

Learning to Rank Answers to Why-Questions

Suzan Verberne^{*}
s.verberne@let.ru.nl

Stephan Raaijmakers[†]
stephan.raaijmakers@tno.nl

Daphne Theijssen^{*}
d.theijssen@let.ru.nl

Lou Boves^{*}
l.boves@let.ru.nl

ABSTRACT

The goal of the current research project is to develop a question answering system for answering *why*-questions (*why*-QA). Our system is a pipeline consisting of an off-the-shelf retrieval module followed by an answer re-ranking module. In this paper, we aim at improving the ranking performance of our system by finding the optimal approach to learning to rank. More specifically, we try to find the optimal ranking function to be applied to the set of candidate answers in the re-ranking module. We experiment with a number of machine learning algorithms (i.e. genetic algorithms, logistic regression and SVM), with different cost functions.

We find that a learning to rank approach using either a regression technique or a genetic algorithm that optimizes for MRR leads to a significant improvement over the TF-IDF baseline. We reach an MRR of 0.341 with a success@10 score of 58.82%. We also see that, as opposed to logistic regression and genetic algorithms, SVM is not suitable for the current data representation. After extensive experiments with SVMs, we still reach scores that are below baseline.

In future work, we will investigate in more detail the limitations of our re-ranking approach: which set of questions cannot be answered in the current system set-up and why?

General Terms

Question Answering, *Why*-Questions, Learning to Rank

1. INTRODUCTION

The goal of the current research project is to develop a question answering system for answering *why*-questions (*why*-QA). In a QA system, *why*-questions need a different approach from factoid questions since their answers are explanations that cannot be stated in a single phrase. Answers to

why-questions tend to be at least one sentence and at most one paragraph in length [17]. Therefore, passage retrieval (as opposed to named entity retrieval, which is generally used for factoid QA) appears to be a suitable approach to *why*-QA.

In previous work, we have developed a passage retrieval system for *why*-QA [18]. This pipeline system consists of an off-the-shelf retrieval engine (Lemur¹), extended with a re-ranking module that is specifically tuned for ranking the answers to *why*-questions. In the re-ranking module, a set of features is extracted from the question and each of the candidate answers retrieved by Lemur. The values of these features are combined in a ranking function that is used for re-ordering the set of candidate answers.

The task of finding the optimal ranking function for a specific information retrieval task is referred to as ‘learning to rank’ in the literature [12]. Until now, we have mainly focused on improving the ranking performance of our system by adapting and expanding the feature set used for re-ranking [18]. This has led to a set of 37, mostly linguistic, features.

In the current paper, we aim at improving the ranking performance of our system by finding the optimal approach to learning to rank. More specifically, we try to find the optimal ranking function to be applied to the set of candidate answers in the re-ranking module. We vary our experimental settings in two dimensions: the machine learning techniques (genetic algorithms, logistic regression and support vector machines), and the cost function. In all experimental settings, we keep the set of 37 features that we found to be relevant in previous work.

This paper is organized as follows: in Section 2, we discuss related work on approaches to learning to rank. In Section 3 we describe the resources that we use for our experiments and we specify the characteristics of the machine learning data. Section 4 defines the machine learning problem that we consider in our learning to rank experiments. In Section 5 and 6 we describe the experiments that we conducted and the results we obtained. The results are discussed in Section 7. Section 8 contains our conclusions.

2. RELATED WORK

¹See <http://www.lemurproject.org/>

^{*}Dept. of Linguistics, Radboud University Nijmegen

[†]TNO Information and Communication Technology, Delft

As explained in Section 1, we vary our experimental settings in two dimensions: the machine learning techniques that we use and the cost function that we implement. Therefore, we discuss related work in two subsections.

2.1 Machine learning techniques for learning to rank

Most approaches to learning to rank consider the problem as a case of supervised learning. The training set is a matrix of feature vectors for a set of instances (the items to be ranked). Each item is assigned a label representing its relevance ground truth. A supervised learning problem can be solved by regression and classification techniques. In [12], many approaches to learning to rank are discussed.

In previous work, we used a genetic algorithm for finding the optimal ranking function. Genetic algorithms are devised for sampling (finding an optimum in) a very large data space. The definition of ‘optimum’ here is defined by the so-called fitness function in the genetic algorithm. Genetic algorithms have been applied to learning to rank problems and other retrieval optimization problems by several researchers in the field [16, 6, 15]. The approach presented in [16] resembles our approach: it defines the learning problem as the search for the optimal weight vector for a given feature vector.

2.2 Cost functions for learning to rank

One of the advantages of genetic algorithms is that the cost function (fitness function) is user-defined². In [6], a number of fitness functions that are derived from evaluation measures (such as average precision) are compared for their effectiveness.

An important aspect of the cost function in learning to rank problems is the definition of the ordering principle: items can be placed on an ordinal scale based on a score that is assigned to them³ or they can be ordered relative to other items in the list by defining for each possible pair of items which of the two is more relevant. The latter learning principle is called ‘pairwise preference learning’, and was introduced by Joachims [9], who created the learning algorithm Ranking SVM based on this principle. In pairwise preference learning, the measure that is optimized is Kendall Tau:

$$\tau = (P - Q)/(P + Q) \quad (1)$$

in which P is the number of concordant item pairs (the two items are ordered correctly) and Q is the number of discordant item pairs (the two items are ordered incorrectly). Pairwise preference learning has been studied in more detail by Furnkranz and Hullermeier [7] and applied to several ranking problems such as combining rankings from multiple retrieval systems by Carterette and Petkova [3].

3. DATA AND SYSTEM SET-UP

3.1 Resources

For our experiments, we used the Wikipedia INEX corpus [5]. This corpus consists of all 659,388 articles extracted

²This is not unique for genetic algorithms but it is one of the most typical characteristics of genetic algorithms.

³This score generally is the probability, assigned by a classification or regression model, that the item has either of the two labels [2].

from the online Wikipedia in the summer of 2006, converted to XML format.

Before indexing the corpus, we segmented all Wikipedia documents in passages. We decided on using a semi-fixed passage size of 500 to 600 characters (excluding all XML markup) with an overflow to 800 for the purpose of completing sentences⁴. We create passage overlap by starting each new passage at a paragraph or sentence boundary halfway the previous passage⁵. For Wikipedia articles that contain less than 500 characters in total, we included the complete text as one passage.

Our segmentation process gives an index of 6,365,890 passages with an average length of 429 characters (standard deviation 194) per passage⁶. We separately saved the document title and section heading as metadata for each passage.

For development and testing purposes, we exploited the WebClopedia question set by Hovy et al. [8]. This set contains questions that were asked to the online QA system *answers.com*. Of these questions, 805 (5% of the total set) are *why*-questions. For 700 randomly selected *why*-questions from this set, we manually searched for an answer in the Wikipedia XML corpus, keeping the remaining questions for future test purposes. 187 questions have at least one answer in the corpus. Extraction of one relevant answer for each of these questions resulted in a set of 187 *why*-questions and their reference answer. Let us give three examples to illustrate the type of data we are working with:

1. "Why do most cereals crackle when you add milk?" — "They are made of a sugary rice mixture which is shaped into the form of rice kernels and toasted. These kernels bubble and rise in a manner which forms very thin walls. When the cereal is exposed to milk or juices, these walls tend to collapse suddenly, creating the famous ‘Snap, crackle and pop’ sounds."
2. "Why didn't Socrates leave Athens after he was convicted?" — "Socrates considered it hypocrisy to escape the prison: he had knowingly agreed to live under the city's laws, and this meant the possibility of being judged guilty of crimes by a large jury."
3. "Why was cobalt named cobalt?" — "The word cobalt comes from the German kobalt or kobold, meaning evil spirit, the metal being so called by miners, because it was poisonous and troublesome (it polluted and degraded the other mined elements, like nickel)."

⁴We assume that answer passages ending in an unfinished sentence are undesirable. However, if the hard maximum of 800 characters is reached, the passage is cut off between two words to prevent non-sentence contexts like tables to result in extremely long passages.

⁵Other work on passage retrieval for QA [10] shows that better retrieval results are achieved with fixed-sized, partly overlapping passages than with structure-based, disjoint passages (e.g. <p>-items, which are very variable in length).

⁶The average length is smaller than the predefined minimum length of 500 characters because after clean-up a significant number of articles is shorter than 500 characters.

In order to be able to do fast evaluation without elaborate manual assessments, we manually created one answer pattern for each of the questions in our set. The answer pattern is a regular expression that defines which of the retrieved passages are considered a relevant answer to the input question. In their original versions, the answer patterns were directly based on the corresponding reference answer, but in the course of the development and evaluation process, we extended the patterns in order to cover as much as possible of the variants of the reference answer that occur in the Wikipedia corpus. By following this iterative process, we prevented to miss relevant answers.

For example, for question 2 above, we developed the following answer pattern based on two variants of the correct answer that occur in the corpus: */(Socrates.* opportunity.* escape.* Athens.* considered.* hypocrisy | leave.* run.* away.* community.* reputation)/*⁷ If a candidate answer matches the answer pattern then this answer is marked relevant, otherwise it is marked irrelevant. This evaluation method means that we defined relevance as a binary variable: an answer passage is either relevant or not.

3.2 System set-up

As we briefly mentioned in Section 1, our system for *why*-QA consists of three pipelined modules: (1) a question processing module that transforms the input question to a query by removing stop words and punctuation; (2) an off-the-shelf retrieval module that retrieves and ranks passages of text that share content with the input query; and (3) a re-ranking module that re-ranks the retrieved passages using features extracted from the question and each of the candidate answers. We aim to find the optimal ranking function to be applied in the re-ranking module. Thus, for our learning to rank experiments, we used the output of the retrieval module (2).

In the retrieval module, we used Lemur to retrieve 150⁸ answers per question and rank them using TF-IDF as it has been built in in Lemur⁹. This gave us a set of 187 questions with 150 candidate answers per question, with for each pair of a question and a candidate answer a TF-IDF score. For re-ranking, feature values needed to be extracted from each of these 28,050 (187 * 150) question-answer pairs.

3.3 Feature extraction

From earlier work [18], we compiled a set of 37 features that are summarized in Table 1. We parsed the questions with the Pelican parser¹⁰ and the candidate answers with the Charniak parser. Then we used a Perl script for extracting all feature values from the question, the answer candidate and both their parse trees.

Each feature represents the similarity between two item sets: a set of question items (for example: all question NPs, or

⁷Note that the vertical bar separates the two alternatives.

⁸We experimented with a higher number of answer candidates but coverage was hardly improved when increasing this number to 500.

⁹In previous work [10], we experimented with other ranking models and TF-IDF came out as the best.

¹⁰See <http://lands.let.ru.nl/projects/pelican/>

the question subject) and a set of answer items (for example: all answer words, or all subjects in the answer). The value that is assigned to a feature is a function of the similarity between these two sets. For determining this similarity, we use a statistic derived from the Jaccard index that is adapted for duplicate terms in either of the two sets. For a set of question word tokens Q , a set of question word types Q' , a set of answer word tokens A and a set of answer word types A' , the similarity S between Q and A is defined as:

$$S(Q, A) = \frac{|Q \cap A'| + |A \cap Q'|}{|Q \cup A|} \quad (2)$$

3.3.1 Description of the features

Syntactic features and Wordnet expansion features. Details on the syntactic features and the WordNet expansion features can be found in Verberne et al. 2008 [18]. The features that deserve some extra attention here, are the features related to question focus (e.g. overlap between the question focus and the title of the answer document). We introduced the term question focus in analogy to linguistically motivated approaches to factoid QA for the topic of the question (“What is the question about?”). We defined three rules for determining the focus of a *why*-question: If the subject is semantically poor, the question focus is the (verbal or nominal) predicate: “Why do people sneeze?”. In case of etymology questions, the focus is the subject complement of the passive sentence: “Why are chicken wings called Buffalo wings?”. In all other cases, the focus is the syntactic subject of the question, e.g. “Why are flamingos pink?” [18].

Cue word feature. The cue word feature is the overlap between a fixed set of explanatory cue words and the set of answer words. We found the cue words in a way that is commonly used for finding answer cues: we queried the key answer words to the most frequent *why*-question on the web (“blue sky rayleigh scattering” for “Why is the sky blue?”) to the MSN Search engine¹¹ and crawled the first 250 answer fragments that are retrieved by the engine. From these, we manually extracted all phrases that introduce the explanation. This led to 47 cue words/phrases such as *because, as a result of, which explains why, etc.*

Document structure features. The six document structure features cover information about the document context of a candidate answer passage: overlap between the question and the title of the Wikipedia document, overlap between the question and the title of the section in which the candidate answer occurs, and the relative position of the candidate answer in the document.

WordNet Relatedness feature. We define the relatedness between a question and an answer as the weighted average of the relatedness between each of the question words and each of the answer words:

$$REL(Q, A) = \frac{\sum_{q=1}^m \sum_{a=1}^n REL(w_q, w_a)}{m} \quad (3)$$

in which Q, A is the question-answer pair under consideration, w_q represents the question words, w_a the answer words, and m is the number of question words. As measure of relatedness, we choose the Lesk measure, which incorporates

¹¹<http://www.live.com>

Table 1: Set of 37 features used in our re-ranking module

| | |
|----------------------------|---|
| TF-IDF | The score that is assigned to a candidate answer by Lemur/TF-IDF in the retrieval module |
| 14 Syntactic feats | Overlap between question constituents (e.g. subject, verb, question focus) and answer words |
| 14 WordNet expansion feat | Overlap between the WordNet synsets of syntactic question constituents and answer words |
| 1 Cue word feat | Overlap between candidate answer and a pre-defined set of explanatory cue words |
| 6 Document structure feats | Overlap between question (focus) words and document title and section heading |
| 1 WordNet Relatedness feat | Relatedness between question and answer according to the WordNet similarity tool [13] |

information from WordNet glosses. It finds overlaps between the glosses of two concepts, also if they belong to different word classes [13].

3.3.2 Resulting feature vectors

Feature extraction led to a vector consisting of 37 feature values for each of the 28,050 items in the data set. We experiment with two types of normalization: L2 normalization over all feature values per item (‘horizontal normalization’) and L1 normalization over all values for one feature, grouped per question (‘vertical normalization’). Each item (representing one question-answer pair) was automatically labeled ‘1’ if the candidate answer matches the answer pattern for the question and ‘0’ if it does not. In total, 295 (1%) items in our set were labeled ‘1’ and the rest was labeled ‘0’.

4. THE LEARNING PROBLEM

Based on the goal of our work (see Section 1) and the data we work with (see Section 3), we can identify the following characteristics of the learning to rank problem that we aim to solve in this paper:

- We aim at developing a system for answering *why*-questions. In the development phase, we use a set of 187 *why*-questions that have been asked to an online QA system. The system that we build should be generalizable to new *why*-questions; it should not depend on a database of previously answered questions. For the evaluation set-up of our experiments, this means that we must split the training and test collections in such a way that are all answers to the same question occur in either the training set or the test set.
- In our data collection, we have much more negative than positive instances (99% has value ‘0’). This class imbalance means that the baseline for classification tasks on these data is extremely high: if a classifier would classify all instances as ‘0’, then accuracy would be 99%. This is not desirable because the evaluation of the results is based on QA evaluation measures (see Section 5.2) and without positive instances in the output, the values of these measures will be zero.
- The previous point would suggest an approach based on ranking optimization, like it is performed by Ranking SVM [9]. However, Ranking SVM expects a ranked ground truth, i.e. multi-level evaluation (as opposed to binary labels 0 and 1). Since we defined the relevance of the answers as a binary variable, our learning problem seems more suited for classification than for ranking optimization.
- We use a set of features between which complex relations exist. Some of our features are correlated and

others even depend on each other. For example, when the overlap between the main verb in the question and the verbs in the answer passage is > 0 then the overlap between the main verb in the question and all words in the answer passage is automatically > 0 as well. For this paper, we did not calculate the correlations and dependencies between all pairs of 37 features, so we cannot completely oversee the complexity of the feature set. Complex feature relations may cause challenges when using linear classification algorithms.

5. EXPERIMENTS

5.1 Baseline

As baseline we use the system setting in which the answers are retrieved and ranked by Lemur/TF-IDF, without application of the re-ranking module. Thus, in the baseline setting, the answers are ranked according to the single feature value TF-IDF.

5.2 Evaluation set-up

After labeling each of the instances with 0 or 1 with use of the answer patterns (see Section 3.1), we count the questions that have at least one relevant answer in the top n ($n = 10, 150$) of the results. This number divided by the total number of questions in our test collection gives the measure $success@n$. For the highest ranked relevant answer per question, we determine the reciprocal rank (RR). If there is no relevant answer retrieved by the system at $n = 150$, the RR is 0. Over all questions, we calculate the mean RR : $MRR@150$.

In the learning to rank stage, we perform 5-fold cross validation on the question set. We keep the 150 answers to each question together in one fold so that we do not train and test on the answers to the same question.

5.3 Learning algorithms and optimization functions

In this section, we give an overview of the learning algorithms and cost functions that we use for our experiments, and how we apply them to our learning problem. In each of the settings, we use the 37-feature set that we described in Section 3.3.

5.3.1 Genetic algorithm

As we pointed out in Section 2, genetic algorithms have the advantage that the cost function (‘fitness function’) is user-defined. This means that genetic algorithms allow us to experiment with different cost functions and to optimize directly for ranking performance (MRR or some related measure). Our aim when training the genetic algorithm is to find the optimal weight vector for our feature vector of 37

feature values. As weights, we use the integers 0 to 10. In terms of the genetic algorithm, each possible weight vector is an individual. For each individual that is generated by the algorithm, our fitness function linearly multiplies the weight vector with the feature vectors for all items in the training set. This leads to a new score for each item.

We experiment with two fitness functions in the genetic algorithm:

1. MRR. The fitness function converts new item scores to ranks by simply sorting them per question, and then calculates MRR over the complete training set. By adapting ('evolving') the weight vector over a number of generations¹², the genetic algorithm optimizes MRR for the training set.
2. Pairwise preference learning. We implement a fitness function that optimizes Kendall Tau (see Equation 1 in Section 2). In the fitness function, all pairs of one positive (1) and one negative (0) item are selected from the training set and their newly calculated scores are compared. If the positive item has a higher score than the negative item, the pair is concordant — otherwise it is discordant. From the counts for concordant and discordant pairs, the fitness function calculates τ .

5.3.2 Logistic regression

We use the *lrm* function from the Design package in R¹³ for training and evaluating models based on logistic regression. Using a set of input variables (features) logistic regression establishes a function that determines the log of the odds that the item is relevant (has label '1'). The log odds are defined as:

$$\ln \text{odds}(rel_i = 1) = \frac{e^{P(rel_i=1)}}{1 + e^{P(rel_i=1)}}, \quad (4)$$

in which $P(rel_i = 1)$ is the probability that item i is relevant.

The regression function that outputs the log odds is defined as follows:

$$\alpha + \vec{\beta}_k \vec{V}_{ik}, \quad (5)$$

in which α is the intercept. $\vec{\beta}_k \vec{V}_{ik}$ are the weights β and values V_i of the features k . The optimal values for α and β_k are found with the help of Maximum Likelihood Estimation (MLE).

In the test phase, the regression function is applied to the instances in the test set, predicting for each item the log odds that it should be categorized as '1'. We convert these log odds to ranks by sorting them per question. This way, we can calculate MRR for the test set.

We experiment with two different cost functions with logistic regression:

1. MLE default regression. We build a logistic regression function using all 37 features from our set as input, without interactions.

¹²In these experiments, we set generation size to 500 and the number of generations to 50.

¹³See <http://cran.r-project.org/web/packages/Design/index.html/>

2. MLE stepwise regression. We use a recursive wrapper that in each step adds the significant feature that gives the highest improvement in terms of MRR for the training set. Then it builds an MLE regression function using the newly added feature and the features kept from previous rounds. The wrapper stops adding features once no improvement is gained anymore or no significant features are left according to the MLE regression module.

5.3.3 SVM

In order to assess the performance of discriminative models of classification (as opposed to regression-based models) to our data, we investigate the use of support vector machines (SVMs) [4].

We use version 6 of *SVM^{light}*¹⁴ for training and testing support vector machines. In the testing phase, *SVM^{light}* assigns a score to each of the data instances. We convert these scores to ranks by sorting them per question. We first experiment with linear and polynomial kernels.

We use *SVM^{light}* in two different cost functions:

1. Classification. SVMs attempt to derive the hyperplane that optimally separates data in different classes (with a margin as large as possible). This hyperplane is described by a linear function in a high dimensional space. The optimization problem for SVMs consists of finding a weight vector \vec{w} and a constant b , such that $\frac{1}{2} \|\vec{w}\|^2$ is minimized w.r.t. $c_i ((\vec{w} \cdot \vec{x}_i - b) \geq 1, (1 \leq i \leq n))$, in which \vec{x}_i a data vector, and $c \in \{1, -1\}$ the class of \vec{x}_i .
2. Pairwise preference learning (Ranking SVM [9])¹⁵.

6. RESULTS

The results that we obtained using the different machine learning techniques and optimization functions are in Table 2. In the case of SVM, we only show the results obtained with the linear kernel, since a polynomial kernel did not improve the results. For all settings, success@150 is 78.5%. This score does not change by re-ranking the results. For significance testing, we used the Wilcoxon Signed-Rank test on paired reciprocal ranks.

Table 2 shows that the best results are obtained with logistic regression (both MLE default as stepwise MLE optimizing MRR) on data that were vertically normalized per question (rows 5 and 7 respectively). Although vertical normalization per question seems to give better results than horizontal vector normalization for all settings, the difference is only significant ($p = 0.024$) for MLE default logistic regression (rows 4 and 5 compared).

If we compare the results for the different settings on the horizontally normalized data, we see that stepwise logistic regression (row 6) and the genetic algorithm optimizing

¹⁴See <http://svmlight.joachims.org/>

¹⁵We are aware of the fact that Ranking SVM expects multi-level relevance as opposed to our binary labeling, but it still is interesting to see what can be done with pairwise preference ranking.

Table 2: Results for all learning settings in terms of MRR and Success@10. Success@150 is equal for all settings: 78.5%. An asterisk (*) indicates a statistically significant improvement ($P < 0.01$ according to the Wilcoxon Signed-Rank test) over the baseline. For each evaluation measure, the highest score is printed in bold face.

| Feature set | learning algorithm and cost function | MRR | success@10 |
|-----------------------------|---|---------------|---------------|
| 1 TF-IDF (baseline) | - | 0.249 | 45.21% |
| 2 37 feats, horizontal norm | genetic, scores to ranks, optimizing MRR | 0.309* | 53.48% |
| 3 37 feats, horizontal norm | genetic, pairwise preference learning, optimizing Tau | 0.301* | 54.01% |
| 4 37 feats, horizontal norm | logistic regression, MLE default | 0.273 | 51.43% |
| 5 37 feats, vertical norm | logistic regression, MLE default | 0.341* | 58.82% |
| 6 37 feats, horizontal norm | logistic regression, MLE stepwise, optimizing MRR | 0.301* | 56.14% |
| 7 37 feats, vertical norm | logistic regression, MLE stepwise, optimizing MRR | 0.328* | 56.69% |
| 8 37 feats, horizontal norm | SVM, linear kernel, pairwise preference learning | 0.048 | 6.95% |
| 9 37 feats, horizontal norm | SVM, linear kernel, classification | 0.053 | 11.62% |

MRR (row 2) give similar results: MRR is around 0.305. Default MLE logistic regression (row 5) on the horizontally normalized data does not give significant improvement over the baseline: MRR is 0.273.

SVM (both classification and ranking with either a linear or a polynomial kernel) performs much worse than that and even lower than baseline: MRR for SVM is around 0.05.

7. DISCUSSION

After we have excluded the possibility of bugs in our experimental set-up for SVM, we follow up with a series of experiments to find out where the bad results with SVM come from and what kind of kernels and hyperparameters are needed for improving them. This is discussed in Section 7.1 below. In Section 7.2 we look at the best-scoring machine learning techniques and cost functions, and present the features they deemed most important.

7.1 More experiments with SVM

We consider three possible causes for the poor results we obtained with SVM: the presence of complex relations between the features in our set, the complexity of the features themselves, and the class imbalance in our data collection (much more negative than positive instances).

7.1.1 Complex feature relations

In Section 4, we pointed out that we use complex (structural and semantic) features. Some of our features are correlated and others even depend on each other. Therefore, we now experiment with a highly simplified version of our feature set in order to find out whether the complex feature relations cause the poor results. To this end, we removed all features except TF-IDF. If we use logistic regression to build a model for the training set using TF-IDF only and we apply this model to the test set, the probabilities assigned to the instances lead to baseline ranking (MRR around 0.24). However, if we try this with SVM, we still get very poor results (MRR around 0.07, depending on the hyperparameter setting we choose). This means that the low scores that we obtain with SVM are not due to complex relations between the features in our set.

7.1.2 Complex features

In order to find out whether the poor results are caused by the complexity of the features themselves¹⁶, we experiment with a set of simple surface features. For every question-answer pair, we create a bag of WordNet expansions. The bag contains all WordNet synonyms, hypernyms, hyponyms, senses, and antonyms for the nouns and verbs in the question and in the candidate answer. For every WordNet expansion word in the bag, we counted its frequency in the bag and L1-normalize it. These L1-normalized frequencies constitute the feature vector for the question-answer pair. It is a probability distribution summing to 1.

L1-normalized data is most naturally learned by multinomial kernels, also known as information diffusion kernels [11]. These are kernels that deploy geodesic distance measures on L1-normalized data. Lafferty and Lebanon [11] argue that geodesic distance is often a better approximation of the information geometry of L1-normalized documents than plain Euclidean distance. Previous work [14] demonstrates that multinomial kernels are able to produce state of the art results for sentiment polarity classification tasks.

In SVM^{light} , we implement a simple, hyperparameter-free multinomial kernel, i.e. the shifted negative geodesic kernel K_{NGD} [19]:

$$K_{NGD}(\vec{x}, \vec{y}) = -2 \arccos \left(\sum_{i=1}^n \sqrt{x_i y_i} \right) + \pi \quad (6)$$

in which \vec{x} is a support vector from the training data, \vec{y} is a feature vector from the test data, and i_1, \dots, i_n is the set of features occurring in both \vec{x} and \vec{y} .

With this kernel, we again obtain a result of $MRR = 0.07$ for the surface WordNet features. This indicates that the complexity of the features does not cause the low scores.

7.1.3 Class imbalance

In Section 4, we pointed out the problem of class imbalance in our data. As a first option for solving this problem, we vary the cost-factor in SVM^{light} by which training errors on positive examples outweigh errors on negative examples¹⁷.

¹⁶TF-IDF itself is a complex measure that combines a number of counts in one function.

¹⁷See the documentation of SVM^{light} for the implementation of the cost-factor.

We experiment with a cost-factor of 10 and 100. This does however not improve our results in terms of MRR, which were still around 0.05.

Next we opt for another method: a sampling based approach akin to bootstrap aggregating or bagging [1]. We again use the simple training data that we described in Section 7.1.2. We sample the training data for a number of n times, drawing with replacement a number of exactly k items from the training data, among which, for every sample, are all relevant instances (labeled ‘1’). For every such sample, a separate classifier is trained and applied to the test data, after which the decision values of all classifiers are averaged to produce the final result. The optimal values of n and k are determined through grid search. For $n = 5$ samples of $k = 400$ items we obtain a result of MRR=0.147. This result, while still well below baseline, at least demonstrates that the original SVM results can be significantly improved by tackling the class imbalance.

7.1.4 Support Vector Regression

We also experiment with support vector regression¹⁸ to find out if support vectors can be used for training regression functions for our 37-features data. We use SVMlib¹⁹ for these experiments. We normalize the feature values vertically per question since we found in Section 6 that this gives better results than horizontal vector normalization. We choose the following parameter values: $c = 0.00195$, $\gamma = 0.000122$, $\nu = 0.1$ and we used an RBF kernel²⁰. With these settings, we obtain an MRR score of 0.338 with success@10 57.75%, which is similar to the results obtained with logistic regression.

7.1.5 Future suggestions

Since we learnt that the poor results from SVM can at least partly be explained by the extreme class imbalance, we plan to apply the bagging method to our original set of 37 features in the near future.

We have one other suggestion for future experiments with SVM: The current setup treats the answer ranking problem essentially as a binary classification problem: questions are paired with answers, and the possible outcomes of this pairing are ‘0’ (irrelevant) and ‘1’ (relevant). From this binary classification problem, we tried to deduce a ranking by treating the binary ground truth as the discretization of a continuous decision function. This effectively may not be the best option, hampering, for one thing, the use of ranking classifiers such as Ranking SVM, which presuppose ranked ground truth. It would seem that this forced way of learning a ranking from binary data only aggravates the problem of class imbalance in our data.

7.2 Important features

¹⁸See <http://svms.org/regression/>

¹⁹See <http://cs.haifa.ac.il/YOSI/PCOMP/>

²⁰We tested linear and RBF kernels with various parameter settings across the possible range of settings for the first training fold. This way, we obtained oracle parameters for this fold. Then, we applied the same parameter settings to the other training folds. We unfortunately did not have time for optimizing parameter settings for all training folds.

Both the genetic algorithm and the stepwise regression approach give good experimental results. This makes it interesting to see which features made the improvement. In order to find out which features are the most important for ranking the answers using the genetic algorithm, we selected the features that were assigned an average weight larger than 7 with a standard deviation smaller than 2 over the five folds (see Table 3). We also had a look into the features that were selected as significant features in at least two of the five folds in the stepwise regression approach (see Table 4).

Table 3: The features that were assigned an average weight > 7 with a standard deviation < 2 over the five folds by the genetic algorithm. Behind each feature in Table 3 is the average weight that was assigned to the feature over the five training folds.

| Feature | Average weight |
|--------------------------------|----------------|
| nonfocus overlap | 10 |
| TF-IDF | 9.8 |
| cue words | 9.8 |
| verb synonym overlap | 9.6 |
| doctitle focus synonym overlap | 7.4 |

Table 4: The features that were selected as significant features in at least two of the five folds in the stepwise regression approach.

| Feature | # folds |
|--------------------------------|---------|
| TF-IDF | 5 |
| doctitle focus synonym overlap | 5 |
| doctitle focus overlap | 4 |
| WordNet relatedness | 3 |
| head overlap | 3 |
| passage position | 2 |
| doctitle synonym overlap | 2 |

There are a few differences between Table 3 and 4, showing that similar results can be obtained with different subsets of our features. This is partly due to feature redundancy: the same information is sometimes described by two different features. E.g. the question’s main verb is always the head of a verb phrase. Therefore, the feature ‘head overlap’ represents partly the same information as the feature ‘verb overlap’.

We see in Table 3 that the presence of cue words can give useful information in re-ranking answer paragraphs²¹. In fact, incorporating the presence of cue words is a step towards recognizing that a passage is potentially an answer to a *why*-question. As argued in Section 1, identifying a passage as a potential answer is the important issue in *why*-QA, since answers cannot be recognized by simple semantic-syntactic units such as named entities as is the case for factoid QA.

In both feature selections, we see the importance of question focus and document title. The importance of question focus for *why*-QA is especially interesting because it is a question feature that is specific to *why*-questions and does not similarly apply to factoids or other question types. Moreover, the overlap between the question focus and the document

²¹In the stepwise regression approach, this feature was only selected in one of the five folds.

title shows that Wikipedia as an answer source can provide QA systems with more information than a collection of plain texts without document structure does. In Table 3, we see that the overlap between the non-focus part of the question and the passage is also important. We can clarify this with a simple example: the question “Why are flamingos pink?” has *flamingo* as focus and *pink* as non-focus part. We can find the answer to this question in the Wikipedia document with title *flamingos*, in the passage that describes their *pink* color. In general, in cases where the question focus leads to the document title, the non-focus part often leads to the answer passage within this document.

8. CONCLUSION

We can draw two important conclusions from the current paper.

First, a learning to rank approach using either a regression technique or a genetic algorithm that optimizes for MRR leads to a significant improvement over the TF-IDF baseline. We reach an MRR of 0.341 with a success@10 score of 58.82%. Although this improvement is significant, the system is still limited to answering 59% of the *why*-questions in the top 10, while almost 80% of questions have a relevant answer somewhere in the top 150. In the near future, we plan to find out which answers are retrieved but not ranked in the top 10 and why. We also plan to investigate the 20% of the questions in our set are not retrieved by our QA system at all.

Second, we found that the results obtained with SVM are very poor compared to the results obtained with logistic regression and genetic algorithms. In future work, it would be interesting to experiment with (1) bagging techniques applied to our set of 37 features, as potential solution for the data imbalance; and (2) a ranked ground truth (multi-level instead of binary labeling), so that ranking classifiers such as Ranking SVM can be better applied to our data.

9. ACKNOWLEDGEMENTS

We would like to thank Hans van Halteren for his experiments with support vector regression. Also, we thank the anonymous reviewers for their valuable comments.

10. REFERENCES

- [1] L. Breiman. Bagging predictors. In *Machine Learning*, pages 123–140, 1996.
- [2] C. Burges, T. Shaked, E. Renshaw, A. Lazier, M. Deeds, N. Hamilton, and G. Hullender. Learning to rank using gradient descent. In *Proceedings of ICML 2005*, volume 22, page 89, 2005.
- [3] B. Carterette and D. Petkova. Learning a ranking from pairwise preferences. In *Proceedings of SIGIR 2006*, pages 629–630. ACM New York, NY, USA, 2006.
- [4] N. Cristianini and J. Shawe-Taylor. *An introduction to support Vector Machines: and other kernel-based learning methods*. Cambridge University Press, New York, NY, USA, 2000.
- [5] L. Denoyer and P. Gallinari. The Wikipedia XML corpus. *ACM SIGIR Forum*, 40(1):64–69, 2006.
- [6] W. Fan, E. Fox, P. Pathak, and H. Wu. The Effects of Fitness Functions on Genetic Programming-Based Ranking Discovery for Web Search. *Journal of the American Society for Information Science and Technology*, 55(7):628–636, 2004.
- [7] J. Furnkranz and E. Hullermeier. Pairwise Preference Learning and Ranking. *Lecture Notes in Computer Science*, pages 145–156, 2003.
- [8] E. Hovy, U. Hermjakob, and D. Ravichandran. A Question/Answer Typology with Surface Text Patterns. In *Proceedings of HLT 2002*, San Diego, CA, 2002.
- [9] T. Joachims. Optimizing search engines using clickthrough data. In *Proceedings of ACM SIGKDD 2002*, pages 133–142. ACM, 2002.
- [10] M. Khalid and S. Verberne. Passage Retrieval for Question Answering using Sliding Windows. In *Proceedings of COLING 2008, Workshop IR4QA*, 2008.
- [11] J. Lafferty and G. Lebanon. Diffusion kernels on statistical manifolds. *Journal of Machine Learning*, 6:129–163, 2005.
- [12] T. Liu, J. Xu, T. Qin, W. Xiong, and H. Li. Letor: Benchmark dataset for research on learning to rank for information retrieval. In *Proceedings of SIGIR 2007 Workshop on Learning to Rank for Information Retrieval*, 2007.
- [13] T. Pedersen, S. Patwardhan, and J. Michelizzi. WordNet:: Similarity-Measuring the Relatedness of Concepts. In *Proceedings of the National Conference on Artificial Intelligence*, pages 1024–1025, 2004.
- [14] S. Raaijmakers. Sentiment classification with interpolated information diffusion kernels. In *Proceedings of the First International Workshop on Data Mining and Audience Intelligence for Advertising (ADKDD’07)*, 2007.
- [15] J. Tiedemann. A Comparison of Genetic Algorithms for Optimizing Linguistically Informed IR in Question Answering. *LECTURE NOTES IN COMPUTER SCIENCE*, 4733:398, 2007.
- [16] A. Trotman. An Artificial Intelligence Approach To Information Retrieval. *Proceedings of the SIGIR 2004 Doctoral Consortium*, page 603, 2004.
- [17] S. Verberne. Paragraph retrieval for why-question answering. In *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 922–922. ACM Press New York, NY, USA, 2007.
- [18] S. Verberne, L. Boves, N. Oostdijk, and P. Coppen. Using Syntactic Information for Improving Why-Question Answering. In *Proceedings of COLING 2008*, 2008.
- [19] D. Zhang, X. Chen, and W. S. Lee. Text classification with kernels on the multinomial manifold. In *Proceedings SIGIR’05*, pages 266–273, 2005.