Hybrid.AI: A Learning Search Engine for Large-scale Structured Data

Sean Soderman, Anusha Kola, Maksim Podkorytov, Michael Geyer, Michael Gubanov
Department of Computer Science
University of Texas at San Antonio

ABSTRACT

Variety of Big data [17, 40, 44, 47, 52] is a significant impediment for anyone who wants to search inside a large-scale structured dataset. For example, there are millions of tables available on the Web, but the most relevant search result does not necessarily match the keyword-query exactly due to a variety of ways to represent the same information.

Here we describe **Hybrid.AI**, a learning search engine for large-scale structured data that uses automatically generated machine learning classifiers and Unified Famous Objects (UFOs) [33] to return the most relevant search results from a large-scale Web tables corpora. We evaluate it over this corpora, collecting *99 queries* and their results from users, and observe significant relevance gain.

ACM Reference Format:
Sean Soderman, Anusha Kola, Maksim Podkorytov, Michael Geyer, Michael Gubanov. 2018. Hybrid.AI: A Learning Search Engine for Large-scale Structured Data. In *WWW '18 Companion: The 2018 Web Conference Companion, April 23–27, 2018, Lyon, France.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3184558.3191600

1 INTRODUCTION

With the advent of large scale data management systems, data scientists and analysts have more information at their disposal than ever before. This influx of data makes retrieval of needed information challenging [6, 39, 47, 56]. Consider a data scientist working with structured data who has a recently mined large scale Web table dataset. If s/he wanted to enrich her information concerning weather in her region, s/he might be inclined to use keyword search in order to find the best records of interest in the dataset. However, a standard keyword-search over structured data, by nature, may provide inaccurate or incomplete search results, even when using sophisticated ranking functions, due to mismatches of relevant information to the query or the presence of relevant terms in irrelevant data rows [5]. Also, most structured data search engines return entire tables instead of the most relevant rows fused from many tables.

Although the properties of human language and text are the root cause of these issues, it is possible to decrease them by analyzing the semantic properties of data. This is what we do in Hybrid.AI, an intelligent search engine that automatically generates machine learning classifiers to identify similar data tuples to return more relevant search results compared to standard keyword search.

In order to generate the aforementioned classifiers, we make use of user-specified keywords to search for rows from tables that contain these keywords as attributes. These keywords are meant to be correlated with a certain object, for example, if we wanted to create a classifier for "jobs", our system would generate training data using keywords such as "salary", "date", and "position". Once this is done, we automatically train the classifier, then use it to cluster table rows that are likely to be job-oriented. Finally, we extract *core attributes* [33] from this set of classified rows. Informally *core attributes* are the most important attributes of an object (e.g. wings for a bird) a critical component of UFOs, a data structure used for fusing similar structured data objects that are represented in different ways [27, 41]. We use *core attributes* as soft constraints, improving the rank of results that are related to a specific object of interest. With this method, we get more relevant results, compared to the standard retrieval and ranking schemes for keyword search over structured data.

Our contributions in this paper are the following:

- **Machine-learning augmented Information Retrieval Scheme for Large-scale Structured Data**: We propose a new fusion-based information retrieval scheme for *structured* data that leverages machine learning classifiers and Unified Famous Objects (UFO) [33]. We extensively evaluate it over *99 queries*, using purely keyword-based retrieval as a baseline, on a large scale structured data set having millions of Web tables and observe significant retrieval relevance gain.
- **A Learning Search Engine for Large-scale Structured Data - Hybrid.AI**, marrying *keyword-search* with generated machine learning classifiers.

We are not the first who made an attempt to search Web tables. For example, [10] describes techniques borrowed from Web search to index and search Web tables. Another recent project focused on Web table search was [54], however, their domain was question answering instead of generalized search over structured data.

The rest of the paper is organized as follows. Section 2 discusses related work. Section 3 describes the system architecture and classifier generation. Section 4 describes a search scenario & classification scenario that illustrate our system. *UFOs*: Unified Famous Objects, are a structure used to abstract away differences in data representation [27, 33, 41]. See Section 5 for a more in-depth discussion on UFOs. To rank tuples from a large-scale corpus of Web tables from [28], we designed and evaluated several ranking functions *optimized* for large-scale structured data, making use of the best one for our system. Refer to Section 6 for more details on ranking. To evaluate our ranking, we compare it to an enhanced version of a standard ranking function popular in structured data search [5, 10, 13, 54] in Section 7. Future work is described in Section **??**. We conclude in Section 8.

WWW '18 Companion, April 23–27, 2018, Lyon, France

ACM ISBN 978-1-4503-5640-4/18/04.
https://doi.org/10.1145/3184558.3191600

2 RELATED WORK

We studied a variety of relevant systems in Web-search, large-scale data management, and information extraction/retrieval. [10] describes different search methods for Web tables. A key technique the authors use for improved search results is through using the AcsDB, a database of statistics concerning the table attributes in the corpus. It is used to compute a coherency score that uses PMI, Pointwise Mutual Information, which rewards items more likely to be paired together. The major difference between their system and our own is that it is ranking entire tables, whereas ours ranks each row (potentially originating from different tables) individually. This allows us to get the most relevant tuples from the entire corpus and compose it into a concise result set. [54] describe a question answering (Q&A) system for Web tables. This system retrieves individual cells from tables to answer a limited class of questions. For example, for the question - *what language do people in France speak*, it identifies the column *main language* in the table countries as the most relevant and lists languages from it. The authors also evaluate the effectiveness of their system using precision and recall.

We evaluate our system using nDCG [38] rather than precision and recall, because our system is a search-engine, not a Q&A system. nDCG takes not only precision and recall, but also *ordering* of the results into account, which is very important for a search-engine.

DBxplorer [5] is a keyword-search engine for relational databases that supports conjunctive keyword queries. It can also join tables, creating rows that contain *all* keywords from a search query, as well as attributes from different tables in this join result. This contrasts with our approach, which uses a more flexible *disjunctive* keyword search, fuses, and ranks tuples by relevance from millions of Web tables having different schemas. *Disjunctive* keyword-search allows retrieval of possibly relevant rows, despite not containing *all* search terms used. DBxplorer ranks rows by using the number of joins involved in the generation of a row rather than using weights of terms from the query and the tuples or more advanced techniques, outlined in Section 6. The authors' reasoning behind using the number of joins for ranking is that tables generated from several joins are harder to comprehend ([5], Section 6.2). This is somewhat similar to using keyword-proximity to help with ranking because tables must be joined until all keywords are present, however it does not consider stronger signals for relevance in search results.

[13] developed a ranking system for database queries. Rather than returning all tuples that satisfy a query, it calculates the *top-k* relevant tuples. To calculate relevance, it uses a global score that depends on user preferences, as well as a conditional score that considers the correlations between specified and unspecified terms in a query. In contrast, our ranking functions are dependent on the terms in the query and the relevance score of a tuple from the Web table.

In [16], a framework for defining the relatedness of tables based on whether or not they can be joined, as well as algorithms for detecting related tables that can be unioned or joined, was created. This was done to enable the retrieval of tables related to an input table, using this input table as a query rather than keywords. The authors defined two definitions for relatedness: *entity complement* and *schema complement.*

The former states that two tables T_1 and T_2, derived from a possibly nonexistent table T, must be created with selections over the same set of attributes in T, using different predicates. In addition to this, the combination of the selected tuples in T_1 and T_2 must consist of everything in T. Lastly, any projections from these selections must be on the same sets of attributes, as well as include the *subject columns*, which define the entity being described in the table.

An example of this would be a table about car models. If table T_1 had the attributes "model", "make", and "year", and the other table T_2 had attributes "name", "warranty" and "horsepower", it would be feasible that both of these tables were projected from a table containing all of these attributes, with some attribute expressing "name" or "model" since they are synonymous. To ensure that this measure of table relatedness makes sense, the authors also ensured the *coherency* of the virtual table. In order to do this, they ensured that the entities within T_1 and T_2 were of the same type, such as "name" and "author" both being identifiers. An example of an incoherent table would be one storing information about baseball cards and theater showtimes. Such a table would have a low coherency score. However, a table storing information about the 2016 and 2017 NBA championships would be deemed sensible.

Concerning *schema complement*, the two tables T_1 and T_2 would have to be created using queries Q_1 and Q_2 that have a similar structure. These queries must be in the form of a projection, additionally, they must select attributes so that T_1 and T_2 have at least one attribute not in common with one another, as well as at least one **in** common. Finally, the union of these two sets of project attributes must consist of the attributes in the virtual table T.

We create tables of related content using classifiers, instead of inferring whether tables could be *coherently* joined or unioned. Due to this, we do not employ such an artificial relatedness measure. We do not focus on ranking related tables, since we rank single tuples at a time in order to form a concise result set from thousands of tables. Also, because of our focus on single tuples, we have no use for inferring whether two results could be formed from similar SQL queries. Their objective of enhancing a user's tables is also different from our vision of unified retrieval of the most relevant tuples from the entire corpus. [12] developed a Web table service built on top of data derived from Microsoft's Bing Web search engine. Their search service uses machine learning to identify the entity column, which contains entities described by the values of other attributes in the table. This is done for queries such as "population of San Antonio" and "population of Bexar county", since both queries may match the same row even though the table may only describe the population of one of these entities. The authors also use static features such as number of rows and PageRank to aid in ranking tables, in addition to a feature based on the cell placement and column/row frequency of certain keyword matches. It is difficult to determine how effective their web table ranking is, since they do not provide an evaluation on it. We provide a thorough evaluation of our ranking in Section 7 using *nDCG*, a widely-used metric for assessing search result relevance [38]. Our method uses machine learning to identify rows belonging to objects of the same class, rather than identifying entities. We also rank using a combination of keyword intersection and *core attribute* matching, not taking into account the position of terms other than for our *keyword proximity* calculation. Finally, we return a combination of rows from different tables rather than an entire table at a time. This *consolidated* search result, including rows from many relevant tables, provides more relevant search results, compared to returning just one of a few entire relevant tables that may or may not have *all* rows relevant to the query.

2.1 Unified Famous Object (UFO): Definitions & Applications

[27, 32, 33, 41] developed a system that provides a unified, object-oriented way to query data from different data sources. Using UFOs hides structural differences between data sources and offers a queryable abstraction that is oblivious to this difference in metadata from different sources. It can also use the UFOs it has already constructed to identify new, familiar objects [3, 7, 14, 19, 20, 29–31, 34, 35, 46, 48]. This is different from our system since it is a system for *data integration* rather than *a search engine*. [33] describe and evaluate algorithms for UFO [27] creation. The authors illustrate the pre-UFO technique for matching attributes, such as BuyItNowPrice to Buy-It-Now-Price, as well as more dissimilar ones such as ConvertedCurrentPrice to CurrentPrice. This was done using exact matching, tokenized matching, and an NLP-enhanced version of the tokenized matching that uses part-of-speech tagging. The authors define *core properties*, or core attributes, which are the attributes that must be present for a UFO to exist. See [33] for more details. Our system focuses on generating these sets of *core properties* using term matching to count their occurrences [41]. We use them to assert the relevance of rows within the dataset, rather than building UFOs for identifying similar objects. [32] applied UFOs [27] to identify and fuse biomedical data, and used it for pre-diagnosing an early-onset Alzheimer's Disease. The authors demonstrated how UFOs simplified comparing a patient's DNA sequence with a reference sequence, resulting in a pre-diagnosis. Our system uses UFO's *core attributes* to boost search result rankings. Our search system also uses unstructured keyword queries rather than XQuery, translating them into SQL. Another key difference is how we return rows from many tables in a single, consolidated result.

3 ARCHITECTURE

Figure 1 displays an overview of our system's components. A brief description of each one follows.

3.1 Dataset

We use a large-scale Web tables dataset of ≈ 86 million Web table tuples (Approximately 55 million non-spam tuples) from [28, 55]. These instances came from Web tables pulled from sources such as online forums, social media sites, product offers, and others. They consist of data items or attributes from these web tables. We link each tuple to the web table it came from by storing an ID for that table in the tuple. We used [42, 53] to store said dataset.

3.2 Ingestion

Similar to E-mail or Web pages, tables extracted from the Web also have spam (examples include empty tables, HTML formatting, junk advertisements, etc) and require cleaning before ingestion. We trained our own J48 web table spam classifier [43, 55] to filter out tables with these characteristics. Using 10-fold cross-validation, a technique for estimating model performance [43] (in this case, the performance of our classifier), we observed 72.6% precision and 70.6% recall.

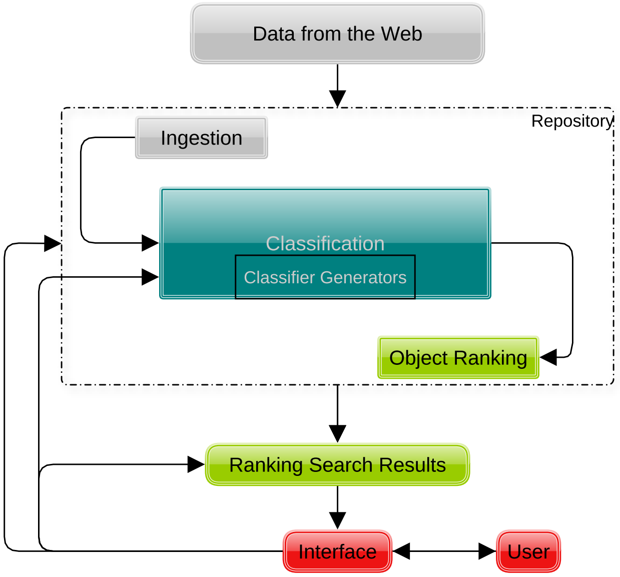


Figure 1: Hybrid.AI Architecture

3.3 Generating training data with SQL

After cleaning the data, once they are ingested into the parallel column store, we retrieve groups of similar objects using SQL queries. These queries are constructed from a set of keywords provided by the user (see Figure 4), such as "album", "title", and "price", which in this case may have been chosen to generate training data for a classifier to identify song oriented data.

3.4 Generating Scalable Machine Learning Classifiers

We automatically generate large-scale machine learning classifiers using the training data generated with the queries in the previous step. For example, the cluster of tables sharing attributes "trailer", "length", and "director" concerns movies, and after being trained on the data in this cluster, the classifier can identify more rows in the corpus associated with this movie data. Note that it is **not** necessarily the case that data positively labeled by this classifier will have the same set of attributes as the generated training data. We use *10-fold cross validation* [43] to evaluate precision/recall of generated classifiers, observing an average of 92.5% precision and 92.1% recall across nine classes of objects (e.g. songs, job postings, blog postings, real estate postings, etc) [19, 21–26, 45, 51].

3.5 Metadata Classifier

In order to extend our solution to be fully automated it is required that we identify those rows which contain attribute labels. That is, we must identify the *descriptive metadata* [18] of a table. A set of

900,000 records was processed to create a vector space with 6,900 features. From these rows, the training data was gathered using a series of rules. For example, one such rule was that from within a single html table, at most one row could be considered metadata. Another rule was that rows with low word count were more likely to be metadata. This training data was then manually checked and pruned for accuracy resulting in 6,500 negative samples and 540 positive samples. Using this training data we produced a *Support Vector Machine* classifier with a *linear kernel* [15] and a *one-vs-one decision function* [37] to identify rows containing metadata. This model was selected to help compensate for the small training set compared to the size of our vector space. After performing *10-fold cross validation* [43] to evaluate the precision and recall we observed an average of 80% precision and 64% recall.

3.6 Search

We designed a *machine-learning augmented* keyword search scheme using classifiers trained in the previous step to return the most relevant search results. Our ranking function uses the classifiers described in the previous paragraph, intersection, proximity, and UFOs [33] to rank search results. Refer to Sections 6 and 7 for more details on the ranking algorithms and their extensive evaluation.

3.7 Interface

An interactive user interface is used for executing search queries on our Web tables corpus. It also functions as a tool for training machine learning classifiers, generating & executing SQL queries for fetching training data (see Figure 4).

4 USAGE SCENARIOS

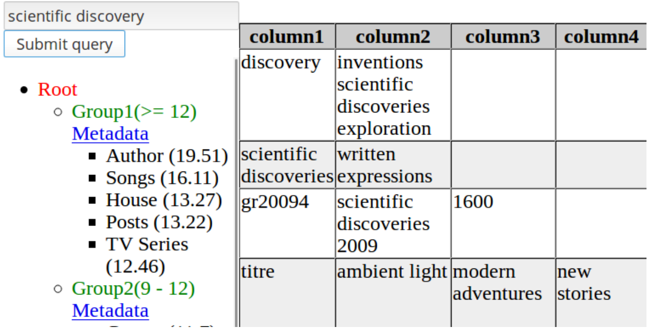
In Figures 2 and 3 are screenshots of the web frontend to our search engine corresponding to our two search scenarios. In the first scenario, the user searches data with the default mode, that is, an intersection-based, keyword-proximity augmented keyword search over Web table tuples. The second scenario enhances this method using UFO *core attributes*, which were collected from classified subsets of our large-scale dataset. This is done whenever a user enters a tag, like ":songs". Rather than filtering all data that does not topically match this tag, we use the attributes of web tables associated with this tag to increase the rank of results sharing those attributes. See Section 6 for more details regarding attribute-driven rank calculation.

After describing how our system carries out search, we illustrate how users can specify keywords, such as "make", "model", and "year", to fetch training data with these attributes (see Figure 4). In order to use our improved ranking function in the search scenarios, we need to first train then use these classifiers to identify all rows that fall under a certain category, such as songs.

4.1 Search Scenarios

Here the user wishes to find articles on scientific discoveries. Using the standard popular ranking functions for structured data gives us vague search results (Figure 2). The main problem with these results is that they contain little useful information, and might actually be pointers to some articles.

With our UFO core-attribute augmented querying scheme in Figure 3, the user specifies a class of objects of interest in the query, here *articles*, which makes the system perform an attribute-matching algorithm after the initial keyword-based retrieval. This algorithm rewards rows that came from tables that share attributes with article-oriented data (see Section 6 for more details).



- Root
  - Group1(>= 12)
    Metadata
    - Author (19.51)
    - Songs (16.11)
    - House (13.27)
    - Posts (13.22)
    - TV Series (12.46)
  - Group2(9 - 12)
    Metadata

column1	column2	column3	column4
discovery	inventions scientific discoveries exploration		
scientific discoveries	written expressions		
gr20094	scientific discoveries 2009	1600	
titre	ambient light	modern adventures	new stories

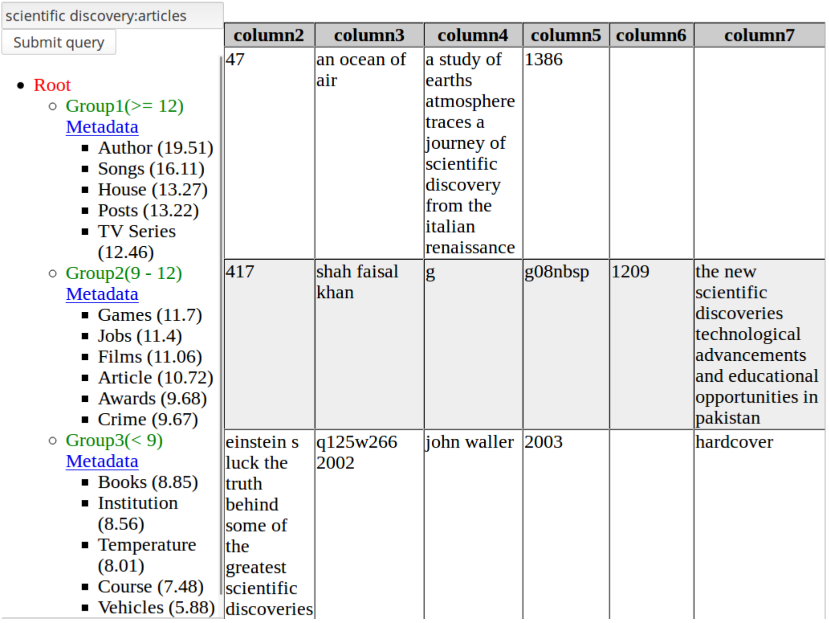
Figure 2: Search results from our large scale web-table corpus. Without a tag, we are faced with data that are most likely URLs or anchor text (rows 1 through 3) pertaining to scientific discovery, rather than titles of articles under this topic.

The results in Figure 3 are a major improvement over those in Figure 2. It is important to note that we do not only see an improvement in ranking (the results are actual scientific literature as opposed to what seem to be website elements), we also observe the ability of our system to return results with different attributes for the same query. This difference in attributes is apparent from the varied content of the data presented in the results in Figure 3.

4.2 Classifier Training Scenario

Here, we describe how to automatically generate a machine learning classifier, identifying rows of a certain type among millions in the corpus of Web tables. The user enters a few descriptive keywords [28] in the bottom left frame in Figure 4, for example "make", "model", "year". We use these keywords to fetch rows corresponding to the web tables that contain them as attributes. In addition to the attribute names, the user can enter lower and upper bounds on the row-length and column-length. Row length is the number of characters in a tuple of a Web table. The column-length is defined as row-length, but with respect to a column of a Web table. Such parameters are needed since we store all of the Web tables in our corpus within a single database table, and the number of attributes as well as rows varies between tables. The user can also pick checkboxes corresponding to each column to indicate which ones the filters apply to. Our interface allows the user to select up to 7 columns in this way. Refer to [28] for more details on classifier generation.

Once the keywords are in and the filters are selected, the user clicks *Generate Training Data* to retrieve training data having these attributes. A sample is shown to the user in the right frame of Figure 4. Finally, the user can select the classifier type (e.g. J48, Naive Bayes) by toggling a radio-button. After that, the user clicks



scientific discovery:articles

- Root
  - Group1(>= 12)
    Metadata
    - Author (19.51)
    - Songs (16.11)
    - House (13.27)
    - Posts (13.22)
    - TV Series (12.46)
  - Group2(9 - 12)
    Metadata
    - Games (11.7)
    - Jobs (11.4)
    - Films (11.06)
    - Article (10.72)
    - Awards (9.68)
    - Crime (9.67)
  - Group3(< 9)
    Metadata
    - Books (8.85)
    - Institution (8.56)
    - Temperature (8.01)
    - Course (7.48)
    - Vehicles (5.88)

column2	column3	column4	column5	column6	column7
47	an ocean of air	a study of earths atmosphere traces a journey of scientific discovery from the italian renaissance	1386		
417	shah faisal khan	g	g08nbsp	1209	the new scientific discoveries technological advancements and educational opportunities in pakistan
einstein s luck the truth behind some of the greatest scientific discoveries	q125w266 2002	john waller	2003		hardcover

Figure 3: Search results from our large scale web-table corpus. Using the tag "articles" improves the results drastically. We can see that the top two rows directly pertain to articles about scientific discoveries. The third is a book that debunks some scientific discoveries made by historically important scientists. Notice how each of these rows are *heterogeneous* in their structure, demonstrating the advantage of fetching individual rows as opposed to entire tables.

the *Generate Model* button, which triggers the process of training the machine-learning classifiers using these generated training data. The training set contains positively labeled training data, created as described above, and negatively labeled training data, which was made by selecting rows from the corpus without positive labels. We use 50% negative rows and 50% positive to create a balanced training set.

After the model is trained, it is run to classify the Web table rows and output a sample to the user in the right frame of Figure 4. If the user is satisfied with the model performance, s/he clicks the *Add to schema* button and the model is added to the list of objects in the left frame. After that, the user can click it in the left frame, and the classified data rows will be displayed in the right frame.

5 UNIFIED FAMOUS OBJECT - SONGS

UFO is an abstraction introduced in [27] to assist in data fusion of the same object from different data sources. An example of a UFO stored in JSON is illustrated in Listing 1. In this UFO for *Songs* the "name" attribute has a collection of its different representations accumulated from different data sources, including those in different languages.

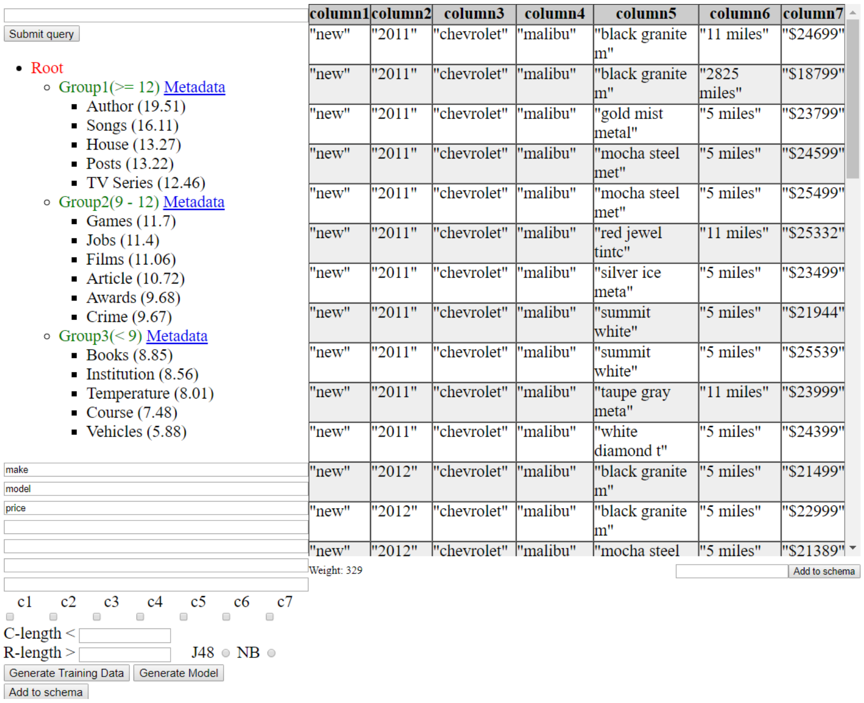
With regards to our usage of *core attributes*, Listing 1 demonstrates a sample of the kinds of attributes we collect and use in the algorithm described in Section 6. Attributes in different languages are still useful for gathering data that is associated with a certain object (here, "songs"). Refer to [27, 32, 33, 41] on UFO definition and more details on automatic UFO construction and data fusion using UFOs.

Listing 1: A Fragment of UFO Songs in JSON

```
{"Songs": {
        "name": ["name", "nom", "nome",
                "タイトル", "naam", "tÃtulo", "title",
                "lyrics", "nombre", "song"],
        "price": ["price", "preis", "prix",
                "prezzo", "prijs", "precio", "preço",
                "perhour"],
        "time": ["time", "length", "länge",
                "lengte", "durée", "durata",
                "duración", "duração"],
        "artist": ["artist", "artista", "artiest",
                "artiste", "interpret"],
        "album": ["album", "Ãilbum", "movie"],
        "download": ["download", "search"],
        "description": ["description",
                "descripción"],
        "music": ["music", "type"],
        "date": ["date", "datum"],
        "show": ["show"],
        "type": ["type", "all styles"]}
    }
```

6 RANKING OF SEARCH RESULTS

Here, we formally describe the two ranking methods we evaluate in Section 7 below. With regards to our baseline method, we used a combination of term *intersection*, which we found to be more effective than a term-frequency based approach, as well as augmented it with keyword proximity. Thus, the effectiveness of this approach



- Root
  - Group1(>= 12) Metadata
    - Author (19.51)
    - Songs (16.11)
    - House (13.27)
    - Posts (13.22)
    - TV Series (12.46)
  - Group2(9 - 12) Metadata
    - Games (11.7)
    - Jobs (11.4)
    - Films (11.06)
    - Article (10.72)
    - Awards (9.68)
    - Crime (9.67)
  - Group3(< 9) Metadata
    - Books (8.85)
    - Institution (8.56)
    - Temperature (8.01)
    - Course (7.48)
    - Vehicles (5.88)

column1	column2	column3	column4	column5	column6	column7
"new"	"2011"	"chevrolet"	"malibu"	"black granite m"	"11 miles"	"$24699"
"new"	"2011"	"chevrolet"	"malibu"	"black granite m"	"2825 miles"	"$18799"
"new"	"2011"	"chevrolet"	"malibu"	"gold mist metal"	"5 miles"	"$23799"
"new"	"2011"	"chevrolet"	"malibu"	"mocha steel met"	"5 miles"	"$24599"
"new"	"2011"	"chevrolet"	"malibu"	"mocha steel met"	"5 miles"	"$25499"
"new"	"2011"	"chevrolet"	"malibu"	"red jewel tintc"	"11 miles"	"$25332"
"new"	"2011"	"chevrolet"	"malibu"	"silver ice meta"	"5 miles"	"$23499"
"new"	"2011"	"chevrolet"	"malibu"	"summit white"	"5 miles"	"$21944"
"new"	"2011"	"chevrolet"	"malibu"	"summit white"	"5 miles"	"$25539"
"new"	"2011"	"chevrolet"	"malibu"	"taupe gray meta"	"11 miles"	"$23999"
"new"	"2011"	"chevrolet"	"malibu"	"white diamond t"	"5 miles"	"$24399"
"new"	"2012"	"chevrolet"	"malibu"	"black granite m"	"5 miles"	"$21499"
"new"	"2012"	"chevrolet"	"malibu"	"black granite m"	"5 miles"	"$22999"
"new"	"2012"	"chevrolet"	"malibu"	"mocha steel	"5 miles"	"$21389"

Figure 4: Automatic generation and training of a large-scale machine learning ensemble recognizing an object of interest, given several descriptive attributes from the user (for example - make, model, year). The user can enter the keywords, click the *Generate Training Data* button that will use the keywords to generate training data. Then, clicking the *Generate Model* button will trigger the process of automatically training the classifier with the generated training data.

makes the challenge of improving it more difficult.

6.1 Baseline Ranking

We use a derivative of an intersection-based keyword ranking scheme with proximity as a baseline to compare with. It is one of the most widely-adapted popular search schemes for structured data [5], on the Web [9], and in document search [36].

ti − idi (*Term Intersection - Inverse Document Intersection*): Our *ranking function* for large-scale structured *datasets* accounts for high redundancy of search keywords in arbitrary database rows, a phenomena we observed for such datasets. This could happen even when the row in question is not spam [55]. An example of this would be a row containing a large amount of information concerning nobility, the word "lady" could appear multiple times. Such a row would expose the weakness of traditional $tf - idf$ ranking in the context of databases when a user issues a query like "lady gaga songs". In this case, the irrelevant information about nobility would be ranked much higher than information about Lady Gaga's music, simply because the word "lady" appeared many times in the row. Thus, we defined the following ranking function to account for this problem, in addition to incorporating all standard ranking features of keyword search:

$$R_{int}(\rho, C, Q) = \sum_{t \in Q} ti(t, \rho) \times idi(C, t) \qquad (1)$$

Where ρ is the row we are ranking, C is our corpus and Q is the user's query. t is a single term $\in Q$. ti is defined as follows:

$$ti(t, \rho) = \begin{cases} 1 & \text{if } t \in \rho \\ 0 & \text{otherwise} \end{cases}$$

Notice that this function checks for the mere *existence* of a term within the row, so it is boolean. Hence, in Equation 1, we are effectively selecting the *idi* weights we wish to sum. These weights are defined by the following measure:

$$idi(C, t) = ln\left(\frac{|C|}{\sum_{i=0}^{|C|} |C_i \cap \{t\}|}\right) \qquad (2)$$

Intuitively, the numerator is the number of rows within the corpus. The denominator is the number of rows term t appears in within the corpus. This is scaled logarithmically so the closer we are to the row-cardinality of the corpus on the bottom, the closer our weight gets to 0. See [49] for justification & formal details regarding IDF.

As a final measure in this base ranking scheme, we use *keyword proximity* [8] to discount the rank of rows containing key terms that occur further apart. We take the total distance (starting at 1 when all words are adjacent) between the *closest* occurring keyword terms in a row and divide the **ti − idi** score by the natural logarithm of this value. This is since we do not wish to penalize the score of possibly relevant rows too harshly. We discount the **ti − idi** score by the natural logarithm of 100 if only *one* keyword is present in the row, as such a result is highly unlikely to be relevant. We came up with this constant experimentally, in order to prevent any rows with only one matched term from appearing if rows with more matched terms exist in the corpus.

6.2 Using trained classifiers and Unified Famous Objects to get more relevant search results

In this scheme, we first extract UFO *core attributes* [33, 41] from each Web tables' subset labeled by our trained classifier to belong to a certain class (e.g. songs), then use them to retrieve the most relevant search results from the entire corpus. We extract this set of core attributes with a query that fetches attribute rows that belong to the classified data. Each row in our corpus contains a filename, identifying the Web table it came from. We join the set of classified *data* rows with all *metadata* rows from the corpus on their filename. We then iterate through this table of attributes, counting the occurrences of all attributes and inserting the attribute frequencies into our table of core attributes. This step is done offline per UFO to reduce overhead during query-time. See [41] for more detail on UFO construction and core attributes. During online query processing, we load the pre-computed set of core attributes into a Java HashMap. We then load a result set of the rows containing at least one term from the query into memory, ranking them according to Equation 1. Finally, we increase the ranking of each tuple in the result set by matching each of its attributes to core attributes, increasing their rank by the natural logarithm of the sum of all matched core attribute frequencies. We perform this logarithmic scaling because of the high frequency of many of the attributes in our dataset. Doing this approximates matching an extra keyword for especially popular attributes, thus making it easier to see how core attributes affect the ranking of rows. An even better ranking could be achieved by using Unified Famous Objects (UFO) not only for core attributes, but also for attribute matching [33]. This will enable us to go beyond simple keyword matching, matching words that are either synonymous or closely related to one another to improve a row's rank. See [33] for details on UFOs and object recognition evaluation for UFOs.

7 EVALUATION

It is important to assess the effectiveness of using our Classifiers+UFOs ranking scheme for structured data. To gauge relevance gain over 99 different queries we collected from users, we compare it with the baseline ranking described above in the "Baseline Ranking" paragraph in Section 6.

To evaluate relevance gain provided by our ranking function, we asked 28 students taking a database class to come up with a few queries they might be interested in executing over our Web tables dataset. After accumulating *99 different queries*, we asked two independent evaluators to assign relevance labels from 1-Bad to 5-Perfect to all (query, search result) pairs generated by our two ranking schemes for these queries. We took 15 top results for each query. We dropped a label whenever the evaluators disagreed. Based on these labels and ranking by our schemes, we took the top 15 results from each query to compute their $nDCG_{15}$, which is an industrial standard measure to evaluate search relevance [4]. It stands for "Normalized Discounted Cumulative Gain" and reflects relevance of search results to the specific query. It computes a value based on user-assigned relevance labels to rows in a result set, discounting their contribution to the overall *DCG* score the lower their position is in the result set. The following equation formalizes this intuition:

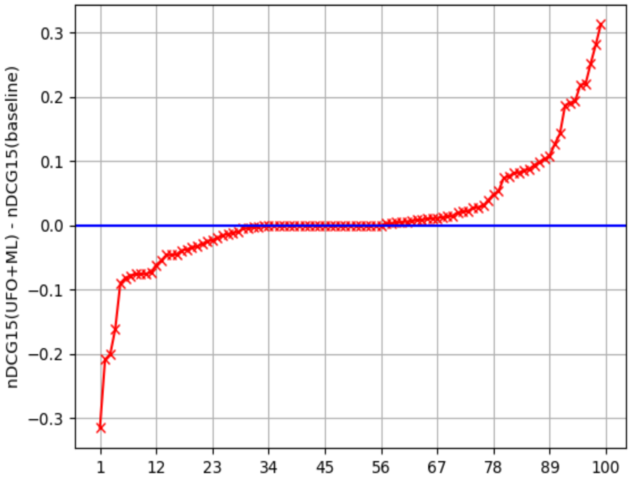


Figure 5: Search relevance evaluation of our system over 99 user queries. Each point is the delta in $nDCG_{15}$ between our AI-augmented and baseline ranking functions.

$$DCG_n(S) = \sum_{i=0}^{n} \frac{S_i.label}{log_2(i+1)} \tag{3}$$

Where S is the set of ranked rows and *label* is the value of the user-assigned relevance label for the row, and n is the number of results from the result set we consider from the query. To compute the *normalized* DCG, we must first compute *iDCG*, the *DCG* value of a search engine that is able to rank the results perfectly. By sorting relevant S by *label* (in descending order), then computing *DCG*, we get *iDCG*. Dividing the *DCG* by *iDCG* then produces our *nDCG* value for a particular query. *nDCG* thusly rewards highly relevant results when their position is more appropriate (higher). Refer to [38] for more formal details on *nDCG*.

In Figure 5, each point is the delta between the $nDCG_{15}$ value of the same query run using two different query processing schemes. The first is *with* UFO core attributes and classifiers, the second is the baseline ranking scheme. Many queries get better as we can see from the graph, at the same time there are queries that got worse. For example, the largest relevance decrease we observed was for the query "fifth element", where several of the most relevant results did not have proper metadata attributes in our dataset. This, combined with terms from iTunes being present in the set of film core attributes, caused the query to fetch more song data for this query instead of film data. More queries improve overall, which is reflected by an average 1.5% increase in nDCG over all 99 queries, which is considered significant in web search [4].

8 CONCLUSION

We described Hybrid.AI - a learning search engine marrying machine learning with keyword-search. We justified that our *Machine Learning+UFO* augmented keyword-search returns more relevant search results than a standard ranking function for keyword search over structured data [1, 2, 11, 50]. We gauged the relevance gain of our algorithms using nDCG, a de-facto standard relevance measure, employed by major Web-search engines [4], and observed significant gain on 99 user queries over a large scale Web table corpus.

9 ACKNOWLEDGMENTS

We would like to thank anonymous reviewers for their feedback on earlier drafts of this paper. This material is based upon work supported by the National Science Foundation (NSF) under Grant No. 1701081.

REFERENCES

[1] [n. d.]. Hybrid.AI: An AI-Augmented Search Engine for Large-scale Structured Data. In *MIT Annual Database Research Conference.*

[2] [n. d.]. Hybrid.Poly: An Interactive Large-scale In-memory Analytical Polystore. In *MIT Annual Database Research Conference.*

[3] Ziawasch Abedjan, John Morcos, Michael Gubanov, Ihab F. Ilyas, Michael Stonebraker, Paolo Papotti, and Mourad Ouzzani. 2015. Dataxformer: Leveraging the Web for Semantic Transformations. In *CIDR.*

[4] Eugene Agichtein, Eric Brill, and Susan Dumais. 2006. Improving Web Search Ranking by Incorporating User Behavior Information. In *SIGIR.*

[5] Sanjay Agrawal, Surajit Chaudhuri, and Gautam Das. 2002. DBXplorer: A system for keyword-based search over relational databases. In *ICDE.*

[6] Albin Ahmeti, Javier D. Fernández, Axel Polleres, and Vadim Savenkov. 2017. Updating Wikipedia via DBpedia Mappings and SPARQL. In *ESWC.*

[7] Bogdan Alexe, Michael Gubanov, Mauricio A. Hernández, C. T. Howard Ho, Jen-Wei Huang, Yannis Katsis, Lucian Popa, Barna Saha, and Ioana Stanoi. 2008. Simplifying Information Integration: Object-Based Flow-of-Mappings Framework for Integration. In *BIRTE.*

[8] Ricardo Baeza-Yates and Walter Cunto. 1999. The ADT proximity and text proximity problems. In *SPIRS.* IEEE, 24–30.

[9] Sergey Brin and Lawrence Page. 1998. The Anatomy of a Large-scale Hypertextual Web Search Engine. In *WWW.*

[10] Michael J Cafarella, Alon Halevy, Daisy Zhe Wang, Eugene Wu, and Yang Zhang. 2008. Webtables: exploring the power of tables on the web. *VLDB* (2008).

[11] Michael A. Casey, Christophe Rhodes, and Malcolm Slaney. 2008. Analysis of Minimum Distances in High-Dimensional Musical Spaces. *IEEE TASLP* 16 (2008), 1015–1028.

[12] Kaushik Chakrabarti, Surajit Chaudhuri, Zhimin Chen, Kris Ganjam, Yeye He, and WA Redmond. 2016. Data services leveraging Bing's data assets. *IEEE Data Eng. Bull.* (2016), 15–28.

[13] Surajit Chaudhuri, Gautam Das, Vagelis Hristidis, and Gerhard Weikum. 2004. Probabilistic ranking of database query results. In *VLDB.* 888–899.

[14] Surya Cheemalapati, Michael Gubanov, Michael Del Vale, and Anna Pyayt. 2013. A real-time classification algorithm for emotion detection using portable EEG. In *IRI.*

[15] Corinna Cortes and Vladimir Vapnik. 1995. Support-vector networks. *Machine learning* 20, 3 (1995), 273–297.

[16] Anish Das Sarma, Lujun Fang, Nitin Gupta, Alon Halevy, Hongrae Lee, Fei Wu, Reynold Xin, and Cong Yu. 2012. Finding related tables. In *SIGMOD.*

[17] Hady ElSahar, Elena Demidova, Simon Gottschalk, Christophe Gravier, and Frédérique Laforest. 2018. Unsupervised Open Relation Extraction. *CoRR* abs/1801.07174 (2018).

[18] Jane Greenberg. 2005. Understanding metadata and metadata schemes. *Cataloging & classification quarterly* 40, 3-4 (2005), 17–36.

[19] Michael Gubanov. 2017. Hybrid: A Large-scale In-memory Image Analytics System. In *CIDR.*

[20] Michael Gubanov. 2017. PolyFuse: A Large-scale Hybrid Data Fusion System. In *ICDE DESWeb.*

[21] Michael Gubanov and Philip A. Bernstein. 2006. Structural text search and comparison using automatically extracted schema.. In *WebDB.*

[22] Michael Gubanov, Philip A. Bernstein, and Alexander Moshchuk. 2008. Model Management Engine for Data Integration with Reverse-Engineering Support. In *ICDE.*

[23] Michael Gubanov, Chris Jermaine, Zekai Gao, and Shangyu Luo. 2016. Hybrid: A Large-scale Linear-relational Database Management System. In *MIT Annual DB Conference.*

[24] Michael Gubanov, Shangyu Luo, Zekai Gao, Luis Perez, and Christopher Jermaine. 2017. Scalable Linear Algebra on a Relational Database System. In *ICDE.*

[25] Michael Gubanov, Shangyu Luo, Zekai Gao, Luis Perez, and Christopher Jermaine. 2018. Scalable Linear Algebra on a Relational Database System. In *to apear in TKDE.*

[26] Michael Gubanov, Shangyu Luo, Zekai Gao, Luis Perez, and Christopher Jermaine. 2018. Scalable Linear Algebra on a Relational Database System. In *to appear in ACM SIGMOD Record.*

[27] Michael Gubanov, Lucian Popa, Howard Ho, Hamid Pirahesh, Jeng-Yih Chang, and Shr-Chang Chen. 2009. IBM UFO repository: Object-oriented data integration. *VLDB* (2009).

[28] Michael Gubanov, Manju Priya, and Maksim Podkorytov. 2017. CognitiveDB: An Intelligent Navigator for Large-scale Dark Structured Data. In *WWW.*

[29] M. Gubanov and A. Pyayt. 2013. ReadFast: High-relevance Search-engine for Big Text. In *ACM CIKM.*

[30] Michael Gubanov and Anna Pyayt. 2016. Type-aware Web search. In *EDBT.*

[31] Michael Gubanov, Anna Pyayt, and Linda Shapiro. 2011. ReadFast: Browsing large documents through UFO. In *IRI.*

[32] Michael Gubanov and Linda Shapiro. 2012. Using Unified Famous Objects (UFO) to Automate Alzheimer's Disease Diagnostics. In *BIBM.*

[33] Michael Gubanov, Linda Shapiro, and Anna Pyayt. 2011. Learning Unified Famous Objects (UFO) to Bootstrap Information Integration. In *IRI.*

[34] Michael Gubanov and Michael Stonebraker. 2014. Large-scale Semantic Profile Extraction. In *EDBT.*

[35] Michael Gubanov, Michael Stonebraker, and Daniel Bruckner. 2014. Text and Structured Data Fusion in Data Tamer at Scale. In *ICDE.*

[36] Lin Guo, Feng Shao, Chavdar Botev, and Jayavel Shanmugasundaram. 2003. XRANK: Ranked keyword search over XML documents. In *SIGMOD.* ACM.

[37] Chih-Wei Hsu and Chih-Jen Lin. 2002. A comparison of methods for multiclass support vector machines. *TNN* 13, 2 (2002), 415–425.

[38] Kalervo Järvelin and Jaana Kekäläinen. 2000. IR evaluation methods for retrieving highly relevant documents. In *SIGIR.* ACM.

[39] Emilia Kacprzak, Laura M. Koesten, Luis Daniel Ibáñez, Elena Simperl, and Jeni Tennison. 2017. A Query Log Analysis of Dataset Search. In *ICWE.*

[40] Laura M. Koesten, Emilia Kacprzak, Jenifer Fay Alys Tennison, and Elena Simperl. 2017. The Trials and Tribulations of Working with Structured Data: a Study on Information Seeking Behaviour. In *CHI.*

[41] Anusha Kola, Harshal More, Sean Soderman, and Michael Gubanov. 2017. Generating Unified Famous Objects (UFOs) from the classified object tables. In *IEEE Big Data.*

[42] Marcel Kornacker and Alexander Behm et al. 2015. Impala: A Modern, Open-Source SQL Engine for Hadoop. In *CIDR.*

[43] Thomas M. Mitchell. 1997. *Machine Learning.* McGraw-Hill, Inc., New York, NY, USA.

[44] Tope Omitola, Sebastián A. Ríos, and John G. Breslin. 2015. *Social Semantic Web Mining.* Morgan & Claypool Publishers.

[45] Steven Ortiz, Caner Enbatan, Maksim Podkorytov, Dylan Soderman, and Michael Gubanov. 2017. Hybrid.JSON: High-velocity Parallel In-Memory Polystore JSON Ingest. In *IEEE Bigdata.*

[46] Manju Priya, Maxim Podkorytov, and Michael Gubanov. 2017. iLight: A Flashlight for Large-scale Dark Structured Data. In *MIT Annual DB Conference.*

[47] Freddy Priyatna, Edna Ruckhaus, Nandana Mihindukulasooriya, Óscar Corcho, and Nelson Saturno. 2017. MappingPedia: A Collaborative Environment for R2RML Mappings. In *ESWC.*

[48] Anna Pyayt and Michael Gubanov. 2013. BigDB: Automatic Machine Learning Optimizer. *CoRR* abs/1301.1575 (2013).

[49] Stephen Robertson. 2004. Understanding inverse document frequency: on theoretical arguments for IDF. *Journal of documentation* 60, 5 (2004), 503–520.

[50] G. Salton, A. Wong, and C. S. Yang. 1975. A Vector Space Model for Automatic Indexing. *CACM* 18, 11 (Nov. 1975), 613–620.

[51] Mark Simmons, Daniel Armstrong, Dylan Soderman, and Michael Gubanov. 2017. Hybrid.media: High Velocity Video Ingestion in an In-Memory Scalable Analytical Polystore. In *IEEE Bigdata.*

[52] Michael Stonebraker. 2012. Big Data Means at Least Three Different Things.... In *NIST Big Data Workshop.*

[53] Mike Stonebraker, Daniel Abadi, and Adam Batkin et al. 2005. C-store: A Column-oriented DBMS. In *VLDB.*

[54] Huan Sun, Hao Ma, Xiaodong He, Wen-tau Yih, Yu Su, and Xifeng Yan. 2016. Table cell search for question answering. In *WWW.*

[55] Santiago Villasenor, Tom Nguyen, Anusha Kola, Sean Soderman, and Michael Gubanov. 2017. Scalable spam classifier for web tables. In *IEEE Big Data.*

[56] Ran Yu, Ujwal Gadiraju, Besnik Fetahu, and Stefan Dietze. 2017. FuseM: Query-Centric Data Fusion on Structured Web Markup. In *ICDE.*