The Thirty-First AAAI Conference on Innovative Applications of Artificial Intelligence (IAAI-19)

Automated Dispatch of Helpdesk Email Tickets: Pushing the Limits with AI

Atri Mandal,[1] **Nikhil Malhotra,**[2] **Shivali Agarwal,**[1] **Anupama Ray,**[1] **Giriprasad Sridhara**[1]
[1]IBM Research AI, Bengaluru, India
[2]IBM Global Technology Services, Bengaluru, India
{atri.mandal,nikhilmal,shivaaga,anupamar,girisrid}@in.ibm.com

Abstract

Ticket assignment/dispatch is a crucial part of service delivery business with lot of scope for automation and optimization. In this paper, we present an end-to-end automated helpdesk email ticket assignment system, which is also offered as a service. The objective of the system is to determine the nature of the problem mentioned in an incoming email ticket and then automatically dispatch it to an appropriate resolver group (or team) for resolution.

The proposed system uses an ensemble classifier augmented with a configurable rule engine. While design of a classifier that is accurate is one of the main challenges, we also need to address the need of designing a system that is robust and adaptive to changing business needs. We discuss some of the main design challenges associated with email ticket assignment automation and how we solve them. The design decisions for our system are driven by high accuracy, coverage, business continuity, scalability and optimal usage of computational resources.

Our system has been deployed in production of three major service providers and currently assigning over 90,000 emails per month, on an average, with an accuracy close to 90% and covering at least 90% of email tickets. This translates to achieving human-level accuracy and results in a net saving of more than 50000 man-hours of effort per annum. Till date, our deployed system has already served more than 700,000 tickets in production.

1 Introduction

The landscape of modern IT service delivery is changing with increased focus on automation and optimization. Most IT vendors today, have service platforms aimed towards 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/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 i.e. 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). Several factors make the dispatcher's job challenging viz. need for in-depth knowledge of the roles and responsibilities of various groups, heterogeneous and informal nature of email text and 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, that is, the dispatching of a ticket to an appropriate resolver group based on the problem description. A number of different approaches have been proposed for automating ticket dispatch (Agarwal, Sindhgatta, and Sengupta 2012)(Shao et al. 2008a)(Shao et al. 2008b)(Parvin, Bose, and Van Oyen 2009). The tickets may be raised in different manners such as through voice, web forms or e-mails 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. Although email assignment may look like a simple text classification problem at first glance it becomes quite complex and challenging when considered at industry scale.

Firstly, for large companies the number of resolver groups can be quite large - of the order of 500 in some cases. Many of these resolver groups cater to overlapping problems which can be disambiguated only with domain specific knowledge. Secondly in most businesses the helpdesk teams themselves undergo constant changes for better efficiency and productivity leading to resolver groups being split, merged or renamed. All of these changes inevitably impact the accuracy of machine learning classifiers. Thirdly the problems assigned to resolver groups themselves slowly change over time. As such a once-trained model becomes outdated over time as it cannot effectively assign tickets mentioning new problems or old problems with a different terminology (Mandal et al. 2018).

Main Contributions

In this paper we present a readily deployable end-to-end automatic email dispatch system, which has the following key features:

1. 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. The choice of ensemble is based on the results of comprehensive study performed with various machine learning and deep learning models as presented in section 4.

2. A rule engine to i) handle domain specific content missed by the ensemble classifier and ii) ensure business continuity. The rules are designed to strategically combine with machine learning methods for effective disambiguation of classes. Rules are specified through a customer-independent framework.
3. An effective retraining strategy which ensures that the models are up-to-date with the changes that happen over time in the email utterances as well as resolver group organization.

We present a comprehensive study of the efficacy of different machine-learning and deep-learning algorithms in helpdesk email ticket classification. 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 90% coverage on all the datasets with the proposed system using minimal computational resources.

The remainder of the paper is organized as follows. Section 2 describes the related work. Section 3 gives an overview of the system used for ticket classification and Section 4 discusses the different components of the system. In Section 5 we present our experimental results while we conclude in Section 6.

2 Related Work

Ticket dispatch has been addressed by (Agarwal, Sindhgatta, and Sengupta 2012) using Support Vector Machines and discriminative keyword approach. They propose semi-automated approach based on confidence scores. We have surpassed their work to i) reach human level accuracy using advanced ensemble techniques for automated dispatch, ii) scale it to hundreds of resolver groups and iii) incorporate 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)(Shao et al. 2008b)(Parvin, Bose, and Van Oyen 2009). The work in (Shao et al. 2008b) approaches the problem by mining resolution sequence data and does not access ticket description at all. Its objective is to come up with ticket transfer recommendations given the initial assignment information. The work in (Shao et al. 2008a) mines historical ticket data and develops a probabilistic model of an enterprise social network that represents the functional relationships among various expert groups in ticket routing. Based on this network, the system then provides routing recommendations to new tickets. This work also focuses on ticket transfers between groups (given an initial assignment) without looking at the ticket text content. The work in (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 when the ticket that has

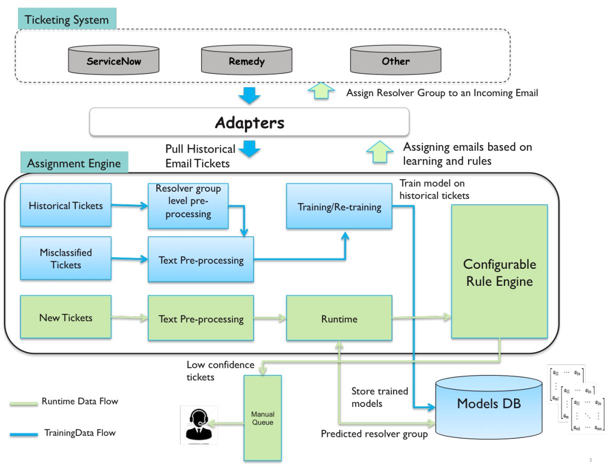


Figure 1: Architecture of the proposed system

been dispatched to a group needs to be assigned to an agent. There are some papers, which 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 8 groups. The work in (Kadar et al. 2011) attempts to classify the incoming change requests into one of the fine-grained activities in a catalog. Some other works (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.

3 System Overview

Figure 1 shows the system architecture along with the data flow diagram. Historical email ticket data is downloaded from the ticketing tool (e.g. Remedy or ServiceNow) using custom-built adapters. The downloaded emails are passed through two stages of pre-processing for data enrichment. The resolver group level pre-processing module uses techniques like resolver group merging, long tail cutoff etc. to reduce the noise in the email data. The training data is further cleaned using text pre-processing methods (Manning et al. 2014). The cleaned email data is then trained using an ensemble of machine learning classifiers and the trained models are stored in a database.

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 the runtime that consists of ensemble classifier and the rule engine. The classification system returns a resolver group along with a confidence score. If the confidence score is above a configured threshold the ticket is routed to the returned resolver group. Otherwise the ticket is assigned back to manual queue for inspection by human agent. The combination of ensemble classifier and rule engine ensures that a high percentage of tickets (more than 90%) are classified automatically by our system with a low error rate.

We will now define key terms used in the rest of the paper. 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 can be defined as:

$$H_{\text{acc}} = \frac{NH_1 \times 100}{N} \; where : N = NH_1 + NH_2$$

In the automated email assignment scenario tickets are assigned primarily by the assignment engine, which combines both ensemble classifier, and rule engine. Let NX be the total number of tickets actually assigned by the ensemble classifier; NX_{corr} be the number of tickets which were predicted by ensemble and for which the resolver group predicted is correct i.e. the ticket was ultimately resolved by the predicted resolver group; NR the number of tickets where the resolver group was predicted by the rule engine; and NR_{corr} the number of tickets which were predicted by rule engine and the predicted resolver group was correct. Then we can define Ensemble Accuracy(X_{acc}),Ensemble Coverage (X_{cov}), Assignment Engine Accuracy (E_{acc}) and Assignment Engine Coverage (E_{cov}) as follows:

$$X_{\text{acc}} = \frac{NX_{\text{corr}} \times 100}{NX}, X_{\text{cov}} = \frac{NX \times 100}{N}$$

$$E_{\text{acc}} = \frac{(NX_{\text{corr}} + NR_{\text{corr}}) \times 100}{NX + NR},$$

$$E_{\text{cov}} = \frac{(NX + NR) \times 100}{N}$$

4 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 etc.) organizes email data into structured fields containing relevant information about the ticket e.g. incident type, creation date, problem description, resolver group etc. 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 (viz. email subject and body) along with the resolver group for training. The steps involved in training data preparation are described below.

Resolver group level pre-processing This type of preprocessing is a one-time effort required during customer onboarding phase. The purpose of this pre-processing is to reduce noise in the training data. We reduce noise and enrich training data for the resolver groups using the following techniques:

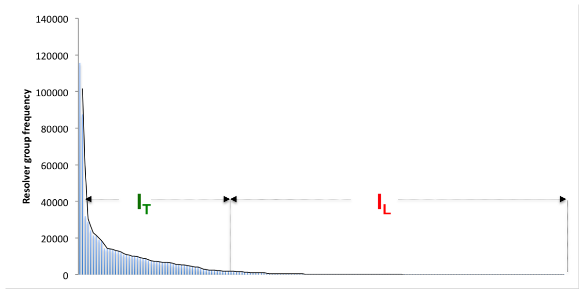


Figure 2: Resolver group frequency distribution showing I_T and I_L for dataset C

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 purpose. **a) Resolver groups with varying escalation levels**: Firstly there are some tickets for which the final assigned group is one of various escalation levels (e.g. Tier1, Tier2 etc.) of the same resolver group. However the helpdesk often assigns directly to only one of these escalation levels. Escalations to higher levels usually happen after additional information is sought from the customer. But it is enough to assign the ticket initially to the default escalation level. **b) Region specific resolver groups**: Sometimes tickets are meant to be resolved based on the user location. In that case the tickets are assigned to a particular region and is resolved by an agent locally. To merge these resolver groups we create a new label and assign all the region specific tickets to this label. After the initial assignment, the correct resolver group can be inferred based on some other fields in the ticket e.g. end-user location or originator email-id etc. using the rule engine.

Long tail cutoff We observed that in most of the datasets there are a large number of resolver groups with very few samples. If we plot a histogram of frequencies these groups will constitute more than 80% of the resolver groups but less than 10% of training data. Our studies indicate that, if the long tail is included in training, the overall accuracy of classification goes down along with a significant increase in training time and model size. By restricting the number of resolver groups in training we reduce noise significantly and also avoid class imbalance, resulting in increase of accuracy. Additionally, the resolver groups, which fall in the long tail, can often be predicted better using the rule engine (refer Table 5) and using some augmentation techniques. As such our strategy was to divide the downloaded historical data into 2 parts viz. $I_H = I_T + I_L$ where I_H is the complete data downloaded for training, I_T is the data used for training classifiers and I_L is the long tail. Resolver groups belonging to I_T will be classified using trained models while those belonging to I_L will be handled exclusively by the rule engine. In our system we use the above strategy to retain at least 98% of data while cutting down the resolver group count to less than 20%. Figure 2 shows the resolver group frequency distribution for one of our datasets (dataset C).The evaluation results are shown in Section 5.

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. 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. Table 1 and Table 2 show the impact of various traditional machine learning models (Mitchell 1997) and deep neural network models that were used. In order to improve accuracy and coverage of the overall service, we use an ensemble (Kuncheva 2004). Each pair of models were combined, and the final ensemble classifier was chosen based on the accuracy and coverage. As explained in Section 4, rule engine is important to handle the long tail in class distribution and the final chosen ensemble classifier in combination with the rule engine forms the classification module of the service.

Training the classifiers We convert the training data samples into word vector representation before applying machine-learning algorithms. We observed that using tf-idf representation increased the accuracy of traditional machine learning algorithms for all datasets by at least 3-4%. Another observation was that using bigrams also improved the accuracy for some datasets. Intuitively we can argue that this is so because some bigrams like 'account creation', 'account deletion', 'password reset' etc. are useful indicators in deciding the resolver group. The hyperparameters were chosen experimentally over 10-fold cross-validation on the datasets.

However, for learning deep neural networks, tf-idf representation being extremely sparse is not useful. Distributed representation of text creates a dense, low-dimensional representation and is perhaps the main reason why deep learning saw major breakthroughs in natural language processing (Mikolov et al. 2013). There are primarily two methods of learning the word embeddings: one in which word embeddings are learnt while training the neural network; and second using pretrained word vectors. We experimented with both methods for classification (models learning word embeddings being referred to CNN-WE, LSTM-WE, and CNN-LSTM-WE in Table 2), and pretrained word-vector representations (100-d GloVe vectors) (Pennington, Socher, and Manning 2014) referred to as CNN-G, LSTM-G and CNN-LSTM-G. A word embedding layer learnt on the input domain is better suited to the domain but requires a lot of training data and increases overall training time. Because of diverse dataset sizes to be handled by our service, we use pretrained 100-dimensional GloVe vectors trained on 6B tokens. We chose GloVe over word2vec as GloVe utilizes local context-based learning (like word2vec) along with global text statistics (as in classical vector space models such as Latent Semantic Analysis) (Pennington, Socher, and Manning 2014).

Rule Engine

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

1. 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.

2. Resolver groups belonging to the long tail: As discussed before, in most datasets 20% of the classes account for more than 90% of the tickets. The learning models are not trained on the remaining 80% classes, which account for less than 10% of the overall ticket data, to reduce noise and avoid class imbalance. Although this improves classification accuracy and time, the model will never learn to predict these classes. For all these classes the rule engine is essential.

3. Presence of resolver groups with similar or overlapping email format: 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 (Table 5). Rule-engine addresses this issue for the confusing classes to override the decision of the machine learning classifier.

Design of rule engine The rule engine is designed to have a customer independent framework for rule specification and is easy to configure using an user interface. The user interface allows specification of rules using values of certain ticket parameters e.g. email subject, description etc. and output of the ensemble classifier. The rule engine can override the output of the ensemble classifier in certain cases. Each rule in the rule engine can be generically expressed by the following equation:

$$\phi_1(f_1, v_1) \wedge \phi_2(f_2, v_2) \wedge \phi_3(f_3, v_3) \wedge (C_\text{E} = R) \Rightarrow C_\text{F}$$

where ϕ_i is a user-defined boolean function which depends on the ticket field f_i and a value v_i, (which can be a keyword, phrase or regular expression derived from the text content of f_i), C_E denotes the ensemble classifier prediction, 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 3 for better understanding. (The value X in the table implies DON'T CARE) [1]. The first rule is used for renaming cases, (ZZZ-SDK renamed to ZZZ-GB-SDK) meaning if the ensemble predicted class is ZZZ-SDK the final prediction will be ZZZ-GB-SDK

[1]Resolver group names (Columns C_F and C_E) are anonymized to preserve confidentiality

Table 1: Comparison of various Machine Learning Algorithms w.r.t. Accuracy and Training Time

		LinearSVM	KNN	LR	m-NB	RF	Adaboost	Gradient Boosting
Dataset A	Accuracy(%)	87.3	80.12	79.48	72.68	81.41	31.5	75.6
	Train-time(s)	7.8	260.5	43	17.3	363.75	4561	8612
Dataset B	Accuracy(%)	83.42	72.58	79.95	64.19	74.91	32.98	65.1
	Train-time(s)	76.12	2218.65	404.05	22.18	7190.16	332.97	95320.1
Dataset C	Accuracy(%)	86.339	67.57	84.29	63.97	76.99	30.43	61.47
	Train-time(s)	1001.06	1921.7	2992	167.5	20799.6	1288.63	126960

Table 2: Comparison of various Deep neural networks w.r.t. Accuracy and Training Time

		MLP	CNN-WE	LSTM-WE	CNN-G	LSTM-G	CNN-LSTM-G
Dataset A	Accuracy(%)	85.8	74	76.94	74.01	71.64	73.24
	Train-time(s)	184.12	183.75	5546.6	160.56	9833.7	1844.8
Dataset B	Accuracy(%)	80.87	77.75	79.35	76.23	80.37	77.7
	Train-time(s)	10858.15	8680.35	86651.57	1926	89280.94	23229.47
Dataset C	Accuracy(%)	83.3	82.67	78.14	81.27	83.51	81.48
	Train-time(s)	2779	10519.99	90149.9	6557.12	687483	60128.5

Table 3: Sample rules

f_1	f_2	f_3	C_E	C_F
X	X	X	ZZZ-SDK	ZZZ-GB-SDK
”HSS”	X	X	ZZZ-MM	ZZZ-HSS-MM
”replenish team”	”xyz”	”abc@xyz”	X	ZZZ-REPLEN
”HSS”AND”EWM”	X	X	X	ZZZ-HSS-WM

irrespective of the field values. The second rule is used for commonly confused sets of resolver groups. Here, ensemble is used to determine the commonly confused set [ZZZ-MM, ZZZ-HSS-MM], then 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 override machine-learning prediction. The fourth rule is similar to third except that it uses 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 two and four. Rules two and four have a common match criteria for field f_1 (viz. the keyword HSS). In these cases we match the more restrictive rule first to minimize the possibility of false positives.

Model Selection and Ticket Dispatch

The email ticket dispatcher actually 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. If the confidence score of the final result is below the configured threshold the ticket is assigned to the manual queue. The pseudo code for the dispatcher algorithm is given in Algorithm 1. In the algorithm, M1 and M2 (with confidence cutoffs $M1\_cutoff$ and $M2\_cutoff$ respectively) denote the ensemble classifier models chosen based on results shown in Table 1 and Table 2. M1 is the model having higher precision. $InvokeClassifier$ invokes

```
Input  : Email-Text, M1_cutoff, M2_cutoff
Output: result = <resolver_grp, confidence>
result = [None, 0.0]
<$M1\_resolver\_group$, $M1\_confidence$> =
  InvokeClassifier(M1, Email-Text)
if $M1\_confidence \geq M1\_cutoff$ then
    result = <$M1\_resolver\_group$, $M1\_confidence$>
end
else
    <$M2\_resolver\_group$, $M2\_confidence$> =
      InvokeClassifier(M2, Email-Text)
    if $M2\_confidence \geq M2\_cutoff$ then
        result = <$M2\_resolver\_group$,
          $M2\_confidence$>
    end
    else if $M1\_resolver\_group$ ==
      $M2\_resolver\_group$ then
        result = <$M2\_resolver\_group$,
          $min(M1\_cutoff, M2\_cutoff)$>
end
result =
  $ApplyRuleEngine(f_1, f_2, f_3, result.resolver\_grp)$
if $result.resolver\_grp \neq None$ then
    /* Assign ticket to $result.resolver\_grp$
        */
    return $result$
end
else
    /* Assign ticket to manual queue
        */
    return [None, 0.0]
```

Algorithm 1: Dispatch Algorithm

a machine-learning classifier and returns the best prediction for resolver group along with the corresponding confidence value. $ApplyRuleEngine$ invokes the rule engine on the results obtained from the ensemble classifier.

Retraining

We have already discussed how 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 e.g. subtle changes in email utterances over time. These changes can happen due to various factors like resolver groups taking on new problems, increased automation etc. To capture these changes effectively we use a sliding window based retraining strategy. The sliding window ensures that training data is refreshed periodically and the most recent changes are retained, so the classifiers remain up-to-date.

However along with recent data we also need to keep learning from past mistakes. As such in addition to the sliding window data we also retain misclassified data from previous periods. In our deployment environment the misclassifications from each slide interval are not directly added for retraining. They are first supervised by a human agent for false negatives. A false negative occurs when the assignment engine predicts correctly but the ticket was transferred due to some other reason not mentioned in the email. This additional supervision is necessary to ensure high quality of training data.

The length (W) of the sliding window is determined such that the training data is 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. For production deployment we used t=7 days and W=90 days. However these values can vary depending on the nature of the dataset.

Deployment and Maintenance

The automated assignment engine is currently deployed as a single tenant service hosted in a Kubernetes cluster (refer to Figure 3). There are three REST APIs to achieve the full functionality, namely i) /train - The call to train method is asynchronous and it returns a training-id, ii) /status - To check the status of a training in progress and iii) /classify - The call to the classifier to predict the resolver group. The ensemble classifier is run periodically on a GPU cluster and the trained models stored in a cloud based object store. The model id, datasource 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.

There are mainly two components of the assignment engine which need active maintenance viz. Retraining and Rule Engine. The maintenance activity involves active monitoring of the misclassified tickets and taking an action based on whether the ticket was confused with a class in I_T or I_L. In the former case, the misclassified sample is added to the dataset for retraining; in the latter case rule engine modifications may be needed like adding a rule to the rule engine to cover the misclassification if possible. The rule engine modifications are handled by subject matter experts in

Table 4: Dataset size, usage and results

	Dataset A	**Dataset B**	**Dataset C**
Number of resolver groups	70	403	428
Duration of the training dataset	6 months	12 months	15months
Email tickets(N) in training set	11562	423343	712320
Duration in deployment	19 months	3 months	16 months
Tickets served/month(T)	1000	40,000	50,000
Total tickets served till date	19000	120,000	680,000
Ensemble Accuracy(X_{acc})	90.07%	86.17%	89.61%
Ensemble Coverage(X_{cov})	93.67%	92.88%	93.83%
Assignment Engine Accuracy(E_{acc})	92.73%	88.66%	92.13%
Assignment Engine Coverage(E_{cov})	97.84%	93.3%	95.5%

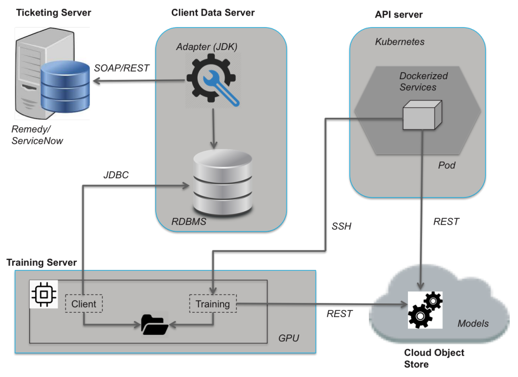


Figure 3: Deployment Architecture

the helpdesk and requires about 5 hrs of manual effort per week. The scripts for generation of misclassification reports, triggering re-training and the entire assignment engine code are maintained by a team of 2 developers (working for 5 hrs/week per account).

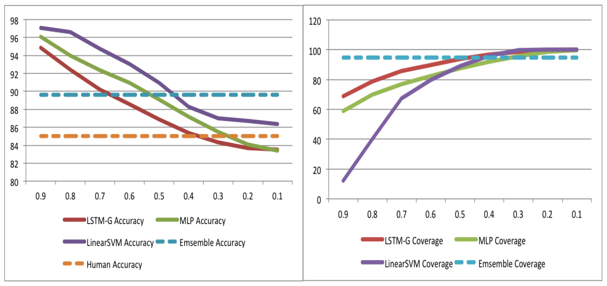
5 Evaluation

This section enumerates the results of evaluation of the assignment engine. For evaluation we have used real datasets from three major helpdesk service provider accounts. The client accounts are from two different domains viz. telecom and supply-chain/logistics. To preserve client confidentiality we henceforth 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 4 CUDA-enabled nodes. The dataset statistics as well as the final accuracy numbers achieved by our system are described in Table 4.

Human Accuracy vs. Assignment Engine Accuracy

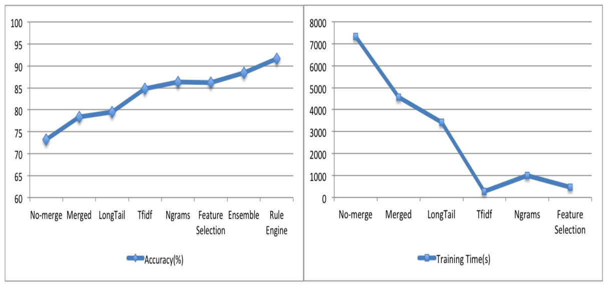
We next look at the optimal selection of algorithms that maximize accuracy and coverage. It is important to note that for business purposes the algorithms need to have at least human-level accuracy along with reasonably high coverage.

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 85%. *Therefore we select the confidence threshold such that the expected accuracy of prediction is at least 85%.*



(a) (b)

Figure 4: At different confidence thresholds(a) Assignment accuracy (b) Assignment coverage.



(a) (b)

Figure 5: Effect of different optimization techniques on (a) Classification Accuracy (b) Training time

Table 5: Improvement of sparse classes with rule engine

Resolver Group	P,R(linearSVM)	P,R(ensemble)	P,R(+rule engine)
MXX-Support	65.22%,28.99%	72.1%,35.77%	90.9%,86.95%
AppXXX-Support	64.76%,37.22%	75.1%,36.5%	89.61%,85.1%
PrXXX-Support	62.38%,40.65%	71.21%,41.6%	89%,87.34%
XXX-HSS-XXX	-	-	96.1%,90.1%
XXXha-CRM	-	-	94.6%,89.5%

This ensures that the selected classifiers operate at least at human level efficiency. Figure 4 shows 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$\geq$ 0.5) and MLP (confidence$\geq$ 0.6) gave a slightly higher accuracy(89.61%) than that of LSTM-G(confidence$\geq$0.5) and linear SVM(X_{acc}=88.38%), although the individual accuracy was marginally higher for LSTM-G compared to MLP. For this reason, as also for other practical considerations like memory and CPU constraints as well as training time our deployment in production uses an ensemble of linear SVM and MLP. For the other two datasets SVM and MLP were clear winners.

Rule Engine Accuracy

Table 5 [2] shows precision(P) and recall(R) metrics for some resolver groups having low accuracy. In each of these cases we were able to improve the accuracy significantly using the rule engine. The first three resolver groups are commonly confused (derived from confusion matrix) and individual rules were applied to predict the correct resolver group. The last two resolver groups belong to I_L, so precision(P)/recall(R) numbers are not available for the ensemble; in these cases accuracy was achieved using rules only.

Performance and scalability

Figures 5 and 6 demonstrate the impact of the different training optimization techniques on the accuracy of prediction, training time and model size of SVM. These charts are shown for only the largest dataset viz. dataset C – but the trend is fairly similar across other datasets as well. The results can be summarized as follows.

[2]Resolver Group names anonymized to preserve confidentiality

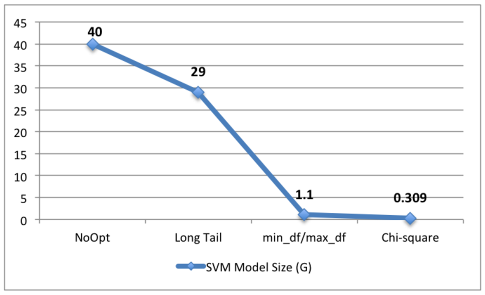


Figure 6: Model size optimization (SVM)

Merging coupled with long tail optimization brings down the number of resolver groups by about 84%(438 to 72) along with a 40% reduction in training time, 27% reduction in model size and 5% improvement in accuracy. This underscores the effectiveness of long tail approach. We also found that using the tf-idf approach with bi-grams proved to be effective across most ML algorithms including SVM. To optimize on the number of features and model size we used the tf-idf approach with parameters max_df=0.8 and min_df=5. Finally we used chi-square statistic (with best k features) to reduce the model size further. Overall using these approaches we were able to achieve a 99% reduction in model size, 93.5% reduction in training time, and a 13% increase in accuracy for dataset C while keeping the coverage constant at about 98%. We also achieved significant speedup in classification time using asynchronous requests and batching. Figure 7 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/sec, which is equivalent to about 1500 times the peak hourly volume (696 requests/hr) 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 summa-

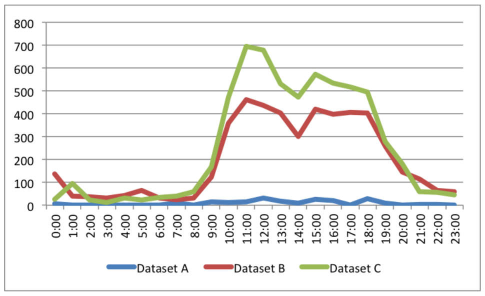


Figure 7: Ticket volumes for a single day

rized in Table 4, we give an estimate of human effort saving, across all accounts. Assuming that a human agent takes about 3 min to read and assign each ticket, the net savings (in min) for an account can be calculated as: $S_{\mathrm{i}} = T \times E_{\mathrm{cov}} \times 3$. Summed across all three accounts this gives a total saving of 51629.04 hrs/annum. Till date, our system has served more than 700,000 tickets in production across the deployed accounts.

Observations

There are three main takeaways from our evaluation results above. The most important observation is that our assignment engine performs better than all traditional machine-learning and deep learning algorithms. *Secondly*, we can see that simple machine learning algorithms like SVM and MLP are often better than more computationally expensive deep learning algorithms in the task of helpdesk email assignment. This result is very significant from a product development standpoint as these algorithms are easy to implement, require minimal computational resources and provide better performance at runtime. However, it must be noted that CNN and LSTM accuracy increases with the size of the dataset and with a very large training data size (more than 5 million) CNN and LSTM start outperforming MLP. Thus our results indicate that an ensemble of SVM and MLP will be a good trade-off for most practical purposes but if we have a large enough dataset and infrastructure is not a concern then the best choices are SVM and CNN-WE/LSTM-G. *Finally*, our results [Table 4] clearly indicate the importance of the rule engine. It not only increases the overall accuracy and coverage of the system but also ensures business continuity.

6 Conclusion and Future Work

We have proposed email ticket assignment engine that uses an ensemble of machine learning techniques combined with a rule engine to perform automated dispatch. Our system achieves human-level accuracy and has already been deployed for three customers in production. However, there is still some scope for improvement of the system e.g. adding support for images and attachments. The system can also be enhanced to handle concept drift better. This is currently handled by using a sliding window of recent data as well as use of the rule engine. However in datasets with high concept drift this method may not give good results over the long run. We need to come up with an effective active learning strategy to handle such scenarios.

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.

Agarwal, S.; Sindhgatta, R.; and Sengupta, B. 2012. Smart-dispatch: Enabling efficient ticket dispatch in an it service environment. In *18th ACM SIGKDD*.

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 *12th ICSOC*, volume 8831 of *Lecture Notes in Computer Science*. Springer.

Di Lucca, G. 2002. An approach to classify software maintenance requests. In *18th IEEE ICSM*.

Kadar, C.; Wiesmann, D.; Iria, J.; Husemann, D.; and Lucic, M. 2011. Automatic classification of change requests for improved it service quality. In *SRII Global Conference*.

Kuncheva, L. I. 2004. *Combining Pattern Classifiers: Methods and Algorithms*. New York, NY, USA: Wiley-Interscience.

Mandal, A.; Malhotra, N.; Agarwal, S.; Ray, A.; and Sridhara, G. 2018. Cognitive system to achieve human-level accuracy in automated assignment of helpdesk email tickets. *ArXiv e-prints*.

Manning, C.; Surdeanu, M.; Bauer, J.; Finkel, J.; Bethard, S.; and McClosky, D. 2014. The stanford corenlp natural language processing toolkit. In *52nd ACL (System Demonstrations)*.

Mikolov, T.; Sutskever, I.; Chen, K.; Corrado, G.; and Dean, J. 2013. Distributed representations of words and phrases and their compositionality. In *26th NIPS*.

Mitchell, T. M. 1997. *Machine Learning*. New York, NY, USA: McGraw-Hill, Inc., 1 edition.

Parvin, H.; Bose, A.; and Van Oyen, M. P. 2009. Priority-based routing with strict deadlines and server flexibility under uncertainty. In *INFORMS WSC 2009*.

Pennington, J.; Socher, R.; and Manning, C. D. 2014. Glove: Global vectors for word representation. In *EMNLP 2014*.

Potharaju, R.; Jain, N.; and Nita-Rotaru, C. 2013. Juggling the jigsaw: Towards automated problem inference from network trouble tickets. In *10th USENIX NSDI*.

Shao, Q.; Chen, Y.; Tao, S.; Yan, X.; and Anerousis, N. 2008a. Easyticket: A ticket routing recommendation engine for enterprise problem resolution. In *34th VLDB*.

Shao, Q.; Chen, Y.; Tao, S.; Yan, X.; and Anerousis, N. 2008b. Efficient ticket routing by resolution sequence mining. In *14th ACM SIGKDD*.

Zeng, C.; Zhou, W.; Li, T.; Shwartz, L.; and Grabarnik, G. Y. 2017. Knowledge guided hierarchical multi-label classification over ticket data. *IEEE Trans. Network and Service Management* 14(2):246–260.