Recognizing Quantity Names for Tabular Data

Yang Yi Zhiyu Chen Jeff Heflin Brian D. Davison
{yay218 | zhc415 | heflin | davison}@cse.lehigh.edu

Dept. of Computer Science and Engineering, Lehigh University
Bethlehem, PA USA

Abstract

In this era of Big Data, there are many public web repositories for people to access, retrieve, and store data. It is natural for dataset search queries to include quantities along with their units. Describing data in terms of units is an important characteristic when that data is to be used, even though such units are often not present in the data schema. However, quantity names are often provided with or without corresponding units in the column name or in an abbreviated format. Quantity names (e.g., *length, weight* and *time*) can be matched to a set of relevant units. Therefore, there is a significant need to automatically determine the quantity names for column values. We investigate the potential to recognize quantity names to which units belong. We assign each column a class label corresponding to the quantity name and thus configure the problem as a multi-class classification task, and then establish a variety of features based on the column name and column content. Using a random forest, we show that these features are useful for predicting quantity names for columns in tables.

1 Introduction

With the increasing number of datasets available online, people in many roles are capturing, storing, and analyzing datasets. Business analysts may search related data to predict an upcoming crisis; journalists want to find data to report; students use data to learn and analyze, etc. As a result, online data have the potential to receive unprecedented attention by people with various backgrounds.

Searching for appropriate datasets is a crucial step before using them. To generate results, query terms must be matched with some columns or rows in the tables. Quantity names (which in QUDT[1] reference ontologies are called quantity kinds) appear commonly in queries to retrieve numeric data, so inferring quantity names is an important and urgent task to improve the ability to match datasets. Quantity names generally correspond to one or more units, and so recognizing and matching quantity names can provide for a broader search scope than simply units. For example, one can imagine a query that asks for data in *inches*, and the system can recognize that as a kind of *length* and look for datasets that contain columns that belong to quantity name *length*, because values in another unit can always be converted.

When examining a set of datasets from *data.gov*, we find that more than 40% of data columns hold numeric values, and of those, about 30% of those columns appear to be quantities that could have units (i.e., more than 10% of all columns). Our goal, then, is to recognize and recommend quantity names for numeric columns which

In: Joint Proceedings of the First International Workshop on Professional Search (ProfS2018); the Second Workshop on Knowledge Graphs and Semantics for Text Retrieval, Analysis, and Understanding (KG4IR); and the International Workshop on Data Search (DATA:SEARCH18). Co-located with SIGIR 2018, Ann Arbor, Michigan, USA – 12 July 2018, published at http://ceur-ws.org

[1] http://www.qudt.org/

could have units. For example, we tag all columns with possible units including *meter, mile, inch, feet* as *length* quantities, and columns with units like *pound, ton, ounce* as *weight* quantities. In prior work on searching data, Au et al. [1] and Thomas et al. [6] point out that inferring data types is a necessary step in generating query-based summaries. They present a method to recognize *dates, numbers, place names*, and *strings*. Our experiment expands the "numbers" type mentioned in their papers into specific quantities, so that more precise and concrete results can be shown to people who are doing dataset searches. In addition, an extensive approach to extract units was proposed by Sarawagi et al. [5] for quantity queries on web tables. They present rule-based and feature-based unit extractors to detect units that already exist in the column name. However, our proposed method can also work on columns for which quantity names or units are not explicitly provided in the column name. This can greatly enlarge the matching scope of dataset queries.

In our experiments, we recognize terms associated with quantity names, and use five common quantity names (that were found in an early cursory examination of the data) as class labels. The rest of the quantity names, plus columns that do not have units, are all counted as a 6^{th} class.

Tabular data (i.e., a data table) is one of the most commonly used type of data. In this paper, we focus on data represented as a two-dimensional table in CSV format. After filtering non-numeric columns, we build features on the rest of the dataset. Half of the features are based upon the column name, with one of them converted from the rule-based unit extractor developed in Sarawagi et al. [5]. In addition, since different quantities have some specific patterns, and features based on column contents are already used by Chen et al. [2] and received good accuracy, we add other features based on column content (refer to Section 4 for details). For our multi-class classification task, we build a dataset containing features of 4,418 instances. We then apply 10-fold cross validation and classification to train and test our model. The results show that our proposed approach can recognize and recommend quantity names for tabular data.

2 Related Work

There is no prior work describing the problem of dataset quantity name recommendation of which we are aware. However, some efforts have proposed methods to detect data types and existing units. In addition, there is work in the area of processing column names and column contents for queries on web tables.

In information retrieval, there is work on web search for tabular data. To match and extract precise data, Thomas et al. [6] and Au et al. [1] introduce methods to infer data types, specifically *strings, numbers*, *boolean values, dates*, and *place names*, which are stored in various formats in web tables. Valera et al. [7] propose a way to discover statistical types of variables in a dataset using Bayesian methods. Sarawagi et al. [5] point out that unit extraction is a significant step for queries on web tables, and design unit extractors for units in column names by developing a unit catalog tree. These methods only extract units and data types that already exist within column names, and cannot be applied to infer unknown units. Inspired by their work, we build features for classification, with one based on Sarawagi et al.'s unit extractor.

To perform feature-based quantity name inference, it is helpful to utilize other datasets which have similar distributions and context. Two papers [3, 8] discuss models to find related tables by computing schema similarity. These models can detect that columns "Shape_Length (mile)" and "Shape_Len" have high similarity. In addition, Ratinov et al. [4] present a way to expand the abbreviation in schemas, which is helpful since quantity names and units, if represented in the column name, are typically stored in abbreviated forms. For instance, expanding abbreviated schemas such as "len" to "length" and "sqmi" to "square miles" is in fact detecting and extracting quantity names and units from column names. Instead of expanding abbreviations, we directly detect unit abbreviations to suggest particular quantity names.

3 Datasets

In our experiments, we use comma-separated value (CSV) format files sampled from the government dataset repository *data.gov*, which contains real datasets on various topics including finance, health, education, climate, etc. One dataset sometimes contains multiple CSV files, so we give each dataset a dataset ID, and each CSV file a CSV ID. Since columns representing quantities must be numeric, we filter out columns with other data types and retain only numeric columns.

After an initial exploration of the *data.gov* data, we select five popular quantities: *length, time, percent, currency*, and *weight* and focus on these quantities in our experiment. They are summarized in Table 1. For each quantity name, we list possible units as well as abbreviations, since units are sometimes provided in the column name by unit itself or in its abbreviated form. Besides these, we include contexts for each quantity name,

Table 1: Terms associated with each quantity name.

Quantity Name	**Units**	**Abbreviation**	**Context**
Length	meter, mile, inch, feet	m, mi, in, ft	height, width
Time	second, minute, hour	sec, s, min, hr, hrs	duration
Percent	percentage	%	accuracy
Currency	dollar, euro, pound	USD, $, EUR, GBP	amount, cost
Weight	gram, kilogram, pound, ounce, ton	g, kg, lb, oz, t	

because a quantity is stored and marked with some related words in many situations. For example, data with column name "duration of a trip" most likely is a quantity of *time*, and a column describing the height of a person should belong to quantity *length*.

We manually labeled columns which belong to the five quantities with 1-5, corresponding to *length*, *time*, *percent*, *currency*, and *weight* quantities, respectively. For those columns whose quantities do not belong to any of these five quantity names, we label them with 0.

When looking at the dataset created at this point, we notice that it contains many duplicate column names. For example, we find that the column name called "Shape_Length" appears around 650 times. This will considerably affect the class distribution and may lead to over-fitting. Therefore, we remove duplicate column names with the same dataset ID. After that, we obtain a dataset with 896 column instances of *length*, 352 instances of *time*, 1031 instances of *percent*, 875 instances of *currency*, and 233 instances of *weight*. (To match the size of the popular *percent* quantity name, we include 1031 instances within the *other* class.) Since the number of instances in each quantity are different, which makes the dataset imbalanced, we deal with this problem by upsampling less frequent classes during cross validation (which will be discussed further in Section 5).

The datasets, as well as the code implementing our proposed method, are publicly available at https://github.com/yay218/RecognizingQuantityName.

4 Features

Establishing good features plays a pivotal role in the process of training models, and will significantly affect performance. In this section, we describe our approach to predict the quantity name by creating features based on column name and column content.

For column content, we consider that all data cells contain numerical values, and we are looking for patterns for each quantity name. For example, *percent* has data cell values from 0 to 100, or 0 to 1. *Years* usually have values from within the past few hundred years. Therefore, we calculate features *maximum value*, *minimum value*, *average value*, and *range* for all column content. Moreover, we notice that some columns such as *year* are integers with a constant length, while some are decimals with one or more digits after the decimal point. Hence, we record the string length of the maximum value in the column. For example, the column with maximum number 5140 has length of 4, and the column with maximum value 3.1415 has length of 6.

Column name also has great impact on quantity name detection. For some quantities, the column name consists of multiple words, and is even longer if units are provided. To take advantage of this, another feature is built to count the number of white-space delimited words in the column name. However, we find that many column names have underscores or utilize CamelCase to connect words, e.g., ExtraFeeAmount, Low_Confidence_Limit. Thus, we add one more feature to count the number of characters in the column name. In addition, Sarawagi et al. [5] propose a rule-based method to extract units that are already provided in the column name. Inspired by their work, we establish a feature that incorporates more rules for analyzing the column name. Besides checking whether quantity names and units appear in possibly abbreviated forms in parentheses (e.g., Perimeter (meter)), after "in" (e.g., Dist. from Coop in miles), and after a dash or underscore (e.g., segment_length_miles), we also look for a match of the context terms in Table 1. This feature records the presence of quantity-specific terms, and consists of an array of 5 boolean features in which each corresponds to a quantity name, and has value 1 if there's a match, and otherwise 0. For example, column name "Canopy height in meters" has 1 in *length* and 0 for the others because *meter* is a unit appearing after "in" and *height* is present, and column name "Trip duration" has 1 in *time* and 0 for others because *duration* matches with the context for the *time* quantity. All features are summarized in Table 2.

Table 2: Features for Classification

Built From	ID	Type	Feature
Column Content	1	Real with length 1	Maximum value
	2	Real with length 1	Minimum value
	3	Real with length 1	Average value
	4	Real with length 1	Range value (maximum - minimum)
	5	Integer with length 1	Length of the maximum value (when expressed as a string)
Column Name	6	Integer with 1ength 1	Number of words
	7	Integer with length 1	Number of characters
	8	Array of 5 booleans	Presence of quantity-specific terms for each quantity name

5 Classification Models

For careful estimation of generalization performance, we apply ten-fold cross validation to the classification model. Since the number of instances for each quantity name are different, we design our cross validation process to upsample the minority classes to handle the imbalanced dataset. Within each round of cross validation, 90% of the instances are training, while other 10% are testing for each split. For the 90% training part, we find the quantity name which has the largest size, and we upsample the other classes with smaller size to match the size of the largest quantity name by replicating instances in those classes. Instances are selected randomly and added to classes containing fewer instances so that each quantity name has the same number of instances. However, we do not upsample the 10% testing part to maintain evaluation fidelity. Thus, for each round of cross validation, the dataset consists of 5568 instances for training (928 for each unit type) and 440 instances for testing in each shuffle within the cross validation process.

For this multi-class classification task, we compare: 1) random forest, 2) Naive Bayes classifier for multivariate Bernoulli models, and 3) SVM with a linear kernel (LinearSVC).

6 Evaluation

In order to evaluate the performance of our approach, we use 10-fold cross validation as mentioned in Section 5. The accuracy of Naive Bayes classifier for multivariate Bernoulli models is 77.3%, and the accuracy of SVM with a linear kernel is 48.7%. The random forest model with 200 trees and max depth 200 produces the best accuracy of 89.5%.

The confusion matrix for the random forest model is shown in Table 3. Most of the falsely predicted instances are quantity names that do not belong to any of the five selected names, in other words, are labeled with 0. For example, "toe(s)" is predicted to be quantity *time*, "sqmiles" to be quantity *length*, and "Number of Boats" to be quantity *weight*. There are many errors for instances which belong to the five selected types too. For instance, some "Shape_Length" are falsely classified as not belonging to any type, while others are correctly recognized. "Elevation, ft" is predicted to not belong to any quantity name although "ft" is already provided in the column name. "Graduation_rate" is falsely predicted to be *length* quantity, "Refunds - Individual Income Tax" is predicted to belong to *percent* instead of *currency*, and "Steel (lbs)" is falsely predicted as not belong to any quantity name.

As our features are built from either the content or the name of each column, we compare the results of models using different subsets of features utilizing the random forest model. The result is shown in Figure 1. The model with features built from full column content only has 61.9% accuracy, and the model with features exclusively from full column names achieves a higher accuracy at 84.8%. Performance of the extraction features alone (corresponding roughly to the rule-based model from Sarawagi et al. [5]) is in between, at 77.2%. Best

Table 3: Confusion Matrix for Random Forest

	Predicted Class					
Actual	Length	Time	Percent	Currency	Weight	Other
Length	86	0	0	0	0	3
Time	0	34	0	0	0	1
Percent	0	0	100	0	0	3
Currency	0	0	1	76	1	9
Weight	0	0	0	0	20	3
Other	1	1	12	4	1	84

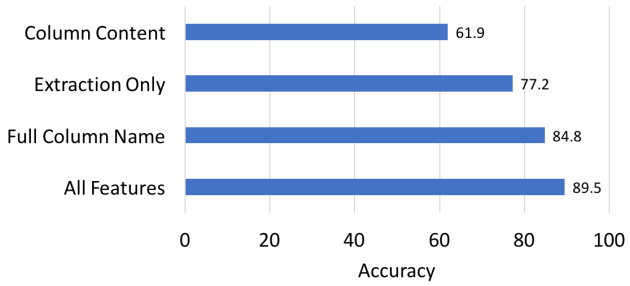


Figure 1: Prediction accuracy depending on features used.

performance (89.5%) is when both column content-based and column name-based types of features are used together.

To explore the importance of each individual feature, we calculate information gain producing the rank of feature importance shown in Figure 2(a). We find that the presence of quantity-specific terms has great importance for the result, and features like number of characters in the column name, maximum value and range in the column content are also useful. Minimum value and number of words in the column name rank the lowest.

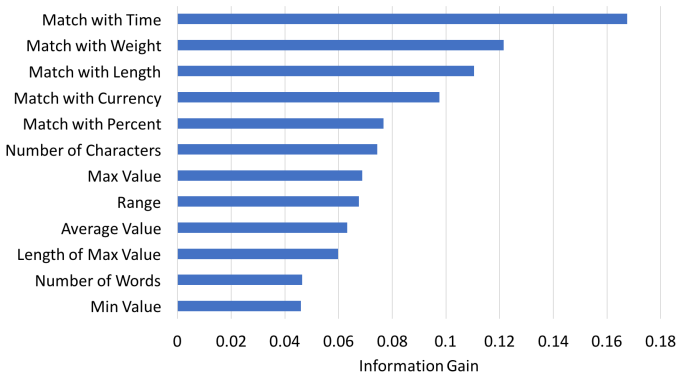
The choices for the maximum tree depth and number of trees can affect performance. The graph of the accuracy against number of trees and max depth is shown in Figure 2(b). We choose a maximum depth of 200 when varying the number of trees, and select 200 trees when studying performance when max depth varies. As we can see in the graph, accuracy rises rapidly with increasing number of trees and maximum depth.

7 Summary and Future Work

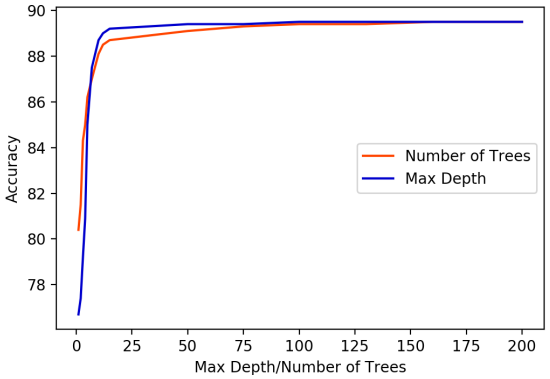
In this paper, we investigate a method to recognize and recommend quantity names for numeric columns in datasets from *data.gov*. From the analysis of column name and column content, we establish a variety of features and manually assign class labels to create this multi-class classification task. Ten-fold cross validation finds the estimated performance of a random forest model, and we show that our approach is able to predict quantity names with surprising accuracy.

This model works well in part because the presence of quantity-specific terms within column names is such a strong signal for recognizing quantity names. Column content improves the overall accuracy and it performs unexpectedly well (over 60% accuracy) when considered alone. In addition, our experiment only focuses on five quantity names, which is likely small enough to lead to high classification accuracy.

Our features based on column name, except without a few features, are very similar to the rule-based approach presented by Sarawagi et al. [5]. They use a feature-based approach that looks at column name and column content, but also create features to handle compound units, which is very different. By applying and expanding their rule-based extractor on our dataset, we get an accuracy of 77.2%. Compared to their work, our method



(a) Feature importance based on information gain.



(b) Accuracy as number of trees and maximum tree depth is varied.

Figure 2: Feature and meta-parameter studies.

better infers quantity names, because we also predict for columns that quantity names and units that are not explicitly provided. Applying our work to dataset indexing and search should greatly improve the matching scope when queries ask for information about quantity types.

Many directions are possible for further work. Features could be developed from exploring the meta data and description of datasets. Some datasets have attribute information provided in the description or an explicit data dictionary in a separate file, which likely contains much more detailed information than column name itself. It will also be interesting and useful to expand this work to actual units. In addition, the table containing terms associated with quantity names can be expanded so that more quantity names and units (including those are not that popular) can be included in predictions. The context words associated with a quantity name in the table can also be more comprehensive.

Acknowledgments

This material is based upon work supported by a Lehigh University internal Collaborative Research Opportunity grant.

References

[1] Au, V., Thomas, P., Jayasinghe, G.K.: Query-biased summaries for tabular data. In: Proc. 21st Australasian Document Computing Symp. ADCS '16 (2016) 69–72

[2] Chen, Z., Jia, H., Heflin, J., Davison, B.D.: Generating schema labels through dataset content analysis. In: Companion Proceedings of The Web Conference. WWW '18 (2018) 1515–1522

[3] Das Sarma, A., Fang, L., Gupta, N., Halevy, A., Lee, H., Wu, F., Xin, R., Yu, C.: Finding related tables. In: Proc. ACM SIGMOD Int'l Conf. on Management of Data. SIGMOD '12 (2012) 817–828

[4] Ratinov, L., Gudes, E.: Abbreviation expansion in schema matching and web integration. In: Proc. IEEE/WIC/ACM Int'l Conf. on Web Intelligence. WI '04 (2004) 485–489

[5] Sarawagi, S., Chakrabarti, S.: Open-domain quantity queries on web tables: Annotation, response, and consensus models. In: Proc. 20th ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining. KDD '14 (2014) 711–720

[6] Thomas, P., Omari, R., Rowlands, T.: Towards searching amongst tables. In: Proc. 20th Australasian Document Computing Symp. ADCS '15 (2015) 8:1–8:4

[7] Valera, I., Ghahramani, Z.: Automatic discovery of the statistical types of variables in a dataset. In: Int'l Conf. on Machine Learning. (2017) 3521–3529

[8] Wick, M.L., Rohanimanesh, K., Schultz, K., McCallum, A.: A unified approach for schema matching, coreference and canonicalization. In: Proc. 14th ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining. KDD '08 (2008) 722–730