TOWARDS SEMI-AUTOMATIC EMBEDDED DATA TYPE INFERENCE

A PREPRINT

Jonathan Lacanlale
Department of Computer Science
Cal State University, Northridge
Northridge, CA
jonathan.lacanlale.608@my.csun.edu

Arun Kumar
Department of Computer Science
University of California, San Diego
San Diego, CA
arunkk@eng.ucsd.edu

October 9, 2019

ABSTRACT

***K*eywords** First keyword · Second keyword · More

1 INTRODUCTION

2 BACKGROUND

3 OVERVIEW

4 DATASET

This section discusses our efforts in creating the labelled dataset for the task of embedded data type inferencing: the label vocabulary, the data sources, the type of raw features we extract from the columns, and the labelling process.

4.1 Label Vocabulary

In order to correctly classify data in a manner that encapsulates what is represented in our dataset, we've created a 6-class vocabulary that is inclusive to varying data types: *Numbers, Datetime, List, URL, Sentence*, and *Custom Object*. These classes represent the predictions of our ML model. The following is explanations of each class, as well as an example.

Numbers These values are typically numeric values embedded within short strings. They are commonly represented as currency or measurement values. For currency, entries vary between string and symbol usage where a column *Price* may contain *$10* and *10 dollars*.

Datetime Datetime is inclusive to the individual varying formats of dates and timestamps, as well as the combination of the two. The following are examples of the different types and how we have taken note of them.

(a) Standard numerical formats with delimiters in-between values. This is typically seen in *Date* which contains row values similar to *MM/DD/YYYY* in which delimiters vary between {*'-', '/', ':'*}. For timestamps, the common format is *HH:MM* with minor variations in the inclusion of seconds and nanoseconds. This is also noted in similar example columns such as *Datetime* which combines the two having *MM/DD/YYYY HH:MM:SS*, *MM-DD-YYYY HH:MM:SS*, and *MM:DD:YYYY HH:MM:SS*.

(b) String representations of dates where some column *Date* will contain the value *October 30, 2019* or *December 1st*. The only applicable variation in time format has been seen in examples such as *12:30pm* or *6:00 AM*.

List Attributes within this class are consist of iterable strings of items while also containing up to two or more delimiters separating those items. Primary examples follow a format similar to `list` objects in programming languages such as *[a, b, c]*. Variation in delimiters has been seen using one of the following: {*',', ';', '>', ' '*}.

URL Attributes belonging to this class must follow standard format for URL's. This requires that the attribute begin with a `protocol` followed by a `domain name` and end with a `domain`. Any following information such as a file path is be optional. Such formats include *https://www.ucsd.edu/* and *https://admissions.ucsd.edu/first-year/*

Sentence This class contains attributes which vary in length. Primary variation is seen between attributes that consist of entire text excerpts and shorter sentences such as article titles and quotes.

Custom Object This is a catch-all class that captures data that cannot be classified in the earlier listed vocabulary. Large variation is seen within this class, with concrete examples such as coordinate points, email address, and physical addresses. This class also includes attributes that are intended for further computer processing such as `JSON` formatted data. In addition, attribute's with a high `NaN` percentage and mixed data types have also been included. This is data that requires that the data scientist to manually examine the row values for proper labelling and/or processing.

4.2 Data Sources

Our raw data comes from over 360 CSV's collected from sources such as Kaggle[2] and UCI's ML repository[3]. Since this project is primarily focused on embedded feature type inferencing, we use data specifically classified as *'Usable with extraction'* from our earlier work[4]. After manually labelling according to our vocabulary, we obtain 541 attributes. All code and work can be found on Github[1].

4.3 Base Featurization

In order to reduce data preparation time, we've reused collected data from our earlier work with slight modification. Our base featurization files contain each of the following:

(1) Column name. We have collected the column name as it can be indicative of the class it belongs to. Common examples are column names such as *date, time, datetime, timestamp* which a human can easily classify as *Datetime*.

(2) Column values. In the event that the column name is not descriptive enough, a human must then manually examine the column's values and infer what the appropriate label should be. For instance, a column called *Dates* implies multiple date entries. Upon further examination of the column's values, a human will realize that the appropriate label is *List*, as the data is a list of dates. For this, we have collected a maximum 5 random samples from the raw CSV.

(3) Descriptive statistics. Finally, a human may choose to analyze the descriptive statistics of a column. For this, we have collected the mean token value, standard deviation of the token value, NaN percentage, and a boolean for any rows having delimiters. Such descriptive statistics prove useful for inferring what label the attribute belongs to. High mean and high standard deviation of the token value may imply that the column is a *Sentence*, due to the variance in length and high token count, where the opposite would occur for *Datetime* and *URL* which would most likely have significantly lower values.

These features are summarized into a single CSV which we utilize for training our ML models. As shown in Figure 1, we extract the attribute name, descriptive statistics, and 5 randomly selected samples which are then extracted for our ML models. We have 541 total attributes.

4.4 Labelling Process

Labelling was performed using a command line tool written in Python which presents the base featurization values for the user to view. In order to maintain consistency, we have created a rule-book for the data scientist to follow. This rule-book has specific cases for each label and has taken into account all attributes seen when manually labelling. The labelling process took roughly 15-20 hours, including creating the tool to interface with the data, readjusting existing rules, and creating new ones based on new and different data seen.

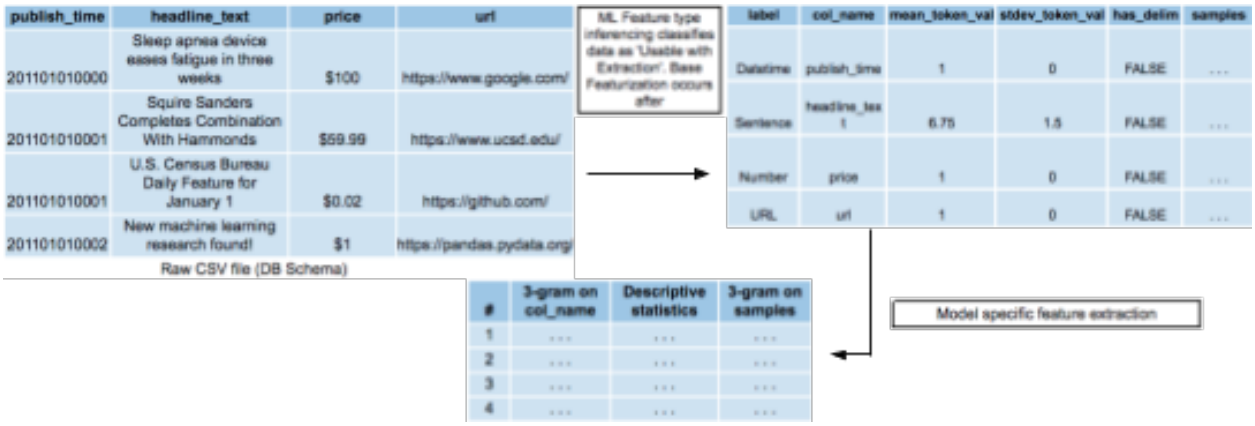

Figure 1: ML embedded feature type inference.

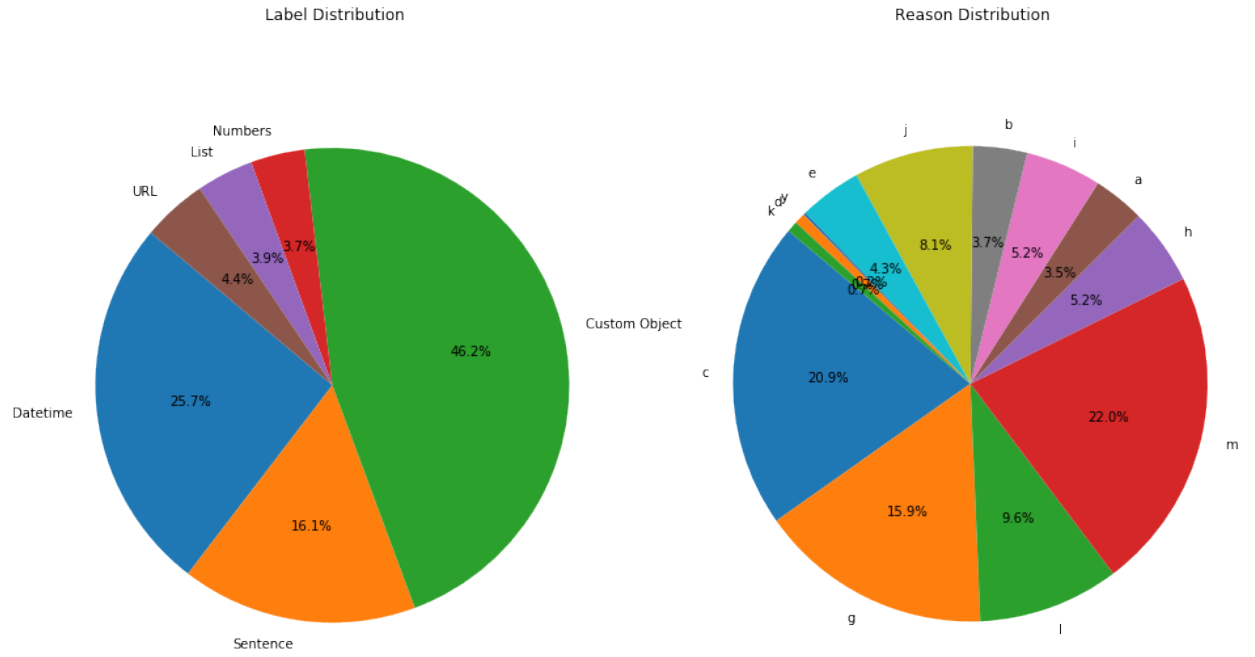

Figure 2: Distribution of 6-class labels on our labeled data.

4.5 Data Statistics

Figure 2 displays our current label distribution, as well as an adjacent chart for reason distribution. Worth noting is that the *Custom Object* class dominates the data set at 46.2% while *List, URL*, and *Numbers* are at the lowest (4.4%, 3.9%, 3.7% respectively). Reason distribution depicts how are data was distributed according to the rulebook we've made for labelling. The rulebook per label is as follows:

Numbers.

a. Number value proceeded and/or preceded by a unit string, as well as possibly in-between e.g. '100 inches', '-0.12amps', '23feet', 'US PER 100 LBS', '$5,000,000.00'

List.

	Datetime	Sentence	Custom Object	Numbers	List	URL
# of NaN's	33618	2861	49638	8360	28525	420
% of NaN's	0.111	0.094	0.412	0.114	0.136	0.019
Mean token val	1	109	4	2	7	1
Stdev token val	0	94	2	0	3	0

Table 1: Mean of different Descriptive Statistics by class in the base featurized data file.

b. Text corpus that contains a delimiter that differs from standard date formats and/or may represent a variable type with the intention for computer processing e.g 'a,b,c', 'word 1; word 2; word 3', 'men|clothing, women|clothing, children|toys etc', ''id': 7, 'name': 'Funny', 'count': 19645'

Datetime.

c. Follows standard numerical formatting seen in applications e.g. '12 .22.16', '1998/01/01', '30-8-2002'

d. Uses a combination of strings and numbers to convey a date e.g. 'December 1', 'October 30, 2019', 'The first of May'

e. Follows standard numerical formatting seen in applications e.g. '12:30', '1:30:20', '15:00'

f. Uses a combination of strings and numbers to convey a time of day e.g. 'Half past 12', '12 PM', '9 o' clock in the morning'

Sentence.

g. Text corpus that does not qualify as a date, timestamp, or URL and conveys significant contextual meaning based on the dataset (title, description, individual information, text excerpt, etc.). e.g. 'Do schools kill creativity?', 'SEE TEXT', 'statistics', 'Hello World'. This text should also not represent a variable type that may be intended for computer processing.

URL.

h. Complete URL that contains a protocol prefix and/or top-level domain suffix with possible additional categorical/product information e.g. 'https://www.ucsd.edu/', 'https://www.researchgate.net/blog'

Custom Object.

i. Multiple numerical values whose context is unrelated to date or time and may require more contextual processing than other numbers e.g. '(-0.022, 2.2]', '(-5.0000, 33.0000)'

j. Phrases containing words that convey specific geographic information e.g. 'address : 9415 Campus Point Dr', 'mailing address: 345 Airport Rd PO Box 1242 Malta', 'MT 59538'

k. Electronic identification data e.g. , 'johndoe@website.com', '/homebrew/recipe/view/246372/freo-pale-ale'

l. Descriptive data that should have been classified by an earlier step.

m. Total of NaN values is too high and/or sample items do not convey contextual background of what the data is

Label distribution balancing is currently planned using synthetic data sets, generating our own similar data based off of existing attributes. In addition to synthetic data sets is the acquisition of new, labelled data that we may use to increase overall class size. Table 1 represents the cumulative average value for our descriptive statistics per class. We observe significant differences between classes and certain statistics. For example, there significantly higher averages for *Sentence*'s *'Mean token val'* and *'Stdev token val'*. *Custom Object* has the highest *'% of NaN's'* value and both *Datetime* and *URL* have a *'Mean token val'* of 1 and *'Stdev token val'* of 0.

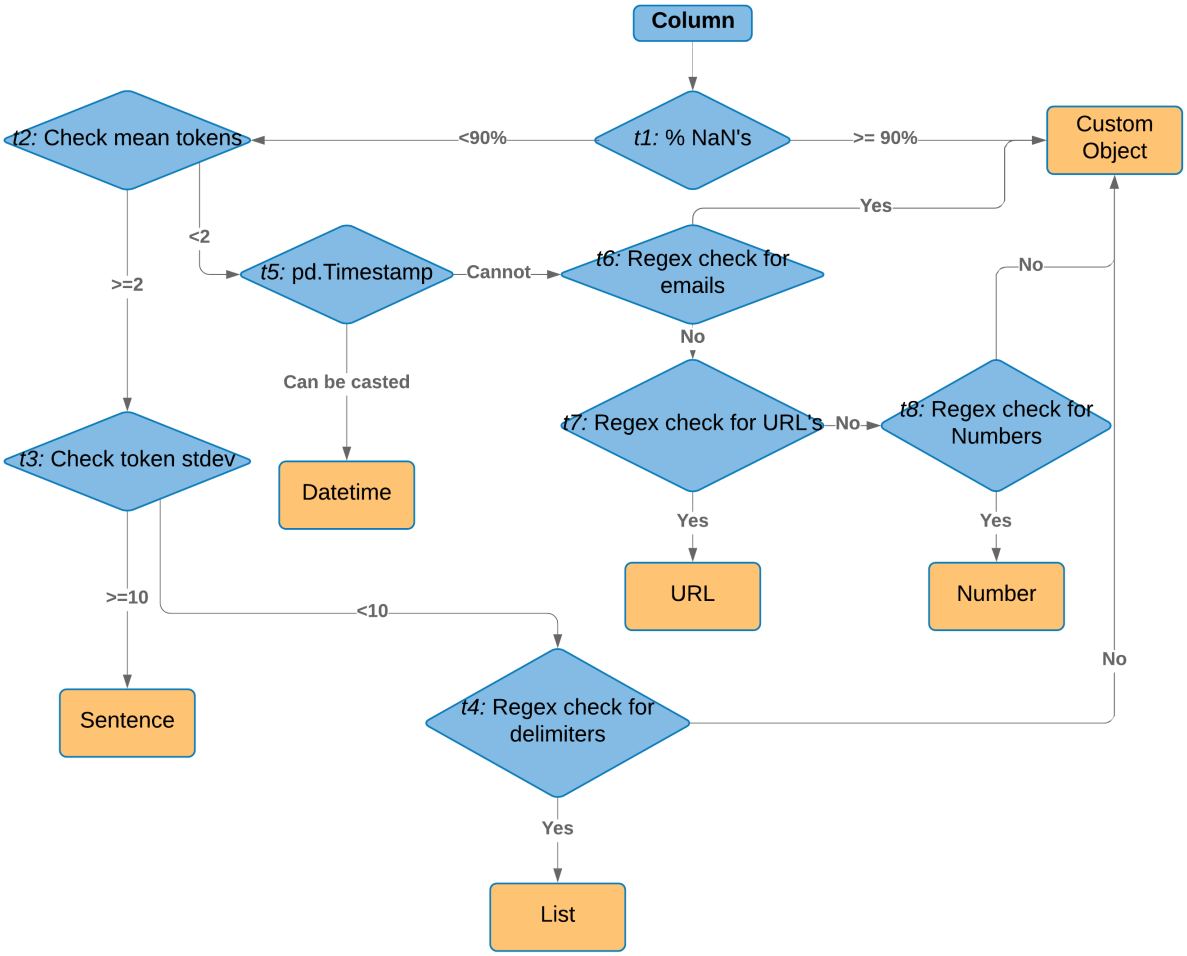

Figure 3: Flowchart of the rule based system. Diamond-shaped nodes are the decision nodes that represents a "check" on the attribute. The final outcome is shown in orange rectangular boxes

5 APPROACHES COMPARED

In the following, we present the features we've extracted from our base featurization file to use to build our ML models. Following is a rule based approach which we've created as a means of baseline comparison. Finally, we discuss our application of classical ML models and present their results.

5.1 Feature Extraction

We observe that attributes with similar names belong to the same class. A common example is in attributes *date, publish_time* and *timestamp* belonging under the label *Datetime*. Similar cases have been seen such as attributes named *url* belonging to the label *URL* as well as *price* and *measurements* belonging to *Numbers*. For this, we've extracted an *n*-gram feature set of the attribute name. In addition to the attribute name, we've also included our descriptive statistics and 2 of the 5 random samples. The random samples were also taken as an *n*-gram feature set.

5.2 Rule-Based Baseline

We have created a rule-approach using Python code. This method utilizes regular expressions (or regex), descriptive statistics, and an external method for checking if the attribute can be casted as a timestamp value. This method was made to mimic how a human may develop an approach using existing tools without the assistance of ML models, as

shown in Figure 3. Each branch leads to a series of "checks" that help determine the appropriate label for the given column (or attribute). We describe all checks in the following descriptions below.

t1: For a column with a NaN % >= 90%, we classify as a *Custom Object*. Columns with a NaN % <90% have their mean token value calculated. If the mean is < 2, it is potentially a *Datetime, Number,* or *URL*. Otherwise, it may be a *Sentence* or *List*. If it is none of the above, it will be classified as a *Custom Object*.

t2: Columns whose mean token value is >=2 have their standard deviation token value calculated. If the standard deviation is >= 10, the column is classified as a *Sentence*. Other wise, it is further checked.

t3: Columns that have a high standard deviation are likely to be *Sentence* attributes. If the standard deviation of the token value is >=10, we classify it as a *Sentence*.

t4: Regex is used to check for delimiters, specifically if their is a word or digit followed by a delimiter (',', '>', '|', ';', ':', '-', '.', '*') and followed by another word or digit. This pattern must be repeated twice or more to be matched by the regex pattern used. Anything that passes will be classified as a *List* and anything that does not will default to *Custom Object*. For instance "1,2,3,4" will pass but "1,2" will not. Actual regex pattern used: `"((\d|\w|')+(,|>|;|:|-|‘.|*)1 \s?( \d|\w|')+)2,"`

t5: The Python library, Pandas[5], offers a built-in object `pd.Timestamp`[6]. If the current column's samples can be casted to a Timestamp object, then it is classified as *Datetime*.

t6: Regex is used to check if the samples are emails. Passing values are classified as *Custom Objects*. The regex used checks that there are two string values with an '@' in-between followed by '.' for the ending of the domain name. For instance, "jonedoe@fakedomain.com" will pass but "invalidemail@fakedomain.invalid" will not. Actual regex pattern used: `"\b[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+.[A-Za-z]2,6 \b"`

t7: Regex is used to check if the sample contains a URL. Passing samples are classified as *URL*'s. Actual regex pattern used: `"(http|ftp|https)://([ \w_-]+(?:(?:.[ \w_-]+)+))([ \w.,@?^=%&:/~+#-]*[ w@?^=%&/~+#-])?"`

t8: Regex is used to check for number values in the string. This regex checks for numbers with a character adjacent to or followed by the value. This regex does match sentences, but this misclassification is avoided since the mean token value is checked earlier on. Actual regex pattern used: `"([ |\w]?( \d|\|.  \d|\d, \d)+ \s?[ |\w]?)"`

5.3 Classical ML Models

We consider classical ML models: logistic regression, RBF-SVM, and Random Forest. The features are our descriptive statistics and *n*-gram featurized sets of the column name and column samples. As a means of saving time, we've reused code from earlier work[4], with slight modification for our features.

6 EMPIRICAL STUDY AND ANALYSIS

We first discuss our methodology, setup, and metrics for evaluating the ML models. We then compare the ML models trained on our data against our rule-based approach. We then present the accuracy results of all models trained on our dataset. Finally, we further analyze the errors of our models.

6.1 Methodology, Set up, and Metrics

Methodolody. We partition our labeled dataset into train and held-out test set with 80:20 ratio. We the perform 5-fold nested cross-validation of the train set, with a random fourth of the examples in a training fold being used for validation during hyper-parameter tuning. For all the classical ML models, we use the Scikit-learn library in Python. We use a standard grid search for hyperparameter tuning, with the grids described in detail below.

Logistic Regression: There is only one regularization parameter to tune: *C*. Larger the value of *C*, the lower the regularization strength, hence increasing the complexity of the model. The grid for *C* is set as $\{10^{-3}, 10^{-2}, 10^{-1}, 1, 10, 100, 10^{3}\}$.

Model		**Stats, Name, Sample 1, Sample 2**	**Stats**	**Name**	**Stats, Name**	**Sample 1**	**Name, Sample 1**	**Stats, Sample 1**	**Stats, Name, Sample 1**
Random Forest	*Train*	0.9381	0.8490	0.9197	0.9479	0.8374	0.9207	0.9311	0.9479
	Validation	0.8078	0.7386	0.7938	0.8217	0.7291	0.7894	0.7871	0.8103
	Test	0.8073	0.7761	0.7743	**0.8275**	0.7817	0.8018	0.778	0.8073
Logistic Regression	*Train*	0.9439	0.6771	0.9201	0.9311	0.8779	0.9381	0.886	0.9415
	Validation	0.8056	0.6713	0.7939	0.7892	0.7431	0.8103	0.7406	0.8033
	Test	0.8128	0.6605	0.7908	0.8	0.8128	**0.8312**	0.7376	0.8220
SVM	*Train*	0.9473	0.7419	0.9132	0.9155	0.8571	0.9346	0.8843	0.9451
	Validation	0.7894	0.6597	0.7893	0.794	0.7315	0.8056	0.7522	0.8023
	Test	**0.8385**	0.7321	0.7908	0.8055	0.8073	0.833	0.8	0.8073

Table 2: 5-fold training, cross-validation, and held-out test accuracy of classical ML models with different feature sets. The bold fonts marks the cases where we noticed highest held-out test accuracy for that model.

Rule-Based Heuristic (A)	Datetime	Sentence	Custom Object	URL	Numbers	List
Datetime	135	0	4	0	0	0
Sentence	0	39	23	0	0	25
Custom Object	9	7	191	0	3	40
URL	0	0	1	23	0	0
Numbers	3	0	4	0	13	0
List	0	0	16	0	0	5

Logistic Regression (B)	Datetime	Sentence	Custom Object	URL	Numbers	List
Datetime	24	0	3	0	0	0
Sentence	0	13	9	0	0	0
Custom Object	1	1	49	0	0	0
URL	0	0	0	2	0	0
Numbers	0	0	1	0	0	0
List	0	1	3	1	0	0

RBF-SVM (C)	Datetime	Sentence	Custom Object	URL	Numbers	List
Datetime	24	0	3	0	0	0
Sentence	0	15	7	0	0	0
Custom Object	0	4	48	0	0	0
URL	0	0	0	2	0	0
Numbers	0	0	1	0	0	0
List	0	2	3	0	0	0

Random Forest (D)	Datetime	Sentence	Custom Object	URL	Numbers	List
Datetime	24	0	2	0	1	0
Sentence	2	13	7	0	0	0
Custom Object	4	2	45	0	1	0
URL	0	0	0	2	0	0
Numbers	0	0	1	0	0	0
List	0	0	5	0	0	0

Table 3: Confusion matrices (A) Rule-based approach, (B) Logistic Regression, (C) RBF-SVM, and (D) Random Forests. Rows represent actual class and columns represent predicted class. Note that the rule-based results are on the entire data set, while the ML models were on the held-out test set.

RBF-SVM: The two hyper-parameters to tune are *C* and γ. The *C* parameter represents the penalty for misclassifying a data point. The higher the *C*, the larger the penalty is for misclassification. The $\gamma > 0$ parameter represents the bandwidth in the Gaussian kernel. The grid is set as follows $C \in \{10^{-1}, 1, 10, 100, 10^3\}$ and $\gamma \in \{10^{-4}, 10^{-3}, 0.01, 0.1, 1, 10\}$.

Random Forest: There are two hyper-parameters to tune: *NumEstimator* and *MaxDepth*. *NumEstimator* is the number of trees in the forest. *MaxDepth* is the maximum depth of the tree. The grid is set as follows: *NumEstimator* $\in \{5, 25, 50, 75, 100\}$ and *MaxDepth* $\in \{5, 10, 25, 50, 100\}$.

Experimental Setup. Due to low dataset size, all models were run locally on a mid-2012 Macbook. Current plans are inplace to move experiments to run on CloudLab[7] using a custom OpenStack profile running Ubuntu 16.10 with 10 Intel Xeon cores and 64GB of RAM.

Metrics. Our key metric is prediction accuracy, defined as the diagonal of the 6 x 6 confusion matrix. We also report the per-class accuracy and their confusion matrices.

6.2 End-to-end Accuracy Results

Rule-based heuristic For our rule-based approach, we achieve a 75% overall accuracy. Due to time constraints, testing using the rule-based approach was on the entire data set unlike the ML models, which were tested using a separate held-out test set. Table 5(A) displays the confusion matrix for our rule-based approach. We observe that the rule-based heuristic is strongest at predicting items such as *Datetime* (89% accuracy) and *URL*'s (95.8% accuracy). This is likely a result of clear rules that prevent confusion and effective regular expressions. Referring back to Table 1, *Datetime* and *URL* both had a cumulative mean token value of 1, having them more likely to be grouped together in the same branch, as seen in Figure 3. For *Datetime*, a method of casting the data ensures that any appropriate attributes will be correctly predicted. As for *URL*'s, any attributes that meet the appropriate format should avoid any earlier regular expressions that it does not meet (specifically **t6** and **t7**). For weaknesses, *List* attributes seem to be the most difficult (5.8%), as there is high variability between attributes, with their only significant distinction being `has_delimiters`. Features such as length and delimiter count vary between attributes. *Custom Object*, *Numbers*, and *Sentence* attributes are also within the lower accuracy percentages (64.1%, 65%, 41.5%, respectively). It is evident that the rules we have made do not encapsulate every existing case, however it would be excruciating and inefficient to exhaustively write a near-"perfect" rule-based approach.

Classical ML Models Table 2 displays our 5-fold accuracy scores utilizing different combinations of features. For all models, descriptive statistics alone is not a sufficient feature, as held-out test scores are at the lowest (Random Forest:

77.6%, Logistic Regression: 66.1%, RBF-SVM: 73.2%). We see similar trends with minor increases for utilizing the column name and a single random sample.

As we increase total features used, we begin to see an increase in accuracy, notably with Random Forests and Logistic Regression at 83%, using `Stats, Name` and `Name, Sample 1` respectively. RBF-SVM achieves its highest accuracy when using the entire feature set, being `Stats, Name, Sample 1, Sample 2` at 83.9%.

We observe that the models are relatively the same in best accuracy scores, however this counters our expectation. For instance, intuition lead us to believe that Random Forests would perform the best while Logistic regression would perform (relatively) worse due to model complexity. However, as shown, this is not the case with models performing within the 83% range. We believe that this is a result of overall low data set size, as well as imbalanced class distribution, however we do expect to see significant changes as we increase the data overall.

Tables 5(B,C,D) show the common trend of our ML models failing to correctly predict *Numbers* and *List* attributes. Again, we believe that this is likely a result of imbalanced class distribution. This is further shown by the held-out test set containing only 5 *List* items and a single *Numbers* attribute. We observe that amongst all models, *URL* attributes achieve a 100% correct prediction rate, although this may change based on increased distribution. Reasoning may be behind the unique and uniform structure of *URL* attributes, however further examination is required. As for the remaining classes, we observe minimal variation between correct and incorrect predictions. The inclusion of additional features and increased data set size, may alter these results, however as seen in accuracy, classical ML models perform relatively similar to one another.

6.3 Analysis of Errors

Amongst all models, incorrect predictions occur the most in attributes that:

- Have no uniform format. For instance, *Sentence* attributes can vary in length between short, concise strings to multiple sentences extracted as text excerpts. Similar instances can be seen in *Custom Object* attributes where an attribute can vary between an email address, physical address, location point, or simply a mixed-data type column and still be classified as a *Custom Object*.
- Have a relatively low label distribution. As mentioned previously, *Numbers*, *URL*, and *List* have the lowest distribution, although *URL* had a 100% correct prediction rate. This is likely due to the uniform structure, though as mentioned, we must further examine such examples. *Numbers* and *List* however are not uniform betwen multiple attribute. For instance, a *List* attribute can range between holding a minimum of 2 items up to any finite number, with no standard delimiter or beginning or starting character (specifically when the *List* starts and ends with some distinct character e.g. `'[...]', '...'`).

7 DISCUSSION

7.1 Results

Our current classical ML models achieve near 83% accuracy. This is an 8% lift accuracy in comparison to our rule-based heuristic which utilizes existing methods. As we improve our current work, the potential to utilize machine learning for embedded type inferencing becomes increasingly tangible. In addition to this is presentation that existing methods are not sufficient for such a task.

7.2 Future Direction

Data set As can be recalled from Figure 2, there is an imbalance within the current data set. Our first solution would be generating a synthetic data set in order to improve label distribution and increase overall data set size. This should prove to be relatively straight-forward for the lacking categories (*URL, List, Numbers*) as their attributes are fairly distinct from other data with specific formats. A secondary approach would be retrieving additional data sets from domains specific to our current needs. For instance, *Numbers* is specifically composed of numeric values embedded within `strings`, used to convey some form of currency or metric system. This could lead to the immediate search for data sets inclusive of pricing, measurements, and other appropriate metrics. The same would apply to *URL* and *List* attributes, increasing label distribution, while adding to other categories as well. This instance can be seen data sets containing *URL* and *Numbers* attributes, while also containing *Datetime* and *Custom object* columns.

Model	Train	Validation	Test
Logistic Regression	0.937510	0.805507	0.814679
Random Forest	0.969322	0.858888	0.849541
SVM	0.946764	0.794012	0.840367

Featurization Our current feature set includes the *n*-gram set of the column name and random samples, as well as our descriptive statistics, however there may be room for additional features. Such additions may include delimiter count, although this would require explicitly stating which delimiters should be accounted for. Further examination is required for potential features, as well as an additional ablation study to gauge model improvement and the usefulness of the feature.

Neural models, time measurements, and setup In our work, we present the results of a rule-based heuristic and classical ML models. Due to time constraints, we have yet to test the effectiveness in neural models. We hypothesize potential accuracy increases, however this work is still in progress. With the addition of neural models would be the usage of computation time and alteration of setup. Mentioned in our methodology, our work was run locally on a mid-2012 model MacBook pro. However, with the change in setup to a CloudLab's systems, we can expect to see better, more consistent training times, with additional timing of prediction results for our labels. Additionally, we can conduct a larger scale ablation study, making note of the best combination of hyper-parameters per feature set.

Existing services Although we have shown our work compared to a rule-based heuristic which utilizes existing methods, such as a Python library or regular expressions, we have yet to test our models against those of existing AutoML tools. There are several comparisons to make such as computation time, accuracy, and methodology, though there is no guarantee that such services will provide this information.

8 Current work

8.1 October 7, 2019

Further error analysis Figure 4 displays detailed information on the models' incorrect predictions. A common trend seen across all models are *List*'s, *Datetime*'s, and *Sentence*'s being incorrectly predicted as *Custom Object*'s. The majority of incorrect predictions comes from *Sentence* items. Further examination of *Sentence* examples shows that the models fail to classify real sentences. This brings us to using new features in an effort to improve our models.

New feature testing Resuming work on the project, the current progress of our ML models consisted of using few features combined with the sample data. Currently, we've created larger amount of features to be used. This includes `mean_stopword_total`, `mean_whitespace_count`, `mean_char_count`, `mean_delim_count`, `stdev_stopword_total`, `stdev_whitespace_count`, `stdev_char_count`, `stdev_delim_count`, `has_url`, `has_date`. Many of these features are taken from the rule-based approach. The current ablation tests combine the features between the *descriptive statistics*, *samples*, and *attribute name*. Table 4 shows current results, however further testing is being performed for the best combination of descriptive statistics.

K-Nearest Neighbors K-Nearest Neighbors was also used after the new features were added. At one neighbor using Euclidean distance, we achieve 88% accuracy and 79% using Levenstein. Additional neighbors brings accuracies between 71% and 79%. Using both both Euclidean and Levenstein distances has accuracies between 77% and 87% based on total neighbors, with 3 neighbors having the best results.

8.2 October 14, 2019

8.3 October 21, 2019

8.4 October 28, 2019

References

[1] Task 3, public repository `https://github.com/lacanlale/SORTINGHAT_T3`

[2] Kaggle Datasets. `https://www.kaggle.com/datasets`.

```
==================== [Random Forest] ====================
>>> Total incorrect predictions: 21, 19.27%
>>> Incorrectly predicted as Custom Object: 18, 85.7%
        >>> List predicted as Custom Object: 5, 27.778%
        >>> Sentence predicted as Custom Object: 9, 50%
        >>> Numbers predicted as Custom Object: 1, 5.5556%
        >>> Datetime predicted as Custom Object: 3, 16.667%
>>> Incorrectly predicted as Datetime: 2, 9.52%
        >>> Custom Object predicted as Datetime: 2, 100%
>>> Incorrectly predicted as Sentence: 1, 4.76%
        >>> Custom Object predicted as Sentence: 1, 100%

==================== [Logistic Regression] ====================
>>> Total incorrect predictions: 13, 11.93%
>>> Incorrectly predicted as Custom Object: 11, 84.6%
        >>> List predicted as Custom Object: 3, 27.273%
        >>> Numbers predicted as Custom Object: 1, 9.0909%
        >>> Datetime predicted as Custom Object: 2, 18.182%
        >>> Sentence predicted as Custom Object: 5, 45.455%
>>> Incorrectly predicted as URL: 2, 15.4%
        >>> List predicted as URL: 1, 50%
        >>> Custom Object predicted as URL: 1, 50%

==================== [SVM] ====================
>>> Total incorrect predictions: 44, 40.37%
>>> Incorrectly predicted as Custom Object: 42, 95.5%
        >>> List predicted as Custom Object: 5, 11.905%
        >>> Datetime predicted as Custom Object: 15, 35.714%
        >>> Sentence predicted as Custom Object: 20, 47.619%
        >>> URL predicted as Custom Object: 2, 4.7619%
>>> Incorrectly predicted as Datetime: 2, 4.55%
        >>> Numbers predicted as Datetime: 1, 50%
        >>> Custom Object predicted as Datetime: 1, 50%
```

Figure 4: Incorrect predictions made by models on the held-out test set. The first line titled `Total incorrect predictions:` displays the total amount and percentage of the held out test set that was incorrectly predicted. Following are the incorrect labels predicted as well as their true labels.

[3] University of California, Irvine: ML Repository `https://archive.ics.uci.edu/ml/index.php`.

[4] Vraj Shah, Premanand Kumar, Kevin Yang, Arun Kumar. Towards Semi-Automatic ML Feature Type Inference. `https://adalabucsd.github.io/papers/TR_2019_SortingHat.pdf`.

[5] pandas: powerful Python data analysis toolkit `https://pandas.pydata.org/pandas-docs/stable/`

[6] Pandas datetime object. `https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.Timestamp.html`

[7] R. Ricci, E. Eide, and C. Team. Introducing CloudLab: Scientific Infrastructure for Advancing Cloud Architectures and Applications.