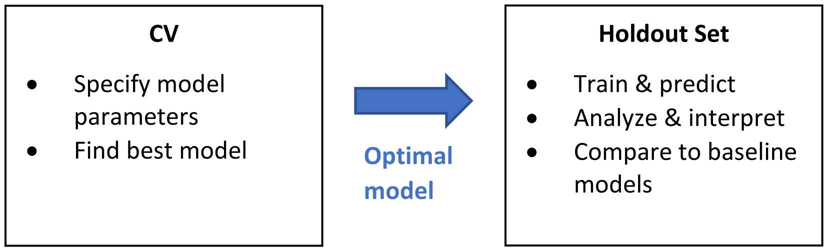
Machine Learning Implementation

1. Roadmap

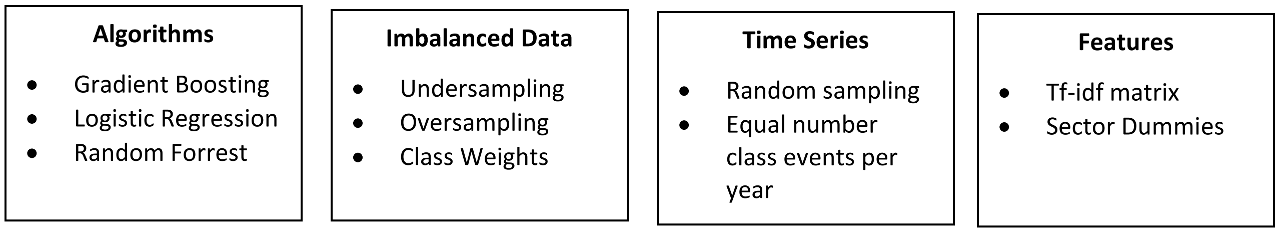
The machine learning problem is structured as a binary classification model. The two nuances are that the data is (i) highly class imbalanced and (ii) in a time-series structure. We start by performing cross-validation across parameters specific to these challenges before identifying the optimal model to apply to the holdout set. After training and predicting the holdout set, we analyze and interpret these standalone results. The final step is to compare the NLP model to the results of other approaches. The baseline comparison is to a similar model using traditional financial ratios and, following our observation of persistence or autocorrelation in drawdowns, we introduce another using only the previous year's drawdown quartile ranking. This latter baseline can be considered a market model as it reflects the market's assessment of short-term company risk.



1.1 Cross-Validation (CV)

1.1.1 Model Parameters

Time-series CV is performed by splitting the validation set into 5 equal parts and using an expanding window approach to define CV sets. Details can be found in the *Data Wrangling* section of the paper. We group the model parameters into 4 sections:



Algorithms denote the usual choice of the learning model we wish to use for training and testing. We select three models suitable for binary classification.

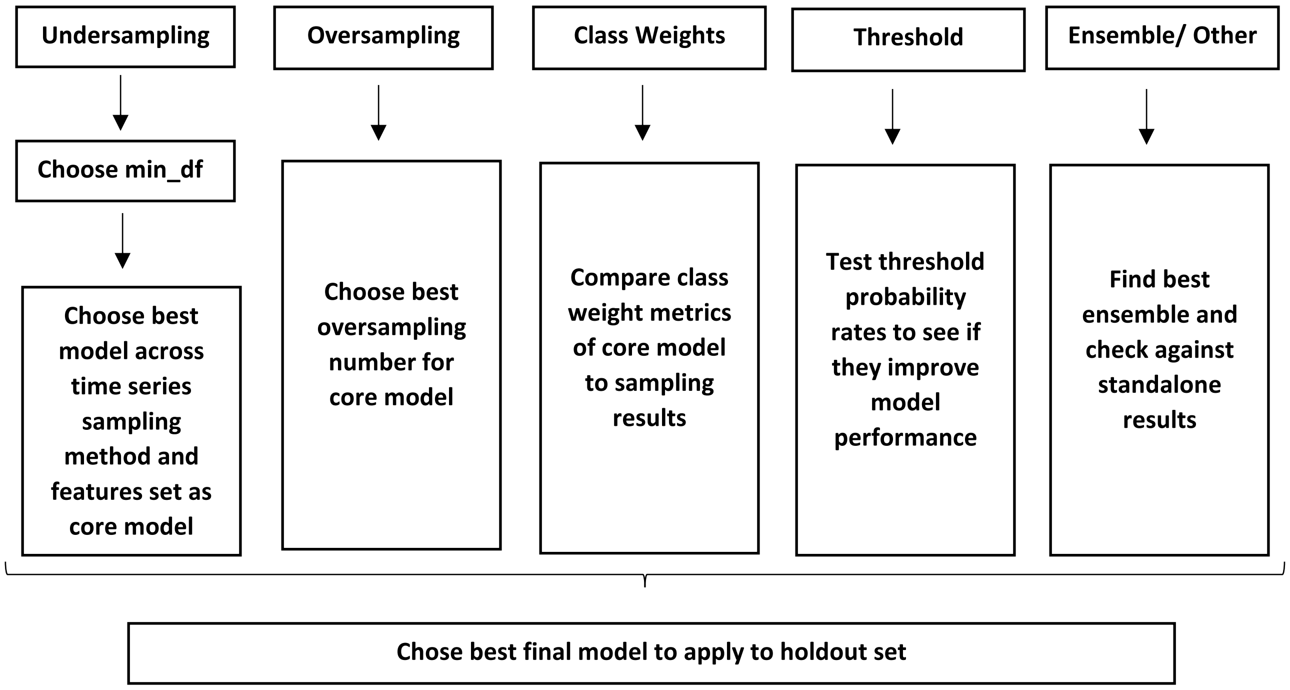
Imbalanced Data considers the three classical approaches of *undersampling*, *oversampling* and using *class weights* to edit the cost function.

Time-Series presents its own challenges and we certainly expect that the annual reports, spanning 22 years, reflect the changing accounting, regulatory and economic landscape of this long period. Resultantly, we validate a random selection of samples against one containing an equal number (ratio) of class events per year.

Features selection is the final group where we consider the addition of two variables to the Tf-idf matrix. Traditional financial measures, such as the *Altman Z-score*, often calibrate their models across sectors and we do likewise by adding *sector dummies* to the features set.

1.1.2 Evaluation Process

There are a lot of moving parts to consider over cross-validation and the below sketch outlines the order and process:


Undersampling
Choose min_df
Choose best model across time series sampling method and features set as core model
Oversampling
Choose best oversampling number for core model
Class Weights
Compare class weight metrics of core model to sampling results
Threshold
Test threshold probability rates to see if they improve model performance
Ensemble/ Other
Find best ensemble and check against standalone results
Chose best final model to apply to holdout set

We do the bulk of the hard work in undersampling where we choose (i) *min_df* for the tf-idf vectorizer, (ii) random sampling or even annual distribution for the time-series sampling and (iii) the optimal features set. This forms the **core model** that we then we use for the rest of this section and without revisiting these parameters.

The core model is used to cross-validate for the optimal oversampling number.

Class weights is then evaluated on the core model and compared against the sampling methods.

Threshold testing involves altering the 50% probability threshold for a positive event prediction and seeing how this impacts evaluation metrics.

We end cross-validation by taking the insights of the previous analysis to construct and test various *ensembles* of algorithms and sampling methods to see if these improve the optimal standalone model. The winner is then used to train and test the holdout set.

1.2 Holdout Set

The holdout set represents 5 years of data (2015-2019) and is tested using an expanding annual window. For example: 2015 is tested on data up to 2014, 2016 is tested on data up to 2015 and so on. These annual results are then aggregated to form consolidated holdout results.

The baseline models used for comparison consist of (i) a financial model (FIN) using a set of common financial ratios as features and (ii) a market model (MKT) using the 1-year lagged annual drawdown quartile of the company as features. The MKT model follows from the discussion on persistence in *Data Exploration*. This allows us to compare the performance of our alternative data model (NLP) against these other approaches.

After comparing headline results, we explore the *business cost of errors* in the confusion matrix by looking at the full drawdown statistics of the confusion matrix classes. Finally, we explore the words with the highest individual probability and group these into natural categories relating to positive event companies.

2. Roadsign

This section describes the metric and weighting scheme that will be used to navigate our way through cross-validation.

2.1 Primary CV Metric: Macro Harmonic Recall

While we will consider several metrics for cross-validation, the principal arbiter will be a version of macro recall. Macro recall is the standard SK-Learn recall function and is simply the evenly weighted arithmetic mean of class recalls. We are more interested in recall than precision as our goal is ambitious and the analyst has many tools beyond annual report language to further scrutinize identified companies. Correctly picking out a decent sized pool of suspect companies for further analysis is more important than limiting this pool for the sake of precision. Equally, we don't want to focus solely on positive event recall as this would lead to a nonsensical model predicting everything as the minority class! Evenly weighting class recall strikes a reasonable balance between these considerations.

The one adjustment we make to standard macro recall is to use the evenly weighted harmonic mean instead of the arithmetic mean. The reason for doing so is straightforward and can be seen from the illustrative example below. Clearly, the 60% metric flatters the model which has miserably failed for one of the classes. In general, the harmonic mean better highlights low outliers.

Recall Class 1	99% (or 21%)
Recall Class 2	21% (or 99%)
Macro Recall	60%
Macro Harmonic Recall	35%

Without digressing into the mathematics, macro harmonic recall will (i) leave the relative ranking of results unchanged to macro recall and (ii) better highlight the difference between ranking members. Resultantly, macro harmonic recall will be the primary metric throughout cross validation. In cases requiring a tie breaker, the standard deviation of macro harmonic recall will be used.

2.2 CV Weighted Average

Unlike regular CV, time-series data uses the expanding window method meaning that each iteration has a different amount of data in their training set. Therefore, it makes sense to factor this into the weighting scheme: more data, more weight. We split the CV set into 5 equal parts and expand by each part when calculating the 4 CV training sets (see *Data Wrangling*). The subsequent part is then used for testing. In general, each CV(*K*) has *K* times the amount of training data as CV(*1*) and, so, has *K%* weight for $K \in \{1,2,3,4\}$.

2.3 Formal Definition

In total, the primary CV metric is calculated as $MHR_k = \sum_{k=1}^{4} k\% \cdot MHR_k$ where $MHR_k = \frac{2 \cdot PR_k \cdot NR_k}{PR_k + NR_k}$ and $MHR = macro\ harmonic\ recall, PR = positive\ class\ recall, NR = negative\ class\ recall$ and k represents the k^{th} cross-validation using expanding windows.

2.4 Domain Metric

As this is a machine learning project, we will stick to the macro harmonic recall metric for cross validation. However, it is worth mentioning that one could also use a domain (business) metric. In this case, the most sensible would be the evenly weighted drawdown of a portfolio short the predicted positive companies and long the predicted negative. We can also use this logic and the continuous nature of the underlying drawdown data to compute the 'cost' of false negatives. While we consider the cost of errors later in the report, using the business metric in cross-validation is something we will leave to future work.

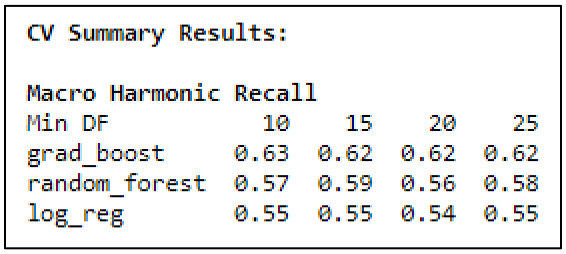
3. Cross-Validation

Enough talk. Let’s get on with CV and see some numbers!

3.1 Undersampling

3.1.1 Choosing min_df

The below table shows the CV results for an undersampled random selection of events per class across various min_df values:



```
CV Summary Results:

Macro Harmonic Recall
Min DF            10    15    20    25
grad_boost      0.63  0.62  0.62  0.62
random_forest   0.57  0.59  0.56  0.58
log_reg         0.55  0.55  0.54  0.55
```

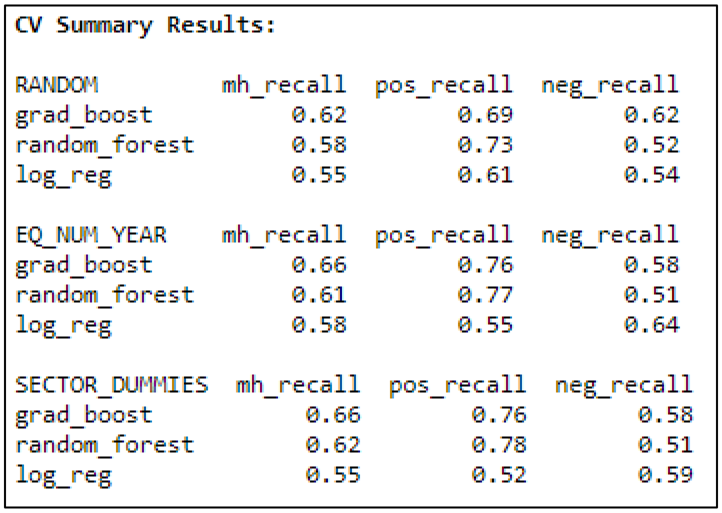
The primary observations are:

- For each learning algorithm, there is **little variance across scores for min_df values**
- **Gradient boost outperforms** other algorithms across all values of min_df

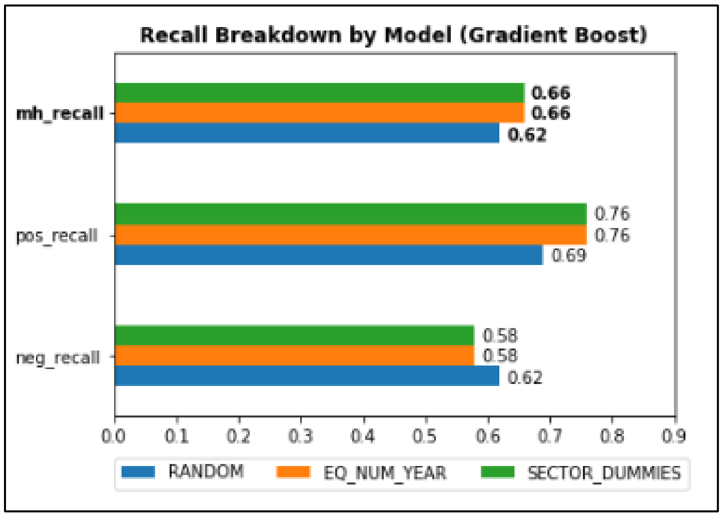
Given these observations, we use min_df = 25 for all subsequent calculations as this dramatically reduces the size of the tf-idf matrix for little give-up, if any, on performance. We continue to track the performance across various algorithms as this comes at minimal computational cost (runtime mostly depends on SciPy. Sparse wrangling of training data) and this may be useful when exploring ensemble opportunities.

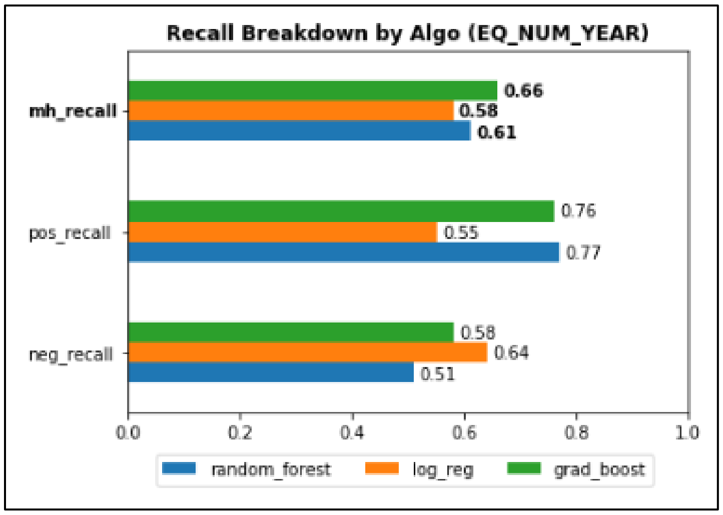
3.1.2 Finding the Core Model

Results



```
CV Summary Results:

RANDOM         mh_recall  pos_recall  neg_recall
grad_boost          0.62        0.69        0.62
random_forest       0.58        0.73        0.52
log_reg             0.55        0.61        0.54

EQ_NUM_YEAR    mh_recall  pos_recall  neg_recall
grad_boost          0.66        0.76        0.58
random_forest       0.61        0.77        0.51
log_reg             0.58        0.55        0.64

SECTOR_DUMMIES  mh_recall  pos_recall  neg_recall
grad_boost           0.66        0.76        0.58
random_forest        0.62        0.78        0.51
log_reg              0.55        0.52        0.59
```





- **Gradient boosting mh_recall continues to outperform** the other algorithms across all cases
- **Distributing the event classes evenly over time leads to the greatest improvement of** mh_recall (62% to 67%)
- **Adding sector dummy variables to the features set results in no improvement** and is subsequently ignored
- **Negative event recall remains exceptionally stubborn** and the improvement in mh_recall is exclusively owing to positive event recall
- **The relative breakdown across recalls suggests gradient boosting and log reg might be an interesting ensemble to check**. There is not much between grad boost and random forest for positive recall and log_reg performs best for negative recall

3.2 Oversampling

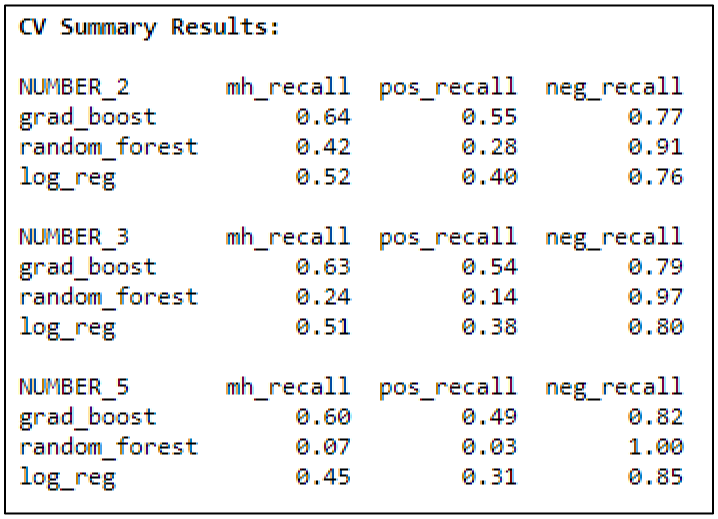
Core Model

- Class samples constructed to have equal number of events per year
- Features set includes the Tf-idf matrix and the 1-year lagged drawdown quartile of the company associated with the document

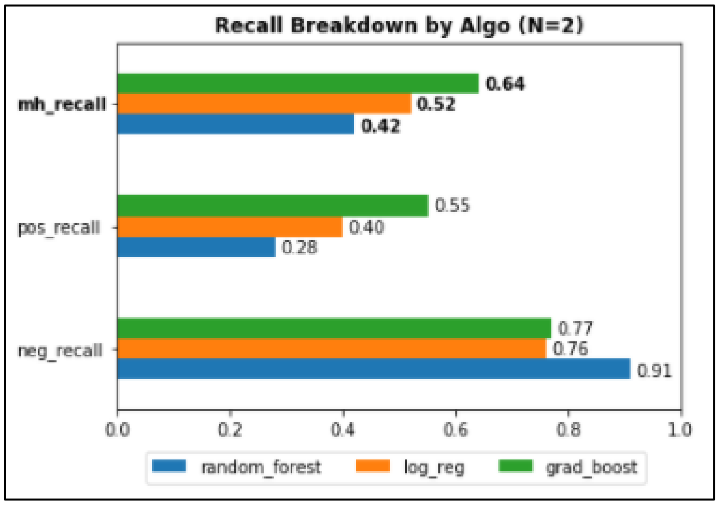
3.2.1 Finding the Optimal Oversampling Number

In this section, we use the core model discovered in undersampling but now apply this for different values of oversampling. The majority class negative event dataset is an order of 20 times larger than the minority class and for the purposes of practical computation, we limit the oversampling to a maximum of 5 times the minority class. Since this “amount” of oversampling exists in a spectrum, we perform cross-validation for $n \in \{2,3,5\}$ where the size of the samples are equal to n times the minority class.

Results



```
CV Summary Results:

NUMBER_2        mh_recall  pos_recall  neg_recall
grad_boost           0.64        0.55        0.77
random_forest        0.42        0.28        0.91
log_reg              0.52        0.40        0.76

NUMBER_3        mh_recall  pos_recall  neg_recall
grad_boost           0.63        0.54        0.79
random_forest        0.24        0.14        0.97
log_reg              0.51        0.38        0.80

NUMBER_5        mh_recall  pos_recall  neg_recall
grad_boost           0.60        0.49        0.82
random_forest        0.07        0.03        1.00
log_reg              0.45        0.31        0.85
```

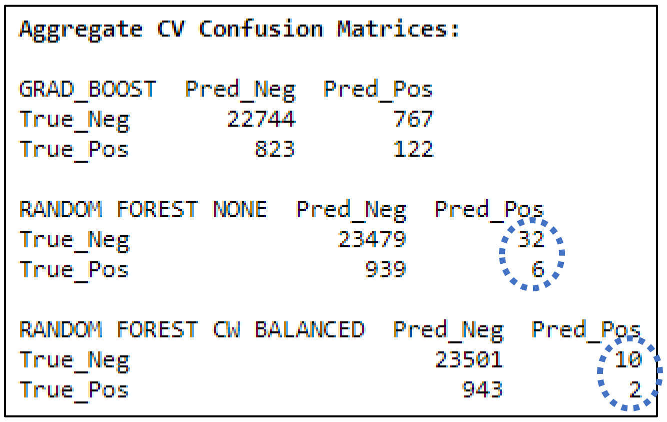


- **The optimal oversampling number is 2.** However, mh_recall is stable across the numbers tested and the difference is marginal
- **Gradient boosting continues to outperform**
- **Optimal undersampling marginally outperforms oversampling** (+2%)
- Despite similar mh_recall numbers, breakdown is significantly different with **oversampling scoring better negative recall and undersampling better positive recall**
- **Logistic regression is the exception** and the recall breakdown suggests we try grad_boost and log_reg as an ensemble
- Stability of performance across algos exhibits significance variance with **random forest showing woeful underperformance as it overfits to the majority class**

3.3 Class Weights

In this section, we use class weights in SK- Learn's Random_Forest model to explore how this compares to oversampling and undersampling. We also run gradient boosting alongside this for like-to-like comparative purposes. The one challenge we have is incorporating the observation of improved performance for the even distribution of classes over time. Pure implementation would result in a training set with an equal number of majority and minority samples and we would just be back to undersampling. To control for this, we set the annual *proportion* of negative events to that of positive events.

Results



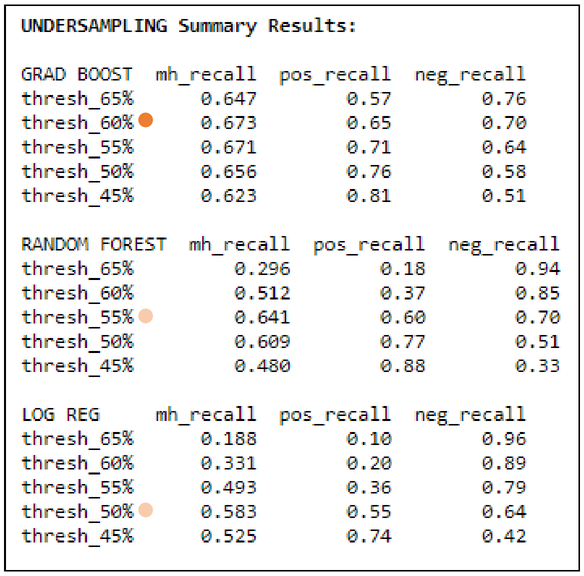
```
Aggregate CV Confusion Matrices:

GRAD_BOOST  Pred_Neg  Pred_Pos
True_Neg       22744       767
True_Pos         823       122

RANDOM FOREST NONE  Pred_Neg  Pred_Pos
True_Neg               23479        32
True_Pos                 939         6

RANDOM FOREST CW BALANCED  Pred_Neg  Pred_Pos
True_Neg                      23501        10
True_Pos                        943         2
```

- **Random forest performs exceptionally badly over both the class balanced and regular cases.** The degree of this poor performance and the lack of improvement seen for the balanced case forced a recheck of this result and coding many times over. No error was discovered.
- Given the disappointing performance and high overfitting to the majority class of random forest in oversampling, we can **surmise that the class weights result are likely to be more a reflection on random forest's suitability for this dataset than offer any indication on a general class weight approach.**

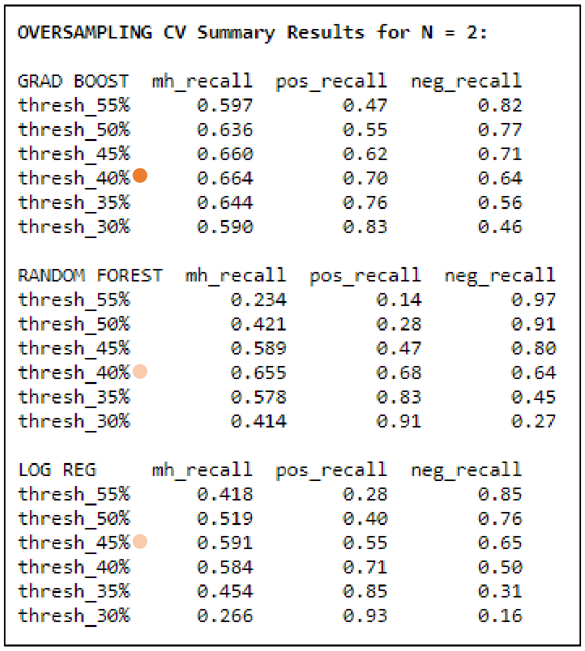
3.4 Threshold Testing

Another approach to fine tune the model is to cross-validate for the probability threshold denoting a positive event. This is automatically set at 50% but can be altered to try and compensate for a model's propensity to over or underpredict the positive event. The below table does this for a variety of threshold levels across undersampling and oversampling.

Results



```
UNDERSAMPLING Summary Results:

GRAD BOOST  mh_recall  pos_recall  neg_recall
thresh_65%      0.647        0.57        0.76
thresh_60%●     0.673        0.65        0.70
thresh_55%      0.671        0.71        0.64
thresh_50%      0.656        0.76        0.58
thresh_45%      0.623        0.81        0.51

RANDOM FOREST  mh_recall  pos_recall  neg_recall
thresh_65%         0.296        0.18        0.94
thresh_60%         0.512        0.37        0.85
thresh_55%●        0.641        0.60        0.70
thresh_50%         0.609        0.77        0.51
thresh_45%         0.480        0.88        0.33

LOG REG     mh_recall  pos_recall  neg_recall
thresh_65%      0.188        0.10        0.96
thresh_60%      0.331        0.20        0.89
thresh_55%      0.493        0.36        0.79
thresh_50%●     0.583        0.55        0.64
thresh_45%      0.525        0.74        0.42
```



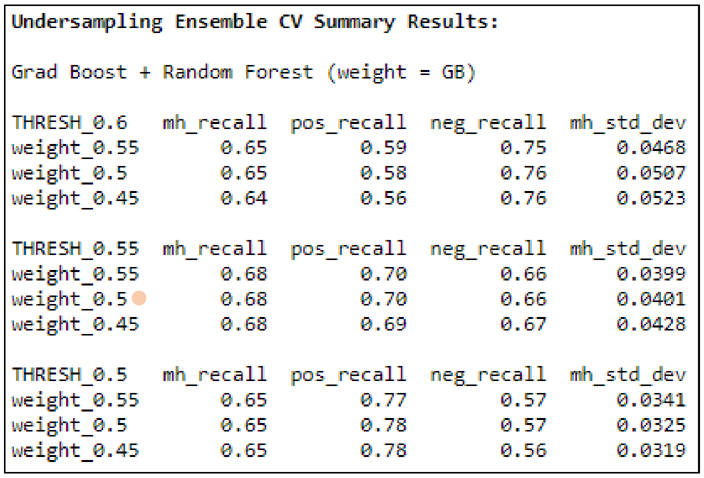
```
OVERSAMPLING CV Summary Results for N = 2:

GRAD BOOST  mh_recall  pos_recall  neg_recall
thresh_55%      0.597        0.47        0.82
thresh_50%      0.636        0.55        0.77
thresh_45%      0.660        0.62        0.71
thresh_40%●     0.664        0.70        0.64
thresh_35%      0.644        0.76        0.56
thresh_30%      0.590        0.83        0.46

RANDOM FOREST  mh_recall  pos_recall  neg_recall
thresh_55%         0.234        0.14        0.97
thresh_50%         0.421        0.28        0.91
thresh_45%         0.589        0.47        0.80
thresh_40%●        0.655        0.68        0.64
thresh_35%         0.578        0.83        0.45
thresh_30%         0.414        0.91        0.27

LOG REG     mh_recall  pos_recall  neg_recall
thresh_55%      0.418        0.28        0.85
thresh_50%      0.519        0.40        0.76
thresh_45%●     0.591        0.55        0.65
thresh_40%      0.584        0.71        0.50
thresh_35%      0.454        0.85        0.31
thresh_30%      0.266        0.93        0.16
```

- **Optimal undersampling improves a meager 1%** to 67% (mh recall) by increasing gradient boosting's threshold to 60%
- **Optimal oversampling improves by 2%** to 66% by decreasing gradient boosting's threshold to 40%
- **Tremendous gains are seen in the poorly performing oversampling models** with a decrease in threshold. Random Forest improves by 40% to 65% by lowering the threshold to 45%
- **The best performing model to this point is undersampling with a gradient boost threshold of 60%**

3.5 Ensembles

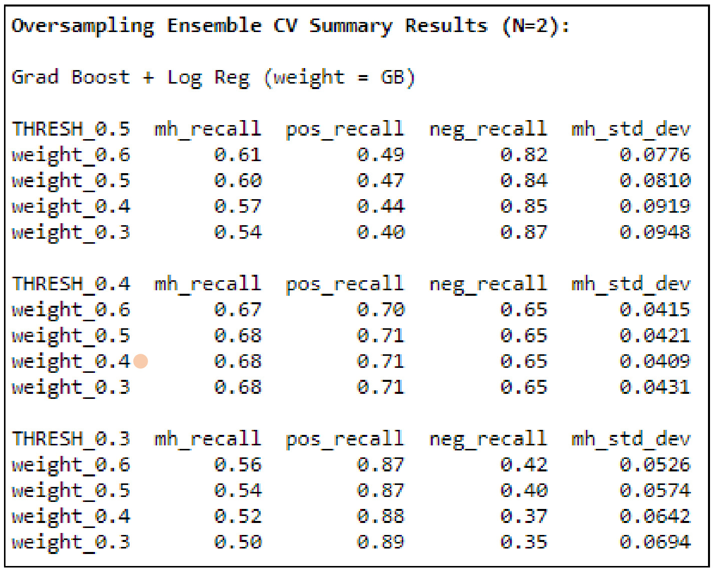
This section looks at three ensemble cases: (i) mixing different learning algorithms for undersampling, (ii) mixing different learning algorithms for oversampling, and (iii) mixing undersampling and oversampling. We cross-validate across both weights and threshold for combined models. The last combination is inspired by undersampling's relatively higher positive recall and oversampling's relatively higher negative recall.

Results



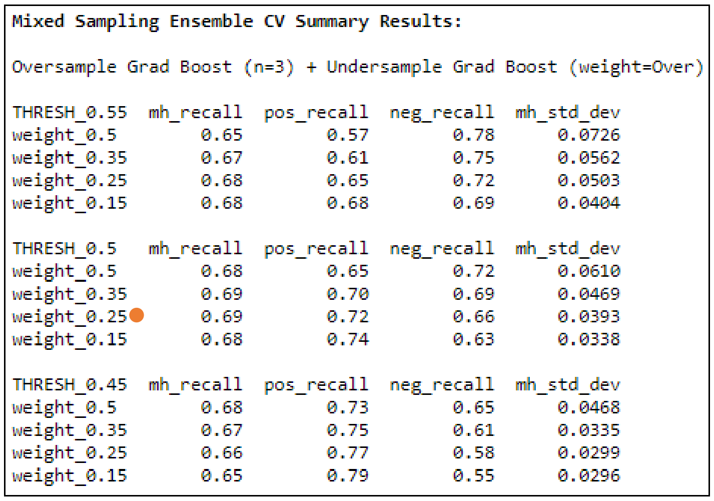
```
Undersampling Ensemble CV Summary Results:

Grad Boost + Random Forest (weight = GB)

THRESH_0.6   mh_recall  pos_recall  neg_recall  mh_std_dev
weight_0.55       0.65        0.59        0.75      0.0468
weight_0.5        0.65        0.58        0.76      0.0507
weight_0.45       0.64        0.56        0.76      0.0523

THRESH_0.55  mh_recall  pos_recall  neg_recall  mh_std_dev
weight_0.55       0.68        0.70        0.66      0.0399
weight_0.5●       0.68        0.70        0.66      0.0401
weight_0.45       0.68        0.69        0.67      0.0428

THRESH_0.5   mh_recall  pos_recall  neg_recall  mh_std_dev
weight_0.55       0.65        0.77        0.57      0.0341
weight_0.5        0.65        0.78        0.57      0.0325
weight_0.45       0.65        0.78        0.56      0.0319
```

- **Undersampling marginally improves to 68% (+1%)** at a threshold of 55% and an even weighting of gradient boost and random forest



```
Oversampling Ensemble CV Summary Results (N=2):

Grad Boost + Log Reg (weight = GB)

THRESH_0.5  mh_recall  pos_recall  neg_recall  mh_std_dev
weight_0.6       0.61        0.49        0.82      0.0776
weight_0.5       0.60        0.47        0.84      0.0810
weight_0.4       0.57        0.44        0.85      0.0919
weight_0.3       0.54        0.40        0.87      0.0948

THRESH_0.4  mh_recall  pos_recall  neg_recall  mh_std_dev
weight_0.6       0.67        0.70        0.65      0.0415
weight_0.5       0.68        0.71        0.65      0.0421
weight_0.4●      0.68        0.71        0.65      0.0409
weight_0.3       0.68        0.71        0.65      0.0431

THRESH_0.3  mh_recall  pos_recall  neg_recall  mh_std_dev
weight_0.6       0.56        0.87        0.42      0.0526
weight_0.5       0.54        0.87        0.40      0.0574
weight_0.4       0.52        0.88        0.37      0.0642
weight_0.3       0.50        0.89        0.35      0.0694
```

- **Oversampling improves to 68% (+2%)** at a threshold of 40% with gradient boosting weighted at 40% and log reg at 60%. The standard deviation of mh recall was needed as a tiebreaker for the weights



```
Mixed Sampling Ensemble CV Summary Results:

Oversample Grad Boost (n=3) + Undersample Grad Boost (weight=Over)

THRESH_0.55  mh_recall  pos_recall  neg_recall  mh_std_dev
weight_0.5        0.65        0.57        0.78      0.0726
weight_0.35       0.67        0.61        0.75      0.0562
weight_0.25       0.68        0.65        0.72      0.0503
weight_0.15       0.68        0.68        0.69      0.0404

THRESH_0.5   mh_recall  pos_recall  neg_recall  mh_std_dev
weight_0.5        0.68        0.65        0.72      0.0610
weight_0.35       0.69        0.70        0.69      0.0469
weight_0.25●      0.69        0.72        0.66      0.0393
weight_0.15       0.68        0.74        0.63      0.0338

THRESH_0.45  mh_recall  pos_recall  neg_recall  mh_std_dev
weight_0.5        0.68        0.73        0.65      0.0468
weight_0.35       0.67        0.75        0.61      0.0335
weight_0.25       0.66        0.77        0.58      0.0299
weight_0.15       0.65        0.79        0.55      0.0296
```

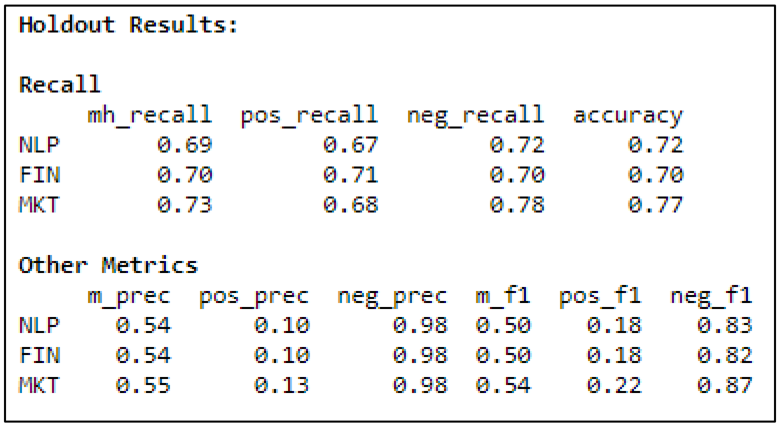
- **Mixed sampling provides the winner with optimal mh recall at 69% and *n=3* for oversampling**
- This occurs with gradient boosting for both models and a weight of 25% for oversampling and a 50% threshold.
- MH recall variance was needed as a tiebreaker to choose the weights

4. Holdout Set

The winning model (mixed sampling ensemble) from CV is used to test the holdout set on an annual expanding window basis from 2015 to 2019 filing date years. Market data for 2019 extends to June 2020 for drawdown calculation purposes.

The current NLP model is then compared to two baseline models using (i) a variety of traditional financial ratios (FIN) and (ii) a market model that simply uses the company's prior year drawdown quartile as features (MKT). For baseline purposes, these models were not optimized. Details of these models can be found in the appendix and more explanation on the discovery of persistence in the bottom quartile rank is in the data exploration section of the full report.

4.1 Headline Results



```
Holdout Results:

Recall
     mh_recall  pos_recall  neg_recall  accuracy
NLP       0.69        0.67        0.72      0.72
FIN       0.70        0.71        0.70      0.70
MKT       0.73        0.68        0.78      0.77

Other Metrics
     m_prec  pos_prec  neg_prec  m_f1  pos_f1  neg_f1
NLP    0.54      0.10      0.98  0.50    0.18    0.83
FIN    0.54      0.10      0.98  0.50    0.18    0.82
MKT    0.55      0.13      0.98  0.54    0.22    0.87
```

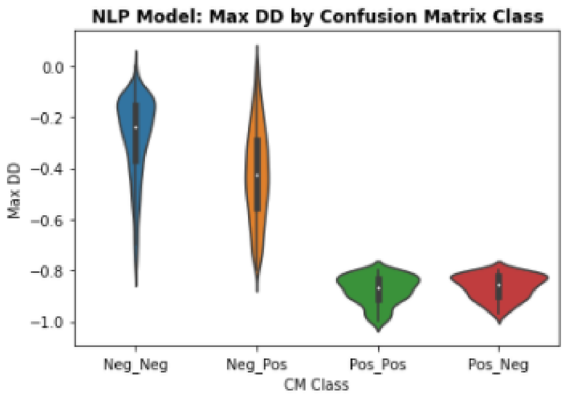
- **Given the subtlety of the information and the first-generation techniques used, the NLP model compares surprisingly well to the baseline models.** It is also encouraging that the testing results are almost identical to those of CV.
- **NLP macro harmonic recall is only 1% off the FIN model** with better negative recall but worse positive recall
- **The MKT model performs the best across all metrics** but is only 1% better than NLP on positive recall
- **Despite its simplicity, it is not surprising that the MKT model performs best as we would expect market variables to incorporate both hard financial information as well as more subtle information** such as governance, company behavior and industry pressures. The lagged nature of the feature also makes sense considering that slow changing structural flaws likely to underpin extreme equity drawdown.
- **While a work in progress, early results suggest that combining the NLP and MKT models (using information only in the features and target set of NLP) provides a model that produces the same macro harmonic recall as MKT** but with positive and negative recall numbers transposed from the current MKT model

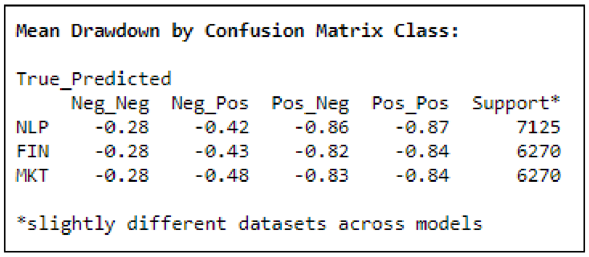
4.2 Analysis

4.2.1 The Cost of Errors

4.2.1.1 ML Perspective (Drawdowns by Confusion Matrix Class)

One of the benefits of having a continuous variable underlying the binary target variable is that we can compute the cost of model errors by calculating the drawdown statistics for each class in the confusion matrix:



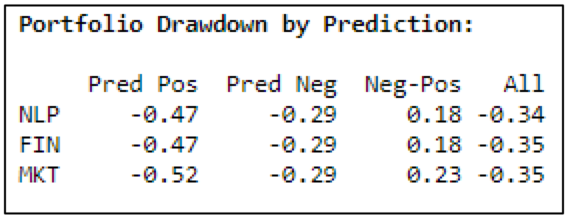


```
Mean Drawdown by Confusion Matrix Class:

True_Predicted
       Neg_Neg  Neg_Pos  Pos_Neg  Pos_Pos  Support*
NLP      -0.28    -0.42    -0.86    -0.87      7125
FIN      -0.28    -0.43    -0.82    -0.84      6270
MKT      -0.28    -0.48    -0.83    -0.84      6270

*slightly different datasets across models
```

- **All models show a similar pattern with the mean drawdown of the false positive set exhibiting far worse drawdown than the true positive set.** This is consistent across all models and suggests that the relatively high proportion of false positives and the associated low precision belie useful information about the classification space.
- While the NLP marginally lags the other models, the violin-plot illustrates the materially different statistics of the drawdown across confusion matrix classes

4.2.1.2 A Business Perspective (Portfolio Performance of Short Predicted Positive vs Long the Rest)

From a business viewpoint, the predicted positive companies would be *valued* lower than the other companies. For an investor (equity or credit), this means one could short (or not invest) in the predicted positive companies and go long the rest. A good metric to get a sense for the return on the strategy would be to look at the difference between the evenly weighted predicted positive portfolio versus (i) a similar portfolio of the rest and (ii) an evenly weighted portfolio of all companies. To be clear, these are not the actual returns (different to drawdowns and not considering time mismatches between max drawdowns) but still gives a good sense of portfolio performance in metrics consistent with the risk-oriented model target:



```
Portfolio Drawdown by Prediction:

     Pred Pos  Pred Neg  Neg-Pos   All
NLP     -0.47     -0.29     0.18 -0.34
FIN     -0.47     -0.29     0.18 -0.35
MKT     -0.52     -0.29     0.23 -0.35
```

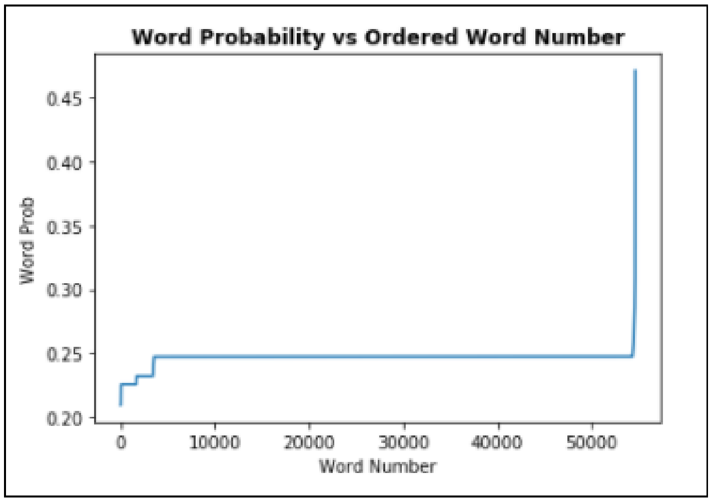
- **While the MKT portfolio performs slightly better, NLP still demonstrates strong “returns”**

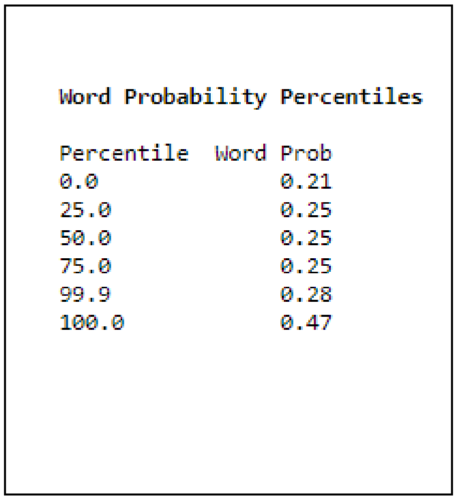
4.3. Interpretation

Interpretation of the model is somewhat challenging, but we can look at the individual words that have the highest probability associated with a positive event. In this section, we limit the analysis of the words to those found only in the holdout set defined by the years 2015 to 2019. This is done to get a better idea of the words explaining the holdout results and not words that might be important based on past events but with no explanatory value on the unseen documents. The caveat throughout this section is that a word can have a high probability while only appearing in one or very few documents that happen to have a positive event. In that respect, not all words will be *generally* predictive.

4.3.1 Word Probabilities

The word probabilities are generated by forming a document matrix where each document only has one unique word and then running this through the model prediction for each annual holdout set. For repeated words, the maximum probability is chosen.





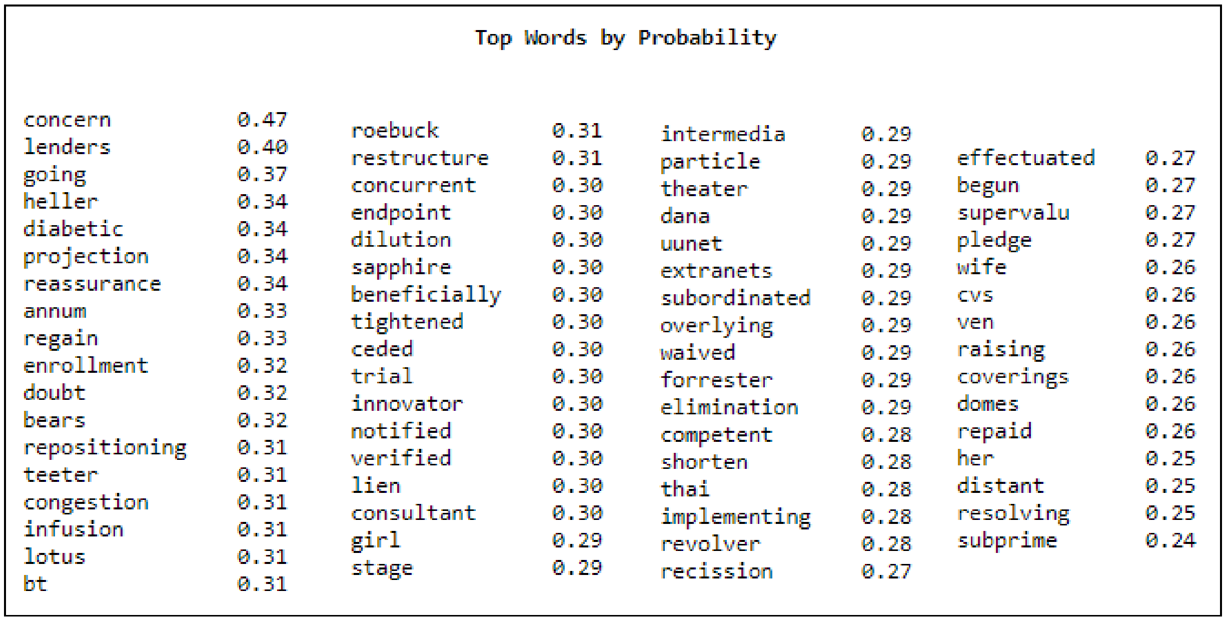
```
Word Probability Percentiles

Percentile  Word Prob
0.0         0.21
25.0        0.25
50.0        0.25
75.0        0.25
99.9        0.28
100.0       0.47
```

- **The maximum single word probability is below 50% and 99.9% of the words are between 20 and 30%. This indicates that it is the combination of words found in the document and not single implicating words that produce documents of high probability.** Given the nuance of the underlying task and the caveat above, this is reassuring.

4.3.2 Top Words

The words below represent the words with the highest probabilities across the holdout sets. They were generated by aggregating the top 20 words from each holdout set. Repeated words take the maximum probability found across the holdout years.

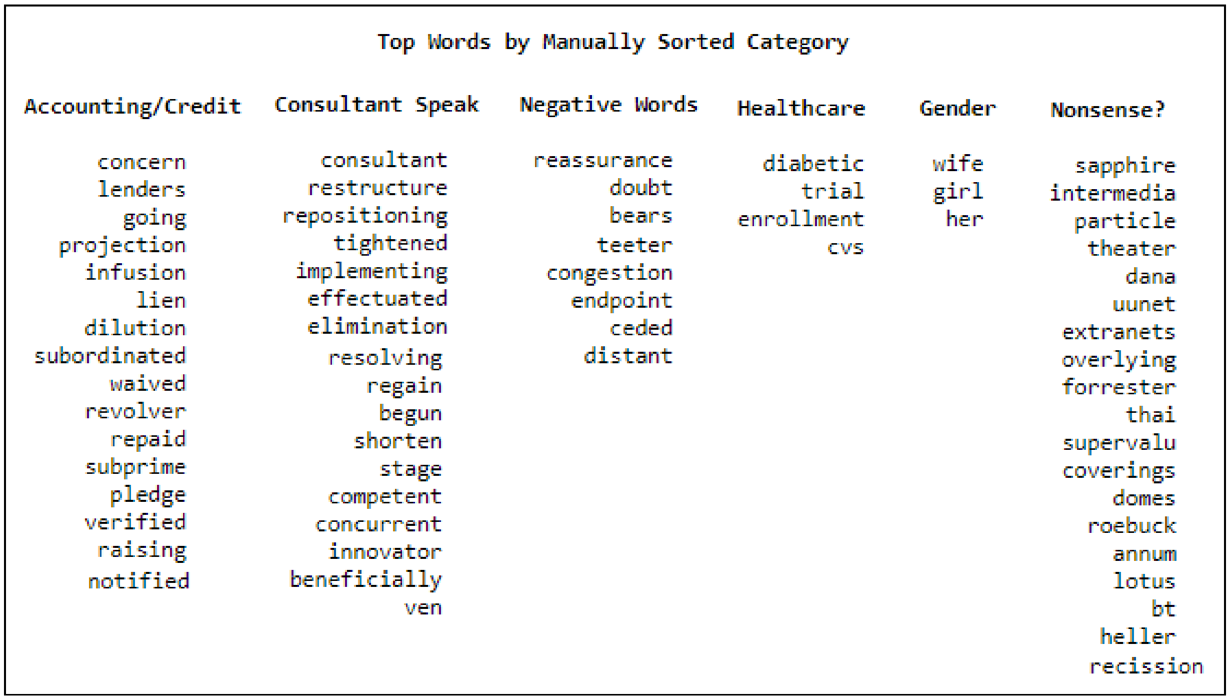
The table below lists these top words in descending order of probability:



Top Words by Probability

concern	0.47	roebuck	0.31	intermedia	0.29	effectuated	0.27
lenders	0.40	restructure	0.31	particle	0.29	begun	0.27
going	0.37	concurrent	0.30	theater	0.29	supervalu	0.27
heller	0.34	endpoint	0.30	dana	0.29	pledge	0.27
diabetic	0.34	dilution	0.30	uunet	0.29	wife	0.26
projection	0.34	sapphire	0.30	extranets	0.29	cvs	0.26
reassurance	0.34	beneficially	0.30	subordinated	0.29	ven	0.26
annum	0.33	tightened	0.30	overlying	0.29	raising	0.26
regain	0.33	ceded	0.30	waived	0.29	coverings	0.26
enrollment	0.32	trial	0.30	forrester	0.29	domes	0.26
doubt	0.32	innovator	0.30	elimination	0.29	repaid	0.26
bears	0.32	notified	0.30	competent	0.28	her	0.25
repositioning	0.31	verified	0.30	shorten	0.28	distant	0.25
teeter	0.31	lien	0.30	thai	0.28	resolving	0.25
congestion	0.31	consultant	0.30	implementing	0.28	subprime	0.24
infusion	0.31	girl	0.29	revolver	0.28		
lotus	0.31	stage	0.29	recission	0.27		
bt	0.31						

To interpret the words, we group them into manually chosen categories:



Top Words by Manually Sorted Category

Accounting/Credit	Consultant Speak	Negative Words	Healthcare	Gender	Nonsense?
concern	consultant	reassurance	diabetic	wife	sapphire
lenders	restructure	doubt	trial	girl	intermedia
going	repositioning	bears	enrollment	her	particle
projection	tightened	teeter	cvs		theater
infusion	implementing	congestion			dana
lien	effectuated	endpoint			uunet
dilution	elimination	ceded			extranets
subordinated	resolving	distant			overlying
waived	regain				forrester
revolver	begun				thai
repaid	shorten				supervalu
subprime	stage				coverings
pledge	competent				domes
verified	concurrent				roebuck
raising	innovator				annum
notified	beneficially				lotus
	ven				bt
					heller
					recission

While the list of very specific and likely spurious words is relatively lengthy, the rest appear to form three natural and unsurprising groups:

- **Accounting / Credit**
- **Consultant Speak / Business Reorganization**
- **Negative sentiment words**

Healthcare words were highly appearing in Data Exploration, but the industry appears to have avoided the worst over the holdout period. Gender is included as it is both amusing and, potentially, insightful. While we do not know the context nor general applicability of these words (likely low), it would not be surprising to find such inappropriate words in the statements of badly governed companies.

5. Further Work

Further work to pursue includes:

- **It would be interesting to see whether ensemble, stacking or joint features engineering across the three models results in better predictions.**
- **Running the model with second generation NLP tools (word2vec seq2seq) better able to parse meaning could considerably improve performance with different interpretive potential**. For example, the fuzzy logic of *word2vec* algebra could lead to some interesting insights.
- **Preprocessing the text could be improved by (i) taking care of the tagging / formatting codes still evident in the word matrix and (ii) removing proper nouns**. Comparing current results with the removal of proper nouns would be particularly interesting as speculative arguments could be made for both sides as better. My own intuition falls on the side of keeping the names as it seems plausible that proper nouns could have predictive capability for our application: common dodgy jurisdictions, questionable auditors, high risk lenders etc.
- **Running the model on a features set of weighted moving average tf-idf vectors per company could prove fruitful.** Following on from the observation of autocorrelation in the market model, we might expect past documents (cumulative information) to be more useful than a 1-year snapshot.
- **Running the model on the change in annual (or annual moving average) company tf-idf as the features set could amp performance**. Abrupt changes in key words or *n-grams* could indicate high short-term risk. It would also be interesting run this change together with the "stock" tf-idf matrix. The intuition being that stock variable tells us which companies are generally risk and the delta indicates higher chances of this being reflected in a nearby positive event.
- **Interpretation could be aided by running an LDA for the features set and seeing which categories are the most predictive.** We already see hints in the results of the top words in the current model and being able to identify categories for leverage, environmental considerations, and gender attitudes could be fascinating with applications in the growth of ESG.
- **Running the model as a multilabel classification model on the drawdown data**. Autocorrelation might be hard to beat but, following from the analysis of drawdowns by confusion matrix class, it would eb interesting to see how NLP predicts annual drawdowns.
- **Calculating actual performance of a portfolio evenly weighting short predicted positive versus long predicted negative.** Again, this follows from the analysis of the holdout results.

Appendix

A1: Baseline FIN Model

The FIN model takes annual financial information from the annual balance sheet *(Sharadar fundamental database*) and uses this to compute various financial ratios consistent with solvency models. Calculations are found in the codebase on GitHub.

The model uses the following common financial ratios as the feature set:

- Net Income / Assets (return on assets)
- Assets / Liabilities (leverage / solvency)
- Net Cash from Operations / Net Income (accruals)
- Net Cash from Operations / Liabilities (debt servicing)
- Net Cash from Operations / Equity (core operational return on equity)
- Net Income / Equity (return on equity)

Gradient boost oversampling was used with the number per sample equal to 3 time the positive set size. Just as in the NLP case, the samples were distributed evenly over time. The model was not optimized and is not meant to represent the best financial features model out there. After all, the NLP model only uses first generation tools and does not represent the most stat of the art NLP model. Rather, the idea is to see if the NLP model can compare to basic financial ratio model.

A2: Baseline MKT Model

Autocorrelation (or momentum) is a common feature of market prices and it is no surprise that we see this evident in the drawdown data- as seen in Data Exploration. It is also intuitive that risky companies, whether related to management style or inherent in the industry, remain so for extended periods. Absolute market prices can vary for a number of macroeconomic reasons and it is more traditional in quantitative finance to take relative rankings. Quartiles are a familiar ranking system and are convenient for the classification problem.

The model uses the following as the features set:

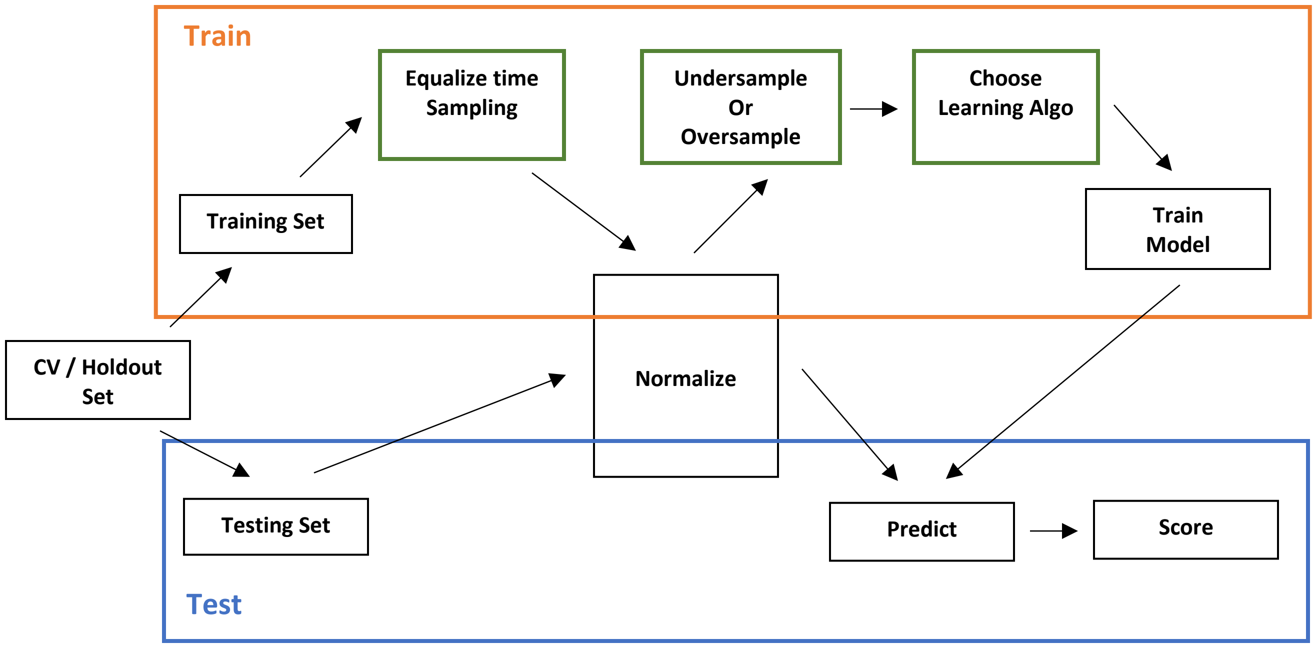
- 1-year lagged drawdown quartile of the company

It is also trained with gradient boost using oversampling (n = 3) and with the same time-series adjustment to the samples as used previously.

A3: Notes on Model Implementation

Full code can be found on GitHub but some notes on the main challenges are provided here:

- **The general process relied on three "work-horse" functions:**
  - Event distribution of classes over time
  - Undersampling
  - Oversampling
- **The main challenge with the class distribution function was that it didn't make sense to simply set the number of events equal per year as this would discard most of the data for the majority class and essentially limit oversampling to drawing with replacement on both sets!** Instead, the ratio of events per year of the majority class was set to be equal to the ratio of events per year of the minority class. In this way, randomly drawing from these separate samples in the same size would preserve equal distribution over time while allowing for the draw of unique data points for the majority class. The ratio was set to be 5 as this appeared practical and in line with the intended maximum oversampling number to be tested in cross-validation.
- **In keeping with the theme of ensuring maximum possible unique data points and managing drawing with or without replacement**, it was important to ensure that undersampling used the *numpy.random.sample()* for both sets and that oversampling used *numpy.random.sample()* for the majority class but *numpy.random.choice()* for the minority class.
- **The biggest programming learning curve was being forced to work with sparse matrices owing to the large size of the tf-idf matrices that were in the order of 40,000x40,000 at times:**
  - **SK-Learn normalize required a sparse *csr matrix* for input and converting the dataframe of the tf-idf matrix and I could not find a conversion option that did not convert through a *todense* numpy matrix as an interim step.** This increased memory usage by a factor of 20 and crashed the system on several occasions. This was overcome by building a function to chunk the dataframe into smaller sizes, converting to a CSR matrix and then stitching these back together through *scipy.sparse.hstack()*. Through this process I also discovered that the SK-Learn models considered allowed one to mix matrix types as *X* was in sparse format and *y* in numpy array format.
  - **In general, the CSR format was quite buggy** with nans mysteriously appearing when taking slices of the tf-idf dataframe. These had to be repaired by searching for null-columns or replacing nan with zero.


Train
Equalize time Sampling
Undersample Or Oversample
Choose Learning Algo
Training Set
Train Model
CV / Holdout Set
Normalize
Testing Set
Predict
Score
Test