Imperial College London

Department of Earth Science and Engineering

MSc in Applied Computational Science and Engineering

Independent Research Project

Model Training and Evaluation

Unlocking Global Capital: AI-Powered Prediction, Access, and Verification of Investor Decision Makers

by

Daniel Bowman

Email: daniel.bowman24@imperial.ac.uk

GitHub username: acse-db1724

Repository: ese-ada-lovelace-2024/irp-db1724

Supervisors:

Antony Sommerfeld

Dr. Yves Plancherel

June 2024

Table of Contents

List of Tables

List of Figures

Abstract

Automating investor contact maintenance is critical for scalable, reliable capital raising in a sector plagued by high turnover and data decay. Existing commercial services like Hunter and ZoomInfo rely on paid lookups and often take days to return results. In contrast, our novel pipeline seamlessly integrates three components—a comprehensive offline template miner that uncovers every common formatting skeleton, a lightweight real-time classifier that instantly predicts the correct template for any new name–domain pair, and on-the-fly third-party deliverability scoring—into one end-to-end system. Performance will be evaluated on held-out contacts, reporting template coverage, prediction accuracy, API precision/recall, and sub-50 ms query latency. This hybrid approach not only outperforms standalone pattern-mining or machine-learning methods but also undercuts costly data-provider fees, democratizing access to fresh, validated investor email information.

1. Introduction to Modelling Phase

This phase of the project focuses on training, evaluating and improving the machine learning models for email template prediction for LP investors. The goal is to accurately infer local-part structure for a given investor-firm name pair, using engineered features derived from both investor name and firm level metadata.

Rather than treating the task as a multi-class classification problem, we have reframed the solution as a ranking formulation – where the model learns to prioritize the correct template among a fixed candidate template list. This aligns better with the real world use case and allows use to provide multiple possible templates with associated confidence scores.

Throughout this phase, performance is monitored using ranking metrics such as Accuracy@1, Recall@3, and Mean Reciprocal Rank (MRR), with both global and domain-specific evaluations. Insights from this modelling phase guide the next stage of augmentation, deployment, and potential model ensembling.

2. Initial LightGBM Model Development

The initial modelling experiments used a LightGBM classifier with a binary objective to predict whether a given candidate template is the correct one for a given investor-firm name pair. A feature matrix was constructed using engineered features from the previous phase, where each row represented a investor, firm, candidate template triple for each of the 404 mined candidate templates.

2.1. Batching Strategy and Memory Constraints

Due to the size of the feature matrix (over 29 million rows with each of the ~80,000 investors) it was not feasible to load all of the data in to my laptops RAM. To address this, a batch-based python generator was developed to stream portions of the feature matrix into the memory during training. Batches were built around the ‘clean_row_id’ from the cleaned LP data to ensure all candidate templates for a given investor were loaded together. This preserved group structure for correct scoring and evaluation. This was unsuited for LightGBM as gradient boosted trees are designed to fit to the entire dataset – rather than train

incrementally to batched data like some deep learning models. Eventually it was rectified by moving onto Google Collab and utilizing the computational power of the VM services.

2.2. Class Imbalance and Scale Weights

Initial experiments used a 70/10/20 split for training, validation and testing based on unique clean LP row ids. Model performance was evaluated using precision, recall and F1 score. However, early results showed a significant imbalance in recall and precision with the model preferring recall by predicting too many positives at the expense of ranking precision. This was likely due to class imbalance due to our single label amongst 404 candidates.

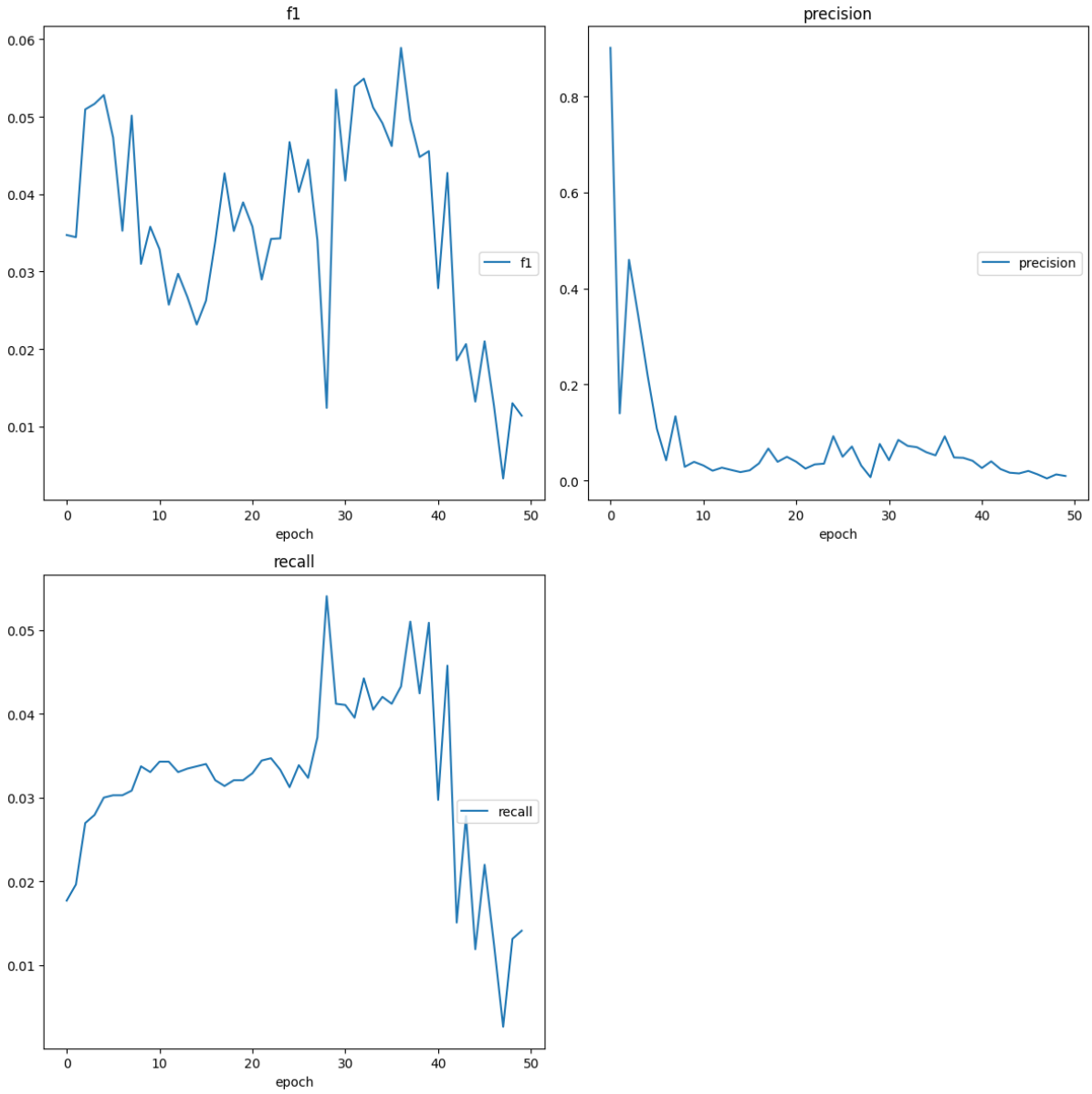

Figure 1: Initial Training

To counteract this, ‘scale_pos_weight’ (a LightGBM hyperparameter) was introduced to rebalance the loss function. This adjustment improved recall slightly but did not yield significant gains in overall ranking quality. At this point, it was clear that binary classification was not well-suited for the task and led to inconsistent predictions where many incorrect templates were assigned high probabilities, especially for firms with diverse naming conventions or infrequent templates.

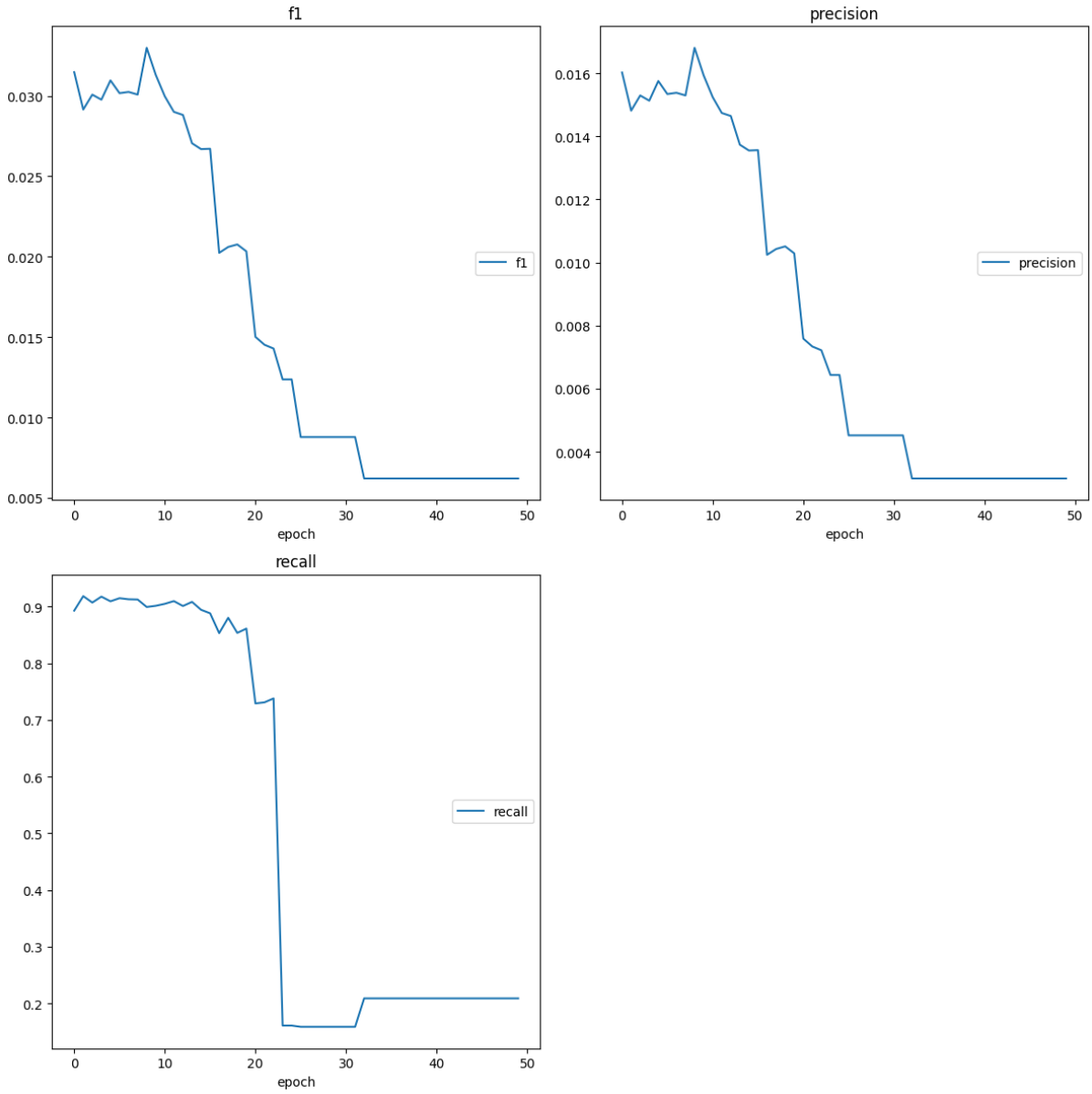

Figure 2: Scale Pos Weight

This insight prompted a shift to LambdaRank, a ranking objective designed for exactly this scenario.

3. Transition to Ranking Objective

Given that email template prediction is inherently a ranking task (with one correct template per investor and hundreds of plausible candidates) it became clear that a pointwise binary objective was inappropriate. Instead, we transitioned to a pairwise ranking objective using LightGBM's LambdaRank, which optimizes for ordering candidates rather than absolute scores.

3.1. Metric Selection and Evaluation Setup

To evaluate the success of our new objective function, we introduced three new metrics to replace precision, recall and F1-score:

- Accuracy@1: a percentage measure of how often the correct template is ranked in the top slot.
- Recall@3: a percentage measure of how often the correct template is ranked in the top three slots.
- Mean Reciprocal Rank (MRR): captures how early in the ranked list the correct template appears, penalizing deeper placements.

These metrics were tracked over validation batches, giving live feedback after each update.

3.2. Model Training and Live Plotting

Training was initially done in small batches to accommodate memory limits. However, LightGBM's ranking models are not designed for incremental updates, as each tree must be fit with a global view of the gradients. This caused significant instability, with early rounds yielding strong gains that quickly plateaued or even collapsed in later updates.

We introduced live plotting of ranking metrics using livelossplot, which made it easy to observe this instability in real time. The model often oscillated between high recall and poor precision, unable to consistently rank templates without overfitting to specific patterns.

This instability prompted us to switch to full batch training to fit the model to the entire data. We also introduced an adaptive learning rate decay as well as experimenting with gradient stability constraints (using LightGBM hyperparameters such as 'max_depth').

4. Template Pruning and Normalization Strategy

Initial mining produced 404 unique email structures, but further analysis revealed that many of these were extremely low coverage and effectively redundant. Variants of the same template – differing only in normalization which should arguably be done by default – accounted for a significant portion of the long tail.

In order to address this, the variants were removed from the template encoding code and the SQL tables were regenerated to account for the new template list such that we can regenerate the feature matrices with the pruned candidate templates.

Subsequent model training using the pruned candidate set demonstrated higher ranking precision, confirming the value of template normalization as a preprocessing step.

```
[1]     train's ndcg@1: 0.383882        train's ndcg@3: 0.670038        val's ndcg@1: 0.377205  val's ndcg@3: 0.664331
Training until validation scores don't improve for 20 rounds
[2]     train's ndcg@1: 0.386607        train's ndcg@3: 0.674788        val's ndcg@1: 0.381213  val's ndcg@3: 0.670745
[3]     train's ndcg@1: 0.386233        train's ndcg@3: 0.675629        val's ndcg@1: 0.380545  val's ndcg@3: 0.670931
[4]     train's ndcg@1: 0.386474        train's ndcg@3: 0.67537 val's ndcg@1: 0.379743  val's ndcg@3: 0.670368
[5]     train's ndcg@1: 0.386474        train's ndcg@3: 0.675344        val's ndcg@1: 0.379743  val's ndcg@3: 0.670368
[6]     train's ndcg@1: 0.375678        train's ndcg@3: 0.671419        val's ndcg@1: 0.374933  val's ndcg@3: 0.66868
[7]     train's ndcg@1: 0.375678        train's ndcg@3: 0.671419        val's ndcg@1: 0.374933  val's ndcg@3: 0.66868
[8]     train's ndcg@1: 0.386607        train's ndcg@3: 0.675424        val's ndcg@1: 0.381213  val's ndcg@3: 0.671103
[9]     train's ndcg@1: 0.386634        train's ndcg@3: 0.675462        val's ndcg@1: 0.381213  val's ndcg@3: 0.670998
[10]    train's ndcg@1: 0.375678        train's ndcg@3: 0.671419        val's ndcg@1: 0.374933  val's ndcg@3: 0.66868
[11]    train's ndcg@1: 0.386607        train's ndcg@3: 0.675424        val's ndcg@1: 0.381213  val's ndcg@3: 0.671103
[12]    train's ndcg@1: 0.386607        train's ndcg@3: 0.675424        val's ndcg@1: 0.381213  val's ndcg@3: 0.671103
[13]    train's ndcg@1: 0.375838        train's ndcg@3: 0.671478        val's ndcg@1: 0.376403  val's ndcg@3: 0.669222
[14]    train's ndcg@1: 0.386634        train's ndcg@3: 0.675462        val's ndcg@1: 0.381213  val's ndcg@3: 0.670998
[15]    train's ndcg@1: 0.375678        train's ndcg@3: 0.671419        val's ndcg@1: 0.374933  val's ndcg@3: 0.66868
[16]    train's ndcg@1: 0.386607        train's ndcg@3: 0.674886        val's ndcg@1: 0.381213  val's ndcg@3: 0.670902
[17]    train's ndcg@1: 0.386607        train's ndcg@3: 0.675424        val's ndcg@1: 0.381213  val's ndcg@3: 0.671103
[18]    train's ndcg@1: 0.375678        train's ndcg@3: 0.671419        val's ndcg@1: 0.374933  val's ndcg@3: 0.66868
[19]    train's ndcg@1: 0.386607        train's ndcg@3: 0.674886        val's ndcg@1: 0.381213  val's ndcg@3: 0.670902
[20]    train's ndcg@1: 0.386607        train's ndcg@3: 0.675424        val's ndcg@1: 0.381213  val's ndcg@3: 0.671103
[21]    train's ndcg@1: 0.375838        train's ndcg@3: 0.671478        val's ndcg@1: 0.376403  val's ndcg@3: 0.669222
[22]    train's ndcg@1: 0.386474        train's ndcg@3: 0.675403        val's ndcg@1: 0.379743  val's ndcg@3: 0.670455
Early stopping, best iteration is:
[2]     train's ndcg@1: 0.386607        train's ndcg@3: 0.674788        val's ndcg@1: 0.381213  val's ndcg@3: 0.670745
```

Figure 3: Pruned Training Results

5. Stratified Error Analysis

5.1. Domain-Level Performance Breakdown

After baseline training and evaluation, performance was stratified by email domain to identify segments where the model underperformed. Each domain's predictions were grouped by row id and ranking metrics were computed by group. This revealed a bimodal distribution where some domains achieved >80% Accuracy@1 many others fell below 20%.

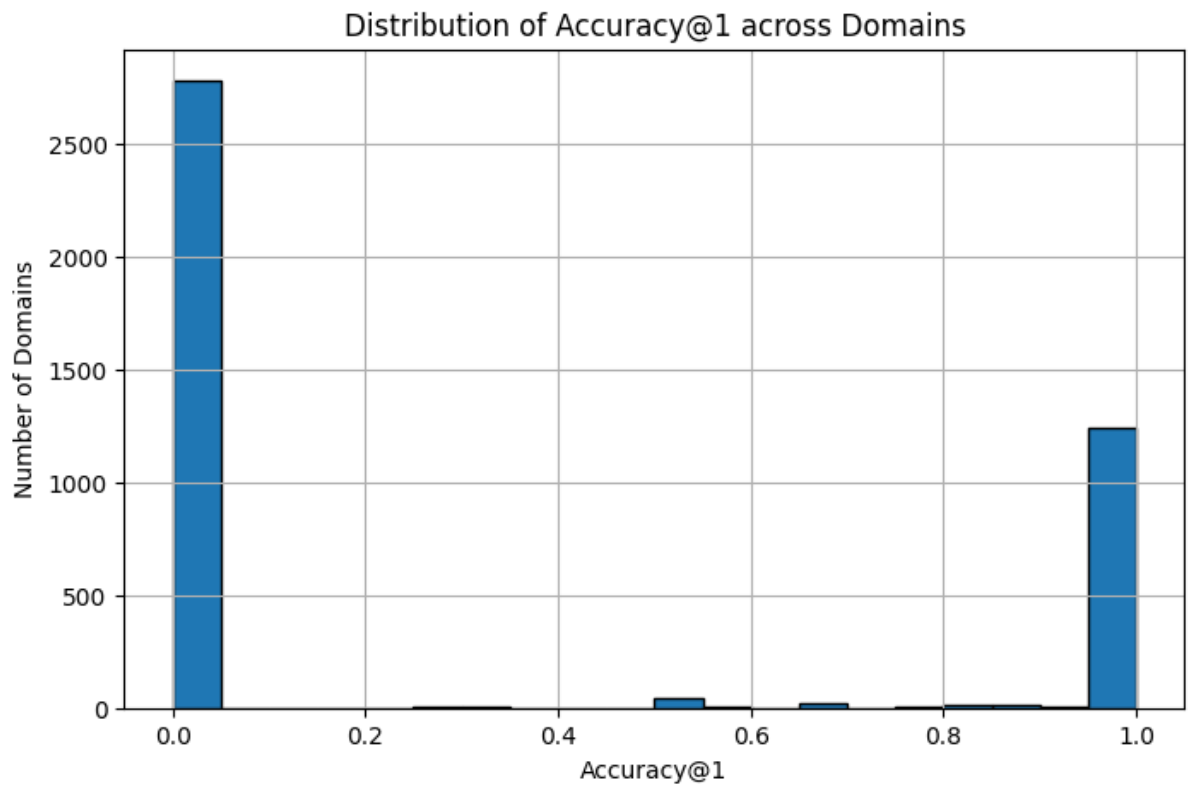

Figure 4: Accuracy Across Domains

Domains with low performance were found to have sparse historical data, high template diversity and shared infrastructure (@gmail.com or subsidiaries).

```
Num Investors Stats (from firm_template_map):
count    2785.000000
mean        9.628366
std        31.867330
min         1.000000
25%         2.000000
50%         5.000000
75%         9.000000
max      1344.000000
Name: num_investors, dtype: float64

Diversity Ratio Stats (from firm_template_map):
count    2785.000000
mean        0.396377
std         0.312849
min         0.008185
25%         0.153846
50%         0.307692
75%         0.500000
max         1.000000
Name: diversity_ratio, dtype: float64
```

Figure 5: Failing Firm Stats

5.2. Template Diversity and Failure Correlation

Domains associated with firms that used many distinct email templates were found to have significantly lower Accuracy@1. When the correct template varied significantly within a single firm, the model was forced to rank among many plausible templates with little firm level statistics.

A scatter plot was used to prove this, showing that high template diversity in firms was correlated with higher failure rates, especially with firms that had fewer than 5 investors.

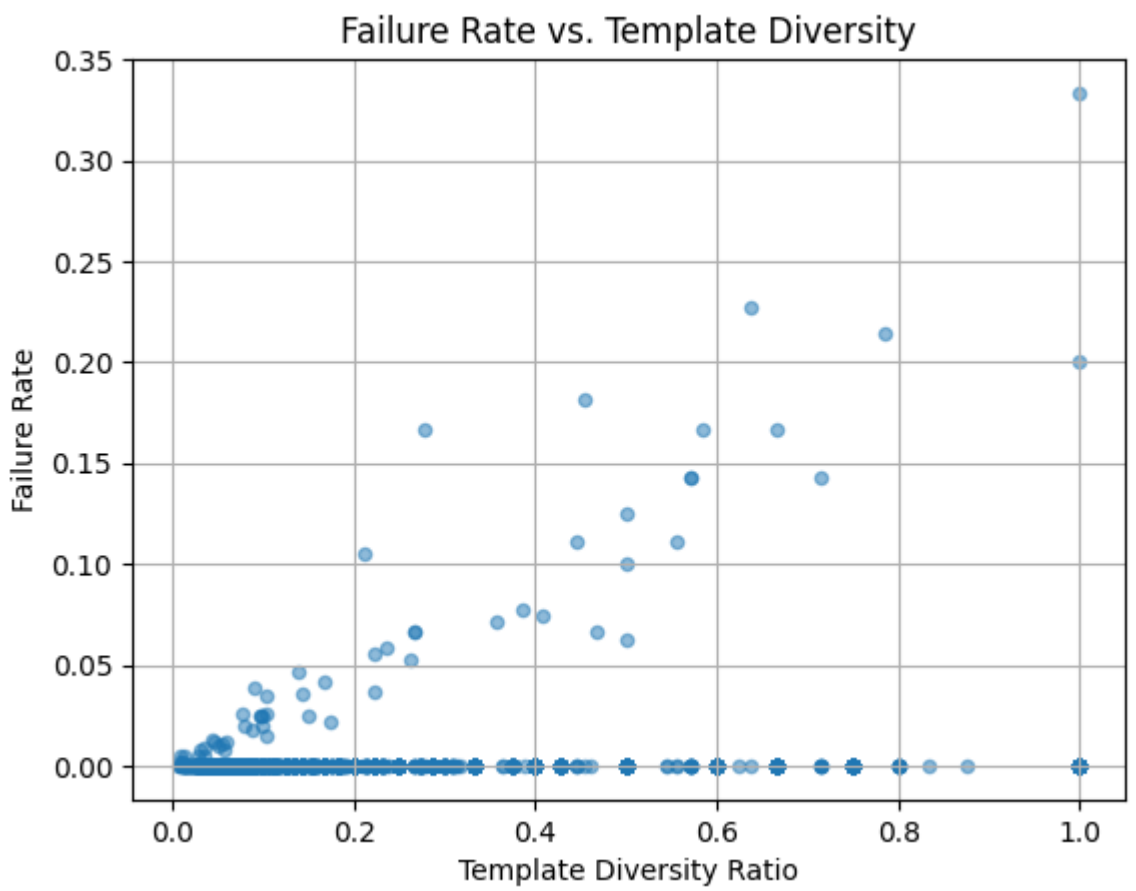

Figure 6: Failure Rate vs. Template Diversity Scatter Plot

5.3. Investor Count Band Analysis

To explore the role of sample size, validation performance was stratified by firm size using investor count bands. This revealed a clear positive correlation between investor count and ranking accuracy.

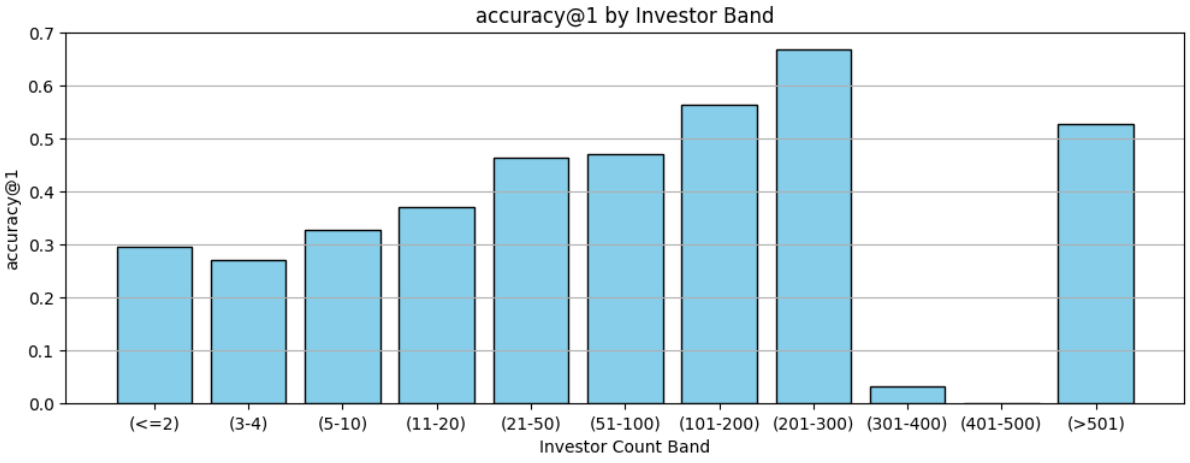

Figure 7: Accuracy Banded by Investor Count

This suggests that the model was overfitting to large firms and underperforming on small ones indicating a need for data augmentation on low-investor firms.

5.4. Feature-Level Disparity in Failure Cases

Failure cases were also analysed based on name structure flags (e.g. ‘has_middle_name’, ‘has_multiple_last_names’, ‘has_nfkd_normalized’). Among the failing validation examples, these features were overrepresented.

	feature	total_occurrences	percent_in_correct	percent_in_wrong
0	is_shared_infra	166	0.286	0.400
1	firm_is_multi_domain	148	0.714	0.349
4	has_nickname	111	0.143	0.268
8	has_multiple_last_names	7	0.143	0.015
3	has_nfkd_normalized	2	0.000	0.005
5	has_multiple_first_names	2	0.000	0.005
6	has_middle_name	1	0.000	0.002
2	has_german_char	0	0.000	0.000
7	has_multiple_middle_names	0	0.000	0.000

Figure 8: Feature Level Stats In Failing Cases

This confirmed that complex naming conventions significantly reduce prediction accuracy. It also motivated the idea of segmented models—training separate predictors for standard vs. complex names to isolate edge cases.

6. Synthetic Padding Strategy

Stratified evaluation revealed that firms with fewer than 5 investors suffered from significantly lower performance. These small sample firms often lacked consistent template usage, making it difficult for the model to generalize. To address this, a synthetic augmentation strategy was adopted to artificially increase the sample size of low investor firms without disrupting global statistics.

Each low investor firm was assigned a synthetic profile. This profile captured the list of templates used by a low investor firm as well as the observed probability that a given template would be used within that firm. We also captured the

conditional probability that a structural flag was used in that firm with that template.

This enabled controlled sampling of new synthetic investors with name–template feature combinations reflective of the original firm.

6.1. Sampling Process and Drift Minimization

For each selected firm, 10-20 synthetic investors were generated. A template was chosen based on the given firm level support, name structures were appropriately sampled and a unique name was given. After generation, feature distributions were recalculated and compared to the original profile to ensure minimal statistical drift. A post-padding drift audit confirmed that sampling stayed faithful to original patterns, with low variance in template selection probabilities and structural feature distributions.

6.2. Impact on Model Performance

After padding, a new feature matrix was generated and the model was retrained on the full dataset on Google Collab. The new results drastically improved with ~85% Accuracy@1 and ~93% Recall@3.

```
[0] train ndcg@1: 0.88226  train ndcg@3: 0.94278  val ndcg@1: 0.71994  val ndcg@3: 0.84847
[1]     train's ndcg@1: 0.882255        train's ndcg@3: 0.942779        val's ndcg@1: 0.719942  val's ndcg@3: 0.848469
Training until validation scores don't improve for 20 rounds
[2]     train's ndcg@1: 0.888038        train's ndcg@3: 0.946874        val's ndcg@1: 0.739981  val's ndcg@3: 0.861974
[3]     train's ndcg@1: 0.895414        train's ndcg@3: 0.955466        val's ndcg@1: 0.766191  val's ndcg@3: 0.892616
[4]     train's ndcg@1: 0.895566        train's ndcg@3: 0.955594        val's ndcg@1: 0.767092  val's ndcg@3: 0.893209
[5]     train's ndcg@1: 0.906558        train's ndcg@3: 0.961495        val's ndcg@1: 0.803148  val's ndcg@3: 0.913123
[6]     train's ndcg@1: 0.906558        train's ndcg@3: 0.961495        val's ndcg@1: 0.803148  val's ndcg@3: 0.913123
[7]     train's ndcg@1: 0.906558        train's ndcg@3: 0.961468        val's ndcg@1: 0.803148  val's ndcg@3: 0.913149
[8]     train's ndcg@1: 0.914675        train's ndcg@3: 0.965774        val's ndcg@1: 0.832201  val's ndcg@3: 0.929222
[9]     train's ndcg@1: 0.915017        train's ndcg@3: 0.965944        val's ndcg@1: 0.833241  val's ndcg@3: 0.929707
[10]    train's ndcg@1: 0.915073        train's ndcg@3: 0.965981        val's ndcg@1: 0.833241  val's ndcg@3: 0.92962
[10] train ndcg@1: 0.91498  train ndcg@3: 0.96600  val ndcg@1: 0.83393  val ndcg@3: 0.92995
[11]    train's ndcg@1: 0.914977        train's ndcg@3: 0.966   val's ndcg@1: 0.833934  val's ndcg@3: 0.929945
[12]    train's ndcg@1: 0.914977        train's ndcg@3: 0.965993        val's ndcg@1: 0.833934  val's ndcg@3: 0.930024
[13]    train's ndcg@1: 0.914938        train's ndcg@3: 0.965998        val's ndcg@1: 0.833934  val's ndcg@3: 0.929989
[14]    train's ndcg@1: 0.914938        train's ndcg@3: 0.96601 val's ndcg@1: 0.833934  val's ndcg@3: 0.929989
[15]    train's ndcg@1: 0.914938        train's ndcg@3: 0.966045        val's ndcg@1: 0.833865  val's ndcg@3: 0.929998
[16]    train's ndcg@1: 0.914938        train's ndcg@3: 0.966045        val's ndcg@1: 0.833865  val's ndcg@3: 0.929998
[17]    train's ndcg@1: 0.914938        train's ndcg@3: 0.966068        val's ndcg@1: 0.833865  val's ndcg@3: 0.930015
[18]    train's ndcg@1: 0.916395        train's ndcg@3: 0.966666        val's ndcg@1: 0.835182  val's ndcg@3: 0.930891
[19]    train's ndcg@1: 0.920665        train's ndcg@3: 0.968312        val's ndcg@1: 0.840105  val's ndcg@3: 0.932901
[20]    train's ndcg@1: 0.921995        train's ndcg@3: 0.968853        val's ndcg@1: 0.842186  val's ndcg@3: 0.933968
[20] train ndcg@1: 0.92264  train ndcg@3: 0.96913  val ndcg@1: 0.84295  val ndcg@3: 0.93447
[21]    train's ndcg@1: 0.92264 train's ndcg@3: 0.969132        val's ndcg@1: 0.842948  val's ndcg@3: 0.93447
...
[73]    train's ndcg@1: 0.930406        train's ndcg@3: 0.972446        val's ndcg@1: 0.850159  val's ndcg@3: 0.938222
[74]    train's ndcg@1: 0.930406        train's ndcg@3: 0.972446        val's ndcg@1: 0.850159  val's ndcg@3: 0.938222
Early stopping, best iteration is:
[54]    train's ndcg@1: 0.930167        train's ndcg@3: 0.97246 val's ndcg@1: 0.849951  val's ndcg@3: 0.938646
```

Figure 9: Padded Training Results

Importantly, performance gains came without sacrificing accuracy on large firms. This suggested the synthetic data was statistically compatible with real investor distributions and did not introduce overfitting artifacts.

7. Benchmarking Against Prompt Only Zero-Shot LLM

To understand the value of our structured LightGBM model, we implemented a baseline zero-shot inference pipeline using OpenAI's gpt-4. The goal was to simulate a naive fallback system: given a name and firm, could an LLM reliably guess the correct email address—without training—based solely on prompt engineering?

This experiment provides a comparative benchmark on model accuracy without feature engineering, reliability of zero shot prediction and the feasibility and cost of using LLMs for real time email resolution.

A dataset of 500 investor–firm–email triples was sampled from the cleaned LP dataset. The LLM was queried with prompts like:

“You are an expert at guessing professional emails.

Given:

- Investor Name: Michael Smith

- Firm Name: Example Capital

Please guess the 3 most likely professional email addresses for this investor at this firm.”

Predictions were scored using the same metrics as our trained model: Accuracy@1, Recall@3, and Mean Reciprocal Rank (MRR).

7.1. Initial Results (Name + Firm Only)

With no additional information, performance was poor. The model achieved ~12% in Accuracy@1 and ~19% in Recall@3.

This confirmed that without structural signals or past data, the LLM struggles to resolve email patterns—especially for firms with unusual templates or investor names requiring normalization.

7.2. Adding Template Hints

We then added historical template formats mined from pattern mining as an additional context in the prompt (e.g., first.last@domain.com, f_last@domain.com). Surprisingly, performance did not significantly improve.

Even when guiding the model toward known template structures, accuracy remained well below 20%, confirming that static prompt-based retrieval was not sufficient.

7.3. Takeaways and Limitations

At 500 samples per batch, even minimal LLM use becomes cost prohibitive. Perhaps using a free model LLAMA or downloading a subset of DeepSeeks network to a local drive would mitigate this but at the cost of less powerful model.

LLM queries using OpenAI's API are on average takes about 3 seconds which completely break our sub 50ms latency goals.

Thus, LLM-based fallback is not viable for production use in this domain. While future fine-tuning or hybrid RAG architectures might improve results, structured learning on a domain-specific feature matrix is decisively more performant, efficient, and explainable for email prediction.

8. CatBoost Benchmarking

Further benchmarking was done against another gradient boosting alternative to see if it would outperform LightGBM. Catboost is known for its robustness with categorical data, automatic handling of missing values, and proprietary ranking objectives like YetiRank.

The setup is exactly the same as LightGBM, with the same engineered features constructed into the same matrix and validated using the same metrics.

8.1. Results

The CatBoost model significantly outperformed prior LightGBM baselines on the same data:

- Accuracy@1: 0.9062
- Recall@3: 0.9969

- MRR: 0.9540
- YetiRank loss: ~0.984 (near-perfect ranking quality on validation)

These metrics represent a notable leap in ranking performance, particularly in:

- Top-1 accuracy (from 0.8616 with LightGBM to 0.9062)
- Mean Reciprocal Rank (from 0.9273 to 0.9540)

Not only was the model more accurate at picking the best template, but it was also better calibrated to place correct candidates higher in the list.

8.2. Implications

This result strongly suggests that CatBoost is better suited for this task than LightGBM in its current configuration. This is likely because of YetiRanks gradient aware pairwise ranking, which may better model subtle template distinctions in template usage. CatBoost is also known for its improved handling of sparsity and rare templates allowing for better generalization of small groups.

9. Segmented Modelling by Name Complexity

Despite improved gains from data augmentation, a consistent pattern of complex named investors (multiple middle names, multiple last names) being overrepresented in the low performing cases.

These cases often required more nuanced template logic involving special tokens. However, such templates had lower support, and the model struggled to distinguish them under a unified prediction space.

To address this, the dataset was explicitly split into two partitions; a standard name set which featured single first name and last name investors; a complex name set which had any of the structural name flags set.

This separation allowed for a reduced candidate template set (as it was split between the two models) and different augmentation patterns across the different models.

9.1. Template Set Partitioning and Feature Regeneration

After partitioning, it was found that the standard name set accounted for ~95% of our dataset, whereas complex names only accounted for ~5%.

Due to this imbalance, aggressive padding was done on the complex subset to ensure sufficient training examples.

Templates unique to complex names were often more diverse and structurally rich, requiring the model to learn from more fragmented and noisy patterns. In contrast, the standard set exhibited high template consistency, often dominated by 2–3 common forms per firm.

9.2. Model Training and Cross-Comparison

Each name group was trained with the same LambdaRank LightGBM objective and evaluated using consistent ranking metrics.

The standard model achieved ~86% in Accuracy@1 and ~99% in Recall@3. Similarly, the complex model achieved ~85% in Accuracy@1 and ~96% in Recall@3. By doing a weighted average (accounting for the 95/5 split of the data between the models) we can quantify the we gained marginal gains ~1% in Accruacy@1. This signifies a useful improvement in our model design but ultimately highlights the models difficulty to generalise to diverse template firms.

Metric	Unified Model	Segmented (Weighted)	Improvement
Accuracy@1	85%	85.96%	+0.96%
Recall@3	99.4%	99.46%	+0.06%
MRR	92.06%	92.59%	+0.53%

Table 1: Cross Comparison of Unified Model and Segmented

10. Hyperparameter Tuning and Final Model Selection

To ensure fairness and optimal performance, we performed systematic hyperparameter tuning using Optuna for both LightGBM and CatBoost models on each dataset.

Each model was tuned for Recall@3 by minimizing 1 – recall on a fixed validation set. For each study, 50 trials were run with randomized search space sampling. Early stopping was used to prevent overfitting and the best parameters were saved for full retraining and evaluation.

10.1. Results: Standard Name Set

Model	Accuracy@1	Recall@3	MRR
LightGBM (tuned)	90.23%	99.64%	94.93%
CatBoost (tuned)	91.47%	99.78%	95.91%

Table 2: Tuned Standard Set Results

CatBoost again outperformed LightGBM by a clear margin, confirming earlier observations even after full parameter optimization. The model achieved >99% Recall@3 with over 91% top-1 accuracy on the validation set.

10.2. Results: Complex Name Set

Model	Accuracy@1	Recall@3	MRR
LightGBM (tuned)	79.53%	95.67%	87.04%
CatBoost (tuned)	83.38%	97.63%	89.99%

Table 3: Tuned Complex Set Results

Here, the performance gap widened. CatBoost not only generalized better on noisy, rare, or structurally diverse name cases but also ranked correct templates more effectively. The +3.9% Accuracy@1 gain and +2% Recall@3 gain are significant for this high-noise segment.