CS 230 – Final Project

Knowledge base construction from richly formatted text using Fonduer: Applications to financial reports

Roberto Seminario, Andrea Aguirre
rseminar@stanford.edu, andreaar@sranford.edu

1. Introduction

Financial reports of publicly listed companies are used all over the world to make important decisions. However, there is a lot of discretion on how companies chose to disclose information, or how to present it in their filings. Financial analysts spend long time scavenging for hard to find information. In this context, designing a tool that automatically extracts financial information from SEC filings could revolutionize the way it is processed and extracted. But, this is not simple.

Knowledge base construction (KBC) is the process of populating a database with information extracted from unstructured data. Relevant progress has been achieved with simple text, but less work has been done with richly formatted data. It is inherently more challenging because the attributes and relations are expressed in a combination of textual, structural, tabular and visual signals. We developed an application for Fonduer, a machine learning based KBC system for richly formatted data that was developed by a group of Stanford professors and PhDs.

We parse html documents and convert them into hierarchical data models. We then create rules (regEx, dictionaries, structural) to preliminary identify pairs of [date, revenue] mentions, called candidates, that will fill the database. Then, we label the potential data candidates and determine marginal probabilities for each pair through weak supervision (data programming and labeling functions). Lastly, we train a model to compute the marginal probability of a candidate being a "True" relation.

2. Dataset / Processing

We have downloaded 2,007 html-formatted quarterly reports (SEC's 10-Q forms) of publicly traded companies. This data corpus is comprised of 3 million sentences, more than 60 million text spans and ~300,000 tables. The corpus is organized in a hierarchical data model with tabular, visual and HTML metadata and stored in a PostgreSQL database. We have randomly divided the data in three parts: a training set (70%), a development set (15%) and a test set (15%).

The Fonduer pipeline and data preparation

Our objective is to find the correct mentions of companies' revenues ('revenue mention') and their respective time period ('period mention'). Since there are over 60 million words in the corpus, the combinatorial potential of all pairs is over 3.5e+15. It is unfeasible to train a model this size. To tackle this, we defined hard-filtering functions called matchers that set forth the criteria to select the correct mentions. Matchers are logical, tabular, format, content, linguistic, and RegeX rules that eliminate text spans in the data corpus from being considered as mentions. We developed six revenue matchers and two period matchers (see Figure 1 for example matcher rules).

Figure 1. Selected matcher rules

Revenue	Integer_matcher	If the first character of the text span is an integer, return True
Revenue	No_whitespace_matcher	If the text span has no whitespace, return True
Revenue	Row_has_revenue_matcher	If the text span's row has a revenue-related word, return True
Period	Period_dict_matcher	If the span has a format similar to "Month Day," then return True
Period	In_table_matcher	If the potential mention is contained within a table, return True

Generating candidates

Matchers provide an initial set of mentions (27,977 period and 14,802 revenue) that combined together span a total of 414 million potential candidates (pairs of mentions). To avoid this combinatorial explosion, we used a hard filtering function that acts on candidates rather than on mentions (a throttler). By using a throttler we forced both mentions in a candidate to belong to the same table and end up with 17,217 candidates.

We measured the quality of the candidates by analyzing the recall of the hard-filtering process. Out of the true candidates, our data pre-processing identified 92% of candidates. For an initial setting, this recall is high enough; however, targeting 99% would be desirable in real life applications.

3. Model

Labeling functions

Although the number of candidates has been reduced from 3.5e+15 to 17,217, they still lack labels. To label the data, we apply weak supervision using the snorkel data programming framework (Alexander Ratner, 2017). The idea behind weak supervision is that it's better to have 100k noisy labels (that are 60% accurate) than 1k true labels (that are 90% accurate). To apply the Snorkel framework, we create labeling functions that, unlike hard-filter rules, are used to model a probability that a particular candidate is true. We included 14 labeling functions that tried to impart the heuristics of what represents a typical revenue and period figure in a financial statement: 1) it is located in the first pages of the document, 2) it is early in a table, 3) its table contains words like 'revenue' 'costs', 'net income', 'taxes', etc, 4) the text spans above the table typically contain words like 'income statement', 5) the revenues figures are not too short, 6) the tables where revenues are located tend to be large tables, etc.

> *"In data programming, we learn the accuracies of the labeling functions (LFs) by observing their agreements and disagreements with each other; we then use the predictions of this generative model as noise-aware training labels." (Ratner, 2017)*

All these labeling functions are fed into the Fonduer labeler module which builds a matrix of labeling outputs of shape [candidates, # labeling functions] which is then fed into a GAN that models 1) the propensity of the functions to label data and 2) the correlation between labeling functions to generate a probability. These probabilities are then the 'labels' we use to train a model

Figure 2: The generative model used by Fonduer to generate marginal probabilities

$\phi_{i,j}^{\text{Lab}}(\Lambda, Y) = \mathbb{1}\{\Lambda_{i,j} \neq \emptyset\}$ $\phi_{i,j}^{\text{Acc}}(\Lambda, Y) = \mathbb{1}\{\Lambda_{i,j} = y_i\}$ $\phi_{i,j,k}^{\text{Corr}}(\Lambda, Y) = \mathbb{1}\{\Lambda_{i,j} = \Lambda_{i,k}\}$	label matrix Λ, is encoded in a generative model pw(Λ, Y) using three factor types, representing the labeling propensity, accuracy (not used because we didn't input true labels), and pairwise correlations of labeling functions
$p_w(\Lambda, Y) = Z_w^{-1} \exp\left(\sum_{i=1}^{m} w^T \phi_i(\Lambda, y_i)\right)$	The vector ϕ is the concatenated vector of the factors described above, w is the weight matrix and Zw-1 is a normalizing constant
$\hat{w} = \arg\min_w -\log \sum_Y p_w(\Lambda, Y)$	We optimize w with respect to the log loss of the probability compared to the noisy label to optimize

We also developed a set of alternative custom marginal probabilities. To compute these, we created a function that calculated a score for each candidate based on the true/false values that each labeling function outputted. However, we didn't take into account correlations between labeling functions (AS Fonduer does). To account for differences in scoring levels between documents, and considering that each document needs to have an output (revenue and date candidates), we normalize the scores per document.

When deciding between a matcher/throttler (hard-filter) and a labeling function (soft-filter), we face a trade-off. By applying hard filters, we reduce the size of the data but we risk sacrificing the model's recall. It was a trade-off between computational feasibility and a more robust/generalizable model that learns as opposed to operate with hard filtering rules.

Model training

We created a feature vector that takes into account several dimensions of information for each candidate: one-hot vectors of surrounding words (including combination of words), style features (font, size, bold, caps, etc.), NLP structure, length, lemma sequences, row and column headers, table characteristics, page location, object hierarchy, html tags, and others. Our input vector has 107,000 features. There is a linear increase in the number of features as the size of the dataset increases. This is mainly because the spans of text surrounding revenue figures tend to be other numbers which differ in every document, so they need to be encoded as one-hot vectors each. Plus, every document has its own set of particular combinations of numbers which are rarely repeated in other documents.

With the set of features, and the weak labels, we ran different model specifications. We tested different structures and saw performance improvement by including additional layers. However, after adding too many layers, the model began overfitting and F1 performance on the dev set suffered. We also tested our custom marginal and compared its performance with Fonduer's labeling. We found that the labels generated by Fonduer were superior to our own custom marginals. Finally, with 2-4 hidden layers, the model seemed not to overfit, but as more layers were added, performance in the development set was affected.

4. Results

To measure the model's performance, we extracted a set of true labeled period-revenue pairs from Capital IQ, a provider of financial information. By doing this, we are able to test the true performance of our model and not only its performance under the noisy labels generated by weak supervision.

As the model estimates marginal probabilities, we had to choose a threshold to classify labels as True or False. Using a unique threshold to cut off values across different documents implied that for certain documents, no single candidate was labeled as True, which is incorrect. Ideally we would've used a bias term for each document to account for the differences between documents. However, this defeats the purpose of the model, which is to generalize well on previously unseen documents. To force having True labeled candidates, we then implemented a threshold selection by document. After testing multiple iterations we found that choosing the top 10 candidates with the highest marginal probabilities as True yielded the highest F1 performance.

The best performing model in terms of recall was the neural network with the smaller structure. While the best performing model in terms of F1 score was the neural network with larger structure. Overall, the performance was similar in both models as shown below

Figure 4: Cross Entropy loss by epoch in the training set

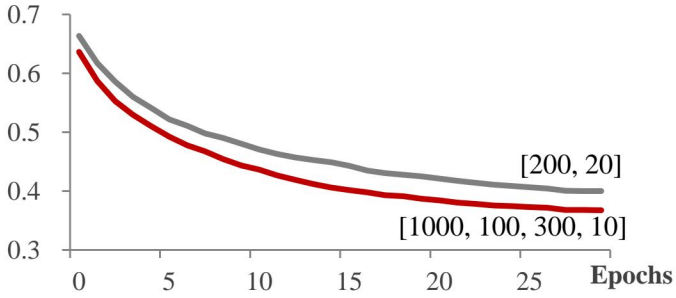


Figure 5: Performance metrics

Larger NN	Development	Test
Accuracy	33.8%	35.7%
Precision	24.8%	28.0%
Recall	95.3%	87.8%
F1 score	39.4%	42.4%

Smaller NN	Development	Test
Accuracy	27.2%	31.0%
Precision	23.2%	27.2%
Recall	96.3%	92.8%
F1 score	37.4%	42.0%

We also compared the performance of our custom labeling functions with those generated by Fonduer. Our custom marginal considerably underperformed the marginal probabilities generated by Fonduer. While the cross entropy loss of the model trained with our custom marginal reached 0.6, the model trained with Fonduer's marginal reached a loss of around 0.35.

Figure 6: Founder marginal histogram

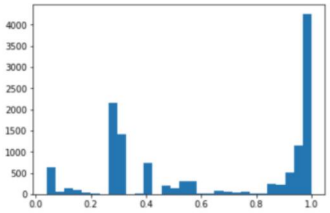


Figure 7: Custom marginal histogram

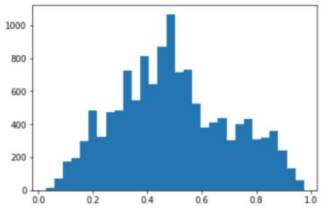
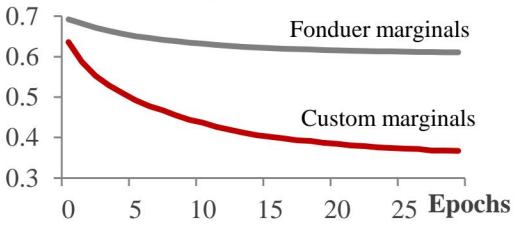
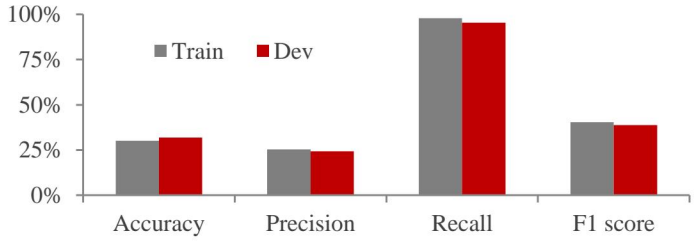


Figure 8: cross-entropy loss using different marginal functions



Finally, to determine whether we were overfitting our data, we compared the performance of our model in the training set and the development set. Although performance was generally better on the training set, the performance differential was low, suggesting that our model is not overfitting.

Figure 9: Performance on train set



5. Conclusions / Expansions

The task of getting accurate labeled candidates among the explosive initial amount of datapoints was a more challenging task than we expected. But through a combination of hard filters and weak supervision, our model was able to pinpoint a handful of revenue and date candidates out of billions of potential pairs. After testing different options for our model, the F1 performance obtained was around 40%. Comparing with the baseline performance of 0% for the broad dataset and 20% for the filtered one, we consider that the performance is decent. However, more than half of the model's performance was in the hard-filtering portion (20%). Hence, as a potential expansion of the model, we could try fewer hard filters and train the model with more candidates (over 1mm ideally). Another potential expansion, to avoid computation unfeasibility, is to trim the features from 107,000 to less than 10,000. As currently, some of the feature vectors are very sparse. Overall, we believe that creating a structural model of financial data is computationally inefficient for a task like this. It is possible that table and page extraction based on heuristics paired with a machine learning model would be more efficient.

CS 230 – Final Project

Members contributions

Roberto Seminario: set up and management of AWS's EC2; creation of PostgreSQL database, definition of matchers and labeling functions, definition of custom marginals, code of neural network model, run iterations of model

Andrea Aguirre: definition of matchers and labeling functions, run iterations of model, prediction classification, performance metric evaluation

References

[1] Sen Wu, Luke Hsiao, Xiao Cheng, Braden Hancock, Theodoros Rekatsinas, Philip Levis, and Christopher Ré. 2018. Fonduer: Knowledge Base Construction from Richly Formatted Data. In Proceedings of 2018 International Conference on Management of Data, Houston, TX, USA, June 10–15, 2018 (SIGMOD'18), 16 pages.
[2] Alexander Ratner, S. H. (November 2017). Snorkel: Rapid Training Data Creation with Weak Supervision. Stanford University.