Answer ranking in Community Question Answering: a deep learning approach

Trabajo Fin de Máster

Lucas Valentin Garcia

Trabajo de investigación para el

Máster en Tecnologías del lenguaje

Universidad Nacional de Educación a Distancia

Dirigido por

Prof. Dr. D. Álvaro Rodrigo

Prof. Dr. D. Roberto Centeno

Septiembre 2022

Abstract

Community Question Answering is the field of computational linguistics that deals with problems derived from the questions and answers posted to websites such as Quora or Stack Overflow. Among some of these problems we find the issue of ranking the multiple answers posted in reply to each question by how informative they are in the attempt to solve the original question.

This work tries to advance the state of the art on answer ranking for community Question Answering by proceeding with a deep learning approach. We started off by creating a large data set of questions and answers posted to the Stack Overflow website.

We then leveraged the natural language processing capabilities of dense embeddings and LSTM networks to produce a prediction for the accepted answer attribute, and present the answers in a ranked form ordered by how likely they are to be marked as accepted by the question asker.

We also produced a set of numerical features to assist with the answer ranking task. These numerical features were either extracted from metadata found in the Stack Overflow posts or derived from the questions and answers texts.

We compared the performance of our deep learning models against a set of forest and boosted trees ensemble methods and found that our models could not improve the best baseline results. We speculate that this lack of performance improvement versus the baseline models may be caused by the large number of out of vocabulary words present in the programming code snippets found in the questions and answers text.

We conclude that while a deep learning approach may be helpful in answer ranking problems new methods should be developed to assist with the large number of out of vocabulary words present in the programming code snippets.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Community Question Answering [1] is the field of computational linguistics that relates to the analysis of questions and answers posted by users of community question answering websites such as Quora[2] or Stack Overflow[3]. In these websites users may freely post questions either in an unrestricted domain (Quora) or in sites specific domains (Stack Overflow). Other website users will then freely post answers that aim to answer the original question, making use of their personal experience and knowledge, or other Internet resources, such as Wikipedia or topic specific websites or manuals.

This simple mechanic introduces several challenges to both site users and moderators or maintainers. First of all the huge volume of posts (either questions or answers to these questions) requires some automated way to organize, categorize and quantify all this information. Some of the most common problems found in community question answering sites related to the questions deal with detecting duplicated questions that were answered previously or detecting questions that do not have a definite answer as they lend themselves to a debate among peers rather than a unique answer. On the array of problems related to the answers we may find the need to select the best answers, either by highlighting a single answer or sorting all the possible answers in a ranked way. Also the answers may be classified depending on whether they are informative or they

simply raise a point in a debate but do not offer a conclusive answer.

Given that all these problems require sifting through large amounts of information it is immediate to apply techniques and procedures from the machine learning and natural language processing fields to attack some of these problems.

In this work we will propose a method to rank multiple answers to questions in community question answering sites found in a data set compiled from Stack Overflow postings. This ranking will be performed using deep learning models fed with dense embeddings of textual data, in addition to numerical features extracted both automatically and manually from the data set.

The reasoning behind these methods is that we believe that it is possible to augment the answer ranking capabilities of a deep learning model accepting a sequence of tokens as input with numerical features derived from the metadata relative to users and posts. In some cases these numerical features will be obtained immediately from the metadata while other numerical features will be created after parsing these metadata and the questions and answers texts.

1.1 Objectives

This MsC dissertation aims to advance the state of the art on community Question Answering (cQA). More specifically we will attempt to delve into new methods to predict the answer selected by the user who posted the question. The reasons to select a concrete answer may be either attending to the usefulness of the answer, how factual it is or even how soon it was posted. In order to better represent the results of the selected answer prediction we will produce a ranking of the multiple answers per question, sorted by how confident the model is that a given answer will be selected as accepted by the user.

Among the more specific objectives we will tackle we could enumerate:

- Produce a relevant data set composed of information retrieved from community Question Answering sites. This data set will be composed by questions and answers text at a minimum, and may be incremented with

additional metadata from either the user asking the question or the user answering the question.

- Analyze the data set and extract additional predictive features in addition to the existing metadata. These predictive features should increment the capabilities of the prediction model in order to improve the prediction results versus a simpler model that only makes use of the textual input.

- Identify sensible metrics that allow a realistic evaluation of the performance reached by the prediction model. Ideally these metrics should point out when a model is not able to rank the accepted answer in first place but is still able to place the answer in a relevant position so it can be clearly distinguished from the rest of answers.

- Carry out different experiments that allow us to gauge how each of the different sections of the models contribute to their performance. We will experiment with different options for the contextual embedding, number of recurrent and predictive layers, and also whether considering only the answer or the ensemble of the question and the answer effect the model performance.

1.2 Structure

This dissertation is organized as follows:

- On Chapter 2 we will first review the current state of the art on the community question answering, along with some of the most relevant problems in this field.

- On Chapter 3 we will present the data set used in our research, describing the process to obtain it and then describing the features in the data set. We will also introduce the methodology to evaluate the experimental results we obtain. Finally we will introduce a set of sensible baseline models that

provide a minimal performance level against which we will later compare our models.

- On Chapter 4 we will describe the experiments we propose, including different ablation studies.
- On Chapter 5 we will review the results we obtain from the experiments, both from the initial baselines and from the more advanced models we created.
- On Chapter 6 we will discuss the results presented before and the relations between the baseline results and the advanced models results.
- Finally on Chapter 7 we will draw some conclusions from the work presented here and introduce some future line of work related to the CQA task.

Chapter 2

State of the art

2.1 Introduction

In this section we will describe the current state of the art for community question answering. We will first describe the differences between classical question answering and community question answering problems. Then we will review the current approaches to community question answering and introduce the answer ranking problem and how it has been tackled in the past. Finally we will review some of the novel data sets related to community question answering.

2.2 Community Question Answering

Community Question Answering is the field of computational linguistics that deals with the problems arising from questions and answers posted by users to websites such as Quora or Stack Overflow [4].

It is a field with a relatively long history, since the online communities where users may post questions to be answered by other users have now existed for many years.

Community questions answering must not be confused with the classical question answering field where machine learning methods are used to obtain

answers to questions about a given corpus. Question answering problems have a much more close ended nature since it is expected that the question may be fully answered with information present in the input corpus, while community question answering problems are much more open ended in the sense that questions posted by the users may refer a plurality of fields and may require external sources of information to be properly answered.

2.3 Community Question Answering problems

We can find many examples of different research topics, all of them inside the wide field of CQA ([1], [5]).

Some of the most common problems in CQA are related to the questions contents, such as duplicate question detection, question answerability prediction or quality assessment. Other classical CQA problems relate to the answers quality, such as the answer quality assessment, best answer prediction and, finally, answer ranking.

We will now see each of these problems in more detail. First duplicate question detection focuses on identifying existing questions to which newly posted question most resemble so that question answerers don't spend time on these questions. For instance Xu et al. [6] showed that a simple SVM approach may work better than deep learning methods for related question prediction.

The problem of question answerability prediction deals with the fact that some questions remain unanswered even years after they have been posted. This may be due to bad wording on the question or the fact that the question is indeed difficult to be answered, especially by people that do not have the questioner's experience or background. Asaduzzaman et al. [7] found that the number of unanswered questions has increased significantly in the previous 2 years. Then they proposed a classifier that predicts how long a question will remain unanswered that reaches precision values of 0.38 and recall values of 0.45.

Nowadays, CQA sites are experiencing a surge in subpar content. The task of question quality assessment tries to identify high value questions for which it is more worth to spend time answering them. Maxwell Harper et al. [8] created a model that reaches 89.7% classification accuracy identifying informative questions from those only posted for conversational reasons.

Many other problems in CQA relate to answers rather than questions. For instance the problem of answer quality assessment refers to the methods proposed to measure the content of individual answer quality, distinguishing between high quality and low quality answers that bloat the answers page. Suguu et al. [9] achieved a 75.2% accuracy predicting high quality questions with a bidirectional LSTM network with feature fusion model.

Another problem related to answers in CQA is the best answer prediction, as selected by the user posting the original question. Gkotsis et al. [10] used a fusion of textual features with user and answer ratings features to achieve an 84% average precision and 70 % recall. It is to note that they reached this performance thanks to the discretization of their features so they are evaluated in the context of all the possible answers to the questions.

2.4 Answer ranking in Community Question Answering

There have been different attempts at ranking answers to questions in community question answering sites. Some of these attempts followed classical approaches like [11] which used similarity measure between questions and answers to rank the different answers. The work leveraged snippets of web search results for query expansion in answer ranking.

Ginsca et al. [12] used in-depth analysis of the information provided by the users in their profiles in order to discriminate features that are correlated to expertise, focusing then on profile and activity related features to rank answers.

Dalip et al. [13] followed a learning to rank approach based on different

groups of features like features referred to the users, stylistic or structural features.

Bougessa et al. [14] aimed to identify expert users, on the basis that their answers have more relevance than answers posted by other users. The authority scores of users were modeled with a mixture of gamma distributions.

Amancio et al. [15] used recency and quality as criteria to rank answers, on the grounds that a recent and high quality answer is preferable to a high quality, date answer since users value more highly answers with more current content. The work considered an answer as recent not by how new is the date of creation or editing of a given answer, but how current is the content of the answer .

More recently neural network approaches made possible by the advent of high performance GPUs have proved very valuable in the context of answer ranking in community question answering.

Zhou et al. [16] used a gated recurent unit (GRU) [17] with thread-level features to rank the answers. The question and their answers were combined to create question-answer pairs, which are used as input to the model to capture the semantic features and then fed into the model to rank them.

Chen et al. [18] introduced a positional attention based recurrent neural network (RNN) model, which incorporates the positional context of the question words into the answers' attentive representations, assuming that if a word in the question text occurs in an answer sentence, the neighboring words should be given more attention since they intuitively contain more valuable information.

Following the recent trend in many natural language problems, modern language models based on transformer with attention mechanisms that take an embedded representation input like BERT [19] have also been applied to the answer ranking task in community question answering.

Laskat et al. [20] used ElMo and BERT models both from a fine tuning approach as well as from a feature extraction approach to achieve state of the art results in the Semeval 2016 data sets.

Du et al. [21] used Keyword BERT, a variation to the BERT model that adds a keyword-attentive layer that highlights the domain keywords to enhance the semantic interaction of the sentence pair supplied during training, in order to rank answers in a Buddhism related data set.

Maia et al. [22] combined embedded representations of question and answer data with user defined tags for each question to form the input to the transformer model. In this way the model could derive the specific question domain for better answer ranking.

Wang et al. [23] used a Chinese BERT model to extract embedded representations of question and answer text of a Chinese community answering site. Then, two matching strategies (Full matching and Attentive matching) were introduced in the matching layer to complete the interaction between sentence vectors. Finally a Bi-GRU network combined question and answer representations to obtain a measure of similarity to perform answer ranking.

2.5 Community Question Answering data sets

It is to note some of the latest efforts in the community question answering have focused on the curation of data sets that may enable new research lines and works. This trend started with the Yahoo! Answers dataset [24] which combined content from community question answering with manually generated annotations to obtain a dataset suitable for machine learning processing.

The the Community Question Answering task in some of the past SemEval editions ([25, 26, 27]) introduced a new data set retrieved from the Qatar Living forum [1], and proposed tasks related to question and answer similarity, and question to question similarity. On the 2016 and 2017 editions, the SemEval main task was to rank the answers to a question by how useful they were to answer the question. In addition, the 2017 edition introduced two new subtasks: rank the similar questions according to their similarity to the original

[1] https://www.qatarliving.com/

question, and rank the answer posts according to their relevance with respect to the question. The metric used in these SemEval tasks is the mean average precision (MAP).

More recently the AmazonQA dataset [28] incorporated user generated reviews of products to the usual question and answer data set fields so new tasks such as automatic answer retrieval from a question review pair may be performed.

One common pitfall of all these data sets is that they mostly only collect question and answer text with few if any additional metadata. This complicates the task of producing additional non textual features that may be used in problems like answer ranking.

2.6 Conclusions

As we have seen there have been multiple attempts at answer ranking in community question answering. While it was common to use non textual features before the advent of large neural network models, we have not found many instances of modern models that make use of these non textual features. It may be fruitful then to combine modern neural network models with an ensemble of non textual features and verify whether the addition of these features improves the answer ranking capabilities.

In addition it is required to collect a new data set that combines a large corpus of question and answer pairs with addition data from users and posts that allow the extraction of informative features for answer ranking.

Chapter 3

Evaluation framework

3.1 Introduction

In this chapter we will introduce the elements that will allow us to conduct experiments on the answer ranking problem. We will start by describing the data set that has been compiled from Stack Overflow posts in order to perform the answer ranking task over community question answering content. Next we will introduce different metrics that we will use to assess the performance of the proposed methods. Finally we will present a set of baseline models based around random forests and gradient boosted trees that make use of simple Bag of Words text representation.

3.2 Data

3.2.1 Stack Overflow data

We will make use of data from the largest community question answering site nowadays as is Stack Overflow. In addition the fact that data from such site is easily accessible through Google BigQuery [29] will greatly simplify the data collection task.

Stack Overflow is a community question answering created in 2008 and nowa-

days it has over 14 million registered users [30]. More than 21 million questions and 31 million answers have been posted to the site. Based on the type of tags assigned to questions, the top eight most discussed topics on the site are: JavaScript, Java, C#, PHP, Android, Python, jQuery, and HTML.

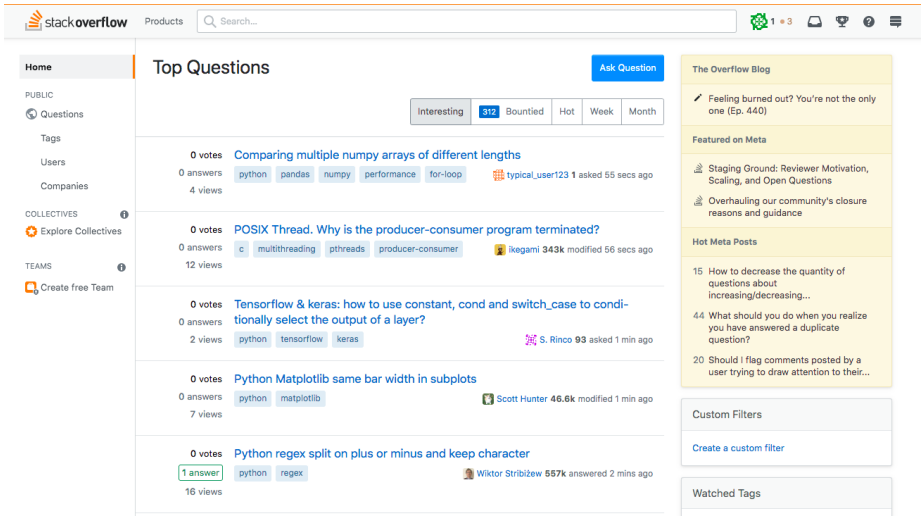


Figure 3.1: Capture of the Stack Overflow site home page

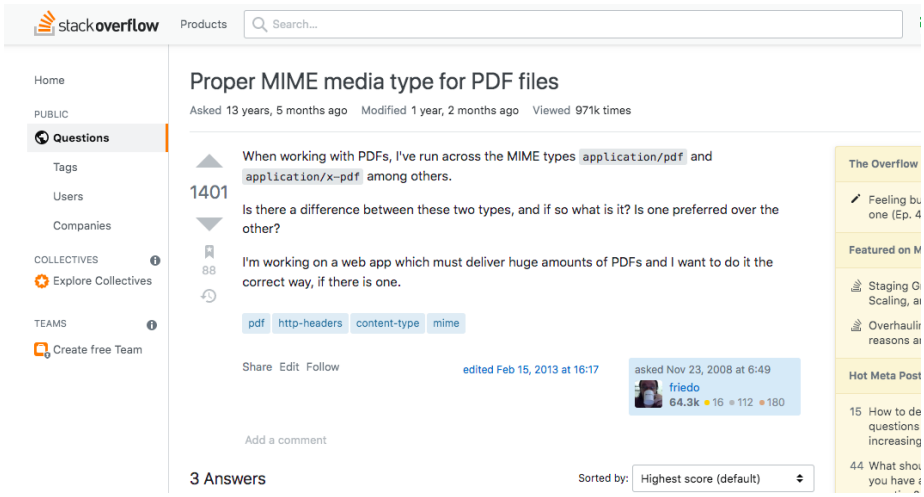


Figure 3.2: Capture of an example question on Stack Overflow

There are many data items available for Stack Overflow posts. We will review some of these data items, focusing on the ones most relative to our answer ranking task.

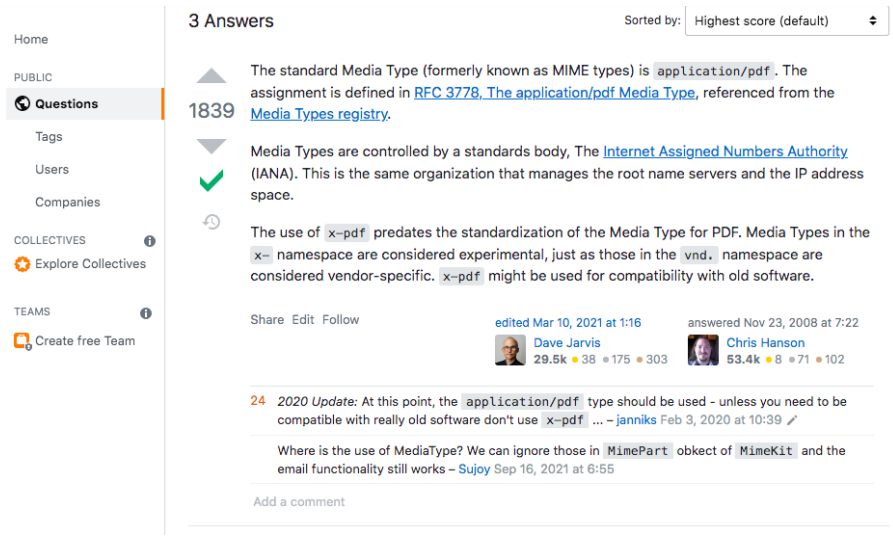


Figure 3.3: Capture of an example answer on Stack Overflow

Given that the Stack Overflow site data we are accessing is available through the database in Google BigQuery we find that all data is available as database tables. More specifically, all the Stack Overflow data is scattered through different tables as we can see in figure 3.4

We will now briefly describe each of these tables with a special focus on the ones holding items of interest for our task.

1. badges: this table holds the badges collected by each user of the site either by posting questions or answers. This table will not be of interest for our task.

2. comments: this table holds the comments posted by users to some of the answers. Usually these comments are posted by users who also replied to the original question so they may not be very informative but rather engage in punctilious analysis that may often miss the point of the original question. This table will not be of interest for our task.

3. post_history: this table holds details on when questions or answers where edited. This table will not be of interest for our task.

4. post_links: this table holds details on which posts in the site link to other

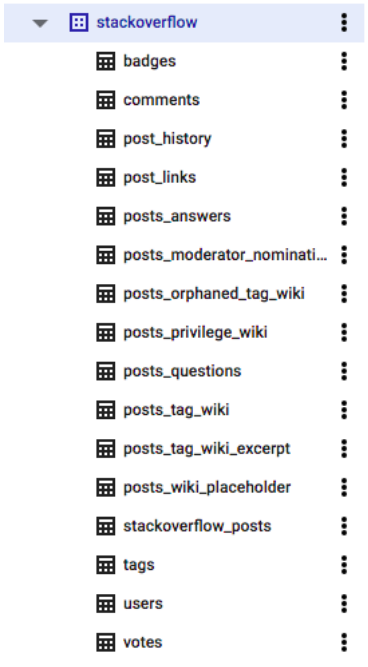


Figure 3.4: Capture from BigQuery site where all the tables with Stack Overflow data are shown

posts. This table will not be of interest for our task.

5. posts_answers: This table holds the answers posted to the questions in the site. This table if of great interest for our task so we will review it in detail, describing each of the columns this tables holds.

   (a) id: this column holds the answer ID. It is not directly applicable to our task but we will use it to link each answer to the question it refers to.

   (b) title: this column holds no values so it will not be of interest for our task.

   (c) body: this column holds the text of the answer. It will be very important for our task.

   (d) accepted_answer_id: this column holds no values so it will not be of interest for our task.

(e) answer_count: this column holds no values so it will not be of interest for our task.

(f) comment_count: this column holds the number of comments posted to the answer. It may be important so we will use as a numerical feature in our experiments.

(g) community_owned_date: this column holds no values so it will not be of interest for our task.

(h) creation_date: this columns holds the timestamp for the answer creation. We will not use it for our task.

(i) favorite_count: this column holds no values so it will not be of interest for our task.

(j) last_activity_date: this column holds the timestamp for when the answer last saw any activity like posting or edits. We will not use it for our task.

(k) last_edit_date: this column holds the timestamp for when the answer was last edited. We will not use it for our task.

(l) last_editor_display_name: this column holds no values so it will not be of interest for our task.

(m) last_editor_user_id: this column holds the user ID that last edited the answer. We will not use it for our task.

(n) owner_display_name: this column holds no values so it will not be of interest for our task.

(o) owner_user_id: this column holds the user ID that posted the answer. We will not use it for our task.

(p) parent_id: this columns holds the ID for the question it refers to. It is not directly applicable to our task but we will use it to link each answer to the question it refers to.

(q) post_type_id: this column holds the type of post it refers to. All rows contain value 2 for answer hence the column has no informative value and we will not use it for our task.

(r) score: this column holds the score obtained by the answer after users up-vote and down-vote it, attending to its contents. It may be very important for our task so we will use it as a numerical feature.

(s) tags: this column holds no values so it will not be of interest for our task.

(t) view_count: this column holds no values so it will not be of interest for our task.

6. posts_moderator_nomination: this table holds the self nominations from users who would like to be promoted to moderators in the site. This table will not be of interest for our task.

7. posts_orphaned_tag_wiki: this table holds information from the wiki section of Stack Overflow. This table will not be of interest for our task.

8. posts_privilege_wiki: This table is also related to the wiki section of Stack Overflow and will not be of interest for our task.

9. posts_questions: this table holds the questions posted by users of the site. This table if of great interest for our task so we will review it in detail.

(a) id: this column holds the question ID. It is not directly applicable to our task but we will use it to link each answer to the question it refers to.

(b) title: this column holds the question title, which introduces the topic of the question. It will be very important for our task.

(c) body: this column holds the question body, where the question is developed. It will be very important for our task.

(d) accepted_answer_id: this column holds the ID for the accepted answer. It is not directly applicable to our task but we will use it to assess whether each answer was marked as accepted or not.

(e) answer_count: this column holds the number of answers posted to each question. It may be important so we will use as a numerical feature in our experiments.

(f) comment_count: this column holds the number of answers posted to each question. It may be important so we will use as a numerical feature in our experiments.

(g) community_owned_date: this column holds no values so it will not be of interest for our task.

(h) creation_date: this columns holds the timestamp for the question creation. We will not use it for our task.

(i) favorite_count: this column holds the count for the number of times any user (who may or may not be the one who posted the question) set the question as favourite in order to track its status. We will not use it for our task.

(j) last_activity_date: this column holds the timestamp for when the answer last saw any activity like posting or edits. We will not use it for our task.

(k) last_edit_date: this column holds the timestamp for when the answer was last edited. We will not use it for our task.

(l) last_editor_display_name: this column holds no values so it will not be of interest for our task.

(m) last_editor_user_id: this column holds the user ID that last edited the answer. We will not use it for our task.

(n) owner_display_name: this column holds no values so it will not be of interest for our task.

(o) owner_user_id: this column holds the user ID that posted the answer. We will not use it for our task.

(p) parent_id: this columns holds the ID for the answer it refers to. It is not directly applicable to our task but we will use it to link each answer to the question it refers to.

(q) post_type_id: this column holds the type of post it refers to. All rows contain value 1 for questions hence the column has no informative value and we will not use it for our task.

(r) score: this column holds the score obtained by the question after site users up-vote and down-vote it, attending to its contents. It may be very important for our task so we will use it as a numerical feature.

(s) tags: this columns holds the tags sets by the user posting the question. These tags usually describe the domain to which the question belongs and could be used to separate questions into different domains. We will not be using for our experiments since we are not interested in discriminating questions by domain for now.

(t) view_count: this column holds the count for the number of times the question was accessed by users on the site. We will not use it for our task.

10. posts_tag_wiki: This table is related to the wiki section of the site and will not be of interest for our task.

11. posts_tag_wiki_excerpt: This table is related to the wiki section of the site and will not be of interest for our task.

12. posts_wiki_placeholder: This table is related to the wiki section of the site and will not be of interest for our task.

13. stackoverflow_posts: this table is a legacy element that used to hold both questions, answers and comments. It is recommended by the data set

documentation not to use it and use tables with names beginning with posts_ instead.

14. tags: this table holds the tags assigned by users to each question. These tags may be used to categorize the question into different domains. This table will not be of interest for our task.

15. users: this table holds data on users posting questions and answers to the site. This table if of great interest for our task so we will review it in detail.

    (a) id: this column holds the question ID. It is not directly applicable to our task but we will use it to link each user to the answer it posts.

    (b) display_name: this columns shows the pseudonym that the user choose to represent herself in the site. We will not use it for our task.

    (c) age: this columns holds the user age. It may be important so we will use as a numerical feature in our experiments.

    (d) creation_date: this column holds the timestamp when the user created the account in the site. We will not use it for our task.

    (e) last_access_date: this column holds the timestamp when the user last accessed the sire. We will not use it for this task.

    (f) location: this columns holds the user location. It may be important so we will use as a numerical feature in our experiments.

    (g) reputation: this column holds the user reputation reached as a consequence of posting questions and answers to the site. It may be important so we will use as a numerical feature in our experiments.

    (h) up_votes: this columns holds the number of up votes received by the user contributions. It may be important so we will use as a numerical feature in our experiments.

    (i) down_votes: this columns holds the number of down votes received by the user contributions. It may be important so we will use as a

numerical feature in our experiments.

(j) views: this column holds the count of accesses to the user profile in the site. It may be important so we will use as a numerical feature in our experiments.

(k) profile_image_url: this column holds the URL address for the user profile picture. It may be important so we will use as a numerical feature in our experiments.

(l) website_url: this column holds the URL address for user website. It may be important so we will use as a numerical feature in our experiments.

16. votes: this table holds data on votes to questions and answers casted by users. This table will not be of interest for our task.

3.2.2 Data retrieval

As we have said we will retrieve Stack Overflow site data through the Google BigQuery service. BigQuery provides an Structured Query Language (SQL) interface so data may accessed through any computing environment that supports such interface. Given that as we have described in the previous section the data that we will be using is spread across several tables we will need to create SQL queries that join the data in these tables.

A detailed description of the SQL queries used to obtain our data would be beyond the scope of this dissertation but we will summarize some of the most important features of the SQL queries in this section. The full text to the SQL queries may be found at Appendix .1.

First we will make use of the question and answer ID codes found in the question and answer tables to join each of the answers in the answers table with the respective questions being answered. In addition we will also make use of the user ID referring to the user who created each answer to join user data to the answer data.

Once we have joined the data from the questions and answers tables we will extract data from the relevant columns identified before to form our data set.

In addition to these join commands we will also perform some filtering in the data: we will only grab data from questions with more than one answer and from questions with an accepted answer. The goal of this filtering is to ensure that all questions in our data set have an accepted answer and more than one answer so ranking them is a non trivial issue.

For our experiments we will our models with balanced data respect to the number of accepted answers; this is, we will use data collections with the same number of accepted answers as of not accepted answers.

Finally we will filter posts with questions asked in 2016 for the training set and posts with questions asked in 2017 for the test test.

3.2.3 Additional Feature Generation

After performing data retrieval from BigQuery we will produce extra features derived from this retrieved data. In particular we will produce some features relative to the question and answer texts such as the word count (n_words), the average word length (avg_word_len), the sentence count (n_sent) and the average and maximum number of words per sentence (avg_n_word_sent, max_n_word_sent). In addition we will also produce an additional feature that will count the number of common words between question and answer (qa_n_common), and an additional feature that will inform whether there were HTTP hyperlinks in the answer body (a_has_urls).

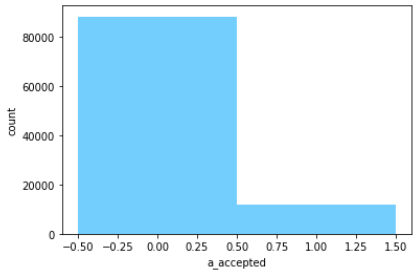
Also for some of the columns in the retrieved data we will encode the presence of absence of data as a Boolean variable, given that the exact content of these columns is not informative but rather the presence or absence is. More specifically we will create Boolean variables to flag the presence of information about the user such as the "About" field of the user's profile, the user's location, the existence of a profile image or a website in the user's profile (has_user_about, has_user_location, has_user_profile_image_url, has_user_website_url).

3.2.4 Exploratory Data Analysis

In this section we will examine closely the features produced in the previous section.

First of all we will check the distribution of a_accepted as this is the target of our answer selection task, as can be seen in figure 3.5.

Figure 3.5: a_accepted class distribution



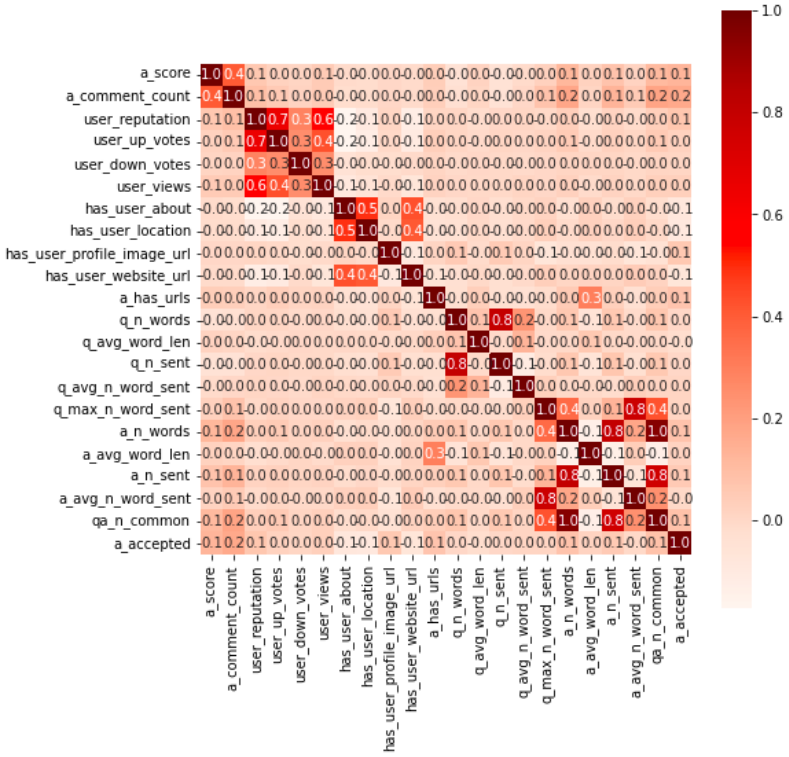
As we can see there is a great imbalance as there are many more not accepted answers that accepted answers. This is something perfectly logical since by definition only one answer per question will be marked as accepted. Now we will produce a heat map of the different features correlation to the a_accepted class, as can be seen in figure 3.6.

As we can see in the heat map for all of the features the correlation value is rather low. We will see it better in table 3.1, where we observe that just a few of the features have a correlation to class higher than 0.1.

Given that most of the features (either retrieved in the SQL query or generated manually) have an extremely low correlation to the a_accepted class we can presume that they will not be very useful to predict the selected answer.

Due to the low correlation of most features we will continue our exploratory data analysis only with those features where we observe an absolute correlation value with the a_accepted class higher than 5%. As we can see in table 3.2 these variables will be has_user_location, has_user_about, has_user_website_url, a_n_sent, has_user_profile_image_url, user_reputation, a_has_urls, a_n_words, qa_n_common, a_score and a_comment_count.

Figure 3.6: a_accepted feature correlation heat map



We will now explore the distribution for these features with respect to the a_accepted class. In order to do so we will produce individual histogram plots per feature, splitting the histogram samples per class. Figure 3.7 show these histograms. As we can see, due to the low correlation between each of these features and the a_accepted class there is not a large difference between the distribution of values for each feature across the class.

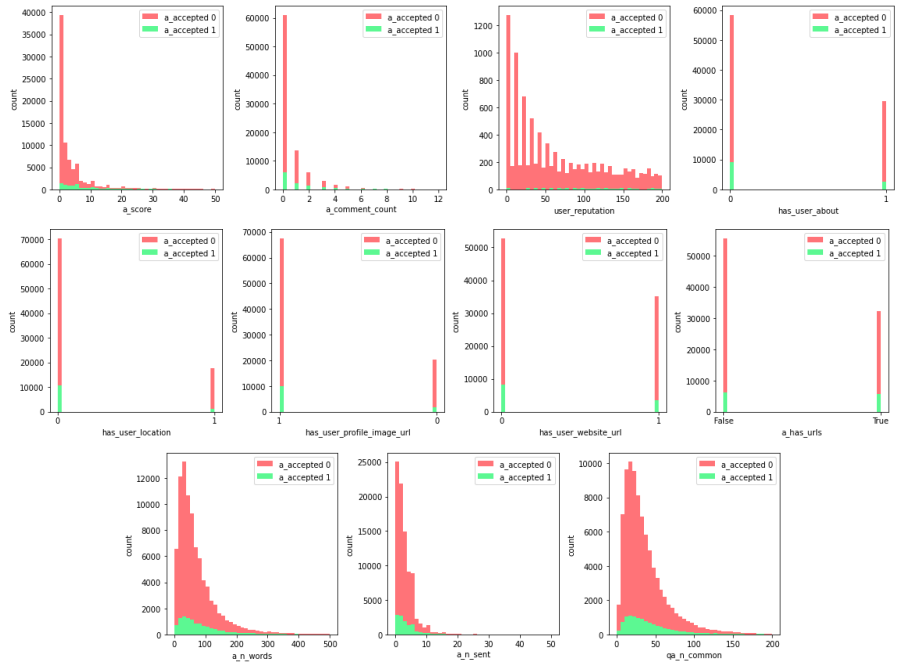
Feature	a_accepted correlation
has_user_location	-0.089347
has_user_about	-0.077484
has_user_website_url	-0.068921
q_avg_word_len	-0.017228
a_avg_n_word_sent	-0.000275
q_n_sent	0.002184
q_n_words	0.002589
user_down_votes	0.008166
q_max_n_word_sent	0.011173
q_avg_n_word_sent	0.014262
a_avg_word_len	0.018356
user_up_votes	0.023303
user_views	0.032269
a_n_sent	0.056832
has_user_profile_image_url	0.058864
user_reputation	0.066050
a_has_urls	0.072321
a_n_words	0.076058
qa_n_common	0.080131
a_score	0.128801
a_comment_count	0.168658

Table 3.1: a_accepted class to feature correlations

Feature	a_accepted correlation
has_user_location	-0.089347
has_user_about	-0.077484
has_user_website_url	-0.068921
a_n_sent	0.056832
has_user_profile_image_url	0.058864
user_reputation	0.066050
a_has_urls	0.072321
a_n_words	0.076058
qa_n_common	0.080131
a_score	0.128801
a_comment_count	0.168658

Table 3.2: Selected a_accepted class to feature correlations

Figure 3.7: Selected features histogram split per class



3.3 Metrics

As we have seen when the data set was described the original class distribution for the accepted answer feature is very unbalanced, hence using a classical binary classification metric such as accuracy would be misleading since a naive model that predicted all answers to not be accepted would yield a large accuracy values.

Due to this class unbalancing it was decided to use figures of merit from the information retrieval field, where multiple results are available for a single query and different metrics allow to easily compare between the different results which ones have been better classified or ranked. The information retrieval metrics we will use are the Mean Reciprocal Rank (MRR) [31] and the Normalized Discounted Cumulative Gain (NDCG) [32].

It was decided to discard the usage of Mean Average Precision (MAP) metric which is the most common in information retrieval problems as we only consider one correct answer per question: the answer accepted by the user who posted the question. In this case MAP will equal the MRR values, so there is no need to add MAP to the metric suite.

In order to define the Mean Reciprocal Rank we will first define the Reciprocal Rank. The Reciprocal Rank measure calculates the reciprocal of the rank at which the first relevant document resulting from an information retrieval query was retrieved. In our case this first relevant document will be the answer accepted by the user who asked the question. Given that we will be evaluating our experiments over a collection of questions and answers sets we will compute the average Reciprocal Rank over all the questions and corresponding answers sets, hence yielding the Mean Reciprocal Rank figure.

$$RR = \sum_{i=1}^{M} \frac{1}{rank_i}$$

Figure 3.8: Reciprocal Rank equation

$$MRR = \frac{1}{N}\sum_{i=1}^{N} RR_i$$

Figure 3.9: Mean Reciprocal Rank equation

The Normalized Discounted Cumulative Gain is another metric from the information retrieval field that measures the performance of an information retrieval system (most frequently a search engine) by ranking the results according to the their relevance in terms of the search query. In order to compute the Normalized Discounted Cumulative Gain we first must calculate the Discounted Cumulative Gain (DCG), this is the sum of the relevance scores of each of the query results divided by a discount factor such that lower ranking results have a smaller effect on the cumulative gain sum.

$$DCG_k = \sum_{i=1}^{k} \frac{relevance_i}{log_2(i+1)}$$

Figure 3.10: Discounted Cumulative Gain equation

Then we will compute the Normalized Discounted Cumulative Gain by dividing the Discounted Cumulative Gain by the Ideal Discounted Cumulative Gain (IDCG), this is the DCG value for a perfect query, where each of the results has the actual importance.

$$NDCG_k = \frac{DCG_k}{IDCG_k}$$

Figure 3.11: Normalized Discounted Cumulative Gain equation

In the specific case of our experiments we will consider that the ideal relevance scores will take the form of a unity value for the accepted answer and zero for the rest of the answers.

We will see some examples of how these metrics will allow us to measure how

well our models perform. Let us consider two arrays of values, one corresponding to the ground truth for the accepted answer and one for the model output. For the ground truth the array will resemble a one hot encoded vector, where only the index relative to the accepted answer is set to one will the other elements are set to zero. On the other hand the array corresponding to the model output will contain a graduation of values belonging to the probability that each of the answers to a question is the accepted answer, as predicted by the model under evaluation.

We will first consider the case of a perfect prediction, where the correct answer was perfectly identified. Hence we will have 2 arrays of data, the first one r_{gold} that contains the ground truth of the accepted answer and r_{eval} with the model output. In the case of a perfect input the value of each array will be

$$r_{gold} = \{0, 0, 1\} \;\; r_{eval} = \{0, 0, 1\}$$

On this example MRR and NDCG respectively will be 1.0 and 1.0 for this input.

We will now consider the case for the worst case prediction error. In this case r_{gold} again contains the ground truth of the accepted answer and r_{eval} with the model output, where the accepted answer was not the correct one:

$$r_{gold} = \{0, 0, 1\} \;\; r_{eval} = \{0, 1, 0\}$$

In this case MRR and NDCG respectively will be 0.333 and 0.565.

3.4 Baselines

We will now introduce a suite of sensible baseline models that allow us to set a minimum expectations for the methods we will propose later.

Given that the data we have collected is very heterogeneous in its nature due to the fact that it contains numerical and textual data belonging to both questions, answers and users we will need to create baseline models that reflect this

plurality of data features.

Hence makes sense to split our baseline models into models that make use of only numerical data, models that make use of textual data and models that use a mixture of numerical and textual data.

As numerical data we will use the features described earlier, both the ones directly extracted from the BigQuery databse queries and the numerical features derived from the question and answer texts.

In the case of textual data we will produce vector space representations of the answer data by using TF-IDF (term frequency–inverse document frequency) weighting of the bag of words, after the usual pre-processing steps of HTML tag and punctuation symbol removal and lower case transformation.

We will pair this hierarchy of models with different forest models, namely random forests, gradient boosted trees and AdaBoost. We will only make use of forest based trees due to their well known features of properly handling large collections of tabular data and their immunity to over-fitting [33].

One important thing to note is that, while the baselines we have described are all of them classification models with a discrete class output we will require to produce an output from them that may be used to rank the different answers by how likely they are to be the accepted answer. We will do so by getting the probability estimate output from each of the models, hence we will get a continuous, numerical figure that will easily allow us to rank each of the answers.

Random Forests

A random forest is an ensemble of decision trees which results in a better estimator than individual trees which suffer of over-fitting due to their low bias and high variance. By bagging a number of uncorrelated trees each of the high variance contributions of the individual trees are cancelled out. Other advantages of forests is their high parallelism and reduced complexity for training.

As their main disadvantage hand random forests lose the interpretability inherent to individual trees. Also, on problems where predictive features are very

linear random forests may not improve the accuracy of individual trees by much.

Gradient Boosted Trees

Gradient boosting allows the improvement of predictions done by an ensemble of weak learners, by incrementally refining the predictive capabilities of its learners. Usually these weak learners take the form of a decision tree and the resulting model is called a gradient boosted trees model. At each step of boosting the ensemble of learners is modified to correct for the previous step prediction errors, effectively trying to minimize the error function between the ground truth and the ensemble predictions. The direction to take in order to update the learners is derived from the gradient of the error function following its opposite direction, hence the name gradient boosting.
Again the main disadvantage with respect to a decision tree is the loss of explainability due to the ensemble nature of the model.

AdaBoost

AdaBoost (short form for Adaptive Boosting) is another ensemble method that tries to overcome individual learners weaknesses in a serial fashion. As learner a simple tree with just 2 leaves is used, and each of the learners is assigned a weight updated by the model in each boosting round. This is different form random forests and gradient boosted trees where all the learners have equal weights.

Chapter 4

Proposed Methods

4.1 Introduction

In this section we will introduce a suite of deep learning models that take embedded representations of the text contained in questions and answers as inputs, and additionally we will augment the model inputs with numerical features obtained from the data set. We will also introduce different ablation studies to observe the effect that different model parameters have on the model performance.

4.2 Hypothesis

We will enumerate a series of hypothesis that we will later test with the proposed experiments.

Firstly we propose that a Siamese neural network structure will allow us to infer informative relations between questions and answers text such that a meaningful performance level may be reached in terms of answer ranking.

Then we believe that augmenting the textual information from questions and answers with numerical features, either relative to the users or to the structure of the text like number of words per sentence, will allow our models to perform better.

Given that we will be using models that receive a sequence of tokens inputs we hypothesize that the maximum sequence length constraint may limit the model's performance.

Relative to these sequences of tokens we hypothesize that the corpus from which the embedding matrix is composed may affect the model performance, in the sense that when corpus that are closer to the online forum environments are used better model performance may be reached.

Finally we hypothesize that increasing the model complexity by adding more layers will improve the model performance for answer ranking.

4.3 Models

In this section we will describe the deep learning models we will use to attempt to improve the baseline results.

As with the baseline models we will also pair a plurality of models with different predictive features, like purely numerical features extracted from the data set, textual features from the question and answers text, and a mixture of both. In the case of numerical features we will simply use the same numerical data as in the baseline models. In the case of textual data we will produce dense embeddings using GloVe [34] to represent the questions and answers in a high dimensional space so they are suitable to be used as inputs to the deep learning models.

We will create token sequences of dimension 100 using GloVe embeddings obtained from Wikipedia [34]. The token sequences length will be limited to 100 tokens (both for question and answer texts), and we will be summarizing each question or answer text into two sentences by ranking the sentences by length and taking the top two. Among the many importance metrics we could use we will aim for low complexity and use a simple sentence length ranking, following the rationale that more meaningful sentences will be longer, due to the larger number of ideas they convey. In addition we will be balancing the training data

set to contain 500 pairs of questions and accepted answers, and 500 pairs of questions and not accepted answers. The test set will be composed by 1000 pairs of questions and answers, following the same class imbalance seen in the exploratory data analysis performed earlier.

The first layer in all of our models will be a Long Short-Term Memory (LSTM) layer. LSTM layers are an especial case of Recurrent Neural Networks (RNNs), this is, neural networks where some of their internal state values are fed back into the input. This allows modelling temporal relations on an input sequence such as a sentence or series of sentences. In particular LSTMs allow overcoming the common issue seen in RNNs of vanishing gradient, seen when training artificial networks with gradient based methods and back-propagation. In these cases each of the network weights are updated by a factor directly proportional to the gradient of the error function with respect to the current weight. After a few training epochs some of these gradients may become very small, effectively preventing the weights from being updated and consequently stopping the network from training further. The existence of recurrent gates in the LSTM cell allows the errors to flow back hence preventing the vanishing gradient problem. In our models we will make use of a bidirectional LSTM (Bi-LSTM), which uses a concatenation of two instances of the same replicated LSTM cell, with one of the instances seeing the sequence input from left to right and with the other instance seeing the sequence input from right to left. This allows the combined cell to process long term dependencies in either direction which is a crucial feature of natural language.

Since the textual data to be processed contains pairs of questions and answers we will replicate the input structure with a stack of Bi-LSTM cells twice, setting these 2 input structures branches in parallel so they receive question and answer text respectively. The output from the last layer in each of the parallel branches will be concatenated and fed into a dense layer that will act as a classification layer, producing a numerical output that will represent the probability that the input answer was selected as accepted by the question author.

Figure 4.1: Tensorflow representation for the deep learning model (including numerical features)

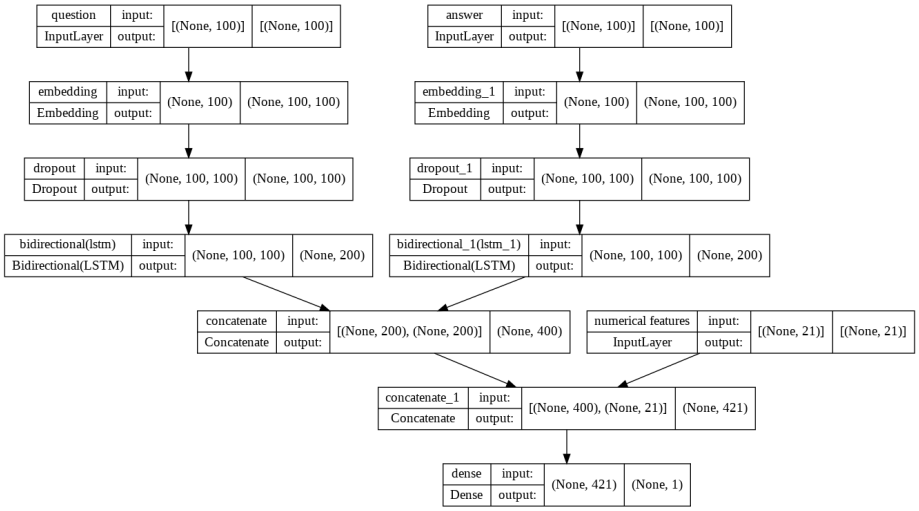


Image 4.1 shows the Tensorflow plot for the deep learning model we will use, where both question and answer token sequences go through embedding layers before being fed into the LSTM layers to obtain the embedded representations, to finally be concatenated together with the numerical features and fed into the classifier head.

For both of the proposed kind of models (models that only make use of textual features and models that make use of textual and numerical features) we will use the Adam optimizer [35], and we will sweep across different values for the learning rate parameter, with values comprised between 1e-1 and 1e-6. We will train our models for 5 epochs using a batch size of 512 samples.

4.4 Ablation studies

Once the basic experiments have been carried out we will select the best performing learning rate and we will use it to run a set of ablation studies where we will modify some of the parameters in the original model to check how they affect the model performance.

First we will experiment with the number of sentences found in the question

and answer embeddings. We will either not restrict the number of sentences in the textual input, or restrict to a few sentences as maximum. This experiment will allow us to see how dependent the answer ranking performance is on the amount of text input.

Similarly we will also sweep across a few different values for the maximum sequence length accepted by the embedding layers at the model input for both question and answer texts, in order to check how restricting the amount of information input to the model affects its performance.

In another experiment we will use both GloVe embeddings trained on Wikipedia text or Twitter text, in order to find out whether the different training source has an effect on the model performance.

In our experiments we will stack different amounts of LSTMs/Bi-LSTMs layers to increase the model complexity and expressiveness, in order to try to improve the quality of its predictions, as these stacked LSTM layers will allow the model to learn higher order representations that may be useful in our task. We will sweep the amount of stacked LSTM layers from 1 to 4. For these experiments we will set a dropout value in the LSTM layers and in the dropout layer after the embedding layers of 0.25, given that stacking many LSTM layers causes the trainable parameter count to raise by a huge amount and we can introduce overfitting in the training process.

Finally we will try different topologies of classification layers. Hence we will use both single layer and multi layer classification heads. In the case of multi layer classification heads we will experiment with different counts of hidden layers to see whether this has an impact on the model performance, given that we are facing a non linear problem where using a multi-layer classification architecture may be better. We will sweep among the possible settings of no hidden layer, a single hidden layer of size 200, a 2 hidden layer structure with sizes 200 and 100, and finally a 3 hidden layer structure with sizes 200, 100 and 50.

Chapter 5

Results

Having defined clearly what our data, models and evaluation methods will be we can now examine the results for each of the experiments described earlier. Initially we will obtain the baseline models' results and then go through each of the proposed experiments that make use of the deep learning models.

5.1 Baseline models

As described in the methods section we have produced different baseline models, making use of different ensembles of features. First we have simple models that only make use of the numerical features. We can see the results for these models in table 5.1. We can see that although all models show very similar performance, the xgboost model has better performance across all metrics.
Afterwards we have created baseline models that make use of a bag-of-words vector space model with TF-IDF weighting. We can see the results for these

Table 5.1: Baseline models results for numerical features

classifier	NDCG@1	NDCG@3	NDCG@5	MRR
rf	0.661982	0.861177	0.866924	0.816129
adaboost	0.688108	0.872320	0.877219	0.834360
xgboost	**0.691622**	**0.874560**	**0.879109**	**0.836995**

Table 5.2: Baseline models results for bag of words features

classifier	NDCG@1	NDCG@3	NDCG@5	MRR
rf	0.496306	0.778007	0.791852	0.712027
adaboost	0.493928	0.773191	0.789303	0.692690
xgboost	**0.517005**	**0.786364**	**0.799849**	**0.717544**

Table 5.3: Baseline models results for ensemble of numerical and bag of words features

classifier	NDCG@1	NDCG@3	NDCG@5	MRR
rf	0.600270	0.830575	0.839080	0.774275
adaboost	0.641216	0.849189	0.855798	0.805733
xgboost	**0.703243**	**0.879077**	**0.883278**	**0.842649**

models in table 5.2. Again we observe very similar performance across the different models with xgboost leading across all metrics.

Finally we have created baseline models that make use of the ensemble of numerical features with the bag-of-words vector space model used in the previous experiment. We can see the results for these models in table 5.3. We find slightly larger differences between models than on previous tables but again we find that xgboost is the best performing model.

5.2 Deep learning models

We can see the results for the base deep learning model without numerical features on table 5.4, and the results for the base deep learning model with numerical features on table 5.5. We see very similar results on both kinds of models, and very slight variation across the learning rate values. As expected adding the numerical features allow the model to perform better than the one using simply textual features.

In addition to obtaining the NDCG and MRR values for different learning rates we will also plot how the MRR varies with the number of answers posted to each question, as seen on figure 5.1. On this plot we have superposed the curve for the obtained MRR values (blue curve) with the theoretical MRR for a random

Table 5.4: Deep learning base model results (no numerical features)

learning_rate	NDCG@1	NDCG@3	NDCG@5	MRR
0.100000	0.465995	**0.775149**	0.785667	0.711209
0.010000	0.455919	0.766004	0.779665	0.703317
0.001000	0.408060	0.752389	0.761931	0.679219
0.000100	0.448363	0.764534	0.777000	0.700882
0.000010	**0.483627**	0.772991	**0.786432**	**0.713980**
0.000001	0.468514	0.771911	0.782428	0.709470

Table 5.5: Deep learning base model results (with numerical features)

learning_rate	NDCG@1	NDCG@3	NDCG@5	MRR
0.100000	0.460957	0.779047	0.784251	0.711485
0.010000	**0.496222**	**0.784116**	**0.796913**	**0.726448**
0.001000	0.418136	0.750740	0.764622	0.682997
0.000100	0.460957	0.780637	0.787035	0.712720
0.000010	0.410579	0.743183	0.758903	0.676742
0.000001	0.435768	0.756588	0.770249	0.690722

ranking (orange curve). We will plot a histogram of MRR values so we can see how often the accepted answer is ranked first, as seen in figure 5.2.

After selecting a learning rate of 0.01 based on the previous results (it maximized the performance of the model with textual and numerical features) we will proceed with the rest of experiments.

Table 5.6 shows the results for the experiments that varied the maximum sentence count. We see the best performance when there is no constraint for the maximum sentence count, and in general performance degrades as less sentences are considered.

Figure 5.1: MRR versus answer count

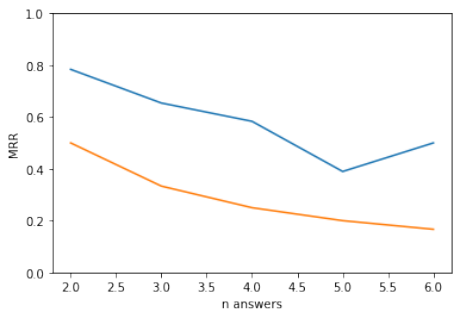


Figure 5.2: MRR histogram

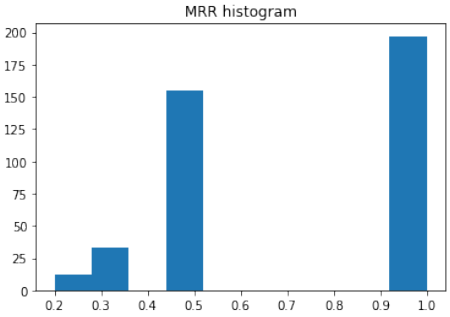


Table 5.6: Deep learning maximum sentence count experiments results

max sentences	NDCG@1	NDCG@3	NDCG@5	MRR
-1	**0.526448**	**0.802948**	**0.811627**	**0.746012**
1	0.450882	0.767712	0.780399	0.704072
2	0.473552	0.777069	0.788891	0.716649
3	0.496222	0.788764	0.794942	0.725063
4	0.430730	0.764053	0.772622	0.694563

Results for the experiments where the maximum sequence length input to the model is swept while not restricting these sequences to have a maximum sentence count may be found at table 5.7. Even though results are very similar between runs we see that setting a 200 token maximum sequence length performs better across most metrics.

The next experiment compared the performance of the GloVe embeddings learned from Wikipedia texts with alternate embeddings learned from tweets. Table 5.8 shows the results of these experiments. Embeddings learned from Wikipedia content performed slightly better than the Twitter ones.

Our next set of experiments increased the count of stacked LSTM layers after

Table 5.7: Deep learning maximum sequence length experiments results

max sequence length	NDCG@1	NDCG@3	NDCG@5	MRR
100	**0.486146**	0.777429	0.790337	0.717758
150	0.450882	0.775718	0.784066	0.708690
200	0.476071	**0.783366**	**0.790849**	**0.719228**
250	0.460957	0.773680	0.784307	0.710495
300	0.465995	0.774160	0.784677	0.709950

Table 5.8: Deep learning embedding experiments results

embedding	NDCG@1	NDCG@3	NDCG@5	MRR
twitter	0.413098	0.755567	0.764999	0.683291
wiki	**0.450882**	**0.769961**	**0.781563**	**0.705542**

Table 5.9: Deep learning LSTM depth experiments results

LSTM depth	NDCG@1	NDCG@3	NDCG@5	MRR
1	**0.460957**	0.769572	**0.783233**	**0.709089**
2	0.448363	0.764204	0.777755	0.700672
3	0.455919	**0.773680**	0.781806	0.707074
4	0.443325	0.765193	0.776906	0.699370

the input embedding. Table 5.9 shows the results of these experiments, where we swept the amount of stacked LSTM layers from 1 to 4. We see that having a single LSTM layer performed better across most metrics.

Finally we experimented how the model behaves when using different hidden layer counts in the classifier head. Table 5.10 shows the results, where we see that having 2 hidden layers performed better, even though differences seen across runs were small.

Table 5.10: Deep learning classifier architecture experiments results

hidden layer count	NDCG@1	NDCG@3	NDCG@5	MRR
0	0.405542	0.745492	0.759374	0.677120
1	0.483627	0.784955	0.794498	0.724139
2	**0.488665**	0.**789663**	**0.795841**	**0.727042**
3	0.450882	0.774069	0.781442	0.706447

Chapter 6

Discussion

After completing all the proposed experiments we will interpret the results we obtained and draw conclusions from this work.

We will start off by reviewing the results from the different ablation studies we performed on the token sequence generation and the model architecture. As can be seen in the results tables on most experiments all the NDCG and MRR metrics behave in the same direction with minor discrepancies. In case of such discrepancies we will favor MRR when choosing a better result as it considers all the possible answers to a query rather than setting a cut point from which no further answers are considered like NDCG does.

We could clearly see an improvement in all metrics when the numerical features were introduced. It is to note that this improvement was also seen in the baseline models, hence reinforcing our hypothesis that using specific features either originated in the data set source as metadata or extracted manually like we did may improve the results of our answer ranking system.

In terms of the token sequence generation we found better results when not trying to summarize the question and answers content into a reduced amount of sentences. We can easily infer that the model behaves better when seeing more information.

The next experiments tried to increase the sequence length and we find that the

results peak with a maximum sequence length of 200 tokens. This is probably due to the fact that this sequence length is enough to contain most of the question or answer content and not requiring padding which may not be meaningful for the model.

After the embedding layer inside the model we find the LSTM layers where representations of the input sequence are generated. Surprisingly we have found that a single LSTM layer outperforms the rest of options. This may be due to the fact that a single LSTM layer is able to produce representations that are meaningful enough for our task, and higher LSTM layer orders only introduce non meaningful higher order representations that do not inform the prediction model any further and may cause the model to over fit.

Finally the experiments where the classifier head structure was changing showed us that using 2 hidden layers improves the model results. This is probably due to the highly non-linear nature of the answer ranking problem, which requires a more complex classifier structure.

Once having reviewed the results of the ablation studies performed before we will compare the performance of the baseline model against the deep learning models we created. If we initially consider the models that do not make use of the numerical features we find that the deep learning model has a similar performance to the best baseline model for bag of words (MRR 0.717544 for xgboost versus MRR 0.713980 for the deep learning model). Then if we consider models that make use of both textual and numerical feature we see that none of the deep learning models variations can even get close to the best baseline model performance (MRR 0.842649 for xgboost versus MRR 0.746012 on the deep learning model maximum sentence count experiment).

We can speculate on a couple of reasons for this poor performance in the deep learning model. First, as the baseline models use a bag of words they are able to observe the contents for the whole of the question or answer text while on the deep learning model the maximum sequence constraint may cause some loss of information. In this case a possible solution would to split the input text into

enough input sequences for the model so all of the text may be processed by the model, to later ensemble the prediction results for these split inputs into a single value.

The second possible reason for the lack of performance in the deep learning model may be attributed to the high number of words out of vocabulary when the input embedding stage is performed. As en example this is the report obtained during an embedding pass:

"*Converted 16514 words (49474 misses)*"

As we can see the embedding missed many more words than were properly converted. We can look more closely at these embedding misses and observe a sample of 20 words missed during the embedding phase:

"*acitvitypackage, belowbody, bindservice, browserpath, callactorsink, createnow, descsortbyname, downloadingprivate, emailbtn, etcby, genfromtxt, hasattribute, hashkey, installthen, inversejoincolumns, justahelperfunction, mockitoannotations, nashornfunction, networksthat, newrandom*"

We can see that all these words are some sort of compound words, formed by two or more words. These are probably variable names from code snippets such as the ones posted by users asking or answering questions related to computer programming problems in Stack Overflow. In this case no possible choice of embedding matrix initialization would help, and we should better be either detecting these compound words and perhaps coding a special token for these situations or detecting whole chunks of code in the question and answer texts. Of course this programming code detection is a whole new problem on its own as computer code may be written in many different languages, with different syntactic rules.

Chapter 7

Conclusions and future work

In this work we have looked at the specific problem of ranking multiple answers to questions in community Question Answering sites. This is a very meaningful problem as these sites have experienced big success and hence each of the posted questions is met with multiple answers, of disparate quality and accuracy. It is crucial to the site users and maintainers to be able to sort through all the possible answers to extract meaningful information from them.

We have first compiled a relevant data set encompassing multiple questions and answers from the Stack Overflow site through the Google BigQuery interface. After exploring the data set we have created multiple predictive features to assist with the task of ranking the best answers to each question.

The next step in our work has consisted of creating meaningful baseline models that make use of both the textual content in each question and answer, and of the numerical predictive features we created earlier.

We have then defined multiple metrics to assess the goodness of the models used in the experiments. These metrics such as the NDCG and MRR were created for the information retrieval field where they have been used to great extent

with huge success.

Afterwards we have run multiple experiments using a deep learning model structured around embedding and LSTM layers. We have checked the impact of changing the settings in different parts of both the token sequence generation and the model structure.

The outcome of these experiments has been that while leaving out the numerical features we could match the baseline models performance, the deep learning models performance severely lacked the baseline performance when both textual and numerical features were considered.

The existence of many out of vocabulary words on the questions and answers text due to the presence of programming code is the most probable cause for this lack of performance on our deep learning models. A possible solution would be to detect the presence of programming code and tokenize it accordingly.

As future lines of work on this area we could suggest using some of the newest natural language models such as the BERT family of Transformer-based models to try to improve the model prediction capabilities or trying to experiment with the creation of general models that may work across multiple Stack Exchange sub-sites, as on this work we focused on using data from the main Stack Overflow site, focused on computer programming questions. Also as we mentioned the creation of methods to properly detect and delimit the existence of programming code would be a very helpful line of work for the answer ranking problem.

Bibliography

[1] Ivan Srba and Maria Bielikova. "A Comprehensive Survey and Classification of Approaches for Community Question Answering". In: *ACM Trans. Web* 10.3 (2016). ISSN: 1559-1131. DOI: 10.1145/2934687. URL: https://doi.org/10.1145/2934687.

[2] *Quora — quora.com.* https://www.quora.com/. [Accessed 05-Jun-2022]. 2022.

[3] *Stack Overflow - Where Developers Learn, Share, Build Careers — stackoverflow.com.* https://stackoverflow.com/. [Accessed 05-Jun-2022]. 2022.

[4] Qiaoling Liu et al. "Predicting Web Searcher Satisfaction with Existing Community-Based Answers". In: *Proceedings of the 34th International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR '11. Beijing, China: Association for Computing Machinery, 2011, pp. 415–424. ISBN: 9781450307574. DOI: 10.1145/2009916.2009974. URL: https://doi.org/10.1145/2009916.2009974.

[5] Pradeep Kumar Roy et al. "Analysis of community question-answering issues via machine learning and deep learning: State-of-the-art review". In: *CAAI Transactions on Intelligence Technology* n/a.n/a (). DOI: https://doi.org/10.1049/cit2.12081. eprint: https://ietresearch.onlinelibrary.wiley.com/doi/pdf/10.1049/cit2.12081. URL:

https://ietresearch.onlinelibrary.wiley.com/doi/abs/10.1049/cit2.12081.

[6] Bowen Xu et al. "Prediction of Relatedness in Stack Overflow: Deep Learning vs. SVM: A Reproducibility Study". In: *Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*. ESEM '18. Oulu, Finland: Association for Computing Machinery, 2018. ISBN: 9781450358231. DOI: 10.1145/3239235.3240503. URL: https://doi.org/10.1145/3239235.3240503.

[7] Muhammad Asaduzzaman et al. "Answering questions about unanswered questions of Stack Overflow". In: *2013 10th Working Conference on Mining Software Repositories (MSR)*. 2013, pp. 97–100. DOI: 10.1109/MSR.2013.6624015.

[8] F. Maxwell Harper, Daniel Moy, and Joseph A. Konstan. "Facts or Friends? Distinguishing Informational and Conversational Questions in Social QA Sites". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '09. Boston, MA, USA: Association for Computing Machinery, 2009, pp. 759–768. ISBN: 9781605582467. DOI: 10.1145/1518701.1518819. URL: https://doi.org/10.1145/1518701.1518819.

[9] Sai Praneeth Suggu et al. "Hand in Glove: Deep Feature Fusion Network Architectures for Answer Quality Prediction in Community Question Answering". In: *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*. Osaka, Japan: The COLING 2016 Organizing Committee, Dec. 2016, pp. 1429–1440. URL: https://aclanthology.org/C16-1135.

[10] George Gkotsis et al. "It's All in the Content: State of the Art Best Answer Prediction Based on Discretisation of Shallow Linguistic Features". In: *Proceedings of the 2014 ACM Conference on Web Science*. WebSci '14. Bloomington, Indiana, USA: Association for Computing Machinery, 2014,

pp. 202–210. ISBN: 9781450326223. DOI: 10.1145/2615569.2615681. URL: https://doi.org/10.1145/2615569.2615681.

[11] Felix Hieber and Stefan Riezler. "Improved Answer Ranking in Social Question-Answering Portals". In: *Proceedings of the 3rd International Workshop on Search and Mining User-Generated Contents*. SMUC '11. Glasgow, Scotland, UK: Association for Computing Machinery, 2011, pp. 19–26. ISBN: 9781450309493. DOI: 10.1145/2065023.2065030. URL: https://doi.org/10.1145/2065023.2065030.

[12] Alexandru Lucian Ginsca and Adrian Popescu. "User Profiling for Answer Quality Assessment in Qamp;A Communities". In: *Proceedings of the 2013 Workshop on Data-Driven User Behavioral Modelling and Mining from Social Media*. DUBMOD '13. San Francisco, California, USA: Association for Computing Machinery, 2013, pp. 25–28. ISBN: 9781450324175. DOI: 10.1145/2513577.2513579. URL: https://doi.org/10.1145/2513577.2513579.

[13] Daniel Hasan Dalip et al. "Exploiting User Feedback to Learn to Rank Answers in Qamp;a Forums: A Case Study with Stack Overflow". In: *Proceedings of the 36th International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR '13. Dublin, Ireland: Association for Computing Machinery, 2013, pp. 543–552. ISBN: 9781450320344. DOI: 10.1145/2484028.2484072. URL: https://doi.org/10.1145/2484028.2484072.

[14] Mohamed Bouguessa, Benoıt Dumoulin, and Shengrui Wang. "Identifying Authoritative Actors in Question-Answering Forums: The Case of Yahoo! Answers". In: *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '08. Las Vegas, Nevada, USA: Association for Computing Machinery, 2008, pp. 866–874. ISBN: 9781605581934. DOI: 10.1145/1401890.1401994. URL: https://doi.org/10.1145/1401890.1401994.

[15] Leandro Amancio, Carina F. Dorneles, and Daniel H. Dalip. "Recency and quality-based ranking question in CQAs: A Stack Overflow case study". In: *Information Processing Management* 58.4 (2021), p. 102552. ISSN: 0306-4573. DOI: https://doi.org/10.1016/j.ipm.2021.102552. URL: https://www.sciencedirect.com/science/article/pii/S030645732100056X.

[16] Xiaoqiang Zhou et al. "Recurrent convolutional neural network for answer selection in community question answering". In: *Neurocomputing* 274 (2018). Query Understanding, pp. 8–18. ISSN: 0925-2312. DOI: https://doi.org/10.1016/j.neucom.2016.07.082. URL: https://www.sciencedirect.com/science/article/pii/S0925231217306744.

[17] Junyoung Chung et al. *Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling*. 2014. DOI: 10.48550/ARXIV.1412.3555. URL: https://arxiv.org/abs/1412.3555.

[18] Qin Chen et al. "Enhancing Recurrent Neural Networks with Positional Attention for Question Answering". In: *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR '17. Shinjuku, Tokyo, Japan: Association for Computing Machinery, 2017, pp. 993–996. ISBN: 9781450350228. DOI: 10.1145/3077136.3080699. URL: https://doi.org/10.1145/3077136.3080699.

[19] Jacob Devlin et al. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. 2018. DOI: 10.48550/ARXIV.1810.04805. URL: https://arxiv.org/abs/1810.04805.

[20] Md Tahmid Rahman Laskar, Jimmy Xiangji Huang, and Enamul Hoque. "Contextualized Embeddings based Transformer Encoder for Sentence Similarity Modeling in Answer Selection Task". English. In: *Proceedings of the 12th Language Resources and Evaluation Conference*. Marseille, France: European Language Resources Association, May 2020, pp. 5505–

5514. ISBN: 979-10-95546-34-4. URL: https://aclanthology.org/2020.lrec-1.676.

[21] Jiangnan Du et al. "Towards a Two-Stage Method for Answer Selection and Summarization in Buddhism Community Question Answering". In: *Artificial Intelligence*. Ed. by Lu Fang et al. Cham: Springer International Publishing, 2021, pp. 251–260. ISBN: 978-3-030-93049-3.

[22] Macedo Maia, Siegfried Handschuh, and Markus Endres. "A Tag-Based Transformer Community Question Answering Learning-to-Rank Model in the Home Improvement Domain". In: *Database and Expert Systems Applications*. Ed. by Christine Strauss et al. Cham: Springer International Publishing, 2021, pp. 127–138. ISBN: 978-3-030-86475-0.

[23] Haoriqin Wang et al. "A Dynamic Attention and Multi-Strategy-Matching Neural Network Based on Bert for Chinese Rice-Related Answer Selection". In: *Agriculture* 12.2 (2022). ISSN: 2077-0472. DOI: 10.3390/agriculture12020176. URL: https://www.mdpi.com/2077-0472/12/2/176.

[24] Chirag Shah and Jefferey Pomerantz. "Evaluating and Predicting Answer Quality in Community QA". In: *Proceedings of the 33rd International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR '10. Geneva, Switzerland: Association for Computing Machinery, 2010, pp. 411–418. ISBN: 9781450301534. DOI: 10.1145/1835449.1835518. URL: https://doi.org/10.1145/1835449.1835518.

[25] Preslav Nakov et al. "SemEval-2015 Task 3: Answer Selection in Community Question Answering". In: (2019). DOI: 10.48550/ARXIV.1911.11403. URL: https://arxiv.org/abs/1911.11403.

[26] Preslav Nakov et al. "SemEval-2016 Task 3: Community Question Answering". In: *Proceedings of the 10th International Workshop on Semantic Evaluation (SemEval-2016)*. San Diego, California: Association for Com-

putational Linguistics, June 2016, pp. 525–545. DOI: 10.18653/v1/S16-1083. URL: https://aclanthology.org/S16-1083.

[27] Preslav Nakov et al. "SemEval-2017 Task 3: Community Question Answering". In: *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*. Vancouver, Canada: Association for Computational Linguistics, Aug. 2017, pp. 27–48. DOI: 10.18653/v1/S17-2003. URL: https://aclanthology.org/S17-2003.

[28] Mansi Gupta et al. *AmazonQA: A Review-Based Question Answering Task*. 2019. DOI: 10.48550/ARXIV.1908.04364. URL: https://arxiv.org/abs/1908.04364.

[29] Felip Hoffa. URL: https://cloud.google.com/blog/topics/public-datasets/google-bigquery-public-datasets-now-include-stack-overflow-q-a.

[30] Wikipedia. *Stack Overflow — Wikipedia, The Free Encyclopedia*. http://en.wikipedia.org/w/index.php?title=Stack%20Overflow&oldid=1085906064. [Online; accessed 10-May-2022]. 2022.

[31] Ellen M. Voorhees and Dawn M. Tice. "The TREC-8 Question Answering Track". In: *Proceedings of the Second International Conference on Language Resources and Evaluation (LREC'00)*. Athens, Greece: European Language Resources Association (ELRA), May 2000. URL: http://www.lrec-conf.org/proceedings/lrec2000/pdf/26.pdf.

[32] Kalervo Järvelin and Jaana Kekäläinen. "Cumulated Gain-Based Evaluation of IR Techniques". In: *ACM Trans. Inf. Syst.* 20.4 (Oct. 2002), pp. 422–446. ISSN: 1046-8188. DOI: 10.1145/582415.582418. URL: https://doi.org/10.1145/582415.582418.

[33] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. Springer Series in Statistics. New York, NY, USA: Springer New York Inc., 2001.

[34] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. "GloVe: Global Vectors for Word Representation". In: *Empirical Methods in Natural Language Processing (EMNLP)*. 2014, pp. 1532–1543. URL: http://www.aclweb.org/anthology/D14-1162.

[35] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2014. DOI: 10.48550/ARXIV.1412.6980. URL: https://arxiv.org/abs/1412.6980.

Appendices

.1 Appendix A: SQL queries

SQL query to get the data set is shown below. `check` will be substituted by = or != to get accepted or not accepted answers, respectively.

```
  SELECT
question.id as q_id,
question.Title AS q_title,
question.Body AS q_body,
question.answer_count as q_answer_count,
question.accepted_answer_id as q_accepted_a,
answer.Id AS a_id,
answer.Body AS a_body,
answer.Score as a_score,
answer.comment_count AS a_comment_count,
user.id as user_id,
user.about_me as user_about,
user.age as user_age,
user.creation_date as user_creation_date,
user.last_access_date as user_last_access_date,
user.location as user_location,
user.reputation as user_reputation,
```

```
user.up_votes as user_up_votes,
user.down_votes as user_down_votes,
user.views as user_views,
user.profile_image_url as user_profile_image_url,
user.website_url as user_website_url,
CASE WHEN question.accepted_answer_id = answer.Id
  THEN '1'
  ELSE '0'
END
 AS a_accepted
FROM `bigquery-public-data.stackoverflow.posts_answers` AS answer
JOIN `bigquery-public-data.stackoverflow.posts_questions` question ON question.Id = answer.parent_id
JOIN `bigquery-public-data.stackoverflow.users` user on user.id = answer.owner_user_id
WHERE answer.post_type_id = 2 AND question.answer_count > 1
 AND question.accepted_answer_id IS NOT NULL
 AND question.accepted_answer_id IN (SELECT Id FROM `bigquery-public-data.stackoverflow.posts_answers`)
 AND question.accepted_answer_id {check} answer.Id
```

```
   AND EXTRACT(YEAR FROM question.creation_date) = 2016
ORDER BY question.ID ASC, answer.Id ASC
LIMIT 100000
```