# Boosting weak ranking functions to enhance passage retrieval for Question Answering

Nicolas Usunier, Massih R. Amini, Patrick Gallinari
Department of Computer Science - Paris 6
University of Pierre and Marie Curie
8, rue du capitaine Scott
75015 Paris, France
{usunier, amini, gallinari}@poleia.lip6.fr

## ABSTRACT

We investigate the problem of passage retrieval for Question Answering (QA) systems. We adopt a machine learning approach and apply to QA a boosting algorithm initially proposed for ranking a set of objects by combining baseline ranking functions. The system operates in two steps. For a given question, it first retrieves passages using a conventional search engine and assigns each passage a series of scores. It then ranks the returned passages using a weighted feature combination. Weights express the feature importance for ranking and are learned to maximize the number of top ranked relevant passages over a training set. We empirically show that using questions from the `TREC-11` question/answering track and the `Aquaint` collection, the proposed algorithm significantly increases both coverage and precision with respect to a conventional IR system.

## Keywords

Passage retrieval in QA, machine learning, boosting weak ranking functions.

## 1. INTRODUCTION

Question Answering (QA) systems perform *passage* or *document* retrieval as a preliminary stage before answer extraction. The retrieval components thus play a major role in QA since they condition the performance of the whole system. Although most evaluations of QA systems do focus on end-to-end performance, some authors have started to evaluate the importance of this preliminary retrieval step [11]. Although we still lack an in-depth analysis of the role and importance of the different operations involved in this retrieval step, some important issues have already been pointed out by different authors. To quote a few of them, let us mention query formulation, boolean vs ranked document retrieval, interaction between passage and document retrieval, etc. Most retrieval systems use some form of passage retrieval, where passages may correspond to paragraph-like units, fixed size chunks of text or sentences. Deciding which type or size of passage is the best is not a trivial issue since it depends on the query formulation, on the search engine and on the extraction strategy.

Nevertheless, it is generally agreed that, at least for factoid questions, retrieving short text extracts and increasing the coverage[1] of passage retrieval engines make easier the extraction step and increases the end-to-end performance. However, experiments reported in [5] show that the overall system performance may decrease even if the coverage increases in the case where too much passages are retrieved. Their conclusion was that a high number of retrieved passages introduces noise in the answer selection process. This suggests that it is important to get high coverage together with a high precision, that is to have all answer-containing passages ranked high in the retrieved list.

A natural and widely used solution for improving passage ranking is to use a two-pass strategy by first selecting a large number $N$ of passages (sometimes selecting only substrings of passages) using a search engine, and second rank or re-rank these passages using a sophisticated ranking strategy, keeping only the $n$ top ranked passages, with $n < N$ (e.g. [8, 1, 6]). The task of such an IR component can then be expressed as maximizing $a@n$ when $n$ is small[2] [5].

We will focus here on passage re-ranking with the objective of maximizing $a@n$ for small $n$. We propose a machine learning (ML) approach to this problem. This approach proceeds in two main steps. First, for a given question and a list of associated retrieved passages, the system associates each passage with a series of local scoring features. These scoring features measure the passage relevance for the query with respect to different query characteristics. Second, these scoring measures are used as input features for a machine learning algorithm to provide a global score and rank the passages. The aim here is to provide the best possible ranking.

The ML algorithm we use here is the *RankBoost* algorithm proposed by [3] for ranking sets of objects. We show experimentally that the proposed approach increases significantly the $a@n$ performance, especially when using a high number of ranking features, which would make the hand-tuning of parameters intractable.

The machine learning of ranking functions is a difficult task. In particular, standard machine learning models are inapplicable in this case. Indeed, a possible ML approach to this ranking problem would be to rephrase it as a classification problem. However this is usually not adequate. One reason is that training data sets may be highly unbalanced (questions in `TREC-11` Q/A track have a mean number of 4 passages containing the answer).

The paper is organized as follows. We first describe in section 2 the generation of ranking features associated to each retrieved pas-

[1] *coverage* is the number of questions the answer of which is contained in the retrieved passages.

[2] Following the definition given in [9], we note $a@n$ (answer-at-$n$) the coverage of the retrieval component for the first $n$ passages.

sage. We then detail the proposed ranking model in section 3 and present an evaluation of the ranking method on the TREC-11 question/answering track using the Aquaint collection in section 4.

## 2. RANKING FEATURES

The method is generic in the sense that it can be applied to any ranking features. Moreover, the method can be used on any document collection, but re-training may be needed to adapt the ranking function to a new corpus. However, it is not a weakness of the approach, since a specific tuning of the parameters may also be needed for hand-crafted combinations depending on the document collection.

The ranking algorithm presented here takes as input a series of local scores for each retrieved passage and outputs a global ranking score. Following [3], we will denote by *ranking features* the functions assigning these local scores. We used an original feature generation method to produce a large number of ranking features and let the ML algorithm learn how to combine these features in order to optimize the final ranking. This is different from most approaches for passage retrieval in QA which rely on a careful manual feature selection.

Feature generation proceeds as follows. Given a question, we first perform an *initial retrieval* of the top $N$ passages using a search engine (section 2.2). A large number of ranking features is then generated for these $N$ passages (section 2.1). These ranking features are then combined with the aim to produce a highly accurate ranking function.

### 2.1 Ranking features generation

Our feature generation process is based on *(a)* WordNet [2]'s links and *(b)* question word importance. In order to take into account some of the WordNet 2.0's links such as: synonyms, gloss words, hypernyms and derivationally related forms, we first created different *word expansion rules*. For rules that may produce a large number of terms, we imposed a maximum number of output words using an *idf*-like scoring function.

For each question, we created *word categories* as the intersection or the union of initial question word categories provided by the QALC question analyzer [1]. These initial categories are syntactic or morpho-syntactic groups which include the question focus [3], main verb, and the question nouns, proper nouns, verbs.

Our automatic ranking feature generation is based on Query Formulation Rules (QFR). A QFR is a rule which generates a query for any given question. It will be represented here as a list of tuples *(word expansion rule, question word category)*. For any question, each QFR will return a query composed of words obtained by applying the word expansion rules to all words in the question word category list. For example, the QFR *[(identity, focus), (synonyms, main verb)]* will return for a given question a query composed of its focus and the synonyms of the main verb. Then, using both word expansion rules and question word categories, we generated QFRs by forming all possible lists of tuples *(word expansion rule, question word category)* and using some restriction heuristics in order to limit their number. We thus created more than 1000 QFRs. We then used these QFRs with the first 100 questions in TREC-11, and compared their $a@n$ performance on this question set for $n = 10, 20, ..., 100$. We created heurtistic selection rules corresponding to the minimal performance that a QFR must present. 112 QFRs matched these minimal performance rules and were consequently selected. From these 112 QFRs we obtained 112 ranking features using the following mapping.

Each QFR $\mathcal{R} : \{questions\} \longmapsto \{queries\}$ is associated with a ranking feature $f_{\mathcal{R}} : \{questions\} \times \{passages\} \longmapsto \mathbb{R} \cup \{\perp\}$:

$$f_{\mathcal{R}}(q,p) = \begin{cases} \text{score}(\mathcal{R}(q),p) \text{ if } p \text{ in the top } N \text{ passages retrieved} \\ \perp \text{ otherwise} \end{cases}$$

where $\perp$ indicates that no score is computed for $p$ by $f$. The scoring function we used in our experiments is the no-normalized cosine measure of the MG search engine [14].

To this set of 112 features, were added 5 QA specific ranking features inspired from IBM's QA system [6] and evaluated as the best performing ones in the comparative study conducted by [11]. The first 4 features are based on *matching words measure*, *mis-match words measure*, *dispersion measure* and *cluster words measure* [6]. We added a fifth feature denoted here as *ISumDF*. For a given passage $p$ and a question $q$, this measure returns $\log(\frac{\mathcal{N}}{\sum_{w \in q \cap p} occ(w)} + 1)$, where $\mathcal{N}$ is the total number of passages in the collection, $occ(w)$ the number of passages in the collection containing word $w$ and $q \cap p$ the set of words in common in $q$ and $p$.

### 2.2 Initial Retrieval

For each question, an initial list of $N$ passages is retrieved. The query sent to the search engine to compute this list was generated using the QFR that performed the best ranking on the first 100 questions of TREC-11. Using this query, the search engine returns a ranked list of passages. We used this ranking as a baseline to compare with our learning system (section 4).

## 3. BOOSTING FOR PASSAGE RANKING IN QA SYSTEMS

Boosting is one of the key learning ideas from the 90s. It was initially introduced for classification [10] in the framework of PAC learning [12] and has been extended since that to other learning problems. The initial motivation for bi-class classification problems was to combine the output of several "weak classifiers", i.e. classifiers whose error rate is better than random, to produce a global classifier with high performance. Boosting algorithms use a sequence of weak classifiers trained on different distributions over the training set. The most popular boosting method is probably *AdaBoost* [4]. Starting from an initial uniform distribution $\mathcal{D}^{Train}$ over the training set, *AdaBoost* iteratively learns a weighted sum of classifiers by adaptively modifying $\mathcal{D}^{Train}$ to assign high probability to misclassified examples. More precisely, each iteration $t$ is composed of two steps. The first one produces a distribution $\mathcal{D}_t^{Train}$ that gives more importance to the training examples that are misclassified by the weighted sum of the classifiers learned so far. The second step learns a new classifier and its associated weight w.r.t. $\mathcal{D}_t^{Train}$. The algorithm thus works by iteratively learning a classifier that corrects errors made by the weighted sum of the classifiers previously learned.

Recently, Freund et al. proposed the *RankBoost* algorithm [3], a boosting method for ranking sets of objects and they applied it to the *collaborative filtering* task and to *web searching*. *RankBoost*

[3] In the QALC system, the focus is the nominal group which represents the main object of the question. It is then a very relevant feature of the question. For example, the focus of question TREC #1394: *In what country did the game of croquet originate ?* is game of croquet.

has been designed for learning to rank examples by combining sets of basic ranking functions computed on each example. Each ranking feature will produce a ranked list of the examples. In this framework, ties are allowed and all examples need not be ordered by a ranking feature. The final ranking is a linear combination of these ranking features. Assume that for all example pairs, one knows which example should be ranked above the other one. The learning criterion to be minimized in *RankBoost* is the number of example pairs whose relative ranking as computed by the final combination is incorrect. As for *AdaBoost*, *RankBoost* works by iteratively reweighting samples and training a weak learner using this distribution. There are two main conceptual differences with *AdaBoost*: first the distribution is defined over pairs of examples, giving more weights to those with an imperfect relative ranking. Second, each weak learner is a simple binary decision function associated to one of the ranking features. Instead of learning a weak classifier at each round, like in *AdaBoost*, this algorithm simply selects one of the weak rules with respect to the current distribution.

## 3.1 Weak rules

The final output of the algorithm is a weighted sum of *weak rules*. [3] propose to use *weak rules* that assign binary scores (0 or 1) to passages. Each rule $h$ is automatically derived from a given ranking feature. Given a feature $f$ and a threshold $\theta$, $h$ is defined as follows:

$$\forall p, h(p) \quad = \quad \begin{cases} 1 \text{ if } f(p) \geq \theta \\ 0 \text{ if } f(p) < \theta \\ nd \text{ if } f(p) = \perp \end{cases}$$

where $nd \in \{0, 1\}$. To passages $p$ left unranked by $f$ (i.e. $f(p) = \perp$), the rule assigns the default score $nd$. Practically, a fixed number $N_\theta$ of thresholds $\theta$ is given.

The reason for using simple rules like the one above is computational efficiency. With the above rules, computational time of the *weak learner* step is linear in the number of *weak rules* whereas using the initial ranking features would lead to a complexity linear in the number of ranking features multiplied by the number of passages $N_{passages}$. The number of *weak rules* we can derive is $N_\theta * 2 * N_{features}$ where $N_{features}$ denotes the number of available ranking features. In practice, $N_\theta << N_{passages}$.

## 3.2 Rankboost for passage ranking

The goal is to rank answer bearing passages over the others for a given question in the training set.

Formally, let us denote by $\mathcal{Q}$, the set of questions in the training set and $\mathcal{P}_q^1$, $\mathcal{P}_q^0$, respectively the set of retrieved passages containing or not the answer for $q$. At each round $t$, the proposed approach maintains a distribution $D_t$ over the pairs of passages in $\mathcal{P} = \bigcup_q \mathcal{P}_q^0 \times \mathcal{P}_q^1$. A high weight assigned to a pair indicates that the current ranking function ranks the answer-containing passage below the non-relevant one. The algorithm starts with an initial uniform distribution over all pairs in $\mathcal{P}$. At each round $t$, the *weak learner* selects a weak rule $h_t$ and its associated weight $\alpha_t$. This weak rule is the one that minimizes the probability of misordering with respect to $D_t$. The distribution $D_t$ is updated using $h_t$ and weight $\alpha_t$ for all pair $(p_q^0, p_q^1) \in \mathcal{P}$ as follows:

$$D_{t+1}(p_q^0, p_q^1) = \frac{D_t(p_q^0, p_q^1) \exp(\alpha_t(h_t(p_q^0) - h_t(p_q^1)))}{Z_t} \qquad (1)$$

where, $Z_t$ is a normalization factor chosen so that $D_{t+1}$ will be a distribution: $Z_t = \sum_q \sum_{p_q^0, p_q^1} D_t(p_q^0, p_q^1) \exp(\alpha_t(h_t(p_q^0) - h_t(p_q^1)))$.

For a weight $\alpha_t > 0$, if $h_t$ ranks $p_q^1$ below $p_q^0$ then the new distribution value of the pair increases; otherwise it decreases. The final ranking function is $H = \sum_t \alpha_t h_t$.

The goal of the algorithm is to minimize the probability of misordering of the final ranking function $H$. The proposed training criterion called the *ranking loss - Rloss* is formally defined as follows:

$$Rloss = \sum_q \sum_{p_q^0, p_q^1} D(p_q^0, p_q^1)[[H(p_q^0) \geq H(p_q^1)]] \qquad (2)$$

Where $D$ is the initial distribution over all pairs of passages and $[[pr]]$ is a binary function defined to be 1 iff predicate $pr$ holds. This loss function simply counts the number of example pairs whose relative ordering is incorrect.

It can be shown that the $Rloss$ function is upper bounded by:

$$Rloss \leq \prod_t Z_t$$

At each round $t$, the choice of $h_t$ and $\alpha_t$ is made so that $Z_t \leq 1$. As a consequence, this upper bound on $Rloss$ decreases at each iteration. Such a choice is always possible. The optimization of $Rloss$ is then carried out by minimizing its upper bound $\prod_t Z_t$ or equivalently by minimizing $Z_t$ at each round. The algorithm is detailed below.

**Algorithm 1** The boosting algorithm for passage retrieval in Q/A

- Define the set of passages returned by the search engine for a each question $q$, and initialize $\nu_1^q$ et $\nu_0^q$ as well as the probability distribution over all questions in the training set:

$$\nu_1^q(p_q) \quad = \quad \begin{cases} 1/|\mathcal{P}_q^1| \text{ if } p_q \in \mathcal{P}_q^1 \\ 1/|\mathcal{P}_q^0| \text{ if } p_q \in \mathcal{P}_q^0 \end{cases}$$

$$a_1^q \quad = \quad \frac{1}{|\mathcal{Q}|}, \forall q \in \mathcal{Q}$$

For $t = 1, ..., T$

- Train a weak learner $h_t$ using $D_t$
- Compute $\alpha_t$ which minimizes the number of misorderings.
- Update $D_t$ : $\forall (p_q^0, p_q^1) \in \mathcal{P}_q^0 \times \mathcal{P}_q^1 \qquad D_{t+1}(p_q^0, p_q^1) = a_{t+1}^q \nu_{t+1}^q(p_q^0) \nu_{t+1}^q(p_q^1)$

$$\forall q, a_{t+1}^q \quad = \quad \frac{a_t^q Z_t^{0q} Z_t^{1q}}{Z_t}$$

$$\nu_{t+1}^q(p_q) \quad = \quad \begin{cases} \frac{\nu_t^q(p_q) \exp(-\alpha_t h_t(p_q))}{Z_t^{1q}} \text{ if } p_q \in \mathcal{P}_q^1 \\ \frac{\nu_t^q(p_q) \exp(\alpha_t h_t(p_q))}{Z_t^{0q}} \text{ if } p_q \in \mathcal{P}_q^0 \end{cases}$$

where, $Z_t^{0q}, Z_t^{1q}$ and $Z_t$ are:

$$Z_t^{0q} \quad = \quad \sum_{p_q \in \mathcal{P}_q^0} \nu_t^q(p_q) \exp(\alpha_t h_t(p_q))$$

$$Z_t^{1q} \quad = \quad \sum_{p_q \in \mathcal{P}_q^1} \nu_t^q(p_q) \exp(-\alpha_t h_t(p_q))$$

$$Z_t \quad = \quad \sum_{q \in \mathcal{Q}} a_t^q Z_t^{0q} Z_t^{1q}$$

Output the scorer $H = \sum_{t=1}^T \alpha_t h_t$

### 3.2.1 Weak learner

The *weak learner* we used in our experiments is the one presented in [3]. We found that their algorithm has a low time-space complexity though other simple algorithms may be easily implemented.

In [3] it is proved that selecting a *weak rule* $h_t$ which minimizes $Z_t$ can be carried out by selecting $h_t$ which maximizes $|R_t|$ where $R_t = \sum_q \sum_{p_q^0, p_q^1} (h_t(p_q^1) - h_t(p_q^0)) D_t(p_q^0, p_q^1)$.

This optimization leads to the following update rule for weight $\alpha_t$:

$$\alpha_t = \frac{1}{2} \log \frac{1 + R_t}{1 - R_t}$$

This choice for $\alpha_t$ and $h_t$ guarantees that $Z_t \leq 1$

### 3.2.2 The learning algorithm

Computing the distribution $D_t$ at each round is of quadratic complexity in the number of example pairs. [3] proposed a linear complexity method by replacing this distribution in the product space by the product of distributions in the sample space. Here, we need to optimize the ranker for different questions, i.e. for different lists of retrieved passages. The lists themselves are independent one from the other. We used here the following definition for $D_t$:

$$\forall (p, p') \in \mathcal{P},\ D_t(p, p') = \sum_{q \in \mathcal{Q}} a_t^q \nu_t^q(p) \nu_t^q(p')$$

Where $\forall q, \nu_t^q$ is a weighting function over the passages retrieved for $q$. This is a straightforward extension of the algorithm described in [3] for a single list of examples. The update rules of these functions $\nu_t^q$ and $a_t^q$ are given in algorithm 1.

## 4. EXPERIMENTS AND RESULTS

In our experiments we used the `TREC-11` question/answer set and the `Aquaint` document collection. The set of questions was split into a *training set* of 250 questions. The remaining 250 questions were used to evaluate the proposed ranking system. Passage retrieval was performed as in [1]: documents of the `Aquaint` corpus were cut every 20 lines with an overlap of 10 lines between two consecutive passages. The *Managing Gigabyte* (MG) [14] search engine was used for finding passages relevant to a given query.

These experiments aim at evaluating the ability to rank the passages which contain the answer above the ones which do not. For this reason, in our evaluation, the questions which had no answer in the top $N$ passages retrieved by MG were discarded from both the *training* and the *test* set. All in all, we kept 151 and 156 questions respectively in the *training* and the *test* set. For each question, the retrieved passages were then converted into 117 dimensional ranking score vectors. The value $-1$ was used to represent missing scores (remember a passage needs not be ranked by all features).

The proposed boosting algorithm was compared with two other systems - the MG search engine, which provides an initial ranking of retrieved passages and a linear support vector machine (SVM) [13]. The latter system is widely used in various learning tasks for solving two-class classification problems [7]. A 2-class SVM was trained to classify passages as relevant or not, using as input the same 117 ranking features as the boosting method. Relevant passages are those containing the answer to the question. The SVM is a classifier which learns a separating hyperplane between the examples of the 2 classes in the feature space. Each of the two obtained half-spaces is associated with a class label. Every passage belongs to one of the two half-spaces, and is then given the corresponding class label. The distance of the passage to the separating hyperplane, also called the *margin*, represents the confidence of the SVM that this passage belongs to the corresponding class. Thus, a natural way to rank passages with the SVM is to score them with respect to their distance to the separating hyperplane. This ranking method with the SVM was the one we used.

We used in all experiments, the *strict* evaluation criterion [11] which states that a passage $p$ contains the answer to a question $q$ if

- a substring of $p$ matches an answer pattern given by NIST for $q$ and
- the passage is associated to a document which was assessed to contain the answer in `NIST` judgement file.

### 4.1 Results

Figure 1 shows the *ranking loss* with respect to the number of iterations in our algorithm for the training and test sets. The *ranking loss* indicates the mean rank (as a percentage of the maximal rank) of passages containing the question answer. The figure shows that the *ranking loss* on the test set consistently decreases (from 0.4 to 0.1) in the learning stage. The boosting method behaves well: the system being trained on about 150 questions generalizes well on the 156 questions from the test set.

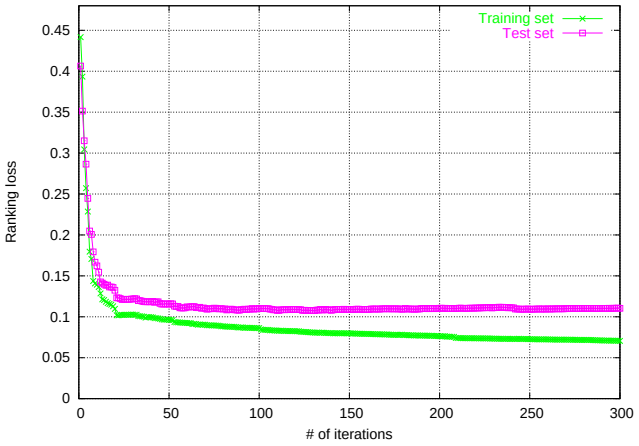


**Figure 1: Ranking loss vs rounds of boosting on the training and the test set.**

In figure 2, we show the number of times a scoring feature was selected in 300 boosting rounds. This feature selection indicates the importance of different scoring features in the final ranking function. In this plot, a $y$-value of 0 for a ranking feature means that the feature was useless in our experiments.

The last five features are the ones inspired by IBM's scoring features. The last three from this set are the most used by the boosting algorithm. They are the `mismatch` measure, the `matching` measure and our `ISumDF` measure (section 2). A rapid analysis of the other selected features shows that the `question focus head`, `the 5 question words with highest idf`, `the question proper nouns`, and `the WordNet expansion rules` *synonyms* and *hyperonyms*, on various question words categories were also among the most used features.

**Table 1: Coverage for different systems in % for 156 questions in the test set having at least one answer bearing passage in the first 500 passages returned by the search engine. For the Boosting algorithm, the coverage is given at four different iterations.**

| First $N$ passages | MG (%) | SVM (%) | Boosting (%) | | | |
|---|---|---|---|---|---|---|
| | | | IT=50 | IT=100 | IT=200 | IT=300 |
| 1 | 10.2 | 5.1 | 18.6 | 21.15 | **22.4** | **22.4** |
| 5 | 24.3 | 19.2 | 45.5 | **48.1** | 46.8 | 46.1 |
| 10 | 32.7 | 33.9 | 59.6 | **60.9** | **60.9** | 58.9 |
| 20 | 46.1 | 46.1 | **72.4** | 69.9 | 69.9 | 69.9 |
| 30 | 51.3 | 50.6 | **78.8** | 76.3 | 76.9 | 75.6 |
| 40 | 56.4 | 55.7 | 83.9 | 81.4 | 83.3 | **84.6** |
| 50 | 60.2 | 60.9 | 86.5 | 85.2 | **87.2** | **87.2** |
| 100 | 75 | 69.9 | 92.9 | 92.9 | **93.6** | 92.3 |
| 200 | 87.2 | 83.9 | **97.4** | **97.4** | **97.4** | 95.5 |
| 300 | 92.3 | 91 | **99.3** | **99.3** | **99.3** | 98.7 |
| 400 | 97.4 | 96.1 | **100** | 99.3 | 99.3 | 99.3 |
| 500 | **100** | **100** | **100** | **100** | **100** | **100** |

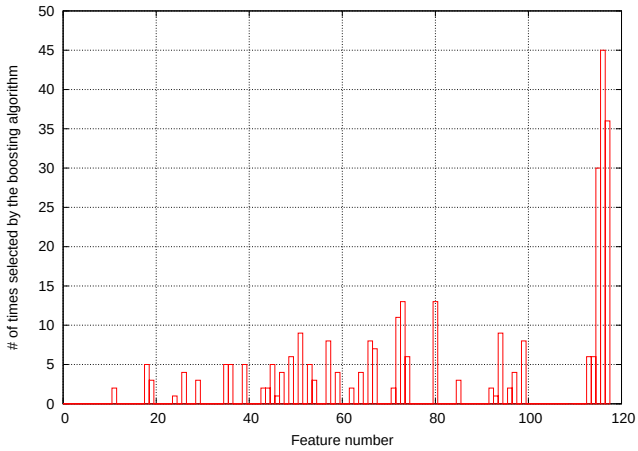


**Figure 2: Feature selection for 300 iterations of boosting.**

We also note that, features using `WordNet derivationally related forms` or `gloss words` were not selected at all.

Finally, we compare in table 1, the QA-specific measure $a@n$ performance of our algorithm with the initial ranking of the search engine MG and with the SVM classifier. SVM shows poor generalization performance, and is lower than MG ranking for almost all values of $n$. This is an interesting fact, illustrating the difficulty of applying classical classification models to the problem of ranking. Indeed standard classification models are not appropriate to highly unbalanced datasets where almost all examples are labeled to belong nearly to one class (class-label 0 in our case). The boosting algorithm performed well on the test corpus. The $a@n$ performance with $n = 5$ is comparable to the search engine's performance $a@20$. For $a@20$, the coverage is $70\%$ and it reaches $90\%$ at $n = 100$. This means for example that 109 questions in the test set have at least one answer bearing passage when keeping only the top 20 passages ranked with the algorithm. Of course, the boosting algorithm is generic and further refinement on the ranking feature would probably allow an additional performance increase.

## 5. CONCLUSION AND DISCUSSION

We have described the application of the *RankBoost* algorithm proposed by [3] to passage retrieval for Q/A systems. We have shown empirically its effectiveness on the `TREC-11` question/answering track using the `Aquaint` data collection. The algorithm allows for an important $a@n$ performance increase compared to a machine learning classifier (SVM) and to a baseline IR system. The main contribution of the paper is to provide a general framework for learning automatically the weights of different ranking functions. More sophisticated ranking functions and boosting models have still to be investigated. The model has also to be tested on other collections with heterogeneous questions and with different IR engines.

## 6. ACKNOWLEDGMENTS

This work was supported in part by the IST Programme of the European Community, under the PASCAL Network of Excellence, IST-2002-506778. This publication only reflects the authors' views. The authors are also grateful to Brigitte Grau of the Department of Computer Science in LIMSI who introduced us to Q/A and provided the QALC question analyzer [1].

## 7. REFERENCES

[1] G. de Chalendar, T. Dalmas, F. Elkateb-Gara, O. Ferret, B. Grau, M. Hurault-Plantet, G. Illouz, L. Monceaux, I. Robba, and A. Vilnat. The question answering system QALC at LIMSI, experiments in using web and wordnet. In *Proceedings of the Eleventh Text REtrieval Conference (TREC 2002)*, 2002.

[2] C. D. Fellbaum. *WordNet, an Electronic Lexical Database*. MIT Press, Cambridge MA, 1998.

[3] Y. Freund, R. Iyer, R. E. Schapire, and Y. Singer. An efficient boosting algorithm for combining preferences. *Journal of Machine Learning Research*, 4:933–969, 2003.

[4] Y. Freund and R. E. Schapire. Experiments with a new boosting algorithm. In *Proceedings of the Thirteenth International Conference of Machine Learning*, pages 148–156, 1996.

[5] R. Gaizauskas, M. A. Greenwood, M. Hepple, I. Roberts, H. Saggion, and M. Sargaison. The university of sheffield's trec 2003 Q&A experiments. In *Proceedings of the Twelfth Text REtrieval Conference (TREC 2003)*, 2003.

[6] A. Ittycheriah, M. Franz, W.-J. Zhu, A. Ratnaparkhi, and R. J. Mammone. IBM's statistical question answering

system. In *Proceedings of the Ninth Text REtrieval Conference (TREC 2000)*, 2000.

[7] T. Joachims. Text categorization with support vector machines: learning with many relevant features. In *Proceedings of the Tenth European Conference on Machine Learning*, pages 137–142, 1998.

[8] B. Katz, J. Lin, D. Loreto, W. Hildebrandt, M. Bilotti, S. Felshin, A. Fernandes, G. Marton, and F. Mora. Integrating Web-based and corpus-based techniques for question answering. In *Proceedings of the Twelfth Text REtrieval Conference (TREC 2003)*, 2003.

[9] C. Monz. Document retrieval in the context of question answering. In *Proceedings of the 25th European Conference on Information Retrieval Research (ECIR-03)*, 2003.

[10] R. E. Schapire. The strength of weak learnability. *Machine Learning*, 5(2):197–227, June 1990.

[11] S. Tellex, B. Katz, J. Lin, G. Marton, and A. Fernandes. Quantitative evaluation of passage retrieval algorithms for question answering. In *Proceedings of the 26th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 2003)*, 2003.

[12] L. G. Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, 1984.

[13] V. Vapnik. *The Nature of Statistical Learning Theory*. Springer-Verlag, New York, 1995.

[14] I. H. Witten, A. Moffat, and T. C. Bell. *Managing Gigabytes: Compressing and Indexing Documents and Images*. Morgan Kaufmann, San Fransisco, 1999.