Learning to Rank QA Data

Evaluating Machine Learning Techniques for Ranking Answers to Why-Questions

Suzan Verberne
CLST, RU Nijmegen
s.verberne@let.ru.nl

Hans van Halteren
CLST, RU Nijmegen

Daphne Theijssen
Dept. of Linguistics,
RU Nijmegen

Stephan Raaijmakers
TNO, Delft

Lou Boves
CLST, RU Nijmegen

ABSTRACT

In this work, we evaluate a number of machine learning techniques for the purpose of ranking answers to *why*-questions. We use a set of 37 linguistically motivated features that characterize questions and answers. We experiment with a number of machine learning techniques in various settings. The purpose of the experiments is to assess how the different machine learning techniques can cope with our highly imbalanced binary relevance data. We find that with all machine learning techniques, we eventually obtain an MRR score that is significantly above the TF-IDF baseline of 0.25 and not significantly lower than the best score of 0.35. Regression techniques seem the best option for our learning problem.

1. INTRODUCTION

The research reported in this paper is part of a project that aims at developing a question answering system for *why*-questions (*why*-QA). Answers to *why*-questions tend to be at least one sentence and at most one paragraph in length [22]. Therefore, passage retrieval appears to be a suitable approach to *why*-QA.

In previous work [24] we describe a system for *why*-QA that consists of a pipeline of an off-the-shelf passage retrieval engine (Lemur[1]), and a re-ranking module that uses a set of features extracted from the question and each of the candidate answers. 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. This has led to a set of 37, mostly linguistically motivated, features representing the degree of overlap between a question and each of its candidate answers. In previous work, we used a genetic algorithm for finding the optimal weights for combining the 37 features [23].

[1]See http://www.lemurproject.org/

In the current paper, we aim at improving the ranking performance of our system by finding the optimal approach to learning to rank [14] while keeping our feature set unchanged. More specifically, we try to find the optimal ranking function to be applied to the set of candidate answers in the re-ranking module.

The task of learning to rank in the context of a QA system has two important challenges (which are discussed in more detail in Sections 2 and 3). First, in QA systems relevance is often treated as a binary variable: a candidate answer is either relevant (correct), or irrelevant (incorrect). Following approaches in factoid QA, we decided to consider answer relevance for *why*-questions to be a binary variable (see Section 3). Therefore, a suitable approach for learning answer relevance would be to consider the task as a classification problem. All operational QA systems, however, aim at presenting a ranked list of answer candidates for each individual input question [17]. For our learning-to-rank set-up, this means that we have to induce a ranked list of answers from binary relevance judgments[2].

A second specific problem in learning to rank QA data is the high imbalance between positive and negative instances (correct and incorrect answers) in the training set: there tend to be much more negative than positive instances [21]. As we will see in Section 3, this is also the case for our *why*-QA data.

In this paper, we evaluate a number of machine learning techniques for the task of learning a ranking function for *why*-answers: Naive Bayes, Support Vector Classification, Support Vector Regression, Logistic Regression, Ranking SVM, and a Genetic Algorithm. We apply these techniques in pointwise, pairwise and listwise set-ups. Most, but not all, of these techniques require tuning of several hyperparameters. In Section 4, we discuss the techniques that we evaluated and how we applied them to our learning-to-rank task.

The goal of this paper is to compare the machine learning techniques listed above in their performance on our task: inducing a ranking from a binary classification. In doing so, we cast the problem in three different ways: optimizing the ranking pointwise, pairwise and listwise.

The goal of this paper is to compare the machine learning techniques listed above in their performance on our task:

[2]In ranking binary relevance data, the goal is to rank the correct answers higher than the incorrect answers. There is no ranking among the (in)correct answers themselves.

inducing a ranking from a binary classification. In doing so, we will pay attention to three factors: (1) the distinction between the pointwise approach, in which candidate answers are classified individually (over all questions), and the pairwise and listwise approaches, in which the ranking within each cluster of answers is optimized; (2) the distinction between techniques based on classification and techniques based on regression. (3) Since most, but not all, of these techniques require tuning of several hyperparameters, we investigate the distinction between techniques with and without hyperparameters that must be tuned. In Section 4, we discuss the techniques that we evaluated and how we applied them to our learning-to-rank task. In all cases we will use a measure for the quality of the ranking as the evaluation criterion.

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

2. RELATED WORK

In this section, we discuss related work on the two main characteristics of the QA learning problem mentioned in Section 1: Learning to rank binary relevance data (Section 2.1) and the problem of imbalanced data (Section 2.2).

2.1 Learning to Rank binary relevance data

As explained in Section 1, one of the characteristics of evaluation in QA is that relevance is generally defined as a binary variable[3] [25] while the output of a QA system is a ranked list of answers that are clustered per input question [17]. Learning a ranking function for a problem with binary relevance judgments can be achieved in three ways: (1) inducing a ranked list from a binary classification of all question-answer pairs irrespective of the clustering of the answers (pointwise approach), (2) classifying pairs of correct and incorrect answers for their mutual order and optimizing the proportion of correctly ordered answers (pairwise approach), or (3) optimizing a cost function on the ordering of answers within one answer cluster (listwise approach).

The pointwise approach

One approach to learning answer relevance in QA is considering the learning problem a classification task. In the training phase, the clustering of answers per question is ignored: the task of the classifier is to learn whether an instance should be classified as '1' or '0', irrespective of the answer cluster it belongs to. Relations between the candidate answers to the same question are ignored.

A ranked answer list can then be induced by letting the classifier assign a score to each instance in the test set, expressing the probability that it should be classified as correct and then ordering the answers per question according to these scores (ordinal sort) [2]. Techniques that can be applied in this approach are classifiers (such as Naive Bayes and Support Vector Classification) and regression techniques (such as Logistic Regression and Support Vector Regression) [5, 9].

The pairwise approach

An alternative way of learning a ranking for the answers within a cluster is to classify pairs of correct and incorrect answers for their mutual order and optimizing the proportion of correctly ordered answers. This learning principle is called 'pairwise preference learning', and was introduced by Joachims [12], who created the learning algorithm Ranking SVM based on this principle. Pairwise preference learning has been studied in more detail by in [8] and applied to several ranking problems such as combining rankings from multiple retrieval systems in [4].

The listwise approach

A third way of learning to rank answers with binary relevance judgments is the listwise approach, in which a cost function for the ordering of answers within one answer cluster is optimized. A description of learning algorithms based on the listwise approach can be found in [26] for data with multiple ranking levels and and [3] who applied listwise ranking also to data with binary relevance ranks. The listwise approach is devised for learning to rank data with multiple relevance levels. Therefore, we expect the listwise approach to be less suitable for our binary relevance data than the pointwise and pairwise approaches.

In [23] we followed the listwise approach by implementing a genetic algorithm for finding the optimal ranking function. Genetic algorithms are devised for finding an optimum in a very large data space. The meaning 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 [20, 7, 19]. The approach presented in [20] resembles our approach: it defines the learning problem as the search for the optimal weight vector for a given feature vector.

The ranking performance can be defined and implemented in the fitness function in different ways. In [7], a number of fitness functions that are derived from ranking evaluation measures (such as Mean Average Precision) are compared for their effectiveness. In [23], Mean Reciprocal Rank (MRR) is used as optimization measure in the fitness function.

2.2 The problem of imbalanced data

As mentioned in Section 1, class imbalance is a general problem in learning to rank QA data [21]. Class imbalance is especially problematic for classifiers because the classification baseline is extremely high: if 98% of the instances in the training set has the label 'incorrect', then classifying all instances as 'incorrect' gives an accuracy of 98%. This hampers the optimization process.

The problem has been acknowledged by many researchers in the machine learning field [11, 21, 1, 18]. Because SVMs are very popular for all sorts of classification tasks, much work on tackling the problem of imbalanced data is focused on making SVMs robust to imbalance. In the literature, three approaches to curing problematic class imbalances for

[3]Although the quality of answers to *why*-questions can be judged on a multi-level scale, we found that these judgments are very subjective. Therefore, we decided to consider relevance to be a binary variable: does the passage answer the question, or not?

classifiers are discussed: undersampling the majority class, oversampling the minority class and cost-modifying according to the same ratio as the class balance. In general, the latter approach gives the best results for various classifiers [11, 18]. In Section 4.2, we explain our attempts for curing the class imbalance in our data.

Class imbalance causes fewer problems for regression techniques than for classifiers. In the regression model, the so-called 'intercept' value moves the outcome of the regression function up or down towards the bias in the data. If the class imbalance is not too extreme, the intercept can be adapted so that the regression function is robust against it [15]. In Section ??, we come back to the effect of class imbalance on regression techniques.

Ranking techniques such as Ranking SVM are not sensitive to class imbalance. This is because they employ pairs of correct and incorrect answers from the same cluster, thereby balancing the training data or they optimize for a ranking criterion based on the highest ranked correct answer per cluster (independent of how many correct answers there are in the cluster). In Section 4.4, we discuss these techniques in more detail.

3. DATA AND SYSTEM SET-UP

3.1 Resources

For our experiments, we used the Wikipedia INEX corpus [6]. This corpus consists of all 659,388 articles extracted from the online Wikipedia in the summer of 2006, converted to XML format. Before indexing the corpus, we segmented all Wikipedia documents into passages. We decided to use a semi-fixed passage size of 500 to 600 characters (excluding all XML markup) with an overflow to 800 for the purpose of completing sentences.We created passage overlap by starting each new passage at a paragraph or sentence boundary halfway the previous passage. For Wikipedia articles that contain fewer than 500 characters in total, we included the complete text as one passage. Our segmentation process produced an index of 6,365,890 passages. We separately saved the document title and section heading as metadata for each passage because this information is used in our feature set.

For development and testing purposes, we exploited the Webclopedia question set by Hovy et al. [10]. This set contains questions that were asked to the online QA system *answers.com*. Of these questions, 805 (5% of the total set) were *why*-questions. For 700 randomly selected *why*-questions from this set, we manually searched for an answer in the Wikipedia XML corpus. Two examples 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."

For 186 of the 700 *why*-questions from the Webclopedia data, we were able to manually find the answer in the Wikipedia corpus[4]. Thus, our data collections consists of 186 *why*-questions.

3.2 System set-up

Our system consists of three modules that are run in sequence:

(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. Here, we use Lemur to retrieve 150[5] answers per question and rank them using TF-IDF as it has been built in in Lemur[6]. This gives us a set of 186 questions with 150 candidate answers per question, with for each pair of a question and a candidate answer a TF-IDF score.

(3) A re-ranking module that re-ranks the retrieved passages using features extracted from the question and each of the candidate answers (see Section 3.4 below): 28,050 ($186 * 150$) question-answer pairs (instances) in total. Finding the optimal ranking function for these data is the aim of this paper.

3.3 Ground truth labeling

In general (with the exception of questions for which the definite answer is topic of discussion), *why*-questions have, just like factoid questions, only one correct (and thus complete) answer. However, that answer can be formulated in different ways. As a result, we had to take into account the possibility of variants, which made it impossible to apply fully automatic ground truth labeling. We therefore manually assessed each of the candidate answers in our set as being correct or incorrect[7]. We supported our manual judgments with a set of TREC-style answer patterns: a regular expression for each question that defines which answers should be labeled as correct.

For example, for question 2 above, we developed the following answer pattern after assessing all candidate answers in our set: */(Socrates.* opportunity.* escape.* Athens.* considered.* hypocrisy | leave.* run.* away.* community.* reputation)/*. The pattern is based on two variants of the correct answer that we found in the set of candidate answers[8]. By producing these answer patterns we ensure that our labeling can be reproduced in future experiments.

3.4 Feature extraction

[4]Thus, about 25% of our questions have an answer in the Wikipedia corpus. The other questions are either too specific ("Why do ceiling fans turn counter-clockwise but table fans turn clockwise?") or too trivial ("Why does a chicken cross the road?") for the coverage of Wikipedia in 2006.

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

[6]In previous work [13], we experimented with other ranking models and TF-IDF came out as the best.

[7]The assessments were originally done by one annotator. We are currently working on assessments by a second annotator in order to be able to estimate the difficulty of the assessment task and the reliability of the ground truth annotations.

[8]Note that the vertical bar separates the two alternative formulations.

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 and answer constituents (e.g. subject, verb, question focus)
14 WordNet expansion feats	Overlap between the WordNet synsets of question and answer constituents
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 [16]

From earlier work [24], we compiled a set of 37 features that are summarized in Table 1. We syntactically parsed the questions with the Pelican parser[9] and the candidate answers with the Charniak parser. Then we used a Perl script to extract 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's noun phrases, or the question subject) and a set of answer items (for example: all answer words, or all syntactic 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 used a statistic derived from the Jaccard index that was 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: $S(Q,A) = \frac{|Q \cap A'| + |Q' \cap A|}{|Q \cup A|}$

For a detailed description of the features we use we refer to Verberne et al. 2009 [24].

Resulting feature vectors and normalization

The normalization procedure is as follows. Feature extraction led to a vector comprising 37 feature values for each of the 28,050 items in the data set. We adopted the approach by Liu et al. [14] for feature normalization per answer cluster, because the absolute values of a feature for different questions were not comparable. Moreover, this approach makes it possible to normalize the scores independently of the answers to other questions: it can be performed for every new input question and its answers.

Assume a question Q_i with the candidate answers $A_j (j = 1..150)$. For each feature $F_k (k = 1..37)$, its value x_{ijk} is normalized by transforming it to its z-score: $x'_{ijk} = (x_{ijk} - \mu_{ik})/\sigma_{ik}$ in which μ_{ik} is the mean of all values of feature F_k for the candidate answers to Q_i and σ_{ik} is the standard deviation of all values of feature F_k for the candidate answers to Q_i.

3.5 Evaluation set-up

Each instance in our data was labeled 'correct' if the candidate answer was a correct answer to the question and 'incorrect' if it was not (see Section 3.3). On average, a *why*-question had 1.6 correct answers among the set of 150 candidate answers retrieved by Lemur.

After labeling each of the test instances with correct or incorrect, we counted the questions that have at least one correct answer in the top n ($n = 10, 150$) of the results. This number divided by the total number of questions in our test collection gave the measure *Success@n*. For the highest ranked correct answer per question, we determined the reciprocal rank ($RR = 1/rank$). If there was no correct answer retrieved by the system at $n = 150$, the RR is 0. Over all questions, we calculated the mean RR: MRR@150.

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. For techniques that require tuning of hyperparameters, we use a development set (see Section 4.1). In the training stage, we exclude the questions from our training data for which none of the 150 candidate answers is correct. The test set on the other hand does contain these questions, for which RR will naturally be 0.

3.6 The class imbalance in our data collection

In our data collection, we have many more incorrect than correct answers: the incorrect/correct ratio in our complete training set (all five folds together) is 71 to 1 (98.6% of the instances in the training set has value '0'). Akbani et al [1] consider a data set to be 'highly imbalanced' if the ratio of negative against positive instances is bigger than 50 to 1.

4. EXPERIMENTS

In this section, we describe how we applied each of the learning techniques to our learning problem. In all cases we used the 37-feature set that we described in Section 3.4.

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

4.1 Matrix of techniques

We considered the three learning to rank approaches introduced in section 2.1: the pointwise approach (see Section 4.2), the pairwise approach (Section 4.3) and the listwise approach (Section 4.4). In the pointwise approach, we evaluated the following classification and regression techniques: Naive Bayes, Support Vector Classification, Support Vector Regression and Logistic Regression. In the pairwise approach, we applied the same classification and regression techniques together with Ranking SVM. For the listwise approach, which we expected to be less suitable for our binary relevance data, we evaluated a Genetic Algorithm.

Hyperparameter tuning

For techniques that expect hyperparameter values, we not only evaluated the default hyperparameter setting but we also tried to find optimal values for the hyperparameters using a grid search over the range(s) of likely values. For hyperparameter tuning, it is necessary to use development data that is held out from the training set. We searched for hyperparameter values that give the best results in terms of MRR on the development set. Given the small number

[9]See http://lands.let.ru.nl/projects/pelican/

of questions in our training set[10], we decided to hold out 10 questions with their 150 answers from each training set. Because development sets of 10 questions are quite small, we selected three (non-overlapping) development sets for each fold and tuned three times.

As a further measure to prevent overfitting on the development sets, we selected three good hyperparameter settings for each development set, instead of simply taking the one leading to the best MRR. The three hyperparameter settings were selected as follows: The first was always the one leading to the best MRR on the development set. The second and third were the highest local optima that are further than five steps in the grid away from the first chosen point and from each other (see the descriptions of the used grids in 4.2).

During testing, the outputs of the nine models for the three development sets were combined by addition (after scaling them to a comparable range).

4.2 The pointwise approach

We first investigated the pointwise approach of applying classification and regression techniques to the problem of learning a ranking from binary relevance data. In the training phase, the classifier or regressor learns to classify each instance (question answer pair) as either correct or incorrect, irrespective of the cluster it belongs to. In the test phase, we let the model assign a score to each instance in the data representing the probability that this instance should be classified as correct. The actual ranking is done by a script that sorts the instances per cluster by the output score of the classifier.

As discussed in Section 3.6, our data show a strong class imbalance, with a incorrect/correct ratio in our complete training set of 71. This may cause problems for machine learning techniques that are designed for classification. Therefore, we applied a balancing strategy to all classification and regression techniques that we evaluated. As observed in the literature [11, 18], application of a cost factor is the preferred approach to counter imbalance. If a system did not allow for this, we applied oversampling of the positive instances. We will describe for each individual technique which strategy was applied.

We ran each machine learning technique with its default hyperparameter settings on the original (imbalanced) data and with a balancing strategy (cost factor or oversampling). We only kept the best performing one for hyperparameter tuning.

Naive Bayes classifier (NB)

For experiments with Naive Bayes, we used the `e1071` package in R.[11] This package does not allow for tuning of hyperparameters for Naive Bayes so we only ran the Naive Bayes classifier in its default setting.

SV Classification (SVC) and SV Regression (SVR)

For standard support vector methods, we used LIBSVM.[12] Following the LIBSVM guidelines, we first scaled our data using *svm-scale*. We experimented with support vector classification (SVC) and support vector regression (SVR). For both, we used the RBF kernel.

The RBF kernel expects two hyperparameters: c (the trade-off between training error and margin) and γ (a multiplication factor determining the range of kernel space vector norms). Their default values are $c = 1$ and $\gamma = 1/k$ (with k being the number of instances, giving a γ of 5.5×10^{-5} for our data). During grid search, we varied c from 2^{-13} to 2^{13} and γ from 2^{-13} to 2^{7} in steps of $\times 4$, [13] which closely resembles the grid search suggested in the LIBSVM documentation.[14]

SVC allows us to use a cost factor for training errors on positive instances, which we did. During hyperparameter tuning, we kept the cost factor unchanged at 71 ($-w1 = 71$). For SVR (which does not allow for a cost factor), we used oversampling of the positive instances in such a way that a training set included approximately as many positive as negative instances.

Logistic regression (LRM)

We used the `lrm` function from the `Design` package in R for training and evaluating models based on logistic regression[15]. LRM uses Maximum Likelihood Estimation (MLE) as optimization function. It has a built-in option for data balancing (applying a weight vector to all instances), of which we found that it has exactly the same effect on the data as oversampling the positive instances in the training set. The other hyperparameters in LRM (a parameter for handling collinearity in stepwise approaches and a penalty parameter for data with many features and relatively few instances) are not relevant for our data. Therefore, we refrain from hyperparameter tuning for LRM.

4.3 The pairwise approach

For the pairwise approach, we use the commonly available Joachim's Ranking SVM algorithm. In addition, we use the same classification and regression techniques as in the pointwise approach. We make this possible by transforming our data into instance pairs that can be handled by these techniques (as explained below).

Ranking SVM

We used version 6 of SVM^{light} for our Ranking SVM experiments[16]. Ranking SVM considers the training data to be a set of instance pairs; each pair consists of one positive and one negative answer. On these instances, the system performs pairwise preference learning [12]. The ranking order of a set of training instances is optimized according to Kendall Tau: $\tau = \frac{(N_c - N_d)}{(N_c + N_d)}$, in which N_c is the number of concordant item pairs (the two items are ordered correctly) and N_d is the number of discordant item pairs (the two items are ordered incorrectly).

For RankingSVM, we investigated both the linear and the RBF kernel. The linear kernel only expects the hyperpa-

[10] Around 120 because, as explained in Section 3.5, we excluded the 21% questions without correct answers and 20% for each fold to test on.

[11] See http://cran.r-project.org/web/packages/e1071/index.html

[12] See http://www.csie.ntu.edu.tw/~cjlin/libsvm/

[13] This means that each next value is 4 times as high as the previous, so we go from 2^{-13} to 2^{-11} to 2^{-9} etc.

[14] For SVR, we changed the grid on second thought after we found out that the default hyperparameter setting for our data was not included in the grid search, which led to suboptimal tuning results. In the adapted grid, we varied γ from 2^{-6} to 2^{7} in steps of $\times 2$, multiplied by the default value.

[15] See http://cran.r-project.org/web/packages/Design/index.html/

[16] See http://svmlight.joachims.org/

rameter c; the RBF kernel takes both the hyperparameters c and γ. The default values for these hyperparameters in SVM^{light} are for our data $c = 0.01$ and $\gamma = 5.5 \times 10^{-5}$. For tuning these parameters, we search over the same grid as for SVR and SVC.

4.3.1 *Classification and regression techniques*

To enable the use of pointwise techniques in a pairwise approach, we transformed our data into a set of instance pairs. We presented the answers in pairs of one correct and one incorrect answer (to the same question). We kept the number of features constant (at 37), but we transformed each feature value to the difference between the values of the two answers in the pair. In other words, we created feature vectors consisting of 37 difference values.

In the training data, each instance pair is included twice: 'correct minus incorrect' with class 1 and 'incorrect minus correct' with class 0. As a side-effect, this automatically cures the class imbalance. In the testing phase, we let the classifier assign to each instance pair the probability that it is correctly ordered. Then we transform the data back to normal answer instances by summing the scores for each answer i over all pairs $[i, j]$ in which i is ordered first.

In this pairwise approach[17], we applied the same classifiers and regression techniques as we evaluated for the pointwise approach: Naive Bayes, Support Vector Classification, Support Vector Regression and Logistic Regression.

4.4 The listwise approach

In the listwise approach there is no classification of instances or instance pairs; instead, the ordering of an answer cluster as a whole is optimized. As explained in Section 2.1, the listwise approach might suffer because of our binary relevance data.

Genetic algorithm (GA)

We used a Perl implementation of a genetic algorithm[18] for our experiments. As we pointed out in Section 2.1, genetic algorithms allow us to to optimize directly for ranking performance. This might compensate for the problems due to the binary ranking. Our aim when training the genetic algorithm was to find the optimal weight vector for our feature vector of 37 feature values. As weights, we used the integers 0 to 10. In terms of the genetic algorithm, each possible weight vector is an individual.

We implemented two fitness functions in the genetic algorithm: one performing pairwise preference learning by optimizing for Kendall Tau (Equation 2), and one optimizing for MRR. In each run ('generation'), the GA selects the fittest individuals for crossover ('mating').

By default, the crossover rate is 0.95 and the mutation rate 0.05. For the selection of individuals, we chose tournament selection, because that is the most efficient strategy. We used uniform crossover, because the order of our features in the feature vector is not relevant. In our experiments, we set the generation size to 500 and the number of generations to 50 based on the shape of the learning curve in earlier experiments on the same data.

We did not perform hyperparameter tuning for the GA, because a grid search would need too much computational power.

5. RESULTS AND DISCUSSION

The results that we obtained are in Table 2. For all settings, success@150 is 78.5%. This score cannot change by re-ranking the results. For significance testing, we used the Wilcoxon Signed-Rank test on paired reciprocal ranks (RRs): Per question, we took the RR of the highest ranked correct answer in the two system versions. Then we made 186 pairs of RRs for the two system settings and calculated the Wilcoxon score over them. The highest MRR score that we obtained is 0.35[19] (by SVR for pairwise classification). We will call this the optimum in the remainder of this section.

First, we must note here that it is not relevant to compare the default settings of different techniques to each other since default hyperparameter values may be unapplicable to the data under consideration. However, we can make a number of interesting remarks on the results. The left side of Table 2 shows that especially pointwise-SVC gives very poor results in its default setting on our imbalanced data. But it also shows that if we balance the data, the default settings of LIBSVM already reach a score close to the optimum of 0.35. This suggests that the default hyperparameter values chosen by the developers of LIBSVM are more suitable for balanced data than for our original imbalanced data.

This is confirmed by the results presented in the right side of Table 2. Here we see that if the problem is presented as a pairwise classification problem, the default hyperparameter settings are much more suitable than for the original data representation. We assume that this is because the pairwise representation cures the class imbalance. In general, we see that the results for the pairwise approach much resemble the results for balanced data in the pointwise approach: For SVC (0.318 vs. 0.316), SVR (0.328 vs. 0.320) and LRM (0.307 vs. 0.307) the results are almost exactly the same in these two settings. Table 2 also shows that in Ranking SVM the results for the RBF kernel are only slightly (but not significantly) better than those for the linear kernel. Apparently, our data can be learned properly using a linear kernel, at least in the case of pairwise preference ranking.

For Naive Bayes (NB), however, the results for the pointwise and pairwise approaches are very different. Here we see that presenting the problem as a pairwise classification problem is essential for Naive Bayes being able to predict the data correctly. We suspect that this is because the simplicity of the Naive Bayes model, which is based on the probability of each feature value given the class of the instance. Oversampling the positive instances will only change the prior probabilities for the classes[20]. When presenting the data in pairs, we apply a form of bagging: Each positive answer is

[17]Note that the term pairwise describes this approach in a slightly different way than in Ranking SVM. There, the pairs are used only for parameter optimization while the scored instances are individual answers. Here, the scored instances are pairs while optimization is done instance by instance rather than pairwise.

[18]See http://search.cpan.org/~aqumsieh/AI-Genetic-0.04

[19]The 21% of questions without a correct answer in the top 150 all have an RR of 0. MRR for the successful questions only is 0.45. This is relatively high considering the success@10 score of 56.4% because a large proportion of successful questions has a correct answer at position 1 (Success@1 for all questions including the unsuccessful questions is 24.2%).

[20]Recall Bayes' theorem: $P(A|B) = \frac{P(B|A)P(A)}{P(B)}$. For our

Table 2: Results for the pointwise, pairwise and listwise approaches in terms of MRR@150 and Success@10. An asterisk (*) on an MRR score indicates a statistically significant improvement ($P < 0.01$ according to the Wilcoxon Signed-Rank test) over the TF-IDF baseline. A dagger (†) indicates that the MRR score is not significantly lower than the MRR score of the optimum setting (0.35).

	Pointwise approach						*Pairwise approach*			
	Orig. default		*Balanced default*		*Best tuned*		*Default*		*Tuned*	
techn.	MRR	S@10	MRR	S@10	MRR	S@10	MRR	S@10	MRR	S@10
TF-IDF	0.246	45.2%	N/A	N/A	N/A	N/A	0.246	45.2%	N/A	N/A
NB	0.186	43.6%	0.200	45.7%	N/A	N/A	0.320*†	56.7%	N/A	N/A
SVC	0.100	17.2%	0.318*†	56.4%	0.327*†	55.4%	0.316*†	54.8%	0.337*†	55.4%
SVR	0.342*†	53.8%	0.328*†	57.0%	0.347*†	56.5%	0.320*†	56.5%	**0.350***	56.5
LRM	0.340*†	57.0%	0.307*	54.8%	N/A	N/A	0.307*	55.4%	N/A	N/A
RankingSVM linear kernel	N/A	N/A	N/A	N/A	N/A	N/A	0.313*†	57.0%	0.313*†	56.5%
RankingSVM RBF kernel	N/A	N/A	N/A	N/A	N/A	N/A	0.130	43.6%	0.331*†	55.4%
	Listwise approach									
	Orig. default		*Balanced default*		*Best tuned*					
Genetic Algorithm, τ	0.298*	57.8%	N/A	N/A	N/A	N/A				
Genetic Algorithm, MRR	0.320*†	56.5%	N/A	N/A	N/A	N/A				

included in the data many times, but each time as part of a different instance pair. As a result, all positive instance pairs are different from each other and the algorithm has more chances of making the right decision for one answer. Apparently, the Naive Bayes classifier depends on these multiple chances for proper classification.

Table 2 shows that for regression techniques (SVR and LRM), balancing the data by oversampling or applying a cost factor leads to slightly (not significantly) lower MRR scores. In Section 2.2, we concluded from the literature that class imbalance causes fewer problems for regression techniques than for classifiers because in the regression model, the intercept value moves the outcome of the regression function up or down towards the bias in the data. Building a regression function on data in which the positive instances have been oversampled apparently leads to overfitting.

For the listwise approach (bottom part of Table 2), we see that optimizing the Genetic Algorithm for MRR with the default hyperparameter values leads to a score that is not significantly lower than the optimum. Optimizing for Kendall τ however does not reach the optimum score. An analysis of the output of the Genetic Algorithm shows that τ is suboptimal as optimization measure if MRR is used as evaluation criterion. This is because MRR is much more sensitive for answers at rank 1 than τ.

In order to get an indication of the maximum MRR score than can be reached with our feature set, we decided to combine the output of the approaches and techniques we used. From the three approaches, we included those settings for which the MRR was not significantly worse that the optimum of 0.35, using its best-scoring settings. This was the case for eight settings. We created all possible combinations of these eight settings (we normalized the instance scores to a number between 0 and 1 and then summed them over all techniques) and evaluated the performance of each of he combinations. The maximum MRR that we achieve with a combination of settings is 0.36 (a combination of SVR pointwise, SVR pairwise and Genetic Algorithm MRR listwise). This is not significantly better than the best-scoring individual technique.

data, A represents the class (correct or incorrect) and B the feature values. P(A) is the prior on the class.

6. CONCLUSION

In this paper, we have optimized the re-ranking module of a system for *why*-question answering. The goal of this paper was to compare a number of machine learning techniques in their performance on our task: inducing a ranking from a binary classification with highly imbalanced data. We evaluated learning techniques in pointwise, pairwise and listwise approaches.

We found that with all machine learning techniques, we eventually get to an MRR score that is significantly above the TF-IDF baseline of 0.25 and not significantly lower than the best score of 0.35 (reached by Support Vector Regression in the pairwise approach).

We investigated three factors: (1) the distinction between the pointwise approach, in which candidate answers are classified individually (over all questions) and the pairwise and listwise approaches, in which the ranking within each cluster of answers is optimized; (2) the distinction between techniques based on classification and techniques based on regression; and (3) the distinction between techniques with and without hyperparameters that must be tuned.

With respect to (1), we found that we are able to obtain good results with both the pointwise and the pairwise approaches for our data. The optimum score was reached by Support Vector Regression for the pairwise representation, but some of the pointwise settings reached scores that were not significantly lower than this optimum. We expect that this is because the relevance labeling of our our data is on a binary scale, which makes classification feasible. The good results obtained with the listwise approach, implemented as a Genetic Algorithm that optimizes MRR, are probably due to the fact that this approach allows for optimizing the evaluation criterion directly. Based on these results, it would be interesting to assess the performance of other listwise techniques such as SVM-MAP and AdaRank [26] on our data. We originally did not take these techniques into consideration for our data because they expect a multi-level relevance ground truth.

With respect to (2), we found that classification and regression techniques are equally capable of learning to classify our data in a pointwise setting but only if we balance our data (by oversampling or applying a cost factor) before

presenting it to a classifier. For regression techniques, balancing is not necessary and even has a negative effect on the results. We found that transforming our problem to a pairwise classification task is a good option for curing the class imbalance. This transformation even enables Naive Bayes to classify and rank the data properly.

With respect to (3), we found that for our imbalanced data set, techniques with hyperparameters heavily depend on tuning in order to find sensible hyperparameter values. However, if we solve the class imbalance by balancing our data or presenting the problem as a pairwise classification task then the default hyperparameter values are well applicable to the data and tuning is less important. Since hyperparameter tuning is a process that takes much time and computational power, a technique without hyperparameters, or a technique for which tuning can be done easily without heavy computing, should be preferred if it reaches equal performance to techniques with (more heavy) tuning. In this respect, regression techniques seem the best option for our learning problem: Logistic Regression reaches a score very close to the optimum (MRR is 0.34) without tuning. Support Vector Regression reaches optimal performance (MRR is 0.35) with tuning.

The maximum MRR that we achieve with a combination of settings is 0.36, which is not significantly better than the best-scoring individual technique. This shows that with the current feature set, the ranking performance of our system has reached a ceiling. Given the theoretical optimum of 0.79 (if for all questions with at least one correct answer a correct answer is ranked at position 1), we can conclude that our features are sub-optimal for distinguishing correct from incorrect answers. Since we already invested much time in previous work in finding the best features for describing our data, we conclude that the problem of distinguishing correct and incorrect answers to *why*-questions is more complex than an approach based on textual (overlap) features can solve.

Part of our future work is to investigate the gap between the best MRR that we reach with our feature set (0.36) and the theoretical MRR ceiling (0.785). What are the limitations of the features used? Which aspects of *why*-questions and their answers cannot be described with a set of textual features? What type of information can complement the description of question-answer pairs? In automatically answering complex questions such as *why*-questions, human reasoning and world knowledge seem to play an important role. These problems are the focus of our work in the near future.

7. REFERENCES

[1] R. Akbani, S. Kwek, and N. Japkowicz. Applying support vector machines to imbalanced datasets. *Lecture Notes in Computer Science*, 3201:39–50, 2004.

[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] Z. Cao, T. Qin, T. Liu, M. Tsai, and H. Li. Learning to rank: from pairwise approach to listwise approach. In *Proceedings of the 24th international conference on Machine learning*, pages 129–136. ACM New York, NY, USA, 2007.

[4] 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.

[5] D. Cossock and T. Zhang. Subset ranking using regression. *Lecture Notes in Computer Science*, 4005:605, 2006.

[6] L. Denoyer and P. Gallinari. The Wikipedia XML corpus. *ACM SIGIR Forum*, 40(1):64–69, 2006.

[7] 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.

[8] J. Furnkranz and E. Hullermeier. Pairwise Preference Learning and Ranking. *Lecture Notes in Computer Science*, pages 145–156, 2003.

[9] R. Herbrich, T. Graepel, and K. Obermayer. Large margin rank boundaries for ordinal regression. In *KDD'02: Proceedings of the eighth ACM SIGKDD international conference on Knowledge Discovery and Data Mining*, pages 133–142. ACM, 2002.

[10] E. Hovy, U. Hermjakob, and D. Ravichandran. A Question/Answer Typology with Surface Text Patterns. In *Proceedings of HLT 2002*, San Diego, CA, 2002.

[11] N. Japkowicz and S. Stephen. The class imbalance problem: A systematic study. *Intelligent Data Analysis*, 6(5):429–449, 2002.

[12] T. Joachims. Optimizing search engines using clickthrough data. In *Proceedings of ACM SIGKDD 2002*, pages 133–142. ACM, 2002.

[13] M. Khalid and S. Verberne. Passage Retrieval for Question Answering using Sliding Windows. In *Proceedings of COLING 2008, Workshop IR4QA*, 2008.

[14] 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.

[15] A. Owen. Infinitely imbalanced logistic regression. *The Journal of Machine Learning Research*, 8:761–773, 2007.

[16] 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.

[17] L. Shen and A. Joshi. Ranking and reranking with perceptron. *Machine Learning*, 60(1):73–96, 2005.

[18] Y. Tang, Y. Zhang, N. Chawla, and S. Krasser. SVMs Modeling for Highly Imbalanced Classification. *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, 39(1):281–288, 2009.

[19] J. Tiedemann. A Comparison of Genetic Algorithms for Optimizing Linguistically Informed IR in Question Answering. *Lecture Notes in Computer Science*, 4733:398, 2007.

[20] A. Trotman. An Artificial Intelligence Approach to Information Retrieval. *Proceedings of the SIGIR 2004 Doctorial Consortium*, page 603, 2004.

[21] N. Usunier, M. Amini, and P. Gallinari. Boosting weak ranking functions to enhance passage retrieval for Question Answering. In *SIGIR 2004 workshop on Information Retrieval for Question Answering*, pages 1–6, 2004.

[22] 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.

[23] S. Verberne, L. Boves, N. Oostdijk, and P. Coppen. Using Syntactic Information for Improving Why-Question Answering. In *Proceedings of COLING 2008*, 2008.

[24] S. Verberne, L. Boves, N. Oostdijk, and P. Coppen. What is not in the Bag of Words for *Why*-QA? . 2009. In revision for *Computational Linguistics*.

[25] E. Voorhees and D. Tice. Building a question answering test collection. In *Proceedings of the 23rd annual international ACM SIGIR conference on Research and development in information retrieval*, pages 200–207. ACM New York, NY, USA, 2000.

[26] F. Xia, T. Liu, J. Wang, W. Zhang, and H. Li. Listwise approach to learning to rank: theory and algorithm. In *Proceedings of the 25th international conference on Machine learning*, pages 1192–1199. ACM New York, NY, USA, 2008.