Automated Assignment of Helpdesk Email Tickets: An Artificial Intelligence Life-Cycle Case Study

Shivali Agarwal, Jayachandu Bandlamudi, Atri Mandal, Anupama Ray, Giriprasad Sridhara

■ *In this article, we present an end-to-end automated helpdesk email ticket assignment system driven by high accuracy, coverage, business continuity, scalability, and optimal usage of computational resources. The primary objective of the system is to determine the problem mentioned in an incoming email ticket and then automatically dispatch it to an appropriate resolver group with high accuracy. While meeting this objective, it should also meet the objective of being able to operate at desired accuracy levels in the face of changing business needs by automatically adapting to the changes. The proposed system uses a system of classifiers with separate strategies for handling frequent and sparse resolver groups augmented with a semiautomatic rule engine and retraining strategies to ensure that it is accurate, robust, and adaptive to changing business needs. Our system has been deployed in the production of six major service providers in diverse service domains and currently assigns 100,000 emails per month, on an average, with an accuracy close to ninety percent and covering at least ninety percent of email tickets. This translates to achieving human-level accuracy and results in a net savings of more than 50,000 man-hours of effort per annum. To date, our deployed system has already served more than two million tickets in production.*

The landscape of modern information technology service delivery is changing, with increased focus on automation and optimization. Most information technology vendors today have service platforms aimed toward end-to-end automation for carrying out mundane, repetitive labor-intensive tasks and even for tasks requiring human cognizance. One such task is ticket assignment and dispatch, where the service requests submitted by the end-users to the vendor in the form of tickets are reviewed by a centralized dispatch team and assigned to the appropriate service team and resolver group.

The dispatch of a ticket to the correct group of practitioners is a critical step in the speedy resolution of a ticket. Incorrect dispatch decisions can significantly increase the total turnaround time for ticket resolution, as observed in a study of an actual production system (Agarwal, Sindhgatta, and Sengupta 2012). When such delays occur, it causes customer dissatisfaction as well as monetary penalties for the vendor due to service-level-agreement breaches. Several factors make the dispatcher's job challenging, namely the need for in-depth knowledge of the roles and responsibilities of various groups, the heterogeneous and informal nature of email text, and the high attrition rate in service delivery teams (Mandal et al. 2018).

Given the fact that inefficiencies in dispatch have serious business consequences, there has been a lot of interest in automating the assignment process. A number of different approaches have been proposed for automating ticket dispatch (Agarwal, Sindhgatta, and Sengupta 2012; Shao et al. 2008a, 2008b; Parvin, Bose, and Van Oyen 2009).

We propose a highly reliable and accurate cognitive system for assignment of tickets that come in the form of email. The tickets may be raised in different ways such as through voice, web forms, or emails to a centralized helpdesk team. Our proposed system focuses on email tickets only, but can be applied to other forms of tickets that use text.

Challenges in Cognitive Email Assignment

At first glance, email assignment may look like a simple text-classification problem — but it becomes quite complex and challenging when considered at industry scale.

First, for most big companies the number of resolver groups is quite large — of the order of 500, in some cases. Many of these resolver groups cater to overlapping problems that can be disambiguated only with domain-specific knowledge. Empirical observation shows that it is difficult for training algorithms to understand such subtle differences. Second, in most businesses, the helpdesk teams themselves undergo constant changes for better efficiency and productivity. Business decisions often may lead to resolver groups being split, merged, or renamed. All of these changes inevitably impact the accuracy of machine-learning classifiers. By continuous retraining, the model may catch up with the changes — but by the time it actually catches up, a number of tickets would have been incorrectly assigned, causing disruptions in the business and a poor customer experience. For splits and merges, the impact is worse. Third, the problems assigned to resolver groups themselves slowly change over time. This may not happen over a week, or a month, but over several quarters. As such a once-trained model becomes outdated over time, it cannot effectively assign tickets mentioning new problems or old problems with a different terminology (Mandal et al. 2018).

Main Contributions

This article presents an end-to-end automated ticket dispatch system that addresses all these challenges faced in an enterprise dealing with large volumes of tickets on a daily basis. The system uses a combination of classifiers with separate strategies for handling frequent and sparse resolver groups (referred to in this article as the short-head and long-tail, respectively). To deal with ambiguity in resolver-group assignment, we have designed a rule engine that captures the domain-specific knowledge required for disambiguation.

The need for retraining is a reality in enterprise artificial intelligence (AI)-based solutions due to a gradual shift in business and technology focus that we deal with by having a focused retraining strategy and automatic rule-mining capability for the rule engine. Subtle changes in email utterances over a period of time can lead to loss of accuracy and this is handled by having an intelligent retraining strategy. The automatic rule mining helps in discovering new, merged, or phased-out resolver groups to ensure business continuity. It can be observed that our readily deployable end-to-end automatic email dispatch system is an embodiment of a system that encapsulates complete AI life cycle. It has the following four key features:

Classification Models to Predict Mandatory Fields for Ticket Dispatch

A ticketing tool mandates fields like ticket category and priority or severity to be populated before a ticket can be dispatched to the predicted resolver group. We use different classification models to predict mandatory ticketing fields. To predict the resolver group, we use an ensemble-based classification engine that uses supervised machine learning to understand the nature of the problem from free unstructured email text and assign accurately. We also use a second set of support-vector-machine (SVM)–based classifiers to predict other mandatory fields that are required for creation of the ticket.

Long-Tail Strategy

A few-shot learning strategy based on Siamese networks for the rarely occurring classes (or long tail). By training the long tail separately from the short head, we reduce noise in the training data and also eliminate the problem of class imbalance to a large extent.

Automatic Rule Mining

The rules are designed to strategically combine with machine-learning methods for effective disambiguation of classes. Rules are extracted automatically using a technique based on Gini impurity and association rule mining followed by a verification step. The automatic rule-mining technique removes most of the manual labor involved in understanding resolver-group issues and designing rules from scratch.

Retraining

An effective retraining strategy that combines sliding window-based training with active learning technique. This ensures that the models are up-to-date with the changes that happen over time in the email utterances and resolver-group organization, and at the same time the accuracy is maintained at a desired level.

The results are presented with real customer data from three different datasets — with the largest of them having more than 700,000 emails and as many as 428 resolver groups. We were able to achieve human level accuracy with more than ninety-percent coverage on all the datasets with the proposed system using minimal computational resources.

To the best of our knowledge, this is the first time that human-level accuracy has been reported for a deployed assignment engine at this scale of automation, delivered consistently across datasets of varying sizes and forms. The remainder of the article is organized as follows. First is a description of the type of work involved (Related Work), and then we give

an explanation of the system used for ticket classification (System Overview). We go on to discuss the different components of the system (Assignment Engine Components), present our experimental results (Classification Models), offer our remarks (Evaluation), and then conclude (Conclusions and Future Work).

Related Work

Ticket dispatch is a known problem that has been addressed in the past through different approaches. These approaches have handled the dispatch problem almost in isolation and have not addressed the systemic level issues that arise in a real deployment. We discuss the key known approaches in the following paragraphs.

Ticket dispatch has been addressed by Agarwal, Sindhgatta, and Sengupta (2012) using SVMs and a discriminative keyword approach. They propose a semiautomated approach based on confidence scores. We have surpassed their work to reach human level accuracy using advanced ensemble techniques for automated dispatch, scaled it to hundreds of resolver groups, and incorporated retraining strategies to adapt to changing data. Several other researchers have studied different aspects of the problem of routing tickets to resolver groups (Shao et al. 2008a, 2008b; Parvin, Bose, and Van Oyen 2009). The work presented by Shao et al. (2008b) approaches the problem by mining-resolution-sequence data and does not access ticket description at all. The work by Shao et al. (2008a) focuses on ticket transfers between groups (given an initial assignment) without looking at the ticket-text content. It mines historical ticket data and develops a probabilistic model of an enterprise social network representing functional relationships. The work by Parvin, Bose, and Van Oyen (2009) is different and approaches the problem from a queue perspective. This is more related to the issue of service times and becomes particularly relevant for agent assignment within a group. There are some papers that apply text-classification techniques to handle tickets (Dasgupta et al. 2014; Zeng et al. 2017). The idea is that once ticket category is identified, then the assignment to resolver groups can be done by manual dispatchers quickly. However, none of the works talk about the scale and retraining required in real-life deployment. In Di Lucca (2002), tickets are automatically classified based on description to route them to the right group. However, the work was applied on a small ticket set with only eight groups. The work by Kadar et al. (2011) attempts to classify the incoming change requests into one of the fine-grained activities in a catalog. Some other works by Potharaju, Jain, and Nita-Rotaru (2013) and Agarwal et al. (2017) talk about a holistic approach of ticket category classification, cause analysis, and resolution recommendation. However, they do not automate the process of assignment.

A rule engine is an important part of our system to handle domain-specific cases. There exists a body of literature that attempts automated mining of classification rules in a human-understandable way. The work by De Falco, Cioppa, and Tarantino (2002) presents a genetic programming framework, capable of performing an automatic discovery of classification rules easily comprehensible by humans. The work by Shang et al. (2016) describes the mining of discriminative patterns, which are the prefix paths from root to nodes in a tree-based classification task. These feature discriminative patterns that are concise, and offers high interpretability of the model. Our work leverages a Gini-impurity-based (D'Ambrosio and Tutore 2011) decision-tree model to extract prefix paths, and then applies association rule mining to obtain fine-grained human-understandable rules.

Another challenge that we have to deal with is long-tail classification. This problem can be mapped to few-shot learning, where the model can be learned with a very limited number of training samples. There have been lot of recent advancements in the area of few-shot learning. One of the more prominent works is the one by Mueller and Thyagarajan (2016), which proposed Siamese-architecture language models to learn the semantic similarity between sentences for few-shot text classification. Another body of work (Bao et al. 2019) uses metalearning with distributional signatures for few-shot text classification.

System Overview

Figure 1 shows the system architecture along with the data flow diagram. Historical email ticket data are downloaded from the ticketing tool (for example, Remedy [Remedy Corporation], ServiceNow [ServiceNow Inc.]) using custom-built adapters. The downloaded emails are passed through two stages of preprocessing for data enrichment. The data enrichment module uses techniques like resolver-group merging, long-tail cutoff, and so forth, to reduce the noise in the email data. The training data are further cleaned using text preprocessing methods (Manning et al. 2014). The cleaned email data are then trained using an ensemble of machine-learning classifiers, and the trained models are serialized and stored in a database. For periodic retraining, we use an active-learning–based strategy that has sliding windows along with informative sample selection.

When a user sends an email to the helpdesk account, a ticket is automatically generated and stored in the backend ticketing tool. The newly generated tickets are downloaded by the adapter and classified using a system of classifiers, a dispatcher, and the rule engine. The classification system returns predictions for mandatory ticketing fields along with the corresponding confidence scores. If the confidence score is above a configured threshold, the ticket is routed to the predicted resolver group. Otherwise, the ticket is assigned to the manual queue for inspection by human agent. The combination of classifiers and rule engine ensures that a high percentage of tickets (more than ninety percent) are assigned automatically by our system with a low error rate.

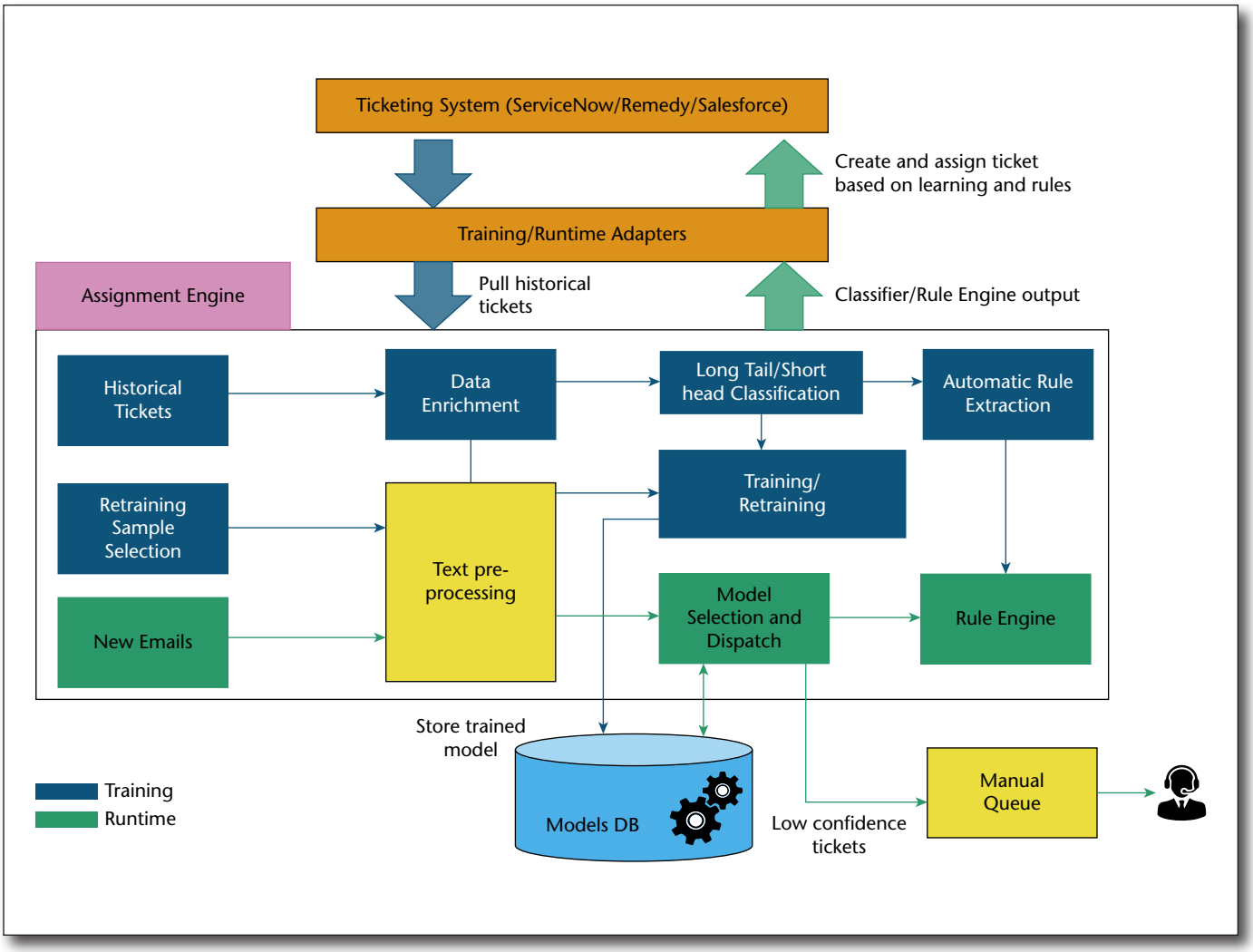

Figure 1. Architecture of the Proposed System.

We will now define key terms used in the rest of the article. Let N be the total number of email tickets. In the manual assignment case, let NH_1 be the number of tickets for which the ticket was ultimately resolved by the same group to which the ticket was initially assigned. Let NH_2 be the number of tickets for which the initial and final groups differ. Then, human-level accuracy H_{acc} can be defined as a ratio of NH_1 to total tickets N. In the automated email assignment scenario, tickets are assigned by the assignment engine, which combines the machine-learning classifiers and rule engine. Let NX be the total number of tickets actually assigned by the classifier system; NX_{corr} be the number of tickets that were predicted by the ensemble and for which the resolver group predicted correctly (that is, the ticket was ultimately resolved by the predicted resolver group); NR be the number of tickets where the resolver group was predicted by the rule engine; and NR_{corr} be the number of tickets predicted by the rule engine and for which the resolver group predicted correctly. Then we can define Classifier Accuracy (X_{acc}) as the ratio of NX_{corr} to NX; Classifier Coverage (X_{cov}) as the ratio of NX to N; Assignment Engine Accuracy (E_{acc}) as the ratio of the sum of NX_{corr} and NR_{corr} with the sum of NX and NR; and Assignment Engine Coverage (E_{cov}) as the sum of NX and NR to N.

Assignment Engine Components

Having defined the system, we now describe in detail the different functional components of the assignment engine.

Preparation of Training Data

This section explains the bootstrapping phase of our system. The ticketing tool (Remedy, ServiceNow, and others) organizes email data into structured fields containing relevant information about the ticket such as incident type, creation date, problem description, resolver group, and so forth. We use custom adapters to connect to the ticketing tool and extract fields

relevant for training. Currently, the adapter extracts only the text portion of the email (namely, email subject and body) along with the resolver group for training. The extracted email data are passed through several layers of enrichment (Mandal et al. 2019). For the sake of completeness, we next discuss the main points.

Merging Related Resolver Groups

Some of the resolver-group labels in the training data can be merged. Merging increases the size of the training data and at the same time reduces the number of unique labels, thus improving training accuracy. We found that there are at least two types of resolver groups that can be merged for assignment purposes — resolver groups with varying escalation levels, and region-specific resolver groups. For more details, we refer the reader to the work by Mandal et al. (2019).

Long-Tail Cutoff

For better training, we divided historical email-ticket data into two parts: $I_T = I_H + I_L$, where I_T is the complete data downloaded for training, I_H is the data corresponding to the frequently occurring resolver groups (short head), and I_L is the data corresponding to the sparse resolver groups (long tail; Mandal et al. 2019). Resolver groups belonging to I_T are classified using an ensemble classifier, and those belonging to I_L are handled using a few-shot learning strategy that is described later in this section. In our system, we use this strategy to retain at least ninety percent of data in the short head while cutting down the resolver group count by about eighty percent. The evaluation results are shown later in the evaluation section.

Text Preprocessing

We applied some text preprocessing on the training data (I_T and I_L) before using it to train a model: we replaced multiple tabs and spaces with a single space, removed control characters, non-ASCII characters, and hypertext-markup-language tags. Other methods such as stemming, lemmatization, and extractive summarization were also tried. We used training data augmentation techniques such as paraphrasing and sample duplication (Mitchell 1997) for resolver groups with small numbers of samples.

Classification Models

This section presents our study on the performance of various machine-learning classifiers in classification of email data, in terms of accuracy and training time. We follow separate strategies for predicting the resolver group as opposed to other mandatory fields. So, first we talk about the prediction of the resolver group and then move on to the prediction of other ticketing fields.

For training the classification models, we concatenate the subject and the body of the email (description) with a space in between and use the resulting string as our training data. The resolver group acts as the label for our training data.

Short-Head Classification

We convert the training-data samples into word-vector representation before applying machine-learning algorithms. We observed that using term frequency-inverse document frequency (tf-idf) representation increased the accuracy of traditional machine-learning algorithms for all datasets by at least three to four percent. Furthermore, use of bigrams also improved the accuracy for some datasets. Intuitively, we can argue that this is so because some bigrams such as account creation, account deletion, or password reset are useful indicators in deciding the resolver group. The hyperparameters were chosen experimentally over 10-fold cross validation on the datasets.

However, for deep-neural-network learning, tf-idf representation, being extremely sparse, is not useful. Our article (Mandal et al. 2019) shows the impact of various traditional machine learning models (Mitchell 1997) and deep-neural-network models (Goodfellow, Bengio, and Courville 2016) in short-head classification. To improve classification accuracy and coverage of the overall service, we used an ensemble (Kuncheva 2004). Each pair of models was combined, and the final ensemble classifier was chosen based on the accuracy and coverage. Our results suggest that a combination of linear SVM- and multilayer-perceptron (MLP)–based classifiers performs best.

Long-Tail Classification

Long-tail classes typically account for about five to ten percent of the training data and have very few training samples per resolver group. For these classes, traditional classifier performance using tf-idf input features is not satisfactory, because models cannot understand the semantic similarity from the few available samples. As such, Mandal et al. (2019) refrains from using machine learning to predict these classes; instead, they use a configurable rule engine. The drawback is that a lot of rules need to be manually created and maintained. To solve this problem, we used a few-shot learning strategy. Specifically, a Siamese adoption of a Long Short Term Memory (LSTM) network with a Manhattan distance metric for learning semantic similarity (Mueller and Thyagarajan 2016) was used. Our approach uses LSTM representations to map ticket descriptions to fixed-length feature vectors and Siamese architecture adoption is used for learning semantic similarity between tickets of different classes. Figure 2 shows a Siamese architecture that comprises two identical LSTM networks and the model is trained to learn similarity between pairs of ticket descriptions.

For each pair of tickets, we create training data using the tuple: (TD_a, TD_b, *similarity*), where TD_a is description of ticket T_a, TD_b is description of ticket T_b, and *similarity* is a binary label (1 or 0) . Depending on whether T_a and T_b are chosen from the same resolver group or from different ones, the value of *similarity* will be 1 or 0, respectively. So, if there are N long-tail classes (LTC_1, LTC_2, ..., LTC_N) and LTC_i

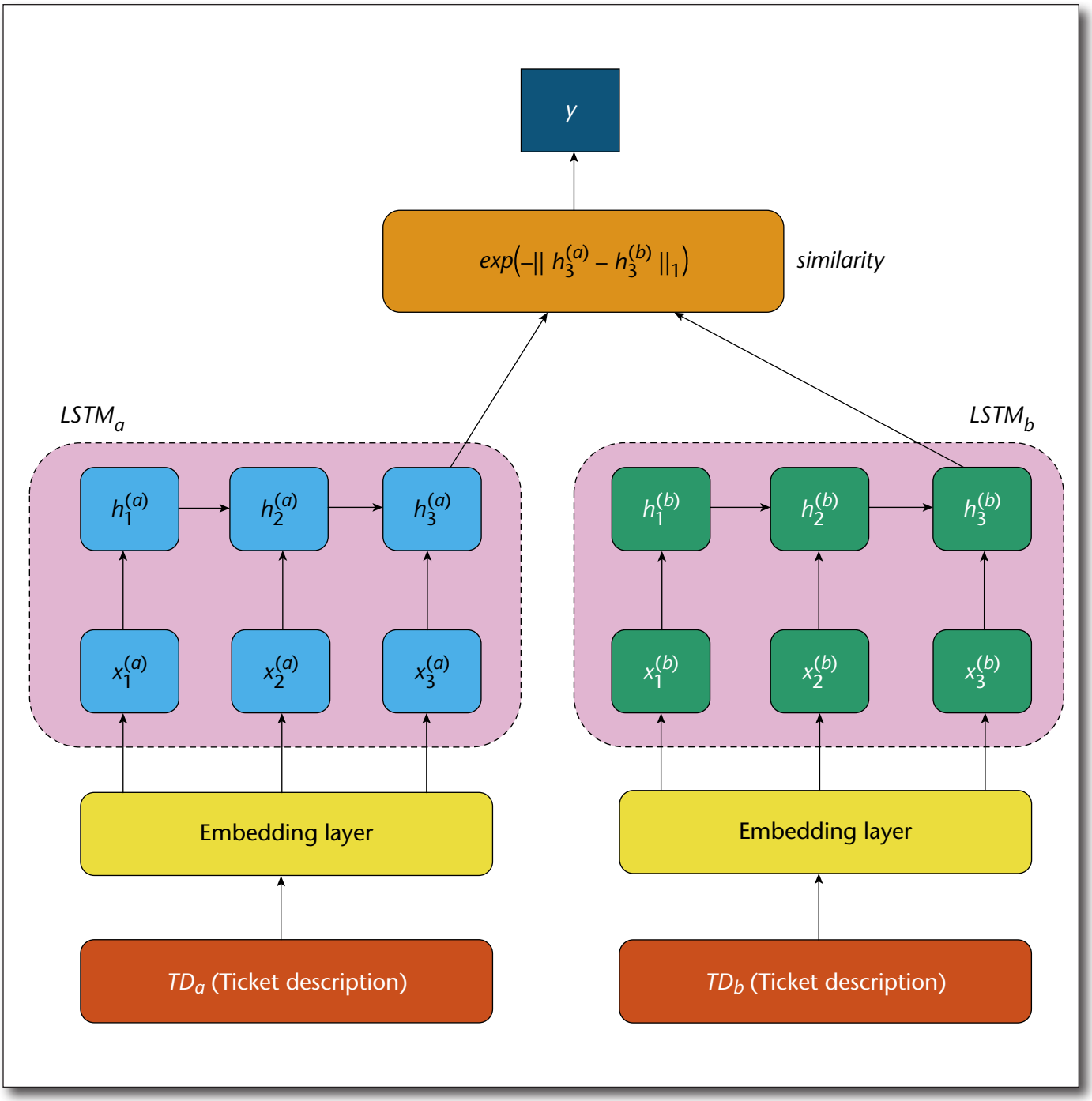

Figure 2. Siamese Architecture.

has n_i tickets, then from each ticket T_x belonging to LTC_i we can create $(n_i - 1)$ similar tuples and $(n_1 + n_2 + ... + n_{i-1} + n_{i+1} + ... + n_N)$ dissimilar tuples. This technique also ensures that we have sufficient training data despite having only a few training samples per class.

We train the Siamese LSTM using the training data obtained above with TD_a, TD_b as inputs to the LSTMs, and *similarity* as label. Model training parameters are LSTM with five hidden units, dropout = 0.5, and ADAM optimizer.[1] The trained Siamese LSTM model predicts the class label for each ticket by finding the most similar ticket with respect to the Manhattan distance metric.

Long-tail classification results are shown in table 1 for three of our biggest datasets. The results indicate that while the Siamese networks are better suited for long-tail classification, SVM is not very far behind. In fact, we observed that when the sparse classes have ten samples or more, SVM often performs better than the Siamese networks. So if storing Siamese-based models is a concern, SVM can be used instead.

Dataset	SVM	Siamese Networks	Logistic Regression	Naive Bayes	MLP
A	68	76	63.3	65	68
B	33	66.7	44.5	22	55.6
C	63.6	72.7	55	50.6	59

Table 1. Comparison of Long-Tail Classification Methods.

Predicting Other Mandatory Fields

Although, as we discussed, predicting the resolver group is the most difficult, we also need to predict other mandatory fields so that the ticket can be created and dispatched successfully. Some of these other fields include incident type (incident or service request, or service restoration), problem category, and priority/severity. These ticketing fields usually have only a few classes and there are enough samples for each class. So the problem of long-tail and class imbalance is virtually nonexistent. As such, we use a simple linear SVM (with our scheme)-based classifier for predicting these fields. We could achieve satisfactory accuracy of greater than eighty-five percent with this method, so we do not use a confidence threshold for these fields, as imposing a cutoff will bring down the coverage of the entire system (as all the fields need to have confidence above cutoff).

Rule Engine

The rule engine is one of the key components of our end-to-end system used to handle scenarios that are typical of an enterprise. Such scenarios generally correspond to business design and decisions and are not amenable to machine-learning or deep-learning classifiers. The three main scenarios are as follows:

Resolver Group Perturbations Driven by Business Decisions

Often resolver groups are either renamed or split or merged to form new resolver groups. These decisions are mostly taken to remove duplication of effort, or to address macro-economic changes. As most of these decisions are sudden, machine-learning models are not able to handle classification for the newly formed classes, which affects services in production.

Resolver Groups Belonging to the Long Tail

As discussed previously, in most datasets, twenty percent of the classes account for more than ninety percent of the tickets. The remaining eighty percent of the classes are predicted separately, using a few-shot learning strategy. However good our few-shot learning model might be, it cannot match the accuracy of the short-head ensemble classifier or that of humans. Hence, to keep the overall accuracy high and at par with human accuracy levels, we have to design rules for predicting these classes.

Presence of Resolver Groups with Similar or Overlapping Email Format

Helpdesk teams usually have a default group that handles lots of different types of tickets. The kinds of problems this default group handle are often very similar to the problems handled by other, more focused groups. As such, the user utterances are also quite similar and hence confusing for machine-learning algorithms. Many helpdesk organizations use fixed templates for submission of certain types of issues. The same template can be used for multiple resolver groups. When these tickets are used to train the machine-learning model, it learns the template structure rather than the actual content. So the classification accuracy is very low for such resolver groups. The rule engine addresses this issue for the confusing classes by overriding the decision of the machine-learning classifier.

Design of the Rule Engine

The rule engine is designed to have a customer-independent framework for rule specification, and is easy to configure using a user interface. The user interface allows the specification of rules that use values of ticket parameters such as email subject, description, and so forth, as well as the output of the machine-learning classifier. The rule engine can override the output of the classifier in certain cases. Each rule in the rule engine can be generically expressed using *implies relation*. The left-hand side is conjunctions of atoms of form $(f_i \supseteq s_i)$, $(C_E = R)$. The right-hand side is C_F, where f_i values are the ticket fields (for example, *description*, *title*, *sender email*, *recipient email*) that are used in the representation of the rule; s_i is the value of field f_i (s_i can be a singleton or a set of terms); C_E is the ensemble-classifier-predicted class; R denotes the resolver group for which the rule is applicable; and C_F is the final resolver group predicted after application of rule engine.

Some sample rules are shown in table 2 for better understanding.[2] The first rule is for renamed resolver groups (R5 renamed to R63). If the ensemble-predicted class is R5, the final prediction will be R63, irrespective of the field values. The second rule is used for commonly confused sets of resolver groups. Here, the ensemble is used to determine the commonly confused set (R17, R42); then the rule is applied to determine the exact resolver group within the set. The third rule uses a combination of values from three different fields to

$f1$	$f2$	$f3$	C_E	C_F
—	—	—	R5	R63
Contains "HSS" only	—	—	R17	R42
Contains "replenish team"	(*abc*)	(*xyz*)	—	R54
Contains both "HSS" and "EWM"	—	—	—	R43

Table 2. Sample Rules.

override machine-learning prediction. The fourth rule is similar to the third, except that it uses a value of only one field to make a prediction. However, the important thing to note is that there is an order of precedence between rules 2 and 4. Rules 2 and 4 have a common match criterion for field f_1 (namely, use of the keyword *HSS*). In these cases, we match the more-restrictive rule first to minimize the chance of false positives.

Automatic Rule Mining

Mandal et al. (2019) relies on manual maintenance of the rule engine. However, as the ticket corpus grows in size, the task of maintaining a wide variety of rules becomes tedious, and having an automated rule-mining strategy becomes inevitable. We use a semi-automatic rule-mining strategy that is able to mine rules with high recall, but relatively low precision. High recall ensures that we don't miss any valid rules; low precision means that there will be a few false positives. As such, the mined rules are passed on to subject-matter experts (SMEs), who verify the rules and retain or remove them, as necessary.

Our approach uses a random-forest model with a Gini-impurity split-criterion on the processed ticket corpus. A random-forest model offers a high level of interpretability and debuggability, which is important for our purpose, as the extracted rules need to be manually verified. (The rule extraction can also be done using complex deep-learning models. But in that case, the generated rules will have little or no human interpretability.) The steps followed in automatic rule mining are described in the sections that follow.

Step 1: Obtaining Confusion Matrix and Confused Resolver-Group Sets

The first step in automated rule mining is to build the confusion matrix. Next, we analyze the confusion matrix and identify the confused sets (consisting of two or more resolver groups). Several such sets can be mined from the confusion matrix. The construction of the confusion matrix, and retrieving the confused class sets, is done automatically using scripts.

Step 2: Extracting Decision-Tree Paths

After obtaining ticket corpus for a confused set, we follow below the steps to obtain decision paths from a random-forest model. The same procedure is followed for all confused sets.

Ticket Corpus Preprocessing. At first, the ticket corpus is preprocessed by using techniques such as tokenization, lemmatization, and named-entity recognition. The preprocessed corpus is then converted to bigram tf-idf features consisting of important tokens, phrases, and entities.

Decision Tree with Gini Impurity. Using the tf-idf features, we train a decision-tree classifier with Gini-impurity split criterion (D'Ambrosio and Tutore 2011). Reason for choosing Gini impurity is due to its interesting property that, at a given split *s*, if all records belong to one class, then Gini impurity becomes zero and the decision-tree path from root to the current-node split always results in a single class or resolver group. In decision-tree-model training, multiple splits happen at each parent node based on the Gini information gain (D'Ambrosio and Tutore 2011). Gain is calculated for all possible features and the feature with highest gain will be used for the split. These splits happen iteratively until the stopping criterion is met. In our approach, we let the splits happen in an unpruned manner — that is, until all records in a leaf node belong to a single resolver group or the leaf node contains a single record.

Figure 3 shows the example decision tree for two classes in the confused class pair (R12, R189) trained on training samples from dataset C(3) using tf-idf features. For example, the decision path ($postpaid \leq 0.5 \cap order \geq 0.5 \cap mobile \leq 0.5$) $\Rightarrow R12$ results in a leaf node containing all eleven samples belonging to R12 class. It is to be inferred that if the ticket description does not contain [*postpaid,mobile*] but it contains *order*, then the decision-tree model predicts the class as R12, and there are eleven such examples in the training set (out of 200) that follow this decision path. The decision path from *root* to corresponding *node* with $gini = 0$ always results in single class prediction.

Decision Paths Extraction. Each node in the decision tree corresponds to a ticket feature; hence extracted decision paths explain the class assignment for each ticket in terms of input ticket features and offers high interpretability. It is possible to extract several paths, but we are only interested in those with

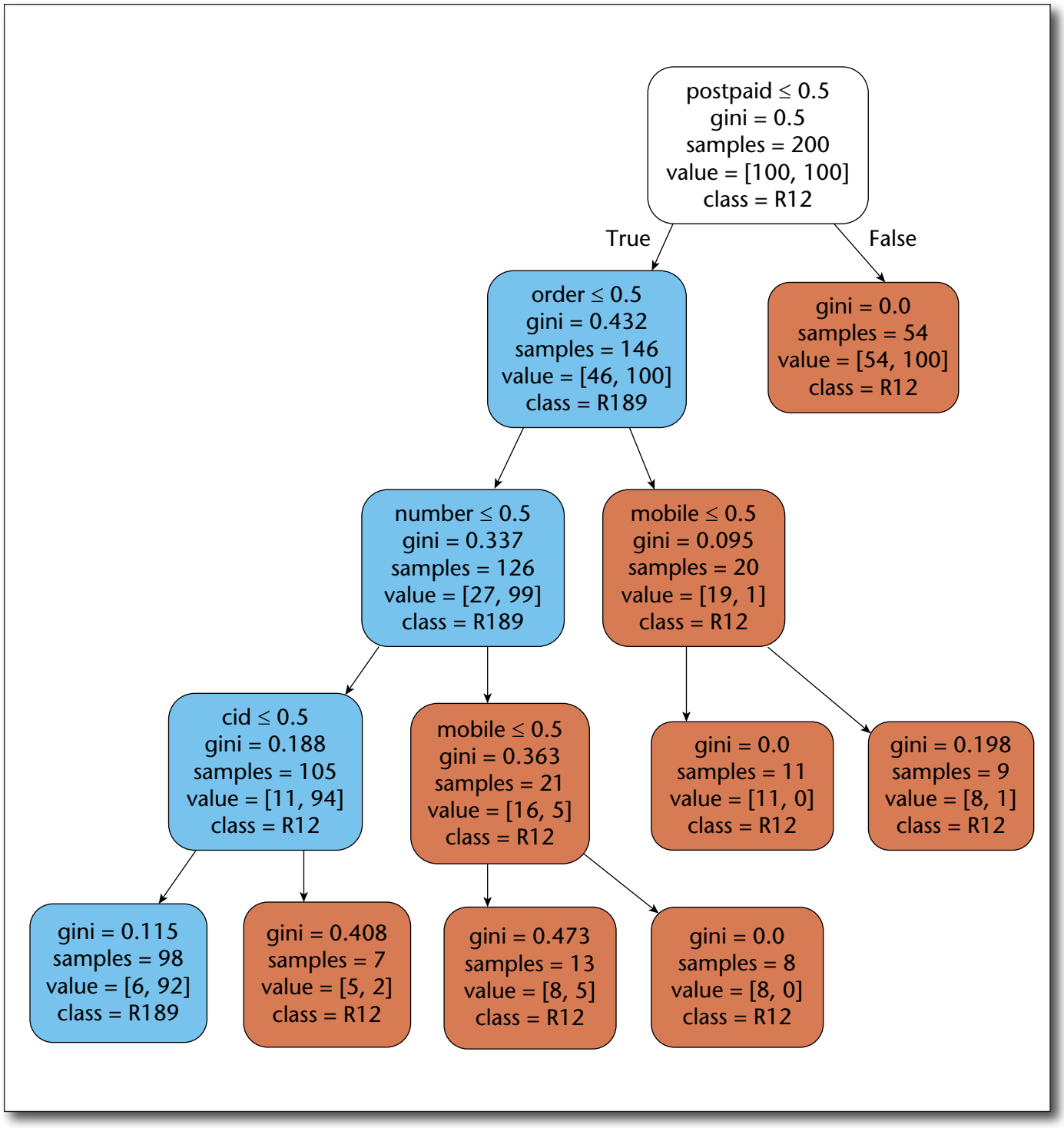

Figure 3. Gini-Impurity Tree.

leaf node having zero Gini impurity and support above a specified cutoff value. The support threshold ensures that we have enough evidence that the decision path is valid, and a rule can be extracted from it.

A single-decision–tree model is prone to over-fitting, and decision paths extracted from just one decision tree are not robust. So we trained a random-forest model that is an ensemble of several decision trees. From each decision tree in the random-forest model, we extracted Gini-criterion decision paths for the next step.

Step 3: Applying Association Rule Mining

One problem with using decision paths extracted from a Gini-impurity–based random forest model is

Description	ML Predicted	Final Class	Correct?	Found by SME?
cancel order	R12	R189	No	No
msisdn imsi	R12	R189	Yes	Yes
mention impacted mobile	R12	R189	Yes	Yes
[firewall access raise, information error screenshot, imsi msisdn imsi]	R12	R189	Yes	No
mention apps impacted	R12	R189	No	No
service delivery	R12	R189	No	No
pending order	R12	R189	No	No
raise tickets remedy	R31	R189	No	No
imsi additional information	R31	R189	Yes	No
impacted mobile imsi	R31	R189	Yes	Yes
firewall access raise	R31	R189	No	No

Table 3. Correctness of Rules Extracted for a Resolver Group in Dataset C (R189).

that a lot of decision paths are generated, some of which may not be useful. To obtain more concise and robust rules, association rule mining (Zhang and Zhang 2002) is applied on top of the extracted decision paths. Association rule mining finds relations and dependencies between elements in an item set. In our case, the item set is a decision path and each element in the item set is a phrase. Some examples of decision paths extracted for resolver-group R189 using Gini impurity are as follows: [information error screenshot, browser, recipient]; [information error screenshot, imsi msisdn imsi, confidential, firewall access raise]; [mobile device, information error screenshot, imsi msisdn imsi, internet, firewall access raise, update]. The downside of these decision paths, which eventually form the rules, is that they overlap with one another and are also long and tedious. All three of these decision paths contain an information error screenshot, and it is a generic phrase that appears in most of the decision paths, but the underlined items information error screenshot, imsi msisdn imsi, and firewall access raise appeared together and there exists a strong relation among them. After application of association rule mining, we get only the following association rule for resolver group R189: (*information error screenshot,imsi msisdn imsi*) → (*firewall access raise*). This means whenever *information error screenshot* or *imsi msisdn imsi* appear together, *firewall access raise* will be there for tickets in resolver-group R189.

The quality and the robustness of association rules are evaluated based on the constraint metrics *Support* and *Confidence*.

The mined association rules are evaluated by SMEs before updating the rule engine. Table 3 shows the mined rules for one such resolver group in one of our datasets.

Model Selection and Ticket Dispatch

The email ticket dispatcher assigns the ticket to a specific resolver group and updates the ticket. The dispatcher combines the results of the two classifiers and rule engine using a dispatch algorithm to output the final prediction and confidence score.

The classifiers are invoked in a specific order for best results. As more than ninety percent of the tickets belong to the short head, the ensemble classifier is invoked first. The long-tail classifier is invoked next, but only if the confidence from the ensemble classifier is below a preconfigured threshold. Finally, the rule engine is invoked irrespective of the output of the classifiers. This is done because the predicted resolver group may belong to one of the confused classes; so there is room for ambiguity even if the classifier prediction has high confidence. If a rule is matched by the rule engine, the classifier output is overridden. If no rule is matched, then we fall back to the classifier that was invoked last in the process. If the confidence from that classifier is high, we retain its prediction, otherwise we assign the ticket to the manual queue.

Retraining

We have already discussed how the rule engine takes care of changes happening in email data due to resolver groups getting renamed, merged, or split. However, there are still other changes in training data that need to be handled, such as subtle changes in email utterances over time. These changes can happen due to various factors like resolver groups taking on new problems, increased automation, and so forth. To capture these changes effectively, we use a sliding-window–based retraining strategy. The sliding window ensures that training data are refreshed periodically and the most recent changes are retained, so the classifiers remain up-to-date.

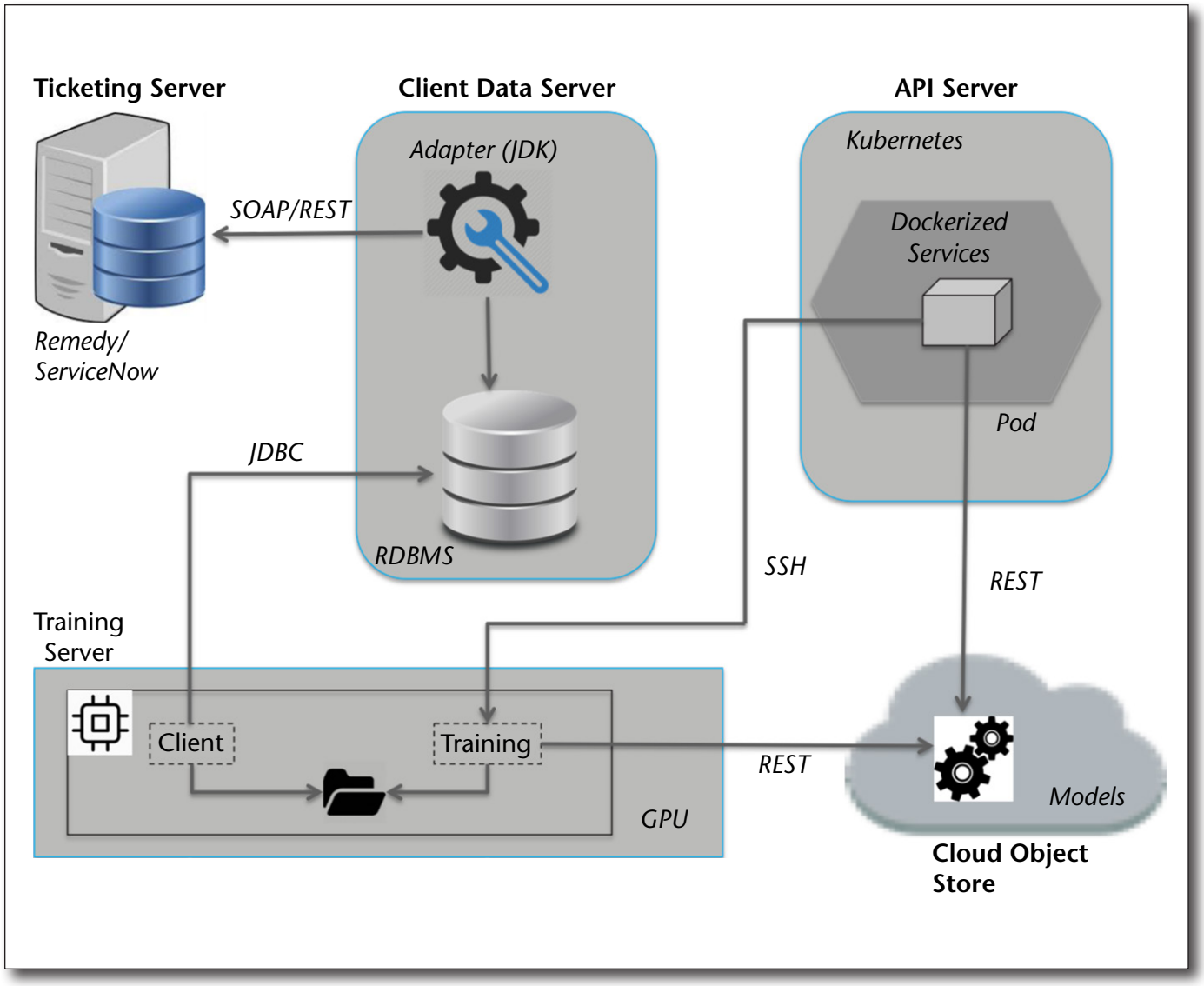

Figure 4. Deployment Architecture.

The length (*W*) of the sliding window is determined such that the training data are sufficient for good accuracy. The slide interval (*T*) is determined by the average time it takes for a ticket to be fully resolved in the system.

Maintaining Model Accuracy Over Time

In addition to the sliding window data, we also retain the most informative samples from previous periods so that we keep learning from past mistakes. We try to automate the process of informative sample selection using active-learning strategies. The most commonly used strategies to retain informative samples are least-confidence, margin sampling, and entropy sampling (Settles 2009). Prior studies have suggested that out of these three techniques, margin and least-confidence metrics are the most suitable for reducing classification error rate, whereas entropy is more appropriate for minimizing the loss function (Fu, Zhu, and Li 2012). Other prior studies have clearly suggested margin sampling as the most effective strategy for multiclass classification problems (Schein and Ungar 2007; Reyes, Altalhi, and Ventura 2018). Based on the evidence given in these articles, we adopt margin sampling as our sample-selection strategy for retraining.

Deployment and Maintenance

The automated assignment engine is currently deployed as a single tenant service hosted in a Kubernetes-cluster deployment architecture (figure 4) with three representational-state transfer, application-program–interface endpoints: *train* — this call-to-train method is asynchronous, and returns a training id; *status* — this checks the status of a training in progress; and *classify* — this calls the classifier to predict the resolver group. The ensemble classifier is run periodically on a graphics-processing-unit cluster

Resolver Groups	Dataset A	Dataset B	Dataset C	Dataset D	Dataset E	Dataset F
Number of Resolver Groups	70	403	428	73	2,892	31
Duration of the Training Dataset	6 months	12 months	15 months	12 months	12 months	6 months
Email Tickets (*N*) in Training Set	11,562	423,343	712,320	28,940	23,811	6,529
Duration in Deployment	31 months	15 months	28 months	10 months	8 months	6 months
Tickets Served/Month (*T*)	2,000	40,000	50,000	2,000	2,000	1,000
Total Tickets Served to Date	62,000	600,000	1,400,000	20,000	16,000	6,000
Assignment Engine Accuracy (Eacc)	92.73%	88.66%	92.13%	93.1	91.5	91.2
Assignment Engine Coverage (Ecov)	97.84%	93.3%	95.5%	96.1	94.5	93

Table 4. Dataset Size, Usage, and Results.

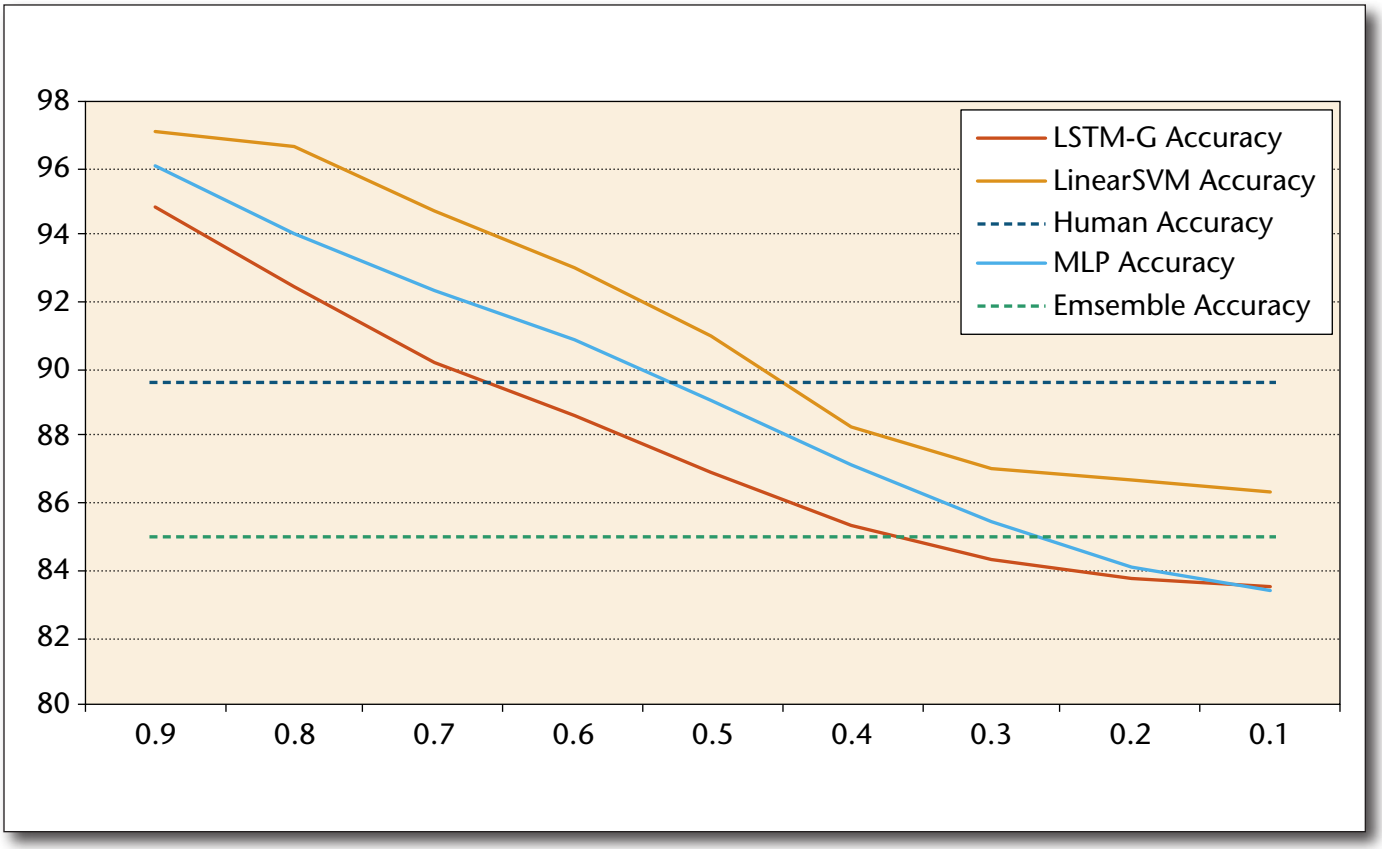

Figure 5. Assignment Accuracy at Different Confidence Thresholds.

and the trained models serialized (using Python object serialization, or pickle) and stored in a cloud-based object store. The model id, data-source name, and creation timestamp are stored separately in a metadata store. The requests for training are handled sequentially. The end point for *classify* can handle multiple requests in parallel. The automatic-rule-mining script is also run periodically and the new rules (if any) are stored in the rule engine.

There is only one component of the assignment engine that needs active maintenance — the rule engine. The maintenance activity involves active supervision of the newly mined rules and retaining only the valid ones in the system. The rule verification is handled by SMEs at the helpdesk and requires about two hours of manual effort per week. The scripts for generation of misclassification reports, triggering retraining and the entire

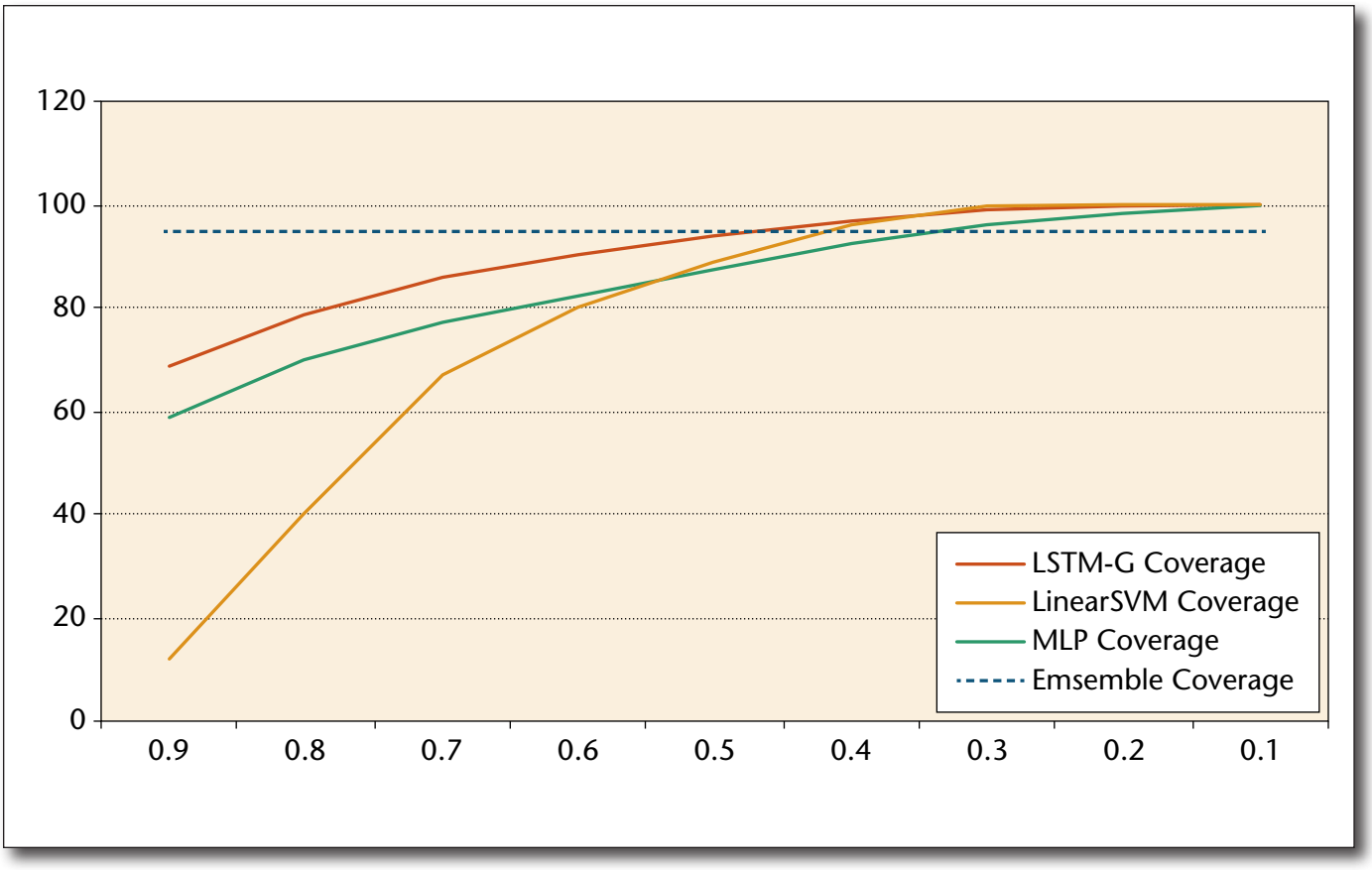

Figure 6. Assignment Coverage at Different Confidence Thresholds.

assignment engine code, are maintained by a team of two developers (working for five hours per week per account).

Evaluation

This section enumerates the results of evaluation of our system. For evaluation, we have used real datasets from three major helpdesk-service-provider accounts. The client accounts are from two different domains — Telecom and Supply-Chain/Logistics. To preserve client confidentiality, we refer to these datasets as dataset A, dataset B, and dataset C, respectively. The datasets were divided into training and test sets with a 90:10 split, and we used 10-fold cross validation on the datasets. All our experiments were run on a NVIDIA Tesla K80 GPU cluster with four CUDA-enabled nodes. We used the open-source machine-learning libraries Python scikit-learn,[3] PyTorch,[4] and Keras[5] for our experiments. The deployed system is similar to our experimental setup, but not identical. The numbers in production may vary slightly. For confidentiality reasons, we cannot reveal exact details of the production setup and accuracy results. The dataset statistics as well as the final accuracy numbers achieved by our system are described in table 4.

Human Accuracy versus Assignment Engine Accuracy

We next look at the optimal selection of algorithms that maximize accuracy and coverage. It is important to note that, for business purposes, the algorithms need to have at least human-level accuracy along with reasonably high coverage. To compute human accuracy, we mined audit logs of the ticketing systems. Our experiments reveal that, across all datasets, the accuracy achieved by human agents is about eighty-five percent. Therefore, we select the confidence threshold such that the expected accuracy of prediction is at least eighty-five percent. This ensures that the selected classifiers operate at least at human-level efficiency. Figures 5 and 6 show the performance of the best three algorithms at different confidence levels (ranging from 0.1 to 0.9). For dataset C, a combination of linear SVM (confidence ≥ 0.5) and MLP (confidence ≥ 0.6) gave a slightly higher accuracy (89.61 percent) than that of generalized LSTM (confidence ≥ 0.5) and linear SVM (X_{acc} = 88.38 percent), although the individual accuracy was marginally higher for generalized LSTM compared with MLP. For this reason, and for other practical considerations such as memory and central-processing-unit constraints as well as training time, our deployment in production uses an ensemble of

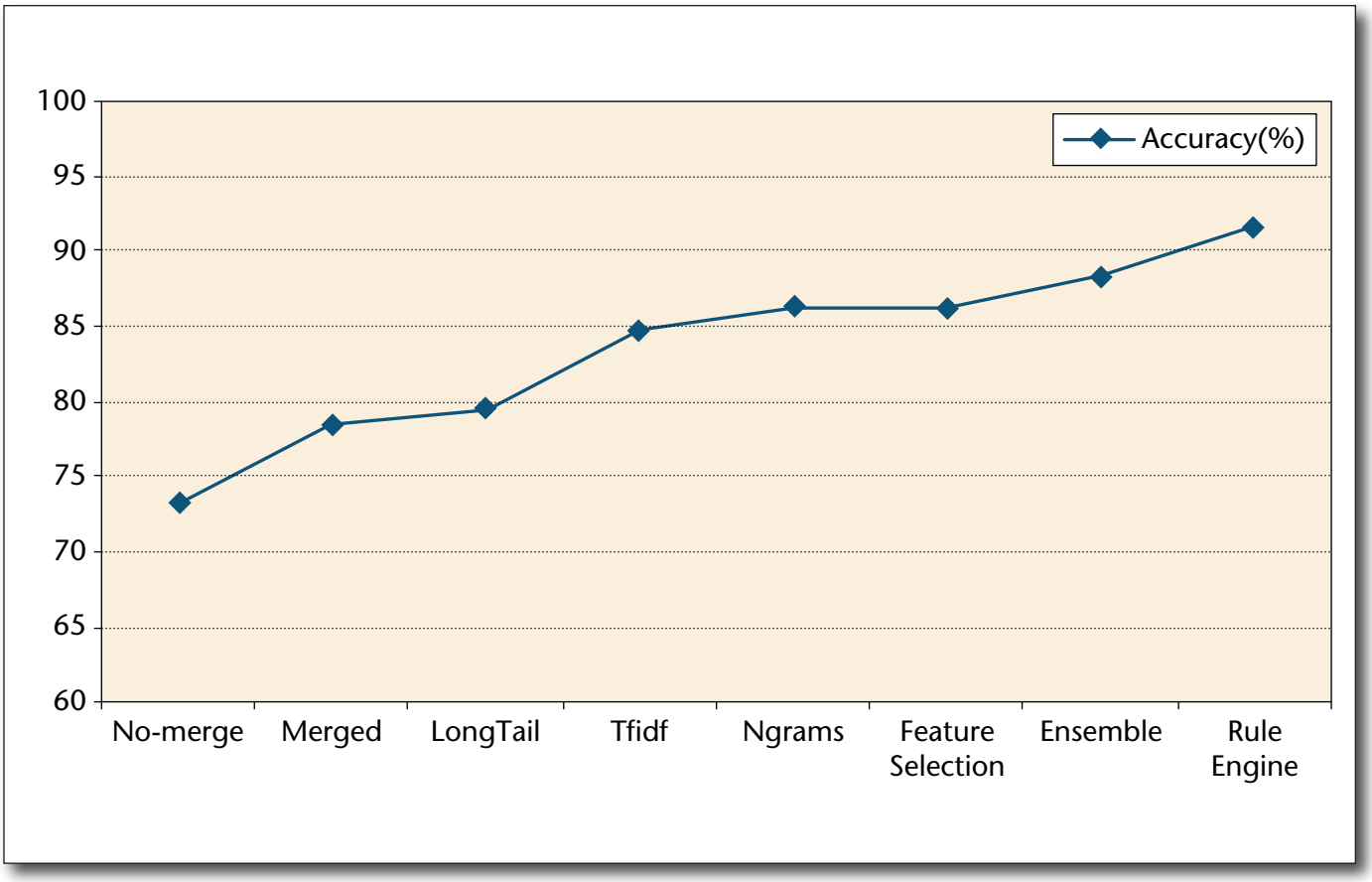

Figure 7. Effect of Different Optimization Techniques on Classification Accuracy.

linear SVM and MLP. For the other two datasets, SVM and MLP were the clear winners.

Rule Mining Accuracy

Table 3 shows the results for rule mining for a particular resolver group in our largest dataset, dataset C. These rules are extracted specifically to resolve the ambiguity among the commonly confused resolver groups R12, R31, and R189. For this experiment, we used a random-forest model with a bagging strategy having the parameters *number of trees* = 20 and *sample size* = 0.2. We set *support* and *confidence* thresholds to ninety percent and 0.9, respectively.

It is important to note that Gini-based rule mining was able to detect each and every rule (3/3) for this resolver group that were earlier predicted by the SMEs, giving a recall value of 100 percent. On the other hand, there were two rules (rules 4 and 9 in table 3) that were not detected by the SMEs, but were found to be valid rules for our dataset. Specifically, rule 4 in the table is a complex rule containing three separate phrases that were correctly mined by our system. These kinds of rules are extremely difficult to create manually. This underscores the usefulness of our rule-mining method. As expected, there were a few false positives — rules that were identified by the Gini method, but found to not be valid. Overall, for this resolver group, the observed precision was 45.45 percent (5:11).

At the same time, it must be remembered that verifying the validity of a rule is much easier for an SME than coming up with the rules from scratch. As such, our method of automatic rule mining reduces the manual labor of SMEs to a large extent.

The overall accuracy of the assignment engine is significantly improved by the use of these SME-validated rules. Experimental results show that the classification accuracy of the long-tail classes can improve by as much as thirty percent with the rule engine. Some of the confused classes in the short head can also be predicted with higher accuracy using the rule engine shown in the paper by Mandal et al. (2019).

Performance and Scalability

Figures 7, 8, and 9 demonstrate the impact of the different training optimization techniques on the accuracy of prediction, the training time, and the model size of SVM. These charts are shown for only the largest dataset (dataset C), but the trend is fairly similar across other datasets as well. The results can be summarized as follows.

Merging coupled with long-tail optimization brings down the number of resolver groups by about

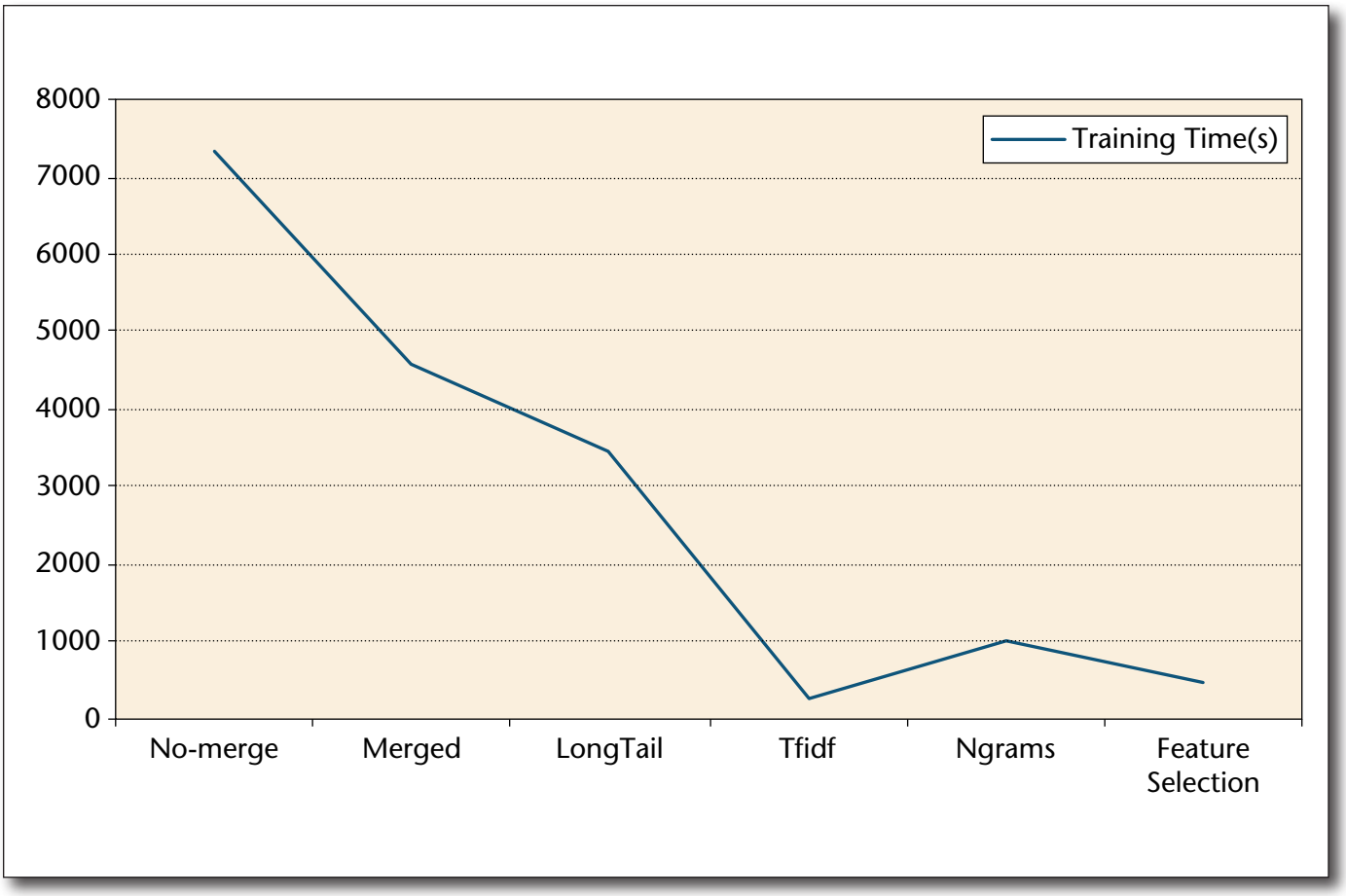

Figure 8. Effect of Different Optimization Techniques on Training Time.

eighty-four percent (438:72) along with a forty-percent reduction in training time, twenty-seven percent reduction in model size, and five-percent improvement in accuracy. This demonstrates the effectiveness of the long-tail approach. We also found that using the tf-idf approach with bigrams proved to be effective across most machine-learning algorithms, including SVM. To optimize on the number of features and model size, we used the tf-idf approach with parameters maximum df = 0.8 and minimum df = 5. Finally, we used the chi-square statistic (with best k features) to further reduce the model size. Overall, using these approaches, we were able to achieve a ninety-nine percent reduction in model size, 93.5 percent reduction in training time, and a thirteen-percent increase in accuracy for dataset C while keeping the coverage constant at about ninety-eight percent. We also achieved significant speedup in classification time using asynchronous requests and batching. Figure 10 shows the peak per-day ticket volumes recorded during the entire training period for each account. Our system was able to achieve a runtime throughput of about 286 requests per second, which is equivalent to about 1,500 times the peak hourly volume (696 requests per hour), as shown in the figure.

Business Impact

Automated assignment of helpdesk tickets results in considerable saving in human effort for large companies having clients across geographies. It reduces the time taken for ticket assignment and, at the same time, minimizes human error. This enables the companies to focus more on innovation and core business needs. Based on our results as summarized in table 3, we give an estimate of human-effort saving across all accounts. Assuming that a human agent takes about three minutes to read and assign each ticket, the net savings (in minutes) for an account can be calculated as: $S_i = T \times E_{cov} \times 3$. Summed across all three accounts, this gives a total saving of 52,380 hours per annum. To date, our system has served more than two million tickets in production across the deployed accounts.

Observations

There are a few important takeaways from our evaluation results above. The most important and useful observation is that our assignment engine performs better than all traditional machine-learning and deep-learning algorithms. This indicates that simple machine-learning algorithms like SVM and MLP fare better than more computationally expensive

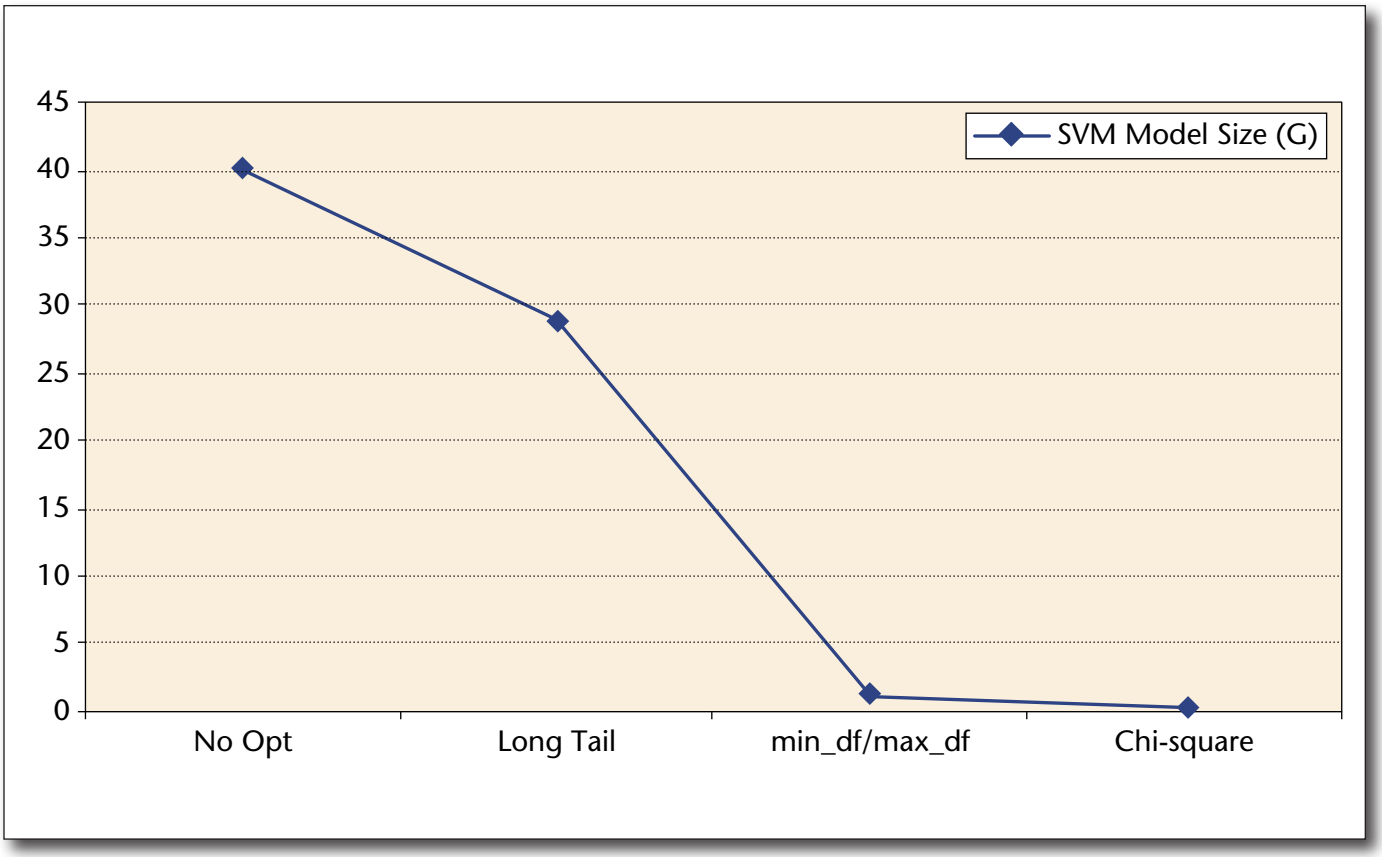

Figure 9. Model Size Optimization (SVM).

deep-learning algorithms in the task of helpdesk email assignment. This result is somewhat surprising and unexpected, but is very significant from a product development standpoint as these algorithms are easy to implement, require minimal computational resources, and still provide better performance at runtime. However, we also observed that with very large training data (more than five million), LSTM starts outperforming MLP. As such, if we have a large dataset and infrastructure is not a concern, then we can include generalized LSTM in the ensemble.

Another key observation is that we need to focus more on resolver-group level preprocessing rather than any kind of text preprocessing. Approaches like merging resolver-group training data and having separate classifiers for short head and long tail are actually much more effective than techniques such as stopword removal or extractive summarization of text. However, data augmentation techniques such as paraphrasing and sample duplication resulted in small improvements in long-tail accuracy for some of the datasets. The rule-mining technique is designed to have high interpretability and high recall. This ensures that SMEs can easily verify whether a rule is correct and discard if needed. Finally, our results (shown in table 4) clearly indicate the importance of the rule engine. It increases the overall accuracy and coverage of the system, and it ensures business continuity. We also see a clear benefit of having automatic rule mining in saving human labor.

Conclusion and Future Work

In this article, we have proposed an end to end ticket dispatch automation system that encapsulates the full AI life cycle. The proposed system achieves human-level accuracy and has already been deployed successfully for six customers in production.

However, there is still some scope for improvement of the system. Firstly, the proposed rule engine is still not completely automated. Although we automatically extract rules, it still requires some amount of manual supervision to verify whether the rule is correct. The challenge here will be to make this process completely automatic without compromising the necessity for continuity in business operations. Secondly, we need to look at email attachments in addition to text for classification. In a lot of cases, users will only send a screenshot or log snippet with hardly any text. These cases cannot be handled by the present system. Alternate modalities such as voice and video can also be considered in future. Thirdly, for some information-technology operations systems, the tickets cannot be understood properly without looking at other artifacts, such as runtime execution logs. For these systems, we need to combine our ticket understanding with

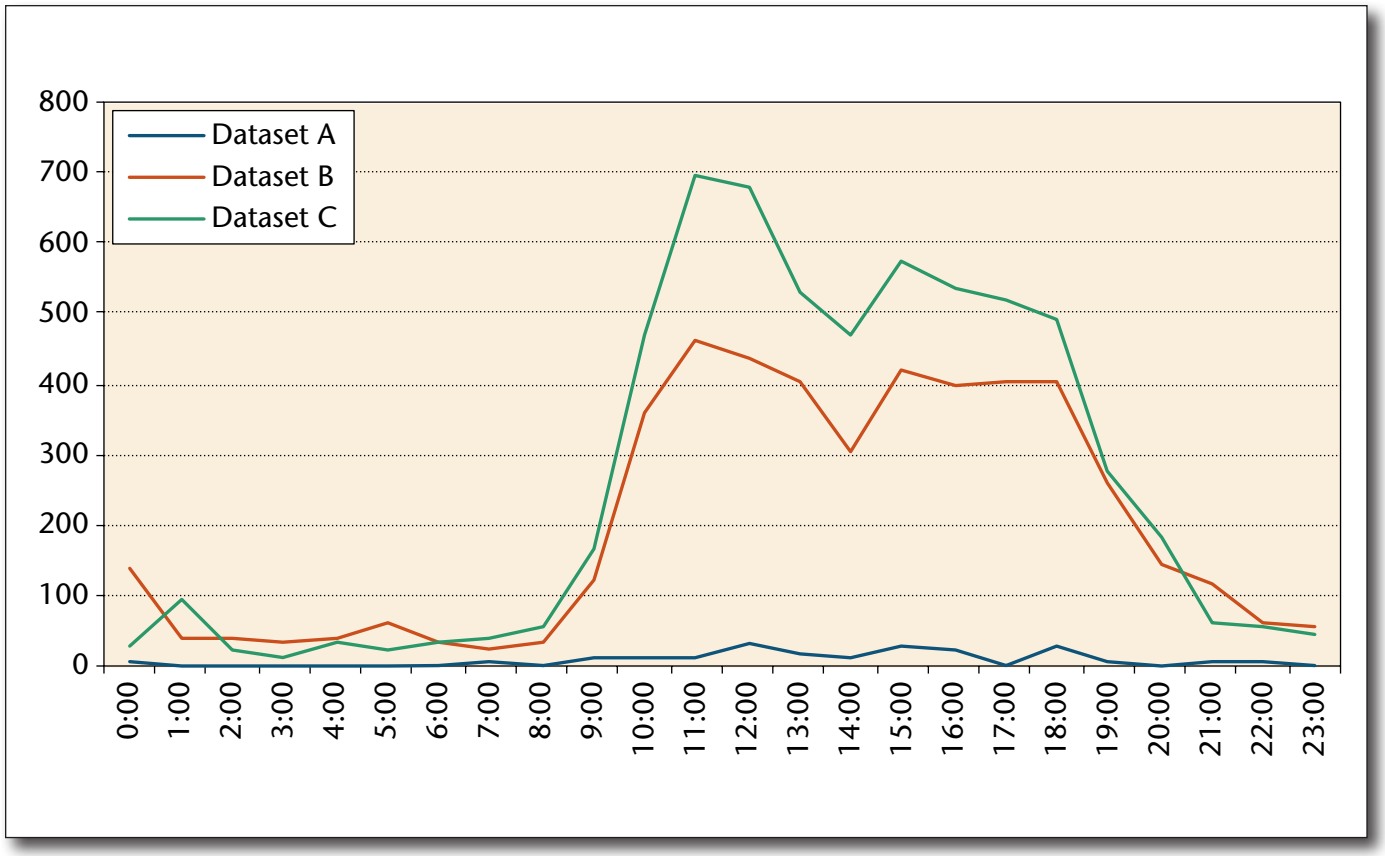

Figure 10. Ticket Volumes for a Single Day.

log-mining techniques for accurate dispatch. We are currently working on implementing such a system.

Notes

1. As referenced in the book: A Method for Stochastic Optimization, Diederik P. Kingma and Jimmy Ba, 2014, eprint={1412.6980}, archivePrefix={arXiv}, primaryClass={cs.LG}
2. Resolver group names (columns C_F and C_E) are anonymized to preserve confidentiality.
3. See the article, "Machine learning in Python" (2011), by Pedregosa et al., in the Journal of Machine Learning Research, volume 12, October, 2825-2830
4. "An Imperative Style, High-Performance Deep Learning Library", by Paszke et al. (2019) in the book "Advances in Neural Information Processing Systems", edited by H. Wallach et al., 8024–8035, published by Curran Associates, Inc., url = http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf
5. Keras, written by Chollet et al. (2015) published by GitHub, url - https://github.com/fchollet/keras

References

Agarwal, S.; Aggarwal, V.; Akula, A. R.; Dasgupta, G. B.; and Sridhara, G. 2017. Automatic Problem Extraction and Analysis From Unstructured Text in IT Tickets. *IBM Journal of Research and Development* 61(1): 41–52. doi.org/10.1147/JRD.2016.2629318.

Agarwal, S.; Sindhgatta, R.; and Sengupta, B. 2012. SmartDispatch: Enabling Efficient Ticket Dispatch in an It Service Environment. In *Proceedings of the 14th Association for Computing Machinery (ACM) Special Interest Group on Knowledge Discovery and Data Mining (SIGKDD) International Conference*. New York, NY: Association for Computing Machinery. doi.org/10.1145/2339530.2339744.

Bao, Y.; Wu, M.; Chang, S.; and Barzilay, R. 2019. *Few-Shot Text Classification with Distributional Signatures*. arXiv:1908.06039. Ithaca, NY: Cornell University Library.

D'Ambrosio, A., and Tutore, V. A. 2011. Conditional Classification Trees by Weighting the Gini Impurity Measure. In New Perspectives in Statistical Modeling and Data Analysis, S. Ingrassia, R. R. Rocci, and M. Vichi, eds., 273–80. Berlin, Germany: Springer. doi.org/10.1007/978-3-642-11363-5_31.

Dasgupta, G.; Nayak, T. K.; Akula, A. R.; Agarwal, S.; and Nadgowda, S. J. 2014. Towards Auto-Remediation in Services Delivery: Context-Based Classification of Noisy and Unstructured Tickets. In Service-Oriented Computing: 12th International Conference, Volume 8831, Lecture Notes in Computer Science. Berlin, Germany: Springer.

De Falco, I. D.; Cioppa, A. D.; and Tarantino, E. 2002. Discovering Interesting Classification Rules with Genetic Programming. *Applied Soft Computing* 1(4): 257–69. doi.org/10.1016/S1568-4946(01)00024-2.

Di Lucca, G. 2002. An Approach to Classify Software Maintenance Requests. In 18th International Conference on Software Maintenance (ICSM 2002). Piscataway, NJ: Institute of Electrical and Electronics Engineers (IEEE). doi.org/10.1109/ICSM.2002.1167756.

Fu, Y.; Zhu, X.; and Li, B. 2012. A Survey on Instance Selection for Active Learning.

Knowledge and Information Systems 35(2): 249–83. doi.org/10.1007/s10115-012-0507-8.

Goodfellow, I.; Bengio, Y.; and Courville, A. 2016. *Deep Learning*. Cambridge, MA: The MIT Press.

Kadar, C.; Wiesmann, D.; Iria, J.; Husemann, D.; and Lucic, M. 2011. Automatic Classification of Change Requests for Improved IT Service Quality. In Service Research and Innovation Institute Global Conference. Piscataway, NJ: Institute of Electrical and Electronics Engineers (IEEE). doi.org/10.1109/SRII.2011.95.

Kuncheva, L. I. 2004. *Combining Pattern Classifiers: Methods and Algorithms*. New York: Wiley-Interscience. doi.org/10.1002/0471660264.

Mandal A., Malhotra N., Agarwal S., Ray A., Sridhara G. (2018) Cognitive System to Achieve Human-Level Accuracy in Automated Assignment of Helpdesk Email Tickets. In: Pahl C., Vukovic M., Yin J., Yu Q. (eds) Service-Oriented Computing. ICSOC 2018. Lecture Notes in Computer Science, vol 11236. Springer, Cham.

Mandal, A.; Malhotra, N.; Agarwal, S.; Ray, A.; and Sridhara, G. 2019. Automated Dispatch of Helpdesk Email Tickets: Pushing the Limits With AI. *Proceedings of the AAAI Conference on Artificial Intelligence*. 33(1): IAAI-19. doi.org/10.1609/aaai.v33i01.3301938a.

Manning, C.; Surdeanu, M.; Bauer, J.; Finkel, J.; Bethard, S.; and McClosky, D. 2014. The Stanford Corenlp Natural Language Processing Toolkit. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (System Demonstrations)*. Stroudsberg, PA: Association for Computational Linguistics. doi.org/10.3115/v1/P14-5010.

Mitchell, T. M. 1997. *Machine Learning*. New York, NY: McGraw-Hill.

Mueller, J., and Thyagarajan, A. 2016. Siamese Recurrent Architectures for Learning Sentence Similarity. In *Proceedings of the Thirtieth Association for the Advancement of Artificial Intelligence (AAAI) Conference,* 2786–92. Palo Alto, CA: Association for the Advancement of Artificial Intelligence (AAAI) Press.

Parvin, H.; Bose, A.; and Van Oyen, M. P. 2009. Priority-Based Routing with Strict Deadlines and Server Flexibility under Uncertainty. In *Proceedings of the 2009 Winter Simulation Conference (WSC) 2009*. New York, NY: Association for Computing Machinery (ACM). doi.org/10.1109/WSC.2009.5429275.

Potharaju, R.; Jain, N.; and Nita-Rotaru, C. 2013. Juggling The Jigsaw: Towards Automated Problem Inference From Network Trouble Tickets. Paper presented at the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 2013), Lombard, IL, April 2–5. www.usenix.org/conference/nsdi13/technical-sessions/presentation/potharaju.

Reyes, O.; Altalhi, A.; and Ventura, S. 2018. Statistical Comparisons of Active Learning Strategies Over Multiple Datasets. *Knowledge-Based Systems* 145(4): 275–88.

Schein, A. I., and Ungar, L. H. 2007. Active Learning for Logistic Regression: An Evaluation. *Machine Learning* 68(3): 235–65. doi.org/10.1007/s10994-007-5019-5.

Settles, B. 2009. Active Learning Literature Survey. Computer Sciences Technical Report 1648, University of Wisconsin–Madison. www.burrsettles.com/pub/settles.activelearning.pdf.

Shang, J.; Tong, W.; Peng, J.; and Han, J. 2016. Dpclass: An Effective but Concise Discriminative Patterns-Based Classification Framework. In *Proceedings of the 2016 SIAM International Conference on Data Mining*, 567–5. Philadelphia, PA: Society for Industrial and Applied Mathematics. doi.org/10.1137/1.9781611974348.64.

Shao, Q.; Chen, Y.; Tao, S.; Yan, X.; and Anerousis, N. 2008a. Easyticket: A Ticket Routing Recommendation Engine for Enterprise Problem Resolution. In 34th International Conference on Very Large Data Bases (VLDB). New York, NY: Association for Computing Machinery. doi.org/10.14778/1454159.1454193.

Shao, Q.; Chen, Y.; Tao, S.; Yan, X.; and Anerousis, N. 2008b. Efficient Ticket Routing by Resolution Sequence Mining. In *Proceedings of the 14th Association for Computing Machinery (ACM) Special Interest Group on Knowledge Discovery and Data Mining (SIGKDD) International Conference*. New York, NY: Association for Computing Machinery. doi.org/10.1145/1401890.1401964.

Zeng, C.; Zhou, W.; Li, T.; Shwartz, L.; and Grabarnik, G. Y. 2017. Knowledge Guided Hierarchical Multi-Label Classification Over Ticket Data. *IEEE eTransactions on Network and Service Management* 14(2): 246–60. doi.org/10.1109/TNSM.2017.2668363.

Zhang, C., and Zhang, S., editors. 2002. *Association Rule Mining: Models and Algorithms*. Berlin, Germany: Springer. doi.org/10.1007/3-540-46027-6.

Shivali Agarwal is a senior researcher at IBM Research, Bengaluru, India. She holds a PhD in Computer Science in the area of Programming Languages from the Tata Institute of Fundamental Research, Mumbai. She joined the IBM Research Lab, Delhi, in the Programming Languages and Software Engineering group in 2008 soon after receiving her PhD. A year later, she joined the group working on services research and has since made significant contributions to optimize and automate the information-technology operations for IBM's service delivery business. She also has more than twenty-four conference publications, book chapters, and journal papers, and holds sixteen patents in her area of work. Currently, she is working in the hybrid-cloud department, where she is leading the research on applying AI and software engineering techniques for the modernization of enterprise legacy applications.

Jayachandu Bandlamudi is a staff engineer at IBM Research, India with a Master's degree from the University of Illinois at Urbana-Champaign. He is highly familiar with building scalable AI systems. His research interests are in data mining, machine learning, and natural language processing. In his spare time, he likes to fix motorcycles and ride in the countryside.

Atri Mandal is a senior research engineer at IBM Research, India. Mandal received his Masters degree in Information and Computer Science from the University of California, Irvine. He has over seventeen years of experience in software development and research and has worked on many interesting and complex research problems in the areas of log analytics, distributed data mining, hybrid-cloud, machine learning, and DevOps.

Anupama Ray is a research scientist at IBM Research, India. She joined IBM in January 2017 after receiving her PhD in Electrical Engineering from the Indian Institute of Technology, Delhi. Her research focuses on developing and applying machine learning algorithms for neuro-linguistic programming, natural language generation, and multimodal applications.

Giriprasad Sridhara is a research software engineer with IBM Research, India. He has a PhD in computer science from the University of Delaware (USA). His research interests include software engineering, machine learning, deep learning, and natural language processing. He has over sixteen publications in leading international conferences and holds sixteen patents.