NLP Prospectus Project

seh*

February 2023

1 Introduction and summary

In this project, we investigate how to utilize the information "hidden" in prospectuses. We do so by using modern natural language processing techniques on a number of datasets we gather from various sources. Our results show both indications the predictive power of such techniques, as well as the potential usefulness of NLP-based tools.
We first describe our dataset(s), some of which we generate from internal Capital Four documents, and some of which we scrape from other providers (namely, ESMA and SEC). We primarily focus our efforts on 10-K filings which we can link to a USHY ISIN. We successfully scrape and link around 1100 such filings. To process these datasets, we develop a module of functionality to parse and analyze text documents, all of which works generically for any text. We use this functionality to generate a selection of textual features, on top of which we aim to model downgrade probability for bonds. To this end, we frame the problem as a case of survival analysis, and thus use Cox proportional hazards regression to estimate the downgrade risk.

As a baseline model, we use just two covariates: the relative spread at issue for the given bond, and the 1-year change in average spread for the index. With this baseline, we aim to summarize the information about the bond available in the market at issue. We then add various NLP-based features to this baseline. We broadly investigate five different sets of features: descriptive statistics, textual features, topic modelling, word embeddings, and GPT-based features. We summarize our findings below:

- From the descriptive statistics, the length of the *Legal Proceedings* section turns out

*Reachable at `sebastian.holmby@gmail.com`

to be a significant predictor, as well as the number of times the word "leverage" is mentioned.

- In terms of textual features, the best predictor found was the sentiment of the *Business Summary* section. Here, a more negative tone proved to be correlated with higher downgrade risk, so the 'straightforward' hypothesis holds true.
- Next, we use the recently released GPT word embeddings to encode the summary of each document, yielding a vector representation of the section. We then use this representation as features, and they prove to be useful predictors for downgrade risk - i.e., they improve upon the baseline outlined earlier. In general, however, there are indications that this approach would benefit massively from more documents/bonds.
- We also use GPT to synthesize and summarize the risks outlined in the *Risk Factors* section. In particular we query GPT to find risks related to the parameters used in the C4 fundamental scoring. We then use topic modelling to find common topics across the corpus of documents, meaning we can find out which documents discuss similar risks. The loading on each topic is then used as input features, and this proves to be our most successful model - as well as probably being the one with the most potential for improvements. Finally, we also use topic modelling by itself, without the GPT summarization, but this fails to outperform our baseline model.

In conclusion, many of the features manage to improve upon the market baseline - which is in itself a valid predictor of downgrade risk. However, the NLP features do not perform as well as this baseline on their own. Thus, we conclude that the analysis holds a lot of valuable information, but one should think of it as additional information on top of the fundamental information available about the bond. We explain every part of the project in further detail in the rest of this report.

Contents

2 High yield data

2.1 EUR - ESMA

Since Nov 2020 all prospectuses for European bonds must be submitted to a Prospectus Register[1] under ESMA (European Securities and Market Authorities). Thus, we can go through the European HY universe, get the ISINs, and look up prospectuses for each bond issued since 2020. This yields around 115 prospectuses - and thus it seems not all bonds are stored there, despite the previously mentioned requirement. Furthermore, there seems to be a bias towards banks in the dataset. The script to scrape these prospectuses - which is mostly an exercise in scraping, and takes a while to run (overnight) - is found at `scrape_esma.py`.

2.2 US - EDGAR (SEC)

All publicly listed US companies' filings to the SEC are publicly available through `EDGAR`[2], the SEC's search engine. Using this service requires a bit of know-how, unfortunately. In our case, there are a number of filings that may (when combined) constitute a 'prospectus'. In particular, these are typically made through a 424B2, 424B3, or 424B5 filing. The latter of these three seems to be most consistent, but which filing is made depends on the company's previous filings and other legal positions. Furthermore, some of these filings may also be made in the context of issuing shares of common stock. As such, the filtering becomes fairly important in ensuring one scrapes only the correct prospectuses.

Perhaps the most important note here is that in public US prospectuses, the *Risk Factors* section typically refers back to the most recent 10-K filing[3] for an overview of the risks involved in investing in the company. As such, much of the content we aim to extract from prospectuses are actually contained in 10-K filings in the case of US bonds. 10-K filings are required to follow a very specific format, including a pre-set selection of sections. This allows us to very easily extract only the desired sections of text from such filings. In particular, we can do so *extremely* easily by using the `sec_api`[4] package available in Python for a small fee for full API access.

[1] https://registers.esma.europa.eu/publication/searchRegister?core=esma_registers_priii_securities

[2] https://www.sec.gov/edgar/search/

[3] The 10-K is an annual filing every publicly listed company must do, which describes the company's current financials, risks, and business outlook

[4] https://sec-api.io/

Leading on from the aforementioned, the primary dataset used in this project is *solely* the 10-K filings associated with companies that have issued a bond of interest. We iterate over the US HY index, returning the latest available 10-K at the issuance of the bond, and use this as our 'prospectus'. This amounts to around 1500 10-K filings, with a nice dispersion in time throughout 2002-2023. The script that scrapes these filings and saves them locally in JSON format can be found at `scrape_edgar.py`.

2.3 Holdings

We also have a number of prospectuses available in holdings, through Sharepoint, but given the availability of public prospectuses as mentioned above, we focus our efforts in there. We do so in particular because the task of matching prospectus to ISIN is much easier for the aforementioned datasets.

3 Leveraged loans data

We can also consider docs related to leveraged loans (LLs). Here, there does not exist a classic prospectus as the ones printed for bonds. Rather, a typical LL issue includes a *Senior Facilities Agreement* (SFA) and a *Confidential Information Memorandum* (CIM). The former is written in very formal and lengthy legal language, and the format of SFAs is generally very similar across LLs - in particular, SFAs written by the same sponsors are often close to identical in almost all language in the documents. In contrast, CIMs are more akin to a pitch-deck, attempting to sell the deal through more colorful language, and in a less rigid structure than bond prospectuses. As such, both of these document types have some issues if they are to be used in our modelling; SFAs are extremely similar in their language, and the language is very formal and neutral, meaning we can extract almost no signal from analyzing them. On the other hand, CIMs vary greatly from cases to case. One CIM may be a Word-like document, text-heavy and descriptive, while another may be a PowerPoint presentation consisting almost exclusively of images or tables. Furthermore, there are no requirements as to what to include in a CIM, meaning comparability between these documents becomes very difficult. We still do some preliminary investigations into LLs, which yield some interesting findings, but overall the task becomes much harder here compared to the bond space.

3.1 Holdings

To get a data set of LL docs, we can scrape the C4 sharepoint sites. We do so by finding all issuers with a LL deal attached to them, and returning all documents tagged as CIMs. This approach is not perfect - for example, an issuer may also have a bond with associated prospectus, and this prospectus is often tagged in the same category. Issuers may also have multiple docs concerning different deals, and we have no automatic way of distinguishing which document is associated with which deal. As such, we manually verify each document - this is fortunately not too cumbersome. From this scraping, we retrieve a total of 88 CIMs. The script is available at `ll_holdings_scrape.R`.

4 Parsing

As a large contribution from this project, we build a collection of functionality to help support parsing and analyzing text documents, both individually and as corpora. This functionality is primarily contained in the `c4_parser.py` file, split into `PdfParser`, `TextAnalyzer`, and `TextParser` classes. The classes work as follows:

- The `PdfParser` class is built to extract content from prospectus or prospectus-like document (SFAs, for example). Given a PDF file, the class attempt to find a table of contents, and use this information to determine where each section in the document starts and ends. Through the methods of the class we can then extract sections by their title (we allow the title to differ a little from the queried title if an exact match cannot be found). We also extract the full text, and specifically the Risk Factors section (if such a section exists), since it is of specific interest to us.
- Given an extracted text from the above, we can then use the `TextParser` class to process the text. This is a simple class, which mostly allows us to strip a given text of unnecessary punctuation and stopwords, before tokenizing the text (breaking it into words). The primary motivation is thus to ensure that this preprocessing of the text in the same way for all texts. Finally, the class includes the method `extract_fls`, which uses FinBert[5] to extract forward-looking statements from a given string (more on this later).
- Finally, the `TextAnalyzer` class is used to computed the textual features which we will see introduced below. Like the `TextParser`, all methods available through this class works for any generic text, and is thus easily portable to other NLP projects.

[5] https://huggingface.co/yiyanghkust/finbert-fls

Aside from this, there is also a selection of other functionality available in terms of visualizing, reporting and interpreting documents. In particular, the `write_report` function is written as screener/processor of a prospectus with a linked ISIN. Given such a pair, the function will extract all text, sections and forward-looking statements from the document, before computing the features outlined later in this report. Furthermore, it also summarizes key points from the document, such as what the main risks are. This can be done for any given section using the `PdfParser` module. Finally, we combine this with performance data into a single one-page report that we can generate in seconds. An example is available in `NLP/report_XS2363235107.html`

5 Textual and descriptive features

Next, we consider some textual features of a prospectus which *may* hold valuable information for the purpose of predicting performance of an asset. For the purpose of inspiration as to what textual features may be useful, we make use of a number of references, amongst those the ones outlined in the bibliography ([1] [4] [3] [2] [6]).

5.1 Descriptive statistics

First, we consider perhaps the simplest features of a prospectus, the descriptive statistics. By this we mean the non-textual features of a prospectus; for example, we include the length of the entire prospectus, and the length of the *Risk Factors* section (both in pages/words/characters). We also include *keyword counts*, that is, how many times does (variations of) a given word appear in a document. For this, we test the set of words $\{'leverage', 'adjustment', 'adjusted', 'dividend'\}$, and calculate both the absolute number of times each word is mentioned, as well as how big a proportion of the total set of words is the given word.

5.2 Textual features

Next, we consider some textual features. All of these features are computable through the `TextAnalyzer` class in the `c4_parser` module.

5.2.1 Sentiment analysis

First and foremost, we want to gauge the *sentiment* of the analyzed texts. This can be detected through the tone of each sentence, one at a time. To compute the sentiment of a given

sentence, we use the `FinBERT` language model[6], available in Python through `huggingface`.[7] Confusingly, there are two models with the name FinBERT - we refer here to the one published in 2022 by Huang, Allen H., Hui Wang, and Yi Yang. This model is a BERT model pre-trained on financial communication text, which has afterwards been trained to gauge sentiment by training on 10000 manually annotated sentences. The outputted label for a given sentence is a label $\hat{y}$ which is one of positive, negative and neutral, and a "confidence" score of this label, $c_{\hat{y}}$, which ranges from 0 to 1.

Given a text with N sentences, we then compute the positive fraction of the text as follows:

$$PosFrac = \frac{\sum_{i=1}^{N} [\hat{y} = pos] \cdot c_{\hat{y}}}{\sum_{i=1}^{N} c_{\hat{y}}}$$

with corresponding scores for negative and neutral fractions. As such, the fractions are weighted by the model's confidence in the given label. The sentiment features we include are thus these three fractions, as well as the ratio between the positive and negative ratio $\left(\frac{PosFrac}{NegFrac}\right)$.

An alternative to this (which we have previously employed in another project) would be to summarize the sentiment in a single number.

$$Score = \frac{\sum_{i=1}^{N} ([\hat{y} = pos] - [\hat{y} = neg]) \cdot c_{\hat{y}}}{\sum_{i=1}^{N} [\hat{y} = pos \vee \hat{y} = neg]}$$

This scoring thus completely ignores neutral sentences. During other experiments including the neutral sentences in the normalization proved to put unnecessary emphasis on shorter sentences, even though they weren't generally more positive/negative. Thus it is not part of the normalization. The score always ranges between -1 and 1, with a score of 1 (-1) being a completely positive (negative) document with full confidence in all sentences. A sentimentally equally weighted document should thus end up around 0 in score. We do not include this score in this project (for now), but it is a worthwhile consideration if one desires to summarize sentiments in a single number.

As an example, the sentence

"The Group may be exposed to increased competition in the electricity and gas markets in

[6] https://finbert.ai/sentiment

[7] https://huggingface.co/yiyanghkust/finbert-tone

Bulgaria and abroad, including as a result of the ongoing liberalisation of the Bulgarian electricity sector."

is predicted to be negative with a confidence of 0.905, neutral with a confidence of 0.095, and positive with a confidence of zero.

There is one, final, quirk to mention about sentiment analysis from the perspective of credit. Since sentiment analysis models are typically trained on a not-that-large number of *manually* annotated sentences, some of the nuances of credit may be wrongly interpretted by the model. As an example, leverage going down is generally a good thing, but the wording - i.e., something going *down* - is typically indicative of a negative thing. As such, sentiment analysis models will typically encounter some issues along these lines.

5.2.2 Subjectivity

The *subjectivity* score of a text refers to the tone the text is written in. If the tone is more formal and objective, the subjectivity score will be lower, whereas a less formal, subjectively written text scores higher. We compute subjectivity scores using the `TextBlob` library available in Python - unfortunately, there is very little documentation in terms of how this subjectivity score is calculated. The outputted score ranges from 0 to 1, where a higher score indicates a more subjective sentence.

As an example, the text

"We think 2023 is going to be our best year yet."

has a subjectivity score of 1, while the text

"Changes in the market price of emission allowances could negatively impact the Group's activities."

has a subjectivity score of 0.4.

5.2.3 Readability

Another well-known feature to describe text documents is the *readability* of the text. Many measures exist to describe such readability; amongst those, we mention the Gunning fog index, Flesch–Kincaid tests, and several AI-powered measures.[8]

[8]For an overview, see https://en.wikipedia.org/wiki/Readability

We test several of the mentioned features, and conclude that their correlation is extremely high. As such, we decide that just a single of these measures is adequate to describe the readability of texts. For this purpose, we pick the Gunning fog index. This index outputs the readability of a text as the estimated grade level necessary to understand the text; for example, a Gunning fog index of 10 corresponds to a high school sophomore, while a score of 17 corresponds to the level of a college graduate. The score is calculated as

$$0.4\left[\left(\frac{\text{words}}{\text{sentences}}\right)+100\left(\frac{\text{complex words}}{\text{words}}\right)\right]$$

where *complex words* are defined as words consisting of more than three syllables. As some may observe, the score is very similar in idea to *lixtal*, taught in Danish schools.

5.2.4 Yule's index of diversity

As a final textual feature, we consider Yule's *index of diversity* - most commonly just referred to as Yule's *I*. Yule's I is a measure of the diversity/richness of a given vocabulary, and is calculated as

$$I=\frac{N_{doc}\cdot N_{doc}}{N_{freq}-N_{doc}}$$

where

$$N_{doc}=\text{number of words in the document}$$
$$N_{freq}=\text{sum of the squared frequencies of each word in the document}$$

We also calculate Yule's K, which is simply $^{c}/_{I}$, where c is some constant, typically 10^4.

5.3 Topic modelling

Moving away from the 'simpler' textual features, we also perform *topic modelling* on the prospectuses. Informally put, topic modelling algorithms seek to find common topics across different texts in a corpus of texts. To find these topics, we choose to use the Latent-Dirichlet allocation model (LDA).[9] LDA is an unsupervised model, i.e., we have no input on the topics found. The method groups words into topics by their co-occurences in the given corpus, and then computes how much each document in the corpus loads on each topic.

The hypothesis behind using topic modelling for our case is that it may, for example, be able to group similar risks into topics together. Given a set of topics representing such

[9]We use the Python implementation available through `gensim`, which can be found at https://radimrehurek.com/gensim/models/ldamodel.html

groups of risks, a document's loading on a given topics would then give us an idea of how exposed the given bond is to the risk. However, it is not certain that we have enough data for this approach to be successful - namely, we have some very long texts, but relatively few, and as such, it may be difficult to generalize across the documents. A common technique [4] is to filter out topics with below-average loading, so we test model both with and without this modification.

5.4 Word embeddings

A *word embedding* simply refers to a vector representation of a given word, and is at the very core of natural language processing. Traditionally, models that convert words to vector representations were either dictionary-based or trained on the to-be-embedded text/corpus. Perhaps the most well-known example of this is the Word2vec algorithm, which uses a neural network to embed a given corpus of text, with the outputted vector having a user-specified dimensionality. However, recent developments such as BERT (by Google) and GPT (by OpenAI) have significantly improved the representation quality of word embeddings. These models are trained on huge amounts of text documents, and as such they excel at constructing word embeddings - in particular, they are extremely proficient at modifying the representation according to the context the word appears in. In a satisfactory word embedding, similar words should be close (their Euclidean distance should be small), while less similar words should be further apart. But newer, more sophisticated language models can carry a lot more information, including short- and long-term context, grammatical tenses, and other correlations that are difficult for a human to spot. As a result, these embeddings are typically of a much higher dimension - for example, in GPT-4, words are represented by 1536-dimensional embeddings. For a classic example of the intuition of word embeddings, consider the three cases shown in Figure 1.

We first compute the embeddings of each sentence, and then average these embeddings to get a representation for the entire document. Alternatively, one could attempt to embed the entire document as one - however, many of our documents are longer than the maximum length possible in the models, and as such we average each sentence (or paragraph, as we will explain later). In theory, embedding the entire document should better capture the long-term context of documents, but alas, it is infeasible with current methods.

Given a d-dimensional vector representation $\mathbf{v}_i$ of document i, we can use $\mathbf{v}_i$ as the 'features' of the document. If one thinks of each dimension as representing one aspect/feature of a given document, then these high-dimensional embeddings should be able to accurately

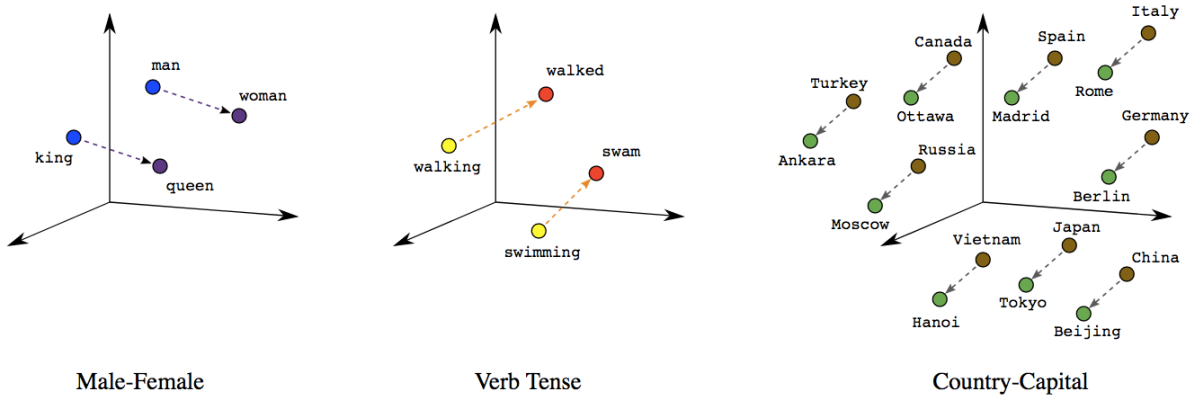

Figure 1: A classic example of word embeddings. Here, 3-dimensional embeddings are shown for a number of cases, and we can see the similarities between words, tenses, and country/capital pairs. From developers.google.com.

describe a text through these numerical representations. As such, we can attempt to model downgrade risk based on this representation - as for example was done for municipality bonds [3]. For each section (and full text), we compute the following two representations, and test them as features:

- We compute a 20-dimensional `Doc2vec` representation, using a window size of 20 to allow some context, and requiring each word to occur at least 10 times. Doc2vec is very similar to Word2vec, but optimized for documents instead of words. This is done through the `gensim` library in Python, and serves as a simple (and quick) baseline for word embedding approaches.
- Next, we embed each text through GPT-4's embedding API - more specifically using the `text-embedding-ada-002` model.[10] Since each input to this model can be at most 8000 tokens (words + spaces etc.), we split the text into chunks of this size, and average these embeddings. This results in a 1536-dimensional embedding for each document. If we were to use this embedding directly as features, our model would be *very* prone to overfitting, with more covariates than subjects. As a remedy, we perform a principal components analysis to reduce the representation to 10 dimensions. This means 80% of the variance remains explained, but we are much less prone to overfitting, leading to a more robust model.

[10] https://openai.com/blog/new-and-improved-embedding-model

For both of these cases, we will test performance using only the embeddings, as well as adjusting for market-available information about the bond. This will be described further in later sections, when we specify the architecture of each model.

5.5 Forward-looking statements

When we consider the predictive power of a prospectus (as in this project), we are trying to model how the language in the prospectus impacts the performance of the asset in the future. Given this, it seems intuitive to focus on the *forward-looking statemts* (FLS), that is, statements in the prospectus which predicts future business conditions, risk factors, performance, etc. Previous research exists on the topic of extracting FLS from prospectuses, and then extracting textual features *only* from the set of FLS [4]. In this mentioned study, the prospectuses are from IPOs, which bears some similarity to the issuance of a new bond/loan.

When extracting FLS, we are working under the hypothesis that the majority of the content of interest lies in these statements, and the rest of the content is less important in terms of predictive power. Under this hypothesis, we reduce the amount of noise by removing non-forward-looking sentences. In terms of algorithms, there has been a thread of research on how extract FLS from texts (see, e.g., the aforementioned paper [4]). For our purpose, we aim for an FLS-extractor that is good for texts in the financial domain. Luckily, this is available through the `FinBERT` language model[11], where the language model has been trained on a huge corpus of financial domain text, and the FLS extractor has been further trained on 3,500 manually annotated sentences from Management Discussion and Analysis section of annual reports of Russell 3000 firms, *specifically* for the task of FLS extraction. The model classifies a given sentences into one of three categories; *Specific FLS*, *Non-specific FLS* or *Not FLS*. Given a sentence, the model then estimates the probability of the sentence belonging to each of these three categories (summing to 1). For each section (and the entire text) of a prospectus/filing, we extract any sentences that are predicted to be one of the two former labels. We then compute all of the *other* aforementioned features for these FLS as well.

5.6 GPT synthesizing

As a final textual 'feature', we attempt to summarize specific parts of a given text using ChatGPT. In particular, we use the parameters from the Capital Four fundamental scoring

[11] https://finbert.ai/fls

project[12] as guideline for which parts of a bond are important when judging the fundamentals of a company. We then feed in the prospectus to GPT, prompting about a given parameter - for example, we could use the query

"How is the company described in the below text positioned in the market, in comparison to its competitors?

to gauge the competitive positioning of the company. In theory, this should helps us summarize the text while retaining only the important parts of it.

The outputted answer is heavily dependent on the wording of the query, and thus it may take some more experimentation to reach optimal queries for each parameter. As a proof of concept, we prompt GPT to outline the technology risks related to the business in the Risk Factors section. This gives us a fairly short, synthetic description of the technology risks for each bond, which ideally should contain less noise, as we now aim to include only significant risk factors. On top of these summaries, we perform topic modelling, allowing us to both visualize topics across the corpus, but also use topic loadings for each document as features, similar to what was described in Section 5.3.

[12] Various documentation available at `F:/Research/_C4 General Research/C4 Scoring/2020 project/`

6 Statistics

We can then consider the distributions of these features across the corpora. In Figures 2-6, we show distributions of selected features for various sections of 10-K documents. The same statistics have been produced for both European high yield and leveraged loans, but we omit them here due to lack of space. Of particular note, it was concluded that leveraged loan information memorandum were *significantly* more positive in their tone, almost the opposite of the very negative bond prospectus language. The reasoning for this probably lies in the fact that a LL CIM is closer to a pitch deck, and needs not include any potential risks or legal language.

6.1 US HY 10-K statistics

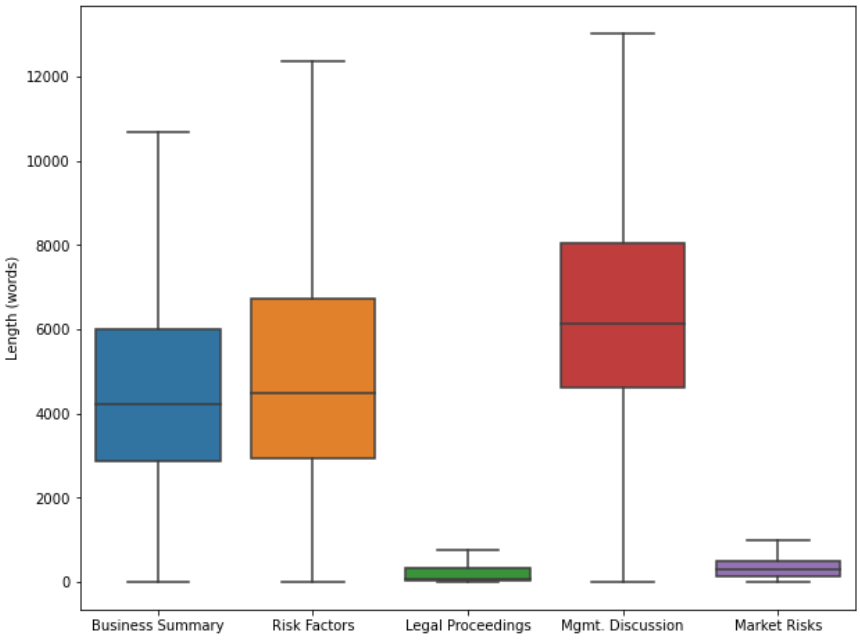

Figure 2: Length of sections across 10-K filings.

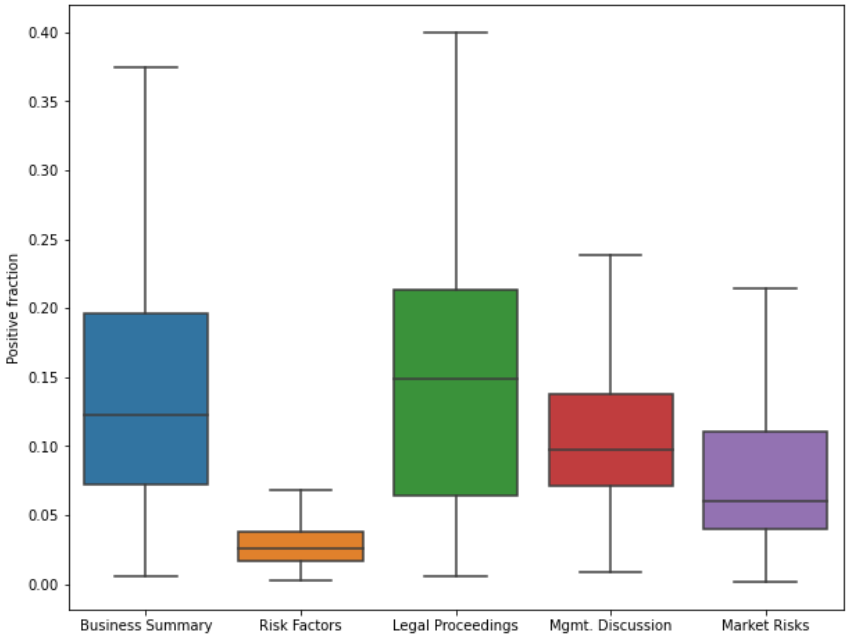

Figure 3: Fraction of positive sentences across sections in 10-K filings.

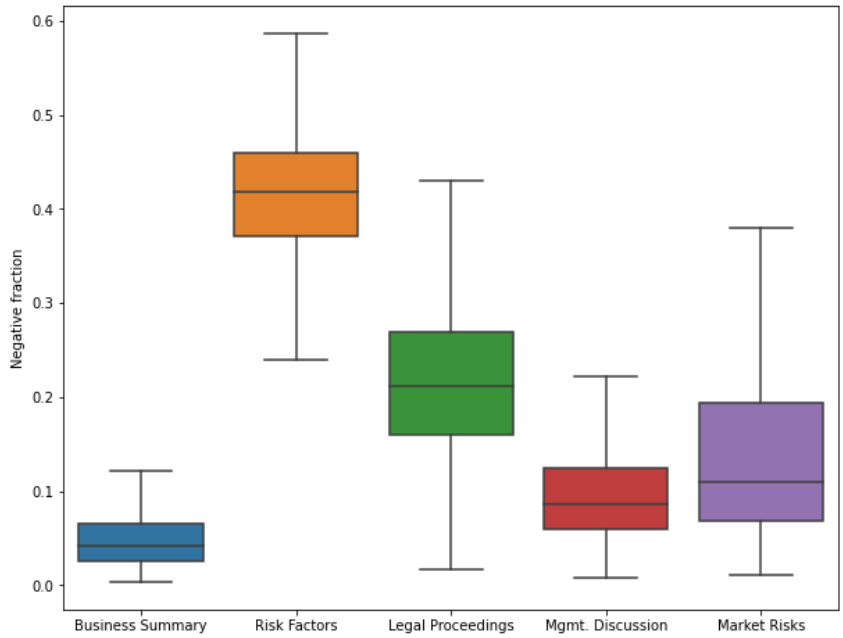

Figure 4: Fraction of negative sentences across sections in 10-K filings.

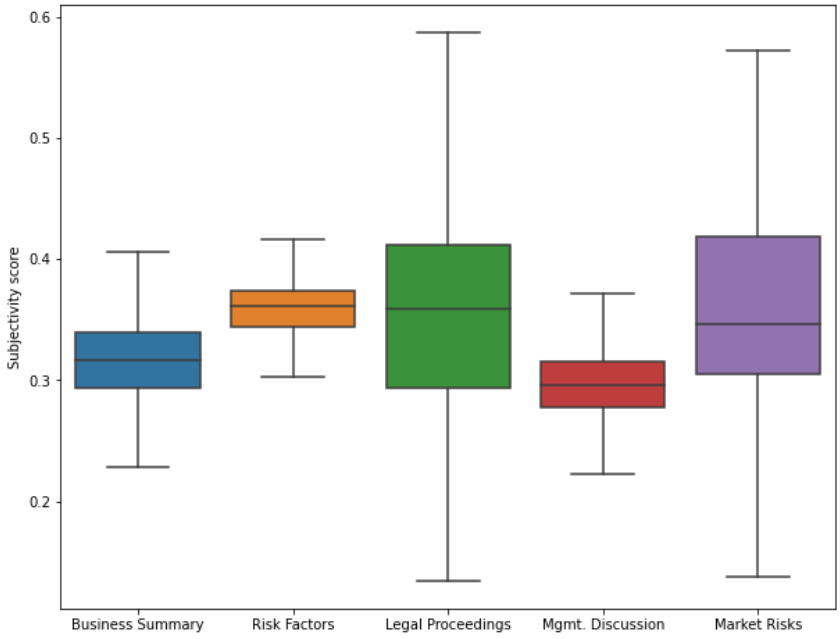

Figure 5: Subjectivity scores across sections in 10-K filings.

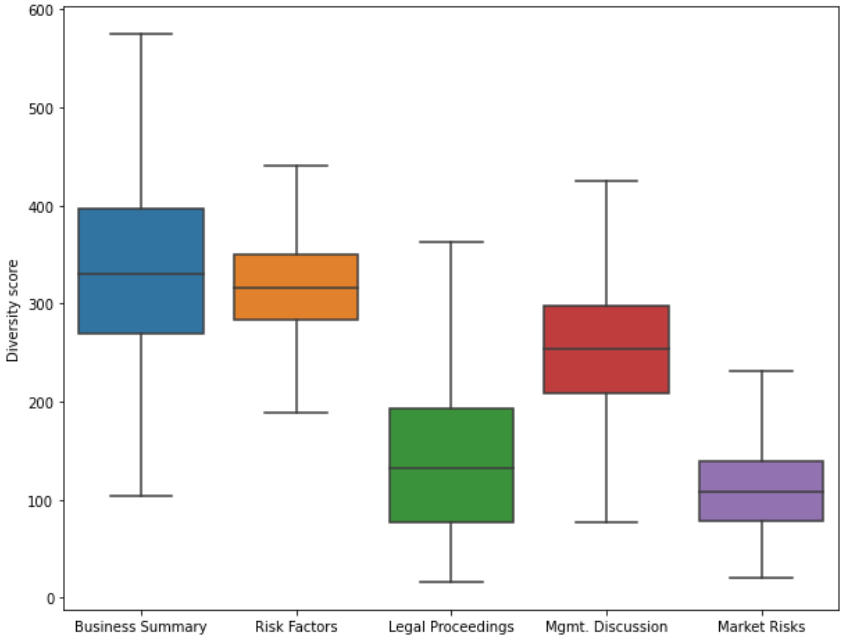

Figure 6: Yule's index of diversity across sections 10-K filings.

7 Modelling approach

We can use the textual features in a number of different predictive models. Overall, there are three paradigms we could consider. In all of the below, assume t_0 = issue date and $t = t_0 + \tau$, where τ is some non-zero amount of time (we will typically assume three years). The three options are - in broad strokes - as follows:

1. Regression against a continuous variable. This could be attempting to predict the spread at issue, or (perhaps more useful) the (relative) spread change from issue to t. This would most simply be done with a linear regression.
2. Classification into a categorical variable. This could be predicting whether a bond/loan downgrades at least one notch from issue to t, or whether it moves below some cutoff (say CCC- or similar) within this time-frame. This could (most simply) be done with at logistic regression or a random forest classifier.
3. Predicting time to event. This could, for example, be predicting time till downgrade k notches below issued rating, or time till some relative spread change. This is then a survival analysis problem, and typically, one would predict this using a Cox proportional hazards regression. This allows us to e.g. give a downgrade probability from issue to t.

All three of these approaches are viable options, but the latter in particular has a number of nice properties for us. Namely, it allows us to include *all* data points in our modelling - using the two other approaches, we are forced to exclude names with less than three years of history. Furthermore, it is intuitively the way we want to model our problem, and it makes names easily comparable in risk at various time horizons. For these reasons, we focus our efforts on the cox proportional hazards regression.

8 Cox proportional hazards regression

In CoxPH, we model a baseline hazard rate $\lambda_0(t)$, which we then adjust for each input data point X_i according to a number of covariates (factors). As mentioned above, in our case the hazard rate is exactly the *probability of downgrade* below issued rating. To put it briefly[13], the predicted hazard rate at time t for a given data point X_i can be described as

[13]For a more thorough - but quickly readable - overview, see http://courses.washington.edu/b515/l17.pdf

$$\lambda(t, X_i) = \lambda_0(t) \cdot \exp(\beta_1 X_{i,1} + \beta_2 X_{i,2} + \cdots + \beta_k X_{i,k})$$

where $X_{i,k}$ is the k'th feature of input X_i. As such, $\lambda_0(t)$ is constant for all input points.

Given a CoxPH model, we evaluate its predictive ability by scoring the *concordance* of the model. Concordance of a model M with definitions

$$M \, : \, \mathrm{R}^d \times \mathrm{Z}_+ \, \rightarrow \mathrm{R}_+$$

and

$$M(X, t) = \lambda(X, t)$$

is scored as the *fraction* of pairs that are ordered correctly by M. More formally, given two datapoints (bonds/loans) X_i and X_j, where we assume (without loss of generality) that X_i downgraded before X_j, we can then define the indicator

$$[X_i, X_j]$$

to be 1 when $M(X_i, t) < M(X_j, t)$, and 0 otherwise. As such, this indicator is true when the model correctly predicts the order of downgrade for a given pair of names. The concordance is then calculated as the sum of this indicator for every pair in data set, i.e.,

$$Concordance(M, \mathbf{X}) = \frac{1}{(n(n-1))/2} \sum_{i=1}^{n} \sum_{j>i}^{n} [X_i, X_j]$$

where $\mathbf{X} = \{X_1, X_2, \ldots, X_n\}$ is the entire dataset. As one may observe, this measure means that a model with a concordance of 1 is a perfect model (it predicts every ordering correctly) and a model with a concordance of 0 is a perfectly wrong model (it predicts every ordering wrongly). By randomly predicting the order of each pair, one thus expects an overall concordance of 0.5. For a model to hold some predictive power, we require that it has a concordance significantly above 0.5. To avoid data leaking, the concordance is only calculated for the test set, after the model has been trained on the training set.

8.1 Baseline zero-covariates model

Before introducing any covariates into our model, we can consider $\lambda_0(t)$, the zero-covariates estimator of the hazard rate. This function is consistent across any set of covariates since it is exactly the curve for which all coefficients of covariates are zero. For the 10-K dataset, the baseline hazard rate $\lambda_0(t)$ looks as seen in Figure 7. This baseline survival curve will then be 'nudged' in different directions (up/down) when we add covarites, given a different survival curve for each name (unless they have identical covariates).

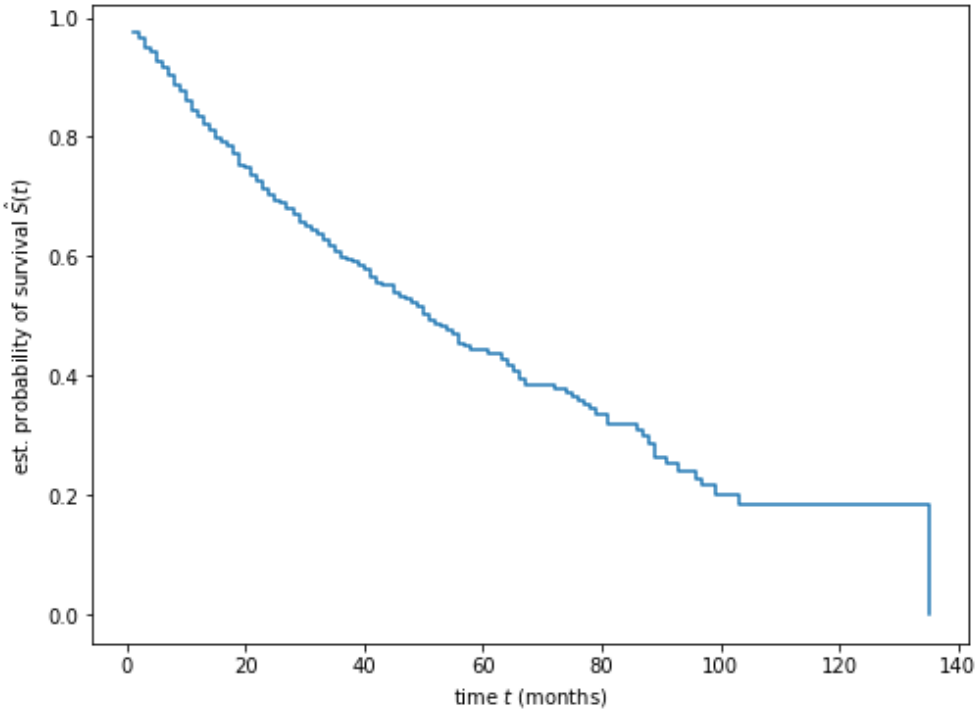

Figure 7: Estimated survival probability $\hat{S}(t)$ for the US HY 10-K dataset. Here, time t is denoted in months, and 'death' is downgrade below issued rating.

8.2 Normalization and imputation

Finally, we note our procedures on normalization of features, and imputation of missing covariates. With regards to the former, we follow the common approach and scale all features to have mean 0 and standard deviation 1. This is helpful in several regards, and in particular it simplifies comparability between features.

Next, we tackle the issue of missing covariates in our data sets. As we have seen, our regression models take as input a number of features (covariates) for each bond/prospectus. In the ideal world, all of these features would be available for all bonds. Unfortunately, this is often not the case - for example, some older prospectuses may not include certain section that weren't a requirement years ago, or some section may be empty if the company has no remarks on the given section (e.g. legal proceedings). As such, we must choose a way to handle this issue. In general, there are three approaches to consider:

1. Complete cases only. In this case, one would exclude data points where some features are missing. This means the signal is retained completely for those points, but one loses any signal that may be present in the excluded data points. If the number of excluded points is fairly low, this is often a good choice. However, if many of the data points are missing only a few features, this method can lead to excessive dropping of data.
2. Data imputation from other features. Here, in contrast to the above, one instead wants to retain all data points. To do so, a missing value in a given column is imputed from the columns where values are present for the given data point. This imputation can be done in a number of ways, for example through a k nearest neighbors classification imputation, where the value of the missing feature is set to the average of the k nearest neighbors according to the *non-missing* features. [14] This approach can have some issues if missing columns are not correlated in any way to those present, which is specifically an issue for our 10-K dataset. In this case, we run the danger of the imputed values adding more noise than signal.
3. Averaging. By averaging, we mean simply taking the mean of the given column, and using that as the value of that column. Since we have normalized all columns to have zero mean, this effectively means setting all missing values to zero. In essence, this means we include no signal from those features, as $\beta_i \cdot 0 = 0$. This does have one issue,

[14] This can be easily done through various libraries, amongst those https://scikit-learn.org/stable/modules/generated/sklearn.impute.KNNImputer.html

namely that the signal data points with missing covariates may generally have lower signal, given that several of the covariates may be zero. However, given that we don't necessarily have any intuition of the 'sign' of many of our features[15], this may not be that much an issue overall.

We experiment with all three options. The complete case options leaves us with a data set containing only $\frac{1}{3}$ of the original data points if we want to test all features. We deem this reduction too large and thus do not pursue the option further. For data imputation, the approach seems viable, but a lack of correlation between different features makes the imputation fairly poor. In particular, correlation is mostly high for features computed for the same section (so, e.g, positive fraction and subjectivity score of the Risk Factors section), but when one of such features is missing, the others typically will be too (following the previous example, neither will be available if the Risk Factors section is not available). As such, the imputation algorithms perform poorly, and seem to add little signal in comparison to noise. We thus proceed onward using the averaging-approach. This is in particular implies that after our normalization, a (hypothetical) data point with all features missing simply has the baseline survival probability and hazard curve.

8.3 Validation

To validate a given model, we perform K-fold cross validation. Here, one randomly splits the dataset into K partitions. Iteratively, the model is then fit on $K-1$ of these partitions and tested on the left-out, K'th partition. This procedure is repeated K times, until all partitions have been used as the test set. This reduces the probability of introducing bias in the model/feature selection process, as the model is both tested and trained on every part of the dataset. To improve this even more, we can *repeat* the K-fold cross validation, using a different random seed each time. This reduces the probability of introducing bias by a lucky/unlucky partitioning of the dataset. Generally, unless noted otherwise, we set $K = 10$, and also set N. the number of repetitions, to 10. This means each model is trained and tested on 100 different partitions of the test set. Finally, to estimate the concordance of model, we then average this score for each partition, i.e.,

$$ValidatedConcordance(M, \mathbf{X}, K, N) = \frac{1}{K \cdot N} \sum_{i=1}^{N} \sum_{j=1}^{K} Concordance(M_{i,j}, \mathbf{X}_{i,j}^{test})$$

[15]Consider, for example, whether you think a more positive sentiment in the risk factors is a sign of fewer severe risks (so higher survival probability), or an attempt to influence reader perceptions of the text to the more positive side (so a potential deception). For many of the features outlined in section 5, one can in similar veins hypothesize the 'sign' of the feature to go either way.

where

$\mathbf{X}_{i,j}^{test}$ is the test fold in the j'th partitioning in repetition i.
$\mathbf{X}_{i,j}^{train}$ is the train train in the j'th partitioning in repetition i.
$M_{i,j}$ is the model trained on $\mathbf{X}_{i,j}^{train}$
K is the number of folds in the K-fold cross validation.
N is the number of repetitions in the validation scheme.

8.4 M_0 - Baseline market-informed model

If our results are to be of any meaning, our model must outperform the baseline market 'model'. With this we mean simply predicting downgrades based on two things; how the market valuates the asset, and the market environment at t_0. As such, we first define this baseline model from these two criteria. We describe the market's valuation of the asset by it's spread relative to the average spread of the index, and we model the current market environment as the 1y change in average spread. As such, this gives us our baseline model:

$$\lambda(t|X_i) = \lambda_0(t) \cdot \exp\left(\beta_1 \cdot \left(\text{OAS}_M^{t_0} - \text{OAS}_i^{t_0}\right) + \beta_2 \left(\text{OAS}_M^{t_0} / \text{OAS}_M^{t_0 - 1y}\right)\right)$$

where

$$\begin{aligned}
\lambda(t|X_i) &= \text{Hazard ratio at } t \text{ for } X_i \\
\lambda_0(t) &= \text{Baseline hazard ratio at } t \\
\text{OAS}_M^{t_0} &= \text{Average OAS of index at } t_0 \\
\text{OAS}_i^{t_0} &= \text{OAS of } X_i \text{ at } t_0
\end{aligned}$$

One particularity of this is why we use the 1y change of average market spread, and not the level. The reasoning behind this is that the momentum of the market tells us a lot more about market conditions than the level. To see this, consider how different the market environment may be if the last year widened 100bps to 500, compared to tightening 300bps to the same level. As such, we hypothesize that the momentum of the spread better catches the environment.

The idea is then that we build further upon this model, using some of the previously described features to improve the accuracy (*concordance*).

8.5 Feature selection

When testing the features outlined in section 5, it is imperative that we do so systematically. In this regard, we generally compare two given models by their *Akaike information criterion* (AIC), which is given as

$$AIC(M) = 2k_M - 2\ln \hat{L}_M$$

where M is the given model, k_M is the number of estimated parameters in M, and $\hat{L}_M$ is the maximized value of the likelihood function for M. This means that the model is penalized for having too many parameters through the k_M term. As an alternative, we could consider the Bayesian information criterion (BIC), where the former term is replaced with something proportionate to the number of subjects, typically logarithmic, e.g.,

$$BIC(M) = \ln N \cdot k_M - 2\ln \hat{L}_M$$

where N is the number of subjects (bonds). Due to their penalizing on number of parameters, both criteria are used in an attempt to avoid overfitting, but in practice BIC often leads to underfitting, as too few meaningful parameters are selected. As such, we use AIC as one of the main ways to gauge the comparative performance of models, alongside the aforementioned concordance index, which gives us an idea of the accuracy of the models. By itself, an AIC score does not tell us much about how accurate a model is.

9 Results

We summarize here the results of our experiments, split into sections according to the various features describe earlier. All results and scores are according to 10-fold cross-validated testing, and in all cases M_0 refers to the baseline model described in Section 8.4. All results are from experiments ran on the 10-K data set described earlier.

As an **important note**, we have tested only *hypotheses* which we have formulated throughout the project. That is, out of the 100 or so features we compute in total (many more if you consider all the different ways we could e.g. do topic modelling or word embeddings), we do not run a thorough parameter search to find the highest-possible concordance or AIC score from all variations of these features. This is because of primarily two reasons; first, we want our hypothesis to be explainable, and second, such a procedure may often lead to building models that are overfit to the given sample, and not actually representative for what one attempts to model. This latter factor means that performance may deteriorate when moving

Model	**Concordance**	AIC	Log-likelihood
M_0	0.63495	3645.5	-1820.8
M_0 + log_3_length	0.64154	3642.9	-1818.4
M_0 + 1_neg_frac	0.63849	3642.6	-1818.3
M_0 + log_3_length + 1_neg_frac	0.64619	3641.2	-1816.6
M_0 + 7A_leverage	0.63910	3643.5	-1818.8
M_0 + log_3_length + 1_neg_frac + 7A_leverage	**0.64930**	**3639.6**	**-1814.8**
log_3_length + 1_neg_frac + 7A_leverage	0.55247	3695.0	-1844.5

Table 1: Highlighted results for the descriptive statistics features. Here, *log_3_length* refers to the logarithm of the length of the *"Legal Proceedings"* section, *1_neg_frac* refers to the negative fraction of the *"Legal Proceedings"* section, and *7A_leverage* refers to the absolute word count of the word *leverage* in the *"Quantitative and Qualitative Disclosures About Market Risk"* section. The number of eligible bonds is 1106.

such a model to production.

For each model, we report concordance, AIC, and log-likelihood. We recall that for the first and last measures, higher scores are better, while the opposite is true for AIC. Bold text highlights the best model w.r.t. a given measure, and M_0 still refers to our baseline model.

9.1 Descriptive statistics and textual features

First, we start with the simplest of features, the descriptive statistics and the textual features. Here, we include only results for those features that actually improved results on top of M_0 - it suffices to say that those not included do no improve upon this model, but they may often still improve upon *no* model, i.e., they are often better predictors than randomly deciding.

The results for the models can be seen in Table 1. Here, we can see that three of the features managed to improve upon our baseline model:

- *log_3_length*, the logarithm of length of the *"Legal Proceedings"* section. The hypothesis behind this feature was that a longer such section implies that the company is facing more legal proceedings, which is a risk, increasing downgrade probability. The sign of the coefficient agrees with this, as a longer section increases downgrade risk.

Model	**Concordance**	AIC	Log-likelihood
M_0	0.64222	2110.9	-1053.4
M_0 + topic mod. on FLS 1A	0.63395	2115.0	-1051.5
topic mod. on FLS 1A	0.50134	2157.6	-1074.8
M_0 + topic mod. on GPT "Tech Risk" 1A	**0.66036**	**2075.4**	**-1027.7**
topic mod. on GPT "Tech Risk" 1A	0.54956	2118.9	-1051.5

Table 2: Highlighted results for topic modelling experiments. The second and third models use topic modelling on FLS from 1A, the *Risk Factors* section. The fourth and fifth models use topic modelling on the GPT query regarding technology risk, as outlined in Section 5.6. The best performing model uses the latter, while the former fails to outperform the baseline. The number of eligible bonds is 693.

- *1_neg_frac*, the fraction of negative sentences in the *"Business Summary"* section. There was no clear hypothesis on the direction the sentiment should go, but the coefficient indicates that a more negative wording is correlated with higher downgrade risk.
- *7A_leverage*, the word count for the word *leverage* in the *"Quantitative and Qualitative Disclosures About Market Risk"* section. The intuition here was that the more talk/disclosures around leverage, the higher downgrade risk.

Each of these features improve the baseline, with the best performing model being the one including the baseline and all three features described here. This model reaches almost 65% concordance, while the baseline model lies at 63.5%. Without the two baseline features, the model scores just above 55%. *log_3_length* is the best stand-alone indicator of the three. Overall, the three features improve upon the baseline, but do not compare without the baseline features.

9.2 Topic modelling

Next, we consider the results of our topic modelling experiments. We split these experiments into two; first, the topic modelling based on the FLS from the *Risk Factors* section, and second, the topic modelling based on GPT-queries, as described in Section 5.6. The results can be seen in Table 2. As can be seen from these results, the GPT-based model outperforms all other models when combined with the baseline. On it's own, it is however still significantly behind the baseline model. On the other hand, the 'simpler' topic model underperforms the

Model	**Concordance**	AIC	Log-likelihood
M_0	0.63856	2824.2	-1410.1
M_0 + Doc2Vec embeddings	0.62005	2837.1	**-1396.5**
Doc2Vec embeddings	0.50023	2887.5	-1423.7
M_0 + PCA on GPT embeddings	**0.64393**	**2817.7**	-1396.9
PCA on GPT embeddings	0.56655	2869.4	-1424.7

Table 3: Results for the GPT word embedding models. The second and third models shown both use 20-dimensional Doc2Vec embeddings, while the fourth and fifth use 10 principal components determined from the 1536-dimensional word embeddings generated by GPT. The best performing model adds these latter word embeddings to the baseline model M_0. The number of eligible bonds is 903.

baseline. All in all, topic modelling, in particular with the GPT-querying, seems like a very interesting direction to focus more resources. The query output depends heavily on how the prompt is written, and there is little chance the current example prompt is optimal. However, following the Capital Four fundamental scoring parameters seems like an intuitive and straightforward place to start.

9.3 Word embeddings

Finally, we discuss the results from the most 'hands-off' approach of them all: the word embedding models. We include results for both Doc2Vec and GPT-generated embeddings, and the results can be seen in Table 3. These embeddings are on the extracted FLS of the *Business Summary* section. The Doc2Vec models fail to improve upon the baseline, and by themselves do not even improve upon randomly guessing. This is most likely a sign that more data is needed for this approach - since the embeddings are trained on only 1k documents, the representations simply aren't good enough.

On the other hand, the GPT-generated embeddings (after PCA) manage to improve upon the baseline, albeit less than 1% in concordance. However, both AIC and log-likelihood are significantly improved. As such, it seems adding the embeddings do improve the model, and they are also significantly better than randomly guessing when tested on their own (reaching 56.6% concordance). In general, however, methods such as these - purely neural - are known to require fairly large amounts of data to be efficient. Thus, these results seem to indicate that the direction is viable, but more data would improve the method.

10 Future perspectives

This project was developed, end-to-end, in a timespan of less than three months. Given the current tailwind in NLP, one could have spent three *years* exploring the topic instead. As that amount of time is infeasible in this case, we outline below some promising possibilities/directions for future research. Some should be fairly simple modifications or additions, while others may be entirely different projects by themselves. Common amongst them all is that they came up during the project, but we did not have time to investigate them further.

- The GPT-querying model (with topic modelling, outlined in Section 5.6) was the best performer of the bunch, even though it was a fairly late idea and we did not spend too much time on it. The concept of extracting specific parts of documents, which we deem as more relevant, proved to be an interesting direction. With and outset from the C4 fundamental scoring parameters, one could devote more time towards engineering prompts that are successful in extracting just the right information. In addition, we chose to put topic modelling on top of this first step - one could also perform other analysis on the synthesized summary.
- Topic modelling also worked well on our data set, but leaves a lot of room for improvement. There are several possibilities we discussed w.r.t topic modelling. First, a qualitative layer of selecting topics that we deem 'relevant' could remove spurious topics, and is seen to be done in some other studies [4]. Secondly, it was proposed in a similar paper ([6], p. 20) that one could label each *sentence* as a topic, and then average these labels for an entire document to get a loading on each topic. This seems like a better way to represent topics in very long documents, such as prospectuses. Finally - and most comprehensively - one could look into grouping words into topics not by their co-occurrence in texts, but rather in their co-occurence in *outcomes*. As an example, assume we map a group of words to a spread bucket. Then, a bond with a high loading on that group of words (topic) would perhaps have a high probability of ending up in that spread bucket. This theory/methodology doesn't seem to have been tested anywhere, so the road ahead might be long.
- In the very last phase of this project, Blooomberg released their own language model, *BloombergGPT* [5]. This model is akin to previous GPT models, but it has been specifically trained on text from financial domains - amongst those Bloomberg News and earnings reports. This leads to a 345 billion parameters language model that performs *very* well on a lot of benchmarks - in particular financial domain benchmarks,

but also decently on general-puporse benchmarks.[16] A language model finetuned to the financial domain was another big discussion during this project, but the scale required to build something like BloombergGPT requires an amount of compute power infeasible for C4. Unfortunately, Bloomberg has *not* made their model public - or even accesible via API - and aim to only use it for in-house NLP tasks. Still, the idea of finetuning a GPT model to the domain of credit seems worth investigating, and can definitely be done on a smaller scale.[17]

- For some of the features used in this project, more data points would be very useful - as it is in most cases. Outside from the obvious extension of attempting to gather more documents in the current universes, one could also consider using IG documents to train some of the models. Generally, there is a much higher availability and quality of data and documents for IG names, and particularly the neural models would benefit greatly from simply getting more text.
- A huge opportunity for value lies in the direction of developing more NLP-based tools. An example of this is screening/processing tool mentioned in this report, but this could be made much for useful for analysts in particular. Often, such tools can be developed as a stepping stone on the road to larger goals - for example modelling downgrade/default risk. Throughout this project, we also saw the value that NLP analysis on it's own gave, even before the modelling step. As an example, we discovered that LL material is generally much more positively framed than HY equivalents. Using NLP analysis to uncover biases in investment decisions seems like a low-hanging fruit for the future.
- Finally, one could consider the possibility of developing an analyst-guided summarization engine based on GPT. Given a text or recording, one could generate a handful of summaries/extraction, and then a C4 analyst could choose the best summary. This would beneficial in both directions - analyst would get the best summary of the handful (they selected it), and the model would improve more the analyst's liking. This is similar to how GPT-3 was trained to become ChatGPT.

[16] ... according to the authors. No way to reproduce it for anyone outside Bloomberg.

[17] For example, through OpenAI's newly available finetuning https://platform.openai.com/docs/guides/fine-tuning

References

[1] Adrian Amzallag. Parsing prospectuses: A text-mining approach. *ESMA TRV*, 2022.

[2] Lauren Cohen, Christopher J. Malloy, and Quoc Nguyen. Lazy prices. *Academic Research Colloquium for Financial Planning and Related Disciplines*, 2019.

[3] Luke Jordan. Bond default prediction with text embeddings, undersampling and deep learning. *CoRR*, abs/2110.07035, 2021.

[4] Jie Tao, Amit V. Deokar, and Ashutosh Deshmukh. Analysing forward-looking statements in initial public offering prospectuses: a text analytics approach. *Journal of Business Analytics*, 1(1):54–70, 2018.

[5] Shijie Wu, Ozan Irsoy, Steven Lu, Vadim Dabravolski, Mark Dredze, Sebastian Gehrmann, Prabhanjan Kambadur, David Rosenberg, and Gideon Mann. Bloomberggpt: A large language model for finance, 2023.

[6] Yangyang Fan Yi Yang, Kunpeng Zhang. Analyzing firm reports for volatility prediction: A knowledge-driven text-embedding approach. *INFORMS Journal on Computing 34(1):522-540.*, 2021.