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

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

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

ABSTRACT

Big Data is now pervasive, leading to many large-scale problems such as classification, extraction (IE), and entity matching (EM). Industrial solutions to these problems very commonly use hand-crafted rules. Today, however, very little is understood about the usage of such rules. In this paper we explore this issue. We describe characteristics that set these systems apart from problems commonly considered in academia. We describe common first solutions, their limitations, and reasons for using rules. We show examples of extensive rule usage in industrial systems. Contrary to popular perceptions, we show that there is a rich set of research challenges on rule generation, evaluation, execution, and maintenance. We illustrate these challenges with ongoing work at WalmartLabs and UW-Madison. Throughout the paper, to make the discussion concrete, we focus on product classification at WalmartLabs, but touch on other types of Big Data systems as well (e.g., those on IE and EM). Our main conclusion is that using rules (together with other techniques such as learning and crowdsourcing) is fundamental to building semantics intensive Big Data systems, and that it is increasingly critical to address rule management, given the tens of thousands of rules such industrial systems often use and manage today in an ad-hoc fashion.

1. INTRODUCTION

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

Today, however, very little is understood about the usage of rules in such Big Data industrial systems. What kinds of rules do they typically use? Who write 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 in industry? How many rules are there in a typical system, and which parts of the system use them? Are we talking about 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 some rules? What else is there? 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 information extraction (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, information extraction, clustering, entity matching, vertical search, knowledge base construction, and tweet interpretation, among others. These systems often have to use a lot of domain knowledge to produce results of high quality, where the notion of quality is typically captured by precision and recall. Numerous such systems have been deployed in practice, and more are coming.

To make the discussion concrete, we start by focusing on Big Data classification systems, in particular on Chimera, a product classification system at WalmartLabs. (This system has been described in detail in [35]; here we only focus on aspects that are relevant to the topic of this paper, namely, rule usage.)

We discuss four important characteristics of the Chimera problem. First, the data is large scale, never ending, and ever changing; it arrives in batches at various time intervals. Second, Chimera must maintain high precision at all time (at 92% or higher); low recall is initially tolerated, but the system must improve recall over time. Third, since the data is ever changing and "unpredictable", system performance can quickly degrade. Thus, it is critical that we have a way to quickly debug, "scale down" the system, repair, and restore the system. Finally, the team size is limited, with a few CS developers and domain analysts, and some crowdsourcing budget. As a result, a lot of first-responder work as well as ongoing monitoring, repair, and improvement work are done by domain analysts, who have little or no CS background.

We then discuss how these characteristics set Chimera-like systems apart from those considered in academia. In brief,

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

SIGMOD '15 Melbourne, Victoria, Australia

academia typically seeks to develop *algorithms* to perform *one-shot* classification, whereas industry seeks to develop *systems* to perform *ongoing* classification, with drastically different objectives.

Next, we describe a popular learning-based solution that is commonly employed as the first solution to classification problems, and in fact, was the first solution we tried for Chimera. We discuss the limitations of this solution and the reasons for using rules. There are many. Rules are used when it is much easier to write rules than to figure out how 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 to write rules to "patch" the system behavior, until when the CS developers can diagnose and fix learning-based problems (if necessary). Analysts write rules to handle cases that learning cannot yet handle, as well as "corner cases". Rules give analysts and developers the ability to rapidly trace errors and adjust the system (e.g., scaling it down). Liability concerns may require certain predictions to be explainable, which in turn requires writing rules, and so on. Many of these reasons are fundamental and so rules will likely to be used in the foreseeable future.

In the next step, 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, entity matching systems at WalmartLabs, and knowledge base construction systems and tweet tagging and monitoring systems at Kosmix.

In the second part of the paper, we turn our attention to research challenges regarding rules. Here, contrary to the popular perception in academia, we show that there is a rich set of research challenges, regarding rule generation, evaluation, execution, and maintenance. These challenges span the entire gamut from low-level system challenges such as how to speed up rule execution on a machine cluster, to high-level user challenges, such as how to interact with domain analysts to help them write rules.

To further illustrate these challenges, we describe ongoing rule management work at WalmartLabs for product classification, information extraction, and product matching. Specifically, we describe a solution to help domain analysts refine a rule (by expanding a regular expression used by the rule), thereby reducing the time required for this step from hours to mere minutes. We describe a solution that uses labeled data to automatically generate a large set of rules, thereby significantly improving the system's recall. We also briefly describe our ongoing work in helping analysts write entity matching rules with clear semantics, evaluating rules using crowdsourcing, helping analysts refine entity matching rules using skyline techniques, indexing data and rules for faster rule execution, and efficiently executing rules on large machine clusters.

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 typically need to utilize a wide range of techniques, including learning, rules, using knowledge bases, and crowdsourcing. We believe that using rules is a fundamental technique in such cases, for reasons discussed in this paper, and must be used in conjunction with other techniques. Contrary to popular perceptions, there is a rich research agenda regarding rules. 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, addressing the above research agenda on rule management is becoming increasingly critical, and the goal of this paper is to help make a small step toward realizing this agenda.

2. BIG DATA CLASSIFICATION

As mentioned in the introduction, to make the discussion concrete, we will start by focusing on Big Data classification systems, in particular on Chimera, a product classification system at WalmartLabs.

In this section we first briefly describe the problem of Chimera (see [35] for more details). Next, we discuss characteristics that set this problem apart from those commonly considered in academia. Finally, we describe a popular learning-based solution (that was the first solution we tried for Chimera). Section 3 discusses limitations of this solution, reasons for rules, and how Chimera uses rules today. Subsequent sections build on these discussions to discuss rule usage in other types of Big Data systems.

2.1 Product Classification with Chimera

The Chimera system has been developed at WalmartLabs in the past few years to classify millions of product items into 5000+ product types [35]. 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 (because new products appear all the time) from thousands of Walmart vendors.

We maintain more than 5000 mutually exclusive *product types*, such as "laptop computers", "area rugs", "laptop bags & cases", "dining chairs", "decorative pillows", and "rings". This set of product types is constantly being revised. For the purpose of this paper, however, we will assume that this set remains unchanged; managing a changing set of types in a principled fashion 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

In building Chimera, we face 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 5000+ types. Chimera is also "never ending": it is deployed 24/7, with batches of data arriving at various 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

{ "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: 1''- 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.

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., 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, such that users can easily find this item by navigating walmart.com pages. Clearly, incorrect classification will put the item on the wrong "shelf", making it difficult to find and producing a bad customer experience.

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 performance may suddenly degrade, i.e., 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, i.e., disabling "bad parts" of the currently deployed system. For example, suppose Chimera suddenly stops classifying clothes correctly, e.g., 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 "on board" 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 today. 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 performance 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 burden of "babysitting" the system to the analysts. In addition, we would 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 seeks to develop *algorithms* to perform *one-shot* classification, whereas industry seeks to develop *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 first and foremost 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, 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 can, and should, use classification algorithms already developed in academia, but it will also have to utilize 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 Chimera-like Big Data classification systems. But first, to set the stage, we describe a popular learning-based solution that is commonly employed as the first solution in such systems (and in fact, was the first solution we tried for Chimera).

3.1 A Popular Learning-Based Solution

It is clear that at this scale we cannot manually classify all incoming products, nor write manual rules for all 5000+ 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 learning 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 will try to debug and improve the system, then apply it again to the batch. Note that the system may also decline to make predictions for certain items in the batch. In such cases, these items are sent to a team that will attempt to manually label as many items as possible, while the CS developers and analysts will 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 attempt to improve Chimera further, whenever they have time, by creating more training data and revising the 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" this solution 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, and the set of such rules can be encoded in a rule-based module, to be added to the system.

Analysts also often write rules to generate training data. For example, an analyst may write a rule to classify products as "motor oil". Next, he or she 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 to be quite 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 classification mistake, it is often very hard to find the reason. Once we have found the reason, it is still very 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 use 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. A psychologist may be able to diagnose and fix this problem later. But in the mean time, X's friends will simply apply a behavior filtering rule to ignore all stock related predictions of X.

As another analogy, suppose a car has a malfunctioning fuel light. Whenever it comes up indicating a near-empty tank, in reality there is still 1/3 of a tank, for another 80 miles. While waiting for a mechanic to diagnose and fix this problem, the driver will just apply a simple rule to ignore the fuel light, knowing that the car can still be driven for about 80 miles more.

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. Consider for instance "handbags". How do we obtain a representative sample for this product type? All the items named "satchel", "purse" and "tote" are handbags and it is hard to collect a complete list of these represen-

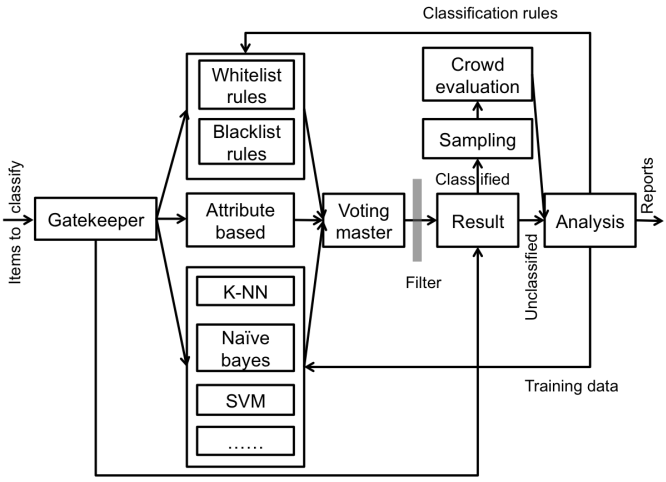

Figure 2: The Chimera system architecture.

tative items. Another example is "computer cables", which could include 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 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: Related to the above issue, a problem that we hinted at earlier is that we do not know a priori the items for 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" (for smart phone). The distribution of all product items (over the 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. Such cases are difficult to handle using machine learning. 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 are not available in the system, and it is also difficult to generate sufficient training data for these cases. As a result, we cannot reliably classify them using learning algorithms. On the other hand, it is often possible to write rules to quickly address many of these cases.

Another problem is that it is often difficult to use machine learning alone to "go the last mile". That is, we can use learning to achieve a certain precision, say 90%. But then increasing the precision from 90% to near 100% is often very 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 and Adjust the System Quickly: Recall that we want to be able 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 quickly. Then if that rule misclassifies widely, we can simply disable it with minimal impacts on the rest of the system. Thus, rules also 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 that motivates using rules, the classification results of Chimera may be curated further by people in other units downstream. Such curations can be captured using rules then add back to Chimera, to improve its performance in the future.

3.3 How Chimera Uses Rules Today

We now describe how Chimera uses rules (in conjunction with learning).

The Architecture: First, we briefly describe the overall Chimera architecture, as shown in Figure 2. 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: rule-, attribute/value-, and learning-based. There is one rule-based classifier that consists of a set of whitelist rules and blacklist rules created by the analysts (see below). There is one attribute- and 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.). The rest is a set of classifiers using learning: nearest neighbors, Naive Bayes, Perceptron, etc.

Given an item, all classifiers will make predictions (each prediction is a set of product types optionally with weights). The Voting Master and the Filter combine these predictions into a final prediction. The pair ⟨product item, final predicted product type⟩ is then added to the result set.

Next, we take a sample from the result set, and use crowdsourcing to evaluate the sample. Briefly, 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 sent back 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 and is sent to the analysts (the line labeled "unclassified" in the figure). 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: As described, the analysts can write rules for virtually all modules in the system, to improve the accuracy and exert fine-grained control. Specifically, the analysts 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 the analysts 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? → rings
diamond.*trio sets? → rings

The first rule for example states that if a product title contains "ring" or "rings", then it is of product type "rings". Thus it 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 [35]).

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. RULES IN OTHER TYPES OF BIG DATA INDUSTRIAL SYSTEMS

Besides classification systems such as Chimera, many other types of Big Data industrial systems also use rules quite extensively, for many of the same reasons underlying Chimera. We now briefly describe these system types, focusing on those we have worked with in the past few years.

Information Extraction: Consider for example information extraction (IE) systems. 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 returns 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 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 [20]. 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 [21].

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 [12] 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 [12]). 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 [18] 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 [18] 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 [15]. 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).

5. RESEARCH CHALLENGES FOR RULES

So far we have shown that many Big Data industrial systems use rules extensively, for a variety of fundamental reasons. We now show that there is a rich set of research challenges regarding rule generation, evaluation, execution, and maintenance.

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 to classify products of type "motor oil", how can we help the analyst quickly create the regular expression? As another example, when writing entity matching rules that use similarity functions, an analyst often does not know which similarity functions to start with, and why, given the large set of possible functions (e.g., edit distance, Jaccard measure, TF/IDF, etc.). It is highly desirable to have a tool that guides the analyst in this process.

The above challenge considers how to help the analyst while he or she tries to write a *single* rule. In certain cases, the analyst may want to generate as many rules as possible. For example, suppose currently there is no learning-based method yet to classify products into a type t. This can happen for a variety of reasons: there may not yet be enough training data for t; the CS developers may not yet be able to take on this task; or they have started, but it will take a while until they have managed to thoroughly train and test learning-based methods. 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 the overall system recall. A challenge then is how to help the analyst create these rules.

Yet another challenge is how to use crowdsourcing to generate rules. A recent work [20] has considered this topic, but more need to be done. Other challenges include: Should we use declarative or procedural rules? How do we combine the rules? Is it the case that we can execute the rules in any order and yet still achieve the same final result? (Considering these questions help determine what kinds of rules the analyst should write.)

Rule Evaluation: This is currently a major challenge in industry, especially given the large number of rules generated (anywhere from a few hundreds to a few tens of thousands for a system). There are two main methods to evaluate the precision of rules. The first method uses a validation set, e.g., a set of products labeled with the correct product types. Generating this validation set however is often time consuming and very expensive. Another problem is that the set can help evaluate "head" rules (i.e., rules that touch many items), but does not help evaluate "tail" rules (i.e., rules that touch a few items), because items touched by such rules are often not in the set.

The second method uses crowdsourcing (see [20] for example for a way to do it). However, evaluating the precision of tens of thousands of rules using crowdsourcing often incurs prohibitive costs. The work [20] proposes a solution that exploits the overlap in the coverage of the rules to minimize this cost. However, "tail" rules often do not overlap in the coverage. This problem remains unsolved today.

Rule Execution: Given the large number of rules generated, executing all of them often takes quite a bit of time. This is not acceptable for systems in production, where we often want the output in seconds or minutes. Thus, a major challenge here is to scale up the execution of tens of thousands to hundreds of thousands of rules. One 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. Another solution is to consider executing the rules on a cluster of machines (e.g., using Hadoop). For example, given the entity matching rule

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

mentioned earlier, how can we execute it efficiently over two tables, each consisting of 1M product items, using a Hadoop cluster?

Another interesting scenario involving rule execution is when the analyst is still developing a rule R (e.g., extending 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

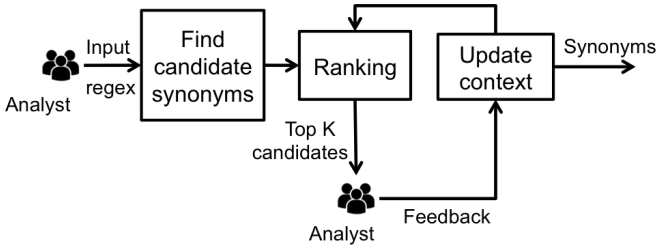

Figure 3: The architecture of the tool that supports analysts in creating rules.

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 type *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.

Finally, a major challenge is to decide how to consolidate or split rules. Ideally, we want to consolidate the rules into a smaller, easier-to-understand set. But we have found that rule consolidation often makes the work much harder for analysts. Briefly, 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 rule A or rule B. Then once the problem has been determined, it is more difficult for analysts 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.

6. ONGOING RULE MANAGEMENT WORK AT WALMARTLABS

As described, there is a rich set of research challenges regarding rule management. These challenges are becoming increasingly critical in industry, where Big Data systems are increasingly deployed, and rules are generated on a daily basis.

So far the challenges have been addressed mostly in an ad-hoc fashion. As the set of rules proliferate, these ad-hoc solutions are becoming increasingly unmanageable. Hence, it is important that we develop principled and effective solutions to rule management. In this section we briefly describe our ongoing effort toward this goal at WalmartLabs. For space reasons, we focus on rule generation for product classification, then briefly describe other ongoing efforts.

6.1 Supporting Analysts in Creating Rules

As discussed earlier, writing rules is time consuming. Hence, we want to help analysts write rules effectively. Toward this goal, at WalmartLabs we have developed a solution to help analysts quickly find "synonyms" to add to a rule under development.

Specifically, when creating rules, analysts often must write regular expressions (regexes). For example, in Chimera, to classify product items into "motor oil", the analyst may initially 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", to be added to the disjunction of the regex. Eventually, the analyst may come up with the following rule:

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

Note that the first disjunction of the regex in the above rule contains 13 terms (e.g., "motor", "engine", etc.). Clearly, finding all such terms is very time consuming for the analysts (often taking hours in our experience at WalmartLabs), not to talk about being error prone. The reason is that this is a recall oriented problem. To solve this problem, the analyst often has to painstakingly "comb" a very large set of product titles, in order to maximize recall and to avoid false positives.

Consequently, we developed a tool that helps the analyst find such terms, which we call "synonyms", in minutes instead of hours (see Figure 3). At the start, the analyst writes a short rule such as Rule R_1 described above. Next, suppose the analyst wants to expand the disjunction in R_1 with all applicable synonyms, assuming a given data set of product titles D. Then the analyst 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).

In the next step, the tool will process the given data set D to find a set of synonym candidates C. Next, it ranks these synonyms, then shows the top k candidates to the analyst. The analyst provides feedback on which candidates are correct. The tool uses the feedback to re-rank the remaining candidates (i.e., those outside the top k; for those in the top k, the analyst already decides which ones are correct and which ones are not). 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: To find the candidate synonyms we generate a set of generalized regexes G from the input regex, then match the generalized regexes over the data D to extract the candidates.

We obtain the generalized regexes from the input regex r by allowing any phrase up to a pre-specified size k in place of the disjunction marked with $\backslash syn$ tag in r. Intuitively, we are only looking for synonyms of the length up to k words. Currently we set $k = 3$. Thus, if the input regex is $(motor \mid engine \mid \backslash syn)$ $oils?$, then the following generalized regexes will be generated:

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

Next, for each title in the data D, we match each regex $g \in G$ to the title and if a match occurs, we extract a candidate synonym. We then 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, *prefix* is the text appearing before the candidate synonym in the title and *suffix* is the text appearing after the candidate synonym in the title.

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

Let C be the set of all candidate synonyms extracted as described above. This set contains the "golden synonyms", those that have been specified by the analyst in the input regex. For instance, in the "motor oil" example, "motor" and "engine" are the golden synonyms. In the next step, we return the set C', which is C with all the golden synonyms removed, as the set of candidate synonyms.

Ranking the Candidate Synonyms: Next, we rank the candidate synonyms in C', based on how similar their contexts are to the contexts of the golden synonyms. This is based on the intuition that similar phrases tend to appear in similar contexts.

Specifically, we employ the vector space model to compute the similarity between the contexts of candidate synonyms and golden synonyms. Recall that the context of a synonym is encoded using a prefix and suffix of the synonym. We now describe how to convert a prefix into a prefix vector. Converting a suffix is done similarly.

Each dimension of the vector space corresponds to a token present in the prefix of a match. We compute the weights of tokens using a TF-IDF [32] weighting scheme. In particular, we compute the prefix vector for a match m as $\vec{P_m} = (pw_{1,m}, pw_{2,m}, ..., pw_{n,m})$, 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.

For each match m, we compute a prefix vector $\vec{P_m}$ as described above, then normalize it to $\hat{P_m}$. Similarly, for each match m we also compute a normalized suffix vector $\hat{S_m}$.

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 the golden synonyms, respectively, in a similar fashion.

We are now in a position to compute the similarity score between a candidate synonym $c \in C'$ to the golden synonyms (and then use this score to rank c). We use cosine similarity for this purpose, and 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 is the weight of the prefix similarity score and w_s is the weight assigned to the suffix similarity (currently set at 0.5).

Incorporating Analyst Feedback: Once we have ranked the candidate synonyms in C', in principle we can just show these synonyms to the analyst to let him or her choose. However, we can do better by treating the analyst's actions as feedback and using this feedback to rerank the candidates.

Specifically, we start by showing the top k candidates to the analyst (currently we set $k = 10$). 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 can use this information to rerank the remaining candidates (i.e., those not in the top k), then show the analyst the next top k, and so on.

We apply the idea of relevance feedback [31] to re-rank the remaining candidates at the end of each iteration. 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 [31], 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 repeats the loop until all candidate synonyms have been exhausted, or until 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 1 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

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.

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.

6.2 Generating New Rules Using Labeled Data

As discussed in Section 5, 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 for t, use it to generate a set of classification rules for t, 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 further 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., ⟨product, type⟩ pairs). We now briefly describe this tool. We start by observing that the analysts often examine product titles then write rules of the form

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

where the a_i are words. Such a rule states that if a product title contains the word sequence $a_1 a_2 \ldots a_n$ somewhere in the title (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 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). 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 [3] 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 S of these rules. Intuitively, we want 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.

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.

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

```
S = ∅
while |S| < q do
    k = max_{i∈R}(|Cov(R_i, D) − Cov(S, D)| ∗ conf(R_i))
    if |Cov(S ∪ R_k, D)| > |Cov(S, D)| then
        S = S ∪ R_k
    else
        return S
    end if
    R = R − R_k
end while
return S
```

Given a set S of rules, if a product title p is "touched" by rules from S, 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 $S*$ of up to q rules that maximizes the sum of $maxconf(p)$ over all product titles that $S*$ 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 S. A baseline greedy algorithm is shown in Algorithm 1, where $\mathcal{R}$ is the 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.

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

```
S = ∅
Divide the rules in R into two subsets R1 and R2
R1 = {Ri ∈ R | conf(Ri) ≥ α}
R2 = {Ri ∈ R | conf(Ri) < α}
S1 = Greedy(R1, D, q)
S2 = ∅
if |S1| < q then
    S2 = Greedy(R2, D − Cov(S1, D), q − |S1|)
end if
S = S1 ∪ S2
return S
```

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 settle 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 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.

6.3 Other Projects at WalmartLabs

In addition to the above two projects, we are also working on several other projects in rule generation, evaluation, and execution, for classification, IE, and EM. Results from these projects will be described in forthcoming reports.

Rule Generation: In entity matching, an analyst can write a wide variety of rules. 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 have investigated how to use skyline techniques popular in the database literature to help pinpoint problematic tuple pairs, thereby helping the analyst refine an EM rule. 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 [20]. 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.

7. RELATED WORK

Much work has addressed the problem of learning rules for specific tasks such as classification [10, 13, 37], information extraction [7, 9, 34], and entity matching [20]. 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 [29] and Big Data classification has received increasing attention [27, 38, 33, 5, 35]. In terms of industrial classification systems, [33] describes the product categorization system at eBay, [5] describes LinkedIn's job title classification system, and Chimera [35] describes the product classification system at WalmartLabs. The first two works do not describe using rules, whereas the work [35] describes using both learning and rules.

Several works have addressed the problem of finding synonyms or similar phrases [19, 25]. But our problem is different in that we focus on finding "synonyms" that can be used to extend a regular expression (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. [24] 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 [26, 23, 14]. Our problem however is different in that we mine frequent sequences [3, 30, 28] then use them to generate regex-based rules. The problem of inducing regexes from positive and negative examples has been studied in the

past [16, 17, 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 [4, 39]. Another line of work selects the rules after all candidate rules have been generated [22, 36, 6], 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 quality, 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 learning, rules, 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. 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] R. Agrawal and R. Srikant. Mining sequential patterns. In *ICDE '95*.

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

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

[6] T. Brijs, K. Vanhoof, and G. Wets. Reducing redundancy in characteristic rule discovery by using integer programming techniques, 2000.

[7] M. E. Califf and R. J. Mooney. Relational learning of pattern-match rules for information extraction. In *AAAI '99/IAAI '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, 2000.

[12] O. Deshpande, D. S. Lamba, M. Tourn, S. Das, S. Subramaniam, A. Rajaraman, V. Harinarayan, and A. Doan. Building, maintaining, and using knowledge bases: a report from the trenches. In *SIGMOD '2013*.

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

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

[17] 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.

[18] A. Gattani, D. S. Lamba, N. Garera, M. Tiwari, X. Chai, S. Das, S. Subramaniam, A. Rajaraman, V. Harinarayan, and A. Doan. Entity extraction, linking, classification, and tagging for social media: A wikipedia-based approach. *PVLDB*, 6(11):1126–1137, 2013.

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

[20] 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*.

[21] M. Hernández, G. Koutrika, R. Krishnamurthy, L. Popa, and R. Wisnesky. HIL: a high-level scripting language for entity integration. In *EDBT '2013*.

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

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

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

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

[26] B. Liu, W. Hsu, and Y. Ma. Integrating classification and association rule mining. pages 80–86, 1998.

[27] T.-Y. Liu, Y. Yang, H. Wan, H.-J. Zeng, Z. Chen, and W.-Y. Ma. Support vector machines classification with a very large-scale taxonomy. *SIGKDD Explor. Newsl.*, 2005.

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

[29] T. M. Mitchell. *Machine Learning*. McGraw-Hill, Inc., 1997.

[30] J. Pei, J. Han, B. Mortazavi-Asl, H. Pinto, Q. Chen, U. Dayal, and M.-C. Hsu. Prefixspan,: mining sequential patterns efficiently by prefix-projected pattern growth. In *ICDE '01*.

[31] J. Rocchio. Relevance feedback in information retrieval. 1971.

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

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

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

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

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

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

[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*.