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

Vraj Shah
Department of Computer Science
University of California, San Diego
San Diego, CA
vps002@eng.ucsd.edu

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

November 27, 2019

ABSTRACT

Contextual data, defined as data that offers contextual background specific to the dataset, can be tedious for data scientists and researchers to work with due to its required manual labelling. Such data is often seen in the form of lists, dates, timestamps, and numerical values meaningful by the stated metric. This labelling process typically utilizes a large amount of time and effort, which can be better allocated onto other significant tasks. We propose the usage of machine learning models in order to automate this process. Using publicly available datasets, we present a list of categories with rules directly based off existing data points to be used for classification. Through the Python scikit-learn library we are able to compare the accuracy of models, such as random forests and logistic regression. We were able to achieve mid-to-high accuracy percentages in terms of their success in automating the labelling process. The accuracy and speed of machine learning models shows promise in automating the labelling process, especially with further parameter tuning.

1 INTRODUCTION

As cited in our earlier work[1], we are able to improve the AutoML pipeline by further automating data preparation. By creating a dictionary based on a large database, we can create a means of data preparation automation. Included in the dictionary is the specific label, *'Usable with Extraction.'* This data is contextual data that requires further feature extraction beyond other labels. All scripts, Jupyter-notebooks, and documentation can be found on GitHub[2].

Problem: Contextual data requires further extraction We define contextual data as data that offers contextual background specific to the dataset. This is often found in the form of dates, timestamps, and numerical values embedded within strings. This data often varies in formats, further illustrated by Table 1. Correctly identifying these items along with the labels we've selected requires further processing for ML models.

Challenge: Labelled Dataset Our dataset is built using data earlier labelled as *'Usable with Extraction'*. This sub dataset came from an original dataset of over 9000 examples, however our dataset size is composed of only 541 data points with a large imbalance between label distribution. This imbalance is the cause of low training results and poor classification of other labels.

Column Name	Sample Value	True Label
datetime_example_1	12/25/2019	Datetime
datetime_example_2	December 25, 2019	Datetime
datetime_example_3	9:00 AM December 25	Datetime
list_example_1	a, b, c, d	List
list_example_2	1; 2; 3; 4	List
numbers_example_1	$10	Numbers
numbers_example_2	10 dollars	Numbers

Table 1: Example data that have different formats but are under the same label.

Label Vocabulary and Labelled Dataset For our approach, we have created a 6 class vocabulary that best describes our dataset, with a catch-all type for varying examples. As mentioned earlier, the current challenge regarding our labelled dataset is data. Of the 9000+ dataset classified and used by SortingHat, only 550 were labelled as *'Usable with Extraction.'* In addition to this low dataset size, our labels often had different formats. Figure 2 only illustrates a few of many examples that have been encountered when manually labelling the dataset. Manually labelling the dataset took 15-20 hours. Further details about the dataset can be found in Section 4.

Featurization and ML Models For a data scientist to manually classify raw data, the most likely approach would be through analyzing the column name as well as column samples. Additionally, the data scientist may make note of significant items such as descriptive statistics, ask yes/no questions about the data (i.e. "Is this a date?"), or recognize specific formats that the data may be in. For our models, we've featurized the dataset to best replicate a data scientists' approach. For our models, we have chosen logistic regression, support vector machine with radial basis kernel, and random forest.

Empirical Analysis We have created a rule-based approach as a means of comparison. Included in this approach is the usage of the Python library Pandas[3] which helps identify `Datetime` objects. We compare our models to this rule-based approach and have found that our results have an 11% increase in accuracy. Following is evaluation of our models. As mentioned previously, we have tested logistic regression, RBF-SVM, and Random Forest. In order to ensure that we are utilizing the best feature set, we perform an ablation study. Our results show that Random Forest is the best performing model. This section of the paper is detailed in Section 6.

Outline Section 2 presents background of our approach and the assumptions made. Section 3 presents a summary of our approach. Section 4 presents our manually labeled dataset which includes our methods and reasoning. Section 5 presents a rule-based approach we have created for comparison. This approach also utilizes Pandas. Section 6 presents an in-depth experimental study, analysis of errors, and our comparison to Pandas advertised method for inferring raw datatypes. Section 7 presents a closer analysis at our best performing model. We conclude by discussing our results and future direction in Section 8. Additionally, we have included earlier results at the end of the paper in our appendix in Section 9.

2 BACKGROUND

2.1 ML Terms and Concepts

We briefly explain the ML concepts and intuition behind the models within the following section.

Concepts Since this task is focused on the classification of data, the task is known as an instance of supervised learning. To briefly explain, the task requires a correctly labelled, training dataset in which the models learn the appropriate parameters. There are several models, however we have only chosen three for our project. Prediction accuracy of the model is reported based on the test dataset. To ensure that our results are consistent, we utilize *k*-fold cross-validation which partitions the labelled dataset into *k* equal subsets with *k*-1 subsets for training and validation and the remaining for testing. The results are then averaged as a single estimate.

Classical ML Models Logistic regression is a linear classifier that finds a hyperplane to separate two classes. For multiclassification problems, logistic regression trains a separate model for each class and predicts whether an example belongs to that class or not. SVM with radial basis kernel applies implicit transformations to the features to map them to a higher-dimensional space and uses this to identify examples that help in separating classes. A decision tree classifies

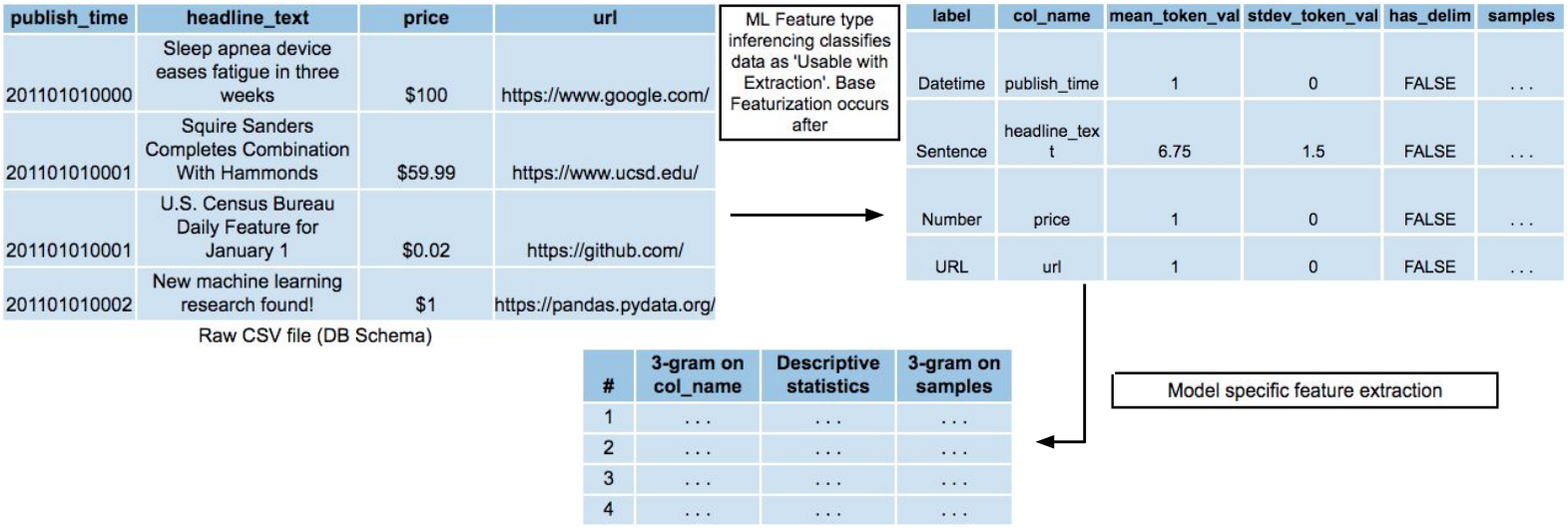

Figure 1: ML embedded feature type inference.

examples by learning a disjunction of conjunctive predicates. Random Forest is an ensemble model that learns multiple decision trees and predicts the mode of the classes given by individual trees.

2.2 Assumptions and Scope

Since our data is dependent on SortingHat's classifications, we assume that any dataset given is a subdataset of those results. This also assumes that all datasets taken from SortingHat's classifications are exclusively those that have been labelled as *'Usable with Extraction.'* Any datasets outside of this label will likely be classified incorrectly as there will be no appropriate labels.

3 OVERVIEW

Figure 1 illustrates our end-to-end review of our problem. Solving this problem required the following manual tasks:

Label Vocabulary We have created a label vocabulary based on the data samples observed. This is further discussed in section 4.5, however to briefly list the labels, we've chosen to include: *Numbers, List, URL, Datetime, Sentence,* and *Custom Object. Custom Object* is our catch-all label that classifies data that significantly varies between other examples.

Labelled Dataset After retrieving *'Usable with Extraction'* data from our earlier work, we manually label the dataset according to our label dictionary. This process is further discussed in Section 4.4.

Features For featurization, we attempt to replicate the steps a data scientist would take in manually labelling the embedded type. This includes utilizing the column name, samples, and descriptive statistics. We extract our features from the raw CSV file in which the data is located and create a feature vector based on the items listed above. For text-based items such as samples and column names, we convert the data into a usable format using n-gram featurization. This step is further explained in Sections 4.3 and 5.1.

ML Models Finally, we use the above steps in this section for training, testing, and validating our ML models. Further detail is discussed in Sections 6 and 7.

4 DATASET

This section discusses our efforts in creating the labelled dataset for the task of embedded data type inferencing. The following subsections include 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[4] and UCI's ML repository[5]. Since this project is primarily focused on embedded feature type inferencing, we use data specifically classified as *'Usable with extraction'* from our earlier work[1]. After manually labelling according to our vocabulary, we obtain 541 attributes. All code and work can be found on Github[2].

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:

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

(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 a large amount including the mean token value, standard deviation of the token value, NaN percentage, and a boolean for any rows having delimiters. In total, our descriptive statistics set 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`. 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 y, where the opposite would occur for *Datetime* and *URL* which would most likely have significantly lower values.

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.

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.

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'

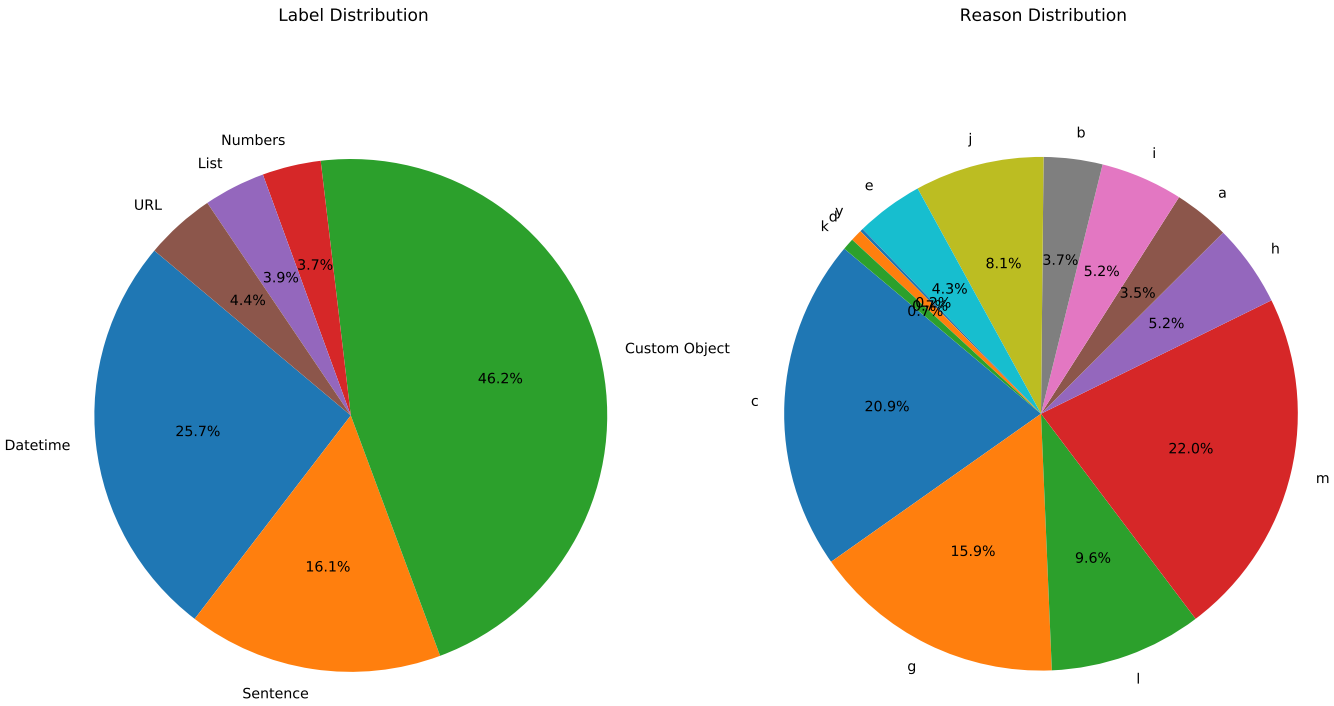

Figure 2: Distribution of 6-class labels on our labeled data.

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 2 represents the cumulative average value for our a few of 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.

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 approach also includes utilizing the Python library, Pandas. It’s use is further explained in item *t5*. 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, offers a built-in object `pd.Timestamp`[7]. If the current column’s samples can be casted to a Timestamp object, then it is classified as *Datetime*.

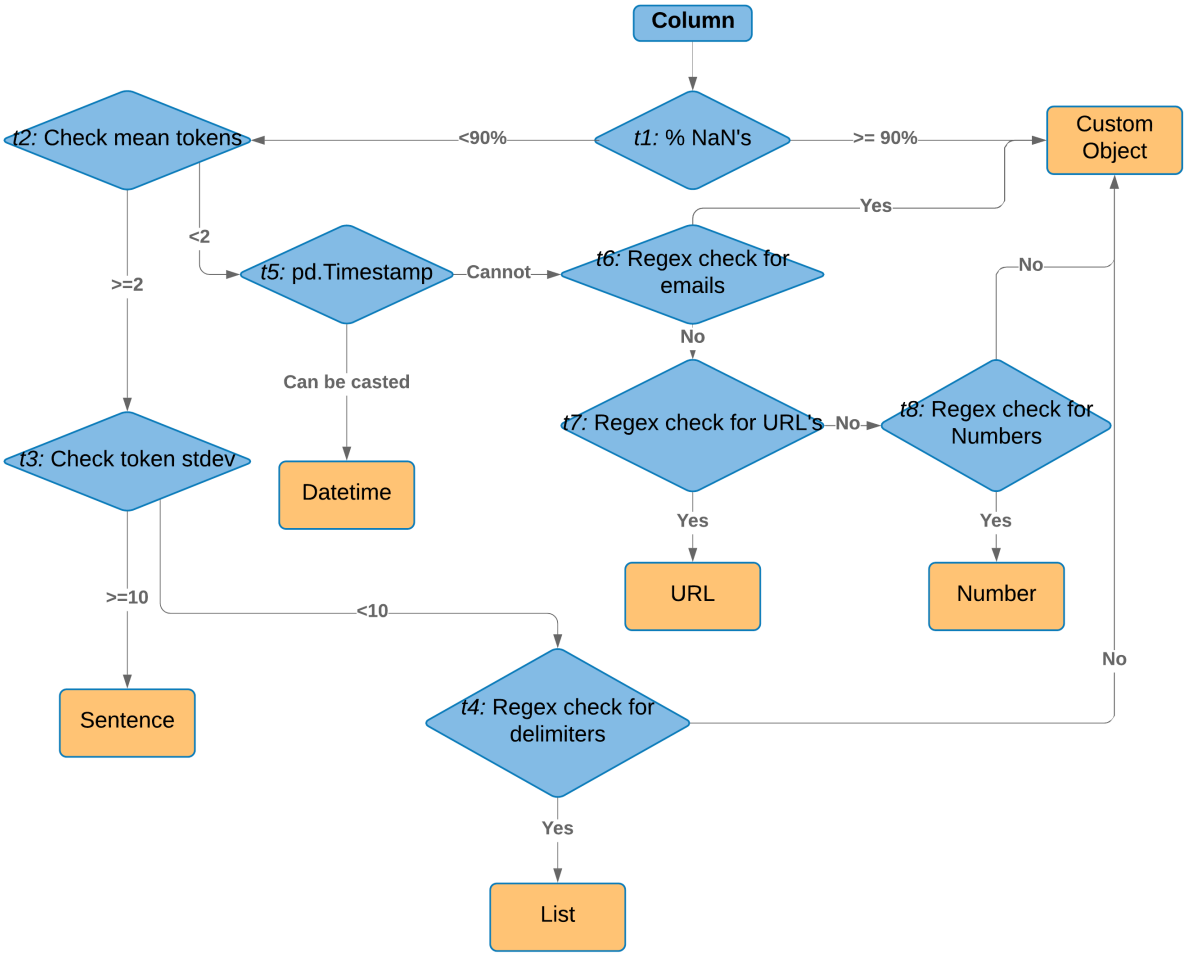

Figure 3: Flowchart of the rule based system. Diamond-shaped nodes are the decision nodes that represents a "check" on the attribute. The final outcome is shown in orange rectangular boxes

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[1], 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 $\{10^{-3}, 10^{-2}, 10^{-1}, 1, 10, 100, 10^{3}\}$.

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[**?**] 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 2, *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 3 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: 77.4%, Logistic Regression: 74.5%, RBF-SVM: 79.1%). 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 RBF-SVM at 84%and Logistic Regression at 83%, using `Stats, Name, Sample 1, Sample 2` and `Stats` respectively. Random Forest achieves the highest overall accuracy when using `Stats, Name, Sample 1` at 85.7%.

Model			Stats, Name, Sample 1, Sample 2	Stats	Name	Stats, Name	Sample 1	Name, Sample 1	Stats, Sample 1	Stats, Name, Sample 1
Logistic Regression	*Train*		0.937510	0.847791	0.920136	0.934600	0.877899	0.938073	0.918985	0.933461
	Validate		0.805507	0.817295	0.793905	0.803234	0.743144	0.810265	0.782491	0.798637
	Test		0.814679	**0.833028**	0.790826	0.829358	0.812844	0.831193	0.744954	0.818349
Random Forest	*Train*		0.961799	0.962384	0.919563	0.969322	0.837411	0.920724	0.942126	0.967005
	Validate		0.847260	0.838145	0.793825	0.858888	0.729083	0.789361	0.824218	0.851938
	Test		0.844037	0.842202	0.774312	0.849541	0.781651	0.801835	0.818349	**0.856881**
SVM	*Train*		0.946764	0.901024	0.913178	0.914915	0.857066	0.934605	0.901635	0.936922
	Validate		0.794012	0.807859	0.789281	0.796177	0.731489	0.805560	0.761481	0.803181
	Test		**0.840367**	0.803670	0.790826	0.831193	0.807339	0.833028	0.811009	0.840367

Table 3: 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.

Rulebased (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	25	0	2	0	1	0
Sentence	0	20	2	0	0	0
Custom Object	4	0	45	0	0	3
URL	0	0	0	2	0	0
Numbers	1	0	0	0	0	0
List	0	0	3	1	0	1

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	3	0	0	0
Sentence	0	19	3	0	0	0
Custom Object	2	2	47	0	0	0
URL	0	0	0	2	0	0
Numbers	1	0	0	0	0	0
List	0	2	3	0	0	0

Table 4: Confusion matrices (A) Rule-based approach, (B) Logistic Regression, (C) RBF-SVM, and (D) RandomForests. Rows represent actual class and columns represent predicted class. Note that the rule-based results are on theentire data set, while the ML models were on the held-out test set.

Tables 3(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 *Comparing to Pandas*

We compared our models to the popular Python library, Pandas *ver. 0.23.3*. The library currently has it's own method for inferring a datatype, `infer_dtype()`[6]. To compare our models, we've taken all *Datetime* items and mapped the other labels as *Not-Datetime*. We then recorded our results for classification and compared it to Panda's built-in method. We anticipated the library to infer it's own *Datetime* type[7], however that was not the result. Shown in Table 6, there are no positive classification results for *Datetime*. The issue encountered is that Pandas fails to correctly infer a datatype from raw data. To receive the correct label, the data must be casted to it's appropriate datatype first. We've also replicated this approach for *Numbers* but also received similar results. In our context, *Number* items were recognized as Pandas' `mixed-integer` and *Datetime* as `string`.

Although Pandas' intended method `infer_dtype()` is unsuccessful at correctly predicting *Datetime* objects, the library does offer a built-in class *pandas.Timestamp*. This class was also used earlier in our rule-based approach because of it's reliability in correctly predicting *Datetime* data. For most entries, the built-in class is capable of correctly identifying between *Datetime* and *Not-Datetime* data, but fails at text representations of dates (i.e *"May 4, 2019"*). Since Pandas is only capable of reliably classifying one of our labels, it is likely not suitable for automating embedded data type inferencing.

	Pandas		***Rulebased***		***Logistic Regression***		***Random Forest***		***RBF-SVM***	
	Datetime	Not-Datetime	Datetime	Not-Datetime	Datetime	Not-Datetime	Datetime	Not-Datetime	Datetime	Not-Datetime
Precision	0	1	1	1	1	0.976	0.96	0.96	1	0.803
Recall	0	0.752	1	1	0.926	1	0.88	0.987	0.259	1
Accuracy	0.752		1		0.982		0.963		0.816	

Table 5: Comparison of methods for Datetime vs Not-Datetime classification. Pandas is utilizing the library method `infer_dtype()`. Each method was used on the held out test set.

Features	*Train*	*Validation*	*Test*
nan %, mean token count, stdev token count, has delims, 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, has email, attr name, sample	**0.9641182876769708**	0.8380379577653034	0.8385321100917432
mean token count, stdev token count, has delims, 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, has email, attr name, sample	0.9513914718941108	0.8403635391606521	0.8275229357798166
stdev token count, has delims, 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, has email, attr name, sample	0.9617927452458742	0.8449879711307137	0.8275229357798164
has delims, 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, has email, attr name, sample	0.9623774817793416	0.8404437316225607	0.8256880733944953
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, has email, attr name, sample	0.9641149367512776	0.840336808340016	0.8330275229357798
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, has email, attr name, sample	0.9612180614894864	0.8449612403100776	0.8366972477064221
mean char count, mean delim count, stdev stopword total, stdev whitespace count, stdev char count, stdev delim count, has url, has date, has email, attr name, sample	0.95717684510346	0.8379577653033948	**0.8550458715596329**
mean delim count, stdev stopword total, stdev whitespace count, stdev char count, stdev delim count, has url, has date, has email, attr name, sample	0.939824076401106	0.8008553862603582	0.8183486238532109
stdev stopword total, stdev whitespace count, stdev char count, stdev delim count, has url, has date, has email, attr name, sample	0.9513747172656448	0.8079657845495858	0.8073394495412846
stdev whitespace count, stdev char count, stdev delim count, has url, has date, has email, attr name, sample	0.949054201223088	0.814915797914996	0.798165137614679
stdev char count, stdev delim count, has url, has date, has email, attr name, sample	0.9496540169221748	0.7963646083934777	0.8055045871559633
stdev delim count, has url, has date, has email, attr name, sample	0.9496456396079416	0.7871157444533547	0.781651376146789
has url, has date, has email, attr name, sample	0.9160777414760828	0.7592889601710773	0.7963302752293578
has date, has email, attr name, sample	0.9218614392225852	0.7754343758353381	0.8110091743119267
has email, attr name, sample	0.8871760073720365	0.754557604918471	0.7871559633027523
attr name, sample	0.969322	**0.858888**	0.8489541
sample	0.837411	0.729083	0.781651

Table 6: Ablation results for Random Forest. In bold is each of the best accuracies for *Train, Validation* and *Test*.

#	*Attribute Name*	*Sample Value*	*mean_word_count*	*mean_whitespace_count*	*True Label*	*RF Prediction*
1	site_info	Sidewalk: Curb side: Cutout	4.79	3.79	List	Custom Object
2	soloists	[{u'soloistName': u'Voigt, Deborah', u'soloist...	5.66	4.66	Custom Object	Sentence
3	duration	30.178	1	0	Numbers	Datetime
4	WHOIS_UPDATED_DATE	16/11/2016	1	0.92	Datetime	Custom Object
5	WHOIS_REGDATE	18/01/1994	1.92	0.92	Datetime	Custom Object
6	Location	-122.316794,37.9240876,0	1	0	Custom Object	Datetime
7	item	Total Value Added	4.75	3.75	Custom Object	List

Table 7: Random Forest misclassification samples. Not all features shown. Results are from the held-out test set.

7 Further Analysis of Random Forest

7.1 *Ablation Results*

To ensure that our features are not creating issues, we have included an ablation study of our entire feature set. Results show that the overall best test accuracy comes from *mean_char_count, mean delim_count, stdev_stopword_total, stdev_whitespace count, stdev_char_count, stdev_delim_count, has_url, has_date, has_email* combined with the attribute name and a single sample at 85.5%. This can be viewed on Table 6.

7.2 *Error Analysis*

Table 7 shows the misclassification samples from the Random Forest model. Not all features are shown and there is currently still work being done to find any significant patterns among certain labels that have been misclassified. There are current issues with finding any significant cause since the held-out test set is much smaller than the overall dataset.

8 DISCUSSION

8.1 Results

Our current classical ML models achieve near 85% accuracy. This is an 11% 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.

8.2 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.

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.

9 RELATED WORK

Existing Embedded Data Type Inferencing As mentioned earlier, the popular library Pandas[3] is unable to correctly infer unlabeled data types. This issue stems from being unable to identify raw datatypes despite having a large dictionary containing items similar to ours including `datetime, time`, and `mixed-integer`. The library requires that the raw data be casted to some existing object prior to using the library's method of inference, a task that this project attempts to semi-automate.

AutoML Platforms Salesforce's TransmogrifAI[8] shows to be an efficient tool for automating the machine learning process, however does not cover data preparation. As advertised, the user should use TransmogrifAI to "Rapidly train good quality machine learnt models with minimal hand tuning"[9]. This emphasizes stages such as feature engineering and model selection, not data preparation or data type inference.

Data Preparation Platforms Trifacta[10] is an example of an existing service that helps enhance productivity during data preparation. The platform provides an interface for preparing data according the the users' desire as well as additional insight about the data given. However, this requires user interaction including manually interacting with the dataset and identifying key features for machine learning projects. This manual work is what our project aims to automate, eliminating the manual work needed and further increasing productivity. Additionally, Trifacta is a paid service with minimal features available at a free level[11], whereas our work is open source.

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 8: 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 9: 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.

References

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

[2] Github: Towards Semi-Automatic Embedded Data Type Inferencing Jonathan Lacanlale `https://github.com/lacanlale/TowardsAutoEmbeddedDataTypeInf`

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

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

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

[6] Efficiently infer the type of a passed val or list-like array of values. `https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.api.types.infer_dtype.html`

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

[8] Automated machine learning for structured data. `https://transmogrif.ai/`

[9] Salesforce TransmogrifAI FAQ `https://docs.transmogrif.ai/en/stable/faq/index.html#what-is-transmogrifai`

[10] An Intelligent Platform that Interoperates with Your Data Investments `https://www.trifacta.com/`

[11] Trifacta Pricing `https://www.trifacta.com/products/pricing/`

10 APPENDIX

10.1 Older Descriptive Statistics Set and Results

Initially, our descriptive statistics set was composed of `num_nans`, `%_nans`, `mean_word_count`, `std_dev_word_count`, `has_delimiters`. Our results and confusion matrices are presented below. (Table 8 and 9, respectively).