SmartTable: A Spreadsheet Program with Intelligent Assistance

Shuo Zhang
University of Stavanger
shuo.zhang@uis.no

Vugar Abdul Zada
University of Stavanger
v.abdulzada@stud.uis.no

Krisztian Balog
University of Stavanger
krisztian.balog@uis.no

ABSTRACT

We introduce SmartTable, an online spreadsheet application that is equipped with intelligent assistance capabilities. With a focus on relational tables, describing entities along with their attributes, we offer assistance in two flavors: (i) for populating the table with additional entities (rows) and (ii) for extending it with additional entity attributes (columns). We provide details of our implementation, which is also released as open source. The application is available at http://smarttable.cc.

CCS CONCEPTS

• **Information systems** → **Environment-specific retrieval**; *Users and interactive retrieval*; *Recommender systems*; Probabilistic retrieval models;

KEYWORDS

Table completion; intelligent table assistance; semantic search

1 INTRODUCTION

Tables are a powerful, effective, and easy-to-use tool for both visual organization and manipulation of data. Tables can be found in vast quantities on the Web, and spreadsheet programs are among the most commonly used desktop applications. Our objective is to equip spreadsheet programs with intelligence assistance capabilities, to aid users while working with tables. In this paper, we focus on one particular type of tables, known as *relational tables* [8, 9]. Relational tables describe a set of entities along with their attributes. We shall refer to the column containing the entities as the *core column.* Typically, it is either the leftmost table column or the second column from the left, in case row sequence numbering is used. The heading labels of the table refer to particular attributes, while data cells hold the values of those attributes. It is also assumed that the table is given a title (caption). See Figure 1 for an illustration.

There exists a number of online tools and resources for table-related tasks, such as table search (Google Fusion Tables [3] and

Figure 1: Example of a relational table T, where c is the table caption, E denotes the core column entities $E = \{e_1, \ldots, e_n\}$, and L is the set of column labels $L = \{l_1, \ldots, l_m\}$.

WikiTables [1]), question answering [6], and entity linking in tables [1]. To the best of our knowledge, our system, called *SmartTable*, is the first online spreadsheet program that provides intelligent table content recommendation. Specifically, our application is capable of providing two kinds of assistance: (i) recommending additional entities, from an underlying knowledge base, to be added to the core column (row population) and (ii) recommending additional entity attributes to be included as columns (column population). Such recommendations are particularly useful in scenarios with an exploratory or recall-oriented nature, i.e., when the user does not have a very clear idea beforehand as to what should be included in the table. Additionally, SmartTable also provides regular table operations, such as adding, deleting, and moving rows and columns, editing cells, and supporting various value types (entities, numbers, currencies, dates, etc.).

Both types of assistance, that is, row and column population, are based on probabilistic models that we developed in prior work [10]. The main contributions of this paper are twofold. First, we integrate the above assistance functionality into an online spreadsheet application. Second, we describe the task-specific indexing structures employed, and evaluate the efficiency of our implementation in terms of response time. SmartTable is implemented using a HTML5 front-end and a Python+ElasticSearch back-end. It uses DBpedia as the underlying knowledge base and a corpus of 1.6M tables extracted from Wikipedia. The implementation is made open source at https://github.com/iai-group/SmartTable and the application is available online at http://smarttable.cc.

2 OVERVIEW

In this section, we provide an overview of the functionality of the SmartTable application, by walking through the process of creating a table from scratch.

- Initially, we start with an empty table, with the table caption, core column entities, column labels, and cell values waiting to be filled. The user is expected to add a few entities and column labels first, along with an optional table caption, to

supply the system with some data to base recommendations on. We shall refer to this (incomplete) table as the *seed table*. See Fig. 2a.

- When adding entities to the core column, the user is presented with a ranked list of suggestions. Additionally, the user can search the underlying knowledge base for entities. See Fig. 2b.
- When adding new columns, the user needs to specify the data type for that column (which can be one of entity, text, date, number, currency, or percentage) and provide a label for that column. For the latter, a ranked list of suggestions are offered, along with a search box to search for additional labels. See Fig. 2c.

3 METHODS

In this section, we introduce the methods underlying the assistance functionality. We refer to Fig. 1 for the notation used for the various table elements. As for the data, we employ a table corpus (TC) extracted from Wikipedia and use DBpedia as the knowledge base (KB); further details about the datasets are given in Sect. 4.1.

3.1 Row population

Row population is the task of generating a ranked list of entities to be added to the core column of a given seed relational table. It is closely related to the task of entity set expansion [2, 5, 7], which is about expanding a seed entity set with additional instances. The main difference between row population and entity entity set expansion is that we can also leverage additional data from the seed table as input, not only the core column entities. The row population task is split into two sub-tasks, which are candidate selection and ranking entities, respectively.

3.1.1 Candidate selection. We identify candidate entities using both the knowledge base and the table corpus. From the knowledge base, we take entities that share the assigned semantic categories with those of the seed entities. From the table corpus, we first find tables similar to the seed table, based on table caption, core column entities, and column heading labels. Then, we take the core column entities from those similar tables as candidates.

3.1.2 Ranking entities. We implement the probabilistic model proposed in [10], which is a multi-conditional probability:

$$P(e|E, L, c) \propto P(e|E)P(L|e)P(c|e) \,,$$

where $P(e|E)$ is entity similarity, $P(L|e)$ denotes column labels likelihood, and $P(c|e)$ is caption likelihood.

Entity similarity is estimated using

$$P(e|E) = \lambda_E P_{KB}(e|E) + (1 - \lambda_E)P_{TC}(e|E) \,,$$

where $P_{KB}(e|E)$ is the average Jaccard similarity between the candidate entity e and each seed entity $e' \in E$, and $P_{TC}(e|E)$ is fraction of tables in the table corpus that contain both the seed and candidate entities out of the number of tables containing the seed entities.

Title: Population growth of European cities

	City	Country	Population growth	⊕
1	Helsinki	Finland	0.12%	
2	Oslo	Norway	3.00%	
3	Stavanger	Norway	0.50%	
4	⊕			

(a) Seed table with some initial data.

Title: Population growth of European cities

	City	Country	Population growth	⊕
1	Helsinki	Finland	0.12%	
2	Oslo	Norway	3.00%	
3	Stavanger	Norway	0.50%	
4				

🔍 Search...

Raseborg

Rønne

Haven ports

Sola

Jämsä

(b) Row population assistance.

Title: Population growth of European cities

	City	Country	Population growth	
1	Helsinki	Finland	0.12%	
2	Oslo	Norway	3.00%	
3	Stavanger	Norway	0.50%	
4	⊕			

🔍 Column Name

A   📅   €   %

Language

Continent

Capital

Venue/Event

(c) Column population assistance.

Figure 2: Screenshots from the SmartTable system.

Column labels likelihood considers the table corpus and is estimated using a Dirichlet-smoothed language model:[1]

$$P(L|e) = \sum_{l \in L} \prod_{t \in l} \frac{tf(t, e) + \mu P(t|\theta)}{|e| + \mu} \,,$$

---

[1] In our original approach [10] this estimate was a two-component mixture. Due to efficiency considerations, we use a simplified version here. The relative difference in terms of effectiveness is below 5%.

where $tf(t, e)$ is the term frequency of t in the column labels of tables containing e and $|e|$ is the sum of all term frequencies for e. The collection language model $P(t|\theta)$ is computed based on the column labels of all tables in TC.

Caption likelihood is a two-component mixture:

$$P(c|e) = \prod_{t \in c} \left(\lambda_c P_{KB}(t|\theta_e) + (1 - \lambda_c)P_{TC}(t|e) \right) ,$$

where the KB component is estimated using a Dirichlet-smoothed entity language model. The TC component is computed as $P_{TC}(t|e) = \#(t, e)/\#(e)$, where $\#(t, e)$ is the number of tables in the table corpus containing entity e in the core column and term t in the table caption, and $\#(e)$ is the total number of tables containing e.

3.2 Column population

Column population is the task of generating a ranked list of column labels to be added to the column headings of a given seed table. It is also implemented as a sequence of two steps: candidate selection and column label ranking.

3.2.1 Candidate selection. Candidate labels are obtained from related tables. To find related tables, we use (i) the table caption, (ii) table entities, and (iii) seed column heading labels as queries. From the matching tables, column labels are extracted as candidates.

3.2.2 Ranking column labels. The related tables, identified in the candidate selection stage, are also utilized in the ranking step. According to the model in [10], the probability of a candidate column label is given by:

$$P(l|E, c, L) = \sum_{T} P(l|T)P(T|E, c, L) ,$$

where T represents a related table, $P(l|T)$ is the label's likelihood given T, and $P(T|E, c, L)$ expresses that table's relevance. The probability $P(l|T)$ is set to 1 if the candidate label l is present in table T and is 0 otherwise. The relevance of a table is estimated as:

$$P(T|E, c, L) \propto P(T|E)P(T|c)P(T|L) ,$$

where $P(T|E)$ denotes entity coverage, $P(T|c)$ is caption likelihood, and $P(T|L)$ is the column labels likelihood.

Entity coverage is the overlap of core column entities in the seed table and in T: $P(T|E) = |T_E \cap E|/|E|$.

Caption likelihood is estimated using term-based similarity between the seed table's caption and the content of T: $P(T|c) \propto sim(T_c, c)$. Here, we employ BM25 scoring.

Column labels likelihood is the overlap between labels of the seed table and those of T: $P(T|L) = |T_L \cap L|/|L|$.

4 IMPLEMENTATION

In this section, we describe the datasets used and indices built, along with technical details of our implementation.

4.1 Datasets

We rely on two data sources: a table corpus and a knowledge base. The knowledge base is DBpedia, version 2015-10.[2] We filter out entities that do not have a short textual description (abstract). After

---

[2]http://wiki.dbpedia.org/dbpedia-dataset-version-2015-10

```
{
    "_index": "entities",
    "_type": "doc",
    "_id": "Hot_pot",
    "_version": 1,
    "found": true,
    "_source": {
      "category": [
        "hong_kong_cuisine",
        "cantonese_cuisine",
        "beijing_cuisine",
        "sichuan_cuisine",
        "yunnan_cuisine",
        "table-cooked_dishes",
        "chongqing_cuisine",
        "stews"
      ],
      "category_count": 8,
      "label": "Hot pot"
    }
}
```

Figure 3: Example entry from the entity index.

filtering, we are left with a total of 4.6M entities. As for the table corpus, we use the WikiTables collection [1], which comprises of 1.65M tables, extracted from Wikipedia. We preprocess tables as follows. Entities are marked up in the original table with hyperlinks. If the link points to an entity that exists in DBpedia, we replace that link with the corresponding entity identifier. Otherwise, we replace the link with the anchor text.

4.2 Indices

We build the following inverted indices:

Table index It contains 1.65M Wikipedia tables (6.4GB). For each table, the following fields are stored: page title, section title, table caption, column labels, table data, and core column entities.

Entities It contains 4.6M DBpedia entities (2GB). For each entity, we store its canonical name (label), and the list and number of categories it is assigned to. See Fig. 3 for an example.

Categories We use Wikipedia's category system, comprising of around 1M categories. For each category, we store the list of entities that are assigned to that category. This index occupies 2GB.

4.3 Implementation

SmartTable is a web application that is comprised of a HTML5 front-end and a back-end based on Python and Elasticsearch.

4.3.1 Front-end. The front-end stack is made up of HTML, CSS, and JavaScript (ECMAScript6 standard). We build on a third-party JavaScript spreadsheet framework called Handsontable,[3] which provides a rich set of functionality for tables, including sorting, conditional formatting, contextual menus, moveable and resizable rows and column, etc. Additionally, we utilize the `Gulp.js`, `Babel.js`, and `Node.js` JavaScript libraries.
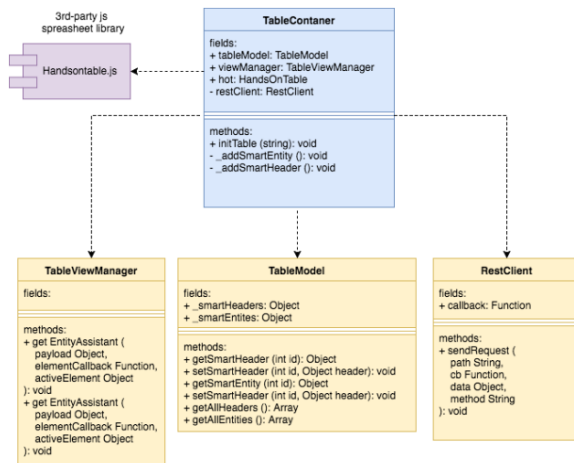
---

[3]https://handsontable.com/

Figure 4: Overview of the application front-end.

For development, we follow the MVC (Model View Controller) software architecture pattern. The system is divided into self-contained components that are easy to debug and maintain, with loose coupling and modularity between the fundamental parts. Figure 4 provides an overview. At the center of front-end lies the TableContainer class, connecting the following components:

- Handontable.js: Third party JavaScript spreadsheet framework.
- TableViewManager.js: Smart Assistant view controller.
- TableModel.js: Provides storage and accessibility to all core column entities and column heading labels.
- RestClient.js: Communication component, which is responsible for request sending and response provision via the respective callback calls.

4.3.2 Back-end. The back-end consists of two parts: a web server and a recommendation engine. The main role of the former is to connect the front-end spreadsheet application (client) with the recommendation engine. The web server is implemented in Python, using the Flask framework.[4] Communication is done over HTTP, with request and response messages encoded in JSON format. The recommendation engine is responsible for generating the ranked list of suggestions (entities and column labels). It uses Elasticsearch as the underlying indexing and retrieval engine. All indices are built using the Nordlys toolkit [4].[5]

5 EVALUATION

In previous work [10], we have performed an extensive evaluation of the row and column population methods in terms of effectiveness. Here, we evaluate our system in terms of efficiency. We measure response time as the time elapsed between receiving the request and sending off the response on the back-end, i.e., net computation time without the network overhead. Using 10 random tables, we vary the number of core column entities (seed entities) and the number of heading column labels (seed labels). The measurements

---

[4]http://flask.pocoo.org/
[5]http://nordlys.cc

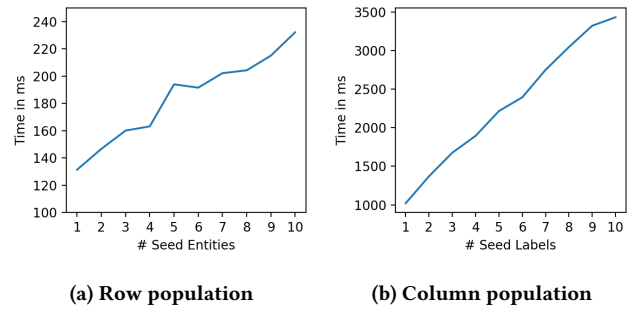| (a) Row population | (b) Column population |

Figure 5: Performance in terms of response time.

are repeated 10 times and averages are reported in Figs. 5a and 5b. We can observe that, in both cases, response time grows linearly with the size of the input. For row population, the response time is beyond 250ms, even with the largest input size, which is considered very acceptable. For column population, responses are a magnitude slower. This is due to the fact that we consider all related tables in our scoring formula. Limiting the computations to the top-k most similar tables may provide a solution; it is left for future work to find a k value that provides a good trade-off between effectiveness and efficiency.

6 CONCLUSION AND FUTURE WORK

We have introduced SmartTable, an online spreadsheet application that is equipped with smart assistance capabilities. Specifically, we aid users working with relational tables by suggesting them additional entities and column heading labels to be included in the table. In future work, we consider diversifying recommendations and plan to extend the scope of content recommendation to data cells as well, by suggesting possible values for them. Furthermore, we intend to integrate table search and table generation functionality, which we developed in recent work [11, 12].

REFERENCES

[1] Chandra Sekhar Bhagavatula, Thanapon Noraset, and Doug Downey. 2015. TabEL: Entity Linking in Web Tables. In *Proc. of ISWC '15*. 425–441.
[2] Marc Bron, Krisztian Balog, and Maarten de Rijke. 2013. Example Based Entity Search in the Web of Data. In *Proc. of ECIR '13*. 392–403.
[3] Michael J. Cafarella, Alon Halevy, and Nodira Khoussainova. 2009. Data Integration for the Relational Web. *Proc. of VLDB Endow.* 2 (2009), 1090–1101.
[4] Faegheh Hasibi, Krisztian Balog, Darío Garigliotti, and Shuo Zhang. 2017. Nordlys: A Toolkit for Entity-Oriented and Semantic Search. In *Proc. of SIGIR '17*. 1289–1292.
[5] Yeye He and Dong Xin. 2011. SEISA: set expansion by iterative similarity aggregation. In *Proc. of WWW '11*. 427–436.
[6] Panupong Pasupat and Percy Liang. 2015. Compositional Semantic Parsing on Semi-Structured Tables. In *Proc. of ACL '15*. 1470–1480.
[7] Chi Wang, Kaushik Chakrabarti, Yeye He, Kris Ganjam, Zhimin Chen, and Philip A. Bernstein. 2015. Concept Expansion Using Web Tables. In *Proc. of WWW '15*. 1198–1208.
[8] Yalin Wang and Jianying Hu. 2002. Detecting Tables in HTML Documents. In *Proc. of DAS '02*. 249–260.
[9] Yalin Wang and Jianying Hu. 2002. A Machine Learning Based Approach for Table Detection on the Web. In *Proc. of WWW '02*. 242–250.
[10] Shuo Zhang and Krisztian Balog. 2017. EntiTables: Smart Assistance for Entity-Focused Tables. In *Proc. of SIGIR '17*. 255–264.
[11] Shuo Zhang and Krisztian Balog. 2018. Ad Hoc Table Retrieval using Semantic Similarity. In *Proc. of WWW '18*. 1553–1562.
[12] Shuo Zhang and Krisztian Balog. 2018. On-the-fly Table Generation. In *Proc. of SIGIR '18*.