Why Big Data Industrial Systems Need Rules and What We Can Do About It

Paul Suganthan G.C.[1], **Chong Sun**[2], **Krishna Gayatri K.**[1], **Haojun Zhang**[1], **Frank Yang**[3], **Narasimhan Rampalli**[4], **Shishir Prasad**[4], **Esteban Arcaute**[4], **Ganesh Krishnan**[4], **Rohit Deep**[4], **Vijay Raghavendra**[4], **AnHai Doan**[1]

[1]University of Wisconsin-Madison, [2]Uber, [3]LinkedIn, [4]@WalmartLabs*

ABSTRACT

Big Data industrial systems that address problems such as classification, information extraction, and entity matching very commonly use hand-crafted rules. Today, however, little is understood about the usage of such rules. In this paper we explore this issue. We discuss how these systems differ from those considered in academia. We describe default solutions, their limitations, and reasons for using rules. We show examples of extensive rule usage in industry. Contrary to popular perceptions, we show that there is a rich set of research challenges in rule generation, evaluation, execution, optimization, and maintenance. We discuss ongoing work at WalmartLabs and UW-Madison that illustrate these challenges. Our main conclusions are (1) using rules (together with techniques such as learning and crowdsourcing) is fundamental to building semantics-intensive Big Data systems, and (2) it is increasingly critical to address rule management, given the tens of thousands of rules industrial systems often manage today in an ad-hoc fashion.

Categories and Subject Descriptors

H.2.4 [**Information Systems**]: Database Management - Systems

Keywords

Rule Management; Big Data; Classification

1. INTRODUCTION

Big Data is now pervasive, leading to many large-scale problems in classification, information extraction (IE), and entity matching (EM), among others. Industrial solutions to these problems very commonly use hand-crafted rules (in addition to techniques such as learning and crowdsourcing) [33, 8, 12]. Examples of such rules are *"if the title of a product contains the phrase 'wedding band' then it is a ring"* and *"two books match if they agree on the ISBNs and the number of pages"*.

Today, however, little is understood about rules in such Big Data industrial systems. What kinds of rules do they typically use? Who writes these rules? Why do they use rules? Are these reasons fundamental, suggesting that rules will be used in the foreseeable future? How pervasive is rule usage? How many rules are there in a typical system? Hundreds of rules, or tens of thousands? Is there any interesting research agenda regarding rules? Isn't it simply the case that someone just sits down and manually writes rules? Can't we just use machine learning?

As far as we can tell, very little if any has been published about these issues, with the lone exception of [8], which shows that rule usage is pervasive in industrial IE systems (see the related work section).

In this paper, we build on our experience at WalmartLabs, Kosmix (a startup acquired by Walmart), and UW-Madison to explore these issues in depth. We consider Big Data systems that focus on *semantics intensive topics*, such as classification, IE, clustering, EM, vertical search, knowledge base (KB) construction, and tweet interpretation, among others. These systems often use a lot of domain knowledge to produce results of high accuracy (e.g., high precision and recall). Numerous such systems have been deployed in practice, and more are coming.

To make the discussion concrete, throughout the paper we will focus on Big Data classification systems, in particular on Chimera, a product classification system at WalmartLabs. This system has been described in [33]; here we only focus on aspects that are relevant to rule usage. Later we will touch on other types of Big Data systems such as IE and EM ones.

We discuss four important characteristics of the classification problem underlying Chimera. (1) The data is large scale, never ending, and ever changing, arriving in batches at irregular time intervals. (2) Chimera must maintain high precision (at least 92%) at all time, and improve recall (which can be low initially) over time. (3) System accuracy can quickly degrade, due to ever-changing and unpredictable data. Thus we need a way to quickly debug, "scale down", repair, and restore the system. (4) The team size is limited, consisting of a few CS developers and domain analysts, and some crowdsourcing budget. As a result, first-responder work as well as ongoing monitoring, repair, and improvement work

*Work done while the second and fifth authors were at WalmartLabs.

SIGMOD '15 Melbourne, Victoria, Australia
Copyright 2015 ACM 978-1-4503-2758-9/15/05 ...$15.00.
http://dx.doi.org/10.1145/2723372.2742784.

are done mostly by domain analysts, who have little or no CS background.

We discuss how these characteristics set Chimera-like problems apart from those considered in academia. Briefly, academia develops *algorithms* to perform *one-shot* classification, whereas industry develops *systems* to perform *ongoing* classification. This fundamental difference produces drastically different objectives.

Next, we describe a common learning-based solution to classification, limitations of this solution, and reasons for using rules. There are many. Sometimes it is much easier to write rules than to learn (e.g., to capture the knowledge that if a product has an attribute called "isbn" then it is a book). When the system makes a mistake, it is often faster and easier for analysts (who have no or minimal CS background) to write rules to "patch" the system behavior, until when CS developers can debug and fix the underlying problems. Rules can handle cases that learning cannot yet handle, as well as "corner cases". Rules enable rapid debugging and adjusting the system (e.g., scaling it down). Liability concerns may require certain predictions to be explainable, which in turn requires rules, and so on. Many of these reasons are fundamental and so rules are likely to be used into the foreseeable future.

Next, we show that, contrary to the popular perception in academia, there is a rich set of research challenges in rule languages, rule system properties and design, rule generation, quality evaluation, rule execution and optimization, and rule maintenance. These challenges span the entire gamut from low-level system challenges (e.g., how to execute tens of thousands of rules on a machine cluster) to high-level user challenges (e.g., how to help domain analysts write rules).

To further illustrate these challenges, we describe ongoing rule management work at WalmartLabs and UW-Madison. Among others, we discuss work that helps domain analysts refine classification rules in mere minutes (rather than hours), and work that uses labeled data to automatically generate a large set of rules, thereby significantly improving the system's recall.

Finally, we discuss how other types of Big Data industrial systems also use rules quite extensively, often for many of the reasons underlying Chimera. We consider for example IE systems being developed at WalmartLabs and elsewhere, EM systems at WalmartLabs, and knowledge base construction systems and tweet tagging and monitoring systems at Kosmix.

We close the paper by circling back to the notion of *semantics intensive* Big Data industrial systems. Fundamentally, *such systems must utilize a lot of domain knowledge, in whatever ways judged most efficient, using a limited team of people with varying technical expertise.* As a result, they often use a wide range of techniques, including rules, learning, knowledge bases, and crowdsourcing. Thus, we believe that rules are fundamental, but must be used in conjunction with other techniques. As rules are being increasingly used in industry, often in an ad-hoc fashion, and as Big Data industrial systems keep accumulating rules, often in the tens of thousands, articulating and addressing a research agenda on rule management are becoming increasingly critical, and the goal of this paper is to help make a small step in this direction.

2. BIG DATA CLASSIFICATION

As mentioned in the introduction, to make the discussion concrete, we will focus on Big Data classification systems, in particular on Chimera, a product classification system at WalmartLabs. Section 6 discusses other types of Big Data systems, such as IE and EM ones.

In this section we briefly describe the classification problem underlying Chimera (see [33] for more details), the main difference between this problem and those commonly considered in academia, and a popular learning-based solution. Section 3 discusses limitations of this solution, reasons for rules, and how Chimera uses rules today.

2.1 Product Classification

The Chimera system has been developed at WalmartLabs in the past few years to classify millions of product items into 5,000+ product types [33]. A *product item* is a record of attribute-value pairs that describe a product. Figure 1 shows three product items in JSON format. Attributes "Item ID" and "Title" are required. Most product items also have a "Description" attribute, and some have more attributes (e.g., "Manufacturer", "Color"). Millions of product items are being sent in continuously from thousands of Walmart vendors (because new products appear all the time).

We maintain more than 5,000 mutually exclusive *product types*, such as "laptop computers", "area rugs", "dining chairs", and "rings". This set of product types is constantly being revised. For the purpose of this paper, however, we will assume that it remains unchanged; managing a changing set of types is ongoing work.

Our goal then is to classify each incoming product item into a product type. For example, the three products in Figure 1 are classified into the types "area rugs", "rings" and "laptop bags & cases" respectively.

2.2 Problem Requirements

Developing Chimera raises the following difficult issues regarding the data, quality, system, and the team.

1. Large Scale, Never Ending, Ever-Changing Data: Chimera is clearly large-scale: it must classify millions of product items into 5,000+ types. Chimera is also "never ending": it is deployed 24/7, with batches of data arriving at irregular intervals. For example, in the morning a small vendor on Walmart's market place may send in a few tens of items, but hours later a large vendor may send in a few millions of items.

Since the data keeps "trickling in", Chimera can never be sure that it has seen the entire universe of items (so that it can take a random sample that will be representative of items in the future, say). This of course assumes that the overall distribution of items is static. Unfortunately at this scale it often is not: the overall distribution is changing, and concept drift becomes common (e.g., the notion "computer cables" keeps drifting because new types of computer cables keep appearing).

2. Ongoing Quality Requirements: When data arrives, Chimera tries to classify as fast and accurately as possible. The business unit requires that Chimera maintain high precision *at all times*, at 92% or higher (i.e., at least 92% of Chimera's classification results should be correct). This is because the product type is used to determine the "shelf" on walmart.com on which to put the item. Clearly, incorrect

```
{ "Item ID" : 30427934,
  "Title" : "Eastern Weavers Rugs EYEBALLWH-8x10 Shag Eyeball White 8x10 Rug",
  "Description" : "Primary Color: White- Secondary Color: White- Construction: Hand Woven- Material: Felted Wool- Pile Height: 1in - Style: Shag SKU: EASW1957" }

{ "Item ID" : 31962310,
  "Title" : "1-3/8 Carat T.G.W. Created White Sapphire and 1/4 Carat T.W. Diamond 10 Carat White Gold Engagement Ring",
  "Description" : "The engagement ring showcases a stunning created white sapphire at its center and a total of 24 round, pave-set diamonds along the top and sides
  of the rib-detailed band." }

{ "Item ID" : 17673919,
  "Title" : "Royce Leather Ladies Leather Laptop Briefcase",
  "Description" : "This handy ladies laptop brief has three zippered compartments.Topped off with two handles and a comfortable shoulder strap." }
```

Figure 1: Three examples of product items.

classification will put the item on the wrong "shelf", making it difficult for customers to find the item.

It is difficult to build a "perfect" system in "one shot". So we seek to build an initial Chimera system that maintains precision of 92% or higher, but can tolerate lower recall. (This is because items that the system declines to classify will be sent to a team that attempts to manually classify as many items as possible, starting with those in product segments judged to be of high value for e-commerce.) We then seek to increase Chimera's recall as much as possible, over time.

3. Ongoing System Requirements: Since the incoming data is ever changing, at certain times Chimera's accuracy may suddenly degrade, e.g., achieving precision significantly lower than 92%. So we need a way to detect such quality problems quickly.

Once detected, we need a way to quickly "scale down" the system, e.g., disabling the "bad parts" of the currently deployed system. For example, suppose Chimera suddenly stops classifying clothes correctly, perhaps because all clothes in the current batch come from a new vendor who describes them using a new vocabulary. Then Chimera's predictions regarding clothes need to be temporarily disabled. Clearly, the ability to quickly "scale down" Chimera is critical, otherwise it will produce a lot of incorrect classifications that significantly affect downstream modules.

After "scaling down" the system, we need a way to debug, repair, then restore the system to the previous state quickly, to maximize production throughputs.

Finally, we may also need to quickly "scale up" Chimera in certain cases. For example, Chimera may decline to classify items sent in by a new vendor, because they belong to unfamiliar types. The business unit however may want to "onboard" these items (i.e., selling them on walmart.com) as soon as possible, e.g., due to a contract with the vendor. In such cases we need a way to extend Chimera to classify these new items as soon as possible.

4. Limited Team Sizes with Varying Abilities: We have limited human resources, typically 1-2 developers, 2-3 analysts, and some crowdsourcing budget. Such limited team sizes are actually quite common. As all parts of society increasingly acquire, store, and process data, most companies cannot hire enough CS developers, having to "spread" a relatively small number of CS developers over a growing number of projects. To make up for this, companies often hire a large number of *data analysts*.

The analysts cannot write complex code (as they typically have no or very little CS training). But they can be trained to understand the domain, detect patterns, perform semantics-intensive QA tasks (e.g., verifying classification results), perform relatively simple data analyses, work with the crowd, and write rules. (It is important to note that domain analysts are not *data scientists*, who typically have extensive training in CS, statistics, optimization, machine learning, and data mining.) Due to limited human resources, when a system like Chimera has become relatively stable, the managers often move most CS developers to other projects, and ask the data analysts to take over monitoring the system. Thus, *if system accuracy degrades, the analysts are often the first responders and have to respond quickly.*

Finally, many CS developers change jobs every few years. When this happens, it shifts even more of the burden of "babysitting" the system to the analysts. In addition, we want the system to be constructed in a way that is easy for newly hired CS developers to understand and extend.

2.3 Comparison with Problems in Academia

As described, the problem facing Chimera (and other similar Big Data industrial systems) is very different compared to classification problems commonly considered in academia. Briefly, academia develops *algorithms* to perform *one-shot* classification, whereas industry develops *systems* to perform *ongoing* classification. This fundamental difference produces drastically different objectives.

Specifically, academia typically seeks to develop algorithms to classify a given set of items, with the objective of maximizing F_1 while minimizing resources (e.g., time, storage). *There is no system nor team requirement.* It is typically assumed that a user will just run the algorithm automatically, in one shot (though sometimes interactive human feedback may be considered).

In sharp contrast, the main objective of Chimera is not to maximize F_1, but to maintain a high precision threshold *at all time*, while trying to improve recall *over time*. System requirements such as the ability to rapidly identify mistakes, scale down the system, scale up the system, etc., are extremely important to enable real-world deployment. Furthermore, this must be done given the constraints on the team's size, abilities, and dynamics over time. The main implication, as we will see, is that Chimera should use not just classification algorithms already developed in academia, but also a broad range of other techniques, including hand-crafted rules, in order to satisfy its ongoing objectives.

3. USING RULES FOR CLASSIFICATION

We now describe why and how rules are used in Big Data classification systems such as Chimera. But first we describe a learning-based solution commonly employed as the first solution in such systems (and in fact, it was the first solution we tried for Chimera).

3.1 A Popular Learning-Based Solution

It is clear that we cannot manually classify all incoming products, nor write manual rules for all 5,000+ product types. Thus, a common first solution is to use machine learning.

In this solution, we start by creating as much training data as possible, using manual labeling and manual rules (i.e., creating the rules, applying them, then manually curating the results if necessary). Next, we train a set of learning-based classifiers (e.g., Naive Bayes, kNN, SVM, etc.), often combining them into an ensemble. When a batch of data comes in, we apply the ensemble to make predictions. Next, we evaluate the precision of the predicted result, by taking one or more samples then evaluating their precision using crowdsourcing or analysts. If the precision of the entire result set is estimated to satisfy the required 92% threshold, we pass the result set further down the production pipeline.

Otherwise we reject the batch, in which case the CS developers and analysts try to debug and improve the system, then apply it again to the batch. The system may also decline to make predictions for certain items in the batch. These items are sent to a team that will attempt to manually label as many items as possible, while the CS developers and analysts attempt to improve Chimera so that it will make predictions for such items in the future. In parallel with these efforts, the CS developers and analysts will improve Chimera further, whenever they have time, by creating more training data, revising the learning algorithms, and adding new learning algorithms.

3.2 The Need for Rules

The above solution when employed in Chimera did not reach the required 92% precision threshold. Subsequently, we decided to "augment" it with rules. In what follows we discuss cases where using rules is necessary or highly beneficial.

The Obvious Cases: Domain analysts often can quickly write a set of rules that capture obvious classification patterns, e.g., if a product has an "isbn" attribute, then it is a book. In such cases, it does not make sense to try learning. Rather, analysts should write these "obvious" rules (to be added to the system in a rule-based module).

Analysts also often write rules to generate training data. For example, an analyst writes a rule to classify products as "motor oil", applies the rule, examines the predictions, corrects or removes wrong predictions, then adds the remaining predictions to the set of training data. In such cases, if the rules seem accurate, we should also use them during the classification process, e.g., by adding them to the rule-based module mentioned above.

Correcting Learning's Mistakes: Recall from Section 2.2 that when the system makes a mistake, we want to find the reason and fix it quickly. However, when the learning ensemble makes a mistake, it is often hard to find the reason. Once we have found the reason, it is still hard to know how to correct it. For example, it is often not clear if we should correct the current training data (and how), add more training data, add new features, or adjust the parameters, etc.

In addition, diagnosing and correcting such mistakes often require deep knowledge about the learning algorithms. So only the CS developers can do this, not the analysts. However, there may not be enough CS developers. The CS developers may not be around to do it right away (recall that once the system has become stable, they may be moved to other projects). Even if the CS developers are around to do it, it may take a while and multiple rounds of trial and error until the problem has been fixed satisfactorily.

In practice, more often than not, *domain analysts are the first responders, and they must correct such mistakes quickly.* But clearly the analysts cannot correct the "deep problems" in the internals of the learning algorithms. They can only do what we call *"shallow behavioral modification"*. Specifically, they try to detect patterns of error (and are often quite good at detecting these), then write hand-crafted rules to handle such patterns. When the CS developers have time, they will come in and try *"deep therapy"*, i.e., adjusting the learning algorithms, if necessary.

As an imperfect analogy, suppose a person X is very good at predicting many things, but often makes irrational predictions regarding stocks. Eventually a psychologist can do "deep therapy" to diagnose and fix this problem. In the mean time, X's friends, not having the training of a psychologist, will simply apply a "shallow behavior filtering rule" to ignore all stock related predictions of X.

Covering Cases that Learning Cannot Yet Handle: In such cases the analysts can write rules to improve the system as much as possible. For example, creating training data for learning algorithms is time consuming. So right now Chimera has no training data for many product types (or has very little training data and hence learning-based predictions for these product types are not yet reliable). In these cases the analysts may want to create as many classification rules as possible, to handle such product types, thereby increasing the recall.

As yet another example, in certain cases it may be very difficult to create *representative* training data for learning algorithms. For instance, how do we obtain a representative sample for "handbags"? All the items named "satchel", "purse" and "tote" are handbags and it is hard to collect a complete list of these representative items. Another example is "computer cables", which includes a variety of cables, such as USB cables, networking cords, motherboard cables, mouse cables, monitor cables, and so on. To obtain a representative sample for such a product type, an analyst may have to search the Web extensively for hours or days, to understand all the various subtypes that belong to the same product type, a highly difficult task. In such cases it may be better for analysts to write rules to classify products already known to belong to these types, then add more rules as new kinds of products are discovered to also belong to these types.

Concept Drift & Changing Distribution: A related problem is that we do not know *a priori* the items of each product type. In fact, the set of such items keeps changing, as new products appear. For example, the set of items for "smart phone" keeps changing, as the notion of what it means to be a smart phone changes, and new types of smart phone appear on the market. Thus, we have a "concept drift". The distribution of all product items (over all product types) is also constantly changing. Today there may be a large portion of products in "Homes and Garden", but tomorrow this portion may shrink, in reaction to seasonal and market changes. The concept drift and changing distribution make it difficult to learn effectively. This in turn increases the number of mistakes made by learning, and requires using even more rules to "patch" the system's behavior.

Covering "Corner Cases": We often have "corner cases" that come from special sources and need to be handled in special ways. For example, Walmart may agree to carry a limited number of new products from a vendor, on a trial basis. Since these products are new, training data for them is not yet available, and it is also difficult to generate sufficient training data for these cases. As a result, we cannot reliably classify them using learning. On the other hand, analysts often can write rules to quickly address many of these cases.

It is also difficult to use learning alone to "go the last mile". We can use it to achieve a certain precision, say 90%. But going from there to near 100% is difficult, because these remaining percentages often consists of "corner cases", which by their very nature are hard to generalize and thus are not amenable to learning. To handle such cases, we often must write rules.

Ability to Trace Errors & Adjust the System Quickly: Recall that we want to debug a mistake and possibly "scale down" the system quickly. Rules are naturally good for this purpose, because if a misclassification is caused by a rule, we can find that rule relatively quickly. If that rule misclassifies widely, we can simply disable it, with minimal impacts on the rest of the system. Thus, rules make the system "compositional". In contrast, even if we can trace an error to a learning module, disabling the module may significantly impact the rest of the system. Furthermore, disabling or correcting a few rules is significantly faster than retraining certain learning modules. Finally, domain analysts (who are often the first responders) can work much more easily with rules than with learning modules.

Business Requirements: Business units often impose constraints that require using rules. For example, legal and liability concerns may require the system to be able to *explain* (or explain *quickly*, should the need arises) why it classifies certain products into certain types (e.g., medicine). In such cases, rules will be used to ensure a clear explanation can be generated quickly. As another example, a business unit may require absolute certainty with respect to certain product types. In such cases a rule is inserted killing off predictions regarding these types, routing such product items to the manual classification team.

Other Considerations: Finally, there is a host of other factors that also motivate using rules. Many systems like Chimera use domain knowledge encoded in knowledge bases (KBs), and a natural way to use such KBs is to write rules. For example, Chimera uses several KBs that contain brand names associated with product types. So if a product title mentions "Apple", say, then Chimera knows that the product belongs to a limited list of types (e.g., phone, laptop, etc.). Chimera applies such reasoning using rules. As another example, the output of Chimera may be curated further by people in other units downstream. Such curations can be captured using rules then added back to Chimera, to improve its future performance.

3.3 How Chimera Uses Rules Today

We now describe how Chimera uses rules (in conjunction with learning). Figure 2 shows the overall architecture (see

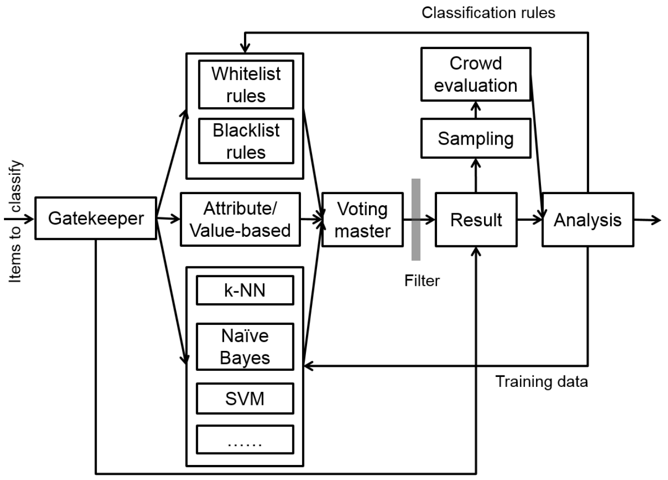

Figure 2: The Chimera system architecture.

[33] for a detailed description). Given items to classify, the Gate Keeper does preliminary processing, and under certain conditions can immediately classify an item (see the line from the Gate Keeper to the Result).

Items surviving the Gate Keeper are fed into classifiers. There are three types of classifiers. (1) A rule-based classifier that consists of a set of whitelist rules and blacklist rules created by the analysts (see below). (2) An attribute/value-based classifier that consists of rules involving attributes (e.g., if a product item has the attribute "ISBN" then its type is "Books") or values (e.g., if the "Brand Name" attribute of a product item has value "Apple", then the type can only be "laptop", "phone", etc.). (3) A set of learning-based classifiers: k-NN, Naive Bayes, SVM, etc.

Given an item, all classifiers make predictions (each prediction is a list of product types together with weights). The Voting Master and the Filter combine these predictions into a final prediction, which is added to the result set.

Next, we use crowdsourcing to evaluate a sample of the result set. Given a pair ⟨product item, final predicted product type⟩, we ask the crowd if the final predicted product type can indeed be a good product type for the given product item. Pairs that the crowd say "no" to are flagged as potentially incorrect, and are sent to the analysts (the box labeled Analysis in the figure). The analysts examine these pairs, create rules, and relabel certain pairs. The newly created rules are added to the rule-based and attribute/value-based classifiers, while the relabeled pairs are added to the learning-based classifiers as training data.

If the precision on the sample (as verified by the crowd) is already sufficiently high, the result set is judged sufficiently accurate and sent further down the production pipeline. Otherwise, we incorporate the analysts' feedback into Chimera, rerun the system on the input items, sample and ask the crowd to evaluate, and so on.

If the Voting Master refuses to make a prediction (due to low confidence), the incoming item remains unclassified. The analysts examine such items, then create rules and training data, which again are incorporated into the system. Chimera is then rerun on the product items.

Rules in Chimera: The analysts can write rules for virtually all modules in the system, to improve the accuracy and exert fine-grained control. Specifically, they can add rules to the Gate Keeper to bypass the system, to the classifiers in form of whitelist- and blacklist rules, to the Combiner to control the combination of predictions, and to the Filter to control classifiers' behavior (here the analysts use mostly blacklist rules). Finally, in the evaluation stage they can examine the various results and write many rules.

We keep the rule format fairly simple so that the analysts, who cannot program, can write rules accurately and fast. Specifically, we ask them to write *whitelist rules* and *blacklist rules.* A whitelist rule $r \rightarrow t$ assigns the product type t to any product whose title matches the regular expression r. Example whitelist rules that our analysts wrote for product types "rings" are:

rings? $\rightarrow$ rings
diamond.*trio sets? $\rightarrow$ rings

The first rule for example states that if a product title contains "ring" or "rings", then it is of product type "rings". Thus this rule would correctly classify the following product titles as of type "rings": "Always & Forever Platinaire Diamond Accent Ring" and "1/4 Carat T.W. Diamond Semi-Eternity Ring in 10kt White Gold". Similarly, a blacklist rule $r \rightarrow NOT\ t$ states that if a product title matches the regular expression r, then that product is not of the type t.

The Chimera system has been developed and deployed for about two years. Initially, it used only learning-based classifiers. Adding rules significantly helps improve both precision and recall, with precision consistently in the range 92-93%, over more than 16M items (see [33]).

As of March 2014, Chimera has 852K items in the training data, for 3,663 product types, and 20,459 rules for 4,930 product types (15,058 whitelist rules and 5,401 blacklist rules; an analyst can create 30-50 relatively simple rules per day). Thus, for about 30% of product types there was insufficient training data, and these product types were handled primarily by the rule-based and attribute/value-based classifiers.

4. A DIVERSE SET OF RESEARCH CHALLENGES FOR RULES

We now discuss how Big Data classification systems such as Chimera raise a rich and diverse set of challenges in rule languages, rule system properties and design, rule generation, quality evaluation, rule execution and optimization, and rule maintenance.

Rule Languages: A major challenge is to define rule languages that *analysts with no or minimal CS background* can use to write rules quickly and accurately. For example, Chimera showed that analysts can write blacklist and whitelist classification rules that apply relatively simple regexes to product titles. But this rule language is somewhat limited. For example, it does not allow analysts to state that "if the title contains 'Apple' but the price is less than $100 then the product is not a phone", or "if the title contains any word from a given dictionary then the product is either a PC or a laptop". Can we develop more expressive rule languages that analysts can use? Should such languages be declarative, procedural, or combining the elements of both? Can analysts write user-defined functions (at least certain relatively simple types of user-defined functions), and how?

Rule System Properties and Design: Over time, many developers and analysts will modify, add, and remove rules from a system. Rules will constantly be fixed, merged, and split. It is important that the system remain robust and predictable throughout such activities. Toward this goal, we need to identify and formalize desirable rule system properties. For example, one such property could be "the output of the system remains the same regardless of the order in which the rules are being executed".

We can then prove that certain systems possess certain properties, or design a rule system (e.g., deciding how rules will be combined, what types of rules will be executed in what order) to exhibit certain properties. For example, in Chimera the rule-based module always executes the whitelist rules *before* the blacklist rules [33]. So under certain assumptions on how the rules interact, we can prove that the execution order among the whitelist rules (or the blacklist rules) does not affect the final output of the rule-based module.

Rule Generation: Writing rules is often time consuming. Hence, a major challenge is to help the analysts write rules faster. For example, when an analyst writes a regex-based rule for the "motor oil" product type, how can we help the analyst quickly create the regex used in the rule?

The above challenge considers the scenario where the analyst tries to create a *single* rule. Sometimes he or she may want to create *many* rules. For example, suppose there is no learning-based method yet to classify products into a type t. This can happen because there may not yet be enough training data for t; or the CS developers may not yet be able to take on this task; or they have started, but it will take a while until learning-based methods have been thoroughly trained and tested. In such cases, the analyst may want to quickly generate as many rules as possible to classify products into type t. Such rules may not completely cover all major cases for t, but they will help increase recall. A challenge then is how to help the analyst create these rules.

In Section 5 we describe ongoing projects at WalmartLabs and UW-Madison to address the above two challenges. Finally, we note that another related challenge is how to use crowdsourcing to help the analysts, either in creating a single rule or multiple rules.

Rule Quality Evaluation: This is a major challenge in industry, given the large number of rules generated (anywhere from a few hundreds to a few tens of thousands for a system). Currently there are three main methods to evaluate the precision of rules. The first method uses a single validation set S, e.g., a set of products labeled with the correct product types, to estimate the precision of *each individual rule.* Generating this validation set S however is often very expensive, even if we use crowdsourcing.

Further, S can only help evaluate rules that touch items in S. In particular, it helps evaluate "head" rules, i.e., rules that touch many items (and so many of these items are likely to be in S). But it often cannot help evaluate "tail" rules, i.e., rules that touch only a few items. An example "tail" rule is "if a product title contains 'Christmas tree', then it belongs to the 'holiday decoration' product type" (assuming that the retailer sells only a few Christmas tree products). The validation set S often does not contain items touched by such "tail" rules. Increasing the size of S can help alleviate this problem, but would increase the cost of creating S (e.g., to label items in S with the correct product types).

The second method to evaluate rule quality creates a validation set per rule. For example, let A be the set of items touched by rule R_A. The work [18] proposes having the crowd evaluate a sample taken from A then using that sample to estimate the precision of R_A (this work actually focuses on entity matching rules but the same algorithm applies to classification rules). However, evaluating the precision of tens of thousands of rules this way incurs prohibitive costs. To address this problem, [18] exploits the overlap in the coverage of the rules. Specifically, let B be the set of items touched by another rule R_B. Assume that A and B overlap, we can sample in $A \cap B$ first (and outside that if necessary), then use the result to evaluate both R_A and R_B, thereby minimizing the number of items that we must sample. Again, this method works well for "head" rules, which touch a large number of items and thus are likely to overlap. It does not work well for "tail" rules, which often do not overlap in the coverage.

The third method gives up the goal of evaluating the individual rules, judging that to be too expensive. Instead, it tries to evaluate the quality of a *set of rules*, e.g., those in a rule-based module. Specifically, given a rule-based module M to evaluate, this method uses crowdsourcing to evaluate a sample taken from those items touched by M, then uses that sample to estimate the precision of M.

Clearly, none of these three methods is satisfactory. Rule quality evaluation therefore requires significantly more efforts, and we briefly describe one such ongoing effort at WalmartLabs and UW-Madison in Section 5.3.

Rule Execution and Optimization: Given the large number of rules generated, executing all of them is often time consuming, thus posing a problem for systems in production, where we often want the output in seconds or minutes. A major challenge therefore is to scale up the execution of tens of thousands to hundreds of thousands of rules. A possible solution is to index the rules so that given a particular data item, we can quickly locate and execute only a (hopefully) small set of rules (see [31] for an example of indexing information extraction rules to speed up their execution; the technique described there can potentially be applied to classification rules). Another solution is to execute the rules in parallel on a cluster of machines (e.g., using Hadoop).

Another interesting challenge concerns when the analyst is still developing a rule R (e.g., debugging or refining it). To do so, the analyst often needs to run variations of rule R repeatedly on a development data set D. To develop R effectively, the data set D often must be quite large. But this incurs a lot of time evaluating R even just once on D, thus making the process of developing R inefficient. To solve this problem, a solution direction is to index the data set D for efficient rule execution.

Rule Maintenance: Once generated and evaluated, rules need to be maintained over a long period of time. This raises many challenges. The first challenge is to detect and remove imprecise rules (despite the development team's best effort, imprecise rules may still be added to the system).

The second challenge is to monitor and remove rules that become imprecise or inapplicable. This may happen because the universe of products and the way products are described are constantly changing, thus making a rule imprecise. The product taxonomy may also change, rendering certain rules inapplicable. For example, when the product type "pants" is divided into "work pants" and "jeans", the rules written for "pants" become inapplicable. They need to be removed and new rules need to be written.

The third challenge is to detect and remove rules that are "subsumed" by other rules. For example, two analysts may independently add the two rules "denim.*jeans? $\rightarrow$ Jeans" and "jeans? $\rightarrow$ Jeans" to the system at different times. It is highly desirable to be able to detect that the first rule is subsumed by the second one and hence should be removed. A related challenge is to detect rules that overlap significantly, such as "(abrasive|sand(er|ing))[-](wheels?|discs?) $\rightarrow$ Abrasive wheels & discs" and "abrasive.*(wheels?|discs?) $\rightarrow$ Abrasive wheels & discs".

Finally, a major challenge is to decide how to consolidate or split rules (such as the two "wheels & discs" rules just listed above). Ideally, we want to consolidate the rules into a smaller, easier-to-understand set. But we have found that rule consolidation often makes certain tasks much harder for analysts. For example, if we consolidate rules A and B into a single rule C, then when rule C misclassifies, it can take an analyst a long time to determine whether the problem is in which part of rule C, e.g., rule A or B. Then once the problem has been determined, it is more difficult for the analyst to fix a single composite rule C, than a simpler rule A or B. Thus, there is an inherent tension between the two objectives of consolidating the rules and keeping the rules "small" and simple to facilitate debugging and repairing.

5. RULE MANAGEMENT WORK AT WALMARTLABS & UW-MADISON

We have described a diverse set of challenges for rule management, which so far have been addressed in an ad-hoc fashion. As rules proliferate, these ad-hoc solutions are becoming increasingly unmanageable. Hence, it is important to develop principled and effective solutions. We now briefly describe our ongoing effort toward this goal at WalmartLabs and UW-Madison. For space reasons, we focus on rule generation for product classification, then touch on other ongoing efforts.

5.1 Supporting Analysts in Creating Rules

As discussed earlier, rule generation is time consuming. Hence, we have developed a solution to help analysts write rules faster, by quickly finding "synonyms" to add to a rule under development.

Specifically, when creating rules, analysts often must write regular expressions. For example, to classify product items into "motor oil", the analyst may examine a set of product titles, then write the rule

R_1: (motor | engine) oils? $\rightarrow$ motor oil,

which states that if a product title contains the word "motor" or "engine", followed by "oil" or "oils", then it is "motor oil". Next, the analyst may examine even more product titles, to find more "synonyms" for "motor" and "engine", then add them to the regex. Eventually, the rule may become:

R_2: (motor | engine | auto(motive)? | car | truck | suv | van | vehicle | motorcycle | pick[-]?up | scooter | atv | boat) (oil | lubricant)s? $\rightarrow$ motor oil.

The first disjunction of the regex in the above rule contains 13 terms (e.g., "motor", "engine", etc.). Clearly, finding all

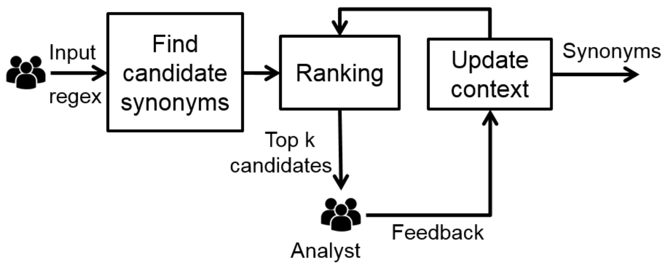

Figure 3: The architecture of the tool that supports analysts in creating rules.

such terms is error-prone and time consuming (often taking hours in our experience). To find these terms, the analyst often has to painstakingly "comb" a very large set of product titles, in order to maximize recall and avoid false positives.

In response, we developed a tool that helps the analyst find such terms, which we call "synonyms", in minutes instead of hours. The analyst start by writing a short rule such as Rule R_1 described above (see Figure 3). Next, suppose the analyst wants to expand the disjunction in R_1, given a data set of product titles D. Then he or she provides the following rule to the tool:

R_3: (motor | engine | \syn) oils? $\rightarrow$ motor oil,

where the string "\syn" means that the analyst wants the tool to find all synonyms for the corresponding disjunction (this is necessary because a regex may contain multiple disjunctions, and currently for performance and manage-ability reasons the tool focuses on finding synonyms for just one disjunction at a time).

Next, the tool processes the given data set D to find a set of synonym candidates C. Next, it ranks these synonym candidates, and shows the top k candidates. The analyst provides feedback on which candidates are correct. The tool uses the feedback to re-rank the remaining candidates. This repeats until either all candidates in C have been verified by the analyst, or when the analyst thinks he or she has found enough synonyms. We now describe the main steps of the tool in more details.

Finding Candidate Synonyms: Given an input regex R, we begin by obtaining a set of generalized regexes, by allowing any phrase up to a pre-specified size k in place of the disjunction marked with the $\backslash syn$ tag in R. Intuitively, we are only looking for synonyms of the length up to k words (currently set to 3). Thus, if R is $(motor \mid engine \mid \backslash syn) \quad oils?$, then the following generalized regexes will be generated:

$(\backslash w+) \quad oils?$
$(\backslash w+\backslash s+\backslash w+) \quad oils?$
$(\backslash w+\backslash s+\backslash w+\backslash s+\backslash w+) \quad oils?$

We then match the generalized regexes over the given data D to extract a set of candidate synonyms. In particular, we represent each match as a tuple <*candidate synonym, prefix, suffix*> , where *candidate synonym* is the phrase that appears in place of the marked disjunction in the current match, and *prefix* and *suffix* are the text appearing before and after the candidate synonym in the product title, respectively.

For example, applying the generalized regex $(\backslash w+) \quad (jean \mid jeans)$ to the title *"big men's regular fit carpenter jeans, 2 pack value bundle"* produces the candidate synonym *carpenter*, the prefix *"big men's regular fit"*, and the suffix *"2 pack value bundle"*. We use the *prefix* and *suffix* (currently set to be 5 words before and after the candidate synonym, respectively) to define the context in which the candidate synonym is used.

The set of all extracted candidate synonyms contains the "golden synonyms", those that have been specified by the analyst in the input regex (e.g., "motor" and "engine" in the "motor oil" example). We remove such synonyms, then return the remaining set as the set of candidate synonyms. Let this set be C.

Ranking the Candidate Synonyms: Next we rank synonyms in C based on the similarities between their contexts and those of the golden synonyms, using the intuition that if a candidate synonym appears in contexts that are similar to those of the golden synonyms, then it is more likely to be a correct synonym. To do this, we use a TF/IDF weighting scheme [29]. This scheme assigns higher scores to contexts that share tokens, excepts where the tokens are very common (and thus having a low IDF score).

Specifically, given a match m, we first compute a prefix vector $\vec{P_m} = (pw_{1,m}, pw_{2,m}, ..., pw_{n,m})$, where $pw_{t,m}$ is the weight associated with prefix token t in match m, and is computed as $pw_{t,m} = tf_{t,m} * idf_t$. Here, $tf_{t,m}$ is the number of times token t occurs in the prefix of match m, and idf_t is computed as $idf_t = log(\frac{|M|}{df_t})$, where $|M|$ is the total number of matches.

Next, we normalize the prefix vector $\vec{P_m}$ into $\hat{P_m}$. We compute a normalized suffix vector $\hat{S_m}$ for match m in a similar fashion.

In the next step, for each candidate synonym $c \in C$, we compute, $\vec{M_{p,c}}$, the mean of the normalized prefix vectors of all of its matches. Similarly, we compute the mean suffix vector $\vec{M_{s,c}}$.

Next, we compute $\vec{M}_p$ and $\vec{M}_s$, the means of the normalized prefix and suffix vectors of the matches corresponding to *all* golden synonyms, respectively, in a similar fashion.

We are now in a position to compute the similarity score between each candidate synonym $c \in C$ and the golden synonyms. First we compute the prefix similarity and suffix similarity for c as: $prefix\_sim(c) = \frac{\vec{M_{p,c}} \cdot \vec{M}_p}{|\vec{M_{p,c}}||\vec{M}_p|}$, and $suffix\_sim(c) = \frac{\vec{M_{s,c}} \cdot \vec{M}_s}{|\vec{M_{s,c}}||\vec{M}_s|}$. The similarity score of c is then a linear combination of its prefix and suffix similarities:

$$score(c) = w_p * prefix\_sim(c) + w_s * suffix\_sim(c)$$

where w_p and w_s are balancing weights (currently set at 0.5).

Incorporating Analyst Feedback: Once we have ranked the candidate synonyms, we start by showing the top k candidates to the analyst (currently $k = 10$). For each candidate synonym, we also show a small set of sample product titles in which the synonym appears, to help the analyst verify. Suppose the analyst has verified l candidates as correct, then he or she will select these candidates (to be added to the disjunction in the regex), and reject the remaining $(k-l)$ candidates. We use this information to rerank the remain-

Product Type	Input Regex	Sample Synonyms Found
Area rugs	(area | \syn) rugs?	shaw, oriental, drive, novelty, braided, royal, casual, ivory, tufted, contemporary, floral
Athletic gloves	(athletic | \syn) gloves?	impact, football, training, boxing, golf, workout
Shorts	(boys? | \syn) shorts?	denim, knit, cotton blend, elastic, loose fit, classic mesh, cargo, carpenter
Abrasive wheels & discs	(abrasive | \syn) (wheels? | discs?)	flap, grinding, fiber, sanding, zirconia fiber, abrasive grinding, cutter, knot, twisted knot

Table 1: Sample regexes provided by the analyst to the tool, and synonyms found.

ing candidates (i.e., those not in the top k), then show the analyst the next top k, and so on.

Specifically, once the analyst has "labeled" the top k candidates in each iteration, we refine the contexts of the golden synonyms based on the feedback, by adjusting the weights of the tokens in the mean prefix vector $\vec{M_p}$ and the mean suffix vector $\vec{M_s}$ to take into account the labeled candidates. In particular, we use the Rocchio algorithm [28], which increases the weight of those tokens that appear in the prefixes/suffixes of correct candidates, and decreases the weight of those tokens that appear in the prefixes/suffixes of incorrect candidates. Specifically, after each iteration, we update the mean prefix and suffix vectors as follows:

$$\vec{M_p'} = \alpha * \vec{M_p} + \frac{\beta}{|C_r|}\sum_{c \in C_r} \vec{M_{p,c}} - \frac{\gamma}{|C_{nr}|}\sum_{c \in C_{nr}} \vec{M_{p,c}}$$

$$\vec{M_s'} = \alpha * \vec{M_s} + \frac{\beta}{|C_r|}\sum_{c \in C_r} \vec{M_{s,c}} - \frac{\gamma}{|C_{nr}|}\sum_{c \in C_{nr}} \vec{M_{s,c}}$$

where C_r is the set of correct candidate synonyms and C_{nr} is the set of incorrect candidate synonyms labeled by the analyst in the current iteration, and α, β and γ are prespecified balancing weights.

The analyst iterates until all candidate synonyms have been exhausted, or he or she has found sufficient synonyms. At this point the tool terminates, returning an expanded rule where the target disjunction has been expanded with all new found synonyms.

Empirical Evaluation: We have evaluated the tool using 25 input regexes randomly selected from those being worked on at the experiment time by the WalmartLabs analysts. Table 5.3 shows examples of input regexes and sample synonyms found. Out of the 25 selected regexes, the tool found synonyms for 24 regexes, within three iterations (of working with the analyst). The largest and smallest number of synonyms found are 24 and 2, respectively, with an average number of 7 per regex. The average time spent by the analyst per regex is 4 minutes, a significant reduction from hours spent in such cases. This tool has been in production at WalmartLabs since June 2014.

5.2 Generating New Rules Using Labeled Data

As discussed in Section 4, in certain cases the analysts may want to generate as many rules to classify a certain product type t as possible. This may happen for a variety of reasons. For example, suppose we have not yet managed to deploy learning methods for t, because the CS developers are not yet available. In this case, even though the analyst cannot deploy learning algorithms, he or she can start labeling some "training data" for t, or ask the crowd to label some training data, use it to generate a set of classification rules, then validate and deploy the rules.

As yet another example, perhaps learning methods have been deployed for t, but the analysts want to use the same training data (for those methods) to generate a set of rules, in the hope that after validation and possible correction, these rules can help improve the classification precision or recall, or both, for t.

At WalmartLabs we have developed a tool to help analysts generate such rules, using labeled data (i.e., $\langle$product, type$\rangle$ pairs). We now briefly describe this tool. We start by observing that the analysts often write rules of the form

$$R_4 : a_1.{*}a_2.{*}\ldots{*}\ a_n \rightarrow t,$$

which states that if a title of a product contains the word sequence $a_1a_2\ldots a_n$ (not necessarily consecutively), then the product belongs to type t.

As a result, we seek to help analysts quickly generate rules of this form, one set of rules for each product type t that occurs in the training data. To do so, we use frequent sequence mining [4] to generate rule candidates, then select only those rules that together provide good coverage and high accuracy. We now elaborate on these steps.

Generating Rule Candidates: Let D be the set of all product titles in the training data that have been labeled with type t. We say a token sequence appears in a title if the tokens in the sequence appear in that order (not necessarily consecutively) in the title (after some preprocessing such as lowercasing and removing certain stop words and characters that we have manually compiled in a dictionary). For instance, given the title *"dickies 38in. x 30in. indigo blue relaxed fit denim jeans 13-293snb 38x30"*, examples of token sequences of length two are {*dickies, jeans*}, {*fit, jeans*}, {*denim, jeans*}, and {*indigo, fit*}.

We then apply the AprioriAll algorithm in [4] to find all frequent token sequences in D, where a token sequence s is frequent if its support (i.e., the percentage of titles in D that contain s) exceeds or is equal to a minimum support threshold. We retain only token sequences of length 2-4, as our analysts indicated that based on their experience, rules that have just one token are too general, and rules that have more than four tokens are too specific. Then for each token sequence, we generate a rule in the form of Rule R_4 described earlier.

Selecting a Good Set of Rules: The above process often generates too many rules. For manageability and performance reasons, we want to select just a subset $\mathcal{S}$ of these rules. Intuitively, we want $\mathcal{S}$ to have high coverage, i.e., "touching" many product titles. At the same time, we want to retain only rules judged to have high accuracy.

Algorithm 1 Greedy($\mathcal{R}, D, q$)

```
1: $\mathcal{S} = \emptyset$
2: while $|\mathcal{S}| < q$ do
3:     $k = \max_{i \in \mathcal{R}}(|Cov(R_i, D) - Cov(\mathcal{S}, D)| * conf(R_i))$
4:     if $|Cov(\mathcal{S} \cup R_k, D)| > |Cov(\mathcal{S}, D)|$ then
5:         $\mathcal{S} = \mathcal{S} \cup R_k$
6:     else
7:         return $\mathcal{S}$
8:     end if
9:     $\mathcal{R} = \mathcal{R} - R_k$
10: end while
11: return $\mathcal{S}$
```

Algorithm 2 Greedy-Biased($\mathcal{R}, D, q$)

```
1: $\mathcal{S} = \emptyset$
2: Divide the rules in $\mathcal{R}$ into two subsets $\mathcal{R}_1$ and $\mathcal{R}_2$
3: $\mathcal{R}_1 = \{R_i \in \mathcal{R} \mid conf(R_i) \geq \alpha\}$
4: $\mathcal{R}_2 = \{R_i \in \mathcal{R} \mid conf(R_i) < \alpha\}$
5: $\mathcal{S}_1 = Greedy(\mathcal{R}_1, D, q)$
6: $\mathcal{S}_2 = \emptyset$
7: if $|\mathcal{S}_1| < q$ then
8:     $\mathcal{S}_2 = Greedy(\mathcal{R}_2, D - Cov(\mathcal{S}_1, D), q - |\mathcal{S}_1|)$
9: end if
10: $\mathcal{S} = \mathcal{S}_1 \cup \mathcal{S}_2$
11: return $\mathcal{S}$
```

Toward this goal, we start by defining a confidence score for each rule. This score is a linear combination of multiple factors, including whether the regex (of the rule) contains the product type name, the number of tokens from the product type name that appear in the regex, and the support of the rule in the training data. Intuitively, the higher this score, the more confident we are that the rule is likely to be accurate.

Given a set $\mathcal{R}$ of rules, if a product title p is "touched" by rules from $\mathcal{R}$, then we can compute $maxconf(p)$ to be the highest confidence that is associated with these rules. Then, given a pre-specified number q (currently set to 500), we seek to find a set $\mathcal{S}$ of up to q rules that maximizes the sum of $maxconf(p)$ over all product titles that $\mathcal{S}$ touches. It is not difficult to prove that this problem is NP hard. Consequently, we seek to develop a greedy algorithm to find a good set $\mathcal{S}$. A baseline greedy algorithm is shown in Algorithm 1, where $\mathcal{R}$ is the input set of rules, and $Cov(R_i, D)$ is the coverage of rule R_i on data set D, i.e., the set of all product titles "touched" by R_i. Briefly, at each step this algorithm selects the rule with the largest product of new coverage and confidence score. The algorithm stops when either q rules have been selected, or none of the remaining rules covers new titles.

A problem with the above algorithm is that rules with low confidence scores may be selected if they have wide coverage. In practice, the analysts prefer to select rules with high confidence score. As a result, we settled on the new greedy algorithm shown in Algorithm 2 (that calls the previous algorithm). In this algorithm, we divide the original set of rules $\mathcal{R}$ into $\mathcal{R}_1$ and $\mathcal{R}_2$, those with confidence scores above or below a given α threshold, referred to as "high confidence" and "low confidence" rules, respectively. We then try to select rules from $\mathcal{R}_1$ first, selecting from $\mathcal{R}_2$ only after we have exhausted the choices in $\mathcal{R}_1$. At the end, we return the set $\mathcal{S}$ of selected rules, where "low confidence" rules will receive extra scrutiny from the analysts, as detailed below.

Empirical Evaluation: We have evaluated the above rule generation method on a set of training data that consists of roughly 885K labeled products, covering 3707 types. Our method generated 874K rules after the sequential pattern mining step (using minimum support of 0.001), then 63K high-confidence rules and 37K low-confidence rules after the rule selection step (using $\alpha = 0.7$).

Next, we used a combination of crowdsourcing and analysts to estimate the precision of the entire set of high-confidence rules and low-confidence rules to be 95% and 92%, respectively. Since both of these numbers exceed or equal the required precision threshold of 92%, we added both sets of rules to the system (as a new rule-based module). The new system has been operational since June 2014, and the addition of these rules has resulted in an 18% reduction in the number of items that the system declines to classify, while maintaining precision at 92% or above. Whenever they have time, the analysts try to "clean up" further the new rule-based module, by examining low-confidence rules and removing those judged to be insufficiently accurate.

5.3 Other Rule Management Projects

In addition to the above two projects, we are working on several other projects in rule generation, evaluation, and execution, not just for classification, but also for IE and EM. We now touch on these projects (which will be described in detail in forthcoming reports).

Rule Generation: In entity matching, an analyst can write a wide variety of rules (e.g., [12, 18]). But what should be their semantics? And how should we combine them? Would it be the case that executing these rules in any order will give us the same matching result? We are currently exploring these questions. In addition, we are also developing solutions to help analysts debug and refine EM rules. In yet another project, we are examining how to help analysts quickly write dictionary-based rules for IE.

Rule Evaluation: Recently we have developed a solution to use crowdsourcing to evaluate rules [18]. When the number of rules is very large, however, crowdsourcing becomes too expensive. We are currently exploring solutions to this problem. A possible direction is to use the limited crowdsourcing budget to evaluate only the most impactful rules (i.e., those that apply to most data items). We then track all rules, and if an un-evaluated non-impactful rule becomes impactful, then we alert the analyst, who can decide whether that rule should now be evaluated.

Rule Execution: We have developed a solution to index data items so that given a classification or IE rule, we can quickly locate those data items on which the rule is likely to match. We have also developed a solution to index these rules, so that given a particular data item (e.g., product title), we can quickly locate those rules that are likely to match this item. Regarding entity matching, we are currently developing a solution that can execute a set of matching rules efficiently on a cluster of machines, over a large amount of data.

6. RULES IN OTHER TYPES OF BIG DATA INDUSTRIAL SYSTEMS

So far we have discussed rules in classification systems such as Chimera. We now briefly discuss rules in other types of Big Data systems, focusing on those we have worked with in the past few years. Our goal is to show that (1) a variety of these systems also use rules quite extensively, for many of the same reasons underlying Chimera, and (2) similar research challenges regarding rule management also arise in these types of systems. Of course, the notion of rules (e.g., syntax and semantics) can be very different depending on the system type, necessitating potentially different solutions to the same challenge.

Information Extraction: A recent paper [8] shows that 67% (27 out of 41) of the surveyed commercial IE systems use rules exclusively. The reasons listed are the declarative nature of the rules, maintainability, ability to trace and fix errors, and ease of incorporating domain knowledge.

Our experience at WalmartLabs supports these findings. Currently, we are building IE systems to extract attribute-value pairs from product descriptions. Our systems use a combination of learning techniques (e.g., CRF, structural perceptron) and rules. For example, given a product title t, a rule extracts a substring s of t as the brand name of this product (e.g., Apple, Sander, Fisher-Price) if (a) s approximately matches a string in a large given dictionary of brand names, and (b) the text surrounding s conforms to a pre-specified pattern (these patterns are observed and specified by the analysts). Another set of rules normalizes the extracted brand names (e.g., converting "IBM", "IBM Inc.", and "the Big Blue" all into "IBM Corporation"). Yet another set of rules apply regular expressions to extract weights, sizes, and colors (we found that instead of learning, it was easier to use regular expressions to capture the appearance patterns of such attributes).

Entity Matching: As another example of Big Data systems that use rules, consider entity matching (EM), the problem of finding data records that refer to the same real-world entity. Many EM systems in industry use rules extensively. For example, our current product matching systems at WalmartLabs use a combination of learning and rules, where rules can be manually created by domain analysts, CS developers, and the crowd [18]. An example of such rules is

$$[a.isbn = b.isbn] \wedge [jaccard.3g(a.title, b.title) \geq 0.8] \Rightarrow\ a \approx\ b,$$

which states that if the ISBNs match and the Jaccard similarity score between the two book titles (tokenized into 3-grams) is at least 0.8, then the two books match (two different books can still match on ISBNs). An EM system at IBM Almaden employs similar entity matching and integration rules [19].

Building Knowledge Bases: Knowledge bases (KBs) are becoming increasingly popular as a way to capture and use a lot of domain knowledge. In a recent work [27] we describe how we built Kosmix KB, the large KB at the heart of Kosmix, from Wikipedia and a set of other data sources. The KB construction pipeline uses rules extensively. As a more interesting example, once the KB has been built, analysts often examine and curate the KB, e.g., by removing an edge in the taxonomy and adding another edge. Such curating actions are not being performed directly on the KB, but rather being captured as rules (see [27]). Then the next day after the construction pipeline has been refreshed (e.g., because Wikipedia has changed), these curation rules are being applied again. Over a period of 3-4 years, analysts have written several thousands of such rules.

Entity Linking and Tagging: Given a text document (e.g., search query, tweet, news article, etc.) and a KB, a fundamental problem is to tag and link entities being mentioned in the document into those in the KB. The paper [3] describes a 10-step pipeline developed at Kosmix to perform this process. Many of these steps use rules extensively, e.g., to remove overlapping mentions (if both "Barack Obama" and "Obama" are detected, drop "Obama"), to blacklist profanities, slangs, to drop mentions that straddle sentence boundaries, and to exert editorial controls. See [3] for many examples of such rules.

Event Detection and Monitoring in Social Media: At Kosmix we built Tweetbeat, a system to detect interesting events in the Twittersphere, then displays tweets related to the events, in real time [37]. This system uses a combination of learning and rules. In particular, since it displays tweets in real time, if something goes wrong (e.g., for a particular event the system shows many unrelated tweets), the analysts needed to be able to react very quickly. To do so, the analysts use a set of rules to correct the system's performance and to scale it down (e.g., making it more conservative in deciding which tweets truly belong to an event).

7. RELATED WORK

Much work has addressed the problem of learning rules for specific tasks such as classification [10, 13, 35], information extraction [7, 9, 32], and entity matching [18]. However, as far as we can tell, no work has discussed the importance of rules and why rules are used in industrial systems, with the lone exception of [8]. That work surveys the usage of rules in commercial IE systems, and identifies a major disconnect between industry and academia, with rule-based IE dominating industry while being regarded as a dead-end technology by academia.

Classification is a fundamental problem in learning and Big Data classification has received increasing attention [25, 38, 30, 6, 33]. In terms of industrial classification systems, [30] describes the product categorization system at eBay, [6] describes LinkedIn's job title classification system, and Chimera [33] describes the product classification system at WalmartLabs. The first two works do not describe using rules, whereas the work [33] describes using both learning and rules. Entity matching is a fundamental problem in data integration, and many rule-based approaches to EM have been considered [36, 12].

Several works have addressed the problem of finding synonyms or similar phrases [17, 23]. But our problem is different in that we focus on finding "synonyms" that can be used to extend a regex. Thus the notion of synonym here is defined by the regex. Many interactive regex development tools exist (e.g., [1, 2]). But they focus on helping users interactively test a regex, rather than extending it with "synonyms". Li et al. [22] address the problem of transforming an input regex into a regex with better extraction quality. They use a greedy hill climbing search procedure that chooses at each iteration the regex with the highest

F-measure. But this approach again focuses on refining a given regex, rather than extending it with new phrases.

Building classifiers using association rules has been explored in [24, 21, 14]. Our problem however is different in that we mine frequent sequences [4, 26] then use them to generate regex-based rules. The problem of inducing regexes from positive and negative examples has been studied in the past [15, 16, 11]. But those approaches do not focus on finding an optimal abstraction level for the regexes, which may result in overfitting the training data. Further, they often generate long regexes, not necessarily at the level of tokens. In contrast, we prefer token-level rules, which is easier for analysts to understand and modify.

As for selecting an optimal set of rules, a line of work focuses on pruning certain rules during the mining process [5, 39]. Another line of work selects the rules after all candidate rules have been generated [20, 34], similar to our setting. These approaches filter rules based on their incorrect predictions on training data. We however only consider those rules that do not make any incorrect predictions on training data. Further, due to the notion of confidence score for rules, the combinatorial problem that arises from our formulation is significantly different.

8. CONCLUSIONS

In this paper we have shown that semantics intensive Big Data industrial systems often use a lot of rules. Fundamentally, we believe this is the case because to maintain high accuracy, such systems must utilize a lot of domain knowledge, in whatever way judged most efficient, using a limited team of people with varying technical expertise. As a result, the systems typically need to utilize a broad range of techniques, including rules, learning, and crowdsourcing. Rules thus are not used in isolation, but in conjunction with other techniques.

We have also shown that there is a rich set of research challenges on rule management, including many that we are currently working on at WalmartLabs and UW-Madison. As industrial systems accumulate tens of thousands of rules, the topic of rule management will become increasingly important, and deserves significantly more attention from the research community.

9. REFERENCES

[1] Regex buddy http://www.regexbuddy.com/.

[2] Regex magic http://www.regexmagic.com/.

[3] A. Gattani et al. Entity extraction, linking, classification, and tagging for social media: A Wikipedia-based approach. *PVLDB*, 6(11):1126–1137, 2013.

[4] R. Agrawal and R. Srikant. Mining sequential patterns. In *ICDE '95*.

[5] E. Baralis and P. Garza. A lazy approach to pruning classification rules. In *ICDM '02*.

[6] R. Bekkerman and M. Gavish. High-precision phrase-based document classification on a modern scale. In *KDD '11*.

[7] M. E. Califf and R. J. Mooney. Relational learning of pattern-match rules for information extraction. In *AAAI '99*.

[8] L. Chiticariu, Y. Li, and F. R. Reiss. Rule-based information extraction is dead! long live rule-based information extraction systems! In *EMNLP '13*.

[9] F. Ciravegna. Adaptive information extraction from text by rule induction and generalisation. In *IJCAI '01*.

[10] W. W. Cohen. Fast effective rule induction. In *ICML '95*.

[11] F. Denis. Learning regular languages from simple positive examples. *Mach. Learn.*, 44, 2001.

[12] A. Doan, A. Y. Halevy, and Z. G. Ives. *Principles of Data Integration*. Morgan Kaufmann, 2012.

[13] P. Domingos. The rise system: conquering without separating. In *ICTAI '94*.

[14] G. Dong, X. Zhang, L. Wong, and J. Li. Caep: Classification by aggregating emerging patterns. In *DS '99*.

[15] H. Fernau. Algorithms for learning regular expressions. In *ALT '05*.

[16] L. Firoiu, T. Oates, and P. R. Cohen. Learning regular languages from positive evidence. In *In Twentieth Annual Meeting of the Cognitive Science Society*, 1998.

[17] S. Godbole, I. Bhattacharya, A. Gupta, and A. Verma. Building re-usable dictionary repositories for real-world text mining. In *CIKM '10*.

[18] C. Gokhale, S. Das, A. Doan, J. F. Naughton, N. Rampalli, J. Shavlik, and X. Zhu. Corleone: Hands-off crowdsourcing for entity matching. In *SIGMOD '14*.

[19] M. Hernández, G. Koutrika, R. Krishnamurthy, L. Popa, and R. Wisnesky. HIL: a high-level scripting language for entity integration. In *EDBT '13*.

[20] W. L. D. IV, P. Schwarz, and E. Terzi. Finding representative association rules from large rule collections. In *SDM '09*.

[21] W. Li, J. Han, and J. Pei. Cmar: accurate and efficient classification based on multiple class-association rules. In *ICDM '01*.

[22] Y. Li, R. Krishnamurthy, S. Raghavan, S. Vaithyanathan, and H. V. Jagadish. Regular expression learning for information extraction. In *EMNLP '08*.

[23] D. Lin. Automatic retrieval and clustering of similar words. In *COLING '98*.

[24] B. Liu, W. Hsu, and Y. Ma. Integrating classification and association rule mining. In *AAAI '98*.

[25] Liu et al. Support vector machines classification with a very large-scale taxonomy. *SIGKDD Explor. Newsl.*, 2005.

[26] I. Miliaraki, K. Berberich, R. Gemulla, and S. Zoupanos. Mind the gap: Large-scale frequent sequence mining. In *SIGMOD '13*.

[27] O. Deshpande et al. Building, maintaining, and using knowledge bases: a report from the trenches. In *SIGMOD '13*.

[28] J. Rocchio. Relevance feedback in information retrieval. In *The SMART retrieval system*. Prentice Hall, 1971.

[29] G. Salton and C. Buckley. Term-weighting approaches in automatic text retrieval. *Inf. Process. Manage.*, 54, 1988.

[30] D. Shen, J.-D. Ruvini, and B. Sarwar. Large-scale item categorization for e-commerce. In *CIKM '12*.

[31] W. Shen, A. Doan, J. F. Naughton, and R. Ramakrishnan. Declarative information extraction using datalog with embedded extraction predicates. In *VLDB '2007*.

[32] S. Soderland. Learning information extraction rules for semi-structured and free text. *Mach. Learn.*, 34, 1999.

[33] C. Sun, N. Rampalli, F. Yang, and A. Doan. Chimera: Large-scale classification using machine learning, rules, and crowdsourcing. *PVLDB*, 2014.

[34] H. Toivonen, M. Klemettinen, P. Ronkainen, K. Hätönen, and H. Mannila. Pruning and grouping discovered association rules, 1995.

[35] S. M. Weiss and N. Indurkhya. *Predictive Data Mining: A Practical Guide*. Morgan Kaufmann, 1998.

[36] S. E. Whang and H. Garcia-Molina. Entity resolution with evolving rules. *Proc. VLDB Endow.*, 3, 2010.

[37] X. Chai et al. Social media analytics: The Kosmix story. *IEEE Data Eng. Bull.*, 36(3):4–12, 2013.

[38] G.-R. Xue, D. Xing, Q. Yang, and Y. Yu. Deep classification in large-scale text hierarchies. In *SIGIR '08*.

[39] X. Yin and J. Han. CPAR: Classification based on Predictive Association Rules. In *SDM '03*.