Active Learning with Table Language Models

Martin Ringsquandl  
Siemens AG  
Munich, Germany  
martin.ringsquandl@siemens.com

Aneta Koleva  
Siemens AG, LMU  
Munich, Germany  
aneta.koleva@siemens.com

Abstract

Despite recent advancements in table language models research, their real world application is still challenging. In industry, there is an abundance of tables found in spreadsheets, but acquisition of substantial amounts of labels is expensive, since only experts can annotate the often highly technical and domain-specific tables. Active learning could potentially reduce labeling costs, however, so far there are no works related to active learning in conjunction with table language models. In this paper we investigate different query strategies in a real-world industrial table language model use case. Our results show that there is potential for improvement and some fundamental questions to be addressed.

1 Introduction

Recent tabular language models (TaLMs) which are based on pre-trained transformers and modality-adapted to tabular data have shown promising results on academic datasets [3]. However, using pre-trained transformer models for complex tasks in industry still requires fine-tuning with substantial amounts of labels. Although tabular data in spreadsheets is ubiquitous in industry, the application of TaLMs in real-world settings is still missing. One reason is that acquiring labels for industrial spreadsheets, e.g. cell-level entity annotations, quickly becomes prohibitively expensive considering that often highly technical and domain specific language can only be annotated by a handful of experts.

Settings with abundant unlabeled data and costly label acquisition are perfect matches for active learning (AL), which aims to minimize the costs of acquiring labeled data by maximizing model’s performance with each new labeled instance. Deep neural networks, such as large-scale transformers, require batch-based query strategies instead of classical AL (one-by-one instance selection), since these large models need to be trained for multiple epochs on batches of labeled data to effectively change the model’s parameters for the next iteration of candidate querying. Otherwise training is inefficient and can lead to overfitting [5]. A popular approach here is to have hybrid query strategies that weigh uncertainty instance-wise scores against some within-batch similarity (diversity) of all instances [10].

In the text data modality, batch typically means a set of sentences or documents. Even for a given token-level task such as named entity recognition (NER), we still want to select the most informative sentences (not single tokens), since all NER labels depend on the full sentence context. In AL terms, NER is a so-called multi-instance problem [8]. Transferring this to tabular data, a batch therefore should be a set of tables. Given a cell-level task, where each cell may have multiple labels, this is now a nested multi-instance problem. Since the instance is the full table, each table has multiple cells and cells have multiple labels. Until now, the table modality for AL has not been explored and it is unclear how to deal with such a novel AL problem. In this paper, we study how well-known AL strategies can be adopted to TaLMs to solve an industrial NER task where named entities are mentioned in tables cells.

Submitted to Table Representation Learning Workshop at NeurIPS 2022. Do not distribute.
Our key contributions are the following:

- We present an industrial NER use case for TaLMs in conjunction with AL.
- We outline novel problems that arise with AL when dealing with tabular data.
- We adopt well-known query strategies to tables and carry out an empirical evaluation on a real-world industrial dataset.

2 Industrial Table NER Use Case

This section describes our real-world table NER use case. Industrial plant operators maintain data about plant equipment, such as actuators, sensors, vessels, etc. This data is typically collected and maintained by engineers in spreadsheets. The spreadsheets are roughly organized in a tabular format, as shown in the example table in Figure 1. In these spreadsheets, each row typically represents information about one or multiple equipment instances. Some columns represent relevant physical properties of these equipment, while others are non-informative. However, the engineers do neither comply to a fixed schema, nor to unified spelling of equipment or properties. The NER task is to automatically extract relevant entities for creating structured specifications of the plant equipment. We phrase this problem as NER task (sub-cell NER) with the following types of entities. The type \( \text{TAG} \) refers to a systematic identifier of an equipment. There are some conventions for generating equipment tags (e.g. NORSOK, KKS), but most plant operators customize them and some sheets do not contain identifiers at all. The type \( \text{EQ} \) is for surface names of equipment types. The type \( \text{QUANT} \) refers to the physical properties/quantities describing the functional specifications of equipment and the type \( \text{UoM} \) stands for unit of measurement.

As these tables can get quite large, e.g. hundreds of rows, labeling all cells in a single table is already too time-consuming for human annotators. However, table cells can contain lots of similar, repetitive content, where it may be enough to provide small amount of labels to reach good performance. Therefore, we are interested in querying strategies that would select the most informative instances, but it is unclear on which level to select those, as shown in the bottom of Figure 1.

![Figure 1: Overview of our industrial sub-cell NER problem and the active learning instance selection.](image)
3 AL and TaLMs

In this section we give a general introduction of AL and define our TaLM for sub-cell NER.

Given an unlabeled dataset \( U \sim \mathbb{P} \) sampled from some true unknown distribution \( \mathbb{P} \). The objective of pool-based AL is to select candidates \( S \subset U \) to be labeled with restricted budget \( m \geq |S| \) and \( m \ll |U| \) such that:

\[
\arg\min_S \mathbb{E}_{x \sim \text{p}} [\ell(f_S(x), y)],
\]

where \( y \) is the corresponding label of instance \( x \), \( \ell(\cdot) \) is a loss function and \( f_S \) is a classification model trained on labeled data \( S \). Usually, AL is performed in an iterative fashion where candidates in the next iteration are sampled considering the model trained on the previous candidates.

In our case \( f \) is a TaLM and \( U \) consists of a set of tables. We define a table as a tuple \( T = (C, H) \), where \( C = \{c_{1,1}, c_{1,2}, \ldots, c_{t,j}, \ldots, c_{n,m}\} \) is the set of table body cells for \( n \) rows and \( m \) columns. Every cell \( c_{i,j} = (w_{c_{i,j},1}, w_{c_{i,j},2}, \ldots, w_{c_{i,j},t}) \) is a sequence of tokens of length \( t \). The table header \( H = \{h_1, h_2, \ldots, h_m\} \) is the set of corresponding \( m \) column header cells, where \( h_j = (w_{h_{j,1}}, w_{h_{j,2}}, \ldots, w_{h_{j,t}}) \) is a sequence of header tokens.

Our TaLM consists of an encoder which will produce a table context-sensitive representation for each token:

\[
w_{h_{1,1}}, \ldots, w_{h_{m,t}}, w_{c_{1,1}}, \ldots, w_{c_{n,m}}, t = \text{ENC}(T),
\]

where ENC is a table-biased transformer encoder using the row-column visibility matrix instead of vanilla attention and only within-cell positional encoding, since table rows/columns have no order [9]. Note that most TaLM architectures like TURL [2] are not directly applicable here, since they already aggregate tokens within cells to get cell-level representations. This is why we resort to a custom TaLM implementation which takes a pre-trained language transformer and modality adapt it using a token-level table visibility matrix, header type encodings and cell-wise token positional encoding.

Each labeled cell has an NER-tag sequence: \( (y_1, y_2, \ldots, y_t) \), where each \( y_i \in \mathcal{Y} \). We use IO tags, thus \( \mathcal{Y} \) is \( \{O\} \cup \{I-, E-, QUANT\} \), where \( \mathcal{ENT} \in \{\text{TAG}, \text{EQ}, \text{QUANT}, \text{UoM}\} \). Our TaLM decoder is a classification layer that projects each token representation into the label space, plus a Softmax activation to assign a normalized score for each class.

4 Query Strategies

The most important aspect of AL is the choice of query strategy. This strategy determines how to select new (unlabeled) instances such that they maximize the informativeness to the current model.

For single-instance tasks, such as text classification, the query strategy is straightforward - every instance has one label so select the instance \( x \) with the most informative label \( y \). In multi-instance tasks such as NER there might be multiple labels (entity spans) in a single sentence. Therefore, we want to select the instance \( x \) with the most informative joint labels \( (y_1, y_2, \ldots, y_t) \). In our table NER case, we have a nested multi-instance problem. Every table instance \( T_k \) has multiple cells, for example body cells \( c_{k,i,j} \), and every such cell has a NER-tag sequence \( (y_{k,i,j,1}, y_{k,i,j,2}, \ldots, y_{k,i,j,t}) \) associated with the \( t \) tokens in the cell.

One could argue that in our table NER case, the query strategy should select whole tables with most informative set of cells. Unfortunately, it is not reasonable to assume that a human annotator would have the time to annotate every cell in a full table. Hence, since a single table would already break the annotation budget, we make the simplifying assumption that cells within a table are independent and adopt common query strategies to select batches of most informative cells.

For this reason we consider two popular strategies in our experiments. The first one is a pure uncertainty-based, batch-agnostic one and the second one is a batch-diverse strategy.

MNLP A popular strategy for uncertainty-sampling in multi-instance problems is the maximized normalized log-probability (MNLP) [7]. It considers the probability of the model’s most likely label sequence to select new instances (here cells) \( c_{i,j} \) with mean normalization to account for the bias
Table 1: Training tables statistics

<table>
<thead>
<tr>
<th>Statistic</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>#tables</td>
<td>55</td>
</tr>
<tr>
<td>#cells</td>
<td>4,774</td>
</tr>
<tr>
<td>#labels</td>
<td>1,112</td>
</tr>
<tr>
<td>labels per cell</td>
<td>0.23</td>
</tr>
</tbody>
</table>

towards longer sequences:

\[
\text{mnlp}(c_{i,j}) = \max_{y_1,\ldots,y_t} \frac{1}{t} \sum_{z=1}^{t} \log p(y_z | w_{c_{i,j},1}, \ldots, w_{c_{i,j},t})
\]  

(3)

Since sums of log-probabilities are negative, the candidates are selected in ascending order w.r.t. to MNLP. MNLP has a major weakness in the batch-based AL setting. The sequences with highest uncertainty for the current model tend to be highly correlated. Hence, when selecting a batch of candidates, there will be substantial similarity within the batch, providing little additional value. For tables this means that most candidates will likely come from the same table or even the same row/column.

BADGE As a batch-diverse strategy we consider Batch Active learning by Diverse Gradient Embeddings (BADGE) [1]. Here, the authors propose to measure uncertainty in the magnitude of the model’s last layer gradients. A diverse set of candidates is then selected using the \(k\)-MEANS++ initialization to sample from the gradient embeddings. In our case, we take the mean of each cell’s token gradient embeddings and feed this into \(k\)-MEANS++ to get cell candidates.

Rand As a simple baseline we also employ a strategy that selects candidate cells uniformly at random from the set of all cells in all available tables.

5 Experiments

In this section we describe the experiments on our real-world industrial dataset.

Dataset The dataset contains spreadsheets from multiple industrial plants. The rows in the original sheets were downsampled, such that each resulting table had a maximum of 5 rows. Expert annotators were then asked to label each table cell-by-cell using the prodi.gy span-based NER annotation tool. After performing a random train-test split, we end up with 55 tables in the training set and 24 in the test set. The training set statistics are shown in Table 1. We can see that in total they contain \(\sim 1,1k\) NER span labels. Dividing this by the number of cells in all tables, we observe that the vast majority \(\sim 77\%\) of cells are only comprised of tokens with label \(O\). This is an extreme class-imbalance - typical for industry settings.

AL Setup For every AL strategy, a single experiment starts with a set of 100 labeled cell candidates \(L\) as seed to initially train our NER TaLM (iteration 0). Then, in each iteration the AL strategy selects a batch \(S\) of 50 unlabeled instances (cells) from \(U\) that are added to \(L\) along with their true labels. Following the suggested protocol from [4], we fine-tune in each iteration from scratch to avoid overfitting data from previous rounds. In each iteration we train for a maximum of 10 epochs and use a small portion of the test set as validation for early stopping. In each experiment, all AL strategies start with the same initial seed. The reported results are the average over 5 different experiments, i.e., 5 different initial seeds. As pre-trained language model we use the ‘bert-base-uncased’ from huggingface. After every experiment we evaluate the model’s final performance against the hold-out test set. As a ceiling benchmark we also train our NER TaLM on the full training dataset. As performance metric we chose the micro-averaged F1-score.
6 Results and Discussion

The final test set performance is shown in Figure 2a. It can be seen that while none of the AL strategies come close to the full performance, MNLP is by far the worst. Interestingly, BADGE has only slight advantages at the beginning and is eventually beaten by random sampling in later iterations. Looking at the minimum and maximum performance of all strategies (cross tick marks), we notice high variance, especially for MNLP, which means the initial seed is crucial for final performance. This is probably due to the missed-class problem, when the initial seed is completely missing some of the classes in such a high-imbalance setting [8]. The Rand strategy is simply the most robust, since later iterations do not depend on the initial seed.

In terms of diversity of samples we show the number of different tables from which the strategies sample cells in each iteration in Figure 2b. This metric seems to be correlated with the F1 scores, suggesting sampling from a wider range of tables is better. It shows that MNLP suffers from selecting similar cells from the same table. Another observation is that even random sampling only samples from roughly half the tables. This is due to an imbalance in table sizes - some large ones dominating the uniform sampling procedure.
**Computational Efficiency** Apart from pure classification performance, we may also consider computational efficiency. As an example, in a cell-level task, a query strategy that retrieves a set of \( k \) candidate cells from a single table versus \( k \) cells which span over \( k \) tables (one cell per table) will perform much fewer computations. Especially, for large tables we may want our query strategy to balance the informativeness of instances versus the number of tables that fit in a batch compute budget. Alternatively, we want the strategy to select the most informative sub-tables rows (context), such that large tables will not impact the compute limits too much. Since TaLMs have a high memory footprint, a very table-diverse query strategy may be much more expensive than others.

**Human Annotators** On the other hand, a table-diverse strategy may also impact the human annotators. When presented with many different tables, the annotator probably needs more time per cell annotation, because every table’s unique schema and context needs to be interpreted. This again raises the question if the full table needs to be shown to the user to make an informed labeling decision or a sub-table is sufficient.

### 7 Related Work

Pre-trained large-scale language models like BERT have been studied in AL setups for different tasks, e.g. text classification [6]. While these models can already show good results when fine-tuning on small labeled datasets, in industry AL is still often necessary as argued in [4], where an empirical study of AL for BERT in text classification was carried out. They consider the challenging, practical scenario with high label imbalance and small annotation budget. Their results show that batch-diverse strategies handle this scenario better than classical ones.

For NER deep conditional random field models have been studied with AL [7] with the focus on computational efficiency, due to expensive training. Further the *missed-class effect* due to the AL’s exploitative strategy has been studied in NER [8]. The authors propose strategies for more informed seed instance selection that can alleviate this problem.

While all of these works are relevant to ours, it is non-trivial to transfer their findings to AL on tables using TaLMs.

### 8 Conclusion

In this paper we describe an industrial use case for table language models and active learning. We define this novel nested multi-instance AL problem and adopt some well-known query strategies to select cell candidates from unlabeled tables. Our empirical results based on a real-world dataset show that these strategies can be applied to TaLMs, but suffer from high dependence on the initial seed and more table-diverse strategies may be needed. Since our work is the first to address this novel problem setting, there are still some open questions for future work regarding computational efficiency and impact on human annotators.

### References


