Infer ML Feature Type via Weak Supervision

Jingwu Xu
University of California, San Diego
jix010@eng.ucsd.edu

ABSTRACT

Machine learning(ML) enigeers spend the majority of their time dealing with data pre-processing in order to fit the feature data into machine understandable data type, like numerical value or categorical value. However, one major step even before data pre-processing is understanding the feature data type. Traditionally, ML enigeers manually inspect the source file and specify the intended data type of each feature column. However, this is very time-consuming and expensive task in reality, and the problem goes worse when it comes with large data files. Here,Wepresent a label creation framework which outputs the feature type by applying a set of labeling heuristics on feature data statistics. By comparing the performance of downstream model trained using programmatically generated feature type with that using manually labeled feature type, it turns out the label creation framework has signicantly speed up the labeling process (months to days development efforts) with lower cost and the downstream model trained using generated feature type has comparable accuracy than that using manually labled feature type, 87% compared to 93%.

1. INTRODUCTION

The success of machine learning models is closely tied with the quality of training data. ML engineers spend most of their time cleaning the data, preparing the data in order to privode high quality of training dataset. However, an important step between getting the raw data and data preprocessing, while easily get ignored by ML research, is understanding the feature data type. Only after knowing the feature data type can ML engineers perform the data scaling for numerical value, data encoding for categorical value, and data transformation for textual value.

Exisiting ML librarys such as numpy or pandas infer feature data type when loading the data file simply by looking at the values of each feature column. Numerical values are infered as type numerical, True/False values are infered as type categorical, and textual values are infered as strings. However, according to my oberservations, there are many cases it could not cover like numerical values are embeded in textual values, or categorical values are coded into numbers, or represented as strings, some textual data are meaningful, while some are not, etc. As a result, ML engineers need to spend a decent amount of time understanding the raw data and figuring out what is the reasonable data type of each feature column they intend to use in the ML model.

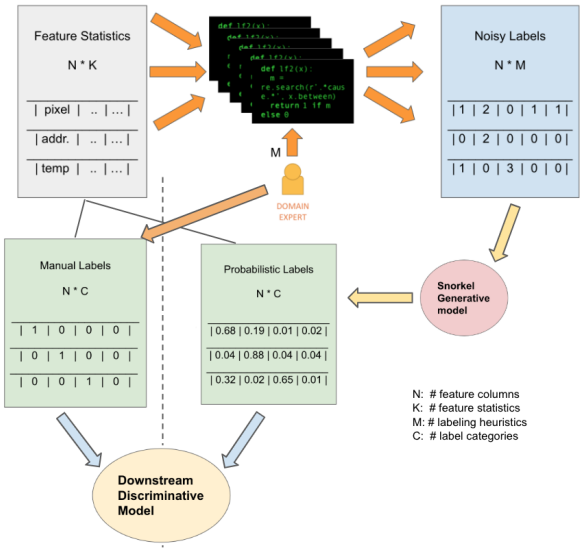

Figure 1: A complete workflow of infering feature data type using two different methods: On the left side, a ML model trained with manual labels. On the right side, a ML model trained with probabilistic labels which reduced from noisy labels. Noisy labels are generated by applying a set of labeling heuristics.

Most commonly, data understanding is done by human inspection into the source file and manually figure out the intended data type of each feature column, here we call this as labeling process. Inferring data type from feature data is not an easy task since companies need to hire domain experts in order to achieve high inference accuracy, which is usually expensive. However, this approach is still only applicable to small data files while impractical on large data files. And thus we need to train a ML model which takes in the feature data as input and predict the data type (label) of that feature column. We can do this by manually labeling hundreds or thousands of feature columns and fit a ML model to do the prediction task. As we talked before, manual labeling process takes considerable amount of time and is expensive. It is more severe when we want more labels in order to achieve higher accuracy for the ML model [9]. This paper presents an alternative labeling process, or say label creation framework, to infer the data type (label) from feature data values via weak supervision, see Figure 1. By applying a set of heuristics on feature data, we can generate noisy labels [2] which will then be fed into Snorkel generative

PRODUCT_ID	PRICE	CATEGORY	COMMENTS
B01EI0Y3IM	24.99	3, 6	I'm never trusting snack bundles like this ever again.
B00J0KQE0G	8.99	1, 3	Crunchy, crispy, tasty cookies! I love them
B0785G3W29	30.53	1, 2, 4	Tastes terrible, but the seller refunded my money right away.

(a)ideal table

PRODUCT_ID	PRICE	CATEGORY	COMMENTS
B01EI0Y3IM	$24.99	Cookies, Snacks	I'm never trusting snack bundles like this ever again.
B00J0KQE0G	CNY 60.47	Chocolate, Cookies	123 haohao chi *&#@!(*#)!
B0785G3W29	€ 27.16	Protein, Chocolate, Peanut	:>>> I like it! 27.16.

(b)real table

Figure 2: Ideal ML data source has values perfectly formated with respect to its data type, however, the real data in life is more messy as value type may vary within single feature column

model to denoise into probabilistic labels. The paper compares two downstream models that trained using manual labels and probabilistic labels. The experiments have shown that the ML models that learned from two different labeling processes achieve comparable good performance while using weak supervision has significantly reduce the development efforts.

2. BACKGROUND

Machine Learning could not success without knowing the data type of each feature. However, inferring the data type for each feature is not an easy task. The common source files we have today are mostly semi-structured, like in CSV, XML, JSON format. Ideally we want each feature column in the source file has the same data type as intended, see Figure 2 (a). For example, all values in feature PRICE are perfect floating numbers that represent the value of price, all values in feature CATEGORY have already encoded as numbers that we can use, all values in comments are meaningful for extraction.

However, in the real world, the data files do not store the values as what we want. The first issue is unreliable source input. Usually these source files are collected from an online-form where users are tending to enter values from their standpoints if without any specific restriction. For example, people enter the incomes with different units according to the currency they use, people enter the date in different format according to their regions. These data files, although nicely formated, is very hard for ML engineers to understand the reasonable date type for feature column. Another problem is table join. Image there are two tables with eactly the same feature column, except for that one table is using True/False and one table is using numbers 0/1. When joining the table, ML engineers need to find out the feature columns with strings True/False are actually the same data type as numbers 0/1, instead of inferring them into categorical and numerical separately. There are also cases like unintended data changes or human interference that could cause the problem.

As can see in Figure 2 (b), we need to think about how many potential feature data types there could be. Remember machine learning models usually understand only two data types, numerical and categorical. However, we definitely want to use price as numerical values instead of textual data, and we want to use the values in CATEGORY as categorical value instead of textual data. Following such ideas, five general data types have been defined as listed in Table 1. *Numerical* data are these with pure numbers, *Categorical* data are these numbers or strings within finite

Table 1: Feature Data Types

Type	Definition
Numerical	numbers that can be directly used as ML numerical feature
Categorical	numbers or strings within finite space that can be directly used or encoded as ML categorical feature
Need Extraction	meaningful textual data that could contains numbers, categories or other information
Unusable	unuseful data which is usually unique id or empty
Context Dependent	meaningful only if we need the context of the feature column

space. *Need Extraction* data are usually textual data that represents some numerical data like '$34.1', or categorical data like '4.5 out of 5', or could be the url, date, etc. *Unusable* data are usually the product, person id which is unique for all feature records. *Context Dependent* data are mainly the uninterpretable features. **In this paper, we do not use *Context Dependent* data type due to time limits.**

There are many challenges when inferring the feature data type. For example, zipcode is usually stored as five-digit number in the source table. Although the values in zipcode features are pure numerical values, zipcode is categorical data and there is point for comparing the value of zipcode digits. The same things happen for movie or film ratings, usually coded as number 0-5 or 0-10. These examples show that there isn't a clear split line between numerical data and categorical data since some categorical data are coded as numbers. Another challenge is distinguishing categorical data versus need extraction data. Categorical data are usually strings that repeat across feature column but limit in a finite space. Need extraction data, however, could also be strings while not necessary in a finite space. For example, Job titles could be very generic which can be viewed as categorical value such as waiter and waitress, while it can also be very specific which loses the categorical characteristics, such as back waiter, busboy, or bartender.

In order to infer the feature data type from feature data, we need to extract some helpful attributes from feature data, which we called feature statistics. Such attributes should form a good summary of all the records in one feature column and ideally direct us to the correct data type as it should be. The attributes we used include feature name, unique value percentage, nan value percentage, max value, min value, mean value, standard deviation, castability, extractability, value length and five sample values, more detail in 4.1.1. To note here, max value, min value, mean value, and standard deviation only apply to pure numerical data, it will be 0 otherwise if data has any characters in it.

3. PROBLEM STATEMENT/SETUP

In traditional supervised learning setting, we are given a training set $D = \{(x_1, y_1), (x_2, y_2)...(x_n, y_n)\}$ of n training examples. For this paper, $x_i \in X$ is the feature statistics and $y_i \in Y$ is the corresponding data type (label). As discussed before, there are a total of five possible data types, namely, numerical, categorical, need extraction, unusable, and context specific, $|Y| = 5$. We want to learn a function f which maps the feature statistics to the feature data type, $f : X \to Y$. To quantify the model performance, a categorical cross entropy loss function l is used. The mapping is

Attribute_name	max_val	mean	min_val	std_dev	sample_1	sample_2	sample_3	sample_4	sample_5	p_dist_val	p_nans	castability	extractability	len_val
report_date	0.00	0.00	0.00	0.00	27/05/09	16/12/08	2012-06-09 00:0…	14/03/16	2004-02-12 00:…	0.01	0.00	0	0	1.40
route_type	0.00	0.00	0.00	0.00	RECYCLING - …	STREET CLEA…	SWEEPER DUM…	YARD TRIMM…	RECYCLING	0.00	0.00	0	0	2.20
Description	0.00	0.00	0.00	0.00	WHITE HANGI…	WHITE METAL …	CREAM CUPID …	KNITTED UNI…	RED WOOLLY …	0.01	0.00	0	0	4.80
Quantity	80,995.00	9.55	-80,995.00	218.08	6	8	2	32	3	0.00	0.00	1	0	1.00

Figure 3: Some Samples of Feature Statistics

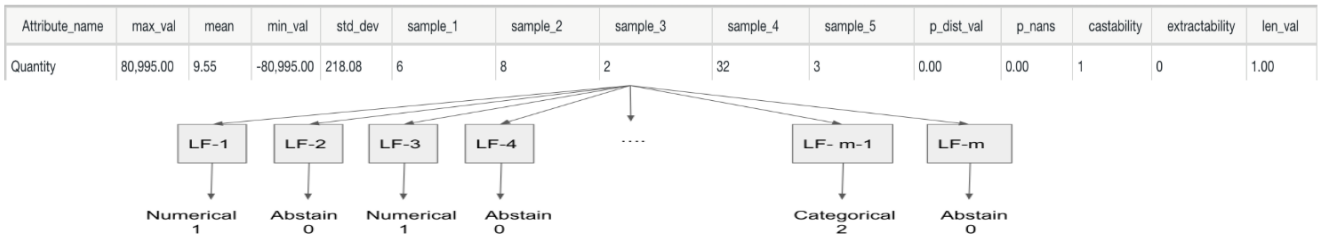

Figure 4: Apply Weak Labeling Heuristics(LF) on Feature Statistics and Ouput Noisy Labels

learned by minimizing the loss function:

$$f^\star = \arg\min_f \sum_{i=1}^{n} l(f(x_i), y_i) \tag{1}$$

where x_i is the feature statistics and y_i is the manually labelled data type.

In weak supervision setting, without knowing the true label (data type) y_i for each feature statistics x_i, we will generate a set of $\tilde{y_{i_j}}(|\tilde{y_i}| = m)$ for each x_i by applying a set of labeling heuristics $h_j(|h| = m)$ on feature statistics, where m is the number of labeling heuristics.

$$\tilde{y_{i_j}} = h_j(x_i) \tag{2}$$

Here $\tilde{y_i}$ is called noisy labels because for feature statistics x_i will have m guessed data types $\tilde{y_i}$ from labeling heuristics. Then we use Snorkel system to denoise the noisy labels into probabilistic labels $\hat{y_{i_k}}(|\hat{y_i}| = C)$ where C is the number of data types defined.

In the real experiment, y_i is usually encoded as one-hot vector. The real differences between y_i and $\hat{y_i}$ is that, y_i has one in the intended data type and 0 elsewhere while $\hat{y_i}$ is a probabilistic distribution among all potential data types. Put in another way, y_i is a vector of deterministic one-hot encoding label vector, $y_i = (y_{i_1}, y_{i_2} \dots y_{i_c})$, where $y_{i_k} = \{0, 1\}$ and c is the number of label classes. While $\hat{y_i}$ is a vector of probabilistic training labels $\hat{y_i} = (\hat{y}_{i_1}, \hat{y}_{i_2} \dots \hat{y}_{i_c})$, where $\hat{y}_{i_k} \in [0, 1]$ and c is the number of label classes. Ideally we want the weak supervision to recover the real data type by minimizing the differences between one hot vector and probability vector:

$$\sum_{k=0}^{c} |\hat{y_{i_k}} - y_{i_k}| \tag{3}$$

After getting probabilistic labels $\hat{y_i}$, the same loss function is used to learn a mapping from x_i to $\hat{y_i}$.

$$\hat{f^\star} = \arg\min_f \sum_{i=1}^{n} l(f(x_i), \hat{y_i}) \tag{4}$$

where x_i is the feature statistics and $\hat{y_i}$ is the probabilistic data types generated using weak supervision.

Now we have two mappings which are able to output the data type given a feature statistics, namely $f^\star$ and $\hat{f^\star}$, where $f^\star$ is trained using manually labeled data types while $\hat{f^\star}$ is trained using denoised probabilistic data types, see Figure 1. The goal of this project aims to compare $f^\star$ and $\hat{f^\star}$ performance on new feature statistics, including prediction accuracy, development efforts, etc.

4. TECHNICAL WORKFLOW

As seen in Figure 1, the workflow of our model includes mainly three parts. The first part is weak supervision. Weak supervision includes generating feature statistics that summarizes important attributes of each feature column, and devising labeling functions that produce cheap but low-quality noisy labels. Second, we apply Snorkel Generative Model to denoise noisy labels into probabilistic labels which give us rough estimation about the underlying distribution of guessed labels. Lastly, we fit a downstream model that we hope to generalize more feature representations from feature statistics and probabilistic labels, and ultimately give better prediction on the real distribution of ground truth labels.

4.1 Weak Supervision

In general, weak supervision aims to get lower-quality labels more efficiently at a higher abstraction level from subject matter experts (SMEs) instead of the groundtruth label. Weak label distributions serve as a way for human supervision to be provided more cheaply and efficiently. SMEs provide higher-level, less precious, partial coverage supervision (e.g. heuristic rules, expected label distributions) on the unlabelled data to guess what could be the intended labels. These labels are usually noisy, lower-quality because overlaps and conflicts problems within the nature of weak supervision. However, these weak labels distribution reflects to some extend where the true label resides.

4.1.1 Feature Statistics

Nomally ML experts apply labeling heuristics directly on the source data, and here is the feature column data. However, each feature column may have infinite data records which makes it impractical to scan all the data samples in one feature column. Here we need to come up with a new compact representation of feature column data that can be used to derive labeling heuristics on it. We define nine feature statistics as well as five sample values from original feature column data, see Figure 3. *Attribute name* is the feature column title, which usually gives the most information on what the feature data type intends to be. Following that are *unique value percentage* and *nan value percentage*, they give us an breif outlook whether the feature data type is categorical or unusable. Then we have common math

statistics for numerical data such as *max value*, *min value*, *mean value*, *standard deviation*. *Length* of sample values is computed by taking the average length of all five sample values, (length of sentence will be the number of words in it). The last two are *castability* and *extractability*, where *castability* means whether any sample value can be cast directly into numerical value, and *extractability* means whether pure numerical value can be extracted from any sample value.

Another benefit by abstracting feature statistics from feature column data is that we can easily apply the feature statistics as input to our downstream generative model. This unified representation of all feature columns allows the downstream model get applied to any new feature column as long as the same feature statistics representation is used.

4.1.2 Labeling Heuristics

In weak supervision setting, we need to come up with labeling heuristics that generate low-quality labels without having access to the ground truth labels [1]. Labeling Heuristics comes from many resources, such as heuristic rules by SMEs, distant supervision [5] by taking advantage of external knowledge base, other classifiers which implement partial of similar tasks, constraints [8] that limit the output space, distributions that guide the label or feature expectations or measurements [3][4], and invariances through data augmentation. Each of the labeling heuristics is usually derived from one resource that particularly handling on specific cases.

In this project setting, we have defined several labeling functions h that aim to guess the data type from feature statistics. Each labeling function we come up with is specified to distinguish one data type from others. Through checking the heuristic rules, the labeling function either has confidence output the guessed label or abstains making an decision.

$$h(x_i) = \begin{cases} c & \text{pass the heuristic rules} \\ 0 & \text{else} \end{cases} \tag{5}$$

where h is the labeling function, x_i is the feature statistics, and c is the specific data type that h aims to guess. 0 means abstain.

Since attribute name is the most intuitive part of the feature statistics, we have used labeling functions that match the attribute name with well-known attribute names for that data type using the idea of distant supervision. For example, attribute name contains 'date' is usually strings that need extraction; attribute name contains 'value' is usually usuable numerical date type; attribute name contains 'class' are usually usuable categorical values. These predefined well-known attribute names usually come from experts who design the table or ML engineers who have better understanding about the attribute naming routine. These labeling functions have rules span across all the data types.

The most commonly used labeling functions are regex matching or pattern extraction, which are designed to discover the underlying pattern for data samples in feature statistics. Such kind of labeling functions is mainly used to distinguish between categorical data and data that need extraction. Since both the two data types usually contain string values, we observe that categorical data are more likley to follow a specific pattern over data that need extraction, which instead usually can be fit into a regex matching. For example, emails are mostly useful information that need to be extracted, by using regex matching, we can easily group emails using regex matching rules the same as we create new email address. Pattern extraction, on the other hand, has a slightly different meaning because it tries to extract the common parts from sample values such as prefix or suffix strings. An example is the parking lot categorical number, which has sample values like 'lot-1', 'lot-7', where we try to extract the pattern 'lot-' and identify the categorical characteristics.

Separating numerical and categorical data can be problematic in such a way that categorical data are encoded as numbers (e.g. movie ratings and zipcode). Heuristic rules such as checking the maximal value against the distinct value are used to detect categorical encoding. The ubiquitous encoding style is usually consecutive numbers that each of them represents a real-world category, see Figure 2 (a). In such case the number of distinct value is the same as the maximal value. Other cases such as zipcode categorical data can be detected by pattern extraction as we talked before.

Unusable data are usually the unique id or the position of a record in the table. Some examples include SSN, employee id, and product id. This data type can be easily guessed by checking the percentage of unique values from feature statistics. Another possibility is the feature column being mostly empty, which means there isn't enough data points in that feature column to make it as usable data type for ML task. Checking the percentage of nan values is a good heuristic rule for this scenario.

Remember, all of these labeling functions as we discussed before are not meant to perfectly guess the true data type of that feature column. They are devised in such a way that can be easily come up with and have confidence in output the labels after checking the heuristic rules. Since most of the labeling functions are designed to cover only a small subset of all possible scenarios, for most of feature statistics the labeling function will abstain making a guess of datat type while outputs only for scenarios that the heuristic rules apply to. They may also have the wrong guess of data type even though the heuristic rules apply for the feature statistics.

4.1.3 Noisy Labels

By applying a set of labeling functions on feature statistics, we can get a set of guessed labels, also called noisy labels [2], see Figure 4. Because these labeling heuristics generate higher-level, less precise, lower-quality labels, there could be overlaps, or conflicts over different labeling functions. As an example in Figure 4, for *Quantity* feature statistics, two labeling functions guess the data type as numerical, one labeling function however guesses data type as categorical and others abstain. The resulting labels vector after applying weak supervision rules hence has overlaping and conflicting labels from different labeling functions.

The distribution of noisy labels should approach to the distribution of true labels. Some of the labeling functions output the same guess of data type for one feature statistics, and the overlaps of guessed labels tell us the true label should more likely to agree with the majority noisy labels. However, someone can always cheat on this idea by adding more labeling functions that output one type of label and let that label has more overlaps than other labels. In order to discover the label distribution from noisy labels and prevent cheating, we need a more mathmatical and systematical way to work out what do noisy labels tell us about the underlying label distribution.

4.2 Snorkel Generative Model

In order to uncover the information that exists in noisy labels, we used Snorkel Generative Model to transform noisy labels into probabilistic labels, see Figure 1. Originally we have the raw feature statistics table with shape $N \times K$, where N is the number of feature columns and K is the

Table 2: Snorkel Model Balances between Labeling Function's Accuracy and Coverage

LFs	Accuracy	Coverage
LF-1	90%	1k labels
LF-2	60%	100k labels

number of statistics we extracted from feature data. After applying a set of labeling functions h on each feature statistics, we got the noisy labels matrix with shape $N \times M$, where M is the number of labeling functions. Since we can not directly use noisy labels as our downstream model's target output, we need to denoise the noisy label into probabilistic labels using Snorkel Generative Model. The probabilistic labels are in shape $N \times C$, where C is the number of label classes, here is the number of data types we defined. Ideally we want the probabilistic labels agree with the ground truth labels by minimizing the difference shown in Equation (3).

There are two main criteria that Snorkel Model examines on the noisy labels that produced by a set of labeling functions, the labeling function's accuracy and coverage [6]. Snorkel model tries to balance between the labeling function's accuracy and coverage to resolve disagreements [7]. For example, see Table 2, there are two labeling functions that we used to guess the data type of the feature statistics, the first labeling function(LF-1) may come from SMEs, which has high accuracy (90%) but only applies to 1k label points. The second labeling function(LF-2) may come from crowdsourcing, which has relatively low accuracy (60%) but generates 100k labels. Snorkel Generative Model first looks into the correlation between all labeling functions' output on one data point and one labeling functions' output on all data points to figure out the accuracy and coverage of that labeling function [1]. Then if there is any disagreement between two labeling functions' output, such as LF-1 and LF-2, because LF-1 has higher accuracy, snorkel finds out its accuracy and coverage and puts more weight on the label probability that agrees with LF-1's output.

However, it is not always true that the label with highest probability denoised from noisy labels is the ground truth label. There are certain cases that designed labeling heuristics fail to cover, while some heuristics rules that are not designed to cover these cases may get satisfied and output its wrong guessing label. As a result, the noisy labels have only few weak-supervised labels that agree with the ground truth label while the majority others are misleading. In such cases, there isn't much snorkel can do. SMEs need to come up with more labeling heuristics to cover all corner cases. The probabilistic labels that snorkel produced under this scenario could be misleading since the argmax of label probability is not necessay consistent with ground truth labels.

4.3 Downstream Discriminative Model

Once we have the probabilistic labels that produced by previous weak supervision steps, we can fit a ML model that maps the feature statistics to desired label. we call this as downstream model since we are using different labels matrixes that generated using two methods, manual labeling versus weak supervision. As described in Section 4.1.1, for all the attributes in feature statistics, we use all but one sample data from five samples. Max, min, mean are nomarlized into 0-1 scale, attribute name and one sample value are encoded using character level embedding. As talked before, since probabilistic labels are not consistent with the underlying true label distribution due to the low-quality weak supervision, we want the downstream model learn some generalized representation from feature statistics rather than noisy labels distribution, and is able to eliminate this inconsistency limited by weak supervision approach.

Table 3: Development cycle for Labeling 8k Samples

	Manual Labeling	Weak Supervision Labeling
Time Sent	1+ Months	3 Days
Workers Involved	Several	1
Other Overhead	Expensive, handle label inconsistency and hand-code labeling rules	Negligible, labeling functions are mostly orthogonal

The downstream discriminative model also allows us to automate the labeling process for feature data type prediction. When we have some new feature column, extracting the same feature statistics representation and feeding it into the downstream model will output the predicted data type of that feature column.

4.3.1 *Convolutional Neural Network Model*

In the Convolutional Neural Network(CNN) model, scaled max, min, mean value, standard deviation, castability, extractability, value length, percentage of unique value and percentage of nan value, all nine attributes are fed into one sequential layer, encoded feature name and one sample value are fed into two seperate Convolutional Neural Network layers, then three layers are concatenated and fed into two densely-connected Neural Network layers with two 20 percent dropout layers, a final softmax layer is used to pull out the predicted label probability. The model is trained using the probalisitic labels with categorical cross-entropy loss function.

4.3.2 *Random Forest Model*

The Random Forest Model only uses the nine attributes in sequential layer as mentioned above. Because Random Forest Model does not support target as probalisitic labels, we take the argmax of probalisitic labels as guessed label for that feature statistics. It will simply learn a mapping from the sequential feature statistics into argmax of probalisitic labels.

5. EXPERIMENT & EVALUATION

The goal of our method is to train a network, f, mapping from inputs (feature statistics) to outputs (feature data type) without needing direct labels of those outputs. We use weak supervision (labeling functions) to generate low-quality noisy labels and use Snorkel Generative Model to denoise into probabilistic labels. Then we fit the Downstream Discriminative Model using inputs and weakly superivsed output labels. We compare this model with that trained with manually generated output labels, regrading to the development efforts, and accuracy. The experiment shows that weak supervision has signicantly speed up the labeling process (months to days development efforts) with lower cost and the downstream model trained using generated feature type has comparable accuracy than that using manually labled feature type, 87% compared to 93%.

5.1 Development cycle

Given the dataset we are using, it includes around 8000 feature columns. In a talk with the Professor, he said it

Table 4: Downstream Model Accuracy% (Evaluated with Ground Truth Labels)

	Ground Truth Labels	Probabilistic Labels	Noisy Mode Labels
Argmax		85.55	
CNN	**93.15**	**87.53**	85.11
Random Forest	91.76	83.29 (argmax)	82.54

Table 5: CNN model trained using probabilistic labels, compares with taking the argmax of probabilistic labels. CNN model generalizes more on predicting categorical and unusable data type.

Argmax Confusion Matrix

	1	2	3	4
1	320	93	12	6
2	43	919	10	27
3	28	4	73	14
4	9	4	0	177

CNN Model Confusion Matrix

	1	2	3	4
1	313	91	11	16
2	23	**953**	4	19
3	25	6	75	13
4	7	2	0	181

CNN Model Trained with Probabilistic Labels

%	Argmax	CNN
Train Accuracy		87.17
Test Accuracy		85.81
Real Accuracy (Ground Truth Labels)	**85.55**	**87.53**

1: Numerical, 2: Categorical, 3: Need Extraction, 4: Unusbale. Rows in confusion matrix are true labels, columns are predicted labels.

took more than one month for several students to manually label those feature column data type. Even after done with crowsourcing labeling, the professor needs to recheck some of the labels to ensure high accuracy. This timespan and efforts could be much higher when dealing with real world extremely large data tables in order to create enough training samples. Other issues, such as labels conflicts due to different labelers, specifying labeling rules to distribute among workers, all create significant work overhead during the development cycle, see Table 3. The labels produced by human inspection usually have high-quality. In this paper, we use these labels as ground truth labels.

However, comparably, writing labeling functions only costs three days for one person to complete. In this project, we start to write these labeling functions without knowing anything beforehand, which means the time we spend also includes the overhead of learning the labeling rules and naming standards. It could be much cheaper for domain experts to write labeling functions and further scale down when distributed to group of people. In addition, there is no overhead for distributing the work of writing labeling functions because most labeling functions are orthogonal, which means no further inconsistency is introduced when aggregating the work from workers. In weak supervision setting, obtaining a set of labeling functions takes up most of the time. After applying a set of labeling function on feature statistics, we get noisy labels. Then we use Snorkel to denoise noisy labels into probabilistics labels. These steps can be done fairly quick, and time spent on these steps is negligible compared to writing labeling functions in the development cycle.

5.2 Downstream Model Accuracy

We fit a downstream model in order to generalize the mapping from feature statistics to feature data type. To conclude above, we have manually labeled data type as ground truth labels, noisy labels by applying labeling functions, and probabilistic labels that denoised using Snorkel model. We would like to compare how the downstream model generalize the feature statistics when learning from different labels. Across our experiments, we provide ground truth labels only for the purpose of evaluation.

Here we create another label set, called Noisy Mode Labels, which is basically taking the mode of noisy labels for each feature statistics (aka. the majority guess among all labeling functions' output for that feature statistics).

$$\bar{y_i} = \arg\max_k count(\hat{y_{i_k}}) \tag{6}$$

The first model we used is CNN, see Section 4.3.1. We first train the model using feature statistics as X and ground truth labels as Y, the model's evaluation accuracy on testing data is 93.15%. Then, we train a same model using feature statistics as X but probabilistic labels as $\hat{Y}$, the model's evaluation accuracy on testing data is 87.53%. Another experiment is added which trains another same model using feature statistics as X but noisy mode labels as $\bar{Y}$, the model's evaluation accuracy on testing data is 85.11%, see Table 4. Two conclusions can be drawn from the results in our experiment setting:

- The downstream model trained via weak supervision (using the probabilistic labels) works comparable good as model that trained via full supervision (using ground truth labels), the accuracy drops slightly from 93% to 87%.
- Applying Snorkel Generative Model on noisy labels and denoising into probabilistic labels allows downstream model to generalize a bit more than simply taking the majority guess of noisy labels, as can see the accuracy improves slightly from 85% to 87%.

The second model we used is Random Forest, see Section 4.3.2. Note that Random Forest model requires deterministic target value, here we simply take the argmax of probabilistic labels. We also compare the performance using three different label sets, namely ground truth labels Y, argmax of probabilistic labels $\hat{Y}$, and noisy mode labels as $\bar{Y}$. Random Forest Model generally works worse than CNN since it uses fewer feature statistics than CNN. The model trained with ground truth labels has accuracy 91.76%, argmax of probabilistic labels is 83.29%, and noisy mode labels is 82.54%, see Table 4. Random Forest model trained using argmax of probabilistic labels degrades more severe than models trained using other label sets, where a 4.2% degradation using argmax of probabilistic labels than 1.4% using ground truth labels and 2.5% using noisy mode labels. By comparing the performance of CNN model and Random Forest model in our experiment setting, we can conclude that:

- Random Forest model generally performs slightly worse than CNN model in terms of prediction accuracy. One potential reason is that we only use nine sequential feature statistics but no textual statistics such as attribute name and sample value.
- Probabilistic labels contains more hidden information

than the argmax of probabilistic labels for the downstream model to generalize and benefit from. Simply taking the argmax of probabilistic labels leads to the degradation of downstream model performance.

Apart from two downstream models as mentioned in Section 4.3, a CNN model and a Random Forest model, we also take the argmax of probabilistic labels and directly compare it with ground truth labels. This comparison is meant to show whether adding a downstream model helps improve the labeling accuracy rather than directly taking the argmax of the probabilistic labels. As shown in Table 4, by taking the argmax of probabilistic labels without any downstream models already has 85.55% accuracy. Adding a CNN model could improve the prediction accuracy while using Random Forest Model in fact decreases the prediction accuracy in our experiment setting. In other words, CNN model could exploit and generalize hidden information from feature statistics and hence improve the prediction accuracy while Random Forest Model can not. It should be noted that predicting labels through taking the argmax of probabilistic labels could not be generalized onto new data samples, since the probabilistic labels is computed from all training samples.

5.3 More about CNN Model

Next we want to explore what the downstream model could generalize from probabilistic labels, compared with not using the downstream model but taking the argmax of probabilistic labels. In reality, because the probabilistic labels are denoised from noisy labels, they may not perfectly reflect the true label's distribution. So we use the downstream model to learn a mapping from feature statistics to probabilistic labels hoping that more generalized features can be uncovered and utilized to improve the prediction accuracy. With that in mind, we conduct an experiment by comparing the accuracy with and without using a downstream model and their confusion matrix to further diagnose what is improved, see Table 5.

As can see in the right table, the downstream CNN model trained using probabilistic labels has training accuracy 87.17% and testing accuracy 85.81%. The model accuracy is measured between the CNN learned probabilities and the donoised probabilities. However, the interesting thing is when we calculate the accuracy between real labels (not probabilities) and labels by taking the argmax of CNN learned probabilities, the accuracy is higher, 87.53% compared to 85.55%, than taking the argmax on denoised probabilisties without downstream model. We conclude that, the CNN model does not fully mimic the label distribution from probabilistic labels, while is able to learn something generic that helps it uncover the real distribution of ground truth labels and has better accuracy on real labels.

In order to diagnose on what aspect does the CNN model outperform than taking the argmax or probabilistic labels, we look into the confusion matrix of both models compared with ground truth labels, see Table 5. By comparing the CNN and Argmax confusion matrix, we can see CNN model does have better performance on predicting *Categorical* data type, with similar prediction accuracy on other data types. The performance improvement is likely due to the information that CNN learned from feature statistics, which helps CNN model correct its predictions from original probabilistic labels. Attributes like *percentage of unique values* from feature statistics may have higher weight on predicting *Categorical* data type regardless of what the noisy probabilistic labels tell. Thus we can conclude, the CNN model is able to uncover and generalize information from input data which enables more accurate prediction on the distribution of ground truth labels.

6. FUTURE WORK

Here I list several problems I came into when doing this project and leave them for future improvement.

Why adding more labeling functions degrades the downstream model performance? We use Snorkel Generative Model to denoise noisy labels into probabilistic labels. However, when I try to add more labeling functions, the downstream model performance degrades. One reason is that most of the new labeling functions I added in order to cover a small subset of corner cases may have huge overlaps with other labeling functions. The overlaps in Snorkel model is important since it is used to infer the labeling function's accuracy when compared with other labeling functions. As a result, we introduce more label interferences than useful labels on noisy data, and Snorkel model could not correctly infer the labeling functions accuracy and coverage.

Should one labeling function classify single or multiple categories? The labeling function in this project is designed in such a way that predicts either one class data type or abstain making predictions, see Equation (5). However, according to the examples from Snorkel people, labeling function usually contains rules that separate multiple labels. It is not clear whether Snorkel does a better job on denoising the noisy labels when each labeling function covers wide range of labels. More work can be done in that direction in order to get more significant improvement on downstream model's accuracy.

Weak supervision without writing labeling functions? When we compare the development cycle for weak supervision and full supervision, we note that writing labeling functions, although cost significantly fewer efforts compared with manual labeling data types, may still cost a considerable amount of efforts when the problem complexity goes up. Snuba provides us a new way of generating labeling functions by manually labeling a small subset of data samples and then automatically generating heuristics [10]. More work can be done in that direction to compare the labeling efforts and the accuracy of downstream model trained with automatically generated heuristics.

7. CONCLUSIONS

A major problem when converting semi-structured datafile into ML data frame is predicting the data type (numerical, categorical, unusable, etc.) of each feature column. However, due to the unreliable input data source, existing methods such as pandas.readcsv or numpy readcsv could not correctly infer them as intended data type. Here I discussed a generative model to infer the feature data type based on the feature statistics through weak supervision. The results have shown that it is feasible, with comparable accuracy, and significantly speedup compared to manual labeling feature data types.

8. ACKNOWLEDGMENTS

Thanks to Professor Arun Kumar, I get tons of help and inslights on this project from Prof. Kumar. Thanks to Vraj Shah for providing downstream model framework. Thanks to Prem for providing the feature statistics generation code.

9. REFERENCES

[1] S. H. Bach, B. D. He, A. Ratner, and C. Ré. Learning the structure of generative models without labeled data. *CoRR*, abs/1703.00854, 2017.

[2] J. Bootkrajang and A. Kabán. Label-noise robust logistic regression and its applications. In P. A. Flach, T. De Bie, and N. Cristianini, editors, *Machine Learning and Knowledge Discovery in Databases*, pages 143–158, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[3] P. Liang, M. I. Jordan, and D. Klein. Learning from measurements in exponential families. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, pages 641–648, New York, NY, USA, 2009. ACM.

[4] G. S. Mann and A. McCallum. Generalized expectation criteria for semi-supervised learning with weakly labeled data. *Journal of Machine Learning Research*, 11:955–984, 2010.

[5] M. Mintz, S. Bills, R. Snow, and D. Jurafsky. Distant supervision for relation extraction without labeled data. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 2 - Volume 2*, ACL '09, pages 1003–1011, Stroudsburg, PA, USA, 2009. Association for Computational Linguistics.

[6] A. Ratner, S. H. Bach, H. R. Ehrenberg, J. A. Fries, S. Wu, and C. Ré. Snorkel: Rapid training data creation with weak supervision. *CoRR*, abs/1711.10160, 2017.

[7] A. J. Ratner, C. M. De Sa, S. Wu, D. Selsam, and C. Ré. Data programming: Creating large training sets, quickly. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 3567–3575. Curran Associates, Inc., 2016.

[8] R. Stewart and S. Ermon. Label-free supervision of neural networks with physics and domain knowledge. *CoRR*, abs/1609.05566, 2016.

[9] S. Takamatsu, I. Sato, and H. Nakagawa. Reducing wrong labels in distant supervision for relation extraction. In *ACL*, 2012.

[10] P. Varma and C. Ré. Snuba: Automating weak supervision to label training data. *Proc. VLDB Endow.*, 12(3):223–236, Nov. 2018.