Sherlock: A Deep Learning Approach to Semantic Data Type Detection

Sherlock: A Deep Learning Approach to Semantic Data Type Detection

Madelon Hulsebos
MIT Media Lab
madelonhulsebos@gmail.com

Kevin Hu
MIT Media Lab
kzh@mit.edu

Michiel Bakker
MIT Media Lab
bakker@mit.edu

Emanuel Zgraggen
MIT CSAIL
emzg@mit.edu

Arvind Satyanarayan
MIT CSAIL
arvindsatya@mit.edu

Tim Kraska
MIT CSAIL
kraska@mit.edu

Çağatay Demiralp
Megagon Labs
cagatay@megagon.ai

César Hidalgo
MIT Media Lab
hidalgo@mit.edu

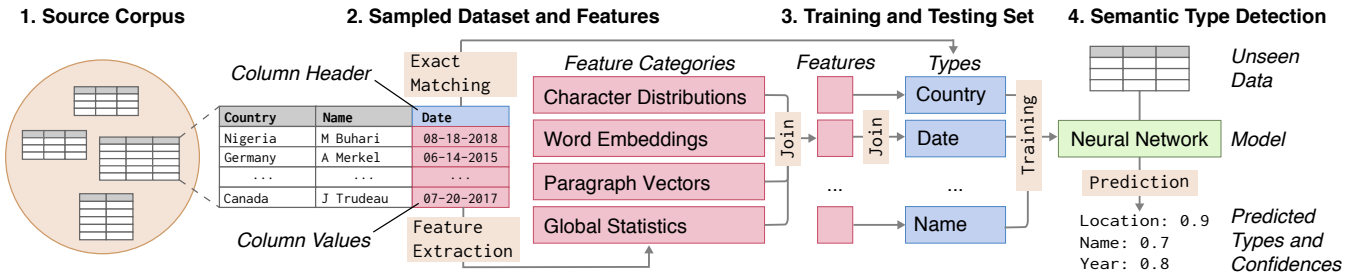

Figure 1: Data processing and analysis flow, starting from (1) a corpus of real-world datasets, proceeding to (2) feature extraction, (3) mapping extracted features to ground truth semantic types, and (4) model training and prediction.

ABSTRACT

Correctly detecting the semantic type of data columns is crucial for data science tasks such as automated data cleaning, schema matching, and data discovery. Existing data preparation and analysis systems rely on dictionary lookups and regular expression matching to detect semantic types. However, these matching-based approaches often are not robust to dirty data and only detect a limited number of types. We introduce Sherlock, a multi-input deep neural network for detecting semantic types. We train Sherlock on 686,765 data columns retrieved from the VizNet corpus by matching 78 semantic types from DBpedia to column headers. We characterize each matched column with 1,588 features describing the statistical properties, character distributions, word embeddings, and paragraph vectors of column values. Sherlock achieves a support-weighted F_1 score of 0.89, exceeding that of machine learning baselines, dictionary and regular expression benchmarks, and the consensus of crowdsourced annotations.

CCS CONCEPTS

• **Computing methodologies** → **Machine learning**; **Knowledge representation and reasoning**; • **Information systems** → **Data mining**.

KDD '19, August 4–8, 2019, Anchorage, AK, USA

ACM ISBN 978-1-4503-6201-6/19/08…$15.00
https://doi.org/10.1145/3292500.3330993

KEYWORDS

Tabular data, type detection, semantic types, deep learning

ACM Reference Format:
Madelon Hulsebos, Kevin Hu, Michiel Bakker, Emanuel Zgraggen, Arvind Satyanarayan, Tim Kraska, Çağatay Demiralp, and César Hidalgo. 2019. Sherlock: A Deep Learning Approach to Semantic Data Type Detection. In *The 25th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '19), August 4–8, 2019, Anchorage, AK, USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3292500.3330993

1 INTRODUCTION

Data preparation and analysis systems rely on correctly detecting types of data columns to enable and constrain functionality. For example, automated data cleaning facilitates the generation of clean data through validation and transformation rules that depend on data type [15, 26]. Schema matching identifies correspondences between data objects, and frequently uses data types to constrain the search space of correspondences [25, 35]. Data discovery surfaces data relevant to a given query, often relying on semantic similarities across tables and columns [6, 7].

While most systems reliably detect *atomic types* such as `string`, `integer`, and `boolean`, *semantic types* are disproportionately more powerful and in many cases essential. Semantic types provide finer-grained descriptions of the data by establishing correspondences between columns and real-world concepts and as such, can help with schema matching to determine which columns refer to the same real-world concepts, or data cleaning by determining the conceptual domain of a column. In some cases, the detection of a semantic type can be easy. For example, an `ISBN` or `credit card number` are generated according to strict validation rules, lending themselves to straightforward type detection with just a few rules.

Table 1: Data values sampled from real-world datasets.

Type	Sampled values
location	TBA | Chicago, Ill. | Detroit, Mich. | Nashville, Tenn.
location	UNIVERSITY SUITES | U.S. 27; NA | NORSE HALL
location	Away | Away | Home | Away | Away
date	27 Dec 1811 | 1852 | 1855 | - | 1848 | 1871 | 1877
date	– –, 1922 | – –, 1902 | – –, 1913 | – –, 1919
date	December 06 | August 23 | None
name	Svenack | Svendd | Sveneldritch | Svengöran
name	HOUSE, BRIAN | HSIAO, AMY | HSU, ASTRID
name	D. Korb | K. Moring | J. Albanese | l. dunn

But most types, including location, birth date, and name, do not adhere to such structure, as shown in Table 1.

Existing open source and commercial systems take *matching-based* approaches to semantic type detection. For example, regular expression matching captures patterns of data values using predefined character sequences. Dictionary approaches use matches between data headers and values with internal look-up tables. While sufficient for detecting simple types, these matching-based approaches are often not robust to malformed or dirty data, support only a limited number of types, and under-perform for types without strict validations. For example, Figure 2 shows that Tableau detects a column labeled "Continent Name" as string. After removing column headers, no semantic types are detected. Note that missing headers or incomprehensible headers are not uncommon. For example, SAP's system table *T*005 contains country information and column *NMFMT* is the standard name field, whereas *INTCA* refers to the ISO code or *XPLZS* to zip-code.

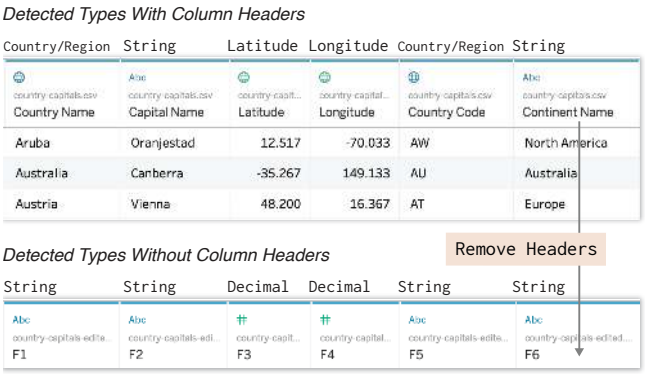

Figure 2: Data types detected by Tableau Desktop 2018.3 for a dataset of country capitals, with and without headers.

Machine learning models, coupled with large-scale training and benchmarking corpora, have proven effective at predictive tasks across domains. Examples include the AlexNet neural network trained on ImageNet for visual recognition and the Google Neural Machine Translation system pre-trained on WMT parallel corpora for language translation. Inspired by these advances, we introduce **Sherlock**, a deep learning approach to semantic type detection trained on a large corpus of real-world columns.

To begin, we consider 78 semantic types described by T2Dv2 Gold Standard,[1] which matches properties from the DBpedia ontology with column headers from the WebTables corpus. Then, we use exact matching between semantic types and column headers to extract 686,765 data columns from the VizNet corpus [14], a large-scale repository of real world datasets collected from the web, popular visualization systems, and open data portals.

We consider each column as a mapping from column values to a column header. We then extract 1,588 features from each column, describing the distribution of characters, semantic content of words and columns, and global statistics such as cardinality and uniqueness. Treating column headers as ground truth labels of the semantic type, we formulate semantic type detection as a multiclass classification problem.

A multi-input neural network architecture achieves a support-weighted F_1-score of 0.89, exceeding that of decision tree and random forest baseline models, two matching-based approaches that represent type detection approaches in practice, and the consensus of crowdsourced annotations. We then examine types for which the neural network demonstrates high and low performance, investigate the contribution of each feature category to model performance, extract feature importances from the decision tree baseline, and present an error-reject curve suggesting the potential of combining learned models with human annotations.

To conclude, we discuss promising avenues for future research in semantic type detection, such as assessing training data quality at scale, enriching feature extraction processes, and establishing shared benchmarks. To support benchmarks for future research and integration into existing systems, we open source our data, code, and trained model at https://sherlock.media.mit.edu.

Key contributions:

(1) **Data** (§3): Demonstrating a scalable process for matching 686,675 columns from VizNet corpus for 78 semantic types, then describing with 1,588 features like word- and paragraph embeddings.
(2) **Model** (§4): Formulating type detection as a multiclass classification problem, then contributing a novel multi-input neural network architecture.
(3) **Results** (§5): Benchmarking predictive performance against a decision tree and random forest baseline, two matching-based models, and crowdsourced consensus.

2 RELATED WORK

Sherlock is informed by existing *commercial and open source systems* for data preparation and analysis, as well as prior research work on *ontology-based*, *feature-based*, *probabilistic*, and *synthesized* approaches to semantic type detection.

Commercial and open source. Semantic type detection enhances the functionality of commercial data preparation and analysis systems such as Microsoft Power BI [20], Trifacta [31], and Google Data Studio [12]. To the best of our knowledge, these commercial tools rely on manually defined regular expression patterns dictionary lookups of column headers and values to detect a limited set of

[1] http://webdatacommons.org/webtables/goldstandardV2.html

semantic types. For instance, Trifacta detects around 10 types (e.g., `gender` and `zip code`) and Power BI only supports time-related semantic types (e.g., `date/time` and `duration`). Open source libraries such as messytables [10], datalib [9], and csvkit [13] similarly use heuristics to detect a limited set of types. Benchmarking directly against these systems was infeasible due to the small number of supported types and lack of extensibility. However, we compare against learned regular expression and dictionary-based benchmarks representative of the approaches taken by these systems.

Ontology-based. Prior research work, with roots in the semantic web and schema matching literature, provide alternative approaches to semantic type detection. One body of work leverages existing data on the web, such as WebTables [5], and ontologies (or, knowledge bases) such as DBPedia [2], Wikitology [30], and Freebase [4]. Venetis et al. [33] construct a database of value-type mappings, then assign types using a maximum likelihood estimator based on column values. Syed et al. [30] use column headers and values to build a Wikitology query, the result of which maps columns to types. Informed by these approaches, we looked towards existing ontologies to derive the 275 semantic types considered in this paper.

Feature-based. Several approaches capture and compare properties of data in a way that is ontology-agnostic. Ramnandan et al. [27] use heuristics to first separate numerical and textual types, then describe those types using the Kolmogorov-Smirnov (K-S) test and Term Frequency-Inverse Document Frequency (TF-IDF), respectively. Pham et al. [23] use slightly more features, including the Mann-Whitney test for numerical data and Jaccard similarity for textual data, to train logistic regression and random forest models. We extend these feature-based approaches with a significantly larger set of features that includes character-level distributions, word embeddings, and paragraph vectors. We leverage orders of magnitude more features and training samples than prior work in order to train a high-capacity machine learning model, a deep neural network. We include a decision tree and random forest model as benchmarks to represent these "simpler" machine learning models.

Probabilistic. The third category of prior work employs a probabilistic approach. Goel et al. [11] use conditional random fields to predict the semantic type of each value within a column, then combine these predictions into a prediction for the whole column. Limaye et al. [19] use probabilistic graphical models to annotate values with entities, columns with types, and column pairs with relationships. These predictions simultaneously maximize a potential function using a message passing algorithm. Probabilistic approaches are complementary to our machine learning-based approach by providing a means for combining column-specific predictions. However, as with prior feature-based models, code for retraining these models was not made available for benchmarking.

Synthesized. Puranik [24] proposes a "specialist approach" combining the predictions of regular expressions, dictionaries, and machine learning models. More recently, Yan and He [34] introduced a system that, given a search keyword and set of positive examples, synthesizes type detection logic from open source GitHub repositories. This system provides a novel approach to leveraging domain-specific heuristics for parsing, validating, and transforming semantic data types. While both approaches are exciting, the code underlying these systems was not available for benchmarking.

3 DATA

We describe the semantic types we consider, how we extracted data columns from a large repository of real-world datasets, and our feature extraction procedure.

3.1 Data Collection

Ontologies like WordNet [32] and DBpedia [2] describe semantic concepts, properties of such concepts, and relationships between them. To constrain the number of types we consider, we adopt the types described by the T2Dv2 Gold Standard,[1] the result of a study matching DBpedia properties [29] with columns from the Web Tables web crawl corpus [5]. These 275 DBpedia properties, such as `country`, `language`, and `industry`, represent semantic types commonly found in datasets scattered throughout the web.

To expedite the collection of real-world data from diverse sources, we use the VizNet repository [14], which aggregates and characterizes data from two popular online visualization platforms and open data portals, in addition to the Web Tables corpus. For feasibility, we restricted ourselves to the first 10M Web Tables datasets, but considered the remainder of the repository in its entirety. We then match data columns from VizNet that have headers corresponding to our 275 types. To accomodate variation in casing and formatting, single word types matched case-altered modifications (e.g., `name` = `Name` = `NAME`) and multi-word types included concatenations of constituent words (e.g., `release date` = `releaseDate`).

The matching process resulted in 6,146,940 columns matching the 275 considered types. Manual verification indicated that the majority of columns were plausibly described by the corresponding semantic type, as shown in Table 1. In other words, matching column headers as ground truth labels of the semantic type yielded high quality training data.

3.2 Feature Extraction

To create fixed-length representations of variable-length columns, aid interpretation of results, and provide "hints" to our neural network, we extract features from each column. To capture different properties of columns, we extract four categories of features: global statistics (27), aggregated character distributions (960), pretrained word embeddings (200), and self-trained paragraph vectors (400).

Global statistics. The first category of features describes high-level statistical characteristics of columns. For example, the "column entropy" feature describes how uniformly values are distributed. Such a feature helps differentiate between types that contain more repeated values, such as `gender`, from types that contain many unique values, such as `name`. Other types, like `weight` and `sales`, may consist of many numerical characters, which is captured by the "mean of the number of numerical characters in values." A complete list of these 27 features can be found in Table 8 in the Appendix.

Character-level distributions. Preliminary analysis indicated that simple statistical features such as the "fraction of values with numerical characters" provide surprising predictive power. Motivated

by these results and the prevalence of character-based matching approaches such as regular expressions, we extract features describing the distribution of characters in a column. Specifically, we compute the count of all 96 ASCII-printable characters (i.e., digits, letters, and punctuation characters, but not whitespace) within each value of a column. We then aggregate these counts with 10 statistical functions (i.e., any, all, mean, variance, min, max, median, sum, kurtosis, skewness), resulting in 960 features. Example features include "whether all values contain a '-' character" and the "mean number of '/' characters."

Word embeddings. For certain semantic types, columns frequently contain commonly occurring words. For example, the `city` type contains values such as *New York City*, *Paris*, and *London*. To characterize the semantic content of these values, we used word embeddings that map words to high-dimensional fixed-length numeric vectors. In particular, we used a pre-trained GloVe dictionary [22] containing 50-dimensional representations of 400K English words aggregated from 6B tokens, used for tasks such as text similarity [16]. For each value in a column, if the value is a single word, we look up the word embedding from the GloVe dictionary. We omit a term if it does not appear in the GloVe dictionary. For values containing multiple words, we looked up each distinct word and represented the value with the mean of the distinct word vectors. Then, we computed the mean, mode, median and variance of word vectors across all values in a column.

Paragraph vectors. To represent each column with a fixed-length numerical vector, we implemented the Distributed Bag of Words version of Paragraph Vector (PV-DBOW) [18]. Paragraph vectors were originally developed to numerically represent the "topic" of pieces of texts, but have proven effective for more general tasks, such as document similarity [8]. In our implementation, each column is a "paragraph" while values within a column are "words": both the entire column and constituent values are represented by one-hot encoded vectors.

After pooling together all columns across all classes, the training procedure for each column in the same 60% training set used by the main Sherlock model is as follows. We randomly select a window of value vectors, concatenate the column vector with the remaining value vectors, then train a single model to predict the former from the latter. Using the Gensim library [28], we trained this model for 20 iterations. We used the trained model to map each column in both the training and test sets to a 400-dimensional paragraph vector, which provided a balance between predictive power and computational tractability.

3.3 Filtering and Preprocessing

Certain types occur more frequently in the VizNet corpus than others. For example, `description` and `city` are more common than `collection` and `continent`. To address this heterogeneity, we limited the number of columns to at most 15K per class and excluded the 10% types containing less than 1K columns.

Other semantic types, especially those describing numerical concepts, are unlikely to be represented by word embeddings. To contend with this issue, we filtered out the types for which at least 15% of the columns did not contain a single word that is present in the GloVe dictionary. This filter resulted in a final total of **686,765 columns** corresponding to **78 semantic types**, of which a list is included in Table 7 in the Appendix. The distribution of number of columns per semantic type is shown in Figure 3.

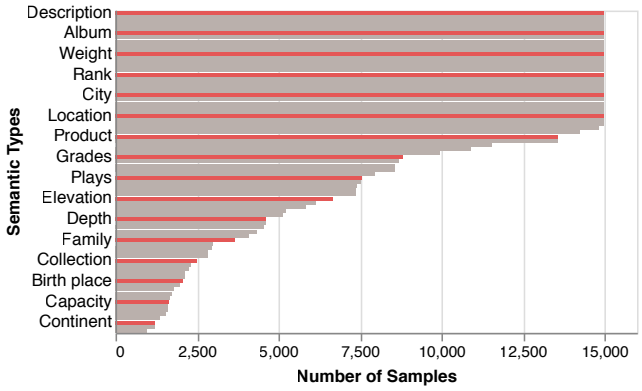

Figure 3: Number of columns per semantic type extracted from VizNet after filtering out the types with more than 15% of the columns not present in the GloVe dictionary, or with less than 1K columns.

Before modeling, we preprocess our features by creating an additional binary feature indicating whether word embeddings were successfully extracted for a given column. Including this feature results in a total of **1,588 features**. Then, we impute missing values across all features with the mean of the respective feature.

4 METHODS

We describe our deep learning model, random forest baseline, two matching-based benchmarks, and crowdsourced consensus benchmark. Then, we explain our training and evaluation procedures.

4.1 Sherlock: A Multi-input Neural Network

Prior machine learning approaches to semantic type detection [19, 33] trained simple models, such as logistic regression, on relatively small feature sets. We consider a significantly larger number of features and samples, which motivates our use of a feedforward neural network. Specifically, given the different number of features and varying noise levels within each feature category, we use a multi-input architecture with hyperparameters shown in Figure 4.

At a high-level, we train subnetworks for each feature category except the statistical features, which consist of only 27 features. These subnetworks "compress" input features to an output of fixed dimension. We chose this dimension to be equal to the number of types in order to evaluate each subnetwork independently. Then, we concatenate the weights of the three output layers with the statistical features to form the input layer of the primary network.

Each network consists of two hidden layers with rectified linear unit (ReLU) activation functions. Experiments with hidden layer sizes between 100 and $1,000$ (i.e., on the order of the input layer dimension) indicate that hidden layer sizes of 300, 200, and 400 for the character-level, word embedding, and paragraph vector subnetworks, respectively, provides the best results. To prevent

overfitting, we included drop out layers and weight decay terms. The final class predictions result from the output of the final softmax layer, corresponding to the network's confidence about a sample belonging to each class, the predicted label then is the class with the highest confidence. The neural network, which we refer to as "Sherlock," is implemented in TensorFlow [1].

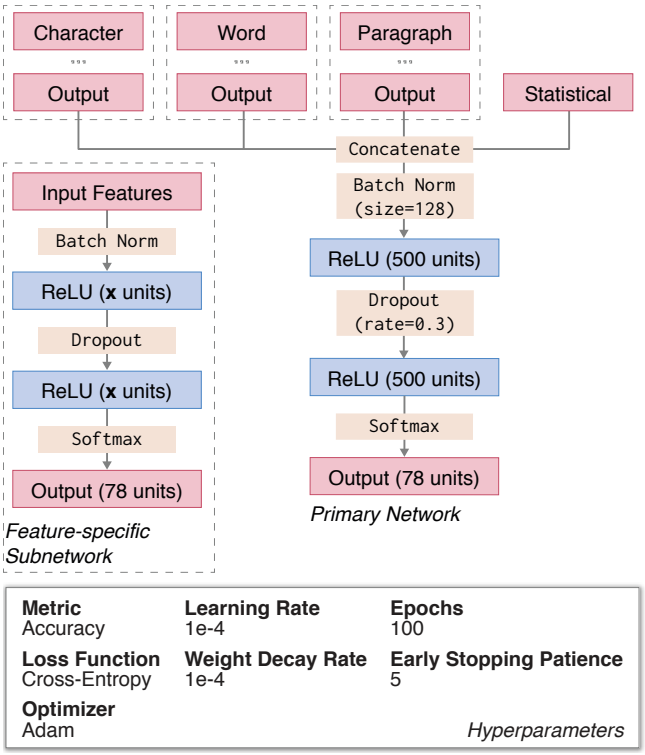

Figure 4: Architecture of the primary network and its feature-specific subnetworks, and the hyperparameters used for training.

4.2 Benchmarks

To measure the relative performance of Sherlock, we compare against four benchmarks.

Machine learning classifiers. The first benchmark is a decision tree, a non-parametric machine learning model with reasonable "out-of-the-box" performance and straightforward interpretation. We use the decision tree to represent the simpler models found in prior research, such as the logistic regression used in Pham et al. [23]. Learning curves indicated that decision tree performance plateaued beyond a depth of 50, which we then used as the maximum depth. We also add a random forest classifier we built from 10 such trees, which often yields significantly better performance. For all remaining parameters, we used the default settings in the scikit-learn package [21].

Dictionary. Dictionaries are commonly used to detect semantic types that contain a finite set of valid values, such as `country`, `day`, and `language`. The first matching-based benchmark is a dictionary that maps column values or headers to semantic types. For each type, we collected the 1,000 most frequently occurring values across all columns, resulting in 78,000 { `value` : `type` } pairs. For example, Figure 5 shows examples of entries mapped to the `grades` type. Given an unseen data column at test time, we compare 1,000 randomly selected column values to each entry of the dictionary, then classify the column as the most frequently matched type.

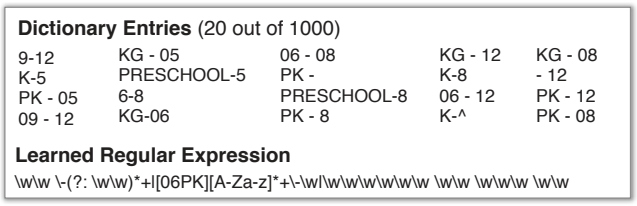

Figure 5: Examples of dictionary entries and a learned regular expression for the `grades` type.

Learned regular expressions. Regular expressions are frequently used to detect semantic types with common character patterns, such as `address`, `birth date`, and `year`. The second matching-based benchmark uses patterns of characters specified by learned regular expressions. We learn regular expressions for each type using the evolutionary procedure of Bartoli et al. [3]. Consistent with the original setup, we randomly sampled 50 "positive values" from each type, and 50 "negative" values from other types. An example of a learned regular expression in Java format for the `grades` type is shown in Figure 5. As with the dictionary benchmark, we match 1,000 randomly selected values against learned regular expressions, then use majority vote to determine the final predicted type.

Crowdsourced annotations. To assess the performance of human annotators at predicting semantic type, we conducted a crowdsourced experiment. The experiment began by defining the concepts of data and semantic type, then screened out participants unable to select a specified semantic type. After the prescreen, participants completed three sets of ten questions separated by two attention checks. Each question presented a list of data values, asked "Which one of the following types best describes these data values?", and required participants to select a single type from a scrolling menu with 78 types. Questions were populated from a pool of 780 samples containing 10 randomly selected values from all 78 types.

We used the Mechanical Turk crowdsourcing platform [17] to recruit 390 participants that were native English speakers and had ≥95% HIT approval rating, ensuring high-quality annotations. Participants completed the experiment in 16 minutes and 22 seconds on average and were compensated 2 USD, a rate slightly exceeding the United States federal minimum wage of 7.25 USD. Detailed worker demographics are described in Appendix A.2. Overall, 390 participants annotated 30 samples each, resulting in a total of 11,700 annotations, or an average of 15 annotations per sample. For each sample, we used the most frequent (i.e., the mode) type from the 15 annotations as the crowdsourced consensus annotation.

4.3 Training and Evaluation

To ensure consistent evaluation across benchmarks, we divided the data into 60/20/20 training/validation/testing splits. To account for

class imbalances, we evaluate model performance using the average F_1-score = $2 \times (\text{precision} \times \text{recall})/(\text{precision} + \text{recall})$, weighted by the number of columns per class in the test set (i.e., the support). To estimate the mean and 95% percentile error of the crowdsourced consensus F_1 score, we conducted 10^5 bootstrap simulations by resampling annotations for each sample with replacement.

Computational effort and space required at prediction time are also important metrics for models incorporated into user-facing systems. We measure the average time in seconds needed to extract features and generate a prediction for a single sample, and report the space required by the models in megabytes.

5 RESULTS

We report the performance of our multi-input neural network and compare against benchmarks. Then, we examine types for which Sherlock demonstrated high and low performance, the contribution of each feature category in isolation, decision tree feature importances, and the effect of rejection threshold on performance.

5.1 Benchmark Results

We compare Sherlock against decision tree, random forest, dictionary-based, learned regular expression, and crowdsourced consensus benchmarks. Table 2 presents the F_1 score weighted by support, runtime in seconds per sample, and size in megabytes of each model.

Table 2: Support-weighted F_1 score, runtime at prediction, and size of Sherlock and four benchmarks.

Method	F_1 Score	Runtime (s)	Size (Mb)
Machine Learning			
Sherlock	0.89	0.42 (±0.01)	6.2
Decision tree	0.76	0.26 (±0.01)	59.1
Random forest	0.84	0.26 (±0.01)	760.4
Matching-based			
Dictionary	0.16	0.01 (±0.03)	0.5
Regular expression	0.04	0.01 (±0.03)	0.01
Crowdsourced Annotations			
Consensus	0.32 (±0.02)	33.74 (±0.86)	–

We first note that the machine learning models significantly outperform the matching-based and crowdsourced consensus benchmarks, in terms of F_1 score. The relatively low performance of crowdsourced consensus is perhaps due to the visual overload of selecting from 78 types, such that performance may increase with a smaller number of candidate types. Handling a large number of candidate classes is a benefit of using an ML-based or matching-based model. Alternatively, crowdsourced workers may have difficulties differentiating between classes that are unfamiliar or contain many numeric values. Lastly, despite our implementing basic training and honeypot questions, crowdsourced workers will likely improve with longer training times and stricter quality control.

Inspection of the matching-based benchmarks suggests that dictionaries and learned regular expressions are prone to "overfitting" on the training set. Feedback from crowdsourced workers suggests that annotating semantic types with a large number of types is a challenging and ambiguous task.

Comparing the machine learning models, Sherlock significantly outperforms the decision tree baseline, while the random forest classifier is competitive. For cases in which interpretability of features and predictions are important considerations, the tree-based benchmarks may be a suitable choice of model.

Despite poor predictive performance, matching-based benchmarks are significantly smaller and faster than both machine learning models. For cases in which absolute runtime and model size are critical, optimizing matching-based models may be a worthwhile approach. This trade-off also suggests a hybrid approach of combining matching-based models for "easy" types with machine learning models for more ambiguous types.

5.2 Performance for Individual Types

Table 3 displays the top and bottom five types, as measured by the F_1 score achieved by Sherlock for that type. High performing types such as `grades` and `industry` frequently contain a finite set of valid values, as shown in Figure 5 for `grades`. Other types such as `birth date` and `ISBN`, often follow consistent character patterns, as shown in Table 1.

Table 3: Top five and bottom five types by F_1 score.

Type	F_1 Score	Precision	Recall	Support
Top 5 Types				
Grades	0.991	0.989	0.994	1765
ISBN	0.986	0.981	0.992	1430
Birth Date	0.970	0.965	0.975	479
Industry	0.968	0.947	0.989	2958
Affiliation	0.961	0.966	0.956	1768
Bottom 5 Types				
Brand	0.685	0.760	0.623	574
Person	0.630	0.654	0.608	579
Director	0.537	0.700	0.436	225
Sales	0.514	0.568	0.469	322
Ranking	0.468	0.612	0.349	439

Table 4: Examples of low precision and low recall types.

Examples	True type	Predicted type
Low Precision		
81, 13, 3, 1	Rank	Sales
316, 481, 426, 1, 223	Plays	Sales
$, $$, $$$, $$$$, $$$$$	Symbol	Sales
Low Recall		
#1, #2, #3, #4, #5, #6	Ranking	Rank
3, 6, 21, 34, 29, 36, 54	Ranking	Plays
1st, 2nd, 3rd, 4th, 5th	Ranking	Position

To understand types for which Sherlock performs poorly, we include incorrectly predicted examples for the lowest precision

type (sales) and the lowest recall type (ranking) in Table 4. From the three examples incorrectly predicted as sales, we observe that purely numerical values or values appearing in multiple classes (e.g., currency symbols) present a challenge to type detection systems. From the three examples of incorrectly predicted ranking columns, we again note the ambiguity of numerical values.

5.3 Contribution by Feature Category

We trained feature-specific subnetworks in isolation and report the F_1 scores in Table 5. Word embedding, character distribution, and paragraph vector feature sets demonstrate roughly equal performance to each other, and significantly above that of the global statistics features, though this may be due to fewer features. Each feature set in isolation performs significantly worse than the full model, supporting our combining of each feature set.

Table 5: Performance contribution of isolated feature sets.

Feature set	Num. Features	F_1 Score
Word embeddings	201	0.79
Character distributions	960	0.78
Paragraph vectors	400	0.73
Global statistics	27	0.25

5.4 Feature Importances

We measure feature importance by the total reduction of the Gini impurity criterion brought by that feature to the decision tree model. The top 10 most important features from the global statistics and character-level distributions sets are shown in Table 6. While word embedding and paragraph vector features are important, they are difficult to interpret and are therefore omitted.

Inspecting Table 6a, we find that the "number of values" in a column is the most important feature. Certain classes like name and requirements tended to contain fewer values, while others like year and family contained significantly more values. The second most important feature is the "maximum value length" in characters, which may differentiate classes with long values, such as address and description, from classes with short values, such as gender and year.

The top character-level distribution features in Table 6b suggest the importance of specific characters for differentiating between types. The third most important feature, the "minimum number of '-' characters", likely helps determine datetime-related types. The fifth most important feature, "whether all values have a ',' character" may also distinguish datetime-related or name-related types. Further study of feature importances for semantic type detection is a promising direction for future research.

5.5 Rejection Curves

Given unseen data values, Sherlock assesses the probability of those values belonging to each type, then predicts the type with the highest probability. Interpreting probabilities as a measure of confidence, we may want to only label samples with high confidence of belonging to a type. To understand the effect of confidence threshold on predictive performance, we present the error-rejection curves of Sherlock and the decision tree model in Figure 6.

By introducing a rejection threshold of 10% of the samples, Sherlock reaches an F_1 score of ~0.95. This significant increase in predictive performance suggests a hybrid approach in which low confidence samples are manually annotated. Note that the higher rejection threshold, the lower the error we make in predicting labels, at the cost of needing more expert capacity.

Table 6: Top-10 features for the decision tree model. "Score" denotes normalized gini impurity.

(a) Top-10 global statistics features (out of 27).

Rank	Feature Name	Score
1	Number of Values	1.00
2	Maximum Value Length	0.79
3	Mean # Alphabetic Characters in Cells	0.43
4	Fraction of Cells with Numeric Characters	0.38
5	Column Entropy	0.35
6	Fraction of Cells with Alphabetical Characters	0.33
7	Number of None Values	0.33
8	Mean Length of Values	0.28
9	Proportion of Unique Values	0.22
10	Mean # of Numeric Characters in Cells	0.16

(b) Top-10 character-level distribution features (out of 960).

Rank	Feature Name	Score
1	Sum of 'D' across values	1.00
2	Mean number of 'M'	0.77
3	Minimum number of '-'	0.69
4	Skewness of ','	0.59
5	Whether all values have a ','	0.47
6	Maximum number of 'g'	0.45
7	Skewness of ']'	0.45
8	Mean number of ','	0.40
9	Mean number of 'z'	0.37
10	Sum of 'n'	0.36

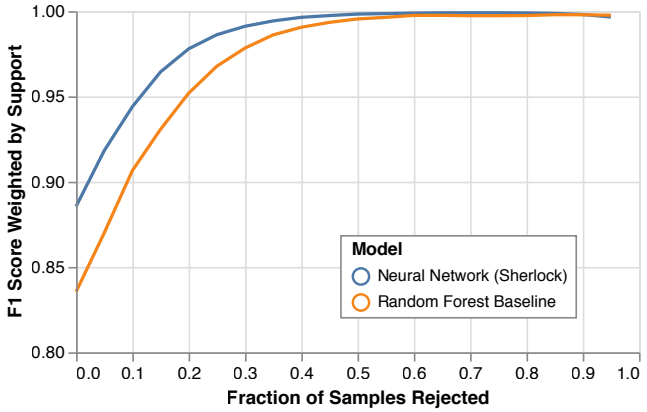

Figure 6: Rejection curves showing performance while rejecting all but the top x% highest confidence samples.

6 DISCUSSION

We began by considering a set of semantic types described by prior work that identifies correspondences between DBPedia [2] and WebTables [5]. Then, we constructed a dataset consisting of matches between those types with columns in the VizNet [14] corpus. Inspection of these columns suggests that such an approach yields training samples with few false positives. After extracting four categories of features describing the values of each column, we formulate type detection as a multiclass classification task.

A multi-input neural network demonstrates high predictive performance at the classification task compared to machine learning, matching-based, and crowdsourced benchmarks. We note that using real-world data provides the examples needed to train models that detect many types, at scale. We also observe that the test examples frequently include dirty (e.g., missing or malformed) values, which suggests that real-world data also affords a degree of robustness. Measuring and operationalizing these two benefits, especially with out-of-distribution examples, is a promising direction of research.

Developers have multiple avenues to incorporating ML-based semantic type detection approaches into systems. To support the use of Sherlock "out-of-the-box," we distribute Sherlock as a Python library[3] that can be easily installed and incorporated into existing codebases. For developers interested in a different set of semantic types, we open source our training and analysis scripts.[2] The repository also supports developers wishing to retrain Sherlock using data from their specific data ecologies, such as enterprise or research settings with domain-specific data.

To close, we identify four promising avenues for future research: (1) enhancing the quantity and quality of the training data, (2) increasing the number of considered types, (3) enriching the set of features extracted from each column, and (4) developing shared benchmarks.

Enhancing data quantity and quality. Machine learning model performance is limited by the number of training examples. Sherlock is no exception. Though the VizNet corpus aggregates datasets from four sources, there is an opportunity to incorporate training examples from additional sources, such as Kaggle,[2] datasets included alongside the R statistical environment,[3] and the ClueWeb web crawl of Excel spreadsheets.[4] We expect increases in training data diversity to improve the robustness and generalizability of Sherlock.

Model predictions quality is further determined by the correspondence between training data and unseen testing data, such as datasets uploaded by analysts to a system. Our method of matching semantic types with columns from real-world data repositories affords both the harvesting of training samples at scale and the ability to use aspects of dirty data, such as the number of missing values, as features. While we verified the quality of training data through manual inspection, there is an opportunity to label data quality at scale by combining crowdsourcing with active learning. By assessing the quality of each training dataset, such an approach would support training semantic type detection models with completely "clean" data at scale.

[2] https://www.kaggle.com/datasets

[3] https://github.com/vincentarelbundock/Rdatasets

[4] http://lemurproject.org/clueweb09.php

Increasing number of semantic types. To ground our approach in prior work, this paper considered 78 semantic types described by the T2Dv2 Gold Standard. While 78 semantic types is a substantial increase over what is supported in existing systems, it is a small subset of entities from existing knowledge bases: the DBPedia ontology [2] covers 685 classes, WordNet [32] contains 175K synonym sets, and Knowledge Graph[5] contains millions of entities. The entities within these knowledge bases, and hierarchical relationships between entities, provide an abundance of semantic types.

In lieu of a relevant ontology, researchers can count frequency of column headers in available data to determine which semantic types to consider. Such a data-driven approach would ensure the maximum number of training samples for each semantic type. Additionally, these surfaced semantic types are potentially more specific to usecase and data ecology, such as data scientists integrating enterprise databases within a company.

Enriching feature extraction. We incorporate four categories of features that describe different aspects of column values. A promising approach is to include features that describe relationships between columns (e.g., correlation, number of overlapping values, and name similarity), aspects of the entire dataset (e.g., number of columns), and source context (e.g., webpage title for scraped tables). Additionally, while we used features to aid interpretation of results, neural networks using raw data as input are a promising direction of research. For example, a character-level recurrent neural network could classify concatenated column values.

Developing shared benchmarks. Despite rich prior research in semantic type detection, we could not find a benchmark with publicly available code that accommodates a larger set of semantic types. We therefore incorporated benchmarks that approximated state-of-the-art data systems, to the best of our knowledge. However, domains such as image classification and language translation have benefited from shared benchmarks and test sets. Towards this end, we hope that open-sourcing the data and code used in this paper can benefit future research.

7 CONCLUSION

Correctly detecting semantic types is critical to many important data science tasks. Machine learning models coupled with large-scale data repositories have demonstrated success across domains, and suggest a promising approach to semantic type detection. Sherlock provides a step forward towards this direction.

REFERENCES

[1] Martín Abadi et al. 2016. TensorFlow: A system for large-scale machine learning. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. 265–283.

[2] Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary Ives. 2007. DBpedia: A nucleus for a web of open data. (2007), 722–735.

[3] Alberto Bartoli, Andrea De Lorenzo, Eric Medvet, and Fabiano Tarlao. 2016. Inference of regular expressions for text extraction from examples. *IEEE Transactions on Knowledge and Data Engineering* 28, 5 (2016), 1217–1230.

[4] Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. 2008. Freebase: A Collaboratively Created Graph Database for Structuring Human Knowledge. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data (SIGMOD '08)*. ACM, New York, NY, USA, 1247–1250.

[5] https://developers.google.com/knowledge-graph

[5] Michael J. Cafarella, Alon Halevy, Daisy Zhe Wang, Eugene Wu, and Yang Zhang. 2008. WebTables: Exploring the Power of Tables on the Web. *Proc. VLDB Endow.* 1, 1 (Aug. 2008), 538–549. https://doi.org/10.14778/1453856.1453916

[6] Raul Castro Fernandez, Ziawasch Abedjan, Famien Koko, Gina Yuan, Samuel Madden, and Michael Stonebraker. 2018. Aurum: A Data Discovery System. 1001–1012.

[7] Raul Castro Fernandez, Essam Mansour, Abdulhakim Qahtan, Ahmed Elmagarmid, Ihab Ilyas, Samuel Madden, Mourad Ouzzani, Michael Stonebraker, and Nan Tang. 2018. Seeping Semantics: Linking Datasets Using Word Embeddings for Data Discovery. https://doi.org/10.1109/ICDE.2018.00093

[8] Andrew M Dai, Christopher Olah, and Quoc V Le. 2015. Document embedding with paragraph vectors. *arXiv preprint arXiv:1507.07998* (2015).

[9] Interactive Data Lab. 2019. Datalib: JavaScript Data Utilities. http://vega.github.io/datalib

[10] Open Knowledge Foundation. 2019. messytables · PyPi. https://pypi.org/project/messytables

[11] Aman Goel, Craig A Knoblock, and Kristina Lerman. 2012. Exploiting structure within data for accurate labeling using conditional random fields. In *Proceedings on the International Conference on Artificial Intelligence (ICAI)*.

[12] Google. 2019. Google Data Studio. https://datastudio.google.com

[13] Christopher Groskopf and contributors. 2016. *csvkit*. https://csvkit.readthedocs.org

[14] Kevin Hu, Neil Gaikwad, Michiel Bakker, Madelon Hulsebos, Emanuel Zgraggen, César Hidalgo, Tim Kraska, Guoliang Li, Arvind Satyanarayan, and Çağatay Demiralp. 2019. VizNet: Towards a large-scale visualization learning and benchmarking repository. In *Proceedings of the 2019 Conference on Human Factors in Computing Systems (CHI)*. ACM.

[15] Sean Kandel, Andreas Paepcke, Joseph Hellerstein, and Jeffrey Heer. 2011. Wrangler: Interactive Visual Specification of Data Transformation Scripts. In *ACM Human Factors in Computing Systems (CHI)*.

[16] Tom Kenter and Maarten De Rijke. 2015. Short text similarity with word embeddings. In *Proceedings of the 24th ACM international on conference on information and knowledge management*. ACM, 1411–1420.

[17] Aniket Kittur, Ed H. Chi, and Bongwon Suh. 2008. Crowdsourcing User Studies with Mechanical Turk. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '08)*. ACM, New York, NY, USA, 453–456.

[18] Quoc Le and Tomas Mikolov. 2014. Distributed representations of sentences and documents. In *International Conference on Machine Learning*. 1188–1196.

[19] Girija Limaye, Sunita Sarawagi, and Soumen Chakrabarti. 2010. Annotating and searching web tables using entities, types and relationships. *Proceedings of the VLDB Endowment* 3, 1-2 (2010), 1338–1347.

[20] Microsoft. 2019. Power BI | Interactive Data Visualization BI. https://powerbi.microsoft.com

[21] Fabian Pedregosa et al. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.

[22] Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*. 1532–1543.

[23] Minh Pham, Suresh Alse, Craig A Knoblock, and Pedro Szekely. 2016. Semantic labeling: a domain-independent approach. In *International Semantic Web Conference*. Springer, 446–462.

[24] Nikhil Waman Puranik. 2012. *A Specialist Approach for Classification of Column Data*. Master's thesis. University of Maryland, Baltimore County.

[25] Erhard Rahm and Philip A. Bernstein. 2001. A Survey of Approaches to Automatic Schema Matching. *The VLDB Journal* 10, 4 (Dec. 2001), 334–350.

[26] Vijayshankar Raman and Joseph M. Hellerstein. 2001. Potter's Wheel: An Interactive Data Cleaning System. In *Proceedings of the 27th International Conference on Very Large Data Bases (VLDB '01)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 381–390.

[27] S Krishnamurthy Ramnandan, Amol Mittal, Craig A Knoblock, and Pedro Szekely. 2015. Assigning semantic labels to data sources. In *European Semantic Web Conference*. Springer, 403–417.

[28] Radim Řehůřek and Petr Sojka. 2010. Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*. ELRA, 45–50.

[29] Dominique Ritze and Christian Bizer. 2017. Matching web tables to DBpedia – a feature utility study. *context* 42, 41 (2017), 19.

[30] Zareen Syed, Tim Finin, Varish Mulwad, Anupam Joshi, et al. 2010. Exploiting a web of semantic data for interpreting tables. In *Proceedings of the Second Web Science Conference*.

[31] Trifacta. 2019. Data Wrangling Tools & Software. https://www.trifacta.com

[32] Princeton University. 2010. About WordNet. https://wordnet.princeton.edu

[33] Petros Venetis, Alon Halevy, Jayant Madhavan, Marius Paşca, Warren Shen, Fei Wu, Gengxin Miao, and Chung Wu. 2011. Recovering semantics of tables on the web. *Proceedings of the VLDB Endowment* 4, 9 (2011), 528–538.

[34] Cong Yan and Yeye He. 2018. Synthesizing type-detection logic for rich semantic data types using open-source code. In *Proceedings of the 2018 International Conference on Management of Data*. ACM, 35–50.

[35] Benjamin Zapilko, Matthäus Zloch, and Johann Schaible. 2012. Utilizing Regular Expressions for Instance-Based Schema Matching. *CEUR Workshop Proceedings* 946.

A APPENDIX

A.1 Supplemental Tables

Table 7: 78 semantic types included in this study.

Semantic Types				
Address	Code	Education	Notes	Requirement
Affiliate	Collection	Elevation	Operator	Result
Affiliation	Command	Family	Order	Sales
Age	Company	File size	Organisation	Service
Album	Component	Format	Origin	Sex
Area	Continent	Gender	Owner	Species
Artist	Country	Genre	Person	State
Birth date	County	Grades	Plays	Status
Birth place	Creator	Industry	Position	Symbol
Brand	Credit	ISBN	Product	Team
Capacity	Currency	Jockey	Publisher	Team name
Category	Day	Language	Range	Type
City	Depth	Location	Rank	Weight
Class	Description	Manufacturer	Ranking	Year
Classification	Director	Name	Region	
Club	Duration	Nationality	Religion	

Table 8: Description of the 27 global statistical features. Asterisks (*) denote features included in Venetis et al. [33].

Feature description
Number of values.
Column entropy.
Fraction of values with unique content.*
Fraction of values with numerical characters.*
Fraction of values with alphabetical characters.
Mean and std. of the number of numerical characters in values.*
Mean and std. of the number of alphabetical characters in values.*
Mean and std. of the number special characters in values.*
Mean and std. of the number of words in values.*
{Percentage, count, only/has-Boolean} of the None values.
{Stats, sum, min, max, median, mode, kurtosis, skewness, any/all-Boolean} of length of values.

A.2 Mechanical Turk Demographics

Of the 390 participants, 57.18% were male and 0.43% female. 1.5% completed some high school without attaining a diploma, while others had associates (10.5%), bachelor's (61.0%), master's (13.1%), or doctorate or professional degree (1.8%) in addition to a high school diploma (12.3%). 26.4% of participants worked with data daily, 33.1% weekly, 17.2% monthly, and 11.0% annually, while 12.3% never work with data. In terms of age: 10.0% of participants were between 18-23, 24-34 (60.3%), 35-40 (13.3%), 41-54 (12.6%), and above 55 (3.8%).