What You Say And How You Say It Matters

Isaac Haberman
Center for Data Science
New York University
ijh216@nyu.edu

Yiyan Chen
Center for Data Science
New York University
yc2462@nyu.edu

Lu Yin
Center for Data Science
New York University
ly1123@nyu.edu

Xiaofan Bai
Center for Data Science
New York University
xb332@nyu.edu

Abstract

To better prepare JP Morgan Chase's Investor Relations Team for upcoming events, we have developed a machine learning modeling framework that generates tag predictions and sample questions for a given set of analysts. By transforming the analyst's questions into word vectors and utilizing cutting edge dimension reduction techniques, we achieved workable results and produced sample questions for each analyst and tag. With the framework, the Investor Relations Team will be able to mathematically judge their intuitions and better prepare other teams in JP Morgan Chase.

1 Introduction

Each quarter, public companies hold an earnings call, a call between the Chief Executive Officer (CEO) and Chief Financial Officer (CFO) of the company and a group of well-known financial analysts. While the call is preceded by opening statements from the company, the question and answer session is the crux of the call. Each side has a goal for the call, with the analysts looking for understanding and insight into the company's previous quarter and into the coming quarters, and the company trying to please the analysts. While these earning calls are not the only communications the analysts and the company have, many of the other events follow a similar methodology. The company aims to convey a positive representation of their work while accurately answering the analysts' questions. Therefore, we sought to use machine learning and deep learning methods to predict the types of questions analysts will ask prior to a call, and what those questions will be.

2 Existing and Related Work

Prior to this project, a previous student team, working for JP Morgan Chase, had a developed a machine learning model to tag questions from earning calls with one of fourteen tags: Accounting and Taxes, AWM, Balance Sheet, Capital, CB, CCB, CIB, Credit Costs, Expenses, Legal, Macroeconomic Updates, Other Topics, Regulatory Topics and Revenue. While we were not given access to the model nor its inner workings, we were given a cursory explanation of its behavior and use case. Each question was pre-processed before being run through a multi-class classifier and classified. The model has a 75% accuracy rate and was used to classify the training data.

Recently, there has been a shift in text generation methods from template rule based systems to neural networks. The traditional rule-based text generation system as mentioned in Dale et al. (2003) is divided into the following steps: context planning, sentence planning, and surface realization. This type of text generation is easy to implement and often performs well on different domains; however,

it does not generalize to other tasks. These issues continue to arise with the advent of probability language models, as they too struggle with new words. Neural networks alleviate some of these shortfalls by using treating words as word embeddings, multidimensional vectors that can be learnt during model training. Lastly, the most recent innovations in neural network text generation, Cho et al. (2014), use the positioning of the other words in the sequence to further improve the performance of the models.

For our purposes, we honed in on the work done on Long-Term Short-Term (LSTM) and Gated Recurrent Units (GRU) models as they represent the state of the art neural network based models. 1 has the derivations for an LSTM, an improvement on the vanilla recurrent neural network (RNN), that combats the memory loss of vanilla RNN's. Note, that $f, i, \tilde{c}_t, c_t, o, h_t$ denote the forget gate, input gate, memory candidate state, updated memory state, output gate, and output as in Cho (2016).

$$
\begin{aligned}
f &= \sigma(W_f \phi(x_t) + U_f h_{t-1}) \\
i &= \sigma(W_i \phi(x_t) + U_f h_{t-1}) \\
\tilde{c_t} &= tanh(W_c \phi(x_t) + U_c h_{t-1} \\
c_t &= f \odot c_{t-1} + i \odot \tilde{c_t} \\
o &= \sigma(W_o \phi(x_t) + U_o h_{t-1}) \\
h_t &= o \odot tanh(c_t)
\end{aligned}
$$

Figure 1: Derivation of LSTM Unit

We further investigate and derive, GRU's, an improvement on the LSTM, that modify the gates and memory of the LSTM. Below is the derivation of the GRU's inner working, courtesy of Cho (2016). Note $r, u, \tilde{h_t}, h_t$ denote the reset gate, update gate, candidate hidden state and hidden state respectively.

$$
\begin{aligned}
r &\in [0,1]^{n_k}, u \in [0,1]^{n_h} \\
r &= \sigma(W_r \phi(x_t)) + U_r h_{t-1} \\
u &= \sigma(W_u \phi(x_t) + U_u (r \odot h_{t-1}) \\
\tilde{h_t} &= g(W \phi(x_t) + U(r \odot (h_{t-1}))) \\
h_t &= (1-u) \odot h_{t-1} + u \odot \tilde{h_t}
\end{aligned}
$$

Figure 2: Derivation of GRU Unit

3 Methods

3.1 Data

The original data-set was a heavily marked-up transcription of 200 earnings calls, conferences, fixed income calls, and investor days. Across the data-set, there were seventy-nine unique analysts and five companies, including the major U.S. banks like JP Morgan Chase, Citigroup and Goldman Sachs. Each transcript was spread over N rows, each row representing a question and answer. Additionally, each row had the following additional features: date of the call, the company participants, the questioner, and the question tag.

Over the course of the project, we built a PDF Parser, a Python script that read through additional earnings calls and events that we were given access to. With those additional events, we added over 100 additional events to our data.

3.2 Exploratory Data Analysis

While the modeling and question generation were the main objectives of the project, there were a few outstanding questions we sought to answer prior to modeling. For each analyst, we derived

the distribution of tags across the calls, hoping to answer whether, given enough calls, each analyst converged in distribution. If so, we theorized that the analysts themselves had little predictive power and it was the calls themselves that predicted the questions. In other words, the analysts all had similar questions prior to the call and it was the just the ordering of the analysts that determined who asked which question. Had the distributions diverged, the analysts themselves could be predictive to the question content and therefore the tag. Additionally, we derived the distribution of tags across calls, and tags across quarters.

3.3 Tag Model

To effectively generate questions for an upcoming earnings call, we first sought to predict which tags each analyst was most likely to use. To do so, we developed a series of classifiers that took in the date, company and a given set of analysts, and output the probability of each of the fourteen tags for each of the analysts.

The first generation classifier (V1), a multi-class classifier, predicted the tag of each question given the analyst, company and date. Given the required format of the data, as well as the model's poor performance, we quickly abandoned this model.

Our second generation classifier (V2) used a "menu" approach to model the tags. Instead of predicting the tag of each question, we sought to predict the probability of each analyst asking a question of each tag. In other words, for each analyst in a given event, we had fourteen rows, one for each tag. To better represent the analysts, we included a one-hot encoded version of them as well as the historical percentages of each use of the tags. To maintain the proper assumptions of predictive modeling, we updated the historical percentages prior to each call, excluding the question for any call that occurred after the event. Similarly, we did the same for both the companies and the quarters. Given the additional features, the model suffered from increased sparsity and dimensionality. To alleviate these concerns, we experimented with dimension reduction algorithms prior to modeling. Despite the feature generation, we were unimpressed with the model inputs and outputs and tested a third generation.

For our third and final generation (V3), we chose to incorporate the questions themselves into our model. Instead of using the historic tag percentages, and one-hot encoding the analysts, we converted the analysts into vectors through topic modeling and tried and "menu" approach again. To do so, we generated the following matrices: *Analyst X Word*, *Tag X Word* and *Company X Word*. For each of those matrices, we used Lucene BM-25 to fill the values before running non-negative matrix factorization to generate reduced matrices. An outline of this can be seen in 3. This process was repeated for each of the three matrices and the one-hot encoded features were replaced. To choose the new dimensionality, we used Žitnik and Zupan (2012)'s Python package to iterate through different dimensionalities before choosing the proper dimension for each of the matrices. A sample visualization of that methodology can be seen in 18.

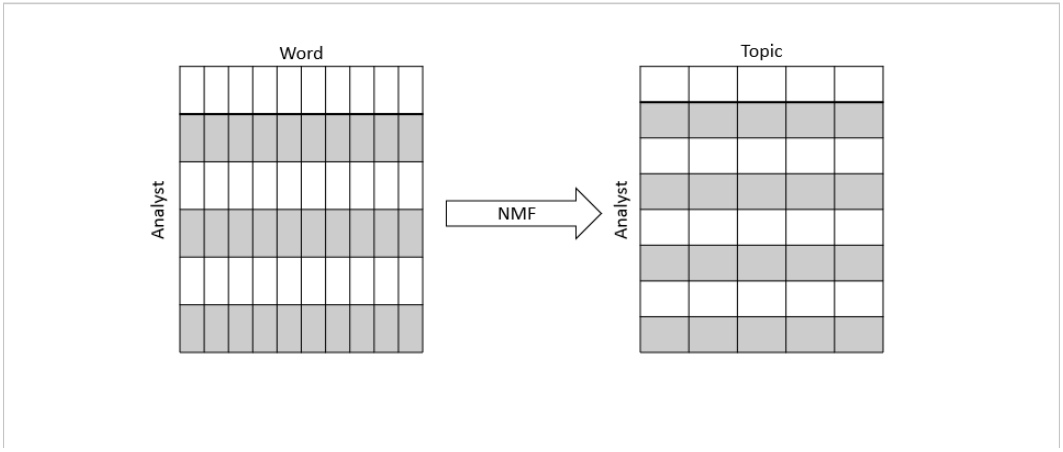

Figure 3: Analyst Dimension Reduction

Since the "menu" approach to modeling does not capture the sequential aspect of the earnings call, we used a greedy naive assumption to generate the final sequence of questions. Beginning with the question with the highest probability of being asked, we ordered the analysts and used their question with the next highest probability to generate a sequence of tags. Continuing in this approach, we moved through the analysts until we had asked enough questions to fulfill the average amount of questions per event. Lastly, to produce sample questions, we built one additional text matrix, *Question X Word*, which we multiplied by both *Analyst X Word* and *Tag X Word* to generate *Question X Analyst* and *Question X Tag* affinity matrices. From those, we averaged the results across the two affinity matrices to choose sample questions for each event. An outline of this process can be seen in 4.

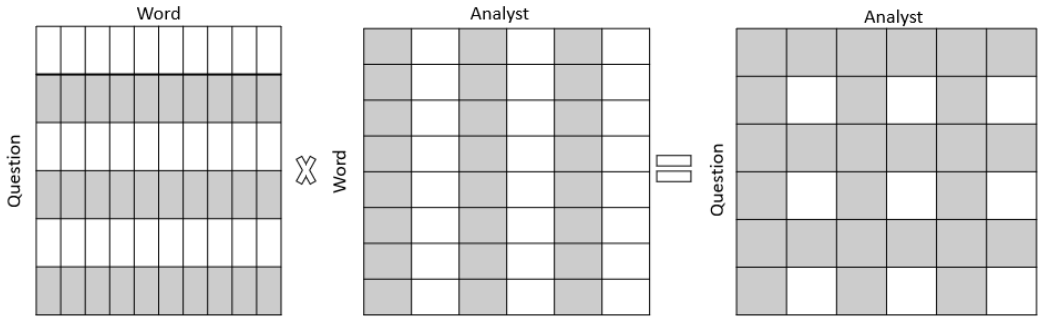

Figure 4: Affinity Matrix Creation

The results of all these processes and models can be seen in 4.

3.4 Question Generation

3.4.1 Tokenization And Word Embeddings

As the questions in our data-set varied in length and complexity, we used advanced cleaning and tokenization methods to pre-process the data before use in the models. For instance, a few of the shorter questions and answers can be seen in 5. To avoid the shorter questions like these, we set the minimum number of words per question to 10.

```
Understood. Okay.
Okay, that's fair.
Okay. All right.
Okay. Thank you.
  Yeah. Understood.
Yeah.
Okay.
Okay.
```

Figure 5: Short Question Examples

For the tokenization, especially for word-level model, we kept as much of the original text as possible, including punctuation, to best hone in on the syntax of the questions. As in most language modeling work, we appended start and end of sequence tokens to our sequences. That being said, our eventual vocabulary size was 5,338 for the word-level model and 99 unique characters for the character-level model.

To improve the model performance, we used pre-trained word embeddings Mikolov et al. (2018) in the word-level model. Since many of the financial terms in our corpus were not present in the pre-trained word embeddings, we also learnt our own embeddings for some of the corpus. We also developed two word-level models because of this, one that used the pre-trained word embeddings and one that learnt its own embeddings as it trained.

3.4.2 Generating Sequences

To generate sequences, we invoked the Markov Assumption, namely that each word is dependent on the *n* previous words. By modeling in this manner, we avoid the deterministic aspect of the modeling and instead are able to build shorter follow-up questions per each sequence. Additionally, since many questions are in fact made-up of shorter less complex question, as seen in 6, we iterate one word at a time, allowing for the formation of the shorter question. A sample of these outputs can be seen in 7. In the example, the 10 previous words are used to predict the next word, hence the prediction of "net", "interest" and "income."

```
'On net interest income, do you have an outlook for how the net interest income dollars could trend from here, assumi
ng that you don't get much help from higher rates, what are the key drivers? And what's kind of your outlook for NIM
and NII dollars for the year?'
```

Figure 6: Original Sentence

```
sample #0: ...on net interest income , do you have an outlook -> 'net'
sample #1: ...net interest income , do you have an outlook for -> 'interest'
sample #2: ...interest income , do you have an outlook for how -> 'income'
sample #3: ...income , do you have an outlook for how the -> ','
sample #4: ..., do you have an outlook for how the net -> 'do'
```

Figure 7: Word-Level Sequence Preparation

While the word-level model requires a sequence of 10 words for usable results, the character-level model requires the use of 60 characters, including white space and punctuation for similar results. 8 has an example of that process below.

```
sample 0: outlook for how the  ---> 'n'
sample 1: utlook for how the n ---> 'e'
sample 2: tlook for how the ne ---> 't'
sample 3: look for how the net ---> ' '
sample 4: ook for how the net  ---> 'i'
```

Figure 8: Character Level Sequence Preparation

3.4.3 Modeling

We implemented LSTM and GRU based models for both word and character based sequence generation. Like Mei et al. (2015), we used encoder-decoders as the basic architecture of our models. 9 shows the inner workings of our Word-level LSTM model. Note the embedding layer reads in word indices and transmogrifies them to vectors of two-hundred dimensions.

Layer (type)	Output Shape	Param #
embedding_5 (Embedding)	(None, 9, 200)	1167600
lstm_9 (LSTM)	(None, 9, 200)	320800
lstm_10 (LSTM)	(None, 200)	320800
dense_9 (Dense)	(None, 200)	40200
dense_10 (Dense)	(None, 5838)	1173438

Figure 9: Word-Level LSTM

As GRU's is an improvement over LSTM's, we also tested a word-level GRU. It's inner working can be seen in 10.

Layer (type)	Output Shape	Param #
embedding_5 (Embedding)	(None, 9, 200)	1167600
lstm_9 (LSTM)	(None, 9, 200)	320800
lstm_10 (LSTM)	(None, 200)	320800
dense_9 (Dense)	(None, 200)	40200
dense_10 (Dense)	(None, 5838)	1173438

Figure 10: Word-Level GRU

As it is difficult to develop word embeddings, especially with a small training set, we tested character-level models. By moving through sequences of characters instead of words, we theorized that the model would have an easier time using the financial terms. Words like "FICC" which would not be used in the word-level model, can be easily caught by the character-level model. Additionally, since there are many fewer individual characters than words, model training time is greatly reduced and calculations are greatly sped up. Lastly, the character-level model is easier to implement than the word-level model as there is no word embedding layer, each letter is used as its one-hot encoded vector. Below are the inner workings of two iterations of our character-level modeling, a unidirectional, 11 and a bidirectional LSTM, 12.

Layer (type)	Output Shape	Param #
lstm_1 (LSTM)	(None, 60, 512)	1253376
dropout_1 (Dropout)	(None, 60, 512)	0
lstm_2 (LSTM)	(None, 60, 512)	2099200
dropout_2 (Dropout)	(None, 60, 512)	0
lstm_3 (LSTM)	(None, 60, 512)	2099200
dropout_3 (Dropout)	(None, 60, 512)	0
lstm_4 (LSTM)	(None, 512)	2099200
dropout_4 (Dropout)	(None, 512)	0
dense_1 (Dense)	(None, 99)	50787
activation_1 (Activation)	(None, 99)	0

Total params: 7,601,763
Trainable params: 7,601,763
Non-trainable params: 0

Figure 11: Character-Level Unidirectional LSTM

Layer (type)	Output Shape	Param #
bidirectional_1 (Bidirection	(None, 60, 1024)	2506752
dropout_1 (Dropout)	(None, 60, 1024)	0
bidirectional_2 (Bidirection	(None, 60, 1024)	6295552
dropout_2 (Dropout)	(None, 60, 1024)	0
bidirectional_3 (Bidirection	(None, 60, 1024)	6295552
dropout_3 (Dropout)	(None, 60, 1024)	0
bidirectional_4 (Bidirection	(None, 1024)	6295552
dropout_4 (Dropout)	(None, 1024)	0
dense_1 (Dense)	(None, 99)	101475
activation_1 (Activation)	(None, 99)	0

Total params: 21,494,883
Trainable params: 21,494,883
Non-trainable params: 0

Figure 12: Character-Level Bidirectional LSTM

For generating the sequence, the language models generally employ a greedy search algorithm to choose the next token in each sequence. At each iteration of the model, the token with the highest

probability of being chosen is appended to the sequence. The more mathematically inclined would recognize this process as maximizing the log loss of the sequence. Another method of sequence generation that we tested but did not use is beam search. Like the greedy search, beam search's goal is to maximize the log odds of the sequence, however, unlike greedy search, beam search considers additional tokens at each iteration. At each iteration through a sequence, we generate N copies of our best N sequences and assign one of each of the top-N suggested tokens to each of the sequences. At the conclusion of that iteration, the top N sequences are taken, using their probabilistic log loss to determine scores. While beam search has the ability to generate higher probabilistic sequences, it is a significantly slower process than using a greedy algorithm to always take the token with the greatest probability at each iteration. A visual explanation of beam search can be seen in 13.

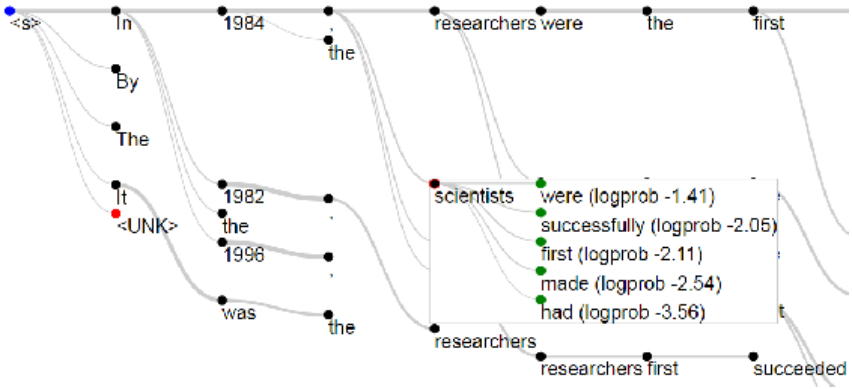

Figure 13: Model Beam Search

4 Results

4.1 Exploratory Data Analysis

We visualized the distribution of tag for each analyst. As we had theorized, for many of the more prevalent analysts, the tag distributions appeared to converge, while for the less frequent analysts, there was more variability. A select few of these results can be see in 14, note that the lower two figures, have similar distributions across the tags, while the upper two figures appear more varied.

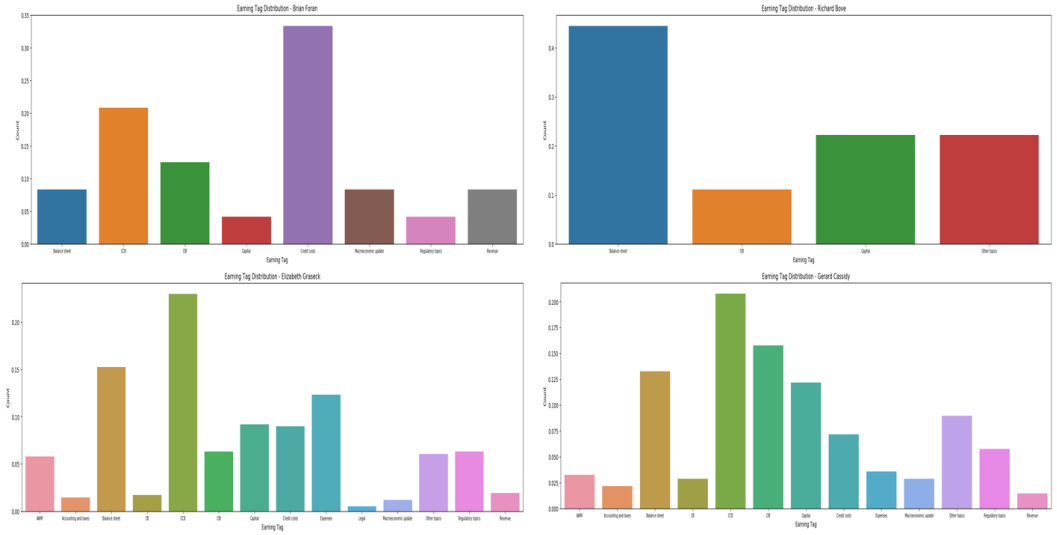

Figure 14: Tag Distributions By Analyst

Additionally, we generated the TF-IDF plots for each of the analysts, expecting to find syntactical differences and possibly additional feature information. Here we noted, a few interesting tidbits as well alongside a lot of seemingly less significant information. We noted that some of the highest scoring words included numbers, such as one billion, names of company participants and manneristic terms like thank you and sorry. These results informed our later modeling decisions such as the dimension reduction and the indexing among data for question generation. 15 has a sample TF-IDF plot for one of the more prevalent analysts in the data-set.

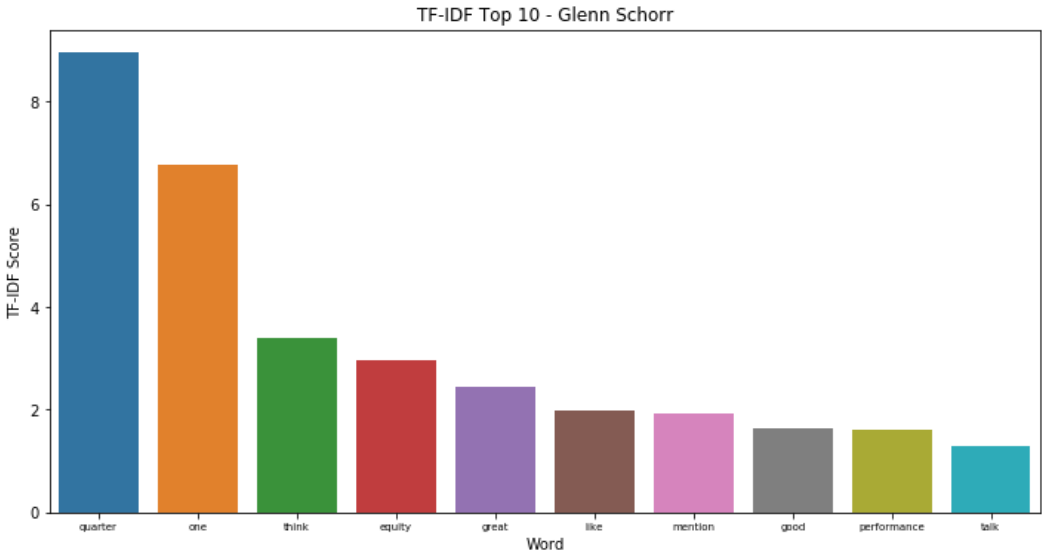

Figure 15: Top-10 Words Per Analyst

We also note that the distribution of tags across quarters was varied heavily. Again, this was to be expected, as there are well-known events that occur in a timely fashion each year. For instance, more than 20% of questions were tagged as Balance Sheet in Q3, while for all other quarters it appeared to

be only 15% of questions. Similarly, Q1 had many more Accounting and Taxes questions than the other quarters. As we did earlier, we also generated TF-IDF plots per quarter, one of which can be seen in 16. In 16, we see that among the most significant terms is the upcoming quarter, Q4, as well as "capital" one of the fourteen question tags.

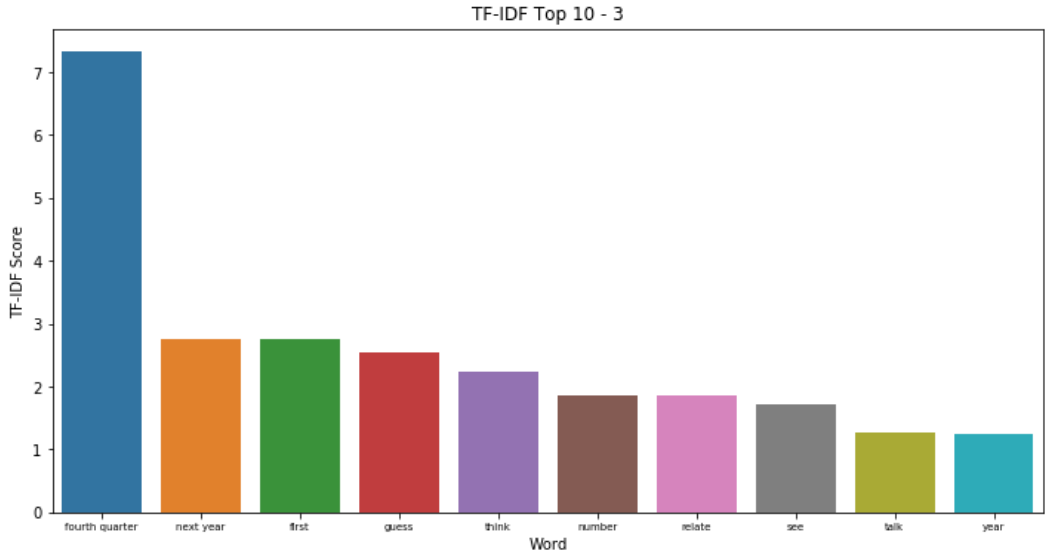

Figure 16: Top-10 Words Per Q3

Lastly, we explored the TF-IDF scores of the tags themselves. Like the previous TF-IDF distributions, we noted both some interesting terms as well as some expected terms. For instance, the highest TF-IDF score for *Macroeconomic Update* was "Mexico", while "trading" was the second highest score for *Regulatory Topics*. We have included the TF-IDF plot for *Regulatory Topics* for posterity sake, in 17.

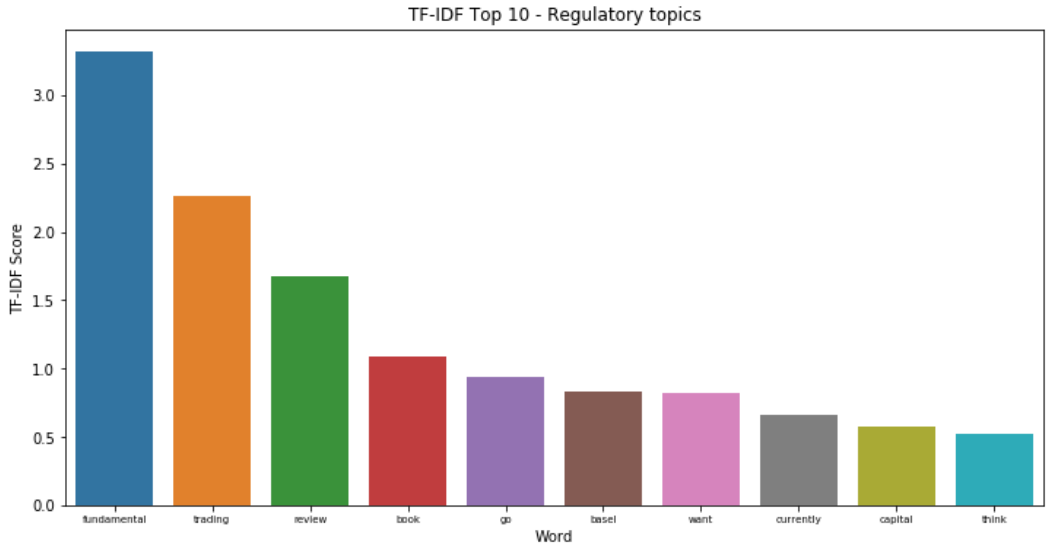

Figure 17: Top-10 Words for Regulatory Topics

4.2 Tag Model

As we mentioned earlier, we tested three successive generations of tag models. Our initial model, V1, was the worst performing of the three. The data format we chose, required a secondary model to predict the number of questions per analyst per call. With the secondary regression model, we had to generate the sequence of events prior to the call before we could model each question. Additionally, since we had excluded all question data, nor had we designed any features for the model, the model has extremely poor performance. The model had a Macro-Average AUC of 21% and an accuracy of 25%.

While V2, greatly outperformed the initial model, it too suffered from dimensionality problems. As we had one-hot encoded the analysts, the tags, the companies and most of the other features, we had over one-hundred features, many of whom, were overly sparse. To combat the dimensionality issues, we tested the use of both principals component analysis and nonnegative matrix factorization prior to modeling. Interestingly, the dimensionality reduction did not increase the model performance, though it did improve the speed of model training. Additionally, the historic tag percentages, the generated features we had built to better summarize the analysts, were not overly good predictors nor did they vastly improve the model. With some cross-validated grid searching, we were able to settle on a model AUC to 68% and our accuracy to 72%.

For V3, we tested a new method for respresenting the one-hot encoding features. As mentioned in 3.3, we used nonnegative matrix factorization to vectorize the analysts, tags, companies and quarters. To choose the proper dimensionality of the vector, we used the Elbow Method on the cophenetic scores over the topics to choose the proper vector size. For instance, 18 shows the cophenetic scores over the dimensionality of the analysts and our choice of 4, due to the large drop-off in scores for five topics.

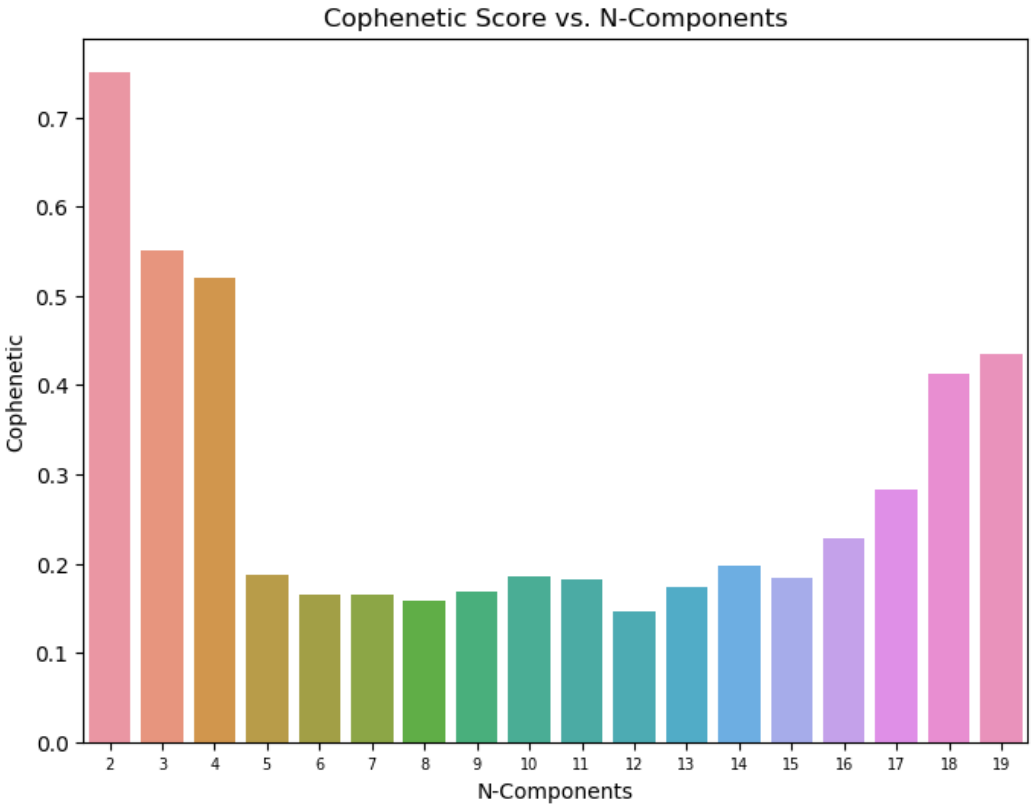

Figure 18: Cophentic Scores for Analyst Dimension Reduction

Similarly, we ran identical tests on the tags, companies and quarters, choosing vectors of 6, 4 and 4 respectively. By vectorizing those features, we were able to reduce the dimensionality of our data-set from over one-hundred features to less than thirty. More importantly, we were able to remove much of the sparsity, as we normalized the outputs of the dimension reduction. After a proper grid search over a Gradient Boosted Classifier, our best estimator across all models, we had an AUC of 79% and an accuracy of 85%. Those values aside, they represent the performance across the "menu" approach and not over a true sequence. We used Platt's Method to calibrate the model, before using our greedy naive method for sequence generation.

Lastly, since we were unsatisfied with our question generation, we used the affinity matrix generation described in 3.3 and visualized in 4. While there was not a determined performance metric for checking the question sampling predictions, our mentors approved of the test results and methodology. Below in 19 we visualize some of the results of our question sampling method.

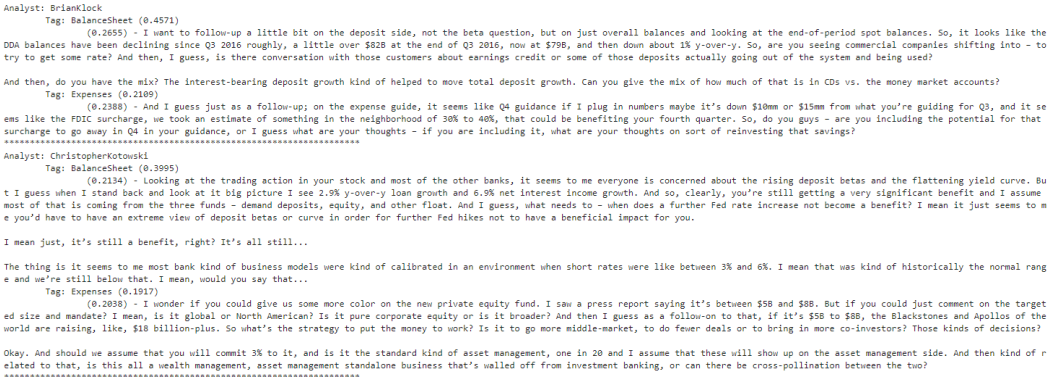

Figure 19: Question Sample from Question Sampling Method

4.3 Question Generation

4.3.1 Word-Level Modeling

The following example 20 shows sample output generated by the baseline word-level model, without pre-trained word embeddings. The first line is the initial seed text (the starting words of the sequence), the second line is the prediction generated by the model. As can be seen, the model can capture short sensical sequences like "are you seeing any evidence...", but struggles with the longer sequences and the long range dependencies of the sequence itself.

```
my question i just wanted to follow up so last year it looks like you grew your average earning assets

in the commercial customers are you seeing any evidence of the fed moves how to the size of that portfolio and also t
he margin carry return and how you think about the fed funds sold of uncertainty in the past few years how are you th
inking about the fed moves
```

Figure 20: Baseline Word-Level Model Output

In the examples below 21 and 22, we see that the model is able to use the financial terminology, though the sentence does not have much sensical meaning.

```
# print(seed_text + '\n')
seed_text = 'balance sheet'
# generate new text
generated = generate_seq(model, seq_length, seed_text, 100)
print(generated)
```

```
and the net interest margin dollars in the back half of the year ?
```

Figure 21: Baseline Word-Level Model Output

```
seed_text = 'can you'
# generate new text
generated = generate_seq(model, seq_length, seed_text, 50)
print(generated)
```

```
reduce your asset yields overall , donâ??t the trade-off on rate sensitivity from here ? i mean 22
% points their loans have growing nicely . it's junk type . or has that impacted your deposit pene
tration that you have in terms of c&i , we get a subordinated debt securities
```

Figure 22: Baseline Word-Level Model Output

We then tested the model with pre-trained word embeddings. The resulting model was a disappointment. Even with additional layers and bidirectional neural networks, the model was unable to learn form the training data. Ultimately the best performing word-level model, baseline included, had a validation accuracy of 30% and validation loss of 5.6. We surmise that much of the score is directly related to our small data-set, and the failure to learn with such little data. Additionally, we believe the pre-trained word embeddings performed poorly as they are trained on a general data-set, the Wikipedia corpus, while our use-case is very specific.

4.3.2 Character-Level Modeling

The character-level models had slightly better performance than the word-level models, given a shorter seed text. Using our best character-level model, our validation loss was 0.981 and had an accuracy of 70%. An example of the character-level models can be seen in 23.

```
[Do you think] you can give us an update on the card balance sheet side , the core loan growth has been seeing redemp
tion and that still be ?...

[Do you think] you guys are thinking about the repo more side , what are you thinking about the two years ?...

[Do you think] that positioned on a big better than standards on mortgage balance sheet in the same target share in c
ommercial side of the balance sheet growth , but I guess what would you expect that . And I fund you accretion that y
ou0?...

[Do you think] that all the reason that we should help us how you have got a core loan growth . But I guess the quest
ion that was some of the deposit growth , but it sounds like you talked about the structure of the year . I think you
mentioned the loan growth ?...

[Do you think] that in terms of funding business that you did that trade to the commercial side , which is the core N
II or is there a second question on the LCR , if you can give us an update on your commercial loans portfolio . I thi
nk you mentioned that you can do in the balance sheet growth in regards to the loan growth on the flattening of the c
ommercial real estate loan to deposit growth , if there are an an additional banks in the mortgage balances when you
have been doing from what you have...
```

Figure 23: Character-Level Model Output

Another example, using the seed text"that that," can be seen in 24. As can be seen in the example, the character-level models can generate financial terms like "NII" and form relatively meaningful sentences. That being said, the model is limited in performance and extremely sensitive to the given seed text. Additionally, without the pre-trained word embeddings, the model is not about to generalize to other tasks.

```
[think about ]that demands , but are you seeing some of the stable strategy but it actually how you think about the r
e-pricing that you0?...

[think about ]the balance sheet and the net interest income in terms of demand between strategy and but I guess I gue
ss the deposit growth in the marks . I guess what were you down a little bit about the loan growth in the portfolio ,
and I think you said that that many yields were up a little bit more regards to the core NII ?...

[think about ]where you can give us an update on the commercial loan growth , there was a strength of the core loan g
rowth with the balance sheet to change at this point of this stage ?...

[think about ]the balance sheet and the industry was a little bit about this year . And then if you can talk a little
bit about the balance sheet . I guess that staying a little bit more funding side , how much of that staying on the c
ore loan growth ?...

[think about ]the balance sheet . And then just a factor question . You mentioned that you can walk a little bit more
color on the market share accounting on the debt interest rates , or are you seeing any consumer side , but what are
you seeing the first quarter . I think you said it would be a little bit more color on the deposits are better than y
our assets and I guess it was a core more mortgage balance sheet pressure in the margin and the securities portfolio
in terms of the balance sheet gro...
```

Figure 24: Character-Level Model Output

5 Conclusions

To best prepare JP Morgan Chase's Investor Relations team, we have developed a series of machine learning models to predict and generate question tags and questions for upcoming earnings calls and other events. To do so, we investigated the historic questions of analysts and noted how their tag distributions converged, while their syntax did not. We developed a series of machine learning models to predict the suspected questions of the analysts in a given call, using a "menu" based approach. Lastly, we developed three question generation techniques; one that statistically samples form existing questions, a word neural network and a character-level neural network.

6 Future Work

6.1 Generalizability

While we applied our methodology to a very specific use-case, earnings calls and other financial events, there are an assortment of professions in which better preparing for questions could be used. For instance, in the medical profession, doctors and nurses often field similar questions across many patients. Given that patients can have similar diseases and therefore similar questions, medical professionals can prepare questions after diagnoses. In other words, since patients will have similar questions for their similar diagnoses, doctors can better prepare beforehand, by checking the output of a model like ours. Additionally, medical professionals will be able to pinpoint which questions which patients will ask, and not prepare the same questions for each patient. As another example, professional educators undoubtedly face torrents of questions during and at the conclusion of each session. If they, the professionals, could accurately predict what questions would be asked prior to a lesson, they could better prepare their lessons. Additionally, like the financial field, both of these fields offer large sample spaces of analysts in the form of doctors and students and specific tags like "diabetes diagnoses" or the "quadratic equation."

6.2 Question Generation

We believe that one of the major issues with our question generation models is the limited corpus we had access to. Therefore, we propose the following theories to further explore our work.

Firstly, we propose better data sequencing. By splitting the larger, multi-part questions into a series of shorter questions, we can undoubtedly improve the accuracy of the model. Additionally, we can design a secondary model that can combine two shorter questions based on their compatibility. Using a methodology similar to Bowman et al. (2015), we believe we can combine these less complex questions and transform them into the larger multi-part questions common in this field.

Secondly, as we used a greedy search algorithm to define the generated sequences, we believe that an implementation of beam search could vastly improve the results. Additionally, we could experiment

with additional metrics that penalize sequences that do not use financial terminology or sequences that are too short.

Lastly, given the limited data-set, it is possible that some of the older methods, like context planning, sentence planning and surface realization could be used to improve model performance. Similar work have been done in Dale et al. (2003).

7 Acknowledgements

We would like to thank Bruno Goncalves, Santiago Salazar, Fernando Cela Diaz, and Boyu Wu at JP Morgan Chase for serving as project mentors. We would also like to thank Professor Richard Bonneau for his guidance.

References

Samuel R. Bowman, Gabor Angeli, Christopher Potts, and Christopher D. Manning. 2015. A large annotated corpus for learning natural language inference. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics.

Kyunghyun Cho. 2016. Natural language understanding with distributed representation.

Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *CoRR*, abs/1406.1078.

Robert Dale, Sabine Geldof, and Jean-Philippe Prost. 2003. Coral: Using natural language generation for navigational assistance. In *Proceedings of the 26th Australasian Computer Science Conference*, pages 35–44, Australia. Australian Computer Society.

Hongyuan Mei, Mohit Bansal, and Matthew R. Walter. 2015. What to talk about and how? selective generation using lstms with coarse-to-fine alignment. *CoRR*, abs/1509.00838.

Tomas Mikolov, Edouard Grave, Piotr Bojanowski, Christian Puhrsch, and Armand Joulin. 2018. Advances in pre-training distributed word representations. In *Proceedings of the International Conference on Language Resources and Evaluation (LREC 2018)*.

Marinka Žitnik and Blaž Zupan. 2012. Nimfa: A python library for nonnegative matrix factorization. *J. Mach. Learn. Res.*, 13:849–853.