Predicting Future   
Financial Performance  
using 10-k’s textual  
and numerical data

Professor: Dr. Anol Bhattacherjee

Team members:

Alysson Silveira

Anurag Shrivastava

Pranali Kanade

# Executive Summary

This work proposed to verify if the use of features derived from textual data, when combined with commonly used financial ratios, can help to make better predictions on corporate financial performance. We applied different machine learning techniques using 10-K fillings of US-based companies from 2012 to 2019 as our corpus and concluded that more complex models such as CNN and RNN are more suited to our intended purpose. The use of simpler word representations like TF-IDF, with or without POS filter, or word to vector, did not show to be effective when using tree-based models.

# Problem Definition & Significance

Different groups of people have interest in firm’s operations and decisions, either because of a legal relationship, represented by explicit or implicit contracts, or even because of a moral relationship, in which some groups claim to be respectfully treated and expect to have their opinions considered on the business decision process (Carroll, 1991). The author highlights that the stakeholder theory gave names and faces to those groups that are most urgent to business, such as: (1) owners, (2) customers, (3) employees, (4) community, (5) competitors, (6) suppliers, (7) social activist groups, (8) public at large, etc.

Finding ways of predicting future performance can have different implications for each group of stakeholders. For instance, owners (or investors) can make decisions concerning their equity participation (selling or buying shares), and even influence the strategy depending on the governance structure of the company. Employees on its turn can use performance predictions to base individual decisions such as stay or pursuit another job, or collective decisions such as demanding for more benefits. Competitors can perhaps set or adjust strategies and tactics to respond market leaders actions, while suppliers can use this information to decide whether to engage or not in new projects involving high R&D costs, or even to decide to adjust their accounts receivables policies, granting more or less credit to customer companies.

Numerous studies are dedicated to exploring the organizational performance. On one hand, most of them are focused on assessing the relationship between performance and other constructs, such as, customer satisfaction (Ittner and Larcker, 1998), social-responsibility (McGuire *et al.*, 1988; Pava and Krausz, 1996; Simpson and Kohers, 2002; Barnett and Salomon, 2006; Margolis *et al.*, 2007), intellectual capital (Tan *et al.*, 2007; Iswatia and Anshoria, 2007), human resource practices (Huselid, 1995), managerial compensation (Gerhart and Milkovich, 1990), stakeholder relations (Berman *et al.*,1999; Choi and Wang, 2009), and even family presence on director’s board (Perlin *et al.,* 2019). On the other hand, there are studies aiming to predict future performance using both statistical techniques (Finger, 1994; Fairfield *et al*., 2009; Orpurt and Zang, 2009; Yunus and Malik, 2012), and machine learning (Creamer and Freund, 2010; Qiu *et al.*, 2014; Hájek *et al.*, 2014; Chen and Howard, 2016; Chang *et al.,* 2016; Lee *et al.*, 2017; Meier *et al.*, 2018).

This work proposes to test if the use of financial ratios combined with textual data from financial statements are more effective on predicting future financial performance of companies.

# Prior Literature

Although the first developments on Natural Language Processing (NLP) goes back to 1950’s, these techniques became more and more popular in academic studies in the past decade, especially when it comes to empirical studies. These advancements can be credited to a variety of reasons, including increase on computational power of personal computers, increasing availability of open data sources, increasing availability of open source software and frameworks, and so on.

Beside the growth in popularity of these techniques lately, the number of published studies in the financial field is still small. The sources of textual data are usually social media posts, , news, and also financial statements. As this study focus on financial statements as sources of data, we observed that published papers tend to be diverse in terms of goals and algorithms applied. One group of articles that apply NLP to financial statements is focused on data extraction to produce more automated pipelines for financial data acquisition (Seng and Lai, 2010; Hering, 2016).The other group of studies is dedicated to the analysis of textual data from financial statements. While some of the works use more simplistic textual data representations and techniques, such as word and page counts, term-document matrix using tf-idf, and even sentiment analysis (Li, 2010; Cho *et al.*, 2010; Loughran and McDonald, 2011; Chen *et al.,* 2013; Meier *et al.*, 2018; El-Haj et al., 2020), more recently, more studies are being published with the application of more advanced algorithms: Recurrent Neural Network – RNN - (Sifa *et al.,* 2019; Masson and Paroubek, 2020), and even combinations of Convolutional Neural Networks and RNNs (Martin *et al.,* 2019).

# Data Source & Preparation

# Textual data

As the goal of our analysis is to check the effectiveness of combining textual and numerical data from financial statements on predicting financial performance, a natural choice of source would be an agent that officially receives and publishes financial statements from all companies open capital companies within a country. In the United States, the Securities and Exchange Commission is the agent responsible for:

“overseeing the nation’s securities markets and certain primary participants, including broker-dealers, investment companies, investment advisers, clearing agencies, transfer agents, credit rating agencies, and securities exchanges, as well as organizations such as the Financial Industry Regulatory Authority, Municipal Securities Rulemaking Board, and the Public Company Accounting Oversight Board” (SEC, 2019, p.p. 7)

SEC quarterly updates the Financial Statement and Notes Data Sets that reunites all textual and numerical data from financial statements, including notes sections[[1]](#footnote-1). For each period, the data extracted from corporate financial reports using eXtensible Business Reporting Language (XBRL) is made available in eight tables (SEC, 2019):

* SUB – Submission data set,
* TAG – Tag data set,
* DIM – Dimension data set,
* NUM – Number data set,
* TXT – Text data set,
* REN – Rendering data set,
* PRE – Presentation data set, and
* CAL – Calculation data set.

Our data preparation process included the combination between fields from different tables, so we could gather all the textual data required for this project, together with company identification data to assure data quality, avoiding repetition of observations and misclassifications. To avoid discrepancy in terms of statements formats, we decided to focus our analysis on 10-K statements, there are audited financial statements that also offers a broad overview of company’s business and financial situation (SEC, 2020).

When analyzing the available data, we could observe that the number of companies that filled 10-k presents a small variation overtime, but when we analyzed the number of text blocks available for each year, 2011 seemed to have some inconsistency when compared to the following years. Besides 2011, we also decided not to use reports from 2020, as they are still not complete, and so, as consequence, our analysis comprehends the period from 2012 until 2019. Data was preprocessed by removing stop words, non-printable characters, punctuation marks, and numeric digits, then tokenization is used for breaking a document down into words. After tokenization, lemmatization was applied to reduce different forms of each word into single items that contain the same meaning. Also, we tested different types of word representation in our analysis. This process, text vectorization, is used to map words or phrases from vocabulary to a corresponding vector of real numbers which is used to find word predictions, word similarities/semantics. We have used TF-IDF, TF-IDF with POS filter (Nouns), and Word to Vector.

Sample of 10K document:



# Numerical data

Although the numerical data from financial statements are also included in the SEC’s Financial Statement and Notes Data Sets, those are far away from being ready to use. As many companies have complex organization structure, with sometimes various child companies, finding or calculating the numbers that represents the whole conglomerate can be challenging. In this study, we decided to use another source for numerical data: the SimFin API. This API also gather data from SEC’s data sets, but it presents preprocessed items (i.e. Revenue), which was extremely beneficial to this project due to time constraints.

Having the correct numerical information for each company is just the first step, as different companies, even within the same segment, can present huge variations terms of financial indicators, such as Revenue, Costs, Income. One way of making companies more comparable is to calculate ratios. These ratios can either be calculated by combining items within the same statement, such as gross profit margin, that is the gross profit divided by revenue (both items can be found in the Income Statements), or even by combining items from different statements, like, for instance, Days of Outstanding Sales, that is calculated using items from Balance Sheet and Income Statement. The list of ratios that we applied in this study, as well as their formulas are presented on Table 1.

|  |  |
| --- | --- |
| Name | Formula |
| Total Debt | Short Term Debt + Long Term Debt |
| Size | Log(Total Assets) |
| Leverage | Total Debt / Total Assets |
| EBIT | Revenue - Cost of Revenue - Selling, General & Administrative |
| EBITDA | EBIT + Depreciation & Amortization |
| ROA | Income (Loss) from Continuing Operations/ Total Assets |
| ROE | Income (Loss) from Continuing Operations/ Total Equity |
| ROIC | Income (Loss) from Continuing Operations /  (Total Assets - Cash & Equivalents - LT Investments & Receivables) |
| Gross Profit Margin | Gross Profit / Revenue |
| EBIT Margin | EBIT / Revenue |
| Net Profit Margin | Income (Loss) from Continuing Operations/ Revenue |
| Asset Turnover | Revenue / Property, Plant & Equipment, Net |
| Financial Leverage | Income Continuing Oper. \* Total Assets / Total Equity /  (Income Continuing Oper. - Non-Oper. Income (Loss)) |
| Operating Leverage | Gross Profit / (Gross Profit - Selling, General & Administrative) |
| Days of Sales Outstanding | (Accounts & Notes Receivable \* 360) / Revenue |
| Days of Payments Outstanding | (Payables & Accruals \* 360) / Revenue |
| Days of Inventory Outstanding | (Inventories \* 360) / Cost of Revenue |

Table 1 - Financial Indicators and Ratios

# Dependent variable

Finally, the last stage of our preparation process was to calculate a dependent variable that measures corporate performance. Choosing a variable that serves as a proxy for this construct can challenging because a company have multiple objectives that can be weighted differently by each group of stakeholders, depending on their own goals. For instance, Carroll (1999) highlighted four types of goals a company can have, when introducing the pyramid of corporate social responsibility. The foundation of the proposed pyramid is the economic responsibility, represented by the profitability of the company, that supports the other blocks, namely, legal, ethical, and philanthropic responsibilities. The variety of groups of responsibilities can give us an idea of how diverse performance measures can be, depending on the focus of the study. Richard *et al.* (2009) also stated that the existence of different groups of interest, with distinct goals, adds complexity to the endeavor of measuring organization performance. The authors also include as other sources of complexity: the heterogeneity of resources, the environment, strategic choices, and the measurement timeframe. To overcome this challenge, Richard *et al.* (2009) proposes that studies should have a strong theoretical rationale to support the measures choices, making clear that the measure is aligned with research context.

In order to choose a performance measure, our strategy was first to focus on one group of stakeholders, the owners. Later we verified that most of the studies that focus on these stakeholders uses measures related to stock prices, while some studies applied measures related to profitability. Studies that uses stock market indicators tends to be aligned with the idea of perfect market, in which the stock prices reflect the firm’s future performance and considers that all investor have all the information available. We decided not to follow this path, because we believe that other factors, including psychological ones (Richard *et al.,* 2009), can affect stock markets, and so, measures derived from financial statements tend to be more reliable (as they follow predefined standards and are audited), and aligned to our goals.

Our dependent variable was derived from the comparison of each company’s income and the average income of their own industry segment. Our dependent variable has a binary nature and assumes value one when the company have income above the segment’s average, and zero otherwise. Also, as our goal is to predict future performance, for each observation (company per year), we considered the calculated dependent variable of the following year.

1. , then 1; otherwise 0. In which *j* is a company, *i* is the period, and *n* is the total number of companies of each segment.

# Exploratory Data Analysis & Visualizations

As stated before, we decided to focus our analysis on the period within 2012 and 2019, because the number of text blocks available on the 2011 data seems inconsistent with the rest of the data available (Figure 1).

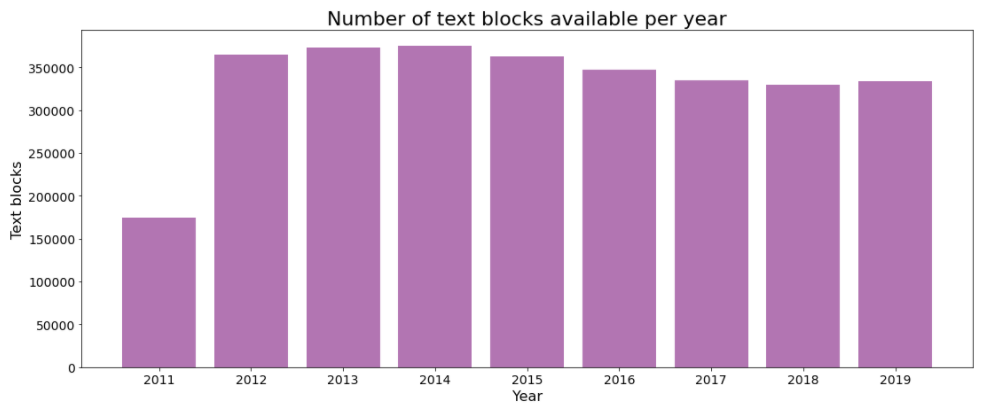


Figure 1 - Number of Text Blocks Available per Year

Also, the number of companies analyzed was lower than what we had originally available on the SEC data sets (Figure 2), because the SimFin API only shows data for US-based companies, instead of all companies that are obligated to filled 10-K with SEC. As consequence, we analyzed 1648 companies from 65 different segments, that accounted for a total of 8117 10-K documents from which we extracted 149,829,124 tokens.

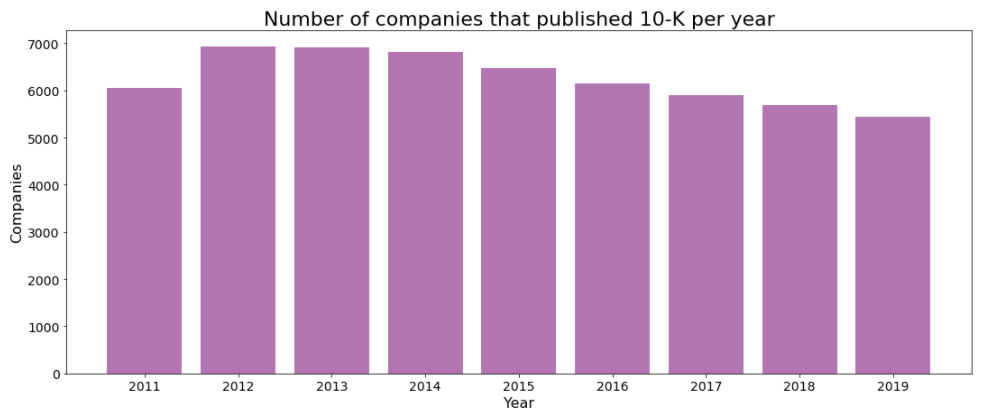
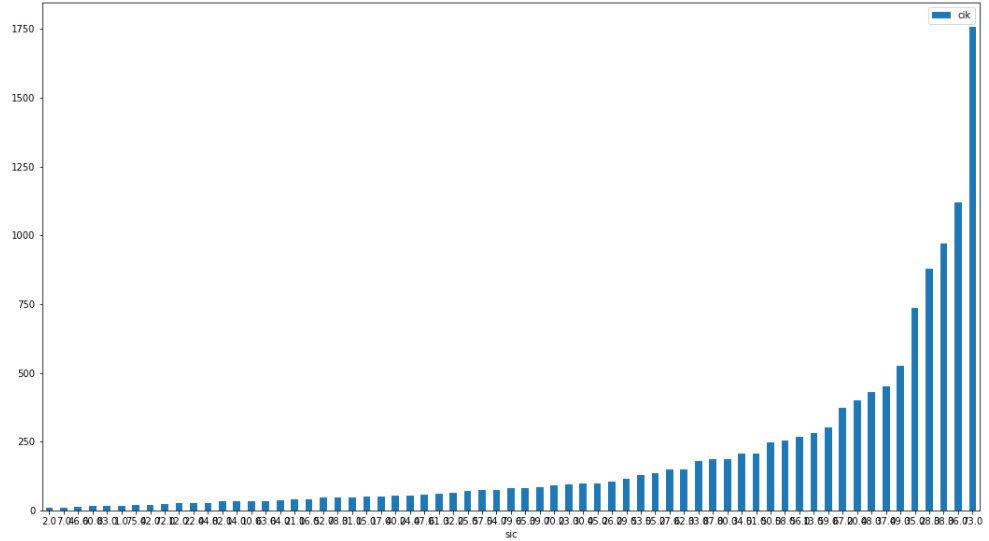


Figure 2- Number of Companies That Published 10-Ks per Year

As we can infer from Figure 3, the distribution of companies per segments is extremely disproportional. While we have almost 1750 published 10-Ks in the Service-Advertisement segment, over the period of our analysis, other segments such as Pipe Lines (No Natural Gas), Agricultural Services, and Agricultural Prod-Livestock & Animal Specialties have less than 20 overall. This divergence can also be better explored in future works.

Figure 3 - Distribution of Available 10-K Statements per Segments

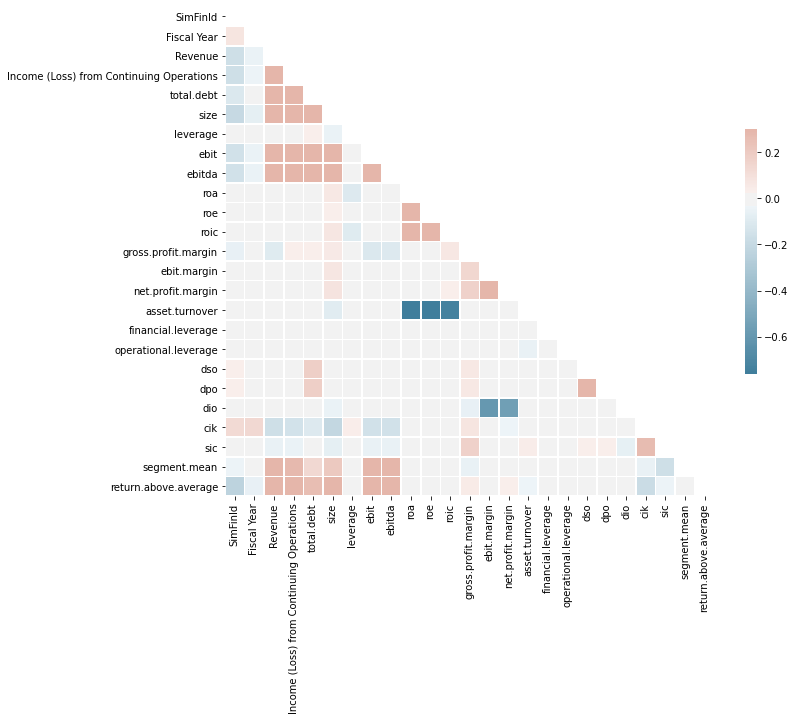


Figure 4 - Correlation Plot

After exploring the data sets in terms of number of companies, statements and text blocks available, we conducted a correlation analysis (Figure 4) on the numerical data, to assure that our independent variables don’t show any alarming correlation that could perhaps bias our models. None of the relationships presented values above 0.8 nor below -0.8, and so they were all kept for modeling.

# Text Analytics & Results

To accomplish our goal of testing if features derived from textual data can improve the accuracy of a financial performance predictive model, we first fitted a model only with numerical features, and later tested other models adding different types of textual-derived features. We chose lightGBM as our first algorithm, as it is a fast and accurate option that uses a tree-based framework. Anyhow, the results were not positive, as all the models that we tested with textual features performed worse than our baseline model that only had numerical features (Table 2). The measure we used to compare the models was Area Under the Curve (AUC).

|  |  |
| --- | --- |
| Model | AUC |
| Numerical features only (baseline) | **0.8862** |
| Numerical features + TF-IDF | 0.8812 |
| Numerical features + TF-IDF (nouns) | 0.8800 |
| Numerical features + Word to Vector | 0.8752 |
| Numerical features + Topic Analysis | 0.8800 |
| Numerical features + Sentiments | 0.8766 |

Table 2 - Comparison Between Models' AUC Scores

After the first set of models, all using lightGBM algorithm, we decided to apply a more complex approach, adapting the architecture suggest by Martin *et al.,* (2019), using a combination of CNN architecture on Word to Vector Inputs, and using it’s outputs together with the numerical features to feed a two-hidden layer neural network (Figure 5).

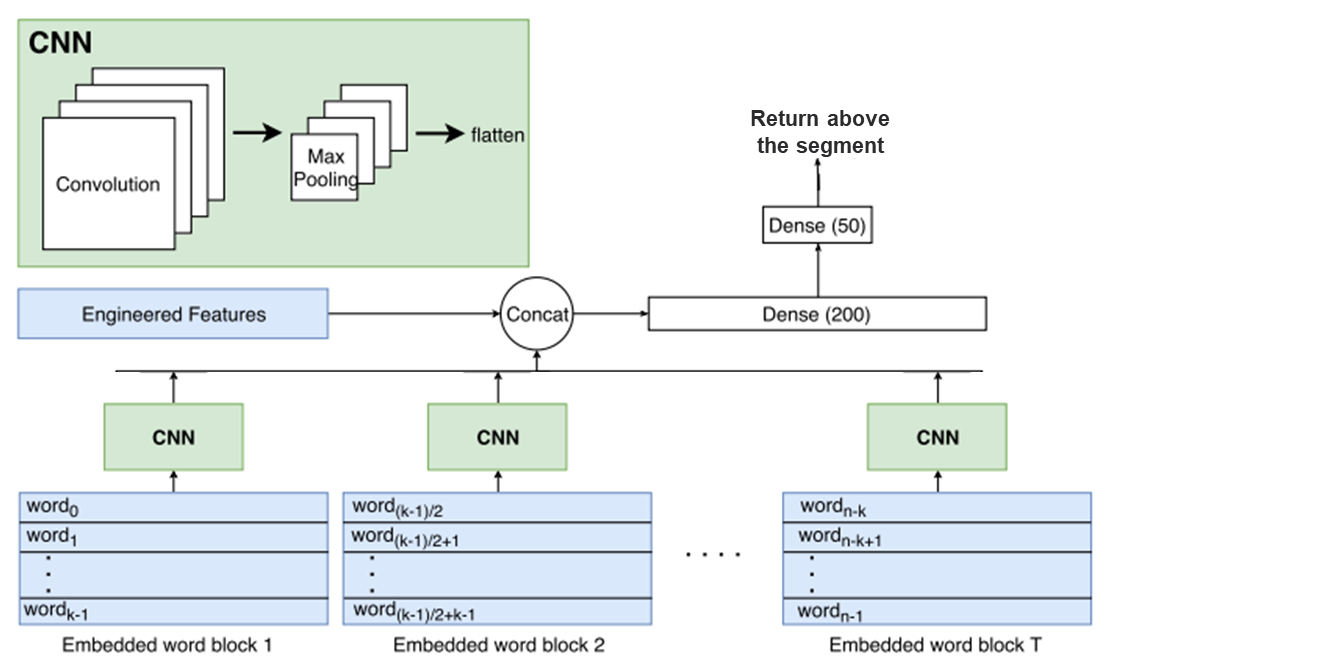


Figure 5 - Final Model Architecture. Adapted from Martin et al. (2019)

The new model achieved an AUC score of 0.9496 on test data, and so, performed better than our baseline model, that scored 0.8862, also using the test set (Table 3).

|  |  |  |
| --- | --- | --- |
| Algorithm | Model | AUC |
| lightGBM | Numerical features only (baseline) | 0.8862 |
| lightGBM | Numerical features + TF-IDF | 0.8812 |
| lightGBM | Numerical features + TF-IDF (nouns) | 0.8800 |
| lightGBM | Numerical features + Word to Vector | 0.8752 |
| lightGBM | Numerical features + Topic Analysis | 0.8800 |
| lightGBM | Numerical features + Sentiments | 0.8766 |
| **Neural Networks** | **Numerical features + (w2v + CNN + NN)** | **0.9496** |

Table 3 – Final Comparison Between Models' AUC Scores

# Insights & Recommendations

Opposed to what we expected, based on some previous studies findings, our work did not find evidence that the use of simple word representations improved models’ accuracies. The results were according to Meier *et al.* (2018) conclusions: bags of words and weighting methods did not have significant influence on predictive power.

Anyhow, the application of a more complex architecture has shown to be more effective in terms of AUC, when compared to simpler models. Even tough CNN are more suited to image data sets, the use of this algorithm was enough to reduce the dimensionality of the data and learn meaningful representations capable of inferring patterns to predict return above the average. Still, for future works, it would be interesting to test different architectures, more suited for textual data set, such as Long Short-Term Memory (LTSM) and Bidirectional LTSM.

In conclusion, our final model can be useful for company owners and investors when setting their short-term investment strategy (1 year ahead). Would also be even more fascinating to develop a model capable of doing good prediction for a longer time horizon.

# References

Barnett, M. L., & Salomon, R. M. (2006). Beyond dichotomy: The curvilinear relationship between social responsibility and financial performance. *Strategic management journal*, *27*(11), 1101-1122.

Berman, S. L., Wicks, A. C., Kotha, S., & Jones, T. M. (1999). Does stakeholder orientation matter? The relationship between stakeholder management models and firm financial performance. *Academy of Management journal*, *42*(5), 488-506.

Carroll, A. B. (1991). The pyramid of corporate social responsibility: Toward the moral management of organizational stakeholders. *Business horizons*, *34*(4), 39-48.

Chang, T. M., Hsu, M. F., Hu, G. H., & Lin, K. P. (2016). Salient corporate performance forecasting based on financial and textual information. In *2016 IEEE International Conference on Systems, Man, and Cybernetics (SMC)* (pp. 959-964). IEEE.

Chen, C. L., Liu, C. L., Chang, Y. C., & Tsai, H. P. (2012). Opinion mining for relating subjective expressions and annual earnings in US financial statements. *arXiv preprint arXiv:1210.3865*.

Chen, F. H., & Howard, H. (2016). An alternative model for the analysis of detecting electronic industries earnings management using stepwise regression, random forest, and decision tree. *Soft Computing*, *20*(5), 1945-1960.

Cho, C. H., Roberts, R. W., & Patten, D. M. (2010). The language of US corporate environmental disclosure. Accounting, Organizations and Society, 35(4), 431-443.

Choi, J., & Wang, H. (2009). Stakeholder relations and the persistence of corporate financial performance. *Strategic management journal*, *30*(8), 895-907.

Creamer, G., & Freund, Y. (2010). Using boosting for financial analysis and performance prediction: application to s&p 500 companies, latin american adrs and banks. *Computational Economics*, *36*(2), 133-151.

El-Haj, M., Alves, P., Rayson, P., Walker, M., & Young, S. (2020). Retrieving, classifying and analysing narrative commentary in unstructured (glossy) annual reports published as PDF files. *Accounting and Business Research*, *50*(1), 6-34.

Fairfield, P. M., Ramnath, S., & Yohn, T. L. (2009). Do industry‐level analyses improve forecasts of financial performance?. *Journal of Accounting Research*, *47*(1), 147-178.

Finger, C. A. (1994). The ability of earnings to predict future earnings and cash flow. *Journal of accounting research*, *32*(2), 210-223.

Gerhart, B., & Milkovich, G. T. (1990). Organizational differences in managerial compensation and financial performance. *Academy of Management journal*, *33*(4), 663-691.

Hájek, P., Olej, V., & Myskova, R. (2014). Forecasting corporate financial performance using sentiment in annual reports for stakeholders’ decision-making. *Technological and Economic Development of Economy*, *20*(4), 721-738.

Hering, J. (2016). The annual report algorithm: Retrieval of financial statements and extraction of textual information. *Available at SSRN 2870309*.

Huselid, M. A. (1995). The impact of human resource management practices on turnover, productivity, and corporate financial performance. *Academy of management journal*, *38*(3), 635-672.

Iswatia, S., & Anshoria, M. (2007). The influence of intellectual capital to financial performance at insurance companies in Jakarta Stock Exchange (JSE). In *Proceedings of the 13th Asia Pacific Management Conference, Melbourne, Australia* (pp. 1393-1399).

Ittner, C. D., & Larcker, D. F. (1998). Are nonfinancial measures leading indicators of financial performance? An analysis of customer satisfaction. *Journal of accounting research*, *36*, 1-35.

Lee, J., Jang, D., & Park, S. (2017). Deep learning-based corporate performance prediction model considering technical capability. *Sustainability*, *9*(6), 899.

Li, F. (2010). The information content of forward‐looking statements in corporate filings—A naïve Bayesian machine learning approach. Journal of Accounting Research, 48(5), 1049-1102.

Loughran, T., & McDonald, B. (2011). When is a liability not a liability? Textual analysis, dictionaries, and 10‐Ks. The Journal of Finance, 66(1), 35-65.

Margolis, J. D., Elfenbein, H. A., & Walsh, J. P. (2007). Does it pay to be good? A meta-analysis and redirection of research on the relationship between corporate social and financial performance. *Ann Arbor*, *1001*, 48109-1234.

McGuire, J. B., Sundgren, A., & Schneeweis, T. (1988). Corporate social responsibility and firm financial performance. *Academy of management Journal*, *31*(4), 854-872.

Meier, J. H., Esmatyar, W., & Frost, R. (2018). The Predictive Power of the Sentiment of Financial Reports. In *ICTERI Workshops* (pp. 30-44).

Orpurt, S. F., & Zang, Y. (2009). Do direct cash flow disclosures help predict future operating cash flows and earnings?. *The Accounting Review*, *84*(3), 893-935.

Pava, M. L., & Krausz, J. (1996). The association between corporate social-responsibility and financial performance: The paradox of social cost. *Journal of business Ethics*, *15*(3), 321-357.

Perlin, M. S., Kirch, G., & Vancin, D. (2019). Accessing financial reports and corporate events with GetDFPData. *Brazilian Review of Finance*, *17*(3), 85-108.

Qiu, X. Y., Srinivasan, P., & Hu, Y. (2014). Supervised learning models to predict firm performance with annual reports: An empirical study. *Journal of the Association for Information Science and Technology*, *65*(2), 400-413.

Richard, P. J., Devinney, T. M., Yip, G. S., & Johnson, G. (2009). Measuring organizational performance: Towards methodological best practice. *Journal of management*, *35*(3), 718-804.

Securities and Exchange Commission (SEC). (2019). *Fiscal Year 2019 Agency Financial Report*. U.S. Securities and Exchange Commission. <https://www.sec.gov/reports-and-publications/annual-reports/sec-2019-agency-financial-report>

Securities and Exchange Commission (SEC). (2020). Retrieved November 08, 2020, from <https://www.investor.gov/introduction-investing/investing-basics/glossary/form-10-k>

Seng, J. L., & Lai, J. T. (2010). An Intelligent information segmentation approach to extract financial data for business valuation. *Expert Systems with Applications*, *37*(9), 6515-6530.

Simpson, W. G., & Kohers, T. (2002). The link between corporate social and financial performance: Evidence from the banking industry. *Journal of business ethics*, *35*(2), 97-109.

Tan, H. P., Plowman, D., & Hancock, P. (2007). Intellectual capital and financial returns of companies. *Journal of Intellectual capital*.

Yunus, N. M., & Malik, S. A. (2012). Developing financial model using financial ratios to predict business performance of IBS construction company in Selangor. In *2012 International Conference on Innovation Management and Technology Research* (pp. 441-445). IEEE.

# Appendix A

Our python codes are divided into two notebooks. Appendix A presents the final model, using Neural Networks.

In [54]:

import pandas as pd

import simfin as sf

from simfin.names import \*

import numpy as np

Import financial data from SimFin API[¶](#Import-financial-data-from-SimFin-API)

In [2]:

sf.set\_data\_dir('~/simfin\_data/')

sf.set\_api\_key(api\_key='free')

df\_income = sf.load\_income(variant='annual', market='us')

df\_balance = sf.load\_balance(variant='annual', market='us')

df\_cashflow = sf.load\_cashflow(variant='annual', market='us')

df\_companies = sf.load\_companies(index=TICKER, market='us')

df\_industries = sf.load\_industries()

Dataset "us-income-annual" on disk (10 days old).

- Loading from disk ... Done!

Dataset "us-balance-annual" on disk (10 days old).

- Loading from disk ... Done!

Dataset "us-cashflow-annual" on disk (10 days old).

- Loading from disk ... Done!

Dataset "us-companies" on disk (10 days old).

- Loading from disk ... Done!

Dataset "industries" on disk (10 days old).

- Loading from disk ... Done!

In [3]:

df\_income\_reduced = df\_income[['SimFinId', 'Currency', 'Fiscal Year','Income (Loss) from Continuing Operations',

'Revenue','Cost of Revenue', 'Gross Profit','Selling, General & Administrative',

'Depreciation & Amortization', 'Non-Operating Income (Loss)']]

df\_balance\_reduced = df\_balance[['SimFinId', 'Currency', 'Fiscal Year','Total Assets','Total Equity',

'Short Term Debt','Long Term Debt', 'Cash, Cash Equivalents & Short Term Investments',

'Long Term Investments & Receivables','Accounts & Notes Receivable','Payables & Accruals',

'Inventories','Property, Plant & Equipment, Net']]

df\_cashflow\_reduced = df\_cashflow[['SimFinId', 'Currency', 'Fiscal Year','Net Cash from Operating Activities',

'Net Cash from Investing Activities', 'Net Cash from Financing Activities']]

Extracting the sign of each cashflow item to calculate lifecycle[¶](#Extracting-the-sign-of-each-cashflow-it)

In [4]:

pd.set\_option('mode.chained\_assignment', None)

df\_cashflow\_reduced['Op\_sign'] = ["+" if company >= 0 else "-" for company in df\_cashflow\_reduced['Net Cash from Operating Activities']]

df\_cashflow\_reduced['Inv\_sign'] = ["+" if company >= 0 else "-" for company in df\_cashflow\_reduced['Net Cash from Investing Activities']]

df\_cashflow\_reduced['Fin\_sign'] = ["+" if company >= 0 else "-" for company in df\_cashflow\_reduced['Net Cash from Financing Activities']]

In [5]:

df\_cashflow\_reduced['lifecycle'] = None

for row in range(0,len(df\_cashflow\_reduced)):

if (df\_cashflow\_reduced['Op\_sign'][row] == '-') & (df\_cashflow\_reduced['Inv\_sign'][row] == '-') & (df\_cashflow\_reduced['Fin\_sign'][row]=="+"):

df\_cashflow\_reduced['lifecycle'][row] = 'Introduction'

elif (df\_cashflow\_reduced['Op\_sign'][row] == '+') & (df\_cashflow\_reduced['Inv\_sign'][row] == '-') & (df\_cashflow\_reduced['Fin\_sign'][row]=="+"):

df\_cashflow\_reduced['lifecycle'][row] = 'Growth'

elif (df\_cashflow\_reduced['Op\_sign'][row] == '+') & (df\_cashflow\_reduced['Inv\_sign'][row] == '-') & (df\_cashflow\_reduced['Fin\_sign'][row]=="-"):

df\_cashflow\_reduced['lifecycle'][row] = 'Mature'

elif (df\_cashflow\_reduced['Op\_sign'][row] == '-') & (df\_cashflow\_reduced['Inv\_sign'][row] == '-') & (df\_cashflow\_reduced['Fin\_sign'][row]=="-"):

df\_cashflow\_reduced['lifecycle'][row] = 'Shake-out'

elif (df\_cashflow\_reduced['Op\_sign'][row] == '+') & (df\_cashflow\_reduced['Inv\_sign'][row] == '+') & (df\_cashflow\_reduced['Fin\_sign'][row]=="+"):

df\_cashflow\_reduced['lifecycle'][row] = 'Shake-out'

elif (df\_cashflow\_reduced['Op\_sign'][row] == '+') & (df\_cashflow\_reduced['Inv\_sign'][row] == '+') & (df\_cashflow\_reduced['Fin\_sign'][row]=="-"):

df\_cashflow\_reduced['lifecycle'][row] = 'Shake-out'

else:

df\_cashflow\_reduced['lifecycle'][row] = 'Decline'

In [6]:

df\_cashflow\_reduced.groupby(['lifecycle'])['lifecycle'].count()

Out[6]:

lifecycle

Decline 773

Growth 4240

Introduction 1681

Mature 8404

Shake-out 1395

Name: lifecycle, dtype: int64

In [7]:

df\_cashflow\_lifecycle = df\_cashflow\_reduced.drop(['Net Cash from Operating Activities','Net Cash from Investing Activities',

'Net Cash from Financing Activities', 'Op\_sign', 'Inv\_sign', 'Fin\_sign'], axis=1)

In [8]:

df\_financial = df\_income\_reduced.merge(df\_balance\_reduced, on=['Report Date', 'SimFinId', 'Currency', 'Fiscal Year']).merge(df\_cashflow\_lifecycle, on=['Report Date', 'SimFinId', 'Currency', 'Fiscal Year'])

df\_financial.reset\_index(level=0, inplace=True)

Ckecking null values on financial data set[¶](#Ckecking-null-values-on-financial-data-)

In [9]:

print(df\_financial.isnull().sum()/len(df\_financial))

Report Date 0.000000

SimFinId 0.000000

Currency 0.000000

Fiscal Year 0.000000

Income (Loss) from Continuing Operations 0.000000

Revenue 0.015886

Cost of Revenue 0.093919

Gross Profit 0.015219

Selling, General & Administrative 0.049961

Depreciation & Amortization 0.563027

Non-Operating Income (Loss) 0.011156

Total Assets 0.000000

Total Equity 0.000121

Short Term Debt 0.337477

Long Term Debt 0.219548

Cash, Cash Equivalents & Short Term Investments 0.002971

Long Term Investments & Receivables 0.711999

Accounts & Notes Receivable 0.084642

Payables & Accruals 0.004669

Inventories 0.306736

Property, Plant & Equipment, Net 0.021100

lifecycle 0.000000

dtype: float64

In [10]:

print(len(df\_financial))

16493

Dropping companies that didn't report revenues or costs. Problably holdings that don't have operation.[¶](#Dropping-companies-that-didn't-report-r)

In [11]:

drop\_row\_null = list(df\_financial.index[df\_financial['Revenue'].isnull() == True]) # 262 rows

df\_financial = df\_financial.drop(drop\_row\_null)

drop\_row\_zero = list(df\_financial.index[df\_financial['Revenue']==0]) #135 rows

df\_financial = df\_financial.drop(drop\_row\_zero)

drop\_row\_null\_cost = list(df\_financial.index[df\_financial['Cost of Revenue'].isnull() == True]) # 262 rows

df\_financial = df\_financial.drop(drop\_row\_null\_cost)

drop\_row\_zero\_cost = list(df\_financial.index[df\_financial['Cost of Revenue']==0]) #32 rows

df\_financial = df\_financial.drop(drop\_row\_zero\_cost)

len(df\_financial)

Out[11]:

14872

Some companies doesn't report values for some of the lines because they don't have it. So it was replaced by zero[¶](#Some-companies-doesn't-report-values-fo)

In [12]:

df\_financial = df\_financial.fillna(0)

Calculating numerical dependent variables[¶](#Calculating-numerical-dependent-variabl)

In [13]:

#Some of the items already carries negative values (e.g. Cost of Revenue and Selling, General & Administrative)

#All formulas were adapted

df\_financial['total.debt'] = df\_financial['Short Term Debt'] + df\_financial['Long Term Debt']

df\_financial['size'] = np.log(1+df\_financial['Total Assets'])

df\_financial['leverage'] = df\_financial['total.debt']/df\_financial['Total Assets']

df\_financial['ebit'] = df\_financial['Revenue'] + df\_financial['Selling, General & Administrative']

df\_financial['ebitda'] = df\_financial['ebit'] - df\_financial['Depreciation & Amortization']

df\_financial['roa'] = df\_financial['Income (Loss) from Continuing Operations']/df\_financial['Total Assets']

df\_financial['roe'] = df\_financial['Income (Loss) from Continuing Operations']/df\_financial['Total Equity']

df\_financial['roic'] = df\_financial['Income (Loss) from Continuing Operations']/(df\_financial['Total Assets'] - df\_financial['Cash, Cash Equivalents & Short Term Investments'] - df\_financial['Long Term Investments & Receivables'])

df\_financial['gross.profit.margin'] = df\_financial['Gross Profit']/df\_financial['Revenue']

df\_financial['ebit.margin'] = df\_financial['ebit']/df\_financial['Revenue']

df\_financial['net.profit.margin'] = df\_financial['Income (Loss) from Continuing Operations']/df\_financial['Revenue']

df\_financial['asset.turnover'] = df\_financial['Revenue']/df\_financial['Property, Plant & Equipment, Net']

df\_financial['financial.leverage'] = df\_financial['Income (Loss) from Continuing Operations']\*df\_financial['Total Assets']/df\_financial['Total Equity']/(df\_financial['Income (Loss) from Continuing Operations']-df\_financial['Non-Operating Income (Loss)'])

df\_financial['operational.leverage'] = df\_financial['Gross Profit']/(df\_financial['Gross Profit']+df\_financial['Selling, General & Administrative'])

df\_financial['dso'] = (df\_financial['Accounts & Notes Receivable']\*360)/df\_financial['Revenue']

df\_financial['dpo'] = (df\_financial['Accounts & Notes Receivable']\*360)/df\_financial['Revenue']

df\_financial['dio'] = (df\_financial['Inventories']\*360)/df\_financial['Cost of Revenue']\*-1

In [14]:

df\_financial = df\_financial[['Report Date','SimFinId', 'Currency', 'Fiscal Year', 'Revenue', 'Income (Loss) from Continuing Operations',

'total.debt', 'size', 'leverage', 'ebit', 'ebitda', 'roa', 'roe', 'roic',

'gross.profit.margin', 'ebit.margin', 'net.profit.margin',

'asset.turnover', 'financial.leverage', 'operational.leverage', 'dso',

'dpo', 'dio', 'lifecycle']]

Gathering Industry Segment Information, cik (unique) number, and company name[¶](#Gathering-Industry-Segment-Information,)

In [15]:

# merging tables to gather companies' names

df\_financial = df\_financial.merge(df\_companies, on ='SimFinId', how ='left').drop(['IndustryId'],axis=1)

companies\_txt = pd.read\_csv(r'C:\Users\alyss\Desktop\MS BAIS\Fall 2020\ISM6930 - Text Analytics\Team Project\Financial Statements NLP\Final Project\companies\_sec.csv')

companies\_txt = companies\_txt.drop('Unnamed: 0', axis=1)

In [16]:

# In the SEC dataset, sic numbers has 4 digits. We are spliting companies by the first 2 digits to reduce the number of industries.

import math

companies\_txt['sic'] = companies\_txt['sic']/100

companies\_txt['sic'] = np.floor(companies\_txt['sic'])

In [17]:

# Removing the punctuation from the names helped to reduce the number of missing data from 145 to 108

companies\_txt['name'] = companies\_txt['name'].str.replace(',',"")

companies\_txt['name'] = companies\_txt['name'].str.replace('.',"")

companies\_txt = companies\_txt.rename(columns={"name": "Company Name"})

companies\_txt = companies\_txt.drop\_duplicates(ignore\_index = True)

companies\_txt = companies\_txt.reset\_index(drop=True)

df\_financial['Company Name'] = df\_financial['Company Name'].str.upper()

df\_financial['Company Name'] = df\_financial['Company Name'].str.replace(',',"")

df\_financial['Company Name'] = df\_financial['Company Name'].str.replace('.',"")

df\_financial = df\_financial.merge(companies\_txt, on='Company Name', how='left')

In [18]:

# Checking companies that we couldn't find the segment

df\_financial['sic'].isnull().sum()/len(df\_financial['sic'])

Out[18]:

0.06224256292906179

dropping the few companies that we couldn't find the segment

In [19]:

df\_financial = df\_financial.drop(list(df\_financial.index[df\_financial['sic'].isnull()==True]))

In [20]:

df\_financial = df\_financial.reset\_index(drop=True)

Calculating the dependent variable[¶](#Calculating-the-dependent-variable)

In [21]:

segment\_mean = pd.DataFrame(df\_financial.groupby(['sic'])['Income (Loss) from Continuing Operations'].agg(np.mean))

segment\_mean = segment\_mean.rename(columns={'Income (Loss) from Continuing Operations':'segment.mean'})

In [22]:

df\_financial = df\_financial.merge(segment\_mean, on='sic', how = 'left')

In [23]:

df\_financial['return.above.average'] = [1 if df\_financial['Income (Loss) from Continuing Operations'][row] >= df\_financial['segment.mean'][row] else 0 for row in range(0, len(df\_financial)) ]

In [24]:

df\_financial = df\_financial.drop\_duplicates().reset\_index(drop=True)

In [25]:

df\_financial['return.above.average'] = df\_financial.groupby(['SimFinId'])['return.above.average'].shift(-1)

In [26]:

df\_financial = df\_financial.dropna().reset\_index(drop=True)

In [27]:

df\_financial['return.above.average'] = df\_financial['return.above.average'].astype('int')

In [196]:

import seaborn as sns

import matplotlib.pyplot as plt

corr = df\_financial.corr()

# Generate a mask for the upper triangle

mask = np.triu(np.ones\_like(corr, dtype=bool))

# Set up the matplotlib figure

f, ax = plt.subplots(figsize=(11, 9))

# Generate a custom diverging colormap

cmap = sns.diverging\_palette(230, 20, as\_cmap=True)

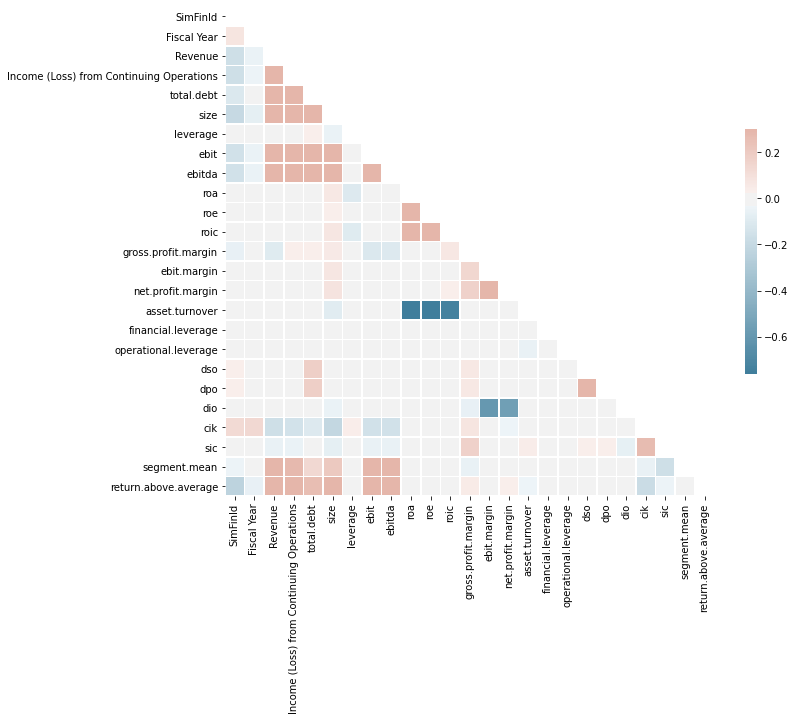
# Draw the heatmap with the mask and correct aspect ratio

sns.heatmap(corr, mask=mask, cmap=cmap, vmax=.3, center=0,

square=True, linewidths=.5, cbar\_kws={"shrink": .5})

Out[196]:

<matplotlib.axes.\_subplots.AxesSubplot at 0x2b34bec3c70>



In [236]:

comp\_seg = pd.DataFrame(df\_financial.groupby('sic')['cik'].count()).reset\_index()

comp\_seg = comp\_seg.sort\_values('cik')

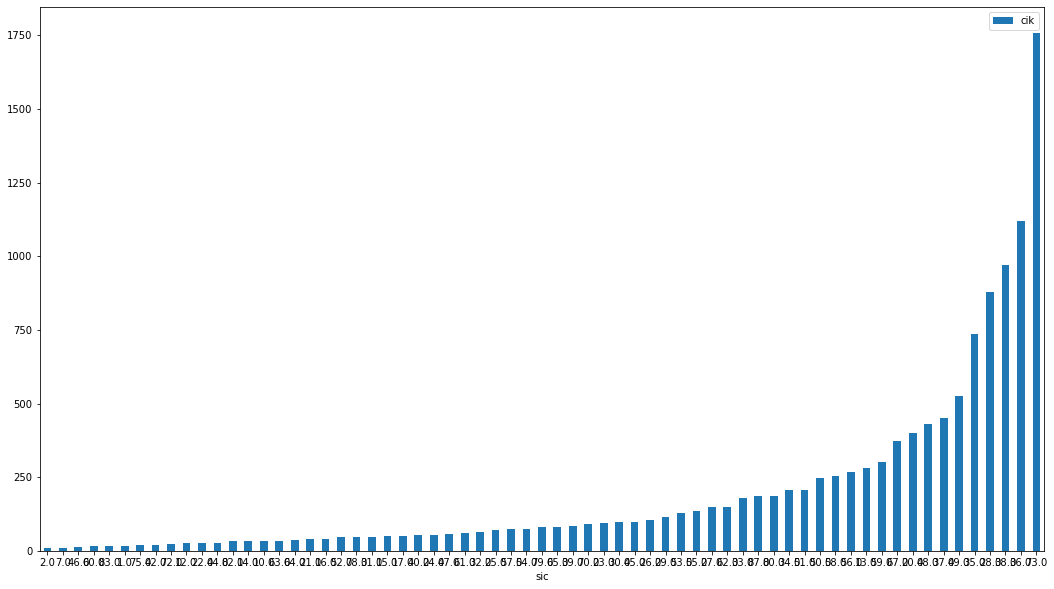
comp\_seg['sic'] = comp\_seg['sic'].astype('str')

import matplotlib.pyplot as plt

comp\_seg.plot.bar(x='sic', y='cik', rot=0, figsize=(18,10))

Out[236]:

<matplotlib.axes.\_subplots.AxesSubplot at 0x2b351398eb0>



Calculate segment growth

In [28]:

companies = df\_financial['cik'].drop\_duplicates()

In [29]:

df\_financial\_final = df\_financial.drop(['Report Date', 'SimFinId', 'Currency', 'Fiscal Year', 'Revenue', 'cik',

'Company Name', 'Income (Loss) from Continuing Operations',

'total.debt', 'ebit', 'ebitda', 'Income (Loss) from Continuing Operations',

'segment.mean'], axis=1)

Baseline model - without text features

In [30]:

#categorical features

from sklearn.preprocessing import OneHotEncoder

categorical\_features = ['sic', 'lifecycle']

df\_financial\_final[categorical\_features] = df\_financial\_final[categorical\_features].astype('str')

cat\_encoder = OneHotEncoder(sparse=False)

cat\_encoder = cat\_encoder.fit(df\_financial\_final[categorical\_features])

col\_names = cat\_encoder.get\_feature\_names(categorical\_features)

cat\_encoder = cat\_encoder.transform(df\_financial\_final[categorical\_features])

#Create a Pandas DataFrame of the hot encoded column

ohe\_df = pd.DataFrame(cat\_encoder, columns = col\_names)

#Concatenate with original data

df\_financial\_final = pd.concat([df\_financial\_final, ohe\_df], axis=1).drop(categorical\_features, axis=1)

In [31]:

df\_fin\_target = df\_financial\_final['return.above.average']

df\_fin\_X = df\_financial\_final.drop('return.above.average',axis=1)

In [32]:

df\_fin\_X.shape

Out[32]:

(12572, 84)

In [33]:

from sklearn.model\_selection import train\_test\_split

train\_x, test\_x, train\_y, test\_y = train\_test\_split(df\_fin\_X, df\_fin\_target, test\_size=0.25, random\_state=17)

In [34]:

# build the lightgbm model

import lightgbm as lgb

clf = lgb.LGBMClassifier(seed = 1)

clf.fit(train\_x, train\_y)

Out[34]:

LGBMClassifier(seed=1)

In [35]:

# view accuracy on test set

from sklearn.metrics import accuracy\_score, roc\_auc\_score

y\_pred=clf.predict(test\_x)

print("acc:",accuracy\_score(test\_y,y\_pred))

print("auc:",roc\_auc\_score(test\_y,y\_pred))

acc: 0.9172764874323894

auc: 0.8862810010099876

Importing textual data[¶](#Importing-textual-data)

In [36]:

import os

import pickle

from sklearn.feature\_extraction.text import TfidfVectorizer

from nltk.stem import WordNetLemmatizer

from nltk import word\_tokenize

In [37]:

#import txts files

with open('filename.pickle', 'rb') as handle:

txts = pickle.load(handle)

txts = txts.drop('Unnamed: 0', axis=1)

with open('subs.pickle', 'rb') as handle\_subs:

subs = pickle.load(handle\_subs)

In [38]:

#filter only service companies from subs file

df\_txt = subs[(subs['fp']=='FY') & (subs['fy']>2011) & (subs['fy']<2020) & subs['cik'].isin(companies)]

In [39]:

df\_txt\_selected = subs[['adsh', 'cik', 'fy']][subs['cik'].isin(companies)]

In [40]:

#get adsh for the selected companies

selected\_adsh = subs['adsh'][subs['cik'].isin(companies)]

# Check the proportion of companies - 23%

len(selected\_adsh)/len(subs)

Out[40]:

0.2300242559444769

In [41]:

#filter selected companies from txts file

txt\_selected = txts[txts['adsh'].isin(selected\_adsh)]

df\_txt = df\_txt\_selected.merge(txt\_selected, on='adsh')

In [42]:

len(df\_txt)

Out[42]:

884760

In [43]:

# Filtering only disclosure text blocks

df\_txt = df\_txt[df\_txt['tag'].str.contains('Disclosure')]

In [44]:

len(df\_txt)

Out[44]:

188794

Merging text blocks[¶](#Merging-text-blocks)

In [45]:

check\_comp = df\_financial.rename(columns={'Fiscal Year':'fy'})

In [46]:

df\_txt['value'] = df\_txt['value'].astype('str')

In [47]:

values = df\_txt.groupby(['adsh','fy'])['value'].agg(' '.join)

In [48]:

values = pd.DataFrame(values)

In [49]:

values.reset\_index(inplace=True)

In [50]:

values = values.merge(df\_txt\_selected, on=['adsh', 'fy'], how='left')

In [51]:

values\_n = values.drop('adsh', axis=1)

In [52]:

df\_merged = check\_comp.merge(values\_n, on=['cik', 'fy'], how = 'right')

In [56]:

df\_merged = df\_merged.replace([np.inf, -np.inf], np.nan)

In [101]:

df\_merged = df\_merged.dropna().reset\_index(drop=True)

Preprocessing texts[¶](#Preprocessing-texts)

In [60]:

#including some non-printable character in the stopwords

from gensim.parsing.preprocessing import STOPWORDS

my\_stop\_words = STOPWORDS.union(set(['A&amp;B', '&amp', '\0', '\1', '\2', '\3', '\4', '\5', '\6', '\a', '\b', '\t', '\n', '\v', '\f', '\r',

'\x0e', '\x0f', '\x10', '\x11', '\x12', '\x13', '\x14', '\x15', '\x16', '\x17', '\x18', '\x19', '\x1a',

'\x1b', '\x1c', '\x1d', '\x1e', '\x1f', '', 'nan', '\_', '\_\_', '\_\_\_', '\_\_\_\_', '\_\_\_\_\_', '\_\_\_\_\_\_', '\_\_\_\_\_\_\_', '\_\_\_\_\_\_\_\_',

'\_\_\_\_\_\_\_\_\_', '\_\_\_\_\_\_\_\_\_\_', '\_\_\_\_\_\_\_\_\_\_\_', '\_\_\_\_\_\_\_\_\_\_\_\_', '\_\_\_\_\_\_\_\_\_\_\_\_\_', '\_\_\_\_\_\_\_\_\_\_\_\_\_\_', '\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_',

'\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_', '\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_', '\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_']))

In [61]:

# tranform texts into a list to iterate faster

value\_list = list(df\_merged['value'].astype(str))

In [62]:

from gensim.utils import tokenize

tokens = [list(tokenize(doc, lower=True, deacc=True, encoding='utf8', errors='strict')) for doc in value\_list]

tokens\_str = [' '.join(row) for row in tokens]

In [63]:

from gensim.parsing.preprocessing import remove\_stopwords

tokens\_str = [remove\_stopwords(h) for h in tokens\_str]

In [64]:

from nltk.stem import WordNetLemmatizer

lemmatizer = WordNetLemmatizer()

lemma = [lemmatizer.lemmatize(h, 'v') for h in tokens\_str]

In [65]:

token\_list = [word.split(' ') for word in lemma]

In [66]:

token\_list = [word for word in token\_list if len(word)>2]

word to vector[¶](#word-to-vector)

In [ ]:

#from gensim.models import Word2Vec

#model = Word2Vec(token\_list, size=300, window=5, min\_count=2, sample=1e-3, sg=1, iter=5)

In [ ]:

#model.save("word2vec.model")

In [67]:

from gensim.models import Word2Vec

model = Word2Vec.load("word2vec.model")

In [68]:

word\_vectors = model.wv

In [69]:

embeddings\_index = {}

for w in word\_vectors.wv.vocab.keys():

embeddings\_index[w] = word\_vectors.wv[w]

<ipython-input-69-16010502633d>:2: DeprecationWarning: Call to deprecated `wv` (Attribute will be removed in 4.0.0, use self instead).

for w in word\_vectors.wv.vocab.keys():

<ipython-input-69-16010502633d>:3: DeprecationWarning: Call to deprecated `wv` (Attribute will be removed in 4.0.0, use self instead).

embeddings\_index[w] = word\_vectors.wv[w]

In [70]:

from keras.preprocessing.text import Tokenizer

from keras.preprocessing.sequence import pad\_sequences

tokenizer = Tokenizer(num\_words=50000)

tokenizer.fit\_on\_texts(token\_list)

sequences = tokenizer.texts\_to\_sequences(token\_list)

In [71]:

num\_words = 50000

embedding\_matrix = np.zeros((num\_words, 300))

for word, i in tokenizer.word\_index.items():

if i >= num\_words:

continue

embedding\_vector = embeddings\_index.get(word)

if embedding\_vector is not None:

embedding\_matrix[i] = embedding\_vector

In [72]:

length = []

for x in value\_list:

length.append(len(x.split()))

max(length)

Out[72]:

33575

In [73]:

x\_train\_seq = pad\_sequences(sequences, maxlen=33575)

In [74]:

x\_train\_seq.shape

Out[74]:

(9765, 33575)

CNN[¶](#CNN)

In [75]:

import tensorflow as tf

import tensorflow.keras as keras

from tensorflow.keras.preprocessing.text import Tokenizer

from tensorflow.keras.preprocessing.sequence import pad\_sequences

from tensorflow.keras.layers import Concatenate, Dense, Input, LSTM, Embedding, Dropout, Activation, GRU, Flatten

from tensorflow.keras.layers import Bidirectional, GlobalMaxPool1D

from tensorflow.keras.models import Model, Sequential

from tensorflow.keras.layers import Convolution1D

from tensorflow.keras import initializers, regularizers, constraints, optimizers, layers

In [128]:

maxlen = 33575

vocab\_size = 50000

embedding\_dim = 300

RNN\_CELL\_SIZE = 25

num\_filters = 50

kernel\_size = 2

input\_A = keras.layers.Input(shape=[81], name='wide\_input')

input\_B = keras.layers.Input(shape=(maxlen,), dtype="int32")

embedded\_sequences = keras.layers.Embedding(vocab\_size, embedding\_dim, weights=[embedding\_matrix], input\_length=maxlen)(input\_B)

convolution =keras.layers.Conv1D(num\_filters, kernel\_size, activation='relu')(embedded\_sequences)

maxpooling = keras.layers.GlobalMaxPooling1D()(convolution)

input\_layer = keras.layers.Concatenate()([input\_A, maxpooling])

dense1 = keras.layers.Dense(200, activation='relu')(input\_layer)

dense2 = keras.layers.Dense(50, activation='relu')(dense1)

output = keras.layers.Dense(1, activation='sigmoid')(dense2)

model\_DL = keras.Model(inputs=[input\_A, input\_B], outputs=output)

model\_DL.compile(optimizer='adam',

loss='binary\_crossentropy',

metrics=keras.metrics.AUC(name='auc'))

In [129]:

model\_DL.summary()

Model: "functional\_3"

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Layer (type) Output Shape Param # Connected to

==================================================================================================

input\_2 (InputLayer) [(None, 33575)] 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

embedding\_1 (Embedding) (None, 33575, 300) 15000000 input\_2[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

conv1d\_1 (Conv1D) (None, 33574, 50) 30050 embedding\_1[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

wide\_input (InputLayer) [(None, 81)] 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

global\_max\_pooling1d\_1 (GlobalM (None, 50) 0 conv1d\_1[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

concatenate\_1 (Concatenate) (None, 131) 0 wide\_input[0][0]

global\_max\_pooling1d\_1[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dense\_3 (Dense) (None, 200) 26400 concatenate\_1[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dense\_4 (Dense) (None, 50) 10050 dense\_3[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dense\_5 (Dense) (None, 1) 51 dense\_4[0][0]

==================================================================================================

Total params: 15,066,551

Trainable params: 15,066,551

Non-trainable params: 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

In [120]:

df\_merged\_final = df\_merged.drop(['Report Date', 'SimFinId', 'Currency', 'fy', 'Revenue', 'cik',

'Company Name', 'Income (Loss) from Continuing Operations',

'total.debt', 'ebit', 'ebitda', 'Income (Loss) from Continuing Operations',

'segment.mean', 'dso', 'dpo', 'dio'], axis=1)

In [121]:

#categorical features

from sklearn.preprocessing import OneHotEncoder

categorical\_features = ['sic', 'lifecycle']

df\_merged\_final[categorical\_features] = df\_merged\_final[categorical\_features].astype('str')

cat\_encoder = OneHotEncoder(sparse=False)

cat\_encoder = cat\_encoder.fit(df\_merged\_final[categorical\_features])

col\_names = cat\_encoder.get\_feature\_names(categorical\_features)

cat\_encoder = cat\_encoder.transform(df\_merged\_final[categorical\_features])

#Create a Pandas DataFrame of the hot encoded column

ohe\_df = pd.DataFrame(cat\_encoder, columns = col\_names)

#Concatenate with original data

df\_merged\_final = pd.concat([df\_merged\_final, ohe\_df], axis=1).drop(categorical\_features, axis=1)

In [110]:

df\_merged\_final = df\_merged\_final.merge(ohe\_df, left\_index=True, right\_index = True)

In [144]:

from sklearn.model\_selection import StratifiedShuffleSplit

#df\_merged\_final = df\_merged.drop('value', axis=1)

split = StratifiedShuffleSplit(n\_splits=1, test\_size=0.25, random\_state=17)

for train\_index, test\_index in split.split(df\_merged\_final, df\_merged\_final['return.above.average']):

train\_set\_A = df\_merged\_final.loc[train\_index]

train\_set\_B = x\_train\_seq[train\_index]

test\_set\_A = df\_merged\_final.loc[test\_index]

test\_set\_B = x\_train\_seq[test\_index]

y\_train = train\_set\_A['return.above.average']

y\_test = test\_set\_A['return.above.average']

train\_set\_A = train\_set\_A.drop(['return.above.average', 'value'], axis=1)

test\_set\_A = test\_set\_A.drop(['return.above.average', 'value'], axis=1)

In [156]:

len(df\_merged\_final)

Out[156]:

9765

In [130]:

history = model\_DL.fit((train\_set\_A, train\_set\_B), y\_train, batch\_size=64, epochs=5, validation\_data=((test\_set\_A, test\_set\_B), y\_test))

Epoch 1/5

115/115 [==============================] - 536s 5s/step - loss: 0.5229 - auc: 0.7999 - val\_loss: 0.3002 - val\_auc: 0.9252

Epoch 2/5

115/115 [==============================] - 529s 5s/step - loss: 0.3404 - auc: 0.9250 - val\_loss: 0.4082 - val\_auc: 0.9493

Epoch 3/5

115/115 [==============================] - 530s 5s/step - loss: 0.2500 - auc: 0.9657 - val\_loss: 0.2224 - val\_auc: 0.9591

Epoch 4/5

115/115 [==============================] - 529s 5s/step - loss: 0.1869 - auc: 0.9807 - val\_loss: 0.2220 - val\_auc: 0.9590

Epoch 5/5

115/115 [==============================] - 531s 5s/step - loss: 0.1400 - auc: 0.9909 - val\_loss: 0.2420 - val\_auc: 0.9633

In [185]:

from sklearn.model\_selection import StratifiedShuffleSplit

split = StratifiedShuffleSplit(n\_splits=1, test\_size=0.4, random\_state=17)

for train\_index, test\_valid\_index in split.split(df\_merged\_final, df\_merged\_final['return.above.average']):

train\_set = df\_merged\_final.iloc[train\_index]

test\_valid\_set = df\_merged\_final.iloc[test\_valid\_index]

train\_set\_B = x\_train\_seq[train\_index]

test\_valid\_set\_B = x\_train\_seq[test\_valid\_index]

split2 = StratifiedShuffleSplit(n\_splits=1, test\_size=0.5, random\_state=17)

for test\_index2, valid\_index in split2.split(test\_valid\_set, test\_valid\_set['return.above.average']):

test\_set = test\_valid\_set.iloc[test\_index2]

valid\_set = test\_valid\_set.iloc[valid\_index]

test\_set\_B = test\_valid\_set\_B[test\_index2]

valid\_set\_B = test\_valid\_set\_B[valid\_index]

y\_train = train\_set['return.above.average']

X\_train = train\_set.drop(['return.above.average', 'value'], axis=1)

y\_valid = valid\_set['return.above.average']

X\_valid = valid\_set.drop(['return.above.average', 'value'], axis=1)

y\_test = test\_set['return.above.average']

X\_test = test\_set.drop(['return.above.average', 'value'], axis=1)

In [175]:

maxlen = 33575

vocab\_size = 50000

embedding\_dim = 300

RNN\_CELL\_SIZE = 25

num\_filters = 50

kernel\_size = 2

input\_A = keras.layers.Input(shape=[81], name='wide\_input')

input\_B = keras.layers.Input(shape=(maxlen,), dtype="int32")

embedded\_sequences = keras.layers.Embedding(vocab\_size, embedding\_dim, weights=[embedding\_matrix], input\_length=maxlen)(input\_B)

convolution =keras.layers.Conv1D(num\_filters, kernel\_size, activation='relu')(embedded\_sequences)

maxpooling = keras.layers.GlobalMaxPooling1D()(convolution)

input\_layer = keras.layers.Concatenate()([input\_A, maxpooling])

dense1 = keras.layers.Dense(200, activation='relu')(input\_layer)

dense2 = keras.layers.Dense(50, activation='relu')(dense1)

output = keras.layers.Dense(1, activation='sigmoid')(dense2)

model\_DL2 = keras.Model(inputs=[input\_A, input\_B], outputs=output)

model\_DL2.compile(optimizer='adam',

loss='binary\_crossentropy',

metrics=keras.metrics.AUC(name='auc'))

model\_DL2.summary()

Model: "functional\_8"

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Layer (type) Output Shape Param # Connected to

==================================================================================================

input\_12 (InputLayer) [(None, 33575)] 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

embedding\_11 (Embedding) (None, 33575, 300) 15000000 input\_12[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

conv1d\_11 (Conv1D) (None, 33574, 50) 30050 embedding\_11[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

wide\_input (InputLayer) [(None, 81)] 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

global\_max\_pooling1d\_6 (GlobalM (None, 50) 0 conv1d\_11[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

concatenate\_12 (Concatenate) (None, 131) 0 wide\_input[0][0]

global\_max\_pooling1d\_6[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dense\_21 (Dense) (None, 200) 26400 concatenate\_12[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dense\_22 (Dense) (None, 50) 10050 dense\_21[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dense\_23 (Dense) (None, 1) 51 dense\_22[0][0]

==================================================================================================

Total params: 15,066,551

Trainable params: 15,066,551

Non-trainable params: 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

In [187]:

history2 = model\_DL2.fit((X\_train, train\_set\_B), y\_train, batch\_size=64, epochs=5, validation\_data=((X\_valid, valid\_set\_B), y\_valid))

Epoch 1/5

92/92 [==============================] - 428s 5s/step - loss: 0.4915 - auc: 0.7829 - val\_loss: 0.6392 - val\_auc: 0.9027

Epoch 2/5

92/92 [==============================] - 425s 5s/step - loss: 0.3817 - auc: 0.9072 - val\_loss: 0.3196 - val\_auc: 0.9354

Epoch 3/5

92/92 [==============================] - 422s 5s/step - loss: 0.2654 - auc: 0.9624 - val\_loss: 0.2560 - val\_auc: 0.9522

Epoch 4/5

92/92 [==============================] - 429s 5s/step - loss: 0.1957 - auc: 0.9776 - val\_loss: 0.2921 - val\_auc: 0.9443

Epoch 5/5

92/92 [==============================] - 423s 5s/step - loss: 0.1562 - auc: 0.9889 - val\_loss: 0.2488 - val\_auc: 0.9507

In [188]:

model\_DL2.evaluate((X\_test, test\_set\_B), y\_test)

62/62 [==============================] - 35s 566ms/step - loss: 0.2467 - auc: 0.9496

Out[188]:

[0.24670052528381348, 0.9496071934700012]

In [200]:

import matplotlib.pyplot as plt

pd.DataFrame(history2.history).plot(figsize=(8,5))

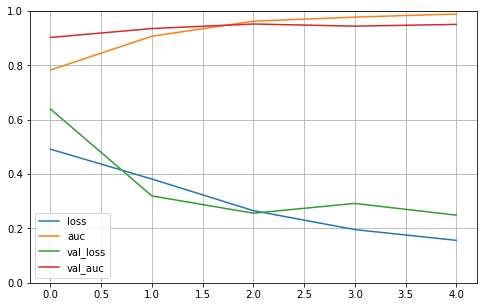
plt.grid(True)

plt.gca().set\_ylim(0,1)

plt.show

Out[200]:

<function matplotlib.pyplot.show(\*args, \*\*kw)>



# Appendix B

Our python codes are divided into two notebooks. Appendix B presents all models tested using lightGBM.

In [1]:

import pandas as pd

# Import the main functionality from the SimFin Python API.

import simfin as sf

# Import names used for easy access to SimFin's data-columns.

from simfin.names import \*

import numpy

In [2]:

# Version of the SimFin Python API.

sf.\_\_version\_\_

Out[2]:

'0.8.1'

In [3]:

sf.set\_data\_dir('~/simfin\_data/')

In [4]:

sf.set\_api\_key(api\_key='free')

In [5]:

df\_income = sf.load\_income(variant='annual', market='us')

Dataset "us-income-annual" on disk (17 days old).

- Loading from disk ... Done!

In [6]:

df\_balance = sf.load\_balance(variant='annual', market='us')

Dataset "us-balance-annual" on disk (17 days old).

- Loading from disk ... Done!

In [7]:

df\_cashflow = sf.load\_cashflow(variant='annual', market='us')

Dataset "us-cashflow-annual" on disk (17 days old).

- Loading from disk ... Done!

In [8]:

df\_companies = sf.load\_companies(index=TICKER, market='us')

Dataset "us-companies" on disk (17 days old).

- Loading from disk ... Done!

In [9]:

df\_industries = sf.load\_industries()

Dataset "industries" on disk (17 days old).

- Loading from disk ... Done!

In [10]:

df\_income\_reduced = df\_income[['SimFinId', 'Currency', 'Fiscal Year','Income (Loss) from Continuing Operations',

'Revenue','Cost of Revenue', 'Gross Profit','Selling, General & Administrative',

'Depreciation & Amortization', 'Non-Operating Income (Loss)']]

In [11]:

df\_balance\_reduced = df\_balance[['SimFinId', 'Currency', 'Fiscal Year','Total Assets','Total Equity',

'Short Term Debt','Long Term Debt', 'Cash, Cash Equivalents & Short Term Investments',

'Long Term Investments & Receivables','Accounts & Notes Receivable','Payables & Accruals',

'Inventories','Property, Plant & Equipment, Net']]

In [12]:

df\_cashflow\_reduced = df\_cashflow[['SimFinId', 'Currency', 'Fiscal Year','Net Cash from Operating Activities',

'Net Cash from Investing Activities', 'Net Cash from Financing Activities']]

Extracting the sign of each cashflow item to calculate life cycle[¶](#Extracting-the-sign-of-each-cashflow-it)

In [13]:

pd.set\_option('mode.chained\_assignment', None)

df\_cashflow\_reduced['Op\_sign'] = ["+" if company >= 0 else "-" for company in df\_cashflow\_reduced['Net Cash from Operating Activities']]

df\_cashflow\_reduced['Inv\_sign'] = ["+" if company >= 0 else "-" for company in df\_cashflow\_reduced['Net Cash from Investing Activities']]

df\_cashflow\_reduced['Fin\_sign'] = ["+" if company >= 0 else "-" for company in df\_cashflow\_reduced['Net Cash from Financing Activities']]

In [14]:

df\_cashflow\_reduced['lifecycle'] = None

for row in range(0,len(df\_cashflow\_reduced)):

if (df\_cashflow\_reduced['Op\_sign'][row] == '-') & (df\_cashflow\_reduced['Inv\_sign'][row] == '-') & (df\_cashflow\_reduced['Fin\_sign'][row]=="+"):

df\_cashflow\_reduced['lifecycle'][row] = 'Introduction'

elif (df\_cashflow\_reduced['Op\_sign'][row] == '+') & (df\_cashflow\_reduced['Inv\_sign'][row] == '-') & (df\_cashflow\_reduced['Fin\_sign'][row]=="+"):

df\_cashflow\_reduced['lifecycle'][row] = 'Growth'

elif (df\_cashflow\_reduced['Op\_sign'][row] == '+') & (df\_cashflow\_reduced['Inv\_sign'][row] == '-') & (df\_cashflow\_reduced['Fin\_sign'][row]=="-"):

df\_cashflow\_reduced['lifecycle'][row] = 'Mature'

elif (df\_cashflow\_reduced['Op\_sign'][row] == '-') & (df\_cashflow\_reduced['Inv\_sign'][row] == '-') & (df\_cashflow\_reduced['Fin\_sign'][row]=="-"):

df\_cashflow\_reduced['lifecycle'][row] = 'Shake-out'

elif (df\_cashflow\_reduced['Op\_sign'][row] == '+') & (df\_cashflow\_reduced['Inv\_sign'][row] == '+') & (df\_cashflow\_reduced['Fin\_sign'][row]=="+"):

df\_cashflow\_reduced['lifecycle'][row] = 'Shake-out'

elif (df\_cashflow\_reduced['Op\_sign'][row] == '+') & (df\_cashflow\_reduced['Inv\_sign'][row] == '+') & (df\_cashflow\_reduced['Fin\_sign'][row]=="-"):

df\_cashflow\_reduced['lifecycle'][row] = 'Shake-out'

else:

df\_cashflow\_reduced['lifecycle'][row] = 'Decline'

In [15]:

df\_cashflow\_reduced.groupby(['lifecycle'])['lifecycle'].count()

Out[15]:

lifecycle

Decline 773

Growth 4240

Introduction 1681

Mature 8404

Shake-out 1395

Name: lifecycle, dtype: int64

In [16]:

df\_cashflow\_lifecycle = df\_cashflow\_reduced.drop(['Net Cash from Operating Activities','Net Cash from Investing Activities',

'Net Cash from Financing Activities', 'Op\_sign', 'Inv\_sign', 'Fin\_sign'], axis=1)

In [17]:

df\_financial = df\_income\_reduced.merge(df\_balance\_reduced, on=['Report Date', 'SimFinId', 'Currency', 'Fiscal Year']).merge(df\_cashflow\_lifecycle, on=['Report Date', 'SimFinId', 'Currency', 'Fiscal Year'])

In [18]:

df\_financial.reset\_index(level=0, inplace=True)

In [19]:

df\_financial.isnull().sum()/len(df\_financial)

Out[19]:

Report Date 0.000000

SimFinId 0.000000

Currency 0.000000

Fiscal Year 0.000000

Income (Loss) from Continuing Operations 0.000000

Revenue 0.015886

Cost of Revenue 0.093919

Gross Profit 0.015219

Selling, General & Administrative 0.049961

Depreciation & Amortization 0.563027

Non-Operating Income (Loss) 0.011156

Total Assets 0.000000

Total Equity 0.000121

Short Term Debt 0.337477

Long Term Debt 0.219548

Cash, Cash Equivalents & Short Term Investments 0.002971

Long Term Investments & Receivables 0.711999

Accounts & Notes Receivable 0.084642

Payables & Accruals 0.004669

Inventories 0.306736

Property, Plant & Equipment, Net 0.021100

lifecycle 0.000000

dtype: float64

Dropping companies that didn't report revenues or costs. Problably holdings that don't have operation.

In [20]:

len(df\_financial)

Out[20]:

16493

In [21]:

drop\_row\_null = list(df\_financial.index[df\_financial['Revenue'].isnull() == True]) # 262 rows

df\_financial = df\_financial.drop(drop\_row\_null)

drop\_row\_zero = list(df\_financial.index[df\_financial['Revenue']==0]) #135 rows

df\_financial = df\_financial.drop(drop\_row\_zero)

drop\_row\_null\_cost = list(df\_financial.index[df\_financial['Cost of Revenue'].isnull() == True]) # 262 rows

df\_financial = df\_financial.drop(drop\_row\_null\_cost)

drop\_row\_zero\_cost = list(df\_financial.index[df\_financial['Cost of Revenue']==0]) #32 rows

df\_financial = df\_financial.drop(drop\_row\_zero\_cost)

In [22]:

len(df\_financial)

Out[22]:

14872

Some companies doesn't report values for some of the lines because they don't have it. So it was replaced by zero

In [23]:

df\_financial = df\_financial.fillna(0)

In [24]:

#Some of the items already carries negative values (e.g. Cost of Revenue and Selling, General & Administrative)

#All formulas were adapted

df\_financial['total.debt'] = df\_financial['Short Term Debt'] + df\_financial['Long Term Debt']

df\_financial['size'] = numpy.log(1+df\_financial['Total Assets'])

df\_financial['leverage'] = df\_financial['total.debt']/df\_financial['Total Assets']

df\_financial['ebit'] = df\_financial['Revenue'] + df\_financial['Selling, General & Administrative']

df\_financial['ebitda'] = df\_financial['ebit'] - df\_financial['Depreciation & Amortization']

df\_financial['roa'] = df\_financial['Income (Loss) from Continuing Operations']/df\_financial['Total Assets']

df\_financial['roe'] = df\_financial['Income (Loss) from Continuing Operations']/df\_financial['Total Equity']

df\_financial['roic'] = df\_financial['Income (Loss) from Continuing Operations']/(df\_financial['Total Assets'] - df\_financial['Cash, Cash Equivalents & Short Term Investments'] - df\_financial['Long Term Investments & Receivables'])

df\_financial['gross.profit.margin'] = df\_financial['Gross Profit']/df\_financial['Revenue']

df\_financial['ebit.margin'] = df\_financial['ebit']/df\_financial['Revenue']

df\_financial['net.profit.margin'] = df\_financial['Income (Loss) from Continuing Operations']/df\_financial['Revenue']

df\_financial['asset.turnover'] = df\_financial['Revenue']/df\_financial['Property, Plant & Equipment, Net']

df\_financial['financial.leverage'] = df\_financial['Income (Loss) from Continuing Operations']\*df\_financial['Total Assets']/df\_financial['Total Equity']/(df\_financial['Income (Loss) from Continuing Operations']-df\_financial['Non-Operating Income (Loss)'])

df\_financial['operational.leverage'] = df\_financial['Gross Profit']/(df\_financial['Gross Profit']+df\_financial['Selling, General & Administrative'])

df\_financial['dso'] = (df\_financial['Accounts & Notes Receivable']\*360)/df\_financial['Revenue']

df\_financial['dpo'] = (df\_financial['Accounts & Notes Receivable']\*360)/df\_financial['Revenue']

df\_financial['dio'] = (df\_financial['Inventories']\*360)/df\_financial['Cost of Revenue']\*-1

In [25]:

df\_financial = df\_financial[['Report Date','SimFinId', 'Currency', 'Fiscal Year', 'Revenue', 'Income (Loss) from Continuing Operations',

'total.debt', 'size', 'leverage', 'ebit', 'ebitda', 'roa', 'roe', 'roic',

'gross.profit.margin', 'ebit.margin', 'net.profit.margin',

'asset.turnover', 'financial.leverage', 'operational.leverage', 'dso',

'dpo', 'dio', 'lifecycle']]

In [26]:

df\_financial.isnull().sum()/len(df\_financial)

Out[26]:

Report Date 0.0

SimFinId 0.0

Currency 0.0

Fiscal Year 0.0

Revenue 0.0

Income (Loss) from Continuing Operations 0.0

total.debt 0.0

size 0.0

leverage 0.0

ebit 0.0

ebitda 0.0

roa 0.0

roe 0.0

roic 0.0

gross.profit.margin 0.0

ebit.margin 0.0

net.profit.margin 0.0

asset.turnover 0.0

financial.leverage 0.0

operational.leverage 0.0

dso 0.0

dpo 0.0

dio 0.0

lifecycle 0.0

dtype: float64

In [27]:

df\_financial = df\_financial.merge(df\_companies, on ='SimFinId', how ='left').drop(['IndustryId'],axis=1)

Import table with companies and segments from the textual data set[¶](#Import-table-with-companies-and-segment)

In [28]:

companies\_txt = pd.read\_csv(r'C:\Users\alyss\Desktop\MS BAIS\Fall 2020\ISM6930 - Text Analytics\Team Project\Financial Statements NLP\Final Project\companies\_sec.csv')

companies\_txt = companies\_txt.drop('Unnamed: 0', axis=1)

In [29]:

import math

companies\_txt['sic'] = companies\_txt['sic']/100

companies\_txt['sic'] = numpy.floor(companies\_txt['sic'])

Removing the punctuation from the names helped to reduce the number of missing data from 145 to 108

In [30]:

companies\_txt['name'] = companies\_txt['name'].str.replace(',',"")

companies\_txt['name'] = companies\_txt['name'].str.replace('.',"")

companies\_txt = companies\_txt.rename(columns={"name": "Company Name"})

companies\_txt = companies\_txt.drop\_duplicates(ignore\_index = True)

companies\_txt = companies\_txt.reset\_index(drop=True)

In [31]:

df\_financial['Company Name'] = df\_financial['Company Name'].str.upper()

df\_financial['Company Name'] = df\_financial['Company Name'].str.replace(',',"")

df\_financial['Company Name'] = df\_financial['Company Name'].str.replace('.',"")

In [32]:

df\_financial = df\_financial.merge(companies\_txt, on='Company Name', how='left')

In [33]:

df\_financial['sic'].isnull().sum()/len(df\_financial['sic'])

Out[33]:

0.06224256292906179

dropping the few companies that we couldn't find the segment

In [34]:

df\_financial = df\_financial.drop(list(df\_financial.index[df\_financial['sic'].isnull()==True]))

In [35]:

df\_financial = df\_financial.reset\_index(drop=True)

Calculating the dependent variable[¶](#Calculating-the-dependent-variable)

In [36]:

segment\_mean = pd.DataFrame(df\_financial.groupby(['sic'])['Income (Loss) from Continuing Operations'].agg(numpy.mean))

segment\_mean = segment\_mean.rename(columns={'Income (Loss) from Continuing Operations':'segment.mean'})

In [37]:

df\_financial = df\_financial.merge(segment\_mean, on='sic', how = 'left')

In [38]:

df\_financial['return.above.average'] = [1 if df\_financial['Income (Loss) from Continuing Operations'][row] >= df\_financial['segment.mean'][row] else 0 for row in range(0, len(df\_financial)) ]

In [39]:

df\_financial = df\_financial.drop\_duplicates().reset\_index(drop=True)

In [40]:

df\_financial['return.above.average'] = df\_financial.groupby(['SimFinId'])['return.above.average'].shift(-1)

In [41]:

df\_financial = df\_financial.dropna().reset\_index(drop=True)

In [42]:

df\_financial['return.above.average'] = df\_financial['return.above.average'].astype('int')

In [43]:

companies = df\_financial['cik'].drop\_duplicates()

In [44]:

df\_financial\_final = df\_financial.drop(['Report Date', 'SimFinId', 'Currency', 'Fiscal Year', 'Revenue', 'cik',

'Company Name', 'Income (Loss) from Continuing Operations',

'total.debt', 'ebit', 'ebitda', 'Income (Loss) from Continuing Operations',

'segment.mean'], axis=1)

Baseline model - without text features

In [45]:

from sklearn.preprocessing import StandardScaler

import numpy as np

pd.set\_option('use\_inf\_as\_na', True)

In [46]:

df\_financial\_final = df\_financial\_final.dropna().reset\_index(drop=True)

In [47]:

numerical\_features = [col for col in df\_financial\_final.columns if col not in ['sic', 'lifecycle','return.above.average']]

In [48]:

# Standardizing numerical features

scaler = StandardScaler()

scaler = scaler.fit(df\_financial\_final[numerical\_features])

df\_financial\_final[numerical\_features]=scaler.transform(df\_financial\_final[numerical\_features])

In [49]:

#categorical features

from sklearn.preprocessing import OneHotEncoder

categorical\_features = ['sic', 'lifecycle']

df\_financial\_final[categorical\_features] = df\_financial\_final[categorical\_features].astype('str')

cat\_encoder = OneHotEncoder(sparse=False)

cat\_encoder = cat\_encoder.fit(df\_financial\_final[categorical\_features])

col\_names = cat\_encoder.get\_feature\_names(categorical\_features)

cat\_encoder = cat\_encoder.transform(df\_financial\_final[categorical\_features])

#Create a Pandas DataFrame of the hot encoded column

ohe\_df = pd.DataFrame(cat\_encoder, columns = col\_names)

#Concatenate with original data

df\_financial\_final = pd.concat([df\_financial\_final, ohe\_df], axis=1).drop(categorical\_features, axis=1)

In [50]:

df\_fin\_target = df\_financial\_final['return.above.average']

df\_fin\_X = df\_financial\_final.drop('return.above.average',axis=1)

In [51]:

from sklearn.model\_selection import train\_test\_split

train\_x, test\_x, train\_y, test\_y = train\_test\_split(df\_fin\_X, df\_fin\_target, test\_size=0.25, random\_state=17)

In [52]:

from sklearn.ensemble import RandomForestClassifier

rfc = RandomForestClassifier(random\_state=1)

rfc.fit(train\_x, train\_y)

Out[52]:

RandomForestClassifier(random\_state=1)

In [53]:

# view accuracy on test set

from sklearn.metrics import accuracy\_score, roc\_auc\_score

y\_pred\_rfc=rfc.predict(test\_x)

print("acc:",accuracy\_score(test\_y,y\_pred\_rfc))

print("auc:",roc\_auc\_score(test\_y,y\_pred\_rfc))

acc: 0.9053140096618357

auc: 0.8522015447302804

In [54]:

# build the lightgbm model

import lightgbm as lgb

clf = lgb.LGBMClassifier(seed = 1)

clf.fit(train\_x, train\_y)

Out[54]:

LGBMClassifier(seed=1)

In [55]:

# view accuracy on test set

y\_pred=clf.predict(test\_x)

print("acc:",accuracy\_score(test\_y,y\_pred))

print("auc:",roc\_auc\_score(test\_y,y\_pred))

acc: 0.9162640901771336

auc: 0.8845557380040139

In [56]:

from sklearn.neural\_network import MLPClassifier

model = MLPClassifier(solver='adam', alpha=1e-5, learning\_rate='adaptive',

early\_stopping=True, activation = 'relu', hidden\_layer\_sizes=(512, 128, 12),

random\_state=1)

model.fit(train\_x, train\_y)

Out[56]:

MLPClassifier(alpha=1e-05, early\_stopping=True,

hidden\_layer\_sizes=(512, 128, 12), learning\_rate='adaptive',

random\_state=1)

In [57]:

# view accuracy on test set

y\_pred\_nn=model.predict(test\_x)

print("acc:",accuracy\_score(test\_y,y\_pred\_nn))

print("auc:",roc\_auc\_score(test\_y,y\_pred\_nn))

acc: 0.900805152979066

auc: 0.869853433071824

Importing textual data[¶](#Importing-textual-data)

In [58]:

import os

import pickle

from sklearn.feature\_extraction.text import TfidfVectorizer

from nltk.stem import WordNetLemmatizer

from nltk import word\_tokenize

In [59]:

#import txts files

with open('filename.pickle', 'rb') as handle:

txts = pickle.load(handle)

txts = txts.drop('Unnamed: 0', axis=1)

with open('subs.pickle', 'rb') as handle\_subs:

subs = pickle.load(handle\_subs)

In [60]:

#filter only service companies from subs file

df\_txt = subs[(subs['fp']=='FY') & (subs['fy']>2011) & (subs['fy']<2020) & subs['cik'].isin(companies)]

In [61]:

df\_txt\_selected = subs[['adsh', 'cik', 'fy']][subs['cik'].isin(companies)]

In [62]:

#get adsh for the selected companies

selected\_adsh = subs['adsh'][subs['cik'].isin(companies)]

# Check the proportion of companies - 23%

len(selected\_adsh)/len(subs)

Out[62]:

0.2300242559444769

In [63]:

#filter selected companies from txts file

txt\_selected = txts[txts['adsh'].isin(selected\_adsh)]

df\_txt = df\_txt\_selected.merge(txt\_selected, on='adsh')

In [64]:

len(df\_txt)

Out[64]:

884760

Merging text blocks[¶](#Merging-text-blocks)

In [65]:

df\_txt['value'] = df\_txt['value'].astype('str')

In [66]:

values = df\_txt.groupby(['adsh','fy'])['value'].agg(' '.join)

In [67]:

values = pd.DataFrame(values)

In [68]:

values.reset\_index(inplace=True)

In [69]:

values = values.merge(df\_txt\_selected, on=['adsh', 'fy'], how='left')

Preprocessing texts[¶](#Preprocessing-texts)

In [70]:

#including some non-printable character in the stopwords

from gensim.parsing.preprocessing import STOPWORDS

my\_stop\_words = STOPWORDS.union(set(['A&amp;B', '&amp', '\0', '\1', '\2', '\3', '\4', '\5', '\6', '\a', '\b', '\t', '\n', '\v', '\f', '\r',

'\x0e', '\x0f', '\x10', '\x11', '\x12', '\x13', '\x14', '\x15', '\x16', '\x17', '\x18', '\x19', '\x1a',

'\x1b', '\x1c', '\x1d', '\x1e', '\x1f', '', 'nan', '\_', '\_\_', '\_\_\_', '\_\_\_\_', '\_\_\_\_\_', '\_\_\_\_\_\_', '\_\_\_\_\_\_\_', '\_\_\_\_\_\_\_\_',

'\_\_\_\_\_\_\_\_\_', '\_\_\_\_\_\_\_\_\_\_', '\_\_\_\_\_\_\_\_\_\_\_', '\_\_\_\_\_\_\_\_\_\_\_\_', '\_\_\_\_\_\_\_\_\_\_\_\_\_', '\_\_\_\_\_\_\_\_\_\_\_\_\_\_', '\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_',

'\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_', '\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_', '\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_']))

In [71]:

# tranform texts into a list to iterate faster

value\_list = list(values['value'].astype(str))

In [72]:

from gensim.utils import tokenize

tokens = [list(tokenize(doc, lower=True, deacc=True, encoding='utf8', errors='strict')) for doc in value\_list]

tokens\_str = [' '.join(row) for row in tokens]

In [73]:

from gensim.parsing.preprocessing import remove\_stopwords

tokens\_str = [remove\_stopwords(h) for h in tokens\_str]

In [74]:

from nltk.stem import WordNetLemmatizer

lemmatizer = WordNetLemmatizer()

lemma = [lemmatizer.lemmatize(h) for h in tokens\_str]

tf-idf vector[¶](#tf-idf-vector)

In [75]:

tf\_vectorizer = TfidfVectorizer(max\_df = 0.7,

min\_df = 0.1)

In [76]:

tf\_vectorizer.fit(lemma)

Out[76]:

TfidfVectorizer(max\_df=0.7, min\_df=0.1)

In [77]:

tfidf = tf\_vectorizer.transform(lemma)

In [78]:

import scipy.sparse

df\_tfidf = pd.DataFrame.sparse.from\_spmatrix(tfidf)

In [79]:

df\_tfidf = values.merge(df\_tfidf, left\_index = True, right\_index = True, how='left')

In [80]:

df\_tfidf.drop(['adsh','value'], axis=1, inplace=True)

In [81]:

df\_financial = df\_financial.rename(columns={'Fiscal Year':'fy'})

In [82]:

df\_fin\_tfidf = df\_financial.merge(df\_tfidf, on=['cik', 'fy'], how='left')

In [83]:

df\_fin\_tfidf = df\_fin\_tfidf.dropna()

In [84]:

len(df\_fin\_tfidf)

Out[84]:

9766

In [85]:

df\_fin\_tfidf = df\_fin\_tfidf.reset\_index(drop=True)

In [86]:

df\_fin\_tfidf\_final = df\_fin\_tfidf.drop(['Report Date', 'SimFinId', 'Currency', 'fy', 'Revenue', 'cik',

'Company Name', 'Income (Loss) from Continuing Operations',

'total.debt', 'ebit', 'ebitda', 'Income (Loss) from Continuing Operations',

'segment.mean'], axis=1)

In [87]:

#categorical features

from sklearn.preprocessing import OneHotEncoder

categorical\_features = ['sic', 'lifecycle']

df\_fin\_tfidf\_final[categorical\_features] = df\_fin\_tfidf\_final[categorical\_features].astype('str')

cat\_encoder = OneHotEncoder(sparse=False)

cat\_encoder = cat\_encoder.fit(df\_fin\_tfidf\_final[categorical\_features])

col\_names = cat\_encoder.get\_feature\_names(categorical\_features)

cat\_encoder = cat\_encoder.transform(df\_fin\_tfidf\_final[categorical\_features])

#Create a Pandas DataFrame of the hot encoded column

ohe\_df = pd.DataFrame(cat\_encoder, columns = col\_names)

#Concatenate with original data

df\_fin\_tfidf\_final = pd.concat([df\_fin\_tfidf\_final, ohe\_df], axis=1).drop(categorical\_features, axis=1)

In [88]:

df\_fin\_tfidf\_target = df\_fin\_tfidf\_final['return.above.average']

df\_fin\_tfidf\_X = df\_fin\_tfidf\_final.drop('return.above.average',axis=1)

In [89]:

from sklearn.model\_selection import train\_test\_split

train\_x, test\_x, train\_y, test\_y = train\_test\_split(df\_fin\_tfidf\_X, df\_fin\_tfidf\_target, test\_size=0.25, random\_state=17)

In [90]:

# build the lightgbm model

clf2 = lgb.LGBMClassifier(seed = 1)

clf2.fit(train\_x, train\_y)

Out[90]:

LGBMClassifier(seed=1)

In [91]:

# view accuracy on train set

from sklearn.metrics import accuracy\_score, roc\_auc\_score

y\_pred=clf2.predict(train\_x)

print("acc:",accuracy\_score(train\_y,y\_pred))

print("auc:",roc\_auc\_score(train\_y,y\_pred))

acc: 0.9993173129437466

auc: 0.9991486034598757

In [92]:

# view accuracy on test set

y\_pred=clf2.predict(test\_x)

print("acc:",accuracy\_score(test\_y,y\_pred))

print("auc:",roc\_auc\_score(test\_y,y\_pred))

acc: 0.9246519246519247

auc: 0.8860457566814429

In [93]:

from sklearn.metrics import confusion\_matrix, classification\_report

print(classification\_report(test\_y, y\_pred))

precision recall f1-score support

0 0.94 0.96 0.95 1861

1 0.86 0.81 0.84 581

accuracy 0.92 2442

macro avg 0.90 0.89 0.89 2442

weighted avg 0.92 0.92 0.92 2442

In [94]:

print(confusion\_matrix(test\_y, y\_pred))

[[1786 75]

[ 109 472]]

tf-idf only with nouns[¶](#tf-idf-only-with-nouns)

In [99]:

from gensim.utils import lemmatize

In [ ]:

lemma = []

for h in tokens\_str:

row\_list= []

for wd in lemmatize(h):

if wd.decode('utf-8').split('/')[1] == 'NN':

row\_list.append(wd.decode('utf-8').split('/')[0])

row\_list = ' '.join(row\_list)

lemma.append(row\_list)

In [ ]:

# Store data (serialize)

#with open('lemma\_pos.pickle', 'wb') as handle2:

# pickle.dump(lemma, handle2, protocol=pickle.HIGHEST\_PROTOCOL)

In [ ]:

#with open('lemma\_pos.pickle', 'rb') as handle:

# lemma = pickle.load(handle)

In [ ]:

tf\_vectorizer = TfidfVectorizer(max\_df = 0.7,

min\_df = 0.1)

In [ ]:

tf\_vectorizer.fit(lemma)

In [ ]:

tfidf = tf\_vectorizer.transform(lemma)

In [ ]:

import scipy.sparse

df\_tfidf = pd.DataFrame.sparse.from\_spmatrix(tfidf)

In [103]:

# Store data (serialize)

with open('df\_tfidf\_lemma\_pos.pickle', 'wb') as handle4:

pickle.dump(df\_tfidf, handle4, protocol=pickle.HIGHEST\_PROTOCOL)

In [104]:

with open('df\_tfidf\_lemma\_pos.pickle', 'rb') as handle:

df\_tfidf = pickle.load(handle)

In [105]:

df\_financial = df\_financial.rename(columns={'Fiscal Year':'fy'})

In [106]:

df\_fin\_tfidf = df\_financial.merge(df\_tfidf, on=['cik', 'fy'], how='left')

In [107]:

df\_fin\_tfidf = df\_fin\_tfidf.dropna()

In [108]:

len(df\_fin\_tfidf)

Out[108]:

9766

In [109]:

df\_fin\_tfidf = df\_fin\_tfidf.reset\_index(drop=True)

In [110]:

df\_fin\_tfidf\_final = df\_fin\_tfidf.drop(['Report Date', 'SimFinId', 'Currency', 'fy', 'Revenue', 'cik',

'Company Name', 'Income (Loss) from Continuing Operations',

'total.debt', 'ebit', 'ebitda', 'Income (Loss) from Continuing Operations',

'segment.mean'], axis=1)

In [111]:

numerical\_features = [col for col in df\_fin\_tfidf\_final.columns if col not in ['sic', 'lifecycle','return.above.average']]

In [112]:

# Standardizing numerical features

scaler = StandardScaler()

scaler = scaler.fit(df\_fin\_tfidf\_final[numerical\_features])

df\_fin\_tfidf\_final[numerical\_features]=scaler.transform(df\_fin\_tfidf\_final[numerical\_features])

C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py:509: UserWarning: pandas.DataFrame with sparse columns found.It will be converted to a dense numpy array.

warnings.warn(

C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py:509: UserWarning: pandas.DataFrame with sparse columns found.It will be converted to a dense numpy array.

warnings.warn(

In [113]:

#categorical features

from sklearn.preprocessing import OneHotEncoder

categorical\_features = ['sic', 'lifecycle']

df\_fin\_tfidf\_final[categorical\_features] = df\_fin\_tfidf\_final[categorical\_features].astype('str')

cat\_encoder = OneHotEncoder(sparse=False)

cat\_encoder = cat\_encoder.fit(df\_fin\_tfidf\_final[categorical\_features])

col\_names = cat\_encoder.get\_feature\_names(categorical\_features)

cat\_encoder = cat\_encoder.transform(df\_fin\_tfidf\_final[categorical\_features])

#Create a Pandas DataFrame of the hot encoded column

ohe\_df = pd.DataFrame(cat\_encoder, columns = col\_names)

#Concatenate with original data

df\_fin\_tfidf\_final = pd.concat([df\_fin\_tfidf\_final, ohe\_df], axis=1).drop(categorical\_features, axis=1)

In [114]:

df\_fin\_tfidf\_target = df\_fin\_tfidf\_final['return.above.average']

df\_fin\_tfidf\_X = df\_fin\_tfidf\_final.drop('return.above.average',axis=1)

In [115]:

from sklearn.model\_selection import train\_test\_split

train\_x\_tn, test\_x\_tn, train\_y\_tn, test\_y\_tn = train\_test\_split(df\_fin\_tfidf\_X, df\_fin\_tfidf\_target, test\_size=0.25, random\_state=17)

In [116]:

from sklearn.ensemble import RandomForestClassifier

rfc2 = RandomForestClassifier(random\_state=1)

rfc2.fit(train\_x\_tn, train\_y\_tn)

Out[116]:

RandomForestClassifier(random\_state=1)

In [117]:

# view accuracy on test set

from sklearn.metrics import accuracy\_score, roc\_auc\_score

y\_pred\_tn=rfc2.predict(test\_x\_tn)

print("acc:",accuracy\_score(test\_y\_tn,y\_pred\_tn))

print("auc:",roc\_auc\_score(test\_y\_tn,y\_pred\_tn))

acc: 0.9021294021294022

auc: 0.8286510592920541

In [118]:

from sklearn.metrics import confusion\_matrix, classification\_report

print(classification\_report(test\_y, y\_pred))

precision recall f1-score support

0 0.94 0.96 0.95 1861

1 0.86 0.81 0.84 581

accuracy 0.92 2442

macro avg 0.90 0.89 0.89 2442

weighted avg 0.92 0.92 0.92 2442

In [119]:

from sklearn.neural\_network import MLPClassifier

model = MLPClassifier(solver='adam', alpha=1e-5, learning\_rate='adaptive',

early\_stopping=True, activation = 'relu', hidden\_layer\_sizes=(512, 128, 12),

random\_state=1)

model.fit(train\_x\_tn, train\_y\_tn)

Out[119]:

MLPClassifier(alpha=1e-05, early\_stopping=True,

hidden\_layer\_sizes=(512, 128, 12), learning\_rate='adaptive',

random\_state=1)

In [120]:

# view accuracy on test set

y\_pred\_tn=model.predict(test\_x\_tn)

print("acc:",accuracy\_score(test\_y\_tn,y\_pred\_tn))

print("auc:",roc\_auc\_score(test\_y\_tn,y\_pred\_tn))

acc: 0.9144144144144144

auc: 0.8745936382360638

word to vector[¶](#word-to-vector)

In [121]:

from gensim.models import Word2Vec

import numpy as np

def average\_word\_vectors(words, model, vocabulary, num\_features):

feature\_vector = np.zeros((num\_features,), dtype="float64")

nwords = 0.

for word in words:

if word in vocabulary:

nwords = nwords + 1.

feature\_vector = np.add(feature\_vector, model.wv[word])

if nwords:

feature\_vector = np.divide(feature\_vector, nwords)

return feature\_vector

In [122]:

from sklearn.base import BaseEstimator, TransformerMixin

class word\_to\_vector(BaseEstimator, TransformerMixin):

'''This lemmatize words. It also, by default, considers nouns, verbs, adjectives and adverbs only'''

def fit(self, X, y=None):

return self

def transform(self, X):

train\_corpus = X

num\_features = 100

# sg = 1 for skip-gram

model = Word2Vec(train\_corpus, size=num\_features, window=5, min\_count=2, sample=1e-3, sg=1, iter=5)

vocab = set(model.wv.index2word)

train\_features = [average\_word\_vectors(sent\_tokens, model, vocab, num\_features) for sent\_tokens in train\_corpus]

avg\_train\_features = np.array(train\_features)

return avg\_train\_features

In [123]:

from gensim.utils import simple\_preprocess

from sklearn.pipeline import Pipeline

pipe\_w2vec = Pipeline([

('wtv', word\_to\_vector())

])

In [124]:

pipe\_w2vec.fit(tokens\_str)

Out[124]:

Pipeline(steps=[('wtv', word\_to\_vector())])

In [130]:

X\_w2vec = pipe\_w2vec.transform(tokens\_str)

In [131]:

## Store data (serialize)

with open('X\_w2vec.pickle', 'wb') as handle3:

pickle.dump(X\_w2vec, handle3, protocol=pickle.HIGHEST\_PROTOCOL)

In [132]:

X\_w2vec\_df = pd.DataFrame(X\_w2vec)

In [133]:

df\_w2vec = values.merge(X\_w2vec\_df, left\_index = True, right\_index = True, how='left')

In [134]:

df\_w2vec.drop(['adsh','value'], axis=1, inplace=True)

In [135]:

df\_fin\_df\_w2vec = df\_financial.merge(df\_w2vec, on=['cik', 'fy'], how='left')

In [136]:

df\_fin\_df\_w2vec = df\_fin\_df\_w2vec.dropna()

In [137]:

len(df\_fin\_df\_w2vec)

Out[137]:

9766

In [138]:

df\_fin\_df\_w2vec = df\_fin\_df\_w2vec.reset\_index(drop=True)

In [139]:

df\_fin\_df\_w2vec\_final = df\_fin\_df\_w2vec.drop(['Report Date', 'SimFinId', 'Currency', 'fy', 'Revenue', 'cik',

'Company Name', 'Income (Loss) from Continuing Operations',

'total.debt', 'ebit', 'ebitda', 'Income (Loss) from Continuing Operations',

'segment.mean'], axis=1)

In [140]:

#categorical features

from sklearn.preprocessing import OneHotEncoder

categorical\_features = ['sic', 'lifecycle']

df\_fin\_df\_w2vec\_final[categorical\_features] = df\_fin\_df\_w2vec\_final[categorical\_features].astype('str')

cat\_encoder = OneHotEncoder(sparse=False)

cat\_encoder = cat\_encoder.fit(df\_fin\_df\_w2vec\_final[categorical\_features])

col\_names = cat\_encoder.get\_feature\_names(categorical\_features)

cat\_encoder = cat\_encoder.transform(df\_fin\_df\_w2vec\_final[categorical\_features])

#Create a Pandas DataFrame of the hot encoded column

ohe\_df = pd.DataFrame(cat\_encoder, columns = col\_names)

#Concatenate with original data

df\_fin\_df\_w2vec\_final = pd.concat([df\_fin\_df\_w2vec\_final, ohe\_df], axis=1).drop(categorical\_features, axis=1)

In [141]:

df\_fin\_w2vec\_target = df\_fin\_df\_w2vec\_final['return.above.average']

df\_fin\_w2vec\_X = df\_fin\_df\_w2vec\_final.drop('return.above.average',axis=1)

In [142]:

from sklearn.model\_selection import train\_test\_split

train\_x, test\_x, train\_y, test\_y = train\_test\_split(df\_fin\_w2vec\_X, df\_fin\_w2vec\_target, test\_size=0.25, random\_state=17)

In [143]:

# build the lightgbm model

clf3 = lgb.LGBMClassifier(seed = 1)

clf3.fit(train\_x, train\_y)

Out[143]:

LGBMClassifier(seed=1)

In [144]:

# view accuracy on train set

from sklearn.metrics import accuracy\_score, roc\_auc\_score

y\_pred=clf3.predict(train\_x)

print("acc:",accuracy\_score(train\_y,y\_pred))

print("auc:",roc\_auc\_score(train\_y,y\_pred))

acc: 0.9959038776624796

auc: 0.9944856982034491

In [145]:

# view accuracy on test set

y\_pred=clf3.predict(test\_x)

print("acc:",accuracy\_score(test\_y,y\_pred))

print("auc:",roc\_auc\_score(test\_y,y\_pred))

acc: 0.9185094185094185

auc: 0.8814237528913536

In [146]:

from sklearn.metrics import confusion\_matrix, classification\_report

print(classification\_report(test\_y, y\_pred))

precision recall f1-score support

0 0.94 0.95 0.95 1861

1 0.84 0.81 0.83 581

accuracy 0.92 2442

macro avg 0.89 0.88 0.89 2442

weighted avg 0.92 0.92 0.92 2442

LSA[¶](#LSA)

In [147]:

from sklearn.decomposition import NMF

model = NMF(n\_components=100)

In [148]:

model.fit(tfidf)

Out[148]:

NMF(n\_components=100)

In [149]:

lsa = model.transform(tfidf)

In [150]:

lsa = pd.DataFrame(lsa)

In [151]:

df\_lsa = values.merge(lsa, left\_index = True, right\_index = True, how='left')

In [152]:

df\_lsa.drop(['adsh','value'], axis=1, inplace=True)

In [153]:

df\_financial = df\_financial.rename(columns={'Fiscal Year':'fy'})

In [154]:

df\_fin\_lsa = df\_financial.merge(df\_tfidf, on=['cik', 'fy'], how='left')

In [155]:

df\_fin\_lsa = df\_fin\_lsa.dropna()

In [156]:

len(df\_fin\_lsa)

Out[156]:

9766

In [157]:

df\_fin\_lsa = df\_fin\_lsa.reset\_index(drop=True)

In [158]:

df\_fin\_lsa = df\_fin\_lsa.drop(['Report Date', 'SimFinId', 'Currency', 'fy', 'Revenue', 'cik',

'Company Name', 'Income (Loss) from Continuing Operations',

'total.debt', 'ebit', 'ebitda', 'Income (Loss) from Continuing Operations',

'segment.mean'], axis=1)

In [159]:

#categorical features

from sklearn.preprocessing import OneHotEncoder

categorical\_features = ['sic', 'lifecycle']

df\_fin\_lsa[categorical\_features] = df\_fin\_lsa[categorical\_features].astype('str')

cat\_encoder = OneHotEncoder(sparse=False)

cat\_encoder = cat\_encoder.fit(df\_fin\_lsa[categorical\_features])

col\_names = cat\_encoder.get\_feature\_names(categorical\_features)

cat\_encoder = cat\_encoder.transform(df\_fin\_lsa[categorical\_features])

#Create a Pandas DataFrame of the hot encoded column

ohe\_df = pd.DataFrame(cat\_encoder, columns = col\_names)

#Concatenate with original data

df\_fin\_lsa = pd.concat([df\_fin\_lsa, ohe\_df], axis=1).drop(categorical\_features, axis=1)

In [160]:

df\_fin\_lsa\_target = df\_fin\_lsa['return.above.average']

df\_fin\_lsa\_X = df\_fin\_lsa.drop('return.above.average',axis=1)

In [161]:

from sklearn.model\_selection import train\_test\_split

train\_x, test\_x, train\_y, test\_y = train\_test\_split(df\_fin\_lsa\_X, df\_fin\_lsa\_target, test\_size=0.25, random\_state=17)

In [162]:

# build the lightgbm model

clf4 = lgb.LGBMClassifier(seed = 1)

clf4.fit(train\_x, train\_y)

Out[162]:

LGBMClassifier(seed=1)

In [163]:

# view accuracy on train set

from sklearn.metrics import accuracy\_score, roc\_auc\_score

y\_pred=clf4.predict(train\_x)

print("acc:",accuracy\_score(train\_y,y\_pred))

print("auc:",roc\_auc\_score(train\_y,y\_pred))

acc: 0.9993173129437466

auc: 0.9991486034598757

In [164]:

# view accuracy on test set

y\_pred=clf4.predict(test\_x)

print("acc:",accuracy\_score(test\_y,y\_pred))

print("auc:",roc\_auc\_score(test\_y,y\_pred))

acc: 0.9246519246519247

auc: 0.8860457566814429

In [165]:

from sklearn.metrics import confusion\_matrix, classification\_report

print(classification\_report(test\_y, y\_pred))

precision recall f1-score support

0 0.94 0.96 0.95 1861

1 0.86 0.81 0.84 581

accuracy 0.92 2442

macro avg 0.90 0.89 0.89 2442

weighted avg 0.92 0.92 0.92 2442

Sentiment[¶](#Sentiment)

In [166]:

from textblob import TextBlob

In [167]:

txtblb = [TextBlob(blob).sentiment.polarity for blob in tokens\_str]

In [168]:

sent = pd.DataFrame(txtblb)

In [169]:

df\_sent = values.merge(sent, left\_index = True, right\_index = True, how='left')

In [170]:

df\_sent.drop(['adsh','value'], axis=1, inplace=True)

In [171]:

df\_financial = df\_financial.rename(columns={'Fiscal Year':'fy'})

In [172]:

df\_fin\_sent = df\_financial.merge(df\_sent, on=['cik', 'fy'], how='left')

In [173]:

df\_fin\_sent = df\_fin\_sent.dropna()

In [174]:

len(df\_fin\_sent)

Out[174]:

9766

In [175]:

df\_fin\_sent = df\_fin\_sent.reset\_index(drop=True)

In [176]:

df\_fin\_sent = df\_fin\_sent.drop(['Report Date', 'SimFinId', 'Currency', 'fy', 'Revenue', 'cik',

'Company Name', 'Income (Loss) from Continuing Operations',

'total.debt', 'ebit', 'ebitda', 'Income (Loss) from Continuing Operations',

'segment.mean'], axis=1)

In [177]:

#categorical features

from sklearn.preprocessing import OneHotEncoder

categorical\_features = ['sic', 'lifecycle']

df\_fin\_sent[categorical\_features] = df\_fin\_sent[categorical\_features].astype('str')

cat\_encoder = OneHotEncoder(sparse=False)

cat\_encoder = cat\_encoder.fit(df\_fin\_sent[categorical\_features])

col\_names = cat\_encoder.get\_feature\_names(categorical\_features)

cat\_encoder = cat\_encoder.transform(df\_fin\_sent[categorical\_features])

#Create a Pandas DataFrame of the hot encoded column

ohe\_df = pd.DataFrame(cat\_encoder, columns = col\_names)

#Concatenate with original data

df\_fin\_sent = pd.concat([df\_fin\_sent, ohe\_df], axis=1).drop(categorical\_features, axis=1)

In [178]:

df\_fin\_sent\_target = df\_fin\_sent['return.above.average']

df\_fin\_sent\_X = df\_fin\_sent.drop('return.above.average',axis=1)

In [179]:

from sklearn.model\_selection import train\_test\_split

train\_x, test\_x, train\_y, test\_y = train\_test\_split(df\_fin\_sent\_X, df\_fin\_sent\_target, test\_size=0.25, random\_state=17)

In [180]:

# build the lightgbm model

clf5 = lgb.LGBMClassifier(seed = 1)

clf5.fit(train\_x, train\_y)

Out[180]:

LGBMClassifier(seed=1)

In [181]:

# view accuracy on train set

from sklearn.metrics import accuracy\_score, roc\_auc\_score

y\_pred=clf5.predict(train\_x)

print("acc:",accuracy\_score(train\_y,y\_pred))

print("auc:",roc\_auc\_score(train\_y,y\_pred))

acc: 0.9787001638448936

auc: 0.9686465417896231

In [182]:

# view accuracy on test set

y\_pred=clf5.predict(test\_x)

print("acc:",accuracy\_score(test\_y,y\_pred))

print("auc:",roc\_auc\_score(test\_y,y\_pred))

acc: 0.9226044226044227

auc: 0.8882538675466434

In [183]:

from sklearn.metrics import confusion\_matrix, classification\_report

print(classification\_report(test\_y, y\_pred))

precision recall f1-score support

0 0.95 0.95 0.95 1861

1 0.85 0.82 0.83 581

accuracy 0.92 2442

macro avg 0.90 0.89 0.89 2442

weighted avg 0.92 0.92 0.92 2442

1. These datasets started to be updated monthly beginning in November 2020, which is outside the range of this work. [↑](#footnote-ref-1)