DeepQA *Jeopardy!* Gamification: A Machine-Learning Perspective

Aaron K. Baughman, Wesley Chuang, Kevin R. Dixon, Zachary Benz, and Justin Basilico

***Abstract*—DeepQA is a large-scale natural language processing (NLP) question-and-answer system that responds across a breadth of structured and unstructured data, from hundreds of analytics that are combined with over 50 models, trained through machine learning. After the 2011 historic milestone of defeating the two best human players in the *Jeopardy!* game show, the technology behind IBM Watson, DeepQA, is undergoing gamification into real-world business problems. Gamifying a business domain for Watson is a composite of functional, content, and training adaptation for nongame play. During domain gamification for medical, financial, government, or any other business, each system change affects the machine-learning process. As opposed to the original Watson *Jeopardy!*, whose class distribution of positive-to-negative labels is 1:100, in adaptation the computed training instances, question-and-answer pairs transformed into true–false labels, result in a very low positive-to-negative ratio of 1:100 000. Such initial extreme class imbalance during domain gamification poses a big challenge for the Watson machine-learning pipelines. The combination of ingested corpus sets, question-and-answer pairs, configuration settings, and NLP algorithms contribute toward the challenging data state. We propose several data engineering techniques, such as answer key vetting and expansion, source ingestion, oversampling classes, and question set modifications to increase the computed true labels. In addition, algorithm engineering, such as an implementation of the Newton–Raphson logistic regression with a regularization term, relaxes the constraints of class imbalance during training adaptation. We conclude by empirically demonstrating that data and algorithm engineering are complementary and indispensable to overcome the challenges in this first Watson gamification for real-world business problems.**

***Index Terms*—Gamification, machine learning, natural language processing (NLP), pattern recognition.**

Manuscript received January 23, 2013; revised June 19, 2013; accepted October 08, 2013. Date of publication October 17, 2013; date of current version March 13, 2014. Sandia National Laboratories is a multiprogram laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under Contract DE-AC04-94AL85000. SAND Number: 2012-10343J.

A. K. Baughman is with IBM Special Events, Research Triangle Park, NC 27703 USA (e-mail: baaron@us.ibm.com).

W. Chuang is with IBM Research Group, Chantilly, VA 22182 USA (e-mail: chuangwe@us.ibm.com).

K. R. Dixon and Z. Benz are with Sandia National Laboratories, Albuquerque, NM 87185 USA (e-mail: krdixon@sandia.gov; zobenz@sandia.gov).

J. Basilico is with Netflix, Inc., Los Gatos, CA 95032 USA (e-mail: jbasilico@netflix.com).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCIAIG.2013.2285651

I. Introduction

CAN a computational system beat a human in a task that requires complex critical thinking and higher levels of intelligence? If so, can the same system be adapted and gamified to solve real business problems? In 1958, H. A. Simon and Allen Newell said that ". . .within ten years a digital computer will be the world's chess champion. . ." and ". . . within ten years a digital computer will discover and prove an important new mathematical theorem." The two founders of AI were wrong about the date but accurate with their predictions. In 1997, Deep Blue beat Gary Kasparov, a chess grandmaster and former World Chess Champion, in chess 3.5 to 2.5 [3]. The victory for the computational system was a phenomenal step forward within computer science. Partly because the problem was a closed domain where the system was designed to search a game tree, the technology behind Deep Blue struggled to find real-world applications yet can be found in games such as *Checkers*, *Chess*, *Go*, and etc.

In 2004, IBM embarked on a radical grand challenge to progress the evolution of thinking machines to the next technology step while also maintaining business relevance for customers. A system named Watson competed on a completely open domain problem on the popular game show *Jeopardy!*. *Jeopardy!* is a television quiz show that started in 1984 and provides answers for a chosen category to three constants that then must buzz in and provide the correct question within 5 s. If a contestant is incorrect, the monetary value of the correct question is deducted from the player's cumulative winnings [11]. For Watson to compete at the highest levels on *Jeopardy!*, a combination of technologies that include open domain question and answering (QA), natural language processing (NLP), machine learning, distributed computing, probabilistic reasoning, and hypothesis generation and representation were developed over a four-year time period [11]. On February 14–16, 2011, Watson beat both Brad Rutter, the largest money winner on *Jeopardy!*, and Ken Jennings, the record holder for the longest winning streak [12].

After the historic computational victory, the world was waiting for the first commercialization of Watson. Over 34.5 million people viewed the competition and 70% of all Americans were aware of Watson. Within a week, traffic to ibm.com increased 550%. After six weeks, 116 clients requested Watson briefs, over 10 000 media stories were written, and 60 university events reached 11 000 students. In addition, Watson was included in the *Time* magazine's "100 most influential" issue and "Top Ten Man-vs.-Machine Moments." The initial nongaming application domains for Watson include the U.S. Government and healthcare.

Watson had to undergo a process of gamification through domain adaptation to be relevant for nongame domains [8]. The original Watson system was built from corpora, algorithms, and machine-learning models specific to *Jeopardy!*. The complex work of taking Watson from the stage of *Jeopardy!* to the public sector included content, functional, and training adaptation. The combination of content and training adaptation presented a unique problem for teaching Watson how to work accurately within government. This is due to the fact that an extreme number of false to correct candidate answers overwhelmed system training. Even if the question dialog was interesting, users would eventually stop using Watson since the reward for teaching the system what was right and wrong had no consequence on accuracy, recall, and precision, which the original utility of the system depends upon [13]. This paper proposes an empirical process to gamify a historical Watson system with a specific implementation of logistic regression, along with data engineering principles for Watson to better distinguish true and false candidate answers under extreme class imbalance. The paper presents to the gaming community a case of adapting complex game mechanics to real-world problems.

II. Related Work

The process of gamification leverages gaming components for nongame play [7], [8]. Within traditional gaming works, the purpose of gamification is to increase user engagement, longevity, and quality through the use of gaming technologies applied to business tasks [7], [13], [26]. Serious games are applications designed for teaching and enhancing human decisions within real-world problems. Examples include 1971 Oregon Trail and the 2013 X-Plane flight simulator for training. Extending further, the imitation game proposed by Alan Turing proposes challenges in which a machine and a human are undistinguishable in games such as *Chess* and question and answer [28]. Watson was built to provide natural language questions to answers in *Jeopardy!*, which manifested into intelligent thinking per Turing. Hayes and Ford were critiques of the Turing test and claimed the measure for AI was much too restrictive and would produce an "artificial con artist" that, in this case, could only play *Jeopardy!*. However, Watson was not just a winner of the imitation game on *Jeopardy!*, but is becoming a teacher, student, and advisor for inferring knowledge from information within real-world problems.

Gamification with domain adaptation of large complex systems breaks the traditional thought that systems are trained and tested on data drawn from identical distributions [1]. In the case of Watson, the training data used for an original problem space now have differing targets or objectives. In some systems, certain elements are domain independent and can be migrated to another problem space. For example, as long as the native language remains English, the type of speech (TOS) tagger within the English slot grammar (ESG) can be implemented in numerous problem spaces without adaptation efforts [24], [25]. By contrast, Ben-David's domain-dependent machine-learning work imposes a bound on the new domain target error and a divergence measure between two domains. Other work such as Gopalan *et al.*'s domain adaptation examines a Grassmann manifold method to shift from the source domain to the target domain where labeled data are available from the source domain but not the target domain [17]. As a comparison, while adapting Watson from *Jeopardy!* to the medical and government domains, both source and target domains contain labeled data.

During Watson domain adaptation, class imbalance occurs drastically under complex process of shifting from a general domain to a specific business domain. The class imbalance problem was investigated by Japkowicz and Stephen [19]. The work examines several data engineering methods, and reviews algorithms such as decision trees (DTs), neural networks (NNs), and support vector machines (SVMs). Japkowicz and Stephen suggest that training data for a classifier can be altered by upsizing the small class at random or with a particular focus, downsizing the large class at random or with a focus, or altering the cost of misclassifying classes.

In terms of algorithms, previous work has reviewed the effects of class imbalance on differing methods. The behavior of DTs, NNs, and SVMs is different but all suffer from the class imbalance. The SVM algorithm has proven to be more robust for a certain set of experiments [19]. However, in Watson, logistic regression ranking and classification was empirically superior to DTs, NNs, and SVMs when applied to *Jeopardy!* questions. The work discussed below examines regression techniques within the context of class imbalance.

Komarek's dissertation gives a thorough treatment from linear regression to logistic regression [21]. The work introduced the conjugate gradient (CG) algorithm, as a comparison to steepest decent. The thesis then furthered logistic regression and its role in classification, in which several parameter estimation methods are compared, including iteratively reweighted least squares (IRLS), conjugate gradient–maximum likelihood estimation (CG–MLE), and Broyden–Fletcher–Goldfarb–Shanno maximum likelihood estimation (BFGS–MLE), where a Newton-like method approximates the Hessian matrix [21]. King and Zeng examine the limited performance of an unregularized logistic regression algorithm on rare event data [20].

Other work has investigated the impact of adding a regularization term to logistic regression to mitigate an extreme class imbalance problem [4]. Lee *et al.*'s efficient L1 regularized logistic regression presented an iterative quadratic approximation to the objective function and used the least angle regression (LARS) algorithm that guarantees convergence for solving the formulated optimization problem [23]. Friedman *et al.* [15] and Wright *et al.* [30] proposed a regularization method called least absolute shrinkage and selection operator (LASSO). The method minimizes the residual sum of squares, with increasing penalty that shrinks more parameters to zero. Consequently, it produces interpretable models such as subset selection and exhibits the stability of ridge regression. Zhang and Oles compared a number of text classification methods, including linear least squares fit (LLSF), logistic regression, and SVM [31]. The paper empirically and statistically discussed their similarity in finding hyperplanes for separating the document class.

There are a wide variety of machine-learning software implementations of logistic regression. The GNU project R provides statistical and graphical tools to assist in statistical analysis.

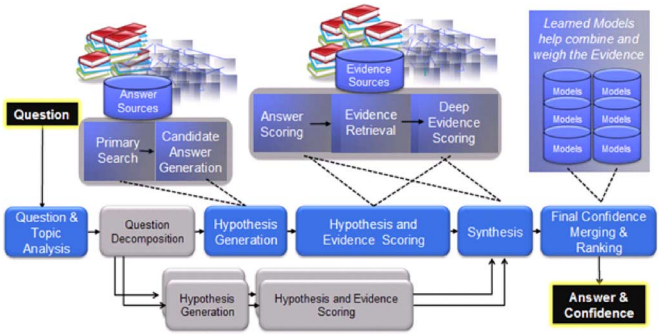

Fig. 1. Watson's massively parallel evidence-based architecture [9], [10].

R is an interpreted language, written primarily in C and Fortran that allows interfaces to machine-learning packages such as Weka. The statistical package SPSS, originally designed for social sciences, is now able to perform a broad range of machine-learning tasks. Popular algorithms such as logistic regression, for example, can be used in SPSS. LIBLINEAR is a C implementation of several linear methods, including L2-regularized logistic regression. Weka, implemented in Java, contains implementations of many machine-learning algorithms such as logistic regression. Apache's Mahout is a machine-learning library with the purpose to build scalable learning systems. Implemented on top of Apache Hadoop using the map-reduce paradigm, Mahout can naturally handle large data sets. Basilico *et al.* developed the cognitive foundry (CF), a collection of components and algorithms for building intelligent systems with an emphasis on making it easy to integrate into real-world systems, such as Watson. Since machine learning is a key technique for creating such systems, CF provides a rich set of interfaces and components to facilitate rapid construction and adaptation of machine-learning algorithms. After logistic regression became the algorithm of choice for Watson, CF was chosen for all experiments within the paper since the package is open source and is easily extensible for the addition of a regularized Newton–Raphson logistic regression algorithm.

III. Watson *Jeopardy!*

Open question and answering is a challenging and important problem within computer science. Its goal is to provide an NLP human interface to precisely answer questions that are justified by supporting evidence. Watson was built with hundreds of analytical techniques and a hundred machine-learning models to parse, understand, and answer a question written in human English language. Fig. 1 depicts Watson's overall system architecture.

As a question is submitted to Watson, question and topic analysis first determine the lexical answer type (LAT), focus, question type, and predicate argument structure or a logic construct [12], [24]. LAT is a word or noun phrase that indicates the type of answer. The focus is the part of the question that contains useful information about the answer. For example, "Secretary Chase just submitted this to me for the third time; guess what, pal. This time I'm accepting it" has a focus of "this" and LAT of "thing" [11]. Further, each question is typed into a question category such as number, bond, etymology, etc. [22]. The natural language pipeline is then applied to the question, which includes the English slot grammar parsing, predicate argument structuring, rule-based named entity detection, intraparagraph anaphora resolution, temporal normalization, temporal arithmetic, pattern-based semantic relation detection, and statistical semantic relation detection [5], [10]–[12], [24]. Next, the extracted features are used for question decomposition for the primary search module. At this stage, Lucene and Indri, a document belief network, and other derived resources are queried to create hypotheses [5]. Each candidate answer or hypothesis is scored and filtered with a logistic regression machine-learning model that ranks the most plausible answers by merging over 400 features such as Indri and Lucene query rank.

The candidate answers that pass the primary search continue through the analytic pipeline for deep evidence chaining and final merging. Hundreds of algorithms are applied to the potential answers to accumulate scores that measure NLP relationships [27]. The large matrix of individual scores is then standardized, combined, and synthesized [16]. The score matrices and candidate answers are combined in the final merger and confidence-ranking framework. Over 50 models produce layers of successive refinement to individually score and rank each candidate answer [16]. For *Jeopardy!*, if the confidence of the first ranked answer was above a learned threshold, Watson would buzz in and provide an answer or question, in terms of the game show. Otherwise, if the risk-to-reward ratio was too high, Watson would not attempt an answer. Over time, the system learned per-category confidence thresholds to determine when it was acceptable to buzz in [11].

A. Watson Jeopardy! Experiments

Watson experiments are composed of corpora, experimental artifacts, code, and a configuration set. The corpora include a plurality of data from ingested raw sources that have been rearranged into a text retrieval conference–deep question-and-answer (TREC–DeepQA) format [6]. This format allows data fields for the creation of title-oriented documents. All the ingested sources are candidates for long document search, short document search, title in clue (TIC), passage in clue (PIC), and passage search [5]. Both Indri and Lucene technologies are used to provide the complementary unstructured search strategies. Example ingested data sources include Harper Collins Dictionary, Oxford Law, Oxford World History, and Britannica Encyclopedia. A Wikimedia set that includes Wikipedia, DBpedia, Wiktionary, and Wikiquote contributes toward the generation of candidate answers and potential hypotheses. Wikipedia is a central source as depicted in [5]. Other types of data sources include Prismatic that represents extracted semantics, analytic specific information, Yago, Wordnet, Verbnet, Geospatial information, and several lexicon definitions.

Natural language questions are encoded within a data structure that Watson can easily parse and accumulate evidence over hundreds of algorithms. The questions can have a surface form of who, what, when, where, and how or *Jeopardy!* answers. Each question has a source that dictates the type of candidate generation strategy. For example, TREC-sourced questions

have an additional candidate answer generation search called PIC as well as additional type coercion systems. Each question is put into a common analysis structure (CAS) for the aggregation of feature extraction, scoring, and evidence retrieval storage. A CAS is the unstructured information management architecture (UIMA) transport object to carry data through pipelines [9]–[11]. Analytic pipelines define components of the system that include question decomposition, primary search, candidate generation, filtering, deep evidence retrieval, final merging and ranking, and display to the user. UIMA provides an architecture to stream data through analytic pipelines. Specific analytic flows can be combined to create any combination of algorithms or pipelines in an object-oriented manner.

The question sets are split into train, test, and validation sets. The stratifications of questions are selected using a uniform random sampling method. The training set is only run through the system during training mode within the customization environment [9]. Several training sets can be defined for each specific machine-learning model. A test set produces a test error metric that can be compared to the training error. Overfitting can be avoided by ensuring that the test set error is equal to or less than the training set error. The third question set is utilized to prevent bias of algorithms over known questions within the test and training set. An answer key to a question is the ground truth for the system to produce exemplars that are used in training as well as in evaluating information retrieval metrics. Each answer can be in the form of a literal string or a regular expression designed to match potentially correct answers returned by Watson.

A selected version of a Watson code base is used to deploy UIMA pipelines and algorithms to remote machines. When running in production mode, Watson is designed to answer a question quickly, since speed is important [9]. In parallel, multiple questions, in CAS format, are submitted to remote machines for analytic processing. The second mode called customization supports the launching of many simultaneous experiments on remote machines.

The code, corpora, CAS set, question identifiers (QIDs), and answer keys are controlled within an experiment by a configuration set. The separation of each element enables flexible design of experiments. For example, an answer key can be updated without the need to change the corpora. The separation of a configuration set and code base maintains independence between algorithms and other experimental artifacts. In doing so, Watson is adequately supporting a metric-driven approach suitable for domain adaptation.

B. Watson Error Analysis

Standard metrics, precision, recall, and accuracy, from the field of information retrieval are used to measure the performance of the question-and-answer system. Precision and recall in traditional information retrieval are defined by

$$pr = \frac{|\{\text{RelevantDocs}\} \cap \{\text{RetrievedDocs}\}|}{|\{\text{RetrievedDocs}\}|} \tag{1}$$

$$\text{recall} = \frac{|\{\text{RelevantDocs}\} \cap \{\text{RetrievedDocs}\}|}{|\{\text{RelevantDocs}\}|} \tag{2}$$

Written to reflect the binary classification class, precision and recall become

$$pr = \frac{TP}{TP + FP} \tag{3}$$

$$\text{recall} = \frac{TP}{TP + FN} \tag{4}$$

and binary accuracy yields

$$ac = \frac{TP + TN}{FP + TP + FN + TN} \tag{5}$$

where the true positive number is $TP \in [0, N_{tp}]$, the false positive number is $FP \in [0, M_{fp}]$, the true negative number is $TN \in [0, N_{tn}]$, and the false negative number is $FN \in [0, M_{fn}]$. The set of all exemplars for training Watson is defined in

$$M_{fp} + M_{fn} + N_{tn} + N_{tp} = |E_s|. \tag{6}$$

By default, (3) is taking all retrieved answers into consideration denoted as pr@100%. The measure can be evaluated at a given cutoff percentile such as pr@N%. Watson implements the pr@N% notation, the highest N% of confident answers are taken into account, where $0 \leq N \leq 100$ [11], [16].

Watson returns a list of possible answers ranked by estimated confidence levels [11], [16]. During experimentation, analysis, or the *Jeopardy!* game, only the highest ranked answer is considered correct. In some cases, the confidence for the highest candidate answer might not justify the risk of a response. With the cutoff rank, the system asserts a compromise between a broad list of candidate answers and their confidence level.

Recall has a distinct meaning in Watson. The notion of recall@M is taken to indicate recall at the highest M ranked answers, where $0 \leq M \leq |E_s|$ [11], [16]. Watson uses this metric to broaden the range of accepting a correct answer. For example, if the correct answer is within the top 10, the question will be considered correct during experimentation.

With the above interpretation, the accuracy would be equal to precision when only the top-rank answer over all questions or pr@100% is considered, as shown in (6) [11]. This is due to the fact that true negative and false negative values can be ignored in this special case

$$ac = \frac{TP}{TP + FP} = pr@100\%. \tag{7}$$

Accuracy and precision would differ, if only the @70% is imposed on precision; that is, $ac \neq pr$ @70% [9].

Adapting metrics in such a manner provides a flexible way to manage the tradeoff between recall and precision.

C. Machine Learning in Watson

Watson computes a confidence value and rank for each answer by combining collections of positive, negative, or neutral supporting analytical scores. Determining how to combine each score would be a daunting task for a human due to the volume, variety, and velocity of algorithm change. Instead, such a task is carried out by Watson's supervised machine-learning techniques, in which human experts' question-and-answer pairs are used to generate training data and to produce models. Each

Hit List Normalization	An initial ranking of all answers
Base	Filter to top 100 and re-standardize
Transfer Learning	Improve the ranking for special question classes that have limited training data – adds a few features
Answer Merging	Merge equivalent or nearly eq. answers
Association	Captures the degree of association of the answer to the question
Swap	Transfer features from one answer to another
Elite	Filter to top 5 answers
Multi Answer	When question asks for 2 or more answers, combine the answers together
Apply	Performs final adjustments to the answer confidence

Fig. 2. Original Watson *Jeopardy!* machine-learning phases [13].

model determines the likelihood that a candidate answer is correct [16].

The Watson machine-learning framework supports an agile environment where thousands of experiments can be run over differing feature sets, algorithms, and corpora [11], [12], [16]. Given the increasing set of domains that Watson is used in, the phase-based architecture enables successive refinement, transfer learning, elitism, standardization, and evidence diffusion that may change between application domains [16]. Each phase has three fundamental steps that include evidence merging, postprocessing, and classifier training/application [16]. Fig. 2 depicts the *Jeopardy!* system machine-learning phases.

Each question has a particular type that may be more highly correlated with specific features than others. In total, the *Jeopardy!* system has 12 specific question classes (QClass) and a default category that can be defined as a route [22]. The QClasses include definition, category relation, fill in the blank, abbreviation, puzzle, etymology, verb, translation, number, bond, multiple choice, date, and default for all other questions [22]. The training data files are stored within the attribute-relation file format (ARFF), which contains feature metadata and values [16]. For example, the elite phase has 579 total features while the hit list normalization maintains 546 features. The features were standardized into traditional z-scores where each feature value is subtracted from the feature mean and divided by the standard deviation [13]. Both standardized and original scores are retained for classification since experiments have indicated that the combination outperforms either one alone. If an input algorithm fails or a feature is missing, an imputed value is used so that the feature does not contribute to the overall answer ranking. Correlation of a value with a target answer is used for feature selection. In addition, a sparse feature filter is added during training to further avoid overfitting if there were too few values for a feature.

Watson uses logistic regression as the machine-learning algorithm. In Section IV, we will provide more detail with how the original algorithm is "engineered" to conquer the challenges we face during domain adaptation.

IV. Watson Gamification

Watson was designed for *Jeopardy!* play with specific game strategies. During play, the system had to decide if it should buzz in, which space on the board to select, and how much to wager for daily doubles and Final *Jeopardy!*. Over 3000 past *Jeopardy!* matches were mined to create average, champion, and grand champion player models [29]. The models were simulated as opponents of Watson to learn game-play strategies. Certain types of questions such as daily doubles were orders of magnitude more important to answer than the other types of questions. During game play, Watson's strategy with respect to performance and question categories changes [29].

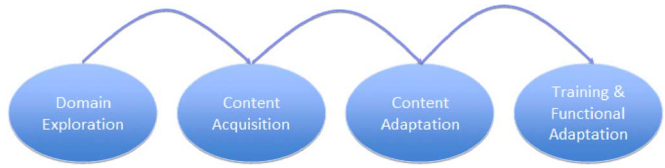

Fig. 3. Domain adaptation process.

Within nongame contexts such as the government and medical domains, understanding how Watson determines an answer is more important than the answer itself. The confidence attached to the answer is an indicator of underlying evidence. Watson game-play mechanics are altered to determine which evidence and answers to show within a business environment. Further, to replace pregame time simulations, large volumes of labeled data from business domains are used to teach Watson how to respond to questions.

In relation to passing Turing's imitation game, it is important for Watson to engage and elevate the importance of the system within business domains. To achieve high levels of accuracy, as defined in this paper, a complex process of domain adaptation modifies Watson's core game-play machine-learning mechanics to solve real-world problems. Fig. 3 depicts the steps for domain adaptation. Outside the scope of the paper, future domain adaptation will include functional adaptation.

An initial domain exploration phase seeks to discover key system usage patterns. Users working on problems for their business provide input such as operational visions. For example, within the healthcare domain, Watson is adapted to the role of a medical assistant, while in government, the system is an evidence-based analyst. Watson can also be trained to become an investment analyst by finding trends among financial records and numbers. To perform any adaptation, candidate corpus sets, questions, use cases, and user interfaces must be developed. To get started, an initial set of corpus is acquired to augment the base Watson system. Adapting the base Watson is an experimental process that consists of three primary steps.

1) Content adaptation: Evolving a corpus into a particular domain by supporting hypothesis generation, evidence scoring, and data retrieval. This includes, through error analysis of experiments, sets of data to be added, removed, or changed by corpus expansion or data preprocessing.
2) Training adaptation: Teaching Watson from a domain-specific training question set and evaluating the system on a test set. Over machine-learning epochs, Watson learns how to combine score components and rank candidate answers based on a confidence score.
3) Functional adaptation: Altering Watson's capability to read and understand an input question, including the addition of domain-specific algorithms to support question analysis, hypothesis scoring, and evidence generation.

Clearly, the process of domain adaptation causes a cascade of changes. For example, when content is added into Watson, the machine-learning models need to learn to combine component-level scores. In addition, content-specific features need to be created from the new data, which requires additional analytics to derive such features. Functional adaptation requires training with new feature values out of corpus expansion to produce new models, and to support the new analytic pipelines [6]. Thus, gamification of Watson becomes a continuous improvement process to optimize system performance such as accuracy, precision, and recall.

A. Class Imbalance

During the process of gamification, the overwhelming majority of generated question labels are false due to missing data, ill-formed training questions, incorrect feature merging, answer key errors, or missing domain features. For example, during gamification, Watson does not understand the question "What country uses the most renewable energy?" since *Jeopardy!* provides the answers, not questions and lacks the correct weighting to combine evidence about the United States. The outnumbering of false to true labels creates an extreme class imbalance, which causes system accuracy to plummet and miserably fail the imitation game.

B. Data Engineering: Extreme Class Imbalance

The overall question set used in this paper contained 5000 unique natural language questions. An evenly weighted random sample without replacement selected a 75% train, 20% test, and 5% validation split. During training, each train question is sent through the question analysis (QA) pipeline and candidate generation to create training instances, also called exemplars, with large feature vectors with over 400 dimensions. Each instance is compared to the answer to generate a label.

Before domain adaptation, an average of 190 instances are created per question contained within the training set that generated one true label for every 100 false labels. Therefore, the ratio of true-to-false labels R_{tf} was 0.01, where

$$R_{tf} = \frac{|T_{\text{labels}}|}{|F_{\text{labels}}|}. \tag{8}$$

In adaptation, while processing Wikipedia, the corpora generated an R_{tf} with less than one true label for every 300 false labels. Oversampling true labels was employed before training to artificially increase the R_{tf} value. For example, after a $5\times$ oversampling on true labels in the feature hit list phase, the ratio approaches one true for every 100 false labels, $R_{tf} = 7025/76\,206 = 0.009$. However, when sources such as Oxford were ingested into the Watson corpora and new domain-specific questions were added to the question set, the R_{tf} ratio decreased. In this case, the feature hit list phase training data produced 104 true labels and 721 077 false labels yielding $R_{tf} = 0.00014$, meaning approximately for each true label, 10 000 false labels were produced. The low R_{tf} metric became a challenging problem for modeling algorithms that are not specifically designed to handle such extreme class imbalance.

Following data engineering practices, oversampling was implemented to increase the true-to-false ratio to 1:100. Several other methods were used to increase the R_{ut} value, as defined in (10), such as answer key vetting. Domain experts semiautomatically examined the correctness of the answer key. If a candidate answer was marked as wrong yet had a high confidence value of correctness, the answer key was likely incorrect. For example, if the question "Who was the first president of the United States?" resulted in G. Washington and not George Washington, the label would have been incorrectly marked false. Answer keys were vetted with experts in particular domains to increase the correctness in human annotations.

Further, the answer was expanded. The answer key was converted from a literal form to regular expressions such that more potential correct answers were found. Regular expressions increased the space as to which an answer could be correct. Other techniques included question set modification; that is, changing the surface form of the question so that Watson could understand the structure of the sentence.

Even so, when machine-learning libraries used the new boosted training set, they either failed to produce models or generated poor models as judged by the confusion matrix. Following (10), a second metric or indicator of a poor training set R_{ut} was defined to determine the ratio of the unique number of questions that have true labels versus the total number of questions. For example, if only a few questions generate instances with true labels, the majority of questions will have false instances, even though the R_{tf} is high. Table V in Section V-C shows corresponding R_{tf} and R_{ut} values.

Before the introduction of algorithm engineering, the maximum accepted class imbalance for an algorithm is defined in

$$\min\{R_{tf}\} = \frac{\left|\sum_{n}^{N} C_n\right|}{\left|\sum_{m}^{M} C_m\right|} \tag{9}$$

as the minimum ratio of C_n or the sum of all true labels to C_m or the sum of all false labels. Further, the second data engineering metric used during gamification includes the minimum ratio of total number of questions that has a true label Q_t over all questions found in

$$\min\{R_{ut}\} = \frac{\left|\sum_{t}^{T} Q_t\right|}{|Q|}. \tag{10}$$

If $\min\{R_{tf}\}$ or $\min\{R_{ut}\}$ is too low for an algorithm, the data must be engineered to compensate for the lack of a regularization metric to be described in Section IV-F. As the corpora, question sets, and answer key change over time, the system will provide new R_{ut} and R_{tf} for indications as to how best to change the data state. Because data engineering alone does not completely eliminate the challenging data state, we turn to algorithm engineering to mitigate the extreme class imbalance.

C. Logistic Regression

In previous sections, we briefly mentioned that Watson's machine-learning algorithm belongs to binary classification,

specifically, logistic regression. The probability density function (pdf) is a commonly used distribution for learning a linear classifier [2], [14]. Logistic regression learns a model of the form

$$f(x) = \frac{1}{1 + e^{-(wx+b)}} \tag{11}$$

where w is the weight vector and b is the bias term [2], [17]. Conceptually, it is using a linear model and passing it through a logistic link function to produce an output value between 0 and 1. In this way, logistic regression can be used to model a probability function by treating $P(y = 1|x) = f(x)$ and $P(y = 0|x) = 1 - f(x)$.

Under the probabilistic interpretation, to fit the logistic regression model based on a set of observed data, we seek to optimize the likelihood of the data

$$\prod_{i=1}^{n} f(x_i)^{y_i} \left(1 - f(x_i)\right)^{1-y_i}. \tag{12}$$

Since directly optimizing this function is difficult, instead the common approach is to minimize the negative log-likelihood. This leads to the following objective [2]:

$$E(w) = -\sum_{i=1}^{n} y_i \log f(x_i) + (1 - y_i) \log\left(1 - f(x_i)\right). \tag{13}$$

There are several approaches to minimizing this objective, including those described in Sections IV-E and IV-F, which are based on using the gradient

$$\nabla E(w) = \sum_{i=1}^{n} \left(y_i - f(x_i)\right) x_i. \tag{14}$$

Since this gradient is straightforward to compute, it leads to several optimization techniques.

D. Iterative Reweighted Least Squares

To solve the optimization problem presented in (14), the iterative reweighted least squares (IRLS) algorithm provides a sequence of steps that can easily be implemented within programming logic. The method finds the maximum likelihood estimate for the objective method, shown in (13). By extension, the form

$$P(C_1|x) = \frac{P(x|C_1)P(C_1)}{P(x|C_1)P(C_1) + P(x|C_2)P(C_2)} \tag{15}$$

is the probabilistic interpretation of the problem where C_1 is the true class and C_2 is the false class. The likelihood $P(x|C_1)$ is equal to (12).

Within IRLS, the likelihood probability is defined by $P(x|C_1)$ and extended to an error gradient as depicted by (14). The error gradient can be extended to the Hessian matrix to support the Newton–Raphson implementation of IRLS. At each step, the likelihood probability is increased when measured against an objective function. Equation (16) depicts iterative beta weight updating for logistic regression implemented with IRLS, with a stochastic gradient-descent error gradient.

TABLE I
CONFUSION MATRIX FROM LOGISTIC REGRESSION TRAINED WITH FILTER MODEL INSTANCES USING SGD

		Actual Class	
Predicted Class		True Positive 0.0	False Positive 0.0
		False Negative 1247.0	True Negative 197316.0

E. Algorithm Engineering: Stochastic Gradient Descent

A learning technique that can be used with logistic regression is stochastic gradient descent (SGD) [2]. It is an extension of gradient descent, which optimizes an objective function by taking steps along the error gradient to minimize the error between targets and predictions. Because gradient descent can be slow to converge and can be hard to scale to large data, SGD is typically performed when the gradient is computed either from a single example or a small batch, where a small step is performed. This is repeated until convergence. The form

$$w^{(\text{new})} = w^{(\text{old})} - \eta \nabla E(w) \tag{16}$$

updates the weight value based on a learning rate η and gradient $(d/dw)E(w)$ [2], [18].

After an experiment with the Watson filter model, we found SGD was not effective for the extreme class imbalance. Table I shows a typical confusion matrix obtained when using SGD within Watson for domain adaptation. The classification produced by SGD is the trivial solution of only negative predictions, which has a high accuracy and low error for one type of predicted class. The algorithm converged too quickly while not considering the smaller number of true-labeled exemplars.

F. Algorithm Engineering: Newton–Raphson Iterative Optimization

We also used the standard Newton–Raphson parameter-optimization formulation to minimize the negative log-likelihood. When applied to logistic regression, the algorithm is typically known as iterative reweighted least squares, as described in Section IV-C. In its general form, it performs updates of the form

$$w^{(\text{new})} = w^{(\text{old})} - H^{-1} \nabla E(w) \tag{17}$$

where w is the weight vector, $E(w)$ is the vector of error terms for the current weights, and H is the second-order derivative or Hessian matrix of the estimate errors with respect to the current weights [2]. Substituting into the definition of logistic regression, we get a restatement of (14) that progresses to the Hessian matrix

$$\nabla E(w) = \sum_{n=1}^{N} (w^T \phi_n - t_n)\phi_n \tag{18}$$

$$H = \nabla \nabla E(w) = \sum_{n=1}^{N} \phi_n \phi_n^T. \tag{19}$$

TABLE II
EXPERIMENT RESULTS DURING DOMAIN ADAPTATION. THE VALUES BETWEEN 1.0E-10 AND 10 WERE OMITTED TO DEPICT THE PEAK OF λ AT 10

λ	Answers Found	Accuracy	Recall @ 10	Precision @ 70%
0	47.28%	1.58%	8.61%	1.69%
1.0E-10	46.49%	24.43%	38.08%	33.47%
10	47.58%	27.99%	40.75%	37.85%
100	47.18%	24.43%	36.99%	33.61%

By contrast, the Newton–Raphson method updates weights from (17) that includes the Hessian matrix of the error function. The second derivative function omitted by SGD causes the algorithm to ignore the rate of change of the gradient. In the event of extreme class imbalance, such omission has been empirically shown in Table I to overfit the majority class. As such, the Newton–Raphson method was chosen for Watson domain adaptation.

G. Algorithm Engineering: Regularization

Through the view of machine learning, the journey of domain adaptation culminated when we added the implementation of regularization to the Newton–Raphson algorithm. The system began recovering from extreme class imbalance without any data engineering.

The error gradient of logistic regression minimizes only the data-dependent error. In such a situation, coefficients for some features can become very large and overfit the data, especially under skew [2]. To discourage overfitting the training set, a regularization term is added to the optimization problem that penalizes high feature weights. The following equation:

$$E(w) = E_D(w) + \lambda E_w(w) \tag{20}$$

depicts the required error function for Watson where $E_D(w)$ is the data error from (19) and $E_w(w)$ is the weight error found as the second term in

$$E(w) = E_D(w) + \frac{\lambda}{2}\sum_{j=1}^{M} |w_j|^q. \tag{21}$$

The hyperparameter λ variable controls the relative importance of the data error [2]. Table II depicts experimental results from the alteration of λ.

Generally, one can use different regularization terms, with common ones from an L^p space. For Watson, we found that L^2 regularization provided a Hilbert space that helps prevent overfitting false labels. These findings are in line with previous work discussed in Section II that found L^2 to stabilize coefficients [4], [15], [23], [30], [31]. The following equation

$$L^2(w) = \frac{\lambda}{2}\left(|w_1|^2 + \cdots + |w_n|^2\right), \qquad \lambda = 1 \tag{22}$$

depicts the L^2 formula. Working with logistic regression, this form of regularization can also be seen as placing a Gaussian prior on each of the weights.

V. DEEPQA EXPERIMENTS

A. Design

To prepare for empirical analysis during domain adaptation, the experimental runs include model training and application. Each experiment includes training dozens of models, generating training data, calculating thresholds, NLP pipeline processing, and incremental model testing. Each experiment is distributed across several thousand IBM Power 750 cores to optimize parallelization. Logistic regression was run on a single core due to the serial nature of the learning algorithm. The time to complete a full run with $\sim$5000 domain questions can range from 18 hours to several days.

The system can be broken down into data, code, and configuration settings that allow control of experimental variables. Although out of the scope for this paper, functional adaptations produce code variables such as the addition of analytics to the question analysis pipeline. Answer key and question set modification serve as data variables, whereas scale-out configurations are configuration variables. Our first set of experiments was to analyze the results from different regularization parameters' λ values, with $\lambda = 0$ to turn off regularization. Next, data engineering experiments were run to explore the effect of modifying question sets and answer keys. Finally, experiments were executed with both regularized logistic regression and data engineering variables.

B. Algorithm Engineering Results

From (21), the original logistic regression was modified to include an L^2 regularization term, where the relative contribution of the regularization term is controlled by the value of λ. The overall results of several experimental runs can be viewed in Table II. We tried several values for λ between 0 (no regularization) and 100. Further, in Table II, the column "answers found" depicts the overall percentage of questions that had a correct answer within the candidate answer list. If an answer was not found, either the answer key or the ground truth should be modified or the system had a knowledge gap.

The line plots in this section show the relation of the percentage of questions answered versus accuracy of questions answered versus confidence. When λ is zero, using only the error term in the optimization, the accuracy becomes nearly zero with most questions ranked above 50, as depicted in Figs. 4 and 5. The confidence values from the pdf are all extremely low, even when a low percentage of questions are answered.

In stark contrast, for any $\lambda > 0$, the accuracy of the system improves, as shown in Figs. 6 and 8 and Table II. In Fig. 6, the contour of the line dips when the highest ranked 4% or less ranked questions are answered. Watson is overconfident about questions the system computes that it knows. The curve indicates that the regularization factor is not yet high enough to account for low occurring exemplar patterns. The rank position histogram shown in Fig. 7 depicts an almost evenly distributed rank position with rank 1 containing 24.43% of the correct answers.

As the regularization term increases, Watson begins to have higher confidence in answers found in rank 1. Figs. 8 and 9 de-

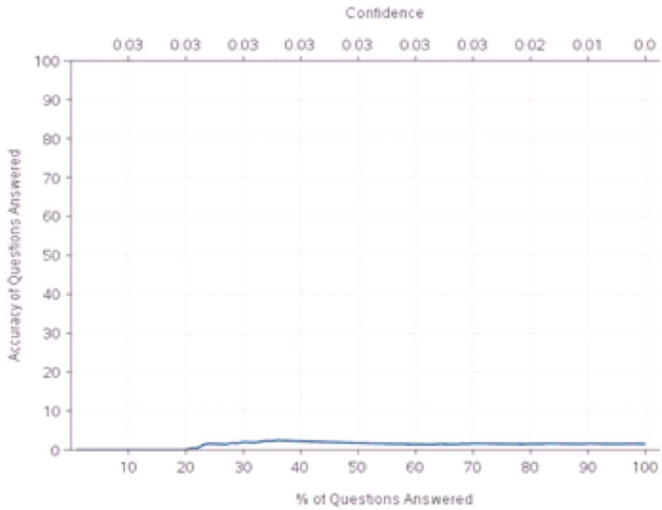

Fig. 4. Accuracy versus percentage of questions answered with lambda 0.

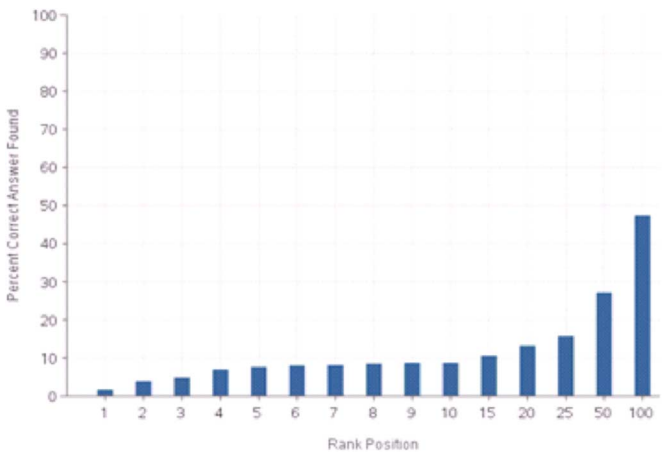

Fig. 5. Rank histogram for lambda 0.

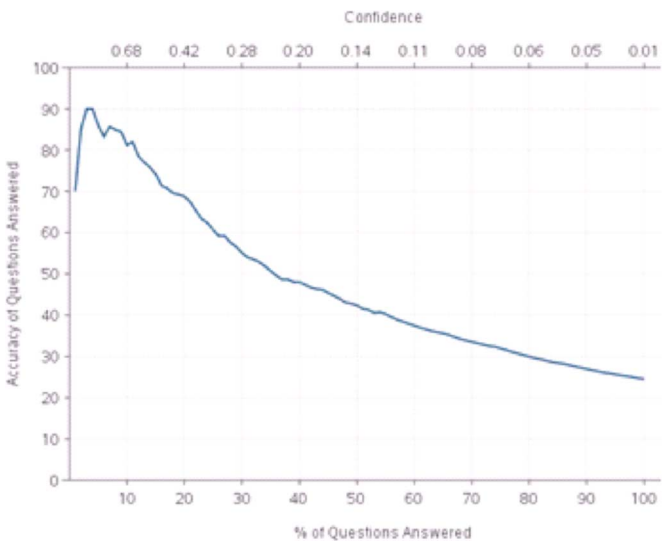

Fig. 6. Accuracy versus percentage of questions answered with lambda 1.0E-10.

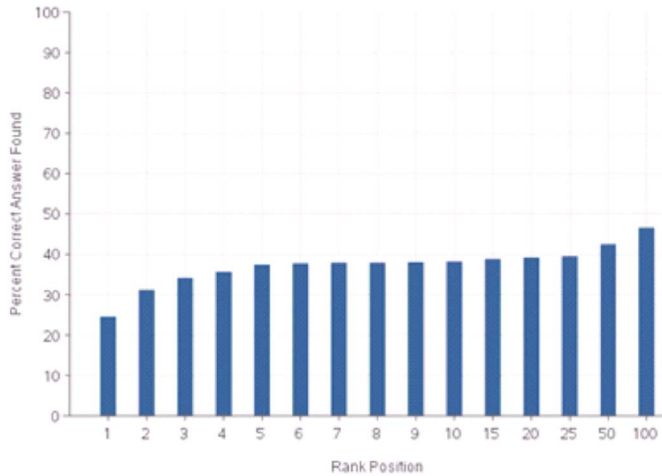

Fig. 7. Rank histogram for lambda 1.0E-10.

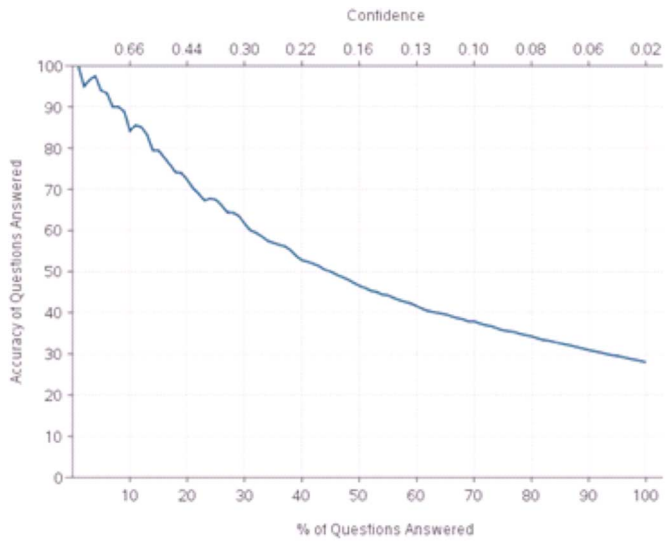

Fig. 8. Accuracy versus percentage of questions answered with lambda 10.

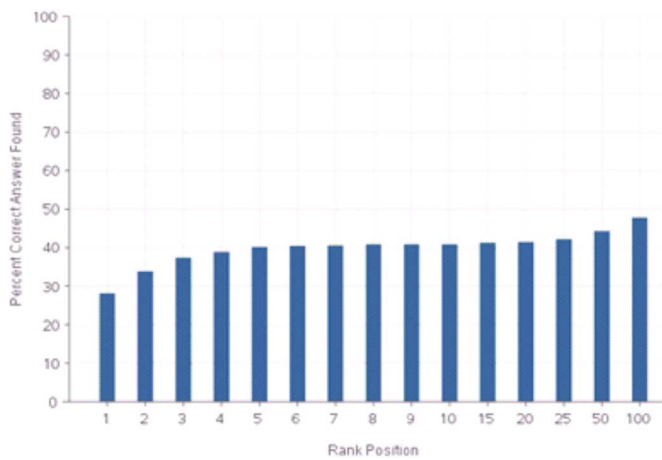

Fig. 9. Rank histogram for lambda 10.

pict that, at $\lambda = 10$, Watson provides the best results before additional training and any functional enhancements for domain adaptation. When Watson was very confident about an answer to a question, the system was more likely correct. As shown in Table II, the overall curve in Fig. 8 provides better accuracy than any lambda smaller than 10. However, as the regularization factor becomes even larger, the system performance degrades. Per Table II, questions with cutoff rank receive the

TABLE III
QUESTION-AND-ANSWER-TYPE EXAMPLE

Question	Literal Answer	Original Regex Answer	Engineered Regex Answer	Updated Question
What controls the Universe?	gravity	(gravity | physics forces)	(gravity | quantum gravity | physics forces | laws of physics)	What scientific force controls the Universe?

same accuracies with $\lambda = 100$ and $\lambda = 1.0^{*}10^{-10}$. However, the rank position of right answers increases as lambda increases beyond 10. As a result, as lambda increases, the algorithm becomes less sensitive to the error term, causing the regularization bias to become too large, which reduces the accuracy. Through hill climbing, an optimal lambda value is reached in Fig. 8 before the algorithm degrades from regularization, as shown in the last row of Table II.

The experimental results clearly show that the regularization term is crucial for Watson to learn to combine evidence in ranking candidate answers in highly imbalanced data.

C. Data Engineering Results

We use a variety of data sets to produce training models during an experiment. To do so, the corpora are held constant while the answer key and question sets are vetted and corrected, to attempt to increase the system performance. Table III depicts how an example question-and-answer pair was altered to remove ambiguous question structure and to increase the coverage of a true label. The question "What controls the Universe?" can be interpreted to have a focus of any noun type. Such focus may shift in the context of politics as compared to the domain of science. Therefore, the question was modified into "What scientific force controls the Universe?" to be more domain specific and focus on the scientific meaning.

The original answer key for the example question in Table III started with only a literal answer of "gravity." The literal answers were converted to regular expressions so that variations of gravity or other acceptable answers were encoded in the ground truth. A modified version of the regular expression answer was created through answer key vetting by experts.

We conducted a series of experiments using these data engineering principles. The initial experiment reveals improvements when answer key vetting and question surface form analysis was performed on the training artifacts. Table IV shows that when all variables are held constant while the answer key is relaxed, the system performance improves. For example, when only perfect matching or literal answers are applied to match candidate answers, Recall@10 maintains 34.70%. However, an improvement of 6.05% is gained with Recall@10 when basic regular expressions are inserted into the answer key. The largest gain occurs when answers, including regular expressions, are vetted with experts and the surface form of questions is disambiguated. Recall@10 improves by 10.1%. Half of the correct answers are found in the top ten candidate answers by relaxing the constraint

TABLE IV
DATA ENGINEERING EXPERIMENTAL RESULTS

λ	Data Engineering	Accuracy	Precision @ 70%	Recall @ 10
10	Vetting Answers and Questions with Regex	29.12%	39.87%	50.85%
10	Basic Regex Answers	27.99%	37.85	40.75%
10	Literal Answers	24.31%	33.98%	34.70%
0	None.	1.58%	1.69%	8.61%
10	None.	27.99%	37.85%	40.75%

TABLE V
RATIO RESULTS DURING DATA ENGINEERING

λ	Data Engineering	R_{tf}	R_{ut}
10	Vetting Answers and Questions and Regex	0.0034	0.5379
10	Basic Regex Answers	0.0031	0.5283
10	Literal Answers	0.0022	0.4520

of answer pattern matching or the regeneration of answers from updated test questions. Precision@70% increases by a smaller amount of 5.89%. Intuitively, a smaller number of new correct answers are found within rank 1 when the top 70% of the questions are answered. Accuracy improved by 4.81% when all of the questions were answered and the right answer was within rank 1.

The indicators R_{tf} and R_{ut} change as the answer key and questions are improved. Table V depicts the change of these two ratios when data engineering techniques are applied to the Watson system. The R_{tf} metric shows that when only literal or direct string matching is used within an answer key, the true-to-false ratio is 1:450. However, when both regular expressions and vetting all aspects of the answer key and questions are applied, the R_{tf} ratio improves to 1:290. The large improvement is also gained in R_{ut}. Through data engineering, the R_{ut} metric improves 8.59%.

VI. CONCLUSION AND FUTURE WORK

Through empirical analysis, we have shown that algorithm and data engineering are complementary techniques for handling extreme class imbalance. A regularized logistic regression function provides significant improvements over an unregularized logistic regression algorithm. The performance in accuracy gains from 1.58% to 27.99%. Further, when the answer key and question set is vetted, recall@10 increases from 13% to 50.85%. The results show that under heavy class imbalance, a regularization term produces good baseline models to start Watson domain adaptation. Therefore, regardless of whether regularization is added to the logistic regression function, the answer key and questions should be vetted.

In practice, though, the overall data state can be altered to compensate for the lack of a regularization term. As the corpora change over time with additional ingested data sources, the alteration of the data state through additional answer key

vetting, question analysis, and oversampling of rare classes will be needed to accommodate the new state. The human intensive work of vetting is very costly in time, over two months for two people, but needs to be traded with performance gain, even if the machine-learning algorithm uses regularization. To further improve the data state, a future statistical knowledge engineering algorithm can rearrange evidence building blocks to produce more true labels. Similarly, a model training session can be combined with automatic hyperparameter tuning of λ to ensure the performance does not drift over time.

Even though Watson is being put to work to help solve real-world business problems, the system still utilizes modified game mechanics to "play." The game-play heuristics now determine at what confidence levels to show supporting evidence of answers. At the conclusion of gamification and the delivery of a Watson system, a human will not be able to distinguish a human subject matter expert from Watson. As Ken Jennings succinctly answered during Final *Jeopardy!*, "I for one welcome our new computer overlords."

Acknowledgment

The authors would like to thank several groups for help and support: J. Chu-Carroll, D. Fallside, J. Fan, D. Gondek, C. Howard, A. Lally, W. Murdock, S. Palani, and P. Parente of IBM; The Cognitive Foundry community; and the Sporting Events Infrastructure Team, specifically, B. O'Connell and S. Cope.

References

[1] S. Ben-David, J. Blitzer, K. Crammer, A. Kulesza, F. Pereira, and J. W. Vaughan, "A theory of learning from different domains," *Mach. Learn.*, vol. 79, pp. 151–175, 2010.

[2] C. M. Bishop, *Pattern Recognition and Machine Learning*. New York, NY, USA: Springer-Verlag, 2006.

[3] M. A. Campbell, J. Hoane, Jr., and F. Hsu, "Deep blue," *Artif. Intell.*, vol. 134, pp. 57–83, 2002.

[4] S. L. Cessie and J. C. Van HouWelingen, "Ridge estimators in logistic regression," *Appl. Stat.*, vol. 41, no. 1, pp. 191–201, 1992.

[5] J. Chu-Carroll, J. Fan, B. K. Boguraev, D. Carmel, D. Sheinwald, and C. Welty, "Finding needles in the Haystack: Search and candidate generation," *IBM J. Res. Develop.*, vol. 56, no. 3/4, pp. 6:1–6:12, May/Jul. 2012, DOI: 10.1147/JRD.2012.2186682.

[6] J. Chu-Carroll, J. Fan, N. Schlaefer, and W. Zadrozny, "Textual resource acquisition and engineering," *IBM J. Res. Develop.*, vol. 56, no. 3/4, pp. 4:1–4:11, May/Jul. 2012, DOI: 10.1147/JRD.2012.2185901.

[7] S. Deterding, R. Khaled, L. Nacke, and D. Dixon, "Gamification: Toward a definition," in *Proc. Gamification Workshop*, Vancouver, BC, Canada, 2011 [Online]. Available: http://gamification-research.org/wp-content/uploads/2011/04/02-Deterding-Khaled-Nacke-Dixon.pdf

[8] S. Deterding, K. O'Hara, D. Dixon, M. Sicart, and L. Nacke, "Gamification: Using game design elements in non-gaming contexts," in *Proc. Extended Abstracts Human Factors Comput. Syst.*, Vancouver, BC, Canada, May 7–12, 2011, pp. 2425–2428.

[9] E. A. Epstein, M. I. Schor, B. S. Iyer, A. Lally, E. W. Brown, and J. Cwiklik, "Making Watson fast," *IBM J. Res. Develop.*, vol. 56, no. 3/4, pp. 15:1–15:12, May/Jul. 2012, DOI: 10.1147/JRD.2012.2188761.

[10] J. Fan, A. Kalyanpur, D. C. Gondek, and D. A. Ferrucci, "Automatic knowledge extraction from documents," *IBM J. Res. Develop.*, vol. 56, no. 3/4, pp. 5:1–5:10, May/Jul. 2012, DOI: 10.1147/JRD.2012.2186519.

[11] D. Ferrucci, E. Brown, J. Chu-Carroll, J. Fan, D. Gondek, A. A. Kalyanpur, A. Lally, J. W. Murdock, E. Nyberg, J. Prager, N. Schlaefer, and C. Welty, "Building Watson: An overview of the DeepQA project," *AI Mag.*, vol. 31, no. 3, 2010, http://dx.doi.org/10.1609/aimag.v31i3.2303, ISSN: 0738-4602.

[12] D. Ferrucci, "Introduction to 'This is Watson'," *IBM J. Res. Develop.*, vol. 56, no. 3/4, pp. 1:1–1:15, May/Jul. 2012, DOI: 10.1147/JRD.2012.2184356.

[13] D. Flatla, C. Guwin, L. Nacke, S. Bateman, and R. Mandryk, "Making calibration tasks enjoyable by adding motivating game elements," in *Proc. 24th Annu. ACM Symp. User Interface Softw. Technol.*, 2011, pp. 403–412.

[14] C. Forbes, M. Evans, N. Hastings, and B. Peacock, *Statistical Distributions*, 4th ed. Hoboken, NJ, USA: Wiley, 2011.

[15] J. Friedman, T. Hastie, and R. Tibshirani, "Regularization paths for generalized linear models via coordinate descent," *J. Stat. Softw.*, vol. 33, no. 1, pp. 1–22, Jan. 2010.

[16] D. C. Gondek, A. Lally, A. Kalyanpur, J. W. Murdock, P. A. Duboue, L. Zhang, Y. Pan, Z. M. Qiu, and C. Welty, "A framework for merging and ranking of answers in DeepQA," *IBM J. Res. Develop.*, vol. 56, no. 3/4, pp. 14:1–14:12, May/Jul. 2012, DOI: 10.1147/JRD.2012.2188760.

[17] R. Gopalan, R. Li, and R. Chellappa, "Domain adaptation for object recognition: An unsupervised approach," in *Proc. IEEE Int. Conf. Comput. Vis.*, 2011, pp. 999–1006.

[18] P. Hayes and K. Ford, "Turing test considered harmful," in *Proc. 14th Int. Joint Conf. Artif. Intell.*, Aug. 1995, vol. 1, pp. 972–977.

[19] N. Japkowicz and S. Stephen, "The class imbalance problem: A systematic study," *Intell. Data Anal.*, vol. 6, no. 5, pp. 429–449, Oct. 2002.

[20] G. King and L. Zeng, "Logistic regression in rare events data," *Political Anal.*, vol. 9, no. 2, pp. 137–163, 2001.

[21] P. Komarek, "Logistic regression for data mining and high-dimensional classification," Ph.D. dissertation, Robot. Inst., Carnegie Mellon Univ., Pittsburgh, PA, USA, 2004.

[22] A. Lally, J. M. Prager, M. C. McCord, B. K. Boguraev, S. Patwardhan, J. Fan, P. Fodor, and J. Chu-Carroll, "Question analysis: How Watson reads a clue," *IBM J. Res. Develop.*, vol. 56, no. 3/4, pp. 2:1–2:14, May/Jul. 2012, DOI: 10.1147/JRD.2012.2184637.

[23] S. Lee, H. Lee, P. Abbeel, and A. Y. Ng, "Efficient L1 regularized logistic regression," in *Proc. Assoc. Adv. Artif. Intell.* , Boston, MA, USA, 2006 [Online]. Available: http://web.eecs.umich.edu/~honglak/aaai06_L1logreg.pdf

[24] M. C. McCord, J. Fan, N. Schlacfer, and W. Zadrozny, "Deep parsing in Watson," *IBM J. Res. Develop.*, vol. 56, no. 3/4, pp. 3:1–3:15, May/Jul. 2012, DOI: 10.1147/JRD.2012.2185409.

[25] M. C. McCord, "Word sense disambiguation in a slot grammar framework," RC23397 (W0411-102), Nov. 10, 2004.

[26] C. I. Muntean, "Raising engagement in e-learning through gamification," in *Proc. 6th Int. Conf. Virtual Learn.*, 2011, paper 42.

[27] J. W. Murdock, J. Fan, A. Lally, H. Shima, and B. K. Boguraev, "Textual evidence gathering and analysis," *IBM J. Res. Develop.*, vol. 56, no. 3/4, pp. 8:1–8:14, May/Jul. 2012, DOI: 10.1147/JRD.2012.2187249.

[28] H. Shah, "Turing's misunderstood imitation game and IBM's Watson success," presented at the AISB Conv., York, UK, 2011.

[29] G. Tesauro, D. C. Gondek, J. Lenchner, J. Fan, and J. M. Prager, "Simulation, learning, and optimization techniques in Watson's game strategies," *IBM J. Res. Develop.*, vol. 56, no. 3/4, pp. 16:1–16:11, May/Jul. 2012, DOI: 10.1147/JRD.2012.2188931.

[30] S. J. Wright, R. D. Nowak, and M. A. T. Figueiredo, "Sparse reconstruction by separable approximation," *IEEE Trans. Signal Process.*, vol. 57, no. 7, pp. 2479–2493, Jul. 2009.

[31] T. Zhang and F. J. Oles, "Text categorization based on regularized linear classification methods," *Inf. Retrieval*, vol. 4, pp. 5–31, 2000.

Aaron K. Baughman received the B.S. degree in computer science from Georgia Institute of Technology, Atlanta, GA, USA, 2002, two certificates in organizational creativity from the Walt Disney Institute in 2003, and the M.S. degree in computer science from The Johns Hopkins University, Baltimore, MD, USA, with a rotation at the National Institutes of Health/The National Institute of Neurological Disorders and Stroke (NIH/NINDS), in 2007.

He is with the Special Events Group, IBM, Research Triangle Park, NC, USA, focusing on predictive cloud and big data for professional golf and tennis sporting circuits. Prior to sports, he was a Technical Lead on a DeepQA (*Jeopardy!*) project. He was named an IBM Master Inventor in 2012, has seven granted patents, and serves on the Big Data Invention Development Team. He has published papers within knowledge discovery and data mining, biometrics, and medical applications.

Wesley Chuang received the Ph.D. degree in computer science from the University of California Los Angeles (UCLA), Los Angeles, CA, USA, in 2001.

He is with the Research Department, Global Business Services of IBM, Chantilly, VA, USA. Prior to joining IBM, he has designed and developed large-scale databases and has published papers in the areas of information retrieval and machine learning. He joined IBM in 2010 and worked on several enabling technologies such as speech recognition and text mining. He is working on commercializing Watson *Jeopardy!*.

Kevin R. Dixon received the Ph.D. degree in electrical and computer engineering from Carnegie Mellon University, Pittsburgh, PA, USA, in 2004.

He is a Principal Member of the Technical Staff at Sandia National Laboratories, Albuquerque, NM, USA, where he leads various research efforts in machine learning and statistical modeling.

Zachary Benz received the B.S. degree in engineering from Harvey Mudd College, Claremont, CA, USA and the M.S. degree in computer science from the University of New Mexico, Albuquerque, NM, USA.

He is a Researcher at Sandia National Laboratories, Albuquerque, NM, USA. His research interests are in cognition, cognitive modeling, and statistical text analysis.

Justin Basilico received the B.A. degree in computer science from Pomona College, Claremont, CA, USA, in 2002 and the M.S. degree in computer science from Brown University, Providence, RI, USA, in 2004.

He is a Senior Researcher/Engineer on the Personalization Science and Engineering team at Neflix, Los Gatos, CA, USA, where he conducts applied machine-learning research for recommendations and personalization systems, with a focus on ranking algorithms. Prior to Netflix, he worked in the Cognitive Systems group at Sandia National Laboratories, Albuquerque, NM, USA.