COTA: Improving the Speed and Accuracy of Customer Support through Ranking and Deep Networks

Piero Molino
Uber AI Labs
San Francisco, California
piero@uber.com

Huaixiu Zheng
Uber Technologies
San Francisco, California
huaixiu.zheng@uber.com

Yi-Chia Wang
Uber Technologies
San Francisco, California
yichia.wang@uber.com

ABSTRACT

For a company looking to provide delightful user experiences, it is of paramount importance to take care of any customer issues. This paper proposes COTA, a system to improve speed and reliability of customer support for end users through automated ticket classification and answers selection for support representatives. Two machine learning and natural language processing techniques are demonstrated: one relying on feature engineering (COTA v1) and the other exploiting raw signals through deep learning architectures (COTA v2). COTA v1 employs a new approach that converts the multi-classification task into a ranking problem, demonstrating significantly better performance in the case of thousands of classes. For COTA v2, we propose an Encoder-Combiner-Decoder, a novel deep learning architecture that allows for heterogeneous input and output feature types and injection of prior knowledge through network architecture choices. This paper compares these models and their variants on the task of ticket classification and answer selection, showing model COTA v2 outperforms COTA v1, and analyzes their inner workings and shortcomings. Finally, an A/B test is conducted in a production setting validating the real-world impact of COTA in reducing issue resolution time by 10 percent without reducing customer satisfaction.

CCS CONCEPTS

• **Computing methodologies** → **Natural language processing**; **Machine learning**; **Neural networks**;

KEYWORDS

customer support machine learning natural language processing deep learning intent detection customer satisfaction

ACM Reference Format:
Piero Molino, Huaixiu Zheng, and Yi-Chia Wang. 2018. COTA: Improving the Speed and Accuracy of Customer Support through Ranking and Deep Networks. In *KDD '18: The 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, August 19–23, 2018, London, United Kingdom.* ACM, New York, NY, USA, Article 4, 10 pages. https://doi.org/10.1145/3219819.3219851

1 INTRODUCTION

Customer support has become an integral part of most companies. In fact, many companies have customer service departments with dedicated representatives/agents to provide support and help customers resolve issues they encounter. Prompt and accurate response to a request is essential for customer satisfaction and retention. Uber's Customer Obsession team is committed to making the customer experience as quick, easy, and accessible as possible.

When customers report problems, it is important to route them to the best possible resolution in a timely manner. Nevertheless, there is often a large amount of situational information associated with each customer ticket, which can be time-consuming to digest and synthesize into an identifiable issue type and corresponding solution for customer support representatives (CSRs). Moreover, the diversity of ways a customer can describe an issue associated with a ticket further complicates the ticket resolution process. Finally, as a company scales, support agents must be able to handle an ever-increasing volume and diversity of support tickets. For example, Uber receives hundreds of thousands of tickets every day on the platform across 400+ cities worldwide. The issues can range from technical errors and fare adjustments, to lost items. Ensuring that agents are empowered to resolve tickets as accurately and quickly as possible is the key.

Prior work has explored the challenge of issue resolution utilizing data mining and machine learning techniques to partially automate the resolution process [e.g., 9, 10, 14, 23]. Hui and Jha [14] applied data mining methods to extract customer-related information from both unstructured text data and structured databases, which CSRs employ for decision making. In [9], the authors built a ML classifier to identify emotional emails sent by customers and suggested that the classifier can be used to automatically route these emails to specialized representatives. Similarly, existing research in spoken dialogue systems aims to build models to detect intent and extract named entities for call classification and routing [e.g., 10, 25, 29].

In contrast to prior work, which mostly focuses on customer information retrieval and intent detection for routing, this work investigates techniques to directly help CSRs improve their speed and accuracy, which in turn leads to better customer experiences. To accomplish this, we build COTA (Customer Obsession Ticket Assistant), an intelligent system based on machine learning (ML) and natural language processing (NLP) techniques that is integrated with Uber's customer support platform. The system provides CSRs with suggested ticket classifications and answers based on ticket content and additional context such as relevant trip information.

In this paper, we share our experiences building COTA and report empirical results after successfully integrating it with Uber's customer support platform. The main contributions of this work are as follows:

- Proposing a new method to convert a multi-classification problem to a ranking one, which significantly improves the performance of feature-engineered classical ML algorithms, especially when there are thousands of classes. The new method is introduced in Section 3.
- Introducing Encoder-Combiner-Decoder, a novel deep learning architecture that allows for heterogeneous input and output types and the injection of prior knowledge in the form of architecture choices. The new architecture is introduced in Section 4.
- Conducting comprehensive experiments to compare different models and uncover their underlying mechanisms and shortcomings. Experiments are discussed in Section 5.
- A/B testing COTA in production to show that it enables quick and efficient issue resolution and improves key business metrics. Business impact is discussed in Section 5.8.

2 INTELLIGENT SYSTEM FOR CSRS

As one of the world's largest ridesharing providers, Uber receives hundreds of thousands of support tickets from users every day. Until COTA, the process of resolving a ticket has been mostly manual. A typical workflow for ticket resolution involves two steps: **contact type identification** and **solution selection**.

When a CSR opens a ticket, their first step is to determine what it is about (e.g. *wrong food delivered* or *trip cancellation*). We refer to this task as **contact type identification** (similar to intent detection in dialogue systems research). With thousands of potential contact types and a deep tree to navigate through, reducing the amount of time a CSR spends identifying a ticket's type is important because it also decreases the time customers have to wait for their issue to be solved.

Once a contact type is chosen, the next step is to select the right solution and reply. Specifically, a customer service team usually maintains a bank of **reply templates** from which agents can select the correct one for each contact. This step of finding the proper solution and selecting the appropriate reply is also time consuming since there are thousands of possible solutions to choose from and each contact type has a different set of protocols and solutions it is associated with. We refer this task as **reply template selection**.

2.1 COTA System Architecture

To solve both contact type identification and reply template selection, Customer Obsession Ticket Assistant (COTA) is designed to help our CSRs improve their speed and accuracy. Built on top of our support platform, COTA is comprised of two ML models that suggest the three most likely contact types and reply templates to CSRs for each support tickets based on its content and context. These two models are the **type model** and the **reply model**, respectively. Fig. 1 depicts the general COTA architecture, which follows a four-step flow:

(1) Once a new ticket is issued by the user, the back-end service collects all relevant features of the contact.
(2) The back-end service then sends these features to the ML models, receives back predictions and stores them.

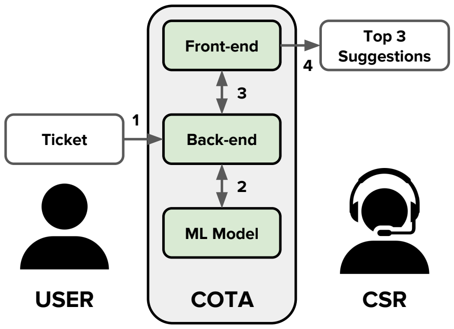

Figure 1: The COTA system architecture is composed of a four-step workflow.

(3) Once a CSR opens a given ticket, the front-end service triggers the back-end service to check if there are any updates to the ticket. If there are no updates, the back-end service will return the stored predictions; if there are updates, it will fetch the updated features and go through Step 2 again.
(4) The top three ranked predictions are suggested to agents; from there, agents make a selection and resolve the support ticket. The decision of returning the top three is a UX decision.

2.2 Model Features

There are four main sources of information often referenced by CSRs when deciding contact type and reply template of a ticket: ticket message and metadata, user-level as well as trip-level information. These sources of user and trip information contain critical ingredients for contact type identification and reply template selection, and as such serve as features for training the model.

Ticket message. The text written by the customer when submitting a ticket. A detailed description of how ticket messages are processed and converted into features through an NLP pipeline is given in Section 3.1.

Ticket metadata. Every support ticket comes with a set of metadata, such as *ticket creation time* and *product type* (e.g., Uber Eats, UberPool and UberX).

User information. Information about the user sending the ticket, i.e. user type (e.g., driver, rider or eater), can provide important signals for both contact type identification and reply template selection. For instance, only eaters would submit a ticket related to the contact type *wrong food delivered*, while the country of the user may affect the policies to solve the tickets, and consequently the appropriate reply template to use.

Trip information. For tickets related to trips, trip-level information can be very helpful in predicting contact types, such as *cancellation* and *mistimed trips*. Examples of trip features are *city*, *estimated time of arrival* and *trip status*.

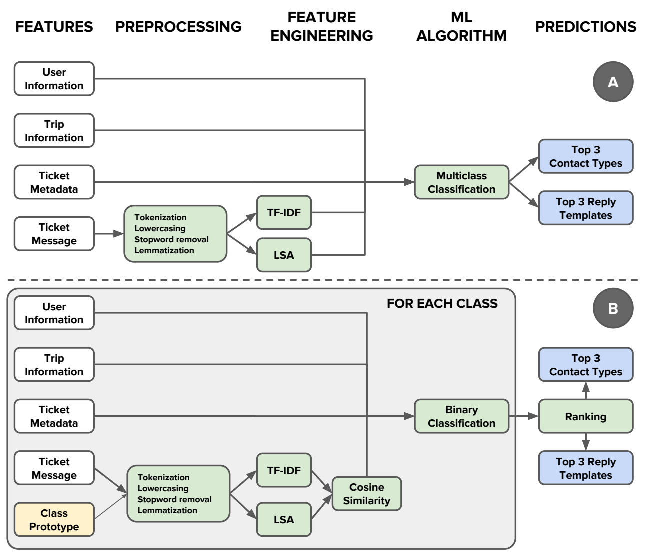

Figure 2: The NLP pipeline built for COTA v1: a) topic vectors are directly used by the classification algorithm and b) cosine-similarity features are engineered and used by the pointwise-ranking algorithm.

3 COTA V1: TRADITIONAL MACHINE LEARNING MODELS

The first version of COTA (COTA v1) is built with topic-modeling-based traditional NLP and ML techniques leveraging a mixture of text, categorical, and numerical features. In order to extract the text features, an NLP pipeline processes incoming ticket messages. This section describes each component of the pipeline shown in Fig. 2.

3.1 NLP Preprocessing

The first step is to analyze text at the word-level and use topic modeling to better understand the meaning of text data. The text is cleaned by removing HTML tags. Next, the message's sentences are tokenized and stop-words are removed. Then, each word is lemmatized to convert different inflected forms into the same base form. Finally, the documents are converted into a bag of words, forming a dictionary.

To understand user intent, topic modeling is performed on the bag of words after preprocessing. First, TF-IDF (term frequency-inverse document frequency) [22] is used to obtain a sparse vector representation. Furthermore, topics are extracted through Latent Semantic Analysis (LSA) [18], that returns a dense vector for each ticket message. We perform Truncated-SVD, that decomposes our term-document matrix T obtained from ticket messages into $T \approx U_k \Sigma_k V_k^\top$, choosing the number of largest singular values k in the Σ matrix by keeping at least 90% of the variance of the term-document matrix. By doing so, we end up with about 200 dimensions. Fig. 3 shows examples of the topics extracted by LSA, and they are meaningful topics with respect to Uber's customer support ticket data, e.g. city related topics, rating related ones, refunds, fare adjustments and so on.

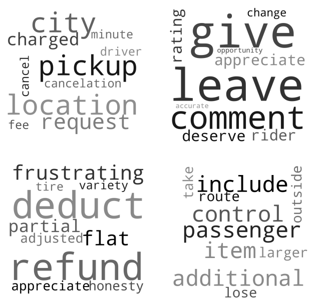

Figure 3: Examples of topics extracted by LSA from text data in customer support tickets.

3.2 Feature Engineering

Two different approaches of using topic-modeling-based vector representations are tested.

The first approach is to directly leverage the topic vectors of ticket messages as features to perform downstream classifications in type and reply models, as shown in Fig. 2(A). However, this direct approach suffers from a high dimensionality of the vectors.

The second approach uses the vectors in an indirect fashion by performing further feature engineering by computing cosine similarity features, as illustrated in Fig. 2(B). Historical tickets associated with each contact type and reply template are collected and a bag-of-words representation is obtained for each of them. The bag-of-words representation of each class i (either contact type and reply template) is transformed into a LSA vector, which we use as a prototype vector p_i. The bag of words of each incoming ticket message j is projected in the same semantic space in order to obtain a vector t_j. The cosine similarity score s_{ij} between each p_i and t_j is then computed and represents the similarity between class i and ticket j. The same process is repeated with TF-IDF vectors in place of LSA ones. Doing so reduces the feature space from hundreds of dimensions of the original vectors to just a handful similarity features.

Similarity feature collection is further expanded by using other class specific information explicitly. For example, in order to obtain an additional prototype, we can use the actual textual content of the reply template. This adds two additional similarity features, one obtained with TF-IDF vectors and the other one with LSA ones.

3.3 Algorithms: Multi-Class Classification vs. Pointwise-Ranking

Given the two approaches to extract features from text described above, two different algorithms are trained on the tasks of contact type identification and reply template selection.

The first one formulates the tasks as a multi-class classification problem where the contact types and reply templates are targets. The model takes in TF-IDF and LSA vectors together with the categorical and numerical features of the other feature families and use them to predict the target class. The pipeline for this straightforward approach is shown in Fig. 2(A).

In order to leverage the engineered cosine-similarity features, a pointwise-ranking [19] algorithm (Fig. 2(B)) scores each ticket-class pair and then ranks classes based on the score. Specifically, a subset of all ticket-class pairs is sampled: the matching one is given a positive label (1) and a random subset of non-matching ones are given a negative label (0). The target to predict is the label Y_{ij} (0/1) for each pair of class i and ticket j. Using the cosine similarity features as well as categorical and numerical features, a binary classification algorithm is built to classify whether or not each ticket-class combination matches. Once the algorithm scores each possible pair, classes are ranked based on their scores and the top-ranked ones are selected.

In COTA v1, both the multi-class classification and point-wise ranking algorithms use the same RandomForest algorithm [12] for learning to predict the correct class of each ticket.

The two approaches are compared in an experiment described in Section 5.4.

4 COTA V2: DEEP LEARNING ARCHITECTURE

In recent years, deep learning models have been heavily adopted for several NLP tasks including syntactic [3] and semantic parsing [20], semantic role labeling [21], recognizing textual entailment [24] and named entity recognition [7], leading to performance improvements. Moreover, the same architectures employed for these tasks have been adopted also for end-to-end applications like summarization [6], machine translation [2], building dialogue systems [11] and, more importantly for our goals, text categorization [16].

Trying to leverage these models is a natural step for COTA, as the tasks to automate can be casted as text classification. In [30] the authors have already shown how deep learning architectures can be effective in the presence of big datasets. The main difference with respect to state-of-the-art text classification architectures, is that they are explicitly designed to deal with textual inputs, while additional input features such as metadata, trip as well as user related information are available in COTA. To incorporate these extensive features, COTA v2 extends the wide-and-deep approach described in [4]. The wide-and-deep approach consists in combining other features with the text ones by concatenating them before the final layer of the architecture. COTA v2 combines this idea with multi-task learning, that accommodates ways to train a model to learn to predict multiple outputs by optimizing losses for all the different outputs at the same time. In [7], the authors showed how training models for related tasks in this fashion can lead to improved performances in all tasks.

4.1 Encoder-Combiner-Decoder Architecture

The combination of deep-and-wide with multi-task learning inspired losses forms the basis of a new general architecture introduced in this paper, the Encoder-Combiner-Decoder (ECD), depicted in Fig. 4. In the encoder part of the architecture, each single input feature is encoded by a sub-part of the model, depending on its type. More formally each encoder is a function $e_{t(\phi_i(x))}(\phi_i(x))$ where $\phi_i(x)$ is the i-th input feature and $t(\phi(x))$ is the type of feature. For instance, text features can by encoded by a Character-Based CNN or by a Bidirectional RNN on the word sequence, categorical

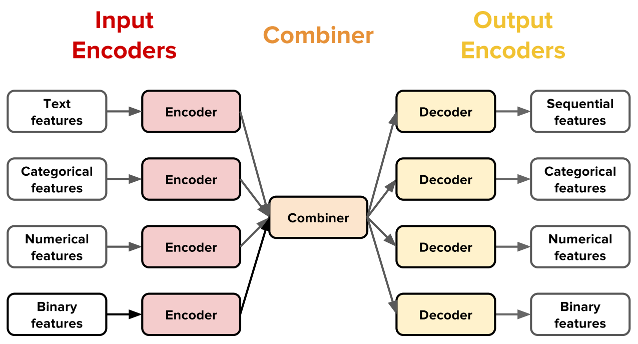

Figure 4: Encoder-Combiner-Decoder Architecture depiction.

features can be encoded through a linear projection into an embedding space, binary feature can be encoded with one single number and numerical features can be encoded through a single neuron that acts as a learned scaling factor. Each of these different encoders outputs a vector encoding for the input feature they deal with. In the combiner part, those vectors are concatenated as they are in the wide-and-deep approach, but the concatenation is optionally followed by fully connected layers that can learn some non-linear combination of the representations obtained so far. The combiner is needed in any circumstance when there is more than one input feature, like in our case, but could be skipped if there's more there is only one input feature. A combiner is defined as a function $c(x)$ so that:

$$c(x) = f([e_{t(\phi_0(x))}(\phi_0(x)), \dots, e_{t(\phi_n(x))}(\phi_n(x))])$$

, where $[\dots]$ is the concatenation operator, n is the number of input features, and f is a multi-layer perceptron. Finally, in the decoder part, different output features have different "heads", with each of them predicting a different output feature depending on their type. Each output decoder can contain an arbitrary number of additional fully connected layers between the output of the combiner and the layer responsible for the prediction. This makes it possible to have a multi-task model with weights shared among all the tasks up to the combiner and have a set of weights that are task-specific for increased flexibility. Each decoder is a function d that returns a loss and is defined as:

$$d_{\phi_i(x)}(x) = f_{\phi_i(x)}(c(x))$$

, where f is again a multi layer perceptron and there could be different fs for different output features. Categorical output features are treated as a multi-class classification task: they use the output of the combiner and pass it through softmax layer, and return a categorical cross entropy loss. Numeric output features are treated as a regression task: they use the output of the combiner to predict a single value, and return a mean squared error loss. Binary output features are treated as a binary classification task: they use the output of the combiner as input for a layer with a logistic activation, and return a binary cross entropy loss. The sum of all the losses coming from the different output features is optimized through any variant of stochastic gradient descent in multi-task learning fashion. The final loss that is optimize is the weighted sum of all

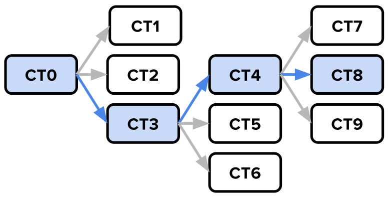

Figure 5: Example of path in the contact type hierarchy used as target sequence. CT stands for contact type. The contact type to predict is CT8 and the target sequence is [CT0, CT3, CT4, CT8].

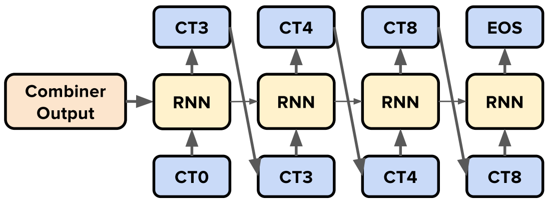

Figure 6: Example of a sequence decoder predicting a path in the contact type tree.

the losses of the output features and is defined as:

$$L(x) = \sum_{i \in o(x)} w_i d_{\phi_i(x)}(x)$$

, where $o(x)$ is the set of indices of output features, and w_i is a user defined weight for the specific output feature.

The ECD architecture provides three key benefits. It directly incorporates all raw input features, eschewing the need for preprocessing outside of mapping textual and categorical features into integers, as well as learning a single model to predict both contact types and reply templates. Finally, the architecture enables an easy method for swapping and comparing different encoders. As a result, six diferent encoders are implemented for textual features: CNN, RNN and CNN followed by RNN, each one working on characters or on words, inspired by [16, 27, 30]. Each of them can be parametrized independently, specifying the number of layers, the size of convolutional filters of each layer and how many stacked RNNs to use, if they should be bidirectional, and their type (Simple RNN [8], LSTM [13] or GRU [5]) among many other hyperparameters.

We are going to release **ludwig**, an open source toolbox implementing the ECD architecture.

4.2 Injecting Prior Knowledge in the Architecture

The ECD architecture makes it possible to further improve model performance by injecting prior knowledge about each task directly at the model architecture level.

4.2.1 Predicting paths in a tree. The first way to inject prior knowledge deals with the decoder for predicting contact types. As previously described, contact types are organized in a hierarchy, but are treated as independent classes in COTA v1. In COTA v2 the hierarchical nature of the contact types is exploited directly by

```
[{name: OF1,
  dependencies: [OF2]},
 {name: OF2,
  dependencies: []},
 {name: OF3,
  dependencies: [OF2, OF1]}]
```

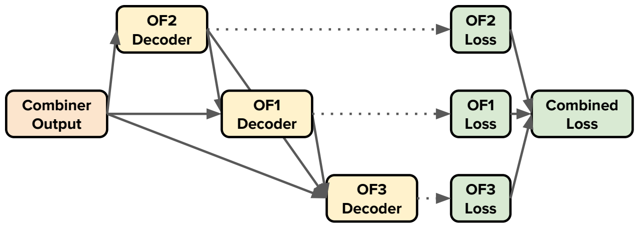

Figure 7: Decoder dependency example. OF stands for output feature.

predicting a path in the tree that leads to the target contact type rather than predicting the contact type directly, as shown in Fig. 5. This is accomplished by adding an additional sequential decoder, similar to what is used in sequence-to-sequence models [26] for machine translation, where the target sequence is the sequence of contact type nodes that describes a path in the tree from the root to the target contact type, such that the last node in the sequence is the actual target contact type. The node prediction at each step is used as input at the following step of sequence prediction, as depicted in Fig. 6. The loss for this output feature is the sum of the cross entropy losses at each step of the sequence generation, while only the last predicted contact type in the sequence is used when assessing its accuracy.

Interestingly, not all model mistakes are equal in terms of time it takes for CSRs to recover from errors. In fact, different model architectures may gracefully degrade and provide help for CSRs even if the answer is not exactly correct. For instance predicting the parent of the correct contact type is better than predicting a contact type in a completely different part of the hierarchy because the CSRs will be just one click away from the correct solution. In contrast, navigating through all the tree to select the correct contact type consumes a considerably larger amount of time. Moreover, searching for solutions using a beam search could even improve handling time, providing an efficient approximation of the k most likely paths rather than the most likely single path. The hypothesis that a recurrent decoder could make more reasonable mistakes is directly tested in Section 5.5.

4.2.2 Adding decoder dependencies. The second way to inject prior knowledge deals with the logical dependency between the two tasks we try to solve. Contact type identification is logically the first step that CSRs undergo in the process of solving a ticket, and depending on the outcome, they decide what reply template to use. From the model perspective, having access to information about the contact type of the ticket is beneficial for suggesting the correct reply template CSRs should use.

As different decoders can have a set of weights that is specific to output feature they try to predict, the output of the contact type decoder is injected as an additional input to the reply template decoder, concatenating it with the output of the combiner. More generally, in the ECD architecture allows for dependencies between output features. The dependencies must form a directed acyclic graph for the computational graph to be built as when building it the outputs of the decoders of all the output features that the current output feature is dependent on is concatenated with the output of the combiner. This concatenation is used as the input to the current decoder, as exemplified in Fig. 7.

The hypothesis is that adding dependencies among output features would help in two ways: a) the overall performance in predicting output features with dependencies should improve due to to the prior knowledge injection and b) the outputs of the decoders will be more coherent with each other, i.e. the number of samples where all output predictions are correct as opposed to the number of samples where only a subset of output predictions is correct should be higher, even in the case of equivalent performance on each single output prediction. Both hypotheses are tested in Section 5.5.

5 EXPERIMENTS AND RESULTS

5.1 Goals and Evaluation Metrics

The goals of the experiments are twofold: to evaluate both the quality of the predictions obtained from these models and also how the use of the models impacts business metrics.

For the first goal, we evaluate models in an offline experiment in terms of their Accuracy and Hits@3 on both contact type identification and reply template selection. The decision to use Hits@3 as a metric is rooted in the UX decision to show the top three model predictions to CSRs, so the number correct predictions we can provide among the ones actually shown to our users is assessed. Moreover, the models should to return correct predictions on both tasks at the same time, so the accuracy of predicting the correct contact type and the correct reply template for the same tickets is considered.

The business metrics that relate directly to how quickly and seamlessly we are able to resolve customer support issues, thereby improving the customer support experience, are the most important ones. The goal is to provide them with a fast customer support experience that solves their issues. For this reason, the handling time of each ticket and overall customer satisfaction is tracked through surveys. The hypothesis is that when using COTA, CSRs will be faster in handling tickets without degrading customer satisfaction. This hypothesis is tested with an online experiment, where A/B testing is performed on a subset of our English-speaking CSRs, using as treatment the intelligent suggestions generated by our COTA system in contrast to control with no suggestions.

5.2 Dataset

Uber's historical customer support data is used to train and evaluate the model performance. As mentioned before, the input features include ticket message, ticket metadata, user-level information, and trip-level information. The target variable for the task of contact type identification is the contact type ID, while for the task of reply template selection, the predicted contact type is used as an additional input, and the target variable is the reply template ID. About 3M tickets were collected and split randomly into train (2.8M), validation (94K) and test (94K) sets. The dataset contains thousands of different contact types and reply templates, exhibiting a long-tail distribution. Contact types in particular are structured in seven levels deep hierarchy. Ticket messages are truncated to 1024 characters, as only 0.001% of them being longer and the majority of them concentrating around 250 characters.

Table 1: Performance on the validation set of the best hyperparameter setting for each model using different text encoders. The reported minutes are relative to a full validation set evaluation. *CT* stands for contact type, *RT* stands for reply template, *Comb.* stands for Combined.

Encoder	CT Acc	RT Acc	Comb. Acc	Min
Char C-RNN	**0.6505**	**0.5155**	**0.5024**	17.5
Word CNN	0.6433	0.5099	0.4960	**2**
Word RNN	0.6413	0.5096	0.4971	8.5
Word C-RNN	0.6315	0.4999	0.4841	6
Char CNN	0.6298	0.4990	0.4815	2.5
Char RNN	0.6262	0.4972	0.4767	36

5.3 Hyperparameters

To find the best hyperparameters for each model, a hyperparameter search is conducted, selecting hyperparameter combinations based on model accuracy for a validation set.

For COTA v1 models, a grid search on the hyperparameters of the Random Forest was run to find the following optimal set : estimators: 100, max depth: 100, max features: $sqrt(\#features)$, min samples leaf: 50.

For COTA v2 models, the hyperparameter search is more complex, as different text encoders needed a completely different set of hyperparameters. A parallel random search is performed, cutting at 100 different configurations for each different encoder architecture. Performance and speed are reported in Table 1. There is a relatively small performance spread between the best and worst hyperparameter combinations of each encoder, with the worst performing encoder (Char RNN) obtaining just 2% lower accuracy than the best performing (Char C-RNN). In the end, Word CNN was chosen as text encoder as it performs less than 1% worse than the Char C-RNN, while being approximatively 9 times faster during both training and prediction.

The best Word CNN model has 256 dimensional word embeddings and 4 parallel 1D convolutional layers of size, respectively, 2, 3, 4 and 5 with 512 filters each. All categorical features were embedded with 256 dimensional embeddings. The combiner does not have fully connected layers, while the two decoders had two fully connected layers of size 512 and 256 respectively. Each numeric feature is encoded using only a batch norm [15] layer. Fully connected layers interleaved by dropout layers with a dropout probability of 0.35. The loss is optimized with Adam [17] using a learning rate of 0.00025 and a batch size of 256.

5.4 COTA v1: Classification vs. Ranking

Experiments on COTA v1 compare multi-class classification and pointwise-ranking algorithms on the same dataset and the same optimized hyperparameters. The results are shown in Table 2.

Table 2: Comparison between Classification and Ranking in COTA v1.

	Contact Type		Reply Template		Combined
	Acc	Hits@3	Acc	Hits@3	Accuracy
Classification	0.4583	0.6118	0.3669	0.5315	0.3190
Ranking	**0.4913**	**0.6716**	**0.4753**	**0.7207**	**0.4552**

Table 3: Categorical decoder (multi-class contact type classification) vs Sequential decoder (predicts paths in the contact type tree) on contact type prediction. *Accuracy+p* indicates accuracy considering also the parent of the target node as a correct prediction.

Contact Type Decoder	Accuracy	Accuracy+p
Categorical	0.6321	0.6743
Sequential	**0.6568**	**0.7342**

The ranking algorithm outperforms the classification algorithm significantly on both type identification and reply selection, as measured by the metrics of accuracy, Hits@3 and combined accuracy of the two tasks. In particular, for the contact type identification, the ranking improves accuracy by more than ~3% and Hits@3 by ~6%. For the reply template selection, the improvement by obtained by ranking algorithm is much more pronounced: ~11% on accuracy and ~19% on Hits@3. This can be attributed to the fact that in the ranking framework for reply template selection, the meta text information (the content) of reply templates is injected directly into the cosine-similarity feature engineering. As expected, it significantly boosts the model performance compared to classification. As a results of the improvements to both type and reply models, ranking algorithm has a much stronger performance on the overall accuracy of both tasks: ~14% more accurate compared to classification. These empirical results highlight the importance of feature engineering in cases where a non-deep-learning algorithm is employed to tackle the task at hand.

5.5 COTA v2: Testing Prior Knowledge Injection Hypotheses

The three hypotheses regarding the knowledge injection in the model architecture discussed in Section 4.2 are tested:

(1) The sequential decoder could learn to make more reasonable mistakes.
(2) Adding a dependency from contact type to reply template should improve performance in predicting reply templates.
(3) Adding a dependency from contact type to reply template should improve performance in predicting both contact type and reply templates at the same time.

Hypothesis 1 is confirmed by the results shown in Table 3. The sequential decoder is slightly more accurate than the categorical one. Furthermore, parents of the correct contact types are included as correct predictions (*Accuracy+p*), as those are considered reasonable mistakes, and the sequential decoder obtains a ~6% higher score in this setting, confirming that it makes more reasonable mistakes than the categorical one.

Table 4: Effect of adding dependency between the reply template decoder and the contact type decoder.

	Reply Template		Combined
Reply Template Decoder	Accuracy	Hits@3	Accuracy
No dependency	0.5061	0.7042	0.4012
With dependency	**0.5471**	**0.7521**	**0.5312**

Table 5: Comparison between COTA v1 and COTA v2.

	Contact Type		Reply Template		Combined
	Acc	Hits@3	Acc	Hits@3	Accuracy
COTA v1	0.4913	0.6716	0.4753	0.7207	0.4552
COTA v2	**0.6568**	**0.7258**	**0.5471**	**0.7521**	**0.5312**

Hypothesis 2 and 3 are confirmed by the results shown in Table 4. Adding the dependency between the reply template decoder and the contact type one improves the performance on the reply template prediction by ~4% accuracy and approximate ~5% Hits@3. The biggest improvement is visible on the combined accuracy of both prediction tasks, where adding the dependency improves the accuracy by ~13% and brings the score close to the accuracy on the reply template alone, confirming that the dependency helps coherence between both outputs.

5.6 Comparison between COTA v1 and v2

After obtaining the best models for COTA v1 and COTA v2, both models are compared directly for all tasks measuring the combined accuracy. The results in Table 5 show how COTA v2 outperforms COTA v1 in all tasks with a variable margin, most notably in contact type accuracy by ~16% and in combined accuracy by ~8% . This result is in line with literature on text classification where deep learning models outperform other ML approaches when big amounts of training data is available.

5.7 Model Analysis of COTA v2

Here, the deep learning models in COTA v2 are analyzed in order to understand how it works and its error modes. The representations learned as a byproduct of training on the impact of class imbalance on model performance are visualized.

To gain insight into the model inner working, the embeddings the model learns for the words in the tickets and for the contact types are visualized. The high-dimensional embeddings space is projected to 2d using t-SNE [28]. Fig. 8(A) shows the word embeddings of a set of keywords often encountered in COTA's use case. Meaningful clusters emerge in the the t-SNE plot. Semantically related words such as "car" and "vehicle", "phone" and "call" appear to be close to each other. Fig. 8(B) shows the embeddings learned for the contact types with each data point corresponding to one unique contact type. The embeddings are extracted from the weights of the last fully-connected layer before the softmax layer in the deep learning model for classifying contact types. Each column of the weight matrix can be interpreted as the embedding

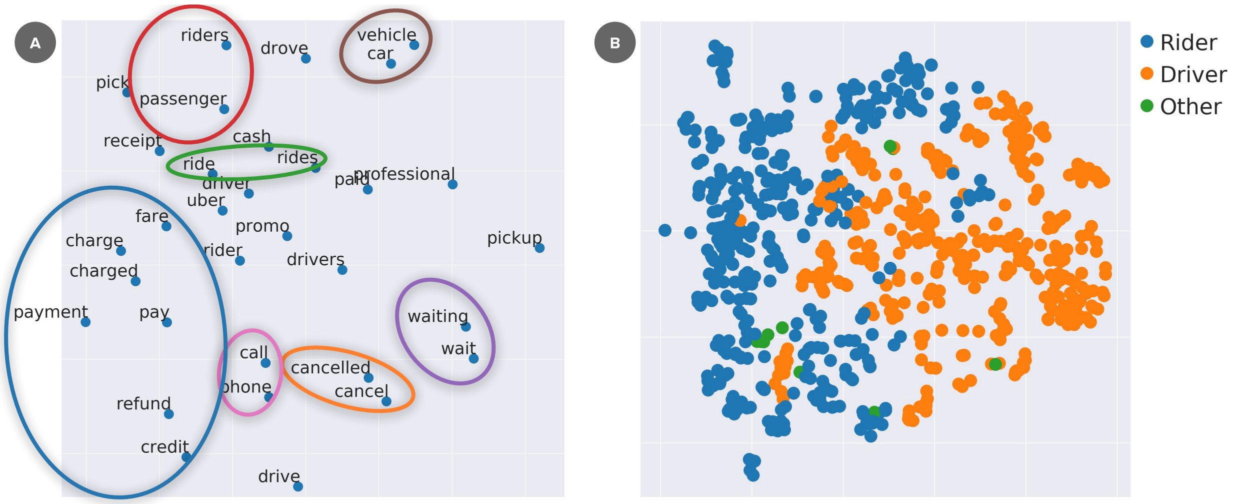

Figure 8: Embeddings learned by the deep learning models: a) word embeddings, b) entity embeddings of contact types.

encoding of a contact type class. The contact types are color coded into three major groups, namely “rider”, “driver”, and “other” (e.g., eater, restaurant, etc.). The t-SNE plot shows clear clustering of rider and driver related contact types. These visualizations intuitively confirm that the model is learning reasonable representations and suggest that the model is capable to capture correlations and semantical connections between words and the relationship between contact types.

Class imbalance is a systematic property of the dataset: some classes (both contact types and reply templates) are rarely used by the CSRs, because those types of issues either rarely occur, or the distribution of issues the users face change continuously due to seasonality. As a result, older classes are less relevant and rarely used. Fig. 9 shows how the F1 score of each class compares against the class frequency; as expected, the model performs much better on frequent classes than rare ones because there is more training data available.

The model analysis led to directions for improving the overall system, including the addition of relevant metadata as input features and the consolidation of the number of both contact types and reply templates to remove unused ones in order to have a more balanced class distribution and a higher amount of data in each class.

5.8 Evaluation of Business Metrics

To measure COTA’s impact, controlled A/B tests are conducted online on English language tickets. In those experiments, there are thousands of agents, randomly assigned into either control or treatment groups. Agents in the control group ware exposed to the original workflow without suggestions, while agents in the treatment group are shown a modified user interface containing suggestions on contact types and reply templates produced by COTA system. We collect tickets solved solely by either agents in the control or treatment group, and measure a few key metrics, including model accuracy, average handle time, and customer satisfaction score obtained through surveys.

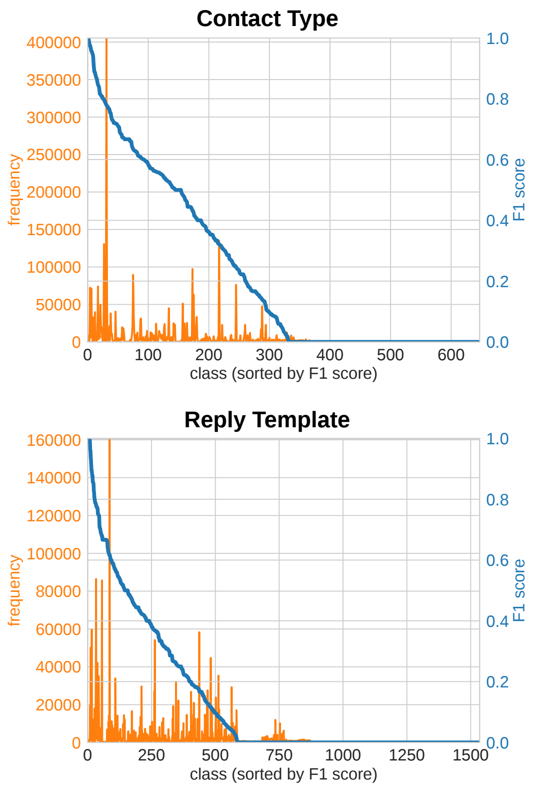

Figure 9: Class frequency compared with F1 score of COTA v2 predictions on both contact types and reply templates.

The online model performance is measured and compared to offline performance. The model performance is consistent in both the offline and online settings. Then, customer satisfaction scores are measured and compared across control and treatment groups.

In general, customer satisfaction often increases by a few percentage points. This finding indicates that COTA delivers the same or slightly higher quality of customer service. Finally, to determine how much COTA affected ticket resolution speed, the average ticket handling time between the control and treatment groups is measured as well. On average, this new feature reduced ticket handling time by $\sim$10% (p-value $\sim 10^{-8}$).

Therefore, by injecting ML intelligence into the ticket solving process, COTA system can significantly improve agent performance and speed up ticket resolution with an improved customer satisfaction.

6 CONCLUSIONS

This paper describes two different model implementations of an intelligent system for improving customer support: COTA v1, a model based on feature engineering, and COTA v2, a model that exploits raw signals through deep learning architectures. In COTA v1, a feature engineering method is employed to transform a classification task with thousands of classes to a pair-wise ranking one. COTA v2 is based on Encoder-Combiner-Decoder, a newly proposed novel deep learning architecture that enables dealing with different inputs in a flexible way, and allows for multi-task learning. Our experiments validate the hypotheses that 1) a ranking objective would perform better than a multi-class classification one in COTA v1, and 2) injection of prior knowledge in the form of architecture choices would improve performance making model's errors more reasonable. COTA v1 and COTA v2 are compared, showing how deep learning architectures perform better than a feature engineering-based architecture when learning from big datasets like the one we used. Insights on the inner working of the model are obtained by visualizing the embeddings it learns, and its shortcomings in dealing with rare classes and imbalance in the class distribution are analyzed. This analysis results in a set of improvements to be implemented in the future version of the system. At the end, COTA is deployed and tested in production and the results show that it can significantly reduce ticket resolution time while improving customer satisfaction.

ACKNOWLEDGMENTS

The authors wish to thank the Uber's Michelangelo team for the support in productionizing the models, the Opus team for the support of GPU resources, Hugh Williams, Andy Harris, Monis Ahmed Khan, Hongwei Li, Alexandru Grigoras, Viresh Gehlawat, Basab Maulik, Chinmay Maheshwari, Christina Grimsley, Chintan Shah, Franziska Bell, Taj Singh, Douglas Bemis, Felipe Petroski Such, Paul Szerlip and Eli Bingham.

REFERENCES

[1] 2014. *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL.* ACL.

[2] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural Machine Translation by Jointly Learning to Align and Translate. In *International Conference on Learning Representations (ICLR).*

[3] Samuel R. Bowman, Jon Gauthier, Abhinav Rastogi, Raghav Gupta, Christopher D. Manning, and Christopher Potts. 2016. A Fast Unified Model for Parsing and Sentence Understanding. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016, August 7-12, 2016, Berlin, Germany, Volume 1: Long Papers.* The Association for Computer Linguistics.

[4] Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, Rohan Anil, Zakaria Haque, Lichan Hong, Vihan Jain, Xiaobing Liu, and Hemal Shah. 2016. Wide & Deep Learning for Recommender Systems. In *Proceedings of the 1st Workshop on Deep Learning for Recommender Systems (DLRS 2016).* ACM, New York, NY, USA, 7–10.

[5] Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation, See [1], 1724–1734.

[6] Sumit Chopra, Michael Auli, and Alexander M. Rush. 2016. Abstractive Sentence Summarization with Attentive Recurrent Neural Networks. In *NAACL HLT 2016, The 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, San Diego California, USA, June 12-17, 2016.* The Association for Computational Linguistics, 93–98.

[7] Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel P. Kuksa. 2011. Natural Language Processing (Almost) from Scratch. *Journal of Machine Learning Research* 12 (2011), 2493–2537.

[8] Jeffrey L. Elman. 1990. Finding Structure in Time. *Cognitive Science* 14, 2 (1990), 179–211.

[9] Narendra K. Gupta, Mazin Gilbert, and Giuseppe Di Fabbrizio. 2013. Emotion Detection in Email Customer Care. *Computational Intelligence* 29, 3 (2013), 489–505.

[10] Narendra K. Gupta, Gökhan Tür, Dilek Hakkani-Tür, Srinivas Bangalore, Giuseppe Riccardi, and Mazin Gilbert. 2006. The AT&T spoken language understanding system. *IEEE Trans. Audio, Speech & Language Processing* 14, 1 (2006), 213–222.

[11] Dilek Hakkani-Tür, Gökhan Tür, Asli Çelikyilmaz, Yun-Nung Chen, Jianfeng Gao, Li Deng, and Ye-Yi Wang. 2016. Multi-Domain Joint Semantic Frame Parsing Using Bi-Directional RNN-LSTM. In *Interspeech 2016, 17th Annual Conference of the International Speech Communication Association, San Francisco, CA, USA, September 8-12, 2016.* ISCA, 715–719.

[12] Tin Kam Ho. 1998. The Random Subspace Method for Constructing Decision Forests. *IEEE Trans. Pattern Anal. Mach. Intell.* 20, 8 (1998), 832–844.

[13] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long Short-Term Memory. *Neural Computation* 9, 8 (1997), 1735–1780.

[14] Siu Cheung Hui and G. Jha. 2000. Data mining for customer service support. *Information & Management* 38, 1 (2000), 1–13.

[15] Sergey Ioffe and Christian Szegedy. 2015. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015 (JMLR Workshop and Conference Proceedings)*, Vol. 37. JMLR.org, 448–456.

[16] Yoon Kim. 2014. Convolutional Neural Networks for Sentence Classification, See [1], 1746–1751.

[17] Diederik P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In *International Conference on Learning Representations (ICLR).*

[18] Tom Landauer and Scott Dooley. 2002. Latent semantic analysis: theory, method and application. In *Proceedings of the Conference on Computer Support for Collaborative Learning: Foundations for a CSCL Community, CSCL '02, Boulder, CO, USA, 2002.* International Society of the Learning Sciences, 742–743.

[19] Yu Lei, Wenjie Li, Ziyu Lu, and Miao Zhao. 2017. Alternating Pointwise-Pairwise Learning for Personalized Item Ranking. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management, CIKM 2017, Singapore, November 06 - 10, 2017.* ACM, 2155–2158.

[20] Chen Liang, Jonathan Berant, Quoc Le, Kenneth D. Forbus, and Ni Lao. 2017. Neural Symbolic Machines: Learning Semantic Parsers on Freebase with Weak Supervision. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics, ACL 2017, Vancouver, Canada, July 30 - August 4, Volume 1: Long Papers.* Association for Computational Linguistics, 23–33.

[21] Diego Marcheggiani and Ivan Titov. 2017. Encoding Sentences with Graph Convolutional Networks for Semantic Role Labeling. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, EMNLP 2017, Copenhagen, Denmark, September 9-11, 2017.* Association for Computational Linguistics, 1506–1515.

[22] Ian McCulloh, Eric Daimler, and Kathleen M. Carley. 2008. Using Term-Frequency-Inverse-Document Frequency of Email to Detect Change in Social Groups. In *Proceedings of the 2008 International Conference on Information & Knowledge Engineering, IKE 2008, July 14-17, 2008, Las Vegas, Nevada, USA.* CSREA Press, 333–337.

[23] Durga Prasad Muni, Suman Roy, Yeung Tack Yan John John Lew Chiang, Antoine Jean-Marie Viallet, and Navin Budhiraja. 2017. Recommending resolutions of ITIL services tickets using Deep Neural Network. In *Proceedings of the Fourth ACM IKDD Conferences on Data Sciences, CODS 2017.* ACM, 14:1–14:10.

[24] Tim Rocktäschel, Edward Grefenstette, Karl Moritz Hermann, Tomas Kocisky, and Phil Blunsom. 2016. Reasoning about Entailment with Neural Attention. In *International Conference on Learning Representations (ICLR).*

[25] R. Sarikaya, G. E. Hinton, and B. Ramabhadran. 2011. Deep belief nets for natural language call-routing. In *2011 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 5680–5683.

[26] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. 2014. Sequence to Sequence Learning with Neural Networks. In *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*. 3104–3112.

[27] Kai Sheng Tai, Richard Socher, and Christopher D. Manning. 2015. Improved Semantic Representations From Tree-Structured Long Short-Term Memory Networks. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing of the Asian Federation of Natural Language Processing, ACL 2015, July 26-31, 2015, Beijing, China, Volume 1: Long Papers*. The Association for Computer Linguistics, 1556–1566.

[28] Laurens van der Maaten and Geoffrey Hinton. 2008. Visualizing Data using t-SNE. *Journal of Machine Learning Research* 9 (2008), 2579–2605.

[29] Puyang Xu and Ruhi Sarikaya. 2013. Convolutional neural network based triangular CRF for joint intent detection and slot filling. In *2013 IEEE Workshop on Automatic Speech Recognition and Understanding, Olomouc, Czech Republic, December 8-12, 2013*. IEEE, 78–83.

[30] Xiang Zhang, Junbo Jake Zhao, and Yann LeCun. 2015. Character-level Convolutional Networks for Text Classification. In *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*. 649–657.