# Incremental Improvement of a Question Answering System by Re-ranking Answer Candidates using Machine Learning


## Abstract

We implement a method for re-ranking top-10 results of a state-of-the-art question answering (QA) system. The goal of our re-ranking approach is to improve the answer selection given the user question and the top-10 candidates. We focus on improving deployed QA systems that do not allow re-training or re-training comes at a high cost. Our re-ranking approach learns a similarity function using n-gram based features using the query, the answer and the initial system confidence as input. Our contributions are: (1) we generate a QA training corpus starting from 877 answers from the customer care domain of T-Mobile Austria, (2) we implement a state-of-the-art QA pipeline using neural sentence embeddings that encode queries in the same space than the answer index, and (3) we evaluate the QA pipeline and our re-ranking approach using a separately provided test set. The test set can be considered to be available after deployment of the system, e.g., based on feedback of users. Our results show that the system performance, in terms of top-n accuracy and the mean reciprocal rank, benefits from re-ranking using gradient boosted regression trees. On average, the mean reciprocal rank improves by 9.15%.


## Introduction

In this work, we examine the problem of incrementally improving deployed QA systems in an industrial setting. We consider the domain of customer care of a wireless network provider and focus on answering frequent questions (focussing on the long tail of the question distribution BIBREF0 ). In this setting, the most frequent topics are covered by a separate industry-standard chatbot based on hand-crafted rules by dialogue engineers. Our proposed process is based on the augmented cross-industry standard process for data mining BIBREF1 (augmented CRISP data mining cycle). In particular, we are interested in

methods for improving a model after its deployment through re-ranking of the initial ranking results. In advance, we follow the steps of the CRISP cycle towards deployment for generating a state-of-the-art baseline QA model. First, we examine existing data (data understanding) and prepare a corpus for training (data preparation). Second, we implement and train a QA pipeline using state-of-the-art open source components (modelling). We perform an evaluation using different amounts of data and different pipeline configurations (evaluation), also to understand the nature of the data and the application (business understanding). Third, we investigate the effectiveness and efficiency of re-ranking in improving our QA pipeline after the deployment phase of CRISP. Adaptivity after deployment is modelled as (automatic) operationalisation step with external reflection based on, e.g., user feedback. This could be replaced by introspective meta-models that allow the system to enhance itself by metacognition BIBREF1 . The QA system and the re-ranking approach are evaluated using a separate test set that maps actual user queries from a chat-log to answers of the QA corpus. Sample queries from the evaluation set with one correct and one incorrect sample are shown in Table TABREF1 .

With this work, we want to answer the question whether a deployed QA system that is difficult to adapt and that provides a top-10 ranking of answer candidates, can be improved by an additional re-ranking step that corresponds to the operationalisation step of the augmented CRISP cycle. It is also important to know the potential gain and the limitations of such a method that works on top of an existing system. We hypothesise that our proposed re-ranking approach can effectively improve ranking-based QA systems.

## Related Work

The broad field of QA includes research ranging from retrieval-based BIBREF2 , BIBREF3 , BIBREF4 , BIBREF5 to generative BIBREF6 , BIBREF7 , as well as, from closed-domain BIBREF8 , BIBREF9 to open-domain QA BIBREF7 , BIBREF10 , BIBREF11 , BIBREF12 . We focus on the notion of improving an already deployed system.

For QA dialogues based on structured knowledge representations, this can be achieved by maintaining and adapting the knowledgebase BIBREF13 , BIBREF14 , BIBREF15 . In addition, BIBREF1 proposes metacognition models for building self-reflective and adaptive AI systems, e.g., dialogue systems, that improve by introspection. Buck et al. present a method for reformulating user questions: their method automatically adapts user queries with the goal to improve the answer selection of an existing QA model BIBREF16 .

Other works suggest humans-in-the-loop for improving QA systems. Savenkov and Agichtein use crowdsourcing for re-ranking retrieved answer candidates in a real-time QA framework BIBREF17 . In Guardian, crowdworkers prepare a dialogue system based on a certain web API and, after deployment, manage actual conversations with users BIBREF18 . EVORUS learns to select answers from multiple chatbots via crowdsourcing BIBREF19 . The result is a chatbot ensemble excels the performance of each individual chatbot. Williams et al. present a dialogue architecture that continuously learns from user interaction and feedback BIBREF20 .

We propose a re-ranking algorithm similar to BIBREF17 : we train a similarity model using n-gram based features of QA pairs for improving the answer selection of a retrieval-based QA system.

Question Answering System

We implement our question answering system using state-of-the-art open source components. Our pipeline is based on the Rasa natural language understanding (NLU) framework BIBREF21 which offers two standard pipelines for text classification: spacy_sklearn and tensorflow_embedding. The main difference is that spacy_sklearn uses Spacy for feature extraction with pre-trained word embedding models and Scikit-learn BIBREF22 for text classification. In contrast, the tensorflow_embedding pipeline trains custom word embeddings for text similarity estimation using TensorFlow BIBREF23 as machine

learning backend. Figure FIGREF5 shows the general structure of both pipelines. We train QA models using both pipelines with the pre-defined set of hyper-parameters. For tensorflow_embedding, we additionally monitor changes in system performance using different epoch configurations. Further, we compare the performances of pipelines with or without a spellchecker and investigate whether model training benefits from additional user examples by training models with the three different versions of our training corpus including no additional samples (kw), samples from 1 user (kw+1u) or samples from 2 users (kw+2u) (see section Corpora). All training conditions are summarized in Table TABREF4 . Next, we describe the implementation details of our QA system as shown in Figure FIGREF5 : the spellchecker module, the subsequent pre-processing and feature encoding, and the text classification. We include descriptions for both pipelines.

Spellchecker We address the problem of frequent spelling mistakes in user queries by implementing an automated spell-checking and correction module. It is based on a Python port of the SymSpell algorithm initialized with word frequencies for German. We apply the spellchecker as first component in our pipeline.

Pre-Processing and Feature Encoding. The spacy_sklearn pipeline uses Spacy for pre-processing and feature encoding. Pre-processing includes the generation of a Spacy document and tokenization using their German language model de_core_news_sm (v2.0.0). The feature encoding is obtained via the vector function of the Spacy document that returns the mean word embedding of all tokens in a query. For German, Spacy provides only a simple dense encoding of queries (no proper word embedding model). The pre-processing step of the tensorflow_embedding pipeline uses a simple whitespace tokenizer for token extraction. The tokens are used for the feature encoding step that is based on Scikit-learn's CountVectorizer. It returns a bag of words histogram with words being the tokens (1-grams).

Text Classification. The spacy_sklearn pipeline relies on Scikit-learn for text classification using a support

vector classifier (SVC). The model confidences are used for ranking all answer candidates; the top-10 results are returned.

Text classification for tensorflow_embedding is done using TensorFlow with an implementation of the StarSpace algorithm BIBREF24 . This component learns (and later applies) one embedding model for user queries and one for the answer id. It minimizes the distance between embeddings of QA training samples. The distances between a query and all answer ids are used for ranking.

Corpora

In this work, we include two corpora: one for training the baseline system and another for evaluating the performance of the QA pipeline and our re-ranking approach. In the following, we describe the creation of the training corpus and the structure of the test corpus. Both corpora have been anonymised.

Training Corpus. The customer care department provides 877 answers to common user questions. Each answer is tagged with a variable amount of keywords or key-phrases ( INLINEFORM0 , INLINEFORM1 ), 3338 in total. We asked students to augment the training corpus with, in total, two additional natural example queries. This process can be scaled by crowdsourcing for an application in productive systems that might include more answers or that requires more sample question per answer or both. The full dataset contains, on average, INLINEFORM2 sample queries per answer totalling in 5092 queries overall. For model training, all questions (including keywords) are used as input with the corresponding answer as output. We generated three versions of the training corpus: keywords only (kw, INLINEFORM3 ), keywords with samples from 1 user (kw+1u, INLINEFORM4 ) and keywords with samples from 2 users (kw+2u, INLINEFORM5 ).

Evaluation Corpus.

The performance of the implemented QA system and of our re-ranking approach is assessed using a separate test corpus. It includes 3084 real user requests from a chat-log of T-Mobile Austria, which are assigned to suitable answers from the training corpus (at most three). The assignment was performed manually by domain experts of the wireless network provider. We use this corpus for estimating the baseline performance of the QA pipeline using different pipeline configurations and different versions of the training corpus. In addition, we use the corpus for evaluating our re-ranking approach per cross-validation: we regard the expert annotations as offline human feedback. The queries in this corpus contain a lot of spelling mistakes. We address this in our QA pipeline generation by implementing a custom spell-checking component.

Baseline Performance Evaluation

We evaluate the baseline model using all training configurations in Table TABREF4 to find a well-performing baseline for our re-ranking experiment. We use the evaluation corpus as reference data and report the top-1 to top-10 accuracies and the mean reciprocal rank for the top-10 results (MRR@10) as performance metrics. For computing the top-n accuracy, we count all queries for which the QA pipeline contains a correct answer on rank 1 to n and divide the result by the number of test queries. The MRR is computed as the mean of reciprocal ranks over all test queries. The reciprocal rank for one query is defined as INLINEFORM0 : The RR is 1 if the correct answer is ranked first, INLINEFORM1 if it is at the second rank and so on. We set RR to zero, if the answer is not contained in the top-10 results.

Results. Figure FIGREF10 shows the accuracy and MRR values for all conditions. We only restrict tensorflow_embedding to the default number of epochs which is 300. At the corpus level, we can observe that the accuracy and the MRR increase when training with additional user annotations for all pipeline configurations. For example, the spacy_sklearn pipeline without spell-checking achieves a top-10 accuracy of INLINEFORM0 and a MRR of INLINEFORM1 when using the kw training corpus with

keywords only. Both measures increase to INLINEFORM2 and INLINEFORM3 , respectively, when adding two natural queries for training. In some cases, adding only 1 user query results in slightly better scores. However, the overall trend is that more user annotations yield better results.

In addition, we observe performance improvements for pipelines that use our spell-checking component when compared to the default pipelines that do not make use of it: The spacy_sklearn kw+2u condition performs INLINEFORM0 better, the tensorflow_embedding kw+2u condition performs INLINEFORM1 better, in terms of top-10 accuracy. We can observe similar improvements for the majority of included metrics. Similar to the differentiation by corpus, we can find cases where spell-checking reduces the performance for a particular measure, against the overall trend.

Overall, the tensorflow_embedding pipelines perform considerably better than the spacy_sklearn pipeline irrespective of the remaining parameter configuration: the best performing methods are achieved by the tensorflow_embedding pipeline with spell-checking. Figure FIGREF11 sheds more light on this particular setting. It provides performance measures for all corpora and for different number of epochs used for model training. Pipelines that use 300 epochs for training range among the best for all corpora. When adding more natural user annotations, using 100 epochs achieves similar or better scores, in particular concerning the top-10 accuracy and the MRR. Re-ranking the top-10 results can only improve the performance in QA, if the correct answer is among the top-10 results. Therefore, we use the tensorflow_embedding pipeline with spellchecking, 100 epochs and the full training corpus as baseline for evaluating the re-ranking approach.

Re-Ranking Approach

Our re-ranking approach compares a user query with the top-10 results of the baseline QA system. In contrast to the initial ranking, our re-ranking takes the content of the answer candidates into account

instead of encoding the user query only. Our algorithm compares the text of the recent user query to each result. We include the answer text and the confidence value of the baseline system for computing a similarity estimate. Finally, we re-rank the results by their similarity to the query (see Algorithm SECREF5 ).

a user query INLINEFORM0 ; the corresponding list of top-10 results INLINEFORM1 including an answer INLINEFORM2 and the baseline confidence INLINEFORM3 ; an updated ranking INLINEFORM4 INLINEFORM5 INLINEFORM6 INLINEFORM7 INLINEFORM8 sort R' by confidences c', descending INLINEFORM9 INLINEFORM10 Re-Ranking Algorithm

We consider a data-driven similarity function that compares linguistic features of the user query and answer candidates and also takes into account the confidence of the baseline QA system. This similarity estimate shall enhance the baseline by using an extended data and feature space, but without neglecting the learned patterns of the baseline system. The possible improvement in top-1 accuracy is limited by the top-10 accuracy of the baseline system ( INLINEFORM0 ), because our re-ranking cannot choose from the remaining answers. Figure FIGREF12 shows how the re-ranking model is connected to the deployed QA system: it requires access to its in- and outputs for the additional ranking step.

We consider the gradient boosted regression tree for learning a similarity function for re-ranking similar to BIBREF17 . The features for model training are extracted from pre-processed query-answer pairs. Pre-processing includes tokenization and stemming of query and answer and the extraction of uni-, bi- and tri-grams from both token sequences. We include three distance metrics as feature: the Jaccard distance, the cosine similarity, and the plain number of n-gram matches between n-grams of a query and an answer.

a train- and test split of the evaluation corpus INLINEFORM0 , each including QA-pairs as tuples

INLINEFORM1 ; the pre-trained baseline QA model for initial ranking INLINEFORM2 and the untrained re-ranking model INLINEFORM3 . evaluation metrics. training of the re-ranking model INLINEFORM4 INLINEFORM5 INLINEFORM6 INLINEFORM7 *R contains top-10 results INLINEFORM8 continue with next QA pair add positive sample INLINEFORM9 *confidence for INLINEFORM10 INLINEFORM11 INLINEFORM12 add negative sample INLINEFORM13 random INLINEFORM14 INLINEFORM15 INLINEFORM16 INLINEFORM17 INLINEFORM18 evaluation of the re-ranking model INLINEFORM19 INLINEFORM20 INLINEFORM21 *top-10 baseline ranking INLINEFORM22 *apply re-ranking INLINEFORM23 INLINEFORM24 Evaluation Procedure (per Data Split)

Re-Ranking Performance Evaluation

We compare our data-driven QA system with a version that re-ranks resulting top-10 candidates using the additional ranking model. We want to answer the question whether our re-ranking approach can improve the performance of the baseline QA pipeline after deployment. For that, we use the evaluation corpus ( INLINEFORM0 ) for training and evaluating our re-ranking method using 10-fold cross-validation, i.e., INLINEFORM1 of the data is used for training and INLINEFORM2 for testing with 10 different train-test splits.

The training and testing procedure per data split of the cross-validation is shown in Algorithm SECREF5 . For each sample query INLINEFORM0 in the train set INLINEFORM1 , we include the correct answer INLINEFORM2 and one randomly selected negative answer candidate INLINEFORM3 for a balanced model training. We skip a sample, if the correct answer is not contained in the top-10 results: we include INLINEFORM4 of the data (see top-10 accuracy of the baseline QA model in Figure FIGREF11 ). The baseline QA model INLINEFORM5 and the trained re-ranking method INLINEFORM6 are applied to all sample queries in the test set INLINEFORM7 . Considered performance metrics are computed using the re-ranked top-10 INLINEFORM8 . We repeat the cross-validation 5 times to reduce effects introduced by

the random selection of negative samples. We report the average metrics from 10 cross-validation folds and the 5 repetitions of the evaluation procedure.

Results. The averaged cross-validation results of our evaluation, in terms of top-n accuracies and the MRR@10, are shown in Table TABREF15 : the top-1 to top-9 accuracies improve consistently. The relative improvement decreases from INLINEFORM0 for the top-1 accuracy to INLINEFORM1 for the top-9 accuracy. The top-10 accuracy stays constant, because the re-ranking cannot choose from outside the top-10 candidates. The MRR improves from INLINEFORM2 to INLINEFORM3 ( INLINEFORM4 ).

## Discussion

Our results indicate that the accuracy of the described QA system benefits from our re-ranking approach. Hence, it can be applied to improve the performance of already deployed QA systems that provide a top-10 ranking with confidences as output. However, the performance gain is small, which might have several reasons. For example, we did not integrate spell-checking in our re-ranking method which proved to be effective in our baseline evaluation. Further, the re-ranking model is based on very simple features. It would be interesting to investigate the impact of more advanced features, or models, on the ranking performance (e.g., word embeddings BIBREF26 and deep neural networks for learning similarity functions BIBREF3 , BIBREF4 ). Nevertheless, as can be seen in examples 1, 2 and 4 in Table TABREF1 , high-ranked but incorrect answers are often meaningful with respect to the query: the setting in our evaluation is overcritical, because we count incorrect, but meaningful answers as negative result. A major limitation is that the re-ranking algorithm cannot choose answer candidates beyond the top-10 results. It would be interesting to classify whether an answer is present in the top-10 or not. If not, the algorithm could search outside the top-10 results. Such a meta-model can also be used to estimate weaknesses of the QA model: it can determine topics that regularly fail, for instance, to guide data labelling for a targeted improvement of the model, also known as active learning BIBREF27 , and in combination with techniques

from semi-supervised learning BIBREF5 , BIBREF28 .

Data labelling and incremental model improvement can be scaled by crowdsourcing. Examples include the parallel supervision of re-ranking results and targeted model improvement as human oracles in an active learning setting. Results from crowd-supervised re-ranking allows us to train improved re-ranking models BIBREF17 , BIBREF19 , but also a meta-model that detects queries which are prone to error. The logs of a deployed chatbot, that contain actual user queries, can be efficiently analysed using such a meta-model to guide the sample selection for costly human data augmentation and creation. An example of a crowdsourcing approach that could be applied to our QA system and data, with search logs can be found in BIBREF0 .

## Conclusion

We implemented a simple re-ranking method and showed that it can effectively improve the performance of QA systems after deployment. Our approach includes the top-10 answer candidates and confidences of the initial ranking for selecting better answers. Promising directions for future work include the investigation of more advanced ranking approaches for increasing the performance gain and continuous improvements through crowdsourcing and active learning.