Machine Learning Engineer Nanodegree

Capstone Project
Andrei Yarmak
August 24th, 2019

I. Definition

Project Overview

When it comes to financial performance of a large company (especially the publicly traded ones), a lot of information is usually readily available for shareholders and the general public. One can analyze annual and quarterly reports, press releases, analyst call transcripts, etc. However, if you want to aggregate any metrics across multiple companies (e.g., to look at marketing spend trends in a given industry over time), data gathering usually requires a lot of tedious manual work. Financial reports are usually posted in multiple different formats, include different sets of metrics, the same metrics can be called different names, a lot of the data is unstructured and buried within pages of text, etc. Even though multiple companies provide some aggregated data as a service, this service usually comes at a very steep subscription fee, which only large businesses can afford. As an alternative, it might be possible to build a suite of machine learning models that would be able to automatically analyze annual reports of public companies, parse key performance metrics, and collect all necessary context (company name, industry, currency used, etc.) to enable the ease of aggregating these data points and making industry-wide comparisons.

Problem Statement

As a first step on a route to fully automated parsing of financial data from publicly available sources, in this capstone project, I want to build a model that locates a statement of operations (also called Profit and Loss statement, or P&L) within an annual financial report. This problem is related to structuring unstructured or loosely structured data. It can be framed in several different ways: as a classification problem (i.e., if we split the text into words and then classify each word as either related or unrelated to statement of operations) or as a regression problem (i.e., using all the contents of the text at the same time (including numerical positions of every word) trying to estimate the positions of upper and lower boundaries for the statement of operations within it.

In this sense, this problem is clearly:

- **Quantifiable**: it can be framed with numerical inputs (word positions, word counts, share of digits in a text, etc.) and outputs (which in classification case would be the probability of the word being a part of the statement of operations and in regression case - numerical positions of upper and lower boundaries of the statement of operation within the text)
- **Measurable**: the problem can be measured by some metric (e.g., classification or regression accuracy)
- **Replicable**: it can be reproduced in pretty much any annual report across most companies and industries

Metrics

My key metric for picking the best model was **F-beta score** of the label corresponding to words within a statement of operations, with beta value higher than 1 (more specifically, a value of 5). The reasons for this choice are three-fold:

1. *One of the predicted categories* (more specifically, the key one - “words within P&L”) *is dramatically smaller in size* than the other two (I’ll an illustration of that later in this report). Because of that, accuracy won't be a good measure, as even getting all of that key label predictions wrong can still coincide with high accuracy on the overall data set
2. *We care much more about recall* in this case than precision. In other words, while having too many false positives is not preferable, it can still be dealt with to some extent later through additional data processing (e.g, by only picking the predicted key labels that are adjacent to each other and collectively have the highest probability scores). False negatives are a much larger problem though, as they essentially mean some of the P&L data will be lost for further analysis. Therefore, a simple F1 score is not gonna be enough, as it assumes equal importance for precision and recall
3. We don't really care about precision and recall between two of the three predicted categories (“word is above P&L” and “word is below P&L”), so if some of those labels are mixed, this should not be a problem. Therefore, we really need to look at the score of our most important label only (“word is within P&L”). Later in this notebook, we will explore whether we can even merge those two other labels (in which case we would have a binary label, and the overall F-beta would work as well)

II. Analysis

Data Exploration

In this project, I will be using publicly available data from EDGAR (Electronic Data Gathering, Analysis, and Retrieval system), a database of all public filings that companies make with US Securities and Exchange Commission (SEC). More specifically, EDGAR provides access to

forms 10-K, which include most important company financial and operating results of the company for a year (annual report), as well a lot of management's commentary on the company's performance.

As part of this project, I collected and labeled all necessary data from the site. More specifically, data collection phase included two major parts:

1) **Web Crawling**: Building a web crawler to navigate EDGAR website and download all relevant files
2) **Label tagging**: Finding and tagging statements of operations in the text (via a combination of RegEx searches and manual checks / overrides)

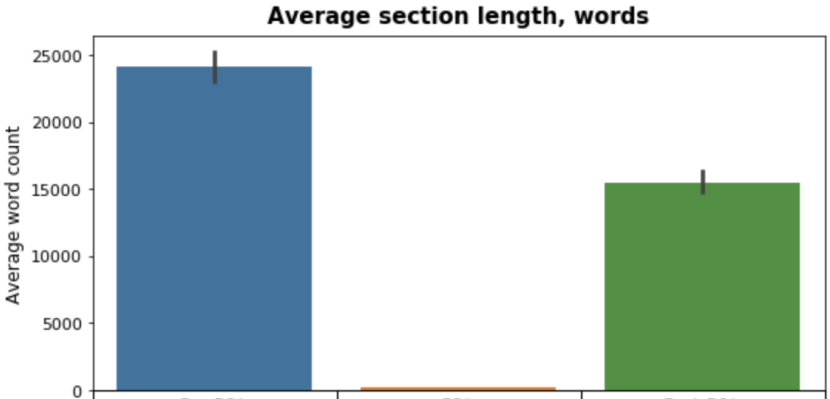
Final dataset included a table of *596 annual reports and statements of operations*. All reports come from the filings made by different companies in the second quarter of 2019. The table included the company name ('dir'), the whole 10-k annual report downloaded from EDGAR ('report'), and a statement of operations (also called Profit and Loss statement, or P&L) parsed from the report ('pnl'):

	dir	report	pnl
0	12 Retech Corp	\n10-K\n1\nform10-k.htm\nUNITED\nSTATES\nSECUR...	3\n12\nRETECH CORPORATION\nConsolidated\nState...
1	1847 Holdings LLC	\n10-K\n1\nefsh_10k.htm\nFORM 10-K\nefsh_10k.h...	3\nTable of Contents\n1847 HOLDINGS LLC\nCONSO...
3	4M Carbon Fiber Corp.	\n10-K\n1\nf10-k4mcarbonfiber.12.31.18.htm\n4M...	2\nSee accompanying notes to the consolidated ...
4	8X8 INC _DE_	\n10-K\n1\na10-kdocument.htm\n10-K\nDocument\n...	50\nTable of Contents\n8X8, INC.\nCONSOLIDATED...
5	AAC Holdings, Inc.	\n10-K\n1\naac-10k_20181231.htm\n10-K PROJECT ...	2\nAAC HOLDINGS, Inc. \nCONSOLIDATED STATEMENT...

The data contained no missing variables or other obvious abnormalities.

Exploratory Visualization

To better understand the data, during pre-processing I plotted the average size of each section of the report, as well as standard deviation:



	Pre-P&L	P&L	Post-P&L
Mean	24126	188	15521
St.deviation	13984	71	9253

As becomes clear from this chart, our key target label ("P&L") represents a very small share of the overall data (less than 0.5%). Based on this observation, it became clear that accuracy couldn't be used as a key success metric in my analysis (as mentioned above), and I decided to stick to the F-score instead.

Algorithms and Techniques

Given the task at hand, I explored three potential model options:

1. ***Multinomial Naive Bayes*** - this model usually scales very well and often performs well in practice. Given the size of the data set, this clearly had to be one of the choices
2. ***Support Vector Machine*** - this model can pick up on more complex relationships, but is notoriously hard to scale. To overcome the scaling issues, I used a linear version of the model (LinearSVC)
3. ***Stochastic Gradient Descent*** - linear models are sometimes very performant, while also easier to understand

I also used a couple of techniques to overcome different challenges with the training:

- To facilitate hyperparameter tuning for Support Vector Machines and Stochastic Gradient Descent, I used **Grid Search** methodology with the following parameter grids:
    - Support Vector Machine:
        - Tolerance for stopping criteria: 1e-3, 1e-4, 1e-5
        - C: 0.1, 1.0, 10.0
        - Dual optimization problem: True, False
    - Stochastic Gradient Descent:

        - Alpha (learning rate): 1e-4, 1e-3, 1e-2, 1e-1, 1e0, 1e1, 1e2, 1e3
        - Loss function: Logarithmic, Modified Huber, Perceptron
        - Penalty: L1, L2

    For Multinomial Naive Bayes, I simply used the default parameters (alpha=1.0, fit_prior=True, class_prior=None) as those usually perform well off the shelf
- With the data set consisting of almost 24 million entries, training some of these models directly proved quite challenging. To make sure that all 3 models can be trained in a reasonable amount of time, I **temporarily reduced the data set to 1% of the original size**. My hypothesis (that was later validated by the modeling outcomes) was that, given the structure of the data, this should still be a relatively representative sample of the underlying data. If we think about the kind of data that was encoded in our features, it contained a lot of feature similarity across the co-located entries. More specifically, when it comes to any pair of two adjacent words, the maximum difference between their feature values might be at most a count of `1` for just one of the first 100 features (i.e. total counts for the 100-word window above the current word) and at most a `1` for one of the remaining 100 features (i.e. total counts for the 100-word window below the current word). So, theoretically, while training on all entries will drive additional precision, for the purposes of picking the right model, a down-sized version was quite enough. After the right model choice was made, I then re-trained it on the full data set

Benchmark

My original plan was to use the average position of the statement of operation (i.e., X% of total words from the top of the report and Y% of words from the bottom) across all 596 reports as my benchmark model. However, this turned out to be an extremely poor benchmark when it comes to F-scores (F-beta <0.01). So, instead, I decided to set a reasonable number based on my expectations from the model.

More specifically, I used the average size of statement of operations (188 words), the average size of the full report (39,835 words), and some assumptions about reasonable relative size of false positives (no more than 50% of true positives) and false negatives (no more than 30% of true positives) to come up with the benchmark confusion matrix:

		Predicted		
		Negative	**Positive**	**Total**
Actual	**Negative**	39,553	94	**39,647**
	Positive	56	132	**188**
	Total	**39,609**	**226**	

From this confusion matrix, I was able to calculate the benchmark precision (58%), recall (70%), and F-beta given the beta of 5 (69%).

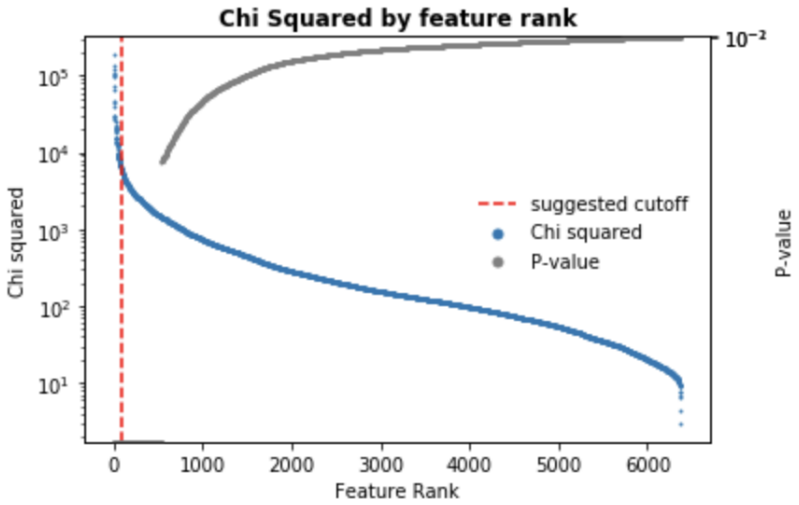
Based on these targets, I set a benchmark F-beta to 70%.

III. Methodology

Data Preprocessing

On top of significant data collection effort (briefly explained above), my project involved a lot of pre-processing work. While a complete list of steps would be too long to mention here, the most important of them included:

- **Data preparation**:
  - Converting the data from a table of 596 annual reports and statements of operations (1 row = 1 report) into a data set of 23,742,514 words (1 row = 1 word)
  - Assigning labels to each word (1 = word is above the statement of operations, 2 = word is within the statement of operations, 3 = word is below the statement of operations)
  - Cleaning up the text (removing punctuation, replacing all numbers with <number> tag to facilitate training, etc.)
- **Feature selection**. The complete text included 105,510 unique words. Making all of them separate features would be completely unreasonable - and unnecessary. Instead, I chose a subset of the most impactful words that I would then use in feature extraction.
  To do that, I used `SelectKBest` model from `sklearn.feature_selection` module to pull 100 best features based on their Chi-squared scores. Chi-squared would estimate whether relative frequency of each word is significantly different across the three sections of the report (used as labels in my Chi-squared test). The chart below illustrates how the features below my suggested cutoff of top 100 features have significantly lower Chi-squared score (mostly in 1,000-s range and below vs. top 100 features having scores around 10 - 100K) and - for the majority of them - a really high p-value:



- **Feature extraction**. This stage included one-hot encoding the 100 words with the highest predictive power (based on Chi-squared test above), and extracting the exact features that would be later used to train our classification model. More specifically, for each of the top 100 words, I extracted counts above and below a given word, within a given window (e.g. within 100 words above and 100 words below). So, my data set ended up including 200 features in total (100 for each top word above and 100 more for each top word below).
  Given the size and high sparsity of the final data set, most of these operations required the use of scipy's sparse matrices (CSR and CSC, more specifically)
- **Feature normalization**. Some of the machine learning models that I wanted to explore (i.e. SVM and SGD) required scaling of the data. For these purposes, I used `sklearn`'s `MaxAbsScaler`. The reason for this choice was that some of the more widely used scalers (e.g. `StandardScaler` or `MinMaxScaler`) don't work well with sparse data. Here are some details from [sklearn documentation]:
  - *Centering sparse data would destroy the sparseness structure in the data, and thus rarely is a sensible thing to do. However, it can make sense to scale sparse inputs, especially if features are on different scales. `MaxAbsScaler` and `maxabs_scale` were specifically designed for scaling sparse data, and are the recommended way to go about this*

Implementation

My implementation was broken down into three major steps:

1. **Pick the best algorithm**

   First, I trained and evaluated my three alternative models (Multinomial Naive Bayes, Linear Support Vector Classifier, and Stochastic Gradient Descent) on 1% of the training data (see Algorithms and Techniques section above for explanation why). Given our key evaluation metric (F-beta), Naive Bayes noticeably out-performed the two contenders

2. **Pick merged vs. unmerged labels**

   At this point, I explored a hypothesis that merging 2 of the 3 labels might lead to better model performance. While the original data includes 3 labels (1 = above P&L statement, 2 = within P&L statement, 3 = below P&L statement), we theoretically only need 2 (P&L vs. non-P&L). However, it was not a clear cut, because, if categories 1 and 3 are significantly different, merging the labels might create additional "confusion" for the model - which actually turned out to be exactly the case

3. **Pick the training set size**

   Finally, I trained my winning modeling combination (Naive Bayes with un-merged labels) on the full data set to achieve additional boost in performance. As expected, while the F-beta increased from 0.70 to 0.71, this improvement can be

considered rather marginal - and the decision to initially sample only 1% of the data for model selection fully validated

Dealing with **memory limitations** was a major complication during implementation. To mitigate it, I involved different techniques during different stages of the project:

- Using sparse matrices where possible
- Accounting for differences between CSR and CSC sparse matrix columns: performing row-level operations with CSR, column-level operations with CSC, and converting between the two
- Where necessary, dropping large variables after they've been used
- Performing model selection on 1% of the data, and then re-training the winning model on the full data set
- Using alternative, more scalable versions of the models where needed (e.g., Linear SVC vc. pure SVC)

Refinement

I performed several refinements to the original approach, some of which worked well and some didn't:

- Gradually adding all the memory-conserving techniques mentioned above. The initial version didn't even allow most of the models to be trained
- Converting all numbers in the data set to a single tag (<number>). This dramatically reduced the number of individual words in the data set and significantly improved the quality of feature selection
- Testing merging of the labels (see explanation in Implementation section above). This hypothesis didn't lead to improvement in model performance
- Reducing the number of words above and below a given word that are accounted for in my 200 features. I re-ran my code for 25- and 50-word windows, and in both cases the performance was worse than the original 100-word version

IV. Results

Model Evaluation and Validation

The best performing model turned out to be a Multinomial Naive Bayes with the following parameters:

- Alpha (additive smoothing parameter): 1.0
- Learn class prior probabilities: True

- Prior probabilities of the classes: None

As discussed in the previous sections, to validate this result I conducted several types of sensitivity analysis:

- Data manipulations: training the model on 100%, 1%, and 0.1% of data. While 100% option turned out to perform the best, both 1% and 0.1% versions weren't dramatically far apart in terms of performance, which gives me a good indication about the stability of this solution
- Model comparison: comparing Naive Bayes performance to SVM and Stochastic Gradient Descent. Naive Bayes significantly outperformed its contenders
- Label merging. While merging the labels didn't lead to superior results, the performance of the model was still relatively good
- Changing feature definitions (going down from 100- to 50- and 25-word windows when calculating word counts above and below a given word). Solutions with smaller windows didn't outperform the initial model, but they still worked quite well, with the performance decreasing in parallel to the window size decrease (i.e., 50-word solution being worse than 100-word one, and 25-word one worse than 50-word one). Again, this consistency gives me confidence in the stability of this model performance under different conditions

Justification

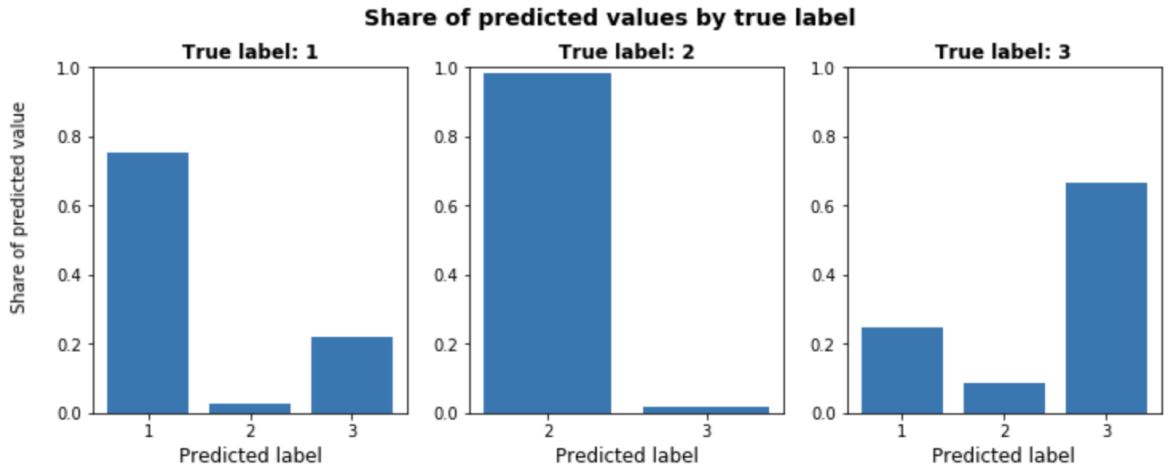
Our target metric, F-beta on the key label ("word is within P&L") for the winning model reached 0.712, which is higher than our benchmark value of 0.7. Recall - a critical metric for our purposes - reached an impressive 0.985!

This result is definitely significant given the purpose of the model. As mentioned above, false positives, while undesirable, don't represent too big of a problem in our case. In some cases, they can still be corrected through data post-processing (e.g, by only picking the predicting labels that are adjacent to each other and collectively have the highest probability scores). Otherwise, even if the borders of the parsed statement of operations are a little bit broader than labeled, this parsing would still be good enough to provide valuable context for the further parsing of the annual report (e.g., defining which numbers correspond to the marketing expenses in each year, etc.). False negatives, on the other hand, are much riskier to have, as they can mean that some data might be missing from the parsed statement of operations.

V. Conclusion

Free-Form Visualization

To further illustrate the outcome, the picture below showcases the final confusion matrix in a bar chart form:



As you can see, almost all of the cases when true label = 2 (i.e., “word is within statement of operations”) were correctly classified by the model. This corresponds well to the recall value of 0.985 and showcases that the share of false negatives in the result is very low.

Reflection

Looking back at this project, it’s interesting to note how much energy and time actually went into data pre-processing.

Data collection obviously took a lot of time as well, but that was mostly expected, as I had to scrape the web and then add labels myself. However, with some brainstorming on ways to automate this process (e.g., by writing helper functions that would make raw text observable in creative ways to streamline the creation of RegEx patterns), even data collection wasn’t as bad.

However, pre-processing took a lot of time for the reasons that I didn’t necessarily predict: partially because of the need for multiple complex data manipulations, but mostly because of data volume and memory timeout issues. To be able to process the data within a reasonable time, I had to dive quite deep into different sparse matrix formats, invest a lot in multiple code optimizations (discussed previously in Implementation section above) and, in general, be very

strategic about how to approach the implementation. Always thinking ahead and planning for what exactly I would need not only in the very next step, but also several steps ahead, was truly a key to significantly speeding up the development.

Improvement

There are multiple ways to improve on the current solution, but I would probably prioritize at least two:

1) First train the models to parse different, easier to spot, financial metrics (e.g., revenue, profit, total assets, total liabilities, shareholder's equity, etc.), and then, instead of tagging all numbers the same way (e.g., with <number> tag as I did in this project), tag those separately (<revenue>, <net_profit>, etc.). This would give the model invaluable information to better differentiate statement of operations from other financial statements (i.e., balance sheet, cash flow statement, etc.), as well as from the rest of the report in general. I would imagine that those high-level financial metrics shouldn't be difficult to parse using some of the same methods that I use in this project
2) Explore some of the advanced NLP models that came out in the last 1-2 years: BERT, GPT-2, ULMFiT, etc. With the use of transfer training, these models provide a lot of previously non-accessible rigor even to the projects with relatively sparse input data