Can Language Models Accelerate Prototyping for Non-Language Data? Classification & Summarization of Activity Logs as Text

José González-Brenes
jose.gonzalezbrenes@airbnb.com
Airbnb
San Francisco, California, USA

ABSTRACT

We report our efforts in developing machine learning models intended to boost the efficiency of operations agents in two-sided marketplaces. These agents are crucial for customer service and quality control, often working with detailed data like activity logs—which are detailed record of user interactions within the marketplace.

We propose transforming structured data (activity logs) into a more manageable text format and then leveraging modern language processing toolkits. These toolkits rely on powerful Large Language Models (LLMs). When these toolkits are used for modeling text, they typically are sophisticated enough to reduce the time machine learning experts spend in feature engineering or designing a custom network architecture.

We demonstrate our approach by summarizing and classifying activity logs. Although the motivation for our work is accelerating the machine learning lifecycle by reducing feature and neural network engineering; our preliminary results suggest that our framework may outperform by 80% the average precision of a similar model that was designed relying heavily on feature engineering. Because our approach relies less on time-consuming tasks for machine learning experts, we believe that it is a promising avenue for further research.

CCS CONCEPTS

• **Computing methodologies** → **Information extraction**; *Neural networks*; • **Information systems** → *Content analysis and feature selection.*

KEYWORDS

activity log processing, structured data summarization, prototyping

ACM Reference Format:
José González-Brenes. 2024. Can Language Models Accelerate Prototyping for Non-Language Data? Classification & Summarization of Activity Logs as Text. In *Proceedings of TSMO 2024 Workshop on Two-sided Marketplace Optimization: Search, Pricing, Matching & Growth in conjunction with KDD Conference (KDD 2024) (KDD-TSMO '24).* ACM, New York, NY, USA, 7 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

In this paper, we report our results of experimenting with a rapid prototyping technique for machine learning models that input structured data. Structured data consists of blocks of data organized by predefined structures to compress recurring information [13]. Our focus is on activity logs, a common data source for two-sided marketplaces. Activity logs record user interactions with the marketplace, including session and device info, along with activities like searching or transactions. Activity logs are considered structured data because a *single instance* contains both (i) multiple rows (for each session or activity), and (ii) multiple columns with rich types (numeric, string, list). No personally identifiable information was read or processed in the analysis presented here.

Two-sided marketplaces often create *operation cases* for users who meet certain criteria. In this work, we focus on operation cases that include activity logs. These logs are sent to, and acted upon by, employees often referred to as *operations agents.* In Figure 1, we depict an example of this workflow, which is the focal point of this study. We do not describe the specifics of the processes that require operation agents to use activity logs. Instead, we provide a high level description of examples of how they are used. The operations agents receive activity logs that may consist of many rows and columns. Each row in the activity log represents a user session. These sessions aren't necessarily continuous; a user can log in on a device one day and continue using that same login in future interactions. Each column in the activity log represents an attribute of the session. In Table 2 we give a high-level view of the columns in the activity logs; these include the "activities" of the session, which describe events in the user journey, such as sign-up and login. Figure 1 shows that the agents are required to identify the rows (*i.e.*, sessions) of the activity log as per business guidelines.

In Example 1 of Figure 1, we see that if one or more rows are selected, the case is marked for follow-up by another business process. The details of this business process are not described here. In Example 2, we see that if no rows are selected, the workflow is complete. This workflow can be highly costly due to its scale and the extensive information contained in the activity logs. Therefore, developing Artificial Intelligence (AI) tools to enhance agent decision-making efficiency could be beneficial. These AI tools are under active research, and it's not possible to determine *a priori* if they will indeed enhance the speed and quality of operations agents' work. Our motivation is to create tools that assist operation agents, not to replace them. Given that optimizing human workflows is an experimental iterative process, investing excessive time in feature engineering or bespoke neural network design may prove impractical and inefficient at the early stages of a project.

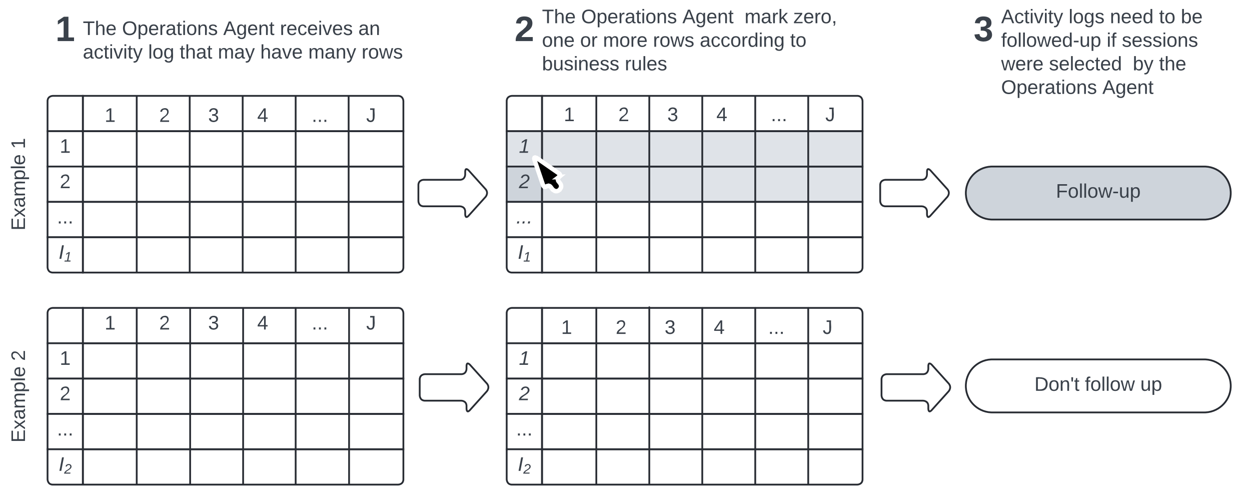
1 The Operations Agent receives an activity log that may have many rows
2 The Operations Agent mark zero, one or more rows according to business rules
3 Activity logs need to be followed-up if sessions were selected by the Operations Agent
Example 1
Example 2
Follow-up
Don't follow up

Figure 1: Two sample cases in a real-life workflow. Agent see an activity log where each row is session, and columns represent a type of information (such as timestamp, or list of activities). Operations agents are required to investigate activity logs and select certain rows based on specific business criteria. If they select any non-zero number of rows, then an additional business process needs to be followed.

In recent years, we've seen significant advancements in the developing of tools that enable rapid prototyping of machine learning applications. For instance, `sklearn` [3] is a popular library that includes many machine learning algorithms. With the rising interest in Generative AI, other toolkits like `hf-transformers` [16] have been proposed for natural language processing. Unfortunately, leveraging these toolkits to activity logs data is not straightforward. In general, structured data like activity logs require that a machine learning specialist designs bespoke feature engineering strategies, neural network architectures, or both.

In this paper, we present our findings on a straightforward but surprisingly effective method of leveraging existing popular natural language toolkits to structured data. The motivation for our work is to develop machine learning tools to support and improve the efficiency of operations agents. We believe that the techniques described here could be used to accelerate the machine learning lifecycle by reducing feature and neural network engineering. Even though our focus is on faster prototyping, our results suggest that our approach can potentially outperform more traditional methods that rely heavily on feature engineering. We demonstrate our approach with two use cases to aid operations agents in dealing with activity log data:

- Activity log summarization: involves the extraction of rows (which represent user sessions) from the activity log. The main objective is to assist in flagging the rows that agents may need to inspect. This corresponds to step 2 in Figure 1.
- Activity log classification: these are models that can flag activity logs that need follow-ups. This corresponds to step 3 in Figure 1.

The rest of this paper is organized as follows. § 2 describes our approach of prototyping activity log models using existing LLM architectures. § 3 describes the two-sided marketplace data used for our experiments. § 4 provides empirical evaluation. § 5 relates our work with prior literature. Finally, § 6 provides concluding remarks.

2 METHODS

Our method involves converting each activity log into a text format (§ 2.1). This step is crucial to our methodology as it modifies the structured data in such a way that it can be processed by LLM architectures. Because of this, we pre-train an activity log model (§ 2.2) — we use an encoder architecture known as BERT [6]. BERT was proposed for natural languages and it isn't designed designed to handle structured data or additional features. Herein lies our contribution: a method to map structured data as text, enabling the use of powerful LLM architectures such as BERT to work with structured data sources like activity logs. The choice of BERT is motivated by its pioneering use of the transformer architecture [15], which allows for enhanced understanding of textual data. Although it's a powerful model, it's also relatively quick to train, offering a balance between performance and computational efficiency. Future research could experiment with larger, more complex architectures. One of BERT's standout features is its use of deep bidirectional representations from unlabeled text. It does this by considering both the left and right context in all layers of the model, which helps it gain a comprehensive understanding of the text. After pre-training our model with BERT, we proceed to fine-tune it for specific tasks. These tasks include classification (§ 2.3) and summarization (§ 2.4), which address our objective to improve the efficiency of operations agents handling activity log data.

2.1 Textual representation of structured data

Algorithm 1 Algorithm that represents the structured data of an activity log as text

```
 1: procedure ActivityLogToText(X: Activity Log, $m$: metadata)
 2:     Initialize empty string output
 3:     Add metadata $m$ to output
 4:     Add new_line token to output
 5:     for each row $i$ in X do
 6:         for each column $j$ in $r$ do
 7:             Add special token to represent column $j$ to output
 8:             Add F($x_{i,j}$) to output
 9:             Add space_token to output
10:         Add new_line token to output
11:     return output
12: procedure F(x)
13:     ▷ Formating function $f$ converts an entry of the Activity
        Log to text ◁
14:     if type($x$)=string then
15:         return $x$
16:     else if type($x$)=list then
17:         Initialize empty string list_output
18:         Sort($x$)
19:         Add "[" to list_output
20:         for each element $e$ in $x$ do
21:             Add F($e$) to list_output
22:             Add space_token to list_output
23:         Remove last space from list_output
24:         Add "]" to list_output
25:         return list_output
26:     else if type($x$)=integer or type($x$)=boolean then
27:         return TO_STRING(x)
```

Algorithm 1 shows how we convert structured data from activity logs into text; we illustrate the process in Figure 2 with an example. This algorithm allows us to leverage powerful language model architectures, such as BERT, for structured data analysis. It begins by initializing a string, *output* to include metadata about the activity log. This metadata is treated as a string, and we will see some examples when we describe our data (§ 3). We iterate through each row in the activity log x: for each column j, it inserts a special token that signifies the name of the column that we are currently processing (line 1). We then add the outputs of a formatting function to the output string, followed by a space. Once we processed all columns in a row, we add a new line character to the output string, symbolizing the end of a row. The formatting function f converts different types of entries in the activity log into text. For example, if an entry x is a string, the function simply returns x. If x is a list, we sort it on alphabetical order or on its content's popularity in the training set— meaning that the most frequently occurring item in the list would appear first. This sorting approach allows us to significantly reduce the complexity of the input space. For example, suppose there are 100 different element types that could populate the list, and a session consists of 3 elements. Without sorting, the model would have to learn from a space of roughly a million $\left(P(100,3)=\frac{100!}{(100-3)!}\right)$ possible token permutations. However, by sorting the activities, the input space is reduced to just a couple hundred thousands $\left(C(100,3)=\frac{100!}{3!(100-3)!}\right)$ token combinations, resulting in a substantial reduction of complexity. Our hypothesis is that this reduction enables the model to learn more efficiently during pre-training, but due to time constraints, we did not validate this empirically. This textual representation enables us to use powerful language model architectures for structured data. We believe that the simplicity and efficiency of this method makes it a valuable tool for prototyping and data analysis in two-sided marketplaces.

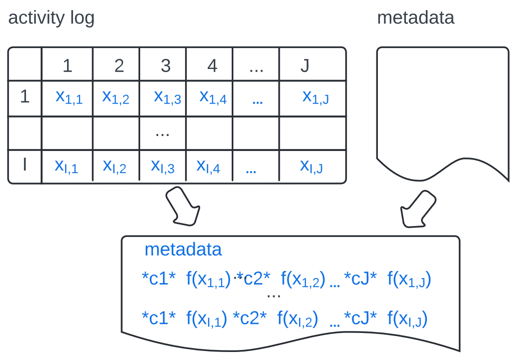
activity log
metadata
1 2 3 4 ... J
1 x1,1 x1,2 x1,3 x1,4 ... x1,J
...
I xI,1 xI,2 xI,3 xI,4 ... xI,J
metadata
c1 f(x1,1) *c2* f(x1,2) ... *cJ* f(x1,J)
...
c1 f(xI,1) *c2* f(xI,2) ... *cJ* f(xI,J)

Figure 2: Example of parsing an activity log as text

The idea of representing numeric values as strings and processing them using a LLM may seem odd at first, but it is quite reasonable in the context of a modern tokenizers. For example, consider a Wordpiece tokenizer [14] that breaks down words into subwords or "wordpieces". A Wordpiece tokenizer allows handling of rare and unseen tokens. For example, the number "21" could be learned to be tokenized as a single entity if it's common in the training data, or with two tokens (up to one per character) if it is uncommon. In a very high-level view, this is not unlike how humans read out certain two-digit numbers with two words (twenty - one) and others with one (twenty). Our numeric representation allows the model to understand some semblance of scale of some values, and aligns with the common practice of discretizing numbers. For example, one-hot encoding is a type of discretization, that transforms each integer number seen in the training set into a new binary feature (*e.g.*, is value equals 2? is it equals 3?, etc), allowing models to capture non-linear relationships between numbers. We argue that treating numbers as tokens effectively discretizes continuous variables. One advantage of our approach is that it handles numeric features in activity logs, even when the quantity of numeric features changes from one log to the next. Traditionally, handling such variability in numerical data requires manual feature engineering—a process where human experts devise methods to aggregate or transform these numbers (averaging, summing, etc.), or designing a neural network architecture that is able to handle this variability. Our method however does not require this manual feature engineering, and we rely on existing neural architectures designed for text. One limitation, however, is that our numeric representation is not exact. Additionally, even larger language models like GPT4 struggle to consistently enumerate numbers. Once we have the activity log represented as text, it is straightforward to use toolkits such as `hf-transformers` for pre-training and fine-tuning.

2.2 Pre-training of Activity Log Embedding Model

After converting the activity logs to text, we pre-train an Activity Log Embedding Model from scratch. As mentioned earlier in the paper, it is not reasonable to use existing BERT parameters because those are usually pre-trained on English or other natural language data that does not resemble Activity Logs. The main goal of this step is to use self-supervised learning to capture the structure and meaning in activity logs. For this, we use a WordPiece tokenizer. Given the unique nature of our work, we specify that the tokenizer should only break text at space and newline characters—since Algorithm 1 defines whitespaces as special characters to map the structured information of the activity logs. We learn the tokenizer word pieces using the Activity Logs in the training set after being processed as text.

Next, we use a Masked Language Model (MLM) method for pre-training. MLM is a method used in BERT that randomizes some parts of the input tokens and predicts them from their context. This forces the model to find meaningful connections between different fields in the activity logs. Because BERT is limited to texts of 512 tokens or smaller, we simply discard additional tokens that do not fit in the context window; however, more sophisticated strategies [19] exist. We leave for future work to investigate the impact of truncation on activity log model. For our experiments, we use the entire training set to learn the tokenizer, and pre-train.

2.3 Fine tuning of Activity Logs for Classification

We fine-tune our pre-trained model to predict whether a given activity log requires follow-up action or not (step 3 in Figure 1). We operationalize this as a binary sequence classification problem. In this context, our goal is to map the sequence of input token (the transformed text-form of the activity logs) to a binary target variable, indicating the candidate label that the activity log requires follow-up action. This classification model doesn't identify the specific rows within the log that require follow-up. Instead, it provides an assessment, indicating whether any session within the given log requires further attention.

2.4 Fine tuning of Activity Logs for Summarization

For summarizing activity logs, we aim to identify the rows of the activity log that agents may need to flag. We framed this as an extractive summarization problem—and is a more granular form of analysis than the classification task. We formulate this solution as a token classification approach. To prepare the data for this task, we use a `BIO` tagging scheme [10]: each token in the (textualized) activity log is assigned a `'B'`, `'I'`, or `'O'` tag, indicating if it's at the beginning of a flagged session, inside a flagged session, or outside any flagged session, respectively. We operationalize this, by assigning a `'B'` to the first token of a flagged session, and marking `'I'` to the rest of the tokens of the session (until a new line is found).

During model training, we optimize a multi-class cross-entropy loss function, which measures the discrepancy between the model predictions and the actual BIO tags. Since this approach requires storing information for each token from the activity log, it is more memory-intensive. To mitigate this, we speed up the training by selectively training only on activity log where a follow-up is required. This is possible because the negative examples will come up from tokens of rows that were not flagged. Thus, the training does not see activity logs with none of the rows unflagged.

	Training	**Dev**	**Test**
# of cases	1.4M	73K	1M
# of rows	9M	472K	722K
% of cases flagged for follow-up	10%	10%	7%
Date range	1/2023-2/2024		3/2024

Table 1: Summary statistics of dataset

3 DATASET

The data we use in this study comes from a prominent two-sided marketplace, and has been anonymized to exclude any personally identifiable information. Our focus is on data generated from a workflow identical to the process depicted in Figure 1. We partition our dataset into training, development, and test subsets. The training set is utilized for model fitting, while the development set aids in hyper-parameter tuning and model selection across epochs. All the results reported in this paper are derived from the test set. To prevent our model from overfitting to seasonal trends in the data, we adopt a time-based split for the test set. While evaluating on new cases of users already reviewed in the training split could be a valid concern, our primary focus is to evaluate on how the system will be used. This approach, contrasting with a random split, takes into account temporal information: data from earlier periods are used for training, while data from later periods are used for testing. Table 1 provides a statistical overview of our data splits. This partitioning ensures that our models are well-trained, tuned, and tested, ensuring robust and reliable performance when deployed in real-world scenarios.

Table 2 outlines the columns present in our activity logs. They include the number of activities per session and the duration of each session, both numeric values. Additionally, they include whether a row was previously selected (boolean), the list of activities within the sessions (list of strings), and certain client features (list of strings). Metadata, represented as a string in two tokens, explains why an agent is reviewing a particular session.

4 EMPIRICAL EVALUATION

We use a precision-recall curve to visualize the performance of our method and compare it against baselines. Precision-recall curve is particularly useful when classes are imbalanced, as it shows the trade-off between precision (how many of the predicted positives

Session features	
How many activities are in each session?	numeric
How many days do each session span?	numeric
Was a row selected previously by an agent?	boolean
Activities in the sessions	list of strings
3 Client features	list of strings
Metadata	
Reason code of why agent is reviewing	string (two tokens)

Table 2: Activity log columns

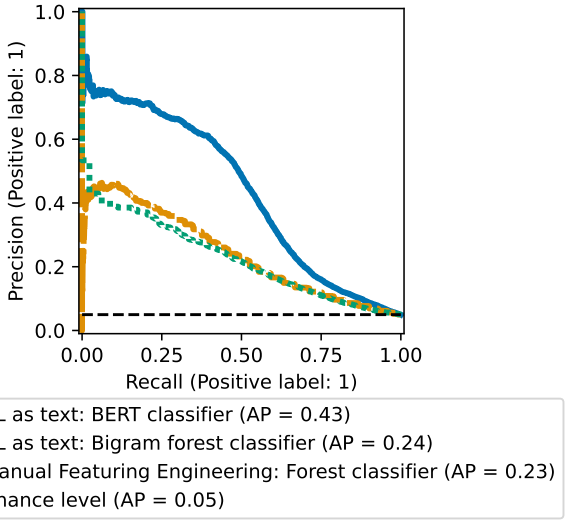

Figure 3

are actually positive) and recall (how many of the actual positives were predicted correctly). The average precision, another metric we report, provides a single-figure summary of the precision-recall curve. It calculates the average of precision values at all recall levels, offering an overall measure of classifier performance.

4.1 Activity Log Classification

Figure 3 compares three different approaches:

These results highlight the potential of employing existing Large Language Model architectures, like BERT, for processing and classifying complex structured data effectively and efficiently. Comparing these results, it's clear that the BERT classifier significantly outperforms the other two approaches. While the bigram random forest and manual feature engineering method have similar performances, they both fall short when compared to the advanced processing and classification capabilities of BERT.

- The **BERT classifier** we built using the text representation produced by applying Algorithm 1 to activity logs, achieved an average precision of 0.43. This is a 79% improvement over the second best baseline.
- We compare it against a **bigram random forest** using the same transformation of Algorithm 1. In preliminary experiments we found no difference between bigram and trigram models—both outperformed unigrams. For this experiment, we did not leverage a Wordpiece tokenizer, and instead aggregated tokens using TFIDF. This baseline yielded an average precision of 0.24. Since this inputs the same textual representation than the BERT model, we can attribute the advantage to BERT's more complex architecture. Although we didn't spend significant effort doing feature engineering or network design, this experiment suggests that our approach can still reap some of the benefits of more modern approaches for neural networks. Additionally, it seems that the textual representation of the activity logs cannot be leveraged using tree-based approaches without additional feature engineering.
- The final method is a baseline approach that relies on **manual feature engineering**. We use the output of a real production system that was designed for a very similar—though not identical task:
  - The labels used during training are related to our case case, but not completely identical.
  - The training data is slightly different.
  - Other small differences.

  The motivation behind employing this system is its validated efficacy as a model. It's not trivial to design an experiment to benchmark against manual feature engineering—as this process depends on how much time is invested, and the skills of the machine learning engineer (which is not easily quantifiable). Future work could allow different machine learning experts to build features and control by amount of time spent. However, because of time constraints we did not explore this route. The average precision of this baseline was only 0.23. Although it is possible that an expert can design features that outperform the BERT model, we think that this comparison can shine light on the difficulty of manual feature engineering, and gives us a ballpark estimate of the relative performance of our approach.

4.2 Activity Log Summarization

Figure 4 illustrates the results of comparing the results of a BERT model against a binary random forest:

- Our **BERT summarizer** model was trained to generate BIO tags for each token within the activity log—in other words, the model predicts a tag (`'B'`, `'I'`, or `'O'`) for each token in the textualized activity log. However, in the context of our task, we are interested in the model's ability to classify entire rows rather than individual tokens. To bridge this gap between token-level prediction and row-level classification, we employed a majority rule approach: if a row had more `'B'` or `'I'` tags than `'O'` tags, we considered the row as selected. This strategy empowered the BERT summarizer model to achieve an average precision of 0.23, demonstrating its capacity to effectively identify pertinent rows within the detailed structure of activity logs.

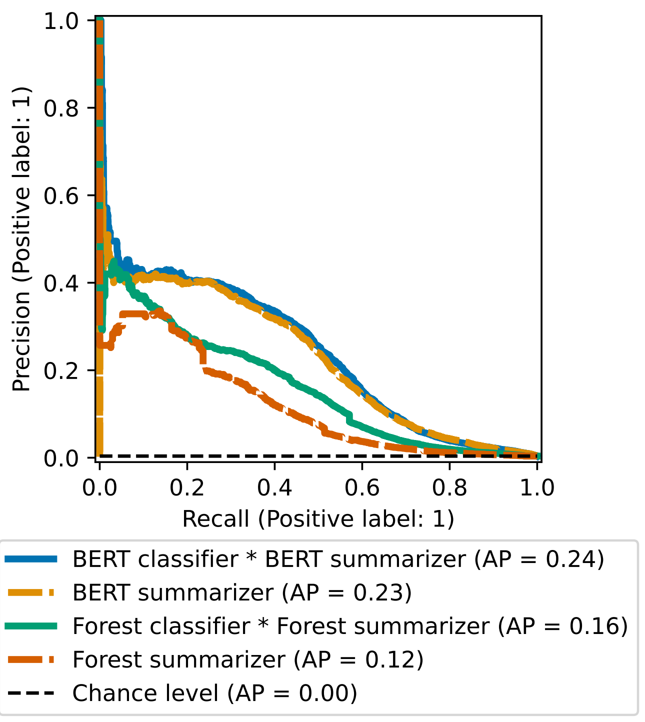

Figure 4

- Because the summarizer model was only trained on cases that were flagged for review, it didn't see negative examples. So, to improve the performance, we multiplied the probability of the summarizer by the probability that the case needs follow-up. This modification led to a better average precision of 0.24.
- We also implemented a **forest summarizer** that uses bigrams and evaluates each row individually, without considering the context of other rows in the case. This is effectively a binary classifier, trained exclusively on cases that required follow-up, and achieved an average precision of 0.12.
- Finally, multiplying the output of the forest summarizer and the random forest achieves an average precision of 0.16.

The results suggest that the BERT model is more effective in identifying the specific rows that require follow-up compared to a binary random forest. This highlights the potential of using pre-trained language models like BERT for handling complex structured data and making precise predictions at a granular level.

5 RELATED WORK

Our work relates to AutoML [8] techniques, which are a family of algorithms designed to reduce the development time for data preparation, feature engineering, model generation, and model evaluation. Although very promising, these techniques are not yet mature and are not truly automatic but for a few applications [9]. Moreover, it is unclear how well they work for activity log data. Our work focuses on activity log data—which is a crucial data source for two-sided market places.

Our work relates to efforts of extending Large Language Models (LLMs) to structured and tabular data. Expansions of LLMs can be categorized into two research lines. The first line includes models that aim to link textual and structural information from tables for better context-aware representations [14, 18]. These LLMs are usually evaluated in downstream tasks related to keyword and content-based table retrieval, and table similarity. These differ from our work because we do not deal with natural language text—we are only interested in facilitating the processing of activity logs. Moreover, it is unclear how these methods would generalize for classification and summarization. The second family aims to capture the relationship between columns and rows [4, 5], or to embed different data types within structured data [1, 17]. These models are evaluated on downstream tasks such as column type annotation, column property annotation, table type detection and table similarity prediction. In relation to our focus on classification and summarization, it is unclear if these approaches would be helpful for summarization. The table type detection task is potentially related to our classification goal, but unfortunately the published results [4] do not compare with traditional methods nor provide enough details to understand the extend on which their technique could be relevant. We leave for future work to explore these promising techniques and run evaluations against our methods.

Alternatively, conventionally pre-trained LLMs have been used to understand structured data [13] with mixed results depending on on table input format, content order, role prompting, and partition markers. Relatedly, growing empirical evidence suggest that deep neural networks struggle when the input space is a simple vector [12].

6 CONCLUSION

This paper presents an innovative approach to be able to use natural language processing toolkits for structured data analysis. Our method of transforming structured data into text format enables us to leverage powerful language model architectures such as BERT. We demonstrate our method's effectiveness through a case study of a two-sided marketplace, where we build models that may be helpful for operations agents in processing and analyzing activity log data.

Training machine learning models is a time and resource-intensive process. In the case of our experiments, the training time spanned about 4 days for pre-training, 4 days for fine-tuning the case classifier, and almost an entire week for the fine-tuning the summarization model using a single Nvidia Tesla T4 GPU. We set the batch sizes as large as the GPU memory allowed. This can be contrasted with the training time for the random forest, which can often be completed within just hours using CPU parallelism. The extended training time for BERT models can also result in additional costs, particularly when considering the need for stronger computational resources, such as GPUs. Another potential limitation in our approach, is that it relies on neural networks that in general are regarded as difficult to interpret. While inference in BERT is deterministic, it is challenging to determine beforehand whether a slight alteration in the input would result in a significant change to the model's output.

Despite their shortcomings, the use of BERT models can potentially lead to significant savings in terms of human labor. Traditional manual feature engineering approaches require extensive human involvement in the iterative process of designing, implementing, and testing features. Our results indicate that using a BERT model can outperform manual feature engineering in terms of average precision. However, it is essential to note that these results are initial findings and more extensive experimentation is necessary. Limitations of our work include that we only compared to a single existing manually engineered model that was not designed for identical goals than ours—it may be more compelling to compare against a large number of machine learning engineers, perhaps using a platform such as Kaggle [2]. Additionally, future work can look at other alternatives to BERT that may be simpler (eg. LSTM [7]) or more complex (eg., Megatron [11]).

The effectiveness of our approach versus more traditional machine learning methods can vary depending on the specific characteristics of the problem and the data; as well as the time and skill of the machine learning engineer. As such, while our results are promising, they should be seen as a motivation for further exploration rather than a definitive conclusion. The potential for cost and labor efficiency combined with the high performance of the LLM models to structured data makes it a compelling avenue for future research.

REFERENCES

[1] Luis Armona, José P González-Brenes, and Ralph Edezhath. 2019. Beyond Word Embeddings: Dense Representations for Multi-Modal Data. In *The Thirty-Second International FLAIRS Conference.*

[2] Casper Solheim Bojer and Jens Peder Meldgaard. 2021. Kaggle forecasting competitions: An overlooked learning opportunity. *International Journal of Forecasting* 37, 2 (2021), 587–603.

[3] Lars Buitinck, Gilles Louppe, Mathieu Blondel, Fabian Pedregosa, Andreas Mueller, Olivier Grisel, Vlad Niculae, Peter Prettenhofer, Alexandre Gramfort, Jaques Grobler, et al. 2013. API design for machine learning software: experiences from the scikit-learn project. *arXiv preprint arXiv:1309.0238* (2013).

[4] Pei Chen, Soumajyoti Sarkar, Leonard Lausen, Balasubramaniam Srinivasan, Sheng Zha, Ruihong Huang, and George Karypis. 2024. HYTREL: Hypergraph-enhanced tabular data representation learning. *Advances in Neural Information Processing Systems* 36 (2024).

[5] Xiang Deng, Huan Sun, Alyssa Lees, You Wu, and Cong Yu. 2022. Turl: Table understanding through representation learning. *ACM SIGMOD Record* 51, 1 (2022), 33–40.

[6] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* (2018).

[7] Alex Graves and Alex Graves. 2012. Long short-term memory. *Supervised sequence labelling with recurrent neural networks* (2012), 37–45.

[8] Xin He, Kaiyong Zhao, and Xiaowen Chu. 2019. AutoML: A Survey of the State-of-the-Art. *CoRR* abs/1908.00709 (2019). arXiv:1908.00709 http://arxiv.org/abs/1908.00709

[9] Shubhra Kanti Karmaker, Md Mahadi Hassan, Micah J Smith, Lei Xu, Chengxiang Zhai, and Kalyan Veeramachaneni. 2021. Automl to date and beyond: Challenges and opportunities. *ACM Computing Surveys (CSUR)* 54, 8 (2021), 1–36.

[10] Lance Ramshaw and Mitch Marcus. 1995. Text Chunking using Transformation-Based Learning. In *Third Workshop on Very Large Corpora.* https://aclanthology.org/W95-0107

[11] Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. 2019. Megatron-lm: Training multi-billion parameter language models using model parallelism. *arXiv preprint arXiv:1909.08053* (2019).

[12] Ravid Shwartz-Ziv and Amitai Armon. 2022. Tabular data: Deep learning is not all you need. *Information Fusion* 81 (2022), 84–90.

[13] Yuan Sui, Mengyu Zhou, Mingjie Zhou, Shi Han, and Dongmei Zhang. 2024. Table meets llm: Can large language models understand structured table data? a benchmark and empirical study. In *Proceedings of the 17th ACM International Conference on Web Search and Data Mining.* 645–654.

[14] Mohamed Trabelsi, Zhiyu Chen, Shuo Zhang, Brian D Davison, and Jeff Heflin. 2022. StruBERT: structure-aware BERT for table search and matching. In *Proceedings of the ACM Web Conference 2022.* 442–451.

[15] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems* 30 (2017).

[16] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 conference on empirical methods in natural language processing: system demonstrations.* 38–45.

[17] Ledell Wu, Adam Fisch, Sumit Chopra, Keith Adams, Antoine Bordes, and Jason Weston. 2018. Starspace: Embed all the things!. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 32.

[18] Pengcheng Yin, Graham Neubig, Wen-tau Yih, and Sebastian Riedel. 2020. TaBERT: Pretraining for joint understanding of textual and tabular data. *arXiv preprint arXiv:2005.08314* (2020).

[19] Manzil Zaheer, Guru Guruganesh, Kumar Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontanon, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, et al. 2020. Big bird: Transformers for longer sequences. *Advances in neural information processing systems* 33 (2020), 17283–17297.

Received 18 May 2024; revised 15 July 2024; accepted 1 July 2024