(context.min_stock)
                       context.min_stock = stock
                       minval = data.current(stock, 'indicator')

        for stock in context.stock_list:
             order_target_percent(stock, 1.0/10)

    else:
        today = str(get_datetime(None))
        today_date = datetime.strptime(today[0:10], "%Y-%m-%d")
        for stock in context.security_list:
            if data.can_trade(stock):
                last_date_str = str(data.current(stock, 'indicatordate'))
                if(last_date_str != 'nan'):

                    last_date = datetime.strptime(last_date_str, "%m/%d/%y")
                    #check if new transcript came out today
                    if (today_date-last_date).days == 0:
                        new_score = data.current(stock, 'indicator')
                        if new_score < data.current(context.min_stock,
'indicator'):
                            order_target_percent(context.min_stock, 0)
                            context.stock_list.remove(context.min_stock)
                            order_target_percent(stock, 1.0/10)
                            context.stock_list.append(stock)
                            updateMin(context, data)

    print len(context.stock_list)
    dayCounter = dayCounter + 1
    pass

def updateMin(context, data):
    minval = -1
    for stock in context.stock_list:
        if data.current(stock, 'indicator') >= minval:
            context.min_stock = stock
            minval = data.current(stock, 'indicator')
```