Information Sciences 379 (2017) 313–337

Contents lists available at ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

Ranked batch-mode active learning

Thiago N.C. Cardoso, Rodrigo M. Silva, Sérgio Canuto, Mirella M. Moro, Marcos A. Gonçalves[*]

Departamento de Ciência da Computação, Universidade Federal de Minas Gerais, Brazil

ARTICLE INFO

Article history:
Received 3 February 2016
Revised 6 October 2016
Accepted 22 October 2016
Available online 28 October 2016

Keywords:
Active learning
Batch-mode
New ranking function
Paradigm change

ABSTRACT

We introduce a new paradigm for *Ranked Batch-Mode Active Learning*. It relaxes traditional Batch-Mode Active Learning (BMAL) methods by generating a query whose answer is an **optimized ranked list** of instances to be labeled, according to some quality criteria, allowing batches to be of arbitrarily large sizes. This new paradigm avoids the main problem of traditional BMAL, namely the frequent stops for manual labeling, reconciliation and model reconstruction. In this article, we formally define this problem and introduce a framework that iteratively and effectively builds the ranked list. Our experimental evaluation shows our proposed Ranked Batch approach significantly reduces the number of algorithm executions (and, consequently, the manual labeling delays) while maintaining or even improving the quality of the selected instances. In fact, when using only unlabeled data, our results are much better than those produced by pool-based batch-mode active learning methods that rely on already labeled seeds or update their models with labeled instances, with gains of up to 25% in MacroF1. Finally, our solutions are also more effective than density-sensitive active learning methods in most of the envisioned scenarios, as demonstrated by our experiments.

1. Introduction

For many applications, the volume of data generated every day is huge and impracticable to be manually read and processed by human analysts. Analyzing such huge volume of data is also impractical without automated systems. In this context, machine learning (ML) algorithms have become increasingly popular for automatically treating large volumes of data.

Supervised ML methods, which historically have produced the best results in the literature, extract important patterns derived from a dataset labeled by human experts, and apply those patterns to unseen data to perform the desired task. However, labeling the training examples is usually expensive because of the necessary expert knowledge and the time-consuming nature of the process. Moreover, a large number of labeled instances is often required in order to achieve an acceptable error rate. Finally, constant training data is also necessary to cope with changes on patterns of use.

A reduction in the effort of creating such training sets has motivated the introduction of *Active Learning* (AL) [1]. This type of technique selects and presents to the analyst/expert (also called *oracle*) instances that should be labeled first based on an estimate of the gain of information they can bring to the overall learning process. This group of instances is called

[*] Corresponding author.
E-mail addresses: thiagon@dcc.ufmg.br (T.N.C. Cardoso), rmsilva@dcc.ufmg.br (R.M. Silva), sergiodaniel@dcc.ufmg.br (S. Canuto), mirella@dcc.ufmg.br (M.M. Moro), mgoncalv@dcc.ufmg.br (M.A. Gonçalves).

http://dx.doi.org/10.1016/j.ins.2016.10.037
0020-0255/© 2016 Elsevier Inc. All rights reserved.

a *query* since it requires answers (i.e., labels) from the oracle[1]. After labeling, such instances are incorporated into the training set with an expectation of rapidly increasing the classifier effectiveness. In general, traditional AL algorithms query one instance at a time in order to update the classification model. Such a limitation can make the manual labeling routine very daunting and inadequate in scenarios with multiple experts working simultaneously in the same dataset, or when the query construction requires an expensive process implying in a long waiting time. For instance, creating a labeled set using a crowdsourcing service like Amazon Mechanical Turk[2] is impractical with such limitations. Also, such crowdsourcing services allow users to create labeling tasks that are outsourced to a large group of freelance workers; therefore, querying one instance at a time takes no advantage of such highly parallel environment.

To alleviate the aforementioned problems, a *Batch-Mode* AL (BMAL) algorithm can be used [2–4]. This class of algorithms query multiple instances at once and such instances can be independently labeled in parallel. However, BMAL by itself is still a time consuming task due to the practical problem of organizing and orchestrating the actual real-world labeling process. Specifically, the fundamental issue in using a BMAL solution lies in organizing the labeling effort efficiently. Once a batch of unlabeled samples is produced, its instances are distributed to the human analysts for labeling, which requires waiting for all of them to finish labeling the batch before these instances can be incorporated into the training set. Only then new learning models can be produced[3] and a new batch of unlabeled samples created (this process goes on and on in this interactive manner).

Moreover, the nature of the dataset/labeling task may require the design of a labeling process that provides some protection from labeling noise, such as having a single instance labeled by several analysts and then checking for inconsistencies in the labels produced by different analysts. This process alone can take many hours if not days (each analyst can tackle the task assigned to him/her within a time frame, and the essentially asynchronous nature of this process over the span of several iterations generates the biggest delays). Once *all* responses are in, we must analyze them following the agreed-upon protocol to determine the final labeling of each instance in the batch. Only then the production of another batch may begin, starting this cumbersome process all over again.

Overall, before building a new batch, important latency points do exist: (i) analyst labeling; (ii) labeling agreement; and (iii) model reconstruction (i.e., learning a new model with the newly labeled batch incorporated into training). This process will be repeated dozens or hundreds of times. The essential problem is that every batch generates a series of new *tasks*, including those to be distributed to the analysts, which must be finished (i.e., waited for) before producing new ones.

Simply increasing the batch size of a batch-mode AL method is not a good option either. Batch-mode methods will produce worse results as the batch size increases, simply because that is not how they were designed to operate. These algorithms *heavily rely* on having many iterations with small batches where the learner model is improved and adapted little by little. Hence, they will not perform well when this process is changed to the worst scenario where the batch becomes larger or in the extreme case, comprises the complete unlabeled set.

To overcome these issues, we introduce the *Ranked Batch-Mode Active Learning* (RBMAL) problem. The main idea is to relax some of the aforementioned limitations such that the query is a *ranked* list according to some quality criteria, instead of an unordered set. The problem is well connected to real world scenarios in which either analysts are paid hourly to build a training set for a given problem or the gains must be maximized on a limited budget. Our new approach allows the algorithm to generate an arbitrarily long query, thus making its execution less frequent. For example, a ranked query containing every available instance could be generated outside working hours. This would allow hired analysts to label instances for a full day, in parallel, if desired, without waiting for the learner update and query reconstruction.

Accordingly, in here we propose a method for ranking an arbitrarily large set of unlabeled instances in descending order of informativeness; that is, the order in which they should be labeled by the oracle, based on the current information provided as input and/or produced by the method at a given point of its execution. In other words, after one or more instances are labeled, the acquired knowledge can be incorporated by the method and used to generate a new, more accurate, ranking of the (still) unlabeled instances.

The main contributions of this work can be summarized as follows:

- We advocate a new and different way of thinking about the BMAL problem (what we call ranked batch-mode active learning – RBMAL), by directly optimizing the ranking of instances to be generated for labeling, instead of focusing on *iterative* small batches of instances necessary for the proper working of BMAL methods. Such rereading of the problem solves some of the drawbacks of current methods (Section 3).
- We propose a framework for solving the problem by building a ranked query even when no labeled data is provided (Section 4). This is performed by *locally and dynamically* optimizing the ranking under creation based on the information currently available on it.
- We propose a simple, yet effective solution for tackling the cold start problem (when there is no initial training data) (Section 5). 冷启动

[1] Note that this notion of query is very different from the one commonly used in Information Retrieval and similar fields.
[2] Amazon Mechanical Turk https://www.mturk.com .
[3] Notice that model reconstruction is a basic premise of BMAL solutions as new batches will be based on how much information it will bring to the updated model. This process itself can incur in considerable latency due to issues such as large datasets, complex numerical optimizations and parameter tuning.

- We perform a thorough experimental evaluation using datasets of different domains with different properties, comparing with several state-of-the-art baselines in different scenarios (Section 6).
- We *quantify* (using a $2^k r$ factorial design study) the importance of each component of our solution in the final results, including their combined contribution (Section 6.2.2). This is *rarely* performed in the AL literature.

Our experimental results demonstrate that the proposed method selects instances with results superior (up to 25%) to pool-based batch-mode active learning algorithms, i.e., the proposed method can be a drop-in replacement for batch-mode methods without their limitations. Our framework also achieves superior effectiveness when compared to those methods with only one ranking built; that is, the initial ranking generated without any labeled data has similar performance to batch-mode models iteratively updated with labeled instances. Our method is also superior to *density-sensitive* active learning methods in most of the envisioned scenarios due to its focus on building a ranking based on the current available information. 设想方案

2. Related work

There are domains in which labeled data come at little or no cost. For example, in spam classification, the user provides labels that are used to improve the spam filter by clicking the "Spam" or "Not Spam" buttons. Other scenarios include rating movies on movies recommendation applications [5] and assigning priority to emails to improve the inbox system quality [6]. However, such easy labeling is usually not available for complex systems. For example, highlighting entities (e.g., person and organization names) and relations (e.g., person works for given entity) in documents [7], investigating genes encoding one kind of enzymes [8], or grouping medical documents into classes [9] can take from 30 minutes to days. Hence, Active Learning (or AL) systems aim to overcome this labeling bottleneck as explained next.

AL is the sub-field of Machine Learning in which the learning algorithm is responsible for querying data that should be annotated by an oracle (a human analyst or an experiment). By querying *informative* instances, the algorithm learns better classification models with fewer examples, thus reducing labeling costs. Systems that use a such type of algorithms are well suited for different Machine Learning problems in which data may be abundant and easily obtained, whereas annotated data is expensive to obtain.

There are three settings for AL applications. In the *Membership Query Synthesis*, the learner may request a label for an instance in the unlabeled set or a synthesized query within the application domain (e.g. [10]). In *Stream-based Selective Sampling*, unlabeled data is drawn one at a time, in a sequential way, and the learner should decide whether or not to query the given instance (e.g. [11]). The last setting is the *Pool-Based Sampling* in which a large pool of unlabeled data is obtained at once (e.g., [12]). One variation of the latter is known as Batch-Mode Active Learning, which is the focus of our work as explained ahead. Stream-based Selective Sampling 是否可以用在线上的标注上？

AL has been extensively studied and used in several different scenarios (for a comprehensive survey, see [1]). Its main challenge is how to select the most informative instance at each iteration. For example, in most Pool-Based Sampling scenarios, the querying and labeling processes occur by selecting one instance at a time. Using traditional algorithms to query more than one instance at a time may not be enough, because instances with high informativeness in respect to a given model can be similar or even congruent. If two similar instances are given as a query to the oracle then the algorithm is not making a good use of the labeling effort. Therefore, such methods are not suitable for scenarios in which the model update is slow or multiple oracles are used in parallel. In [13], a min-max approach is used for selecting the instance that would bring more value to the objective function regardless of the label. Different from RBMAL this method selects only one instance at a time. This means that at each labeled instance a new iteration of the method should be executed.

Batch-Mode Active Learning (BMAL) avoids these issues by allowing the learner to query instances in larger groups. The challenge in BMAL is then how to assemble an optimal query set. Selecting the best instances according to a traditional method may not be optimal since it fails to consider the overlap between instances. For example, consider an unlabeled set in which there are five copies of the most informative instance. Constructing a batch of size five, only considering this information, would lead to querying the same instance multiple times. Even though multiple instances were labeled, only the information of one instance was effectively added to the model. To overcome such an issue, Brinkler [14] presents a method for incorporating diversity in batches generated by using Support Vector Machines (SVM). This is achieved by explicitly weighting the instance informativeness (closeness to SVM's decision boundary) with its similarity from instances already selected for the batch. However, this method fails to consider the dynamic nature of the process and the proper choice of each measure, which are two important aspects of the solution as we shall see. Moreover, it requires constant re-training based on the batches, a drawback that our proposal does not possess. In any case, due to its extensive use throughout the specialized literature, we use this method as one of our baselines. In [15], the objective is to select the set of instances such that the potential performance is maximized when training with this fixed set of instances. Although during this process a ranking of instances is created, it is used only for selecting the fixed batch. Different from RBMAL, uncertainty is not considered during the batch assembly either.

In [16], a batch-mode active learning algorithm for visual concept recognition is proposed that uses classification uncertainty and diversity during the batch selection. After the uncertainty is computed for each instance using a random walk, the batch is selected for labeling by minimizing an objective function. However, the method is not focused on generating an optimized ranking of instances within the batch. Moreover, it may require a large seed labeled set, due to unrealistic assumptions about the balanced nature of the seed. In most real applications, we have skewed class distributions and for generating a balanced seed for the smaller classes, one may need to label a large number of instances of the larger classes before finding representative of the smaller ones. RBMAL does not need any seed set to work.

In [17], the authors optimize a different objective function. In this method the objective is to select documents that are similar to a given user query but are also diverse. In our scenario we want to select instances that are more informative to a classifier but are also diverse (since at a given point uncertainty values are similar for similar instances).

Other solutions include a parallel approach [2,3], a clustering method to obtain diverse batches [18] and an expansion to the networked data scenario [4]. Although they allow using multiple oracles in parallel, they still require to continuously update the model and generate a new batch. In a corporate scenario in which analysts are paid hourly to label instances, continuously updating the model means that analysts' time is being wasted on waiting. Note also that methods that select a batch with size equal to the number of unlabeled instances would return the whole unlabeled set as a query. Since there is no pre-defined labeling order, the oracle would basically label instances in a *random* fashion.

To summarize, current active learning strategies have historically been deployed to rank unlabeled instances in the order they will be labeled. However, we argue that these strategies *have not been designed* to optimize a ranking of instances since they focus on one or few instances at a time. Therefore, we advocate that a ranking built in this way may be sub-optimal.

To trace a parallel comparison, this is like saying that one could use basic regression or classification techniques for *learning to rank (L2R)* tasks [19]. This is valid, but such a suggestion may not correspond to the ideal situation. It was only when researchers started to directly optimizing the ranking function (aka, the *listwise* approach [20]) or the relative order among documents (aka, the *pairwise* approach [20]) that L2R became a very important research area in information retrieval, being now employed by all major search engine companies.

Perhaps the most related approaches to the problem being described in this article are the so-called *density-based* or *density-sensitive active learning* (DSAL) methods. These methods combine an uncertainty measure, which tries to capture the informativeness of a unlabeled instance, with the estimated density around each evaluated unlabeled instance to select new samples for labeling. The rationale behind these family of methods is that one should be interested not only in samples with high uncertainty but also in samples from denser regions more often in order to avoid selecting outliers. The main reasoning is that outliers can often present high uncertainty, but their inclusion in the training set could hurt the model's effectiveness by focusing the learner on irrelevant or less informative regions of the feature space. By combining *density* with *entropy* (or uncertainty), the method should minimize the selection of outliers and focus on more informative samples. The best representatives of this family of methods include [7,21–26]. They differ from each other in the way they calculate the density and/or the entropy component and how they combine them.

Nonetheless, as defined here, RBMAL is heavily different from DSAL solutions. Specifically, the density-sensitive methods use the similarity metric to estimate the density of a particular point in the search space corresponding to the **unlabeled** training set. This strategy is based on getting more points in the denser regions of *this particular search space* in order to avoid outliers and noise. This metric is calculated only once and is static (never recomputed). In other words, this is a *global* property of the unlabeled set.

On the other hand, as we shall see, the similarity in our RBMAL solutions selects the best potential instance to be included and appended to the *ranking of instances currently being built*, based on the information available in the "in-progress" version of this ranking (as well as in any previously labeled available dataset). This information is dynamically updated at each new selection. In other words, this is a *local* property of the ranking being dynamically constructed. This is a crucial difference motivated by the focus (in a given execution of the method) on the ranking currently being generated, in order to optimize it for the (next) labeling round.

Overall, we optimize the objective ranking function with RBMAL, and such a function is substantially different from the ones exploited in batch-mode and density-sensitive active learning solutions. As we shall see in the experimental evaluation, our proposed RBMAL method is superior to the state-of-the-art BMAL and DSAL alternatives in the envisioned scenarios. Next, we formalize the problem and detail the main aspects of the proposed solution.

3. Ranked batch-mode active learning

Usually, enterprises and research groups that employ Machine Learning algorithms in their workflow have a limited budget for content labeling. Using AL can lead to a better planning of the available resources. However, it can also imply in analysts' idle time, while the learner is building the next query.

There are two main problems in the traditional Pool-Based Sampling which complicates its use in corporate scenarios: (i) the impossibility to use multiple oracles in parallel (only one instance is selected for labeling at each iteration) and (ii) the necessity of executing the algorithm for each instance to be labeled. Frequent interruptions for building new queries may compromise the efficiency of the whole labeling process, mainly when the cost of process is high, which is usually the case, as many possible candidate instances have to be evaluated.

By analysing the Batch-Mode AL strategy, one can note that it provides only partial solutions for the aforementioned problems. Since the learner is able to query multiple instances at once, it is possible to use multiple oracles in parallel. However, an uncontrolled increase of the batch size without any treatment of the query can lead to poor use of oracle resources. In other words, arbitrarily increasing the number of queried instances converges to a random strategy given that there is no relative ordering inside a batch. This is also an issue when reducing the frequency at which the algorithm is

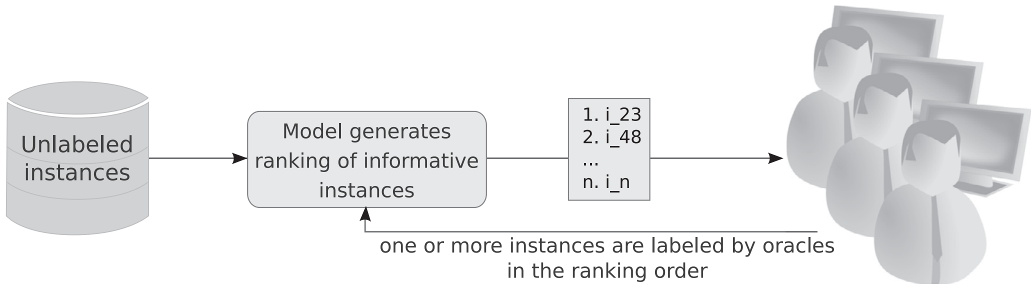

Unlabeled instances
Model generates ranking of informative instances
1. i_23
2. i_48
...
n. i_n
one or more instances are labeled by oracles in the ranking order

Fig. 1. Ranked Batch-Mode AL overview.

iterated. Although the number of times that the execution has to be interrupted for building new queries is smaller, compared to traditional Pool-Based Sampling, it may still be inconveniently frequent. For example, if the batch has 5 instances, the query has to be generated again every 5 new instances; which is a trade-off between the batch quality and its size.

In this context, our goal is to create a workflow of document labeling that is more suitable for several business models. To do so, we present a new AL strategy that generates an ordered batch of instances, which makes the labeling task more productive, more suitable for multiple oracles (in parallel) and more easily adjustable to any budget. As far as we know, this paper provides the first formalization of this problem.

The *Ranked Batch-Mode Active Learning* problem relaxes the Batch-Mode AL by generating an arbitrarily long list of instances in the order they should be labeled by the oracle. The oracle chooses the number of instances to label before the learner model is updated and, if necessary, another query is constructed. With these characteristics it is possible, for example, to execute the algorithm only once a day, generating an instance ranking that is sufficient for the analyst to work at his/her own pace. 其他的BMAL只能生成固定 size 的 batch 么？

As discussed before, in the Ranked Batch-Mode AL the learner should query instances not as a set, but as a ranked list in descending order of informativeness. The main idea is that, by labeling instances in the ranking order, the resultant classifiers maximize a given metric like *Accuracy* or *MacroF1* as soon as possible.

Fig. 1 presents the workflow of Ranked Batch-Mode AL problem. Given the set of already labeled instances and the set of unlabeled instances, the learner generates a ranking containing unlabeled instances in the order that they should be labeled. The oracle labels one or more instances that are incorporated by the learner before a new query is constructed. This process repeats while there are unlabeled instances and available resources (e.g., budget).

The Ranked Batch-Mode AL problem is formally defined ahead.

Definition 1. Let $\mathcal{D}$ be a training set containing instances $d_i = (x_i, c_i)$ where x_i is a vector containing values for each attribute and $c_i \in \mathcal{C}$ a categorical feature indicating its class. Let $\mathcal{U}$ be an unlabeled set consisting of instances $u_i = (x_i)$, that is, only the attribute values are known. The Ranked Batch-Mode Active Learning generates a ranking $\mathcal{Q}$, using information of $\mathcal{D}$, containing instances of $\mathcal{U}$ in the order that they should be labeled by the oracle, aiming to maximize the performance of a given classifier as instances in $\mathcal{Q}$ are labeled in the suggested order.

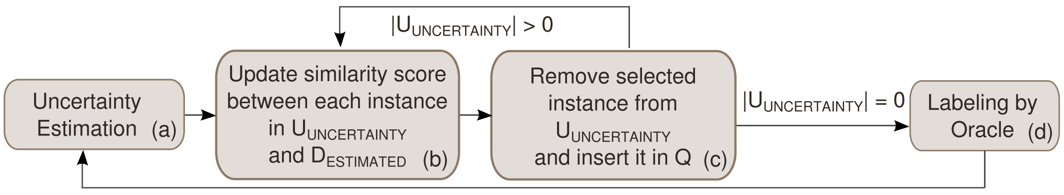
4. Our proposed ranked batch mode active learning framework

Our method aims to give more flexibility to the Batch-Mode AL scenarios by relaxing the necessity of frequently executing the algorithm. Its execution happens continuously in three steps, as discussed next.

In the first step, *uncertainty estimation*, a classifier is trained with all available labeled instances[4] and classifies each instance that should be ranked. The confidence of each classification is then used for associating an uncertainty score with its corresponding instance. This obtained information is then used in the *Ranked Batch Construction* stage in which the instances are ranked. The resultant rank is forwarded for *Labeling by an Oracle*. The oracle labels one or more instances, in the ranking order, and requests another iteration of the method. The number of instances that must be labeled is not fixed and can be adjusted. Also, at any time, the cycle can restart and incorporate the feedback provided.

Fig. 2 illustrates this process. The *Uncertainty estimation* step generates two new sets. The first one is the $\mathcal{U}_{\text{UNCERTAINTY}}$ which contains all instances in $\mathcal{U}$, along with the respective uncertainty scores, that were not yet ranked. The second one is $\mathcal{D}_{\text{ESTIMATED}}$ that represents instances already in $\mathcal{Q}$ or belonging to the labeled set $\mathcal{D}$ (which could be empty, as we shall discuss) at each iteration. In other words, $\mathcal{D}_{\text{ESTIMATED}}$, at each inner iteration, is the expected training set supposing that all instances in $\mathcal{Q}$ are labeled. Step (b) consists in updating the similarity scores of each instance in $\mathcal{U}_{\text{UNCERTAINTY}}$ with regards to expected training set $\mathcal{D}_{\text{ESTIMATED}}$. Given the calculated uncertainty (a) and similarity (b) scores, the instance that is expected to bring more information is selected in Step (c). The instance with higher score in Eq. (1) is selected, inserted in $\mathcal{Q}$ and removed from set $\mathcal{U}_{\text{UNCERTAINTY}}$. This process for ranking construction continues until $\mathcal{U}_{\text{UNCERTAINTY}}$ is empty or a pre-defined

[4] See Section 5 for a discussion about the situation in which there is no available training instances.



$|U_{\text{UNCERTAINTY}}| > 0$
Uncertainty Estimation (a)
Update similarity score between each instance in $U_{\text{UNCERTAINTY}}$ and $D_{\text{ESTIMATED}}$ (b)
Remove selected instance from $U_{\text{UNCERTAINTY}}$ and insert it in Q (c)
$|U_{\text{UNCERTAINTY}}| = 0$
Labeling by Oracle (d)

Fig. 2. Detailed overview of Ranked Batch-Mode AL Framework.

number of instances is selected.

$$\text{finalScore} = \alpha \times (1.0 - \text{similarityScore}) + (1.0 - \alpha) \times \text{uncertaintyScore}. \quad (1)$$

Note that the α parameter is responsible for weighting the impact of each factor in the final score. The main idea is to use this parameter to prioritize diversity on the initial iterations (global view of instance space) while, with the increase of the labeled content, shift the priority to instances in which the classifier is uncertain about. This is based on the assumption that the higher the uncertainty of the classifier on a given instance, the more informative this instance is to the classification process, provided that there is already some information to support this decision. This happens because instances for which the classifier is not confident usually lie in regions of class boundaries that are being defined by the process.

Finally, in Step (d) the generated ranking $\mathcal{Q}$ is presented to one or more oracles for labeling. The content labeled is then incorporated into $\mathcal{D}$, removed from $\mathcal{U}$ and the process of ranking construction restarts with the updated sets. This whole process is detailed in Algorithm 1.

Algorithm 1 Ranked Batch Active Learning algorithm.

```
Procedure RankedBatchActiveLearning
Input: A set with manually labeled instances $\mathcal{D}$
Input: A set with unlabeled instances $\mathcal{U}$
 1: $\mathcal{U}_{\text{UNCERTAINTY}}$ = UncertaintyEstimation($\mathcal{D}$, $\mathcal{U}$)
 2: $\mathcal{D}_{\text{ESTIMATED}}$ = $\mathcal{D}$\{$\mathcal{D}_{\text{ESTIMATED}}$ is the set of instances labeled or ranked\}
 3: $\mathcal{Q}$ = EmptyList()\{$\mathcal{Q}$ is the instance ranking\}
 4: for $u < |\mathcal{U}|$ do
 5:     $s$ = SelectInstance($\mathcal{D}_{\text{ESTIMATED}}$, $\mathcal{U}_{\text{UNCERTAINTY}}$)
 6:     $\mathcal{D}_{\text{ESTIMATED}} = \mathcal{D}_{\text{ESTIMATED}} \cup s$
 7:     $\mathcal{U}_{\text{UNCERTAINTY}} = \mathcal{U}_{\text{UNCERTAINTY}} - s$
 8:     InsertIntoList($\mathcal{Q}$, $s$)
 9:     $u = u + 1$
10: end for
11: $\mathcal{L}$ = WaitForOracleLabel($\mathcal{Q}$)\{$\mathcal{L}$ is the set of instances labeled in this iteration\}
12: $\mathcal{D} = \mathcal{D} \cup \mathcal{L}$
13: $\mathcal{U} = \mathcal{U} - \mathcal{L}$
14: return ($\mathcal{D}$, $\mathcal{U}$)
```

The uncertainty estimation (line 1) is the process of estimating the informativeness of a given instance and is discussed in Section 4.1. As shown in Section 2, the challenge of ranking multiple instances lies in prioritizing informative documents that present diversity with already selected ones. This is achieved by weighting (line 5) the uncertainty with a similarity score, which is further discussed in Section 4.2.

4.1. Uncertainty estimation

At each iteration, an uncertainty score is calculated for each instance in the current unlabeled set $\mathcal{U}$. This score aims to select instances in which a given classifier is not confident regarding their predictions using the current training set $\mathcal{D}$. In other words, an uncertainty score is obtained by training a classifier with the current training set $\mathcal{D}$ and processing, for each instance $u \in \mathcal{U}$, its probability of belonging to each known class. These probabilities are then further processed in order to obtain an uncertainty score. Different estimators can arise from different combinations of classifiers and uncertainty measures. These components are discussed next.

Classifiers. The classifier is the core of the uncertainty estimator. It is expected to provide the probabilities of a given instance belonging to a known class. A classifier that is a good fit for our purposes outputs probabilities that are close to the accuracy estimations. That is, if the classifier outputs a given class with 80% confidence, then it is expected that it will be correct approximately 80% of the cases.

Four different classifiers were considered as the core of the uncertainty estimator: K-Nearest Neighbors (*Knn*) [27], Gaussian Naive Bayes (*NbGauss*) [28], *Rocchio* [29] and Support Vector Machines [30]. They were chosen based on the execution speed, potential to produce reliable uncertainty scores, and the possibility to work with real-valued attributes without the necessity of discretizing its values.[5] This way we avoid the impact of discretization in the results.

In order to output a confidence or probability score, these classifiers had to be adapted. The *Knn* is used with the *Euclidean* similarity function, k parameter set as 10 (for further discussion see [31]), and classification confidence is given by the number of neighbors that belong to the ouput class divided by k. The *NbGauss* confidence is given by the probability output by the algorithm and, finally, the *Rocchio* algorithm uses the *Euclidean* distance metric and its confidence is given by the ratio of the chosen class centroid similarity to the sum of the similarities to all centroids. And in the case of SVM, we relied on a package parameterization to output the proper probability estimations.

Uncertainty score. After the probabilities are produced by the classifier, they must be converted to an uncertainty score. In this article the Least Confident uncertainty score is used. Let p_{c_i} be the probability of an instance belonging to class c_i, then the score can be formally described by $s = 1.0 - \max_i p_{c_i}$. This method was chosen due to its simplicity, the focus on the minimization of the classification error and by the fact that its behaviour is not affected by the number of classes in the dataset.

4.2. Instance ranking

After the uncertainty estimation step, the uncertainty score of every instance in $\mathcal{U}$ is calculated. This score is used in the Instance Ranking phase which is the core of the proposed framework. Instances are iteratively selected until the construction of the ranking $\mathcal{Q}$ is finished. In order to choose the most informative instance at each step, two criteria are considered: the uncertainty score of classifying the instance given the current labeled set ($\mathcal{D}$) and the similarity score between the instance and the current estimated training set ($\mathcal{D}_{\text{ESTIMATED}}$).

The similarity score between an instance and $\mathcal{D}_{\text{ESTIMATED}}$ is calculated as being equal to 1.0 minus the highest similarity between the provided document and an instance belonging to the augmented training set. The main idea is to give high scores to instances that do not share much similarity with already labeled documents. In this way, this score prioritizes instances that will help explore unknown parts of the instance space.

Let ϕ be the similarity function, $\mathcal{S}$ be the set of instances and i the current instance being compared, then the set similarity can be given by Eq. (2).

$$\text{SetSimilarity}(i, \mathcal{S}) = \max_{s \in \mathcal{S}} \phi(i, s) \tag{2}$$

The implementation of the set similarity is described by Algorithm 2.

Algorithm 2 Set Similarity.

```
Procedure SetSimilarity
Input: A set with instances $\mathcal{S}$
Input: An instance $i$ to be evaluated
1: biggestSim = 0.0
2: for all $s \in \mathcal{S}$ do
3:     sim = SimilarityFunction($i, s$)
4:     if sim > biggestSim then
5:         biggestSim = sim
6:     end if
7: end for
8: return biggestSim
```

The Similarity Function can be any function able to map a pair of entities from the instance domain ($\mathcal{I}$) to a real number v: $0 \le v \le 1$. It represents the resemblance between the instances and must hold the following properties:

1. $f : \mathcal{I} \times \mathcal{I} \longrightarrow \mathbb{R}^+$
2. Non-negativity: $\forall x, y \in \mathcal{I}, f(x, y) \ge 0$
3. Maximality: $\forall x, y \in \mathcal{I}, f(x, x) \ge f(x, y)$
4. Symmetry: $\forall x, y \in \mathcal{I}, f(x, y) = f(y, x)$

Four similarity functions were tested: (i) **Chebyshev distance** is the metric defined in a vector space in which the distance between two vectors is the greatest of their differences along any coordinate dimension; (ii) **Cosine similarity** measures the cosine of the angle between two vectors: since the resultant value can be negative, the absolute value was used in order to provide only values between 0 and 1; (iii) **Euclidean distance** is defined as the length of the line segment connecting two points in the Euclidean n-space; and (iv) **Manhattan distance** is the sum of the absolute difference of coordinates.

[5] Other classifiers that could work well in our problem (such as decision trees) require complex discretizations which may not work properly in scenarios with no or only a few labeled instances.

Some of those (Chebyshev, Euclidean and Manhattan) are actually distance metrics. The functions were adapted to hold the requirements; for instance, the complement of the normalized distance is used as a similarity function.

Every time a new instance is added to $\mathcal{Q}$, every document in $\mathcal{U}_{\text{UNCERTAINTY}}$ must have its rank calculated. Such a rank is based on a real value automatically calculated based on the uncertainty score and the current $\mathcal{D}_{\text{ESTIMATED}}$ similarity. The weight of each factor, represented by the α parameter, is used for changing the primary focus of the method according to the amount of labeled instances available. This idea comes from the intuition that when only a few instances have been labeled, it is better to explore the unknown instance space whereas when a larger training set is available, the uncertainty estimation becomes more important, potentially producing better choices.

Let Φ be the set similarity function and Υ be the function that returns the uncertainty for a given instance and train set, then the instance ranking function is given by Eq. (3).

$$q = \arg\max_{u \in \mathcal{U}_{\text{UNCERTAINTY}}} \alpha(1 - \Phi(u, \mathcal{D}_{\text{ESTIMATED}})) + (1 - \alpha)\Upsilon(u, \mathcal{D}) \tag{3}$$

In Algorithm 3, the α parameter is dynamically set based on size of the training and test sets. This way, it is possible to shift the instance prioritization from diversity to uncertainty. More formally, α is set as being the ratio between the training set size and the total available instances $\alpha = \frac{|\mathcal{U}|}{|\mathcal{D}|+|\mathcal{U}|}$. This paremeter is updated every time new labeled content is provided by the oracle.

Algorithm 3 Select Instance.

```
Procedure SelectInstance
Input: The set $\mathcal{D}_{\text{ESTIMATED}}$ that represents $\mathcal{D} \cup \mathcal{Q}$
Input: The set $\mathcal{U}_{\text{UNCERTAINTY}}$ of unlabeled instances, not in $\mathcal{Q}$,tagged with uncertainty score
Input: Current $\alpha$ parameter
1:  bestScore = $-1.0$
2:  bestInstance = $nil$
3:  for all $u \in \mathcal{U}_{\text{UNCERTAINTY}}$ do
4:    similarity = SetSimilarity($\mathcal{D}_{\text{ESTIMATED}}, u$)
5:    uncertaintyScore = UncertaintyScore($u$)
6:    score = $\alpha \times (1.0 -$ similarity$) + (1.0 - \alpha) \times$ uncertaintyScore
7:    if score > bestScore then
8:      bestScore = score
9:      bestInstance = $u$
10:   end if
11: end for
12: return bestInstance
```

4.3. Complexity analysis

RBMAL performs a number of pair-wise comparisons between instances using a similarity function, and is agnostic in regard to the uncertainty estimator. The computational complexity considering the number of operations performed between instances depends on the comparisons performed in the proposed algorithm and the cost of the chosen uncertainty estimator.

The proposed Algorithm 1 executes for each batch. It first executes the uncertainty estimator to update the uncertainty of each instance in the unlabeled pool. Considering that $\mathcal{D}$ is the set of labeled instances, and $\mathcal{U}$ is the set of unlabeled instances, $C_{UNCERTAINTY}$ represents the cost of the uncertainty estimator using the labeled instances $\mathcal{D}$ as evidence to evaluate the uncertainty of the unlabeled instances $\mathcal{U}$.

After the computation of the uncertainty, RMBAL performs pair-wise comparisons between instances using a similarity function. The Algorithm 2 is called to compare each instance with all the elements in the batch. Although this requires $O(k)$ comparisons, we are only concerned with the highest similarity. For this reason it is possible to compute only one similarity at each selected instance by storing the highest similarity so far. Being C_s the cost of computing the similarity between two instances, and k the size of the batch, then the ranking cost can be given by $C_{SELECT} = C_s * |\mathcal{U}| * k$. Considering both the uncertainty and the similarity, the total cost to generate a batch can be given by:

$$C_{TOTAL} = (C_{UNCERTAINTY} + C_{SELECT})$$

Assuming the worst case where a full ranking is being generated ($k = |\mathcal{U}|$), considering the cost to compute similarity operation between two instances as constant, and the cost $C_{UNCERTAINTY} = |\mathcal{U}| * |\mathcal{D}| + |\mathcal{D}| * \log|\mathcal{D}|$ of the kNN estimator (which computes the similarity of each unlabeled example with all labeled ones, followed by the $|\mathcal{D}| * \log|\mathcal{D}|$ operations to sort the similarities), the complexity to compute each batch is $O(|\mathcal{U}| * |\mathcal{D}| + |\mathcal{D}| * \log|\mathcal{D}| + |\mathcal{U}| * |\mathcal{U}|)$. Considering a huge number of labeled instances, i.e., $|\mathcal{U}| = |\mathcal{D}|$, the complexity to generate a batch is $O(|\mathcal{U}|^2)$. It is worth noting that in this scenario, the complexity of both $C_{UNCERTAINTY}$ and C_{SELECT} are $O(|\mathcal{U}|^2)$.

Notice that arbitrarily large rankings enable the analyst (or oracle) to continuously label instances without the need to generate a new batch. In other words, after the ranking is built, the oracle can be provided with a new instance to label with no extra cost. Notice also that this process runs offline in batch.

5. Dealing with the cold start problem

When there is no training data available, that is, $\mathcal{D} = \emptyset$, there is no way to assign uncertainty scores to instances in $\mathcal{U}$. Since the uncertainty score is undefined in this scenario, a special treatment is necessary. The problem of running the Active Learning algorithm without prior labeled information is known as the Cold Start problem.

The first instance that is selected by the algorithm plays an important role not only in the first ranking constructed but also in all subsequent generated rankings. This happens because the first selected instance can influence the direction taken by the learner in the exploration of the instance space. A bad initial selection can make the learner ignore certain regions of the instance space or, in extreme cases, completely overlook certain classes.

Consider one class composed of sub-concepts or clusters, as in a dataset comprising published articles of different domains. For example, for the "Computer Science" class there are different sub-groups like artificial intelligence, databases and image processing that have different vocabularies. The *missed cluster effect* [32] is defined as the problem faced by the Active Learning algorithm when it knows only some sub-concepts of the class, becoming overly confident about the class boundary. As a result, the algorithm focuses on exploring a given area of the instance space at the expense of missing others. This problem may become even worse, with the learner completely overlooking certain classes. This special case of the *missed cluster effect* is called *missed class effect* and it is further described by [33].

When the training set is empty, our framework assigns the same score to every instance in $\mathcal{U}_{\text{UNCERTAINTY}}$ since the similarity score is 1 (there are no instances in $\mathcal{D}_{\text{ESTIMATED}}$) and there is no uncertainty score ($\mathcal{D}$ is empty). This means that, in this setting, the initial instance would be randomly chosen. In order to make a better initial selection (that will hopefully steer the classifier search into a better region) a simple heuristic is used. The instance selected when $\mathcal{D} = \emptyset$ is the one that has the highest average similarity with instances in $\mathcal{U}$. The intuition is: the instance that shares most similarities with the dataset will help the learner to explore unexpected regions, given that in the first iterations of the algorithm the diversity is prioritized over uncertainty. Algorithm 4 presents the method used for choosing the initial seed when no initial training is provided.

Algorithm 4 Seed selection.

```
Input: ChooseSeed
Input: The set of unlabeled instances $\mathcal{U}$
 1: bestAverageSim ← −1.0
 2: bestInstance ← nil
 3: for all $u_1 \in \mathcal{U}$ do
 4:   averageSim = 0.0
 5:   for all $u_2 \in \mathcal{U}$ do
 6:     if $u_1 \neq u_2$ then
 7:       sim = SimilarityFunction($u_1, u_2$)
 8:       averageSim = averageSim + sim
 9:     end if
10:   end for
11:   averageSim = averageSim/($|\mathcal{U}| - 1$)
12:   if averageSim > bestAverageSim then
13:     bestAverageSim = averageSim
14:     bestInstance = $u_1$
15:   end if
16: end for
17: return bestInstance
```

6. Experimental evaluation

This section presents our experimental evaluation organized as follows. Section 6.1.1 describes the datasets. Section 6.1.2 and 6.1.3 present the experimental configuration and the evaluation metrics. An evaluation of potential Uncertainty Estimators and Similarity Functions is presented in Section 6.2.1. Section 6.2.2 evaluates the impact of the main components of the framework using a factorial design. Section 6.2.3 presents a discussion on the proposed strategy for solving the cold start problem. Section 6.2.4 evaluates the instance ranking generated by our method against a random ranking strategy. In Section 6.2.5 a comparison with a pool-based batch-mode baseline is presented, and finally, in Section 6.2.6 a comparison with a state-of-the-art density-sensitive active learning method is performed.

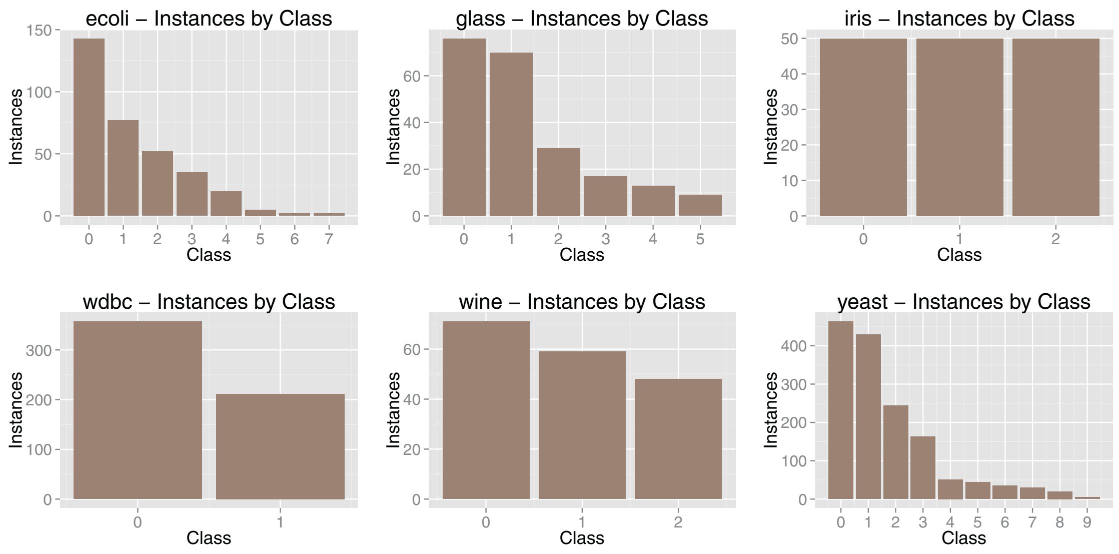


Fig. 3. UCI Datasets' instances by class.

6.1. *Experimental setup*

6.1.1. *Datasets*

We have chosen datasets of different domains from the UCI Machine Learning Repository [34], a well known repository in the Machine Learning community. They are very diverse in terms of size, types and number of classes, as well as class imbalance rates. Classes are categorical attributes and the datasets share the same feature domain (real numbers) in order to allow the use and comparison of the same *uncertainty estimators* and *Similarity Functions*.

Table 3 shows the number of instances in each class for each dataset. Note also that the class distribution in some datasets are very skewed (e.g., *ecoli*). The impact of this characteristic in the experiments will be discussed later.

6.1.2. *Experimental configuration*

To assess the quality of the output ranking, the datasets were randomly split into half multiple times. Each split pair is then used in the following way. The first set corresponds to the unlabeled set ($\mathcal{U}$) containing the instances to be ranked or selected. In some of the performed experiments, a few instances were labeled to work as an initial seed, all extracted from $\mathcal{U}$. The other set, the test one ($\mathcal{T}$), contains instances to be used only in the evaluation. Such experimental setup is common in various active learning approaches [13,14,35]. Moreover, the replication of random half/half divisions is a suggested practice for assessing statistical significances when comparing supervised classification approaches [36]. We assess the statistical significance of our results by means of a paired Wilcoxon signed rank test with 95% confidence and Holm correction to account for multiple tests.

Given that the used data is split at random, in order to guarantee statistical validity, 40 different splits are generated for each dataset and the results are presented along with confidence intervals with 95% of confidence level.

Each unlabeled and test set is evaluated as follows: the unlabeled data is fed to an AL algorithm, which selects the instances to be queried and labeled. After running one or more times, it returns an instance list of elements in $\mathcal{U}$ ranked in the order they should be labeled. This ranking is then evaluated by measuring the quality of classifying $\mathcal{T}$ using increasing portions of the ranked list. The final result of the execution is a chart of classification quality versus the number of instances in $\mathcal{Q}$ used for training.

In this work, a *Knn* classifier is used for evaluating each portion of the generated ranking, where the k parameter is selected by a cross-validation in the training set[6]. This classifier was chosen given its good performance across different domains and the possibility to work with real-valued features. Comparisons with other classifiers are left for future work. It is also important to notice that this classifier is unrelated to the one used as the *uncertainty estimator*. Before presenting the actual results, an overview of the adopted evaluation metrics is presented next.

6.1.3. *Evaluation metrics*

Let $\mathcal{Q}@x$ be the set containing the first x instances of $\mathcal{Q}$. For each list of size x the quality of the classification is estimated using $\mathcal{T}$. This is measured by training a classifier with the set $\mathcal{Q}@x$ and classifying the set $\mathcal{T}$. The resultant classification is

[6] Notice that, as the *Knn* classifier is only used to evaluate portions of the ranked set of instances, there is always training data available to estimate the k parameter.

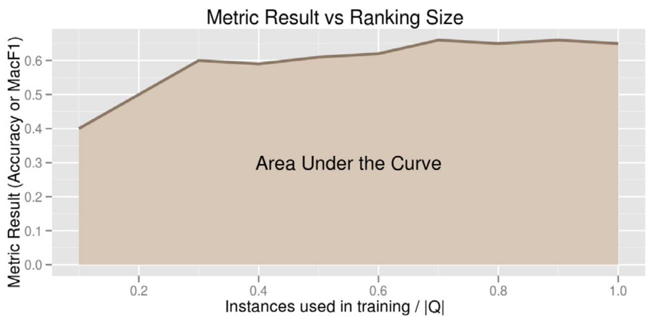


Fig. 4. Area under the curve.

evaluated by calculating its Accuracy and MacroF1 [37]. *Accuracy* and *MacF1* provide a way to measure the classification quality for a given train $\mathcal{Q}@x$ and test $\mathcal{T}$. It is necessary in our case, though, to measure the overall quality of a ranking Q.

A typical result of evaluating $\mathcal{Q}$ can be seen in Fig. 4. One way to obtain a single number for the execution is to measure the area under the metric curve (AUC). This method is used for obtaining a single number from other types of curves [38] and captures the idea of having a high value for a given metric in different parts of the curve. Since the measurements are discrete, the area is the sum of the trapezoids between each two points. Sometimes the interest is in investigating only the beginning of the curve. In such cases, the metric used is the *AUC@x*, i.e., the AUC metric calculated only considering the first *x*% of $\mathcal{Q}$.

6.2. Experimental results and discussion

6.2.1. Evaluation of uncertainty estimation and similarity function

The uncertainty estimator and the similarity function greatly impact the result of the proposed framework. These components should be adapted for specific domains and possibly complexity constraints. In this section, we present how to choose good components for a given problem and analyse the impact of such a choice. In these experiments we use the datasets described in Section 6.1.1

Notice that the choice of the best components of the solution are somewhat domain or dataset-dependent. For instance, experimenting in other domains such as text and/or image classification tasks may bring to the problem specific idiosyncrasies, such as high sparseness (for the former) and the curse of dimensionality (for the latter) that could require specific choices to reach a proper solution. We leave such an investigation for future work.

6.2.1.1. Uncertainty estimator. A good uncertainty estimator should provide a “reliable” probability for each prediction, i.e., one close to the real probability of being correct. That is, if the estimator predicts a given class c_i with probability p_i, then this instance has a real probability, close to p_i, of belonging to class c_i.

One way to visualize the probabilities of a given estimator is by plotting the ratio of correct predictions for each estimated probability. This estimator can then be evaluated by comparing this plot with the one of a perfect estimator. The probabilities output by the classifier are real-valued, then they can be separated into bins. In this work, three estimators are compared by the mean squared error (*MSE*) across each bin. The *MSE* is defined in Eq. (4).

$$MSE = \frac{\sum_{b=0}^{n-1}(e(b) - p(b))^2}{n} \tag{4}$$

where b is a bin with at least one prediction, n is the total number of bins with predictions, $e(b)$ is the correct predictions ratio of the estimator for bin b, and $p(b)$ is the correct predictions ratio of the perfect model for bin b. The larger the difference from the perfect model, the larger is the *MSE* value.

In order to compare the uncertainty estimators, the training set was split multiple times into new training and validation sets. The new training set is used to train a given estimator and the validation set is used to calculate the correct prediction ratio. The test set is used in order to verify the relation between the calculated *MSE* and the result of the proposed method execution.

As mentioned before, four classifiers are compared as uncertainty estimators: the K Nearest Neighbors using the Euclidean distance and the k parameter set to 10 (*Knn10*); Rocchio using Euclidean distance; Gaussian Naive Bayes (*NB Gauss*); and *SVM* with parameters chosen using the *libsvm* grid tool [30].

Fig. 5 presents an example of how the *MSE* is calculated. The bar plot ($e(b)$) represents the estimated probability of a correct prediction made by the classifer while the Error is the distance to the perfect model, i.e., $|e(b) - p(b)|$. In the *glass* dataset, for instance, *Knn* is the classifer that gets closer to the ideal model whereas *Rocchio* and *SVM* tend to provide probabilities that are not really meaningful at all. The tendencies in this qualitative evaluation are confirmed in most cases by the calculated *MSE* in all datasets.

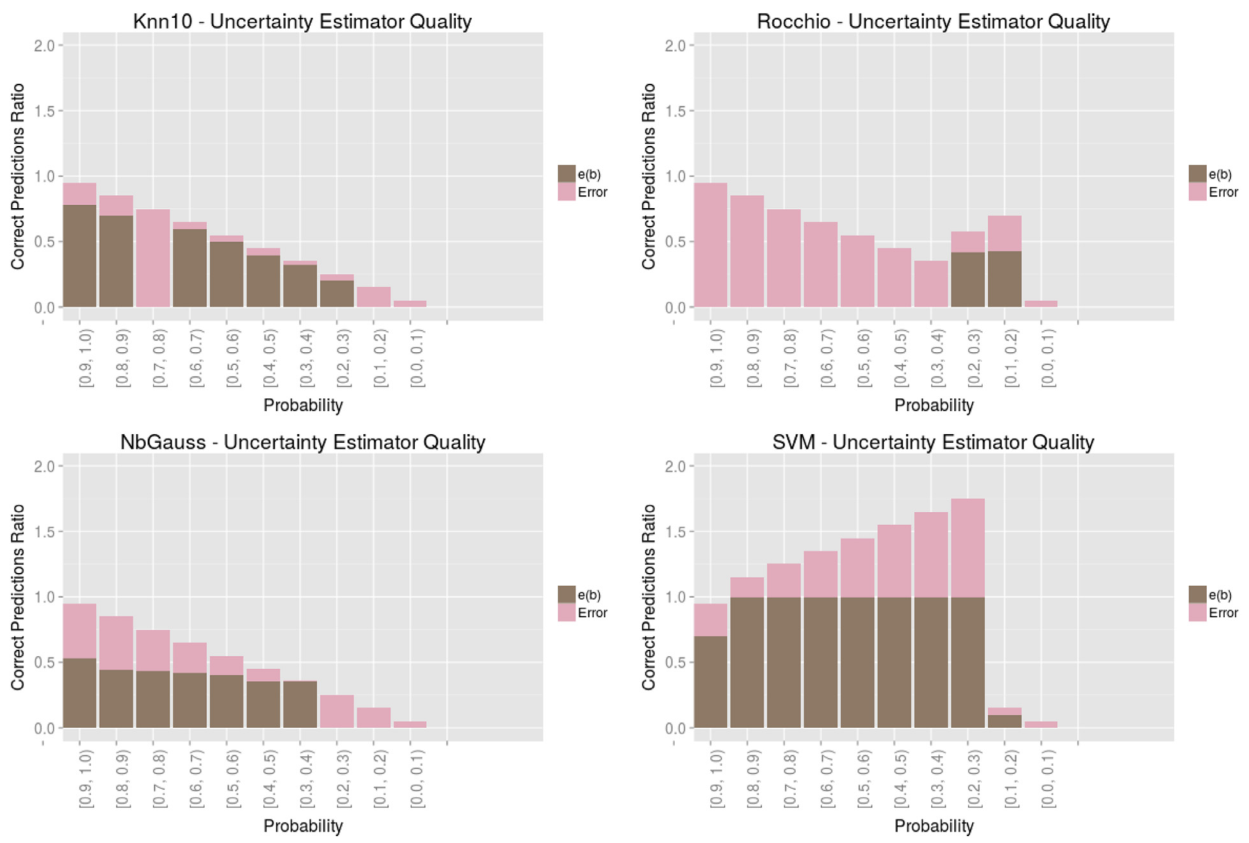


Fig. 5. Uncertainty estimator quality for the *glass* dataset.

Table 1
Mean squared Errors of uncertainty estimators.

Dataset	Knn10	Rocchio	NBGauss	SVM
ecoli	0.003184 ± 0.002294 ▲	0.371932 ± 0.001193	0.064247 ± 0.019010	0.246500 ± 0.020933
glass	0.009501 ± 0.000391 ▲	0.101414 ± 0.045563	0.075931 ± 0.000874	0.227167 ± 0.014622
iris	0.012740 ± 0.004541	0.276062 ± 0.005645	0.005315 ± 0.000343 ▲	0.136500 ± 0.016008
wdbc	0.004390 ± 0.000300 ▲	0.144769 ± 0.002631	0.045327 ± 0.001610	0.076806 ± 0.000982
wine	0.014470 ± 0.000551 ▲	0.341014 ± 0.009549	0.023330 ± 0.012135	0.162500 ± 0.000000
yeast	0.007522 ± 0.000088 ▲	0.124583 ± 0.000296	0.038520 ± 0.001047	0.269158 ± 0.000014

Table 2
AUC-MacF1 of different uncertainty estimators.

Dataset	Knn10	Rocchio	NBGauss	SVM
ecoli	58.04345 ± 2.03743 ●	54.14583 ± 1.77268	58.12891 ± 1.96621 ●	57.79719 ± 2.28984 ●
glass	46.07579 ± 1.76127 ●	44.24015 ± 2.34076 ●	36.82012 ± 1.70224	43.76789 ± 1.68037 ●
iris	91.05253 ± 1.17847 ▲	69.59344 ± 1.93440	87.14806 ± 2.25009	85.45803 ± 0.81237
wdbc	90.01343 ± 1.20737 ●	87.38219 ± 1.79822	88.92472 ± 1.24292	92.01531 ± 0.45255 ●
wine	91.00987 ± 0.80863 ●	91.66246 ± 0.80805 ●	91.32646 ± 0.96706 ●	85.67513 ± 0.88947
yeast	43.30552 ± 0.83688 ●	30.53398 ± 1.33456	39.54735 ± 0.76823	44.62263 ± 0.65529 ●

Table 1 presents the calculated *MSE* between the obtained probabilities and the perfect model along with confidence intervals with 95% of confidence. Estimators that are significantly better than the others are marked with ▲. In order to evaluate whether the *MSE* is a good measure of estimator quality, the whole method was executed assuming that the oracle would label one instance between each ranking calculation. That is, only the first instance of the ranking is used to update the model at each iteration (ideal scenario). The similarity measure used in this experiment is the *euclidean*. The *AUC-MacF1* was then calculated using the generated ranking and the test set that was originally kept apart.

Table 2 presents the *Area Under the Curve (AUC)* for the *MacF1* metric along with 95% confidence intervals. As before, estimators with statistically significant better results are marked with ▲, while statistical ties are marked with (●). As can be

Table 3
AP_{SNR} for different similarity metrics in each dataset.

Dataset	Chebyshev	Cosine	Euclidean	Manhattan
ecoli	2.88929 ± 0.05616	3.15848 ± 0.06074 ▲	3.10008 ± 0.06167	3.11050 ± 0.05862
glass	2.40065 ± 0.05808	2.50758 ± 0.05540	2.51330 ± 0.06029	2.62972 ± 0.06122 ▲
iris	5.31531 ± 0.14527	7.04575 ± 0.36477 ▲	5.72698 ± 0.16771	5.79848 ± 0.17695
wdbc	4.66559 ± 0.09158	4.35074 ± 0.05995	4.91785 ± 0.08253 ▲	4.87181 ± 0.07644
wine	4.66710 ± 0.14012	2.89081 ± 0.03390	5.92183 ± 0.17684	6.26472 ± 0.17594 ▲
yeast	2.49917 ± 0.02408	2.52745 ± 0.02368	2.56371 ± 0.02556	2.66894 ± 0.02665 ▲

seen, the *MSE* results are endorsed by the *AUC-MacF1*, being *Knn10* the best uncertainty estimator in most datasets. *SVM*'s *MSE* results did not seem to impact in the final *AUC-MacF1*. This can happen if the classifier generates poor probabilistic estimates but is able to provide other types of reliable uncertainty estimations, such as distance to the decision border.

Two datasets, though, are outliers in this experiment. The first one is the *iris* dataset, in which the *NbGauss* provides the smallest *MSE* but not the best *AUC-MacF1*. This may be explained by the fact that, although the *NbGauss MSE* is systematically smaller than the *Knn10*'s, these two values are very close. The other interesting case occurs in the *wine* dataset. Although there are big differences on the estimators' *MSE*s for all classifiers, the resultant *AUC-MacF1*s are equivalent with 95% confidence. This can happen in datasets in which the random baseline is strong or the used metric (e.g., *MacF1*) stabilizes very quickly. In any case, due to its small average MSE, throughout this work, we will use Knn10 as the uncertainty estimator.

6.2.1.2. Similarity function. The other key component of the proposed method is the similarity function for comparing a given instance with the currently ranked instances. A good similarity metric is one that returns a high similarity score for instances in the same class and a low similarity score for instances in different classes. One way to measure the quality of a given similarity function is by using the Mean Average Precision (*MAP*) [39] in different datasets.

Average Precision provides ranking evaluations by calculating the average of the precision value obtained after retrieving each relevant document. Let R be the set of relevant documents $\{d_1, d_2 \ldots d_m\}$ and n_i the total number of documents retrieved until document d_i. Then the Average Precision (AP) is defined by Eq. (5).

$$AP = \frac{\sum_{i=1}^{m} p_i}{m} \tag{5}$$

where p_i is defined by Eq. (6).

$$p_i = \frac{i}{n_i} \tag{6}$$

In order to use *AP* to evaluate similarity metrics, one can consider the ranking created by ordering the dataset instances in descending order of similarity given a reference instance (query). Instances belonging to the same class as the query can be considered relevant. Finally, the *MAP* is the mean of the *AP* calculated for each of the dataset's instances [40].

Although *MAP* provides a way to compute the average quality of the similarity metric for different instances of the dataset, it fails to depict how consistent these results are across different queries, allowing outliers to impact the final result. A good similarity function should provide a more reliable *MAP* value for queries of all classes, thus a signal-to-noise ratio ($\text{SNR} = \mu/\sigma$) of the *AP* is used. Let $\mu(AP)$ be the mean and $\sigma(AP)$ be the standard deviation of the *AP* values of all queries, then Eq. (7) presents the formula for the AP_{SNR}.

$$AP_{\text{SNR}} = \frac{\mu(AP)}{\sigma(AP)} = \frac{MAP}{\sigma(AP)} \tag{7}$$

In this work, four similarity metrics are compared: Chebyshev, Cosine, Euclidean and Manhattan. In this set of experiments, the uncertainty estimator is fixed as the *Knn10*, no seed for training is provided ($\mathcal{D}$ starts empty), and it is considered that the oracle labels one instance at each iteration. This labeling process allows the α parameter to distribute its weight equally between the uncertainty estimator and the similarity function. Thus, we can assume that each component has an equivalent contribution to the final result. The metric is calculated in each train/test split so averages and confidence intervals can be provided.

Table 3 presents the results of the AP_{SNR} for different similarity metrics as well as the confidence intervals with 95% of confidence. Table 4, on the other hand, presents *AUC-MacF1* along with the corresponding 95% confidence intervals. As can be seen for all datasets, except *wdbc*, the similarity metric with greater AP_{SNR} is the one with the best *AUC-MacF1* or it is not significantly different from the best one with 95% confidence. This indicates that AP_{SNR} can be used for evaluating similarity metrics, though further experiments may be necessary for more conclusive results.

Another sign of the relation between the AP_{SNR} and the method result is the Pearson correlation between these two values presented in Table 5. The correlation is held for most datasets. The *yeast* dataset presents a small negative correlation mainly because of the similar performance of the different similarity functions. This can be seen by the close AP_{SNR} and *AUC-MacF1* values. The *wdbc* dataset is a clear outlier and is the other dataset in which the correlation does not hold. This

Table 4
AUC-MacF1 of different similarity functions.

Dataset	Chebyshev	Cosine	Euclidean	Manhattan
ecoli	57.272 ± 2.127	59.265 ± 2.206 ▲	58.043 ± 2.037	57.901 ± 2.132
glass	45.620 ± 1.852 ●	46.558 ± 1.444 ●	46.075 ± 1.761 ●	46.265 ± 1.805 ●
iris	91.277 ± 1.195 ●	91.569 ± 1.428 ●	91.052 ± 1.178 ●	91.365 ± 1.179 ●
wdbc	92.038 ± 0.926	94.322 ± 0.332 ▲	90.013 ± 1.207	88.923 ± 1.317
wine	90.792 ± 0.901 ●	89.311 ± 1.058	91.009 ± 0.808 ●	91.102 ± 0.969 ●
yeast	42.882 ± 0.733 ●	43.274 ± 0.825 ●	43.305 ± 0.836 ●	42.791 ± 0.670 ●

Table 5
Correlation between AP_{SNR} and *AUC-MacF1*.

Dataset	Pearson correlation
ecoli	0.8138943
glass	0.6403044
iris	0.7364903
wdbc	−0.9600043
wine	0.9518166
yeast	−0.4169952

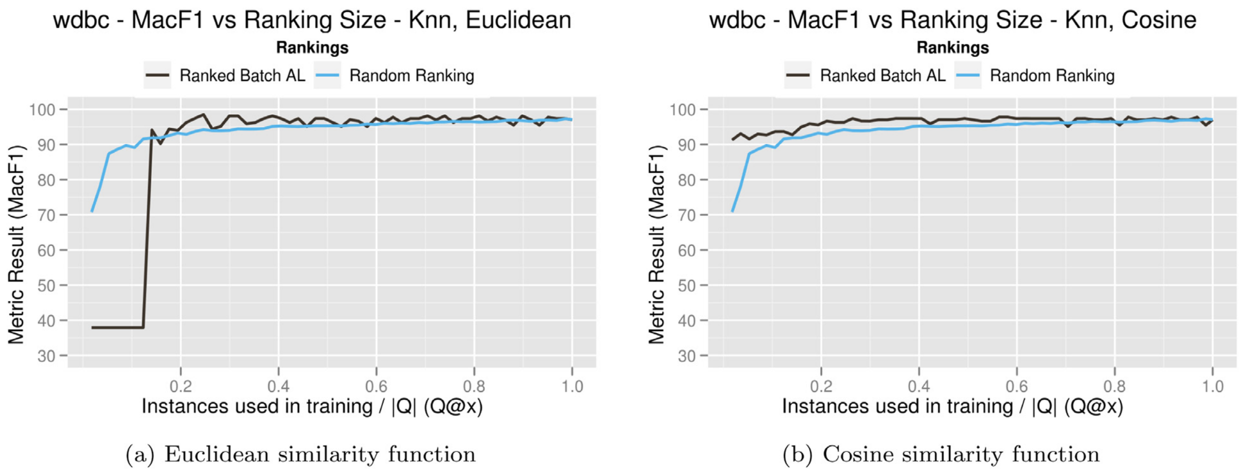


Fig. 6. Comparison of the *wdbc* dataset result when using different similarity functions.

happens because the dataset has only two classes with boundaries not easily determined by the similarity functions. Given that no labeled instances are provided as seed, the classifier does not know the number of classes in the problem. If the similarity function happens to choose instances of the same class, only one concept will be learned, thus hindering the classification performance. In this scenario, a naive similarity function can make a better choice, for example, a random strategy would lead to knowledge about both classes sooner.

This can be better visualized in Fig. 6. In the beginning of the ranking constructed using the *Euclidean* similarity (Fig. 6a), the method fails to select instances from both classes, thus achieving a constant *MacF1* for approximately 10% of the initial labeled set. This start hinders the resultant *AUC-MacF1*. On the other hand, the cosine function (Fig. 6b) is able to discover both classes quicker leading to better classification models early on. Given the presented results, throughout this work, unless noted otherwise, the *Cosine* similarity function will be used due to its small average AP_{SNR}.

6.2.2. Impact of each component

In spite of several AL methods proposed in the literature, the importance of some components of our solution, mainly the uncertainty estimator, have been recognized in different contexts and for different purposes, their relative importance has been rarely investigated. In this work, we evaluate and *quantify* the *relative impact* of each component of our solution as well as their combined contribution. Particularly in the case of the RBMAL problem we have just formalized, this is an original contribution of this work.

One way to evaluate the impact of different components in the final result is by running a $2^k r$ factorial design [41]. In this evaluation, the outcome of an experiment is called *response variable* and each variable that affects the response is called *factor* or *predictor*. Each factor can assume different values (*levels*).

Table 6
Impact of each factor obtained by the factorial design.

Dataset	Amount of variation explained by factor			
	A	B	AB	Error
ecoli	0.878853	0.009723	0.109178	0.002244
glass	0.638297	0.012280	0.324248	0.025172
iris	0.423167	0.050760	0.288955	0.237115
wdbc	0.906906	0.083146	0.002133	0.007813
wine	0.146364	0.249922	0.598597	0.005115
yeast	0.781665	0.066102	0.151970	0.000261

A *full factorial design* investigates every possible combination between factors and levels. Given that the number of combinations can be very large, this factorial design may be very expensive. Thus, there is a simplification called $2^k r$ design, in which each of the k factors has only two levels evaluated. The r stands for the number of times each experiment in repeated in order to measure experimental errors. This factorial design is used as a primary investigation of which factors are relevant for a deeper investigation.

The importance of a factor is represented by the response variation induced by the different levels. Factors that explain a high percentage of variation are considered the most relevant. Given that only two factors will be investigated, the $2^2 r$ factorial design is used here.

In this work, there are two main components to be investigated: the uncertainty estimator (x_A) and the similarity function (x_B). The response variable is the Area Under the Curve of the *MacF1*, considering that at each iteration one document is labeled by the oracle. Thus, by the end of the evaluation, the uncertainty and diversity scores will have been equally weighted throughout the iterations. Since this is a factorial design with replication, the whole split evaluation is conducted multiple times, that is, the same splits are evaluated more than once in order to obtain confidence intervals.

For the uncertainty estimator, the high level is considered to be the classifier with better probability output and the low level is the strategy that assigns a random score for each instance.

The similarity factor has the high level set as the function that better divides instances by their classes, that is, the function that returns high similarity values for instance pairs of the same class and low similarity for instances belonging to different classes. Similarly to the *uncertainty estimator*, the low level is set as being the random strategy that returns a random value for each instance pair.

By doing so, it is possible to assess the impact of the proposed components in the final result in comparison to a naive strategy that outputs random scores. To be consistent with our previous analysis, the high level of the uncertainty estimator will be provided by the *Knn10* classifier and the high level of the Similarity Function is set to the *Cosine* one.

Considering that the classifier parameters should be calculated at each ranking size $\mathcal{Q}@x$, the generated rankings were evaluated multiple times in order to calculate the error in the factorial design. Table 6 presents the amount of the variation that is explained by each factor, being factor A the uncertainty estimator and B the similarity function.

The factorial experiment results show that the impact of the uncertainty estimator in the final result is usually higher than the impact of the similarity function. This means that in order to achieve better results it is usually better to improve the quality of the uncertainty estimator. Two datasets, though, do not present high values for the uncertainty estimator impact. First, in the *iris* dataset, there is a high error in the experiments. This happens because the results in this dataset are highly dependent on the estimated parameter k – small variations in k generate high variations in the response variable. Second, in the *wine* dataset, most of the response variable is explained by the combination of both components. This happens because the resultant metric does not change much regardless of the used component.

6.2.3. Evaluation of the initial selection for the cold start scenario

As discussed, the proposed method can be used to generate a full ranking even when no labeled data is provided. In this scenario, there is no training information for selecting the first instance to be labeled (the cold start problem presented in Section 5). In this work, this problem is tackled by selecting as the first instance the one that is the best representative of the dataset, that is, the one most similar to all others in $\mathcal{U}$. The main idea is that the learner can start by selecting the “most common” knowledge of the dataset.

In this section, the proposed method for selecting the first instance is compared to a random strategy. In the random strategy, a group with one or more instances is randomly selected to compose the beginning of the rank. This is the methodology normally used in Active Learning, and usually has competitive results because the learner gets a sample that obeys the probability distribution of the data.

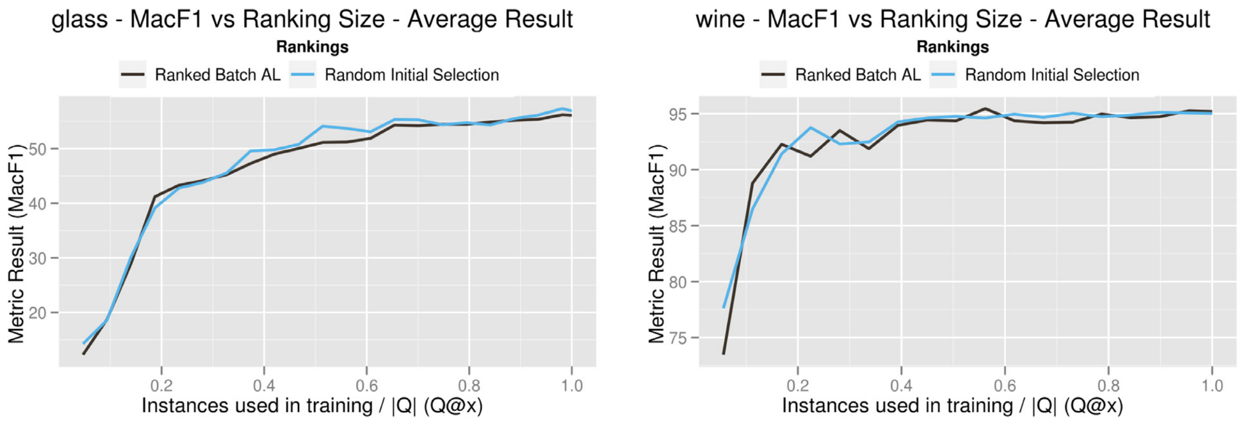
These two methods are compared by calculating the *Area Under the Curve* of the first generated rank. That is, the method is executed without any provided label and the generated rankings are compared.

In the first experiment, the proposed heuristic for initial selection is compared to a random strategy in which the ranking is started with a single instance randomly chosen. Table 7 presents this comparison. Results indicate that there is no substantial difference between the strategies (with a slight advantage for the proposed heuristic in the ecoli dataset). However,

Table 7
Comparison between the proposed heuristic for initial selection versus choosing one random instance.

Dataset	AUC-Acc		AUC-Acc@20%		AUC-MacF1		AUC-MacF1@20%	
	Random	Heuristic	Random	Heuristic	Random	Heuristic	Random	Heuristic
ecoli	79.63	80.26 ▲	69.54	71.01 ●	60.70	61.12 ●	48.41	48.66 ●
glass	55.70	55.10 ●	37.52	35.59 ●	47.80	47.13 ▼	25.14	24.80 ●
iris	92.07	92.09 ●	85.92	85.87 ●	91.79	91.81 ●	84.78	84.86 ●
wdbc	94.25	94.31 ●	90.06	90.31 ●	93.83	93.83 ●	89.38	89.36 ●
wine	93.45	93.22 ●	87.02	87.21 ●	93.23	93.06 ●	85.48	85.82 ●
yeast	48.75	48.98 ●	42.48	42.91 ●	45.71	45.71 ●	36.89	36.52 ●
average	77.30	77.32 ●	68.75	68.81 ●	72.17	72.11 ●	61.68	61.67 ●

▲ Result statistically better than baseline with 95% confidence.
● Result with no statistical difference form baseline with 95% confidence.
▼ Result statistically worse than baseline with 95% confidence.



(a) *glass* (b) *wine*

Fig. 7. *MacF1* comparison of the proposed heuristic for initial selection versus starting with one random instance.

the random strategy has the advantage of not requiring a high computational cost like the "most common" instance selection method. Such similar results of the discussed strategies can be better visualized in Fig. 7. The results for the generated ranking are presented for two datasets: *glass* and *wine*. The dark curve represents the proposed heuristic and the light one represents the ranking started with the random instance. Both curves have similar behaviors, not being clear, which one leads to better *AUC* values.

Although the proposed heuristic and the random strategy show similar results on the first generated ranking, it is not clear the impact of this choice throughout the ranking when instances are labeled by the oracle and the model is iteratively updated. As discussed before, a bad start can hinder the performance of Active Learning algorithms in subsequent choices.

Thus, in this second experiment, the first instance is selected by one of the discussed techniques and its label is queried and assigned. This information is incorporated by the algorithm and a new instance is queried. This process is repeated until there is no instance left in $\mathcal{U}$. The idea is to assess whether the initial choice can lead to different rankings on the long run.

Table 8 presents the results for the ranking generated with one instance labeled per iteration. As can be seen, disregarding the case of the *glass* dataset, in which small losses can be observed, there is now more cases in which the proposed heuristic has advantages, with gains of up 22% in *AUC-MacF1@20%* in the *wine* dataset. Although the differences in averages in some of the methods are not statistically significant with 95% confidence, the proposed method achieves absolute higher averages for every metric. This can be seen as an indicative that the proposed heuristic leads to better results, but further investigation is needed.

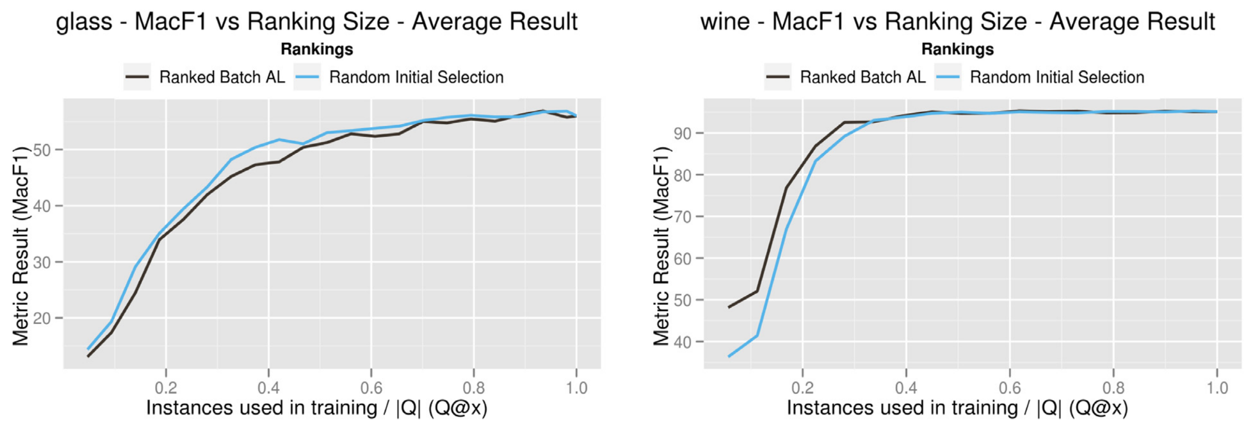
Fig. 8 presents the comparison between the strategies on the *glass* and *wine* datasets. In the dataset with the worst performance of the heuristic (Fig. 8a), the behavior of both curves are very similar. On the other hand, in the *wine* dataset it is easy to see that the proposed heuristic leads to an improvement on the beginning of the ranking, achieving a better classifier with fewer instances. This is an indicative that, given the dataset, the additional cost of the heuristic can actually payoff.

Finally, another possible strategy for building the beginning of the ranking when no labeled instances are provided is to randomly select a larger set of instances. Thus, in the third experiment the first 20% of the ranking is built by randomly selecting instances of the unlabeled dataset. This is a small sample of the underlying probability distribution and a reasonable estimate given the datasets sizes. As in the first experiment, the models are not updated after each selection.

Table 8
Comparison between the proposed heuristic for initial selection versus choosing a random instance with one instance being labeled per iteration.

Dataset	AUC-Acc		AUC-Acc@20%		AUC-MacF1		AUC-MacF1@20%	
	Random	Heuristic	Random	Heuristic	Random	Heuristic	Random	Heuristic
ecoli	79.94	80.22 ●	66.59	66.24 ●	59.21	59.26 ●	38.59	37.26 ●
glass	57.10	56.09 ▼	42.20	40.09 ▼	47.82	46.55 ▼	24.38	21.51 ▼
iris	91.93	92.21 ●	79.98	82.13 ●	91.28	91.56 ●	76.77	78.99 ●
wdbc	94.60	94.84 ▲	89.98	90.71 ●	94.10	94.32 ▲	88.79	89.34 ●
wine	88.54	90.40 ▲	56.29	65.39 ▲	87.15	89.31 ▲	46.51	56.76 ▲
yeast	49.14	49.10 ●	43.35	42.87 ●	43.60	43.27 ●	28.62	27.91 ●
average	76.87	77.14 ●	63.06	64.57 ●	70.52	70.71 ●	50.61	51.96 ●

▲ Result statistically better than baseline with 95% confidence.
● Result with no statistical difference form baseline with 95% confidence.
▼ Result statistically worse than baseline with 95% confidence.



(a) *glass* (b) *wine*

Fig. 8. *MacF1* comparison of the proposed heuristic for initial selection versus starting with a random instance. One instance is labeled and used to updated models at each iteration.

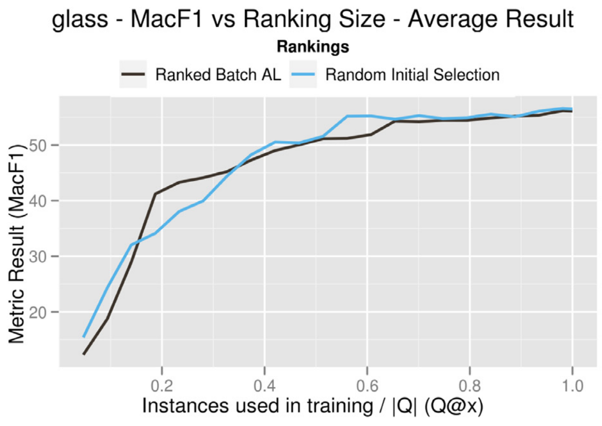
Table 9
Comparison between the proposed heuristic for initial selection (one instance selected) versus choosing a set of random instances.

Dataset	AUC-Acc		AUC-Acc@20%		AUC-MacF1		AUC-MacF1@20%	
	Random	Heuristic	Random	Heuristic	Random	Heuristic	Random	Heuristic
ecoli	80.02	80.26 ●	70.09	71.01 ●	59.58	61.12 ▲	41.02	48.66 ▲
glass	57.49	55.10 ▼	43.06	35.59 ▼	47.49	47.13 ●	27.04	24.80 ●
iris	92.25	92.09 ●	83.53	85.87 ●	91.89	91.81 ●	81.68	84.86 ▲
wdbc	94.45	94.31 ●	91.19	90.31 ▼	93.95	93.83 ●	90.16	89.36 ●
wine	92.36	93.22 ▲	80.87	87.21 ▲	92.06	93.06 ▲	78.19	85.82 ▲
yeast	49.06	48.98 ●	43.71	42.91 ●	43.62	45.71 ▲	29.55	36.52 ▲
average	77.60	77.32 ●	68.74	68.81 ●	71.43	72.11 ●	57.94	61.67 ●

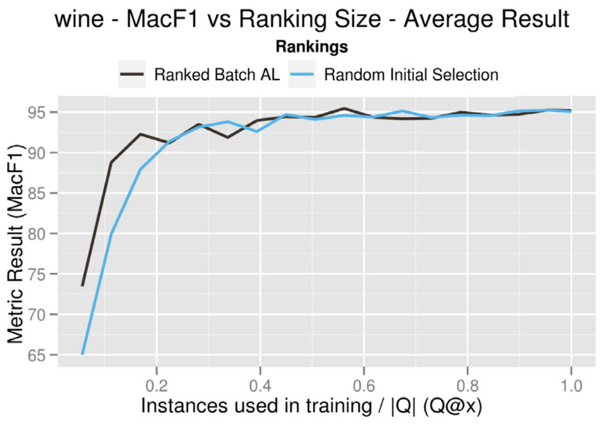
▲ Result statistically better than baseline with 95% confidence.
● Result with no statistical difference form baseline with 95% confidence.
▼ Result statistically worse than baseline with 95% confidence.

Table 9 shows the comparison results. Building the first 20% of the ranking with random instances, as expected, makes the result on the beginning of the ranking equal to the one of a random ranking. This can be a good strategy in datasets in which the proposed method fails to build a good initial ranking. For example, in the *glass* dataset, the accuracy is improved by the random initial selection (although the *AUC-MacF1* shows no significant improvement).

In general, the proposed heuristic presents results that are statistically equal or better than this random initial set selection. This indicates that the initial selection should be small (for example, one instance), and the proposed method should be used for constructing the ranking as proposed in our framework. Fig. 9 presents the comparison of strategies for the *glass* and *wine* datasets. As can be seen, the *MacF1* curve is worse than the curve presented in Fig. 7 when selecting one single initial random instance.



(a) *glass*



(b) *wine*

Fig. 9. *MacF1* comparison of the proposed heuristic for initial selection versus starting with 20% of random instances.

Table 10
Resultant ranking quality when no training is provided Knn Cosine.

Dataset	AUC-Acc		AUC-Acc@20%		AUC-MacF1		AUC-MacF1@20%	
	Random	RBMAL	Random	RBMAL	Random	RBMAL	Random	RBMAL
ecoli	79.56	80.26 ▲	70.23	71.01 ●	57.29	61.12 ▲	41.29	48.66 ▲
glass	56.77	55.10 ▼	43.51	35.59 ▼	46.52	47.13 ●	28.39	24.80 ▼
iris	91.61	92.09 ▲	80.08	85.87 ▲	91.07	91.81 ▲	77.32	84.86 ▲
wdbc	94.05	94.31 ▲	90.06	90.31 ●	93.49	93.83 ▲	88.82	89.36 ●
wine	92.09	93.22 ▲	80.69	87.21 ▲	91.83	93.06 ▲	78.17	85.82 ▲
yeast	48.90	48.98 ●	44.30	42.91 ▼	41.70	45.71 ▲	28.92	36.52 ▲
average	77.16	77.32 ●	68.14	68.81 ●	70.31	72.11 ▲	57.15	61.67 ●

▲ Result statistically better than baseline with 95% confidence.
● Result with no statistical difference form baseline with 95% confidence.
▼ Result statistically worse than baseline with 95% confidence.

To summarize, since the proposed heuristic shows similar or superior performance in the investigated scenarios, it will be used in the next experiments for choosing the first instance of the ranking when no training set is provided. Notice that we are not claiming that this is the "best" possible strategy for all cases. Other strategies proposed in the literature such as pre-clustering [24] could be tested. But the proposed strategy is a simple and reasonable one that works well, as we have seen. We will leave other strategies for the cold start problem to be tested in a future work.

6.2.4. Evaluation of the quality of the full ranking without feedback to the algorithm

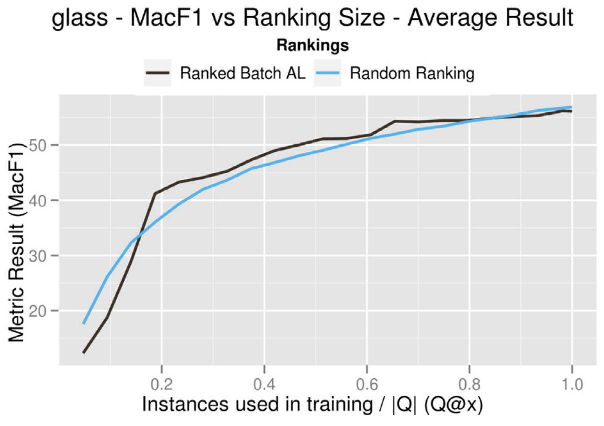
In this section the quality of the ranking generated with our proposed framework is compared to a completely random selection strategy in which only one ranking is generated and no feedback is given to the algorithm. As mentioned before, the random sampling strategy can be a strong baseline given the dataset characteristics [1]. This happens because the random strategy provides an unbiased sample that obeys the underlying probability distributions.

Two different scenarios are considered. First, the full instance ranking is generated without an initial training seed and considering that no label is provided by the oracle. Second, the proposed framework is used to generate a full ranking when half instances are labeled and provided as seed, but no further instances are fed by the oracle.

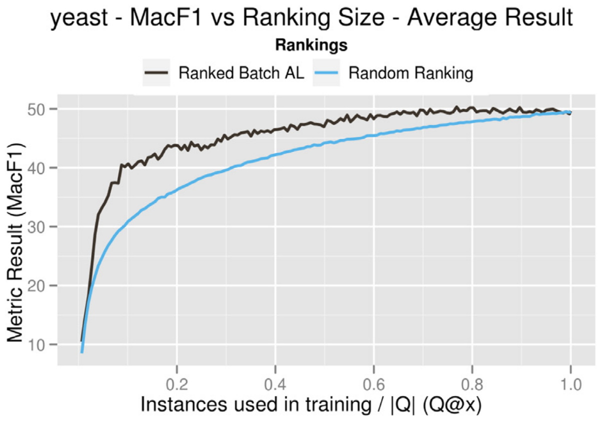
Ranking without training data. As previously discussed, one characteristic of the proposed framework is that the whole set of instances can be ranked even if no seed or labels are provided. In this experiment, the quality of this generated ranking is compared to the random sampling strategy.

A ranking $\mathcal{Q}$ is created using the whole unlabeled set $\mathcal{U}$, which then evaluates the *Accuracy* and *MacF1* of a classifier trained using increasing portions of $\mathcal{Q}$. The random ranking results are obtained by shuffling the instances in $\mathcal{U}$. Note that the calculated ranking is generated without using any labeled instance.

Table 10 presents the results, with the four metrics we have been using for each dataset. As it can be seen, our method, without any training data, presents gains in the *AUC-MacF1* and *AUC-Accuracy* for most datasets and scenarios when compared to the random strategy. Out of 28 results, we are statistically superior to the baseline in 16 cases, tie in 8, and loose in only 4 cases. Looking at specific results, for instance, in the *iris* and *wine* dataset, gains were obtained in all four scenarios, with gains of up to 9.8% for *AUC-Accuracy@20%*. In the *yeast* dataset, we could also obtain gains in *AUC-MacF1* of approximately 9.6% with a small loss of approximately 3.1% in the beginning of the rank (*AUC-Accuracy@20%*).



(a) *glass*



(b) *yeast*

Fig. 10. *MacF1* results for ranking generated without training data.

Table 11
Resultant ranking quality when training is provided Knn Cosine.

Dataset	AUC-Acc		AUC-Acc@20%		AUC-MacF1%		AUC-MacF1@20%	
	Random	RBMAL	Random	RBMAL	Random	RBMAL	Random	RBMAL
ecoli	82.51	83.24 ▲	81.39	81.48 ●	62.88	64.58 ▲	60.44	62.54 ▲
glass	61.69	62.16 ●	59.49	60.43 ●	53.46	54.12 ●	50.13	51.87 ▲
iris	94.48	94.92 ▲	93.70	94.01 ●	94.38	94.83 ▲	93.59	93.89 ●
wdbc	95.40	95.78 ▲	94.93	95.47 ▲	95.02	95.45 ▲	94.51	95.11 ▲
wine	94.57	95.13 ▲	93.94	94.39 ●	94.62	95.15 ▲	94.02	94.40 ●
yeast	50.48	50.96 ▲	49.96	50.25 ●	47.14	48.64 ▲	44.75	46.61 ▲
average	79.85	80.36 ▲	78.90	79.33 ▲	74.58	75.46 ▲	72.90	74.07 ▲

▲ Result statistically better than baseline with 95% confidence.
● Result with no statistical difference form baseline with 95% confidence.
▼ Result statistically worse than baseline with 95% confidence.

In a very few datasets (e.g., *glass*), the generated ranking had a bad start which it was not able to compensate later in the process. The remaining of the ranking stays a little above the random result, but this start is enough to hinder the *AUC* values. In any case, the overall quality of the ranking can be assessed by the *AUC-MacF1*, which is statistically equivalent to the random one. Fig. 10a shows the curve displaying such behavior.

These results can be further highlighted in the resultant chart of *MacF1* versus the number of instances in $\mathcal{Q}$*@x*. Fig. 10b shows the resultant curve in this situation. As we can see, our method provided significant gains over the random curve. Note that the classifiers trained with instances provided in the order of $\mathcal{Q}$ not only start with better quality but also remain with significantly better results throughout the ranking.

Ranking with training data. Despite being able to generate ranks without training any data, it is expected that the use of the query ranking $\mathcal{Q}$ may increase its quality given the available knowledge contained in $\mathcal{D}$. Thus, the next experiment aims to compare our ranking against the randomly generated ranking when some labeled data is provided. Specifically, half of the instances are provided with their labels ($\mathcal{D}$) to build the ranking and the result is compared to the random sampling strategy.

The initial train split is divided once again. Half instances are used as $\mathcal{D}$ (documents provided to the algorithm with labels). The other half of the original training set is used as the unlabeled set $\mathcal{U}$. This set, similarly to the previous experiment, is ranked without additional labeling. The result is the ranked batch $\mathcal{Q}$ containing half instances of the original training split.

In order to evaluate this ranking, each increasing portion of $\mathcal{Q}$ is used in conjunction with the set $\mathcal{D}$ to train the classifier that is evaluated in the test set. The random baseline is generated by using the same $\mathcal{D}$ while shuffling $\mathcal{U}$ to generate $\mathcal{Q}$.

Table 11 presents the results, which, in this scenario, are even more favourable to us than the previous one. In this case, we were statistically superior to the baseline in 19 cases, with 9 ties, and no losses. Note that this situation is potentially harder for our method. As the number of labeled instances initially available increases, the random baseline gets harder to overcome because it provides an unbiased sample of the underlying probability distribution of the data. However, even with this hindrance, our method is able to achieve statistically significant improvements in almost all datasets and metrics. This indicates that our method continues to create rankings better than the random selection as new labels are supplied.

Table 12
Comparison between our method (no labeled data) and a batch-mode AL strategy with the model iteratively updated with labeled instances.

Dataset	AUC-Acc		AUC-Acc@20%		AUC-MacF1		AUC-MacF1@20%	
	Baseline	RBMAL	Baseline	RBMAL	Baseline	RBMAL	Baseline	RBMAL
ecoli	79.71	80.26 ●	70.96	71.01 ●	57.80	61.12 ▲	42.09	48.66 ▲
glass	56.70	55.10 ▼	41.79	35.59 ▼	46.41	47.13 ●	27.99	24.80 ●
iris	92.63	92.09 ●	83.80	85.87 ●	92.22	91.81 ●	81.67	84.86 ▲
wdbc	94.42	94.31 ●	90.96	90.31 ●	93.92	93.83 ●	89.95	89.36 ●
wine	92.58	93.22 ▲	82.12	87.21 ▲	92.30	93.06 ▲	79.62	85.82 ▲
yeast	48.61	48.98 ●	43.74	42.91 ●	41.70	45.71 ▲	29.24	36.52 ▲
average	77.44	77.32 ●	68.89	68.81 ●	70.72	72.11 ●	58.42	61.67 ●

▲ Result statistically better than baseline with 95% confidence.
● Result with no statistical difference form baseline with 95% confidence.
▼ Result statistically worse than baseline with 95% confidence.

6.2.5. Pool-based batch-mode sampling comparison

In this section we compare our method with a state-of-the-art Batch-Mode AL (BMAL) algorithm in two scenarios. First, our method generates a full instance ranking while the baseline is updated with its selected batch at each iteration. Second, our method is fed with the same number of initial labeled instances.

The baseline method used in this section is a Batch-Mode AL method based on Support Vector Machines [14]. Its main idea is to select a batch of informative instances that are also diverse. The informativeness is given by the proximity to the class boundaries (classification hyperplanes), and the similarity between instances is given by the cosine of the angle of induced hyperplanes. The score of each instance to be selected for a given batch b is given by the cosine of the angle of induced hyperplanes. The score of each instance to be selected for a given batch b is given by Eq. (8):

$$s_i = \lambda \times \text{distance}(i) + (1 - \lambda) \times \text{batchSimilarity}(i, b) \tag{8}$$

This method shares some similarities with ours: the λ parameter can, for example, be seen as the α value, although our α parameter is dynamically set according to the evolution of the process, meaning that no tuning is necessary. The uncertainty estimators and the importance of the components in each phase of the process (encapsulated in the adjustable α parameters) are also important differences in the two methods.

In the following experiments, the λ parameter is fixed at 0.5 because this value is used in the original paper that proposed the method and the authors indicate that the used strategy is stable with respect to λ.

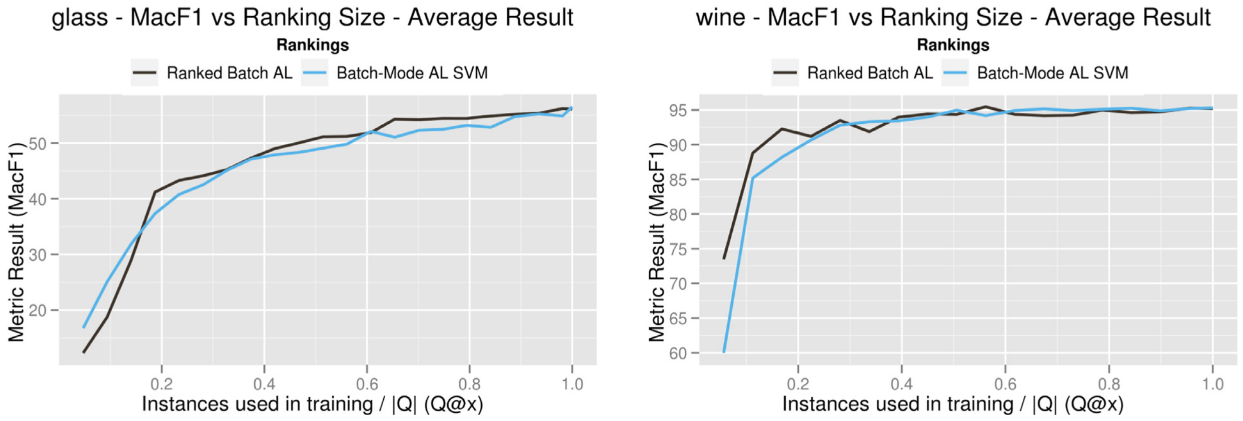
Ranking generated without labels. In this experiment the first ranking generated by our method (without any labeled data) is compared to the final ranking generated by the baseline (with labeled data provided at each iteration). Table 12 presents the comparison results for a batch of size five. Other values for the batch size were also tested and showed similar results. As can be seen, even with this setup that benefits the baseline (labeled data is provided only to the baseline method), our method is able to achieve results superior to the baseline method even when no labeled data is fed to our method. Out of 28 results, we obtained statistically significant gains in 9, a tie 17, and only 2 losses. Best results were obtained in terms of *AUC-MacF1@20%*. For instance, in the *yeast* and *ecoli* dataset there were improvement of 25% and 15.6% on this metric, respectively. In sum, it is possible to avoid a big computational effort maintaining or even improving effectiveness.

Fig. 11 shows two extreme cases. The first one is the *glass* dataset in which the generated ranking is statistically worse than the one generated by the baseline. Note, though, that our method's ranking is generated with one single iteration and given the necessities, it may be a better trade-off to lose some accuracy in order to avoid constant iterations. The second case is in the *wine* dataset in which results are significantly better than the baseline's, with our method having a head-start. This is an important result, which indicates that our method (with only one iteration) is very competitive when compared with similar BMAL methods.

Ranking generated iteratively. Now, we compare our method with the BMAL baseline in a similar scenario for both methods, that is, at each iteration five instances are labeled by the oracle and fed to the algorithms. Both methods, then, update the respective models and query new batches. Table 13 presents the comparison results for a batch of size five. Other values were also tested and showed similar results.

The results show that when the model is updated there is a quality increase in the batch selected by our method. This result reinforces that the proposed method can be a drop-in replacement for traditional Batch-Mode Active Learning methods. For all datasets, the metrics are either significantly equal or better than the baseline method. Even on the *glass* dataset, it is possible to see a statistically significant improvement of 5% in *AUC-MacF1* over the baseline method. For almost all datasets, our method provides a head-start over the baseline, that is, a performance improvement in the first 20% of the selected instances, this is reflected by an improvement in the average that is close to 8%.

As a final comparison, Fig. 12 presents the average curve of *MacF1* versus the size of the ranking used for training of the *glass* and *wine* datasets. It is possible to see that, while the quality on the *wine* dataset is maintained, there is a significant improvement in the *glass* dataset, specially when few instances were selected.



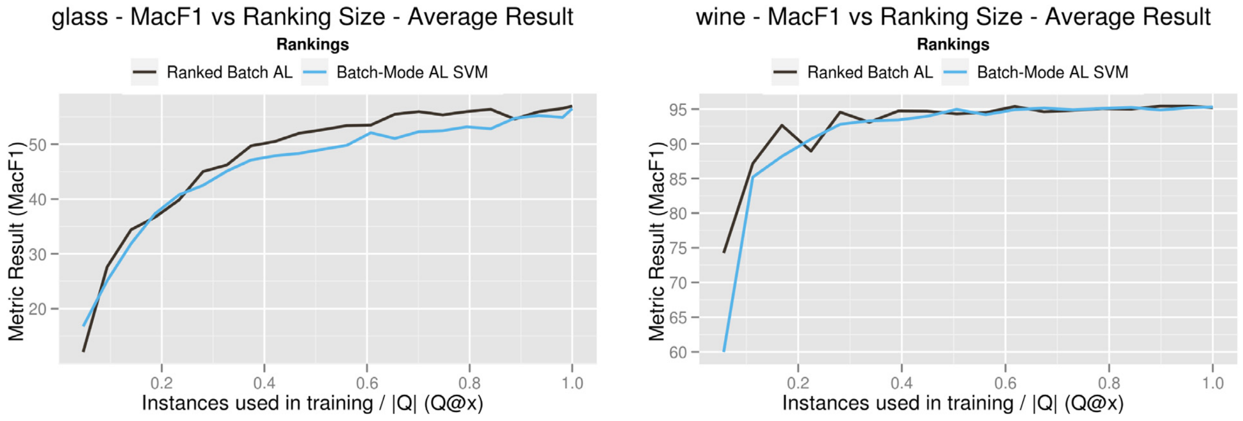
(a) *glass* (b) *wine*

Fig. 11. Comparison between RBMAL and the Batch-Mode AL strategy.

Table 13
Comparison between RBMAL and the Batch-Mode AL strategy.

Dataset	AUC-Acc		AUC-Acc@20%		AUC-MacF1		AUC-MacF1@20%	
	Baseline	RBMAL	Baseline	RBMAL	Baseline	RBMAL	Baseline	RBMAL
ecoli	79.71	80.97 ▲	70.96	70.82 ●	57.80	62.19 ▲	42.09	49.48 ▲
glass	56.70	56.86 ●	41.79	40.89 ●	46.41	48.43 ▲	27.99	28.78 ●
iris	92.63	93.03 ●	83.80	87.04 ▲	92.22	92.79 ●	81.67	86.08 ▲
wdbc	94.42	94.64 ●	90.96	90.44 ●	93.92	94.18 ●	89.95	89.48 ●
wine	92.58	93.43 ▲	82.12	86.84 ▲	92.30	93.21 ▲	79.62	85.29 ▲
yeast	48.61	49.63 ▲	43.74	44.36 ●	41.70	46.96 ▲	29.24	38.61 ▲
average	77.44	78.09 ▲	68.89	70.06 ●	70.72	72.96 ▲	58.42	62.95 ▲

▲ Result statistically better than baseline with 95% confidence.
● Result with no statistical difference form baseline with 95% confidence.
▼ Result statistically worse than baseline with 95% confidence.



(a) *glass* (b) *wine*

Fig. 12. Comparison between our method's generated ranking and the Batch-Mode AL strategy.

6.2.6. Density-sensitive active learning comparison

In this section we compare our method to a state-of-the-art density-sensitive AL (DSAL) method described in [26]. The algorithm uses the well-known entropy measure as an uncertainty metric, combining it with the estimated density around each evaluated unlabeled instance to select new samples for labeling.

Remind that the density-sensitive methods use the similarity metric to estimate the density of a particular point in the search space corresponding to the *unlabeled training set* ($\mathcal{U}$ in our notation). This strategy is based on getting more points in

Table 14
Comparison between RBMAL and DSAL without labels.

Dataset	AUC-Acc		AUC-Acc@20%		AUC-MacF1		AUC-MacF1@20%	
	Baseline	RBMAL	Baseline	RBMAL	Baseline	RBMAL	Baseline	RBMAL
ecoli	73.66	74.65 ●	68.12	67.92 ●	50.75	53.99 ▲	40.25	42.93 ●
glass	56.42	55.41 ●	43.85	41.92 ●	44.82	48.61 ▲	28.84	27.79 ●
iris	93.91	93.76 ●	84.98	83.83 ●	93.57	93.53 ●	82.32	81.84 ●
wdbc	93.14	93.77 ▲	88.17	89.81 ▲	92.40	93.25 ▲	86.48	88.61 ▲
wine	90.07	90.91 ●	80.63	77.82 ●	90.08	90.60 ●	79.16	74.30 ●
yeast	47.44	47.63 ●	44.31	43.41 ●	37.07	40.00 ▲	28.21	30.18 ▲
average	75.77	76.02 ●	68.34	67.45 ●	68.11	70.00 ▲	57.54	57.61 ●

▲ Result statistically better than baseline with 95% confidence.
● Result with no statistical difference form baseline with 95% confidence.
▼ Result statistically worse than baseline with 95% confidence.

the denser regions of the search space in order to avoid outliers and noise. This metric is usually calculated only once, thus being static (never recomputed). In other words, this is a *global* property of the unlabeled set $\mathcal{U}$.

On the other hand, the similarity measure in our RBMAL solutions is used to select the best potential instance to be appended to the *ranking being built*. This selection uses information available in this *"in-progress" ranking of instances* ($D_{estimated}$ in our notation), which may also incorporate labeled information, if it exists. This information is dynamic and updated at each new selection. In other words, this is a *local* property of $D_{estimated}$, which is being dynamically constructed. This is a crucial difference motivated by the fact that we are concentrated in a given execution of the method, on the current ranking being generated in order to optimize it for the labeling round.

As with most uncertainty-based AL methods, at each iteration a model is produced based on the labeled set and used to classify the unlabeled set. The probabilities assigned to each unlabeled sample and class are then used to assess which instances the current model is most uncertain about. In this case, the authors propose using the well-known entropy measure which is defined as:

$$H(x) = -\sum_{c_i \in \mathcal{C}} P(c_i|x) \log P(c_i|x) \tag{9}$$

where $P(c_i|x)$ is the *a posteriori* class probability and $c_i \in \mathcal{C} = \{c_1, c_2, \ldots, c_k\}$. This probability is calculated using the Maximum Entropy method [42].

To estimate the density of an unlabeled instance x, the authors use what they call *K-Nearest-Neighbor-based density* (KNN-density) by taking the K most similar examples $S(x) = \{s_1, s_2, \ldots s_K\}$ of example x and calculating

$$DS(x) = \frac{\sum_{s_i \in S(x)} \cos(x, s_i)}{K} \tag{10}$$

where $\cos(x, s_i)$ is the cosine similarity between instance x and its neighbor s_i.

Finally, for each unlabeled sample, the *density*entropy* measure is defined as:

$$DSH(x) = DS(x) \times H(x) \tag{11}$$

Once this value is computed for all unlabeled samples, they are sorted in descending order of $DSH(.)$, and the first few are selected to be labeled and incorporated into the training set.

To start the selection process, a small initial labeled set is necessary in order to produce a model and calculate the entropy of the instances in the unlabeled set. This DSH baseline uses a clustering technique to select the initial seed training set to bootstrap the AL selection method. We have implemented the method exactly as described in the paper. It starts by selecting an initial seed training set using the k-means clustering algorithm to determine which instances are the most "representative" of the unlabeled set. In this phase, the algorithm selects $k = 10$ random instances and uses the k-means method and the cosine similarity to obtain the centroids for the 10 clusters. It then finds the instances that are closest to these centroids and labels them, creating an initial training set that can be used to perform the actual active selection process based on the $DSH(.)$ metric explained beforehand.

Ranking generated without labels. We run experiments in two scenarios in order to compare our method with the DSH baseline previously described. In the first scenario, we run the DSH baseline using the same labeled initial set (10 instances) and generate a *total ranking* of all unlabeled samples according to their $DSH(.)$ values. This static ranking is then used at each round where the next 5 instances are selected, labeled and inserted into the training set. Thus, in this scenario we do not update the classifier model and use a fixed ranking to select new instances at each round. We propose this scenario in order to show the difference of running a density-sensitive AL as intended, i.e., iteratively with model updates, and running it only once to generate a complete ranking. As argued herein, our method, RBMAL, is specifically designed for this scenario, where a large or complete ranking is generated once or, say, daily, in order to minimize wait time by the human assessors who are labeling the selected instances. In this experiment, no labeled instances are provided to RBMAL.

Table 14 shows the results for the scenario where both methods are run non-iteratively. That is, a single model derived from the initial seed set is used to produce a total ranking of all unlabeled samples and these are selected in batches of 5

Table 15
Comparison between RBMAL and DSAL with the model iteratively updated with labeled instances.

Dataset	AUC-Acc		AUC-Acc@20%		AUC-MacF1		AUC-MacF1@20%	
	Baseline	RBMAL	Baseline	RBMAL	Baseline	RBMAL	Baseline	RBMAL
ecoli	75.06	74.08 ●	70.80	67.61 ●	52.57	52.85 ●	43.43	42.06 ●
glass	57.93	57.78 ●	44.64	45.30 ●	46.05	50.36 ▲	30.38	29.18 ●
iris	94.03	93.73 ●	88.15	86.98 ●	93.75	93.42 ●	86.68	85.22 ●
wdbc	93.61	94.40 ▲	88.75	90.87 ▲	92.99	93.93 ▲	87.26	89.84 ▲
wine	90.29	91.78 ▲	80.88	83.20 ●	90.25	91.69 ▲	79.41	81.77 ●
yeast	47.51	47.47 ●	44.48	42.84 ▼	38.17	39.78 ▲	28.83	29.64 ●
average	76.41	76.54 ●	69.62	69.47 ●	68.96	70.34 ●	59.33	59.62 ●

▲ Result statistically better than baseline with 95% confidence.
● Result with no statistical difference form baseline with 95% confidence.
▼ Result statistically worse than baseline with 95% confidence.

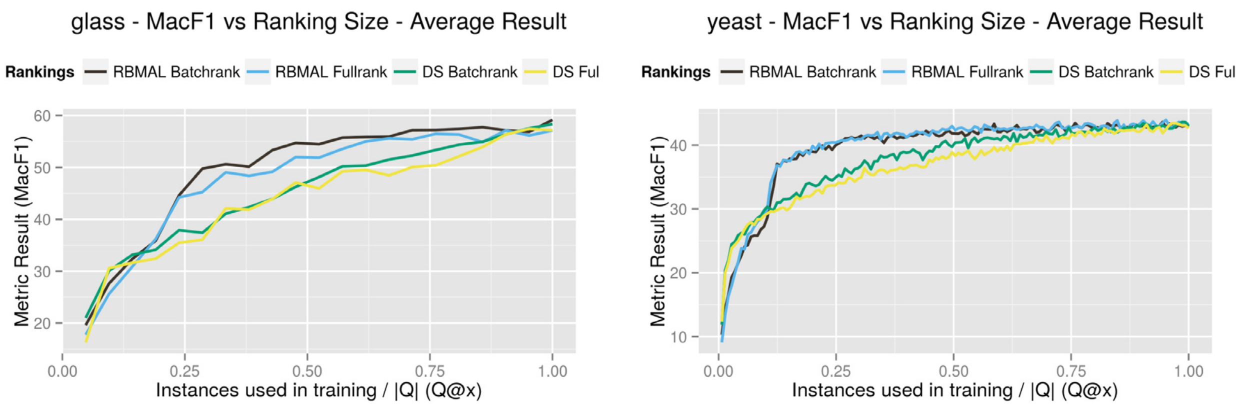


Fig. 13. Comparison between RBMAL and DSAL in both scenarios in two datasets.

without any further model updates. As we can see, our method yields better results in 9 out of 28 cases, tying in 19 cases and never loosing. Results are particularly good in terms of *AUC-MacF1* with gains of up to 8.4% (*glass*). Overall, we can say that, considering the average results, RBMAL generates rankings that are statistically better or equal than those of DSH for all cases.

Ranking generated iteratively. In the second scenario, the DSH baseline is run iteratively as originally intended: the method starts with the 10 instances obtained by k-means clustering and at each round the 5 samples with highest *DSH*(.) value are selected, labeled by the oracle and incorporated into the training set. The method is then run again using the improved model (produced using the expanded training set) to calculate the new *DSH*(.) values for the remaining unlabeled samples. This interactive process is repeated until all unlabeled samples are labeled and incorporated into the training set. In this experiment we want to show that the novel method proposed in this article outperforms a state-of-the-art DSAL technique as originally proposed. Notice that for fairness, in this scenario the RBMAL model is also updated with the same batches of size 5.

Table 15 shows the results for this scenario. As it can be seen, our method performs significantly better than the DSH baseline in 8 out of 28 cases, tying in 19 cases and loosing in only 1. Fig. 13 shows the results for both scenarios in two datasets (*glass* and *yeast*). We call the results for the first scenario “batchrank” and those for the second scenario “fullrank”. In the Figure we can see how RBMAL in batchrank mode is the best method, followed by RBMAL fullrank. The DSH baseline results are also shown in both batchrank and fullrank mode. In fact, results in the middle of the ranking, for instance between 0.25 and 0.75 for *glass* and between 0.15 and 0.50 for *yeast*, are much higher for the two RBMAL variations when compared to those of DSH. Interestingly, the RBMAL results in batchrank mode when compared to the fullrank mode, although superior, are not much higher. Given the incurred costs of the former, this is an interesting trade-off that should be further investigated in the future.

6.3. Lessons learned

Due to the large amount of experiments, we summarize here the main lessons we have learned from our experimental evaluation:

- KNN demostrates to be the most reliable uncertainty estimator among several tested classifiers;
- Similarity functions should be chosen according to the data domain, though Cosine is a good general choice;
- The uncertainty estimator is the component that influences the most the quality of the final ranking;

- Random selection of the first instance is a simple and suitable strategy to deal with the cold start scenario, though the impact of the first selection on the ranking should be further studied;
- Our proposed RBMAL strategy creates rankings that outperform different (strong) baseline algorithms such as Support Vector Machines batch mode AL and Density-Sensitive AL;
- This superiority is observed even in scenarios in which RBMAL does not access any labeled data;
- In the extreme case, the RBMAL ranked batch can contain every instance in the dataset. This can be computed during off-hours, removing batch selection from the labeling loop;
- The proposal of Ranked Batch Mode Active Learning opens new perspectives for novel methods for generating and evaluating batches for labeling.

7. Conclusion

This article has proposed a paradigm change by introducing and directly addressing the Ranked Batch-Mode Active Learning (RBMAL) problem. Motivated by the increasing use of Machine Learning in companies and research groups, in which analysts are paid hourly to label instances to build a training set, we aim to assist businesses and researchers by reducing the analysts' idle time. Specifically, our method relaxes the necessity of Batch-Mode Active Learning to continuously produce new batches, by querying an arbitrarily long ranking of instances. This ranking is achieved by assigning to each instance a score based on its informativeness, and the similarity between this instance and the already selected ones.

We evaluated our framework using six datasets from different domains. It was able to select instances better than the random ranking and showed results comparable to the traditional Batch Mode Active Learning algorithms which are much more inefficient. In fact, in some cases, even with no training data provided, our method was able to achieve better performance than the trained baseline methods. Moreover, our methods have also outperformed state-of-the-art density-based methods in most considered scenarios.

As this work opens a new line of research there is a plethora of future work. First, the two main components of the solution could be further investigated. For instance we could consider other uncertainty estimators such as error estimation, variance reduction or density-sensitivity as well as other similarity functions. One possibility is to work with methods that distinguish between two sources of uncertainty [43]: the aleatoric uncertainty which is due statistical to variability and the epistemic uncertainty which is caused by lack of knowledge (which is our main interest). This would make RBMAL more robust to noise and outliers. Moreover, studying the best components of the solution in other domains such as text or image classification brings completely new challenges, as these domains have their own idiosyncrasies such as high sparseness for the former and the curse of dimensionality for the latter. Devising different strategies for updating the α parameter for different situations is also worth investigating. We also want to exploit other alternatives for the cold start problem, such as the use of pre-clustering methods.

Acknowledgments

This work is supported by InWeb (MCT/CNPq/FAPEMIG grant 573871/2008-6) and by grants from CAPES, CNPq, and FAPEMIG.

References

[1] B. Settles, Active Learning Literature Survey, Technical Report, University of Wisconsin–Madison, 2009. 1648
[2] R. Haertel, et al., Parallel active learning: eliminating wait time with minimal staleness, in: Work. on Act. Learning for NLP, 2010, pp. 33–41.
[3] F. Laws, C. Scheible, H. Schütze, Active learning with amazon mechanical turk, in: Proceedings of EMNLP, 2011, pp. 1546–1556.
[4] L. Shi, Y. Zhao, J. Tang, Batch mode active learning for networked data, ACM Trans. Intell. Syst. Technol. 3 (2) (2012) 33:1–33:25.
[5] J. Bennett, S. Lanning, The netflix prize, KDD Cup and Workshop – in conjunction with KDD, 2007.
[6] D. Aberdeen, O. Pacovsky, A. Slater, The learning behind the gmail priority inbox, NIPS 2010 Workshop on Learning on Cores, Clusters and Clouds, 2010.
[7] B. Settles, M. Craven, L. Friedl, Active learning with real annotation costs, in: Proceedings. of NIPS Work. on Cost-Sensitive Learning, 2008, pp. 1069–1078.
[8] R.D. King, et al., The automation of science, Science 324 (5923) (2009) 85–89.
[9] J. Mostafa, W. Lam, Automatic classification using supervised learning in a medical document filtering application, Inf. Process Manag. 36 (3) (2000) 415–444. http://dx.doi.org/10.1016/S0306-4573(99)00033-3.
[10] D. Angluin, Queries revisited, Theor. Comput. Sci. 313 (2) (2004) 175–194.
[11] A. Beygelzimer, D. Hsu, J. Langford, T. Zhang, Agnostic active learning without constraints, in: Proceedings of NIPS, 2010, pp. 199–207.
[12] D.D. Lewis, W.A. Gale, A sequential algorithm for training text classifiers, in: Proceedings of SIGIR, Dublin, Ireland, 1994, pp. 3–12.
[13] S. jun Huang, R. Jin, Z. hua Zhou, Active learning by querying informative and representative examples, in: J.D. Lafferty, C.K.I. Williams, J. Shawe-Taylor, R.S. Zemel, A. Culotta (Eds.), Advances in Neural Information Processing Systems 23, Curran Associates, Inc., 2010, pp. 892–900.
[14] K. Brinkler, Incorporating diversity in active learning with support vector machines, in: Proceedings. of ICML, 2003, pp. 59–66.
[15] F. Nie, H. Wang, H. Huang, C. Ding, Early active learning via robust representation and structured sparsity, in: Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence, in: IJCAI '13, AAAI Press, 2013, pp. 1572–1578.
[16] Y. Yang, Z. Ma, F. Nie, X. Chang, A.G. Hauptmann, Multi-class active learning by uncertainty sampling with diversity maximization, Int. J. Comput. Vis. 113 (2) (2014) 113–127, doi:10.1007/s11263-014-0781-x.
[17] Y. Yang, D. Xu, F. Nie, J. Luo, Y. Zhuang, Ranking with local regression and global alignment for cross media retrieval, in: Proceedings of the 17th ACM International Conference on Multimedia, in: MM '09, ACM, New York, NY, USA, 2009, pp. 175–184, doi:10.1145/1631272.1631298.
[18] Z. Xu, et al., Representative sampling for text classification using support vector machines, in: Proceedings of ECIR, 2003, pp. 393–407.
[19] T.-Y. Liu, Learning to rank for information retrieval, Found. Trends Inf. Retrieval 3 (3) (2009) 225–331.

[20] Z. Cao, T. Qin, T.-Y. Liu, M.-F. Tsai, H. Li, Learning to rank: from pairwise approach to listwise approach, in: Proceedings of the 24th International Conference on Machine Learning, ACM, 2007, pp. 129–136.
[21] P. Donmez, J. Carbonell, Paired sampling in density-sensitive active learning, in: Proceedings ISAIM, 2008.
[22] P. Donmez, J. Carbonell, P. Bennett, Dual strategy active learning, in: Proceedings ECML, 2007.
[23] A.K. Maccallum, K. Nigam, Employing em and pool-based active learning for text classification, in: Proceedings ICML, 1998.
[24] H.T. Nguyen, A. Smeulders, Active learning using pre-clustering, in: Proceedings ICML, 2004.
[25] N. Roy, A. MacCallum, Towards optinal active learning with applications to text classification, in: Proc. ICML, 2001.
[26] J. Zhu, H. Wang, T. Yao, B.K. Tsou, Active learning with sampling by uncertainty and density for word sense disambiguation and text classification, in: Proceedings of the 22Nd International Conference on Computational Linguistics - vol. 1, ACM Press, Stroudsburg, PA, USA, 2008, pp. 1137–1144.
[27] T. Cover, P. Hart, Nearest neighbor pattern classification, IEEE Trans. Inf Theory 13 (1) (1967) 21–27.
[28] G. John, P. Langley, Estimating continuous distributions in Bayesian classifiers, in: Proceedings of UAI, 1995, pp. 338–345.
[29] D. Hull, Improving text retrieval for the routing problem using latent semantic indexing, in: Proceedings of SIGIR, 1994, pp. 282–291.
[30] C.-C. Chang, C.-J. Lin, LIBSVM: A library for support vector machines, ACM Trans. Intell. Syst. Technol. 2 (2011) 27:1–27:27. Software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm.
[31] T.N.C. Cardoso, Ranked Batch-Mode active learning, Universidade Federal de Minas Gerais, 2012 Master's thesis.
[32] H. Schütze, E. Velipasaoglu, J.O. Pedersen, Performance thresholding in practical text classification, in: Proceedings of CIKM, 2006, pp. 662–671.
[33] K. Tomanek, et al., On proper unit selection in active learning: co-selection effects for named entity recognition, in: Work. on Act. Learning for NLP, 2009, pp. 9–17.
[34] A. Frank, A. Asuncion, UCI machine learning repository, 2010.
[35] S. Tong, D. Koller, Support vector machine active learning with applications to text classification, J. Mach. Learn. Res. 2 (2002) 45–66, doi:10.1162/153244302760185243.
[36] T.G. Dietterich, Approximate statistical tests for comparing supervised classification learning algorithms, Neural Comput. 10 (7) (1998) 1895–1923, doi:10.1162/089976698300017197.
[37] R. Baeza-Yates, B. Ribeiro-Neto, Modern Information Retrieval, Addison-Wesley, 2011.
[38] A.P. Bradley, The use of the area under the roc curve in the evaluation of machine learning algorithms, Pattern Recognit. 30 (7) (1997) 1145–1159.
[39] C.D. Manning, P. Raghavan, H. Schütze, Introduction to Information Retrieval, Cambridge University Press, New York, NY, USA, 2008.
[40] C.A. Heuser, F.N.A. Krieser, V.M. Orengo, Simeval: a tool for evaluating the quality of similarity functions, in: Tutorials, Posters, Panels and Industrial Contributions at the 26th International Conference on Conceptual Modeling, Darlinghurst, Australia, Australia, 2007, pp. 71–76.
[41] R.K. Jain, The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling, Wiley, 1991.
[42] A.L. Berger, S.A.D. Pietra, V.J.D. Pietra, A maximum entropy approach to natural language processing, Comput. Ling. 22 (1) (1996) 39–68.
[43] R. Senge, S. Bösner, K. Dembczyński, J. Haasenritter, O. Hirsch, N. Donner-Banzhoff, E. Hüllermeier, Reliable classification: learning classifiers that distinguish aleatoric and epistemic uncertaintyReliable classification: learning classifiers that distinguish aleatoric and epistemic uncertainty, Inf. Sci. 255 (2014) 16–29, doi:10.1016/j.ins.2013.07.030.