APRENDIZADO ATIVO EM MODO BATCH ORDENADO

THIAGO NUNES COELHO CARDOSO

APRENDIZADO ATIVO EM MODO BATCH ORDENADO

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação do Instituto de Ciências Exatas da Universidade Federal de Minas Gerais como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação.

Orientador: Marcos André Gonçalves
Coorientadora: Mirella Moura Moro

Belo Horizonte, Minas Gerais
04 de julho de 2012

THIAGO NUNES COELHO CARDOSO

RANKED BATCH-MODE ACTIVE LEARNING

Dissertation presented to the Graduate Program in Computer Science of the Universidade Federal de Minas Gerais - Departamento de Ciência da Computação in partial fulfillment of the requirements for the degree of Master in Computer Science.

Advisor: Marcos André Gonçalves
Co-advisor: Mirella Moura Moro

Belo Horizonte, Minas Gerais

July 4, 2012

Cardoso, Thiago Nunes Coelho
C268a Aprendizado Ativo em Modo Batch Ordenado / Thiago Nunes Coelho Cardoso. — Belo Horizonte, Minas Gerais, 2012
xiv, 71 f. : il. ; 29cm

Dissertação (mestrado) — Universidade Federal de Minas Gerais - Departamento de Ciência da Computação
Orientador: Marcos André Gonçalves

Coorientadora: Mirella Moura Moro

1. Computação - Teses. 2. Inteligência Artificial - Teses. 3. Aprendizado do Computador - Teses. I. Orientador. II. Coorientadora. III. Título.

CDU 519.6*82(043)

UNIVERSIDADE FEDERAL DE MINAS GERAIS
INSTITUTO DE CIÊNCIAS EXATAS
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

FOLHA DE APROVAÇÃO

Aprendizado ativo em modo batch ordenado

THIAGO NUNES COELHO CARDOSO

Dissertação defendida e aprovada pela banca examinadora constituída pelos Senhores:

PROF. MARCOS ANDRÉ GONÇALVES - Orientador
Departamento de Ciência da Computação - UFMG

PROFA. MIRELLA MOURA MORO - Co-orientadora
Departamento de Ciência da Computação - UFMG

PROF. ALTIGRAN SOARES DA SILVA
Departamento de Ciência da Computação - UFAM

PROFA. JUSSARA MARQUES DE ALMEIDA GONÇALVES
Departamento de Ciência da Computação - UFMG

Belo Horizonte, 04 de julho de 2012.

Acknowledgments

Initially I would like to thank my advisor Marcos and my co-advisor Mirella for the assistance, support and guidance. I would like to thank Professor Adriano Velloso for the insightful meetings and opinions.

I would like to show my gratitude to Daniel Galinkin, a great friend and fellow student during the Master's degree. Thanks to people at Zahpee for being great friends and partners. Thanks to the lab geeks Thiago Salles, Vitor Oliveira and Daniel Hasan for the never ending Machine Learning discussions. And to my friends Flávio, Renato and André for their patience in my absence.

I would like to thank Roberto and Mariza for assisting me along this path and Carolina for being a partner in work and life. Finally, I would like to thank my family: Márcio, Flor, Bruno, Lucas and Izabela, for the love, support and for being so comprehensive.

"Ancora Imparo"
(Michelangelo)

Resumo

Com a enorme quantidade de informação gerada todos os dias na Internet, fica cada vez mais difícil, se não impossível, processar e administrar manualmente esses dados. Como uma maneira de contornar este problema, algoritmos de Aprendizado de Máquina vêm sendo cada vez mais utilizados nos mais distintos domínios. Um tipo de algoritmo de aprendizado, o Aprendizado Ativo, surgiu como uma maneira de otimizar a fase de treinamento de alguns desses métodos, com a premissa de que os algoritmos são capazes de obter melhores resultados utilizando menos treinamento caso possam escolher quais instâncias devem ser rotuladas. Essa é um premissa especialmente interessante em cenários em que o dado não rotulado é abundante e existe um custo, não desprezível, associado ao processo de rotular uma determinada instância. Em sua forma original, uma instância é rotulada e utilizada pelo algoritmo a cada iteração, o que impossibilita o uso de vários oráculos em paralelo. Para resolver este problema, surgiram os métodos de Aprendizado Ativo em Modo Batch, que são capazes de selecionar mais de uma instância a cada iteração. Apesar de estes métodos resolverem o problema dos múltiplos oráculos, ainda existe uma dependência de se executar o algoritmo a cada batch analisado. Com o crescimento do uso desses métodos em ambientes corporativos, surgiu a necessidade de se evitar iterações constantes e, consequentemente, o tempo ocioso de analistas contratados, que esperam por um novo batch. Nesta dissertação, o problema de Aprendizado Ativo em Modo Batch Ordenado é descrito propiciando um relaxamento do método tradicional em Batch. Ao selecionar uma lista ordenada de instâncias é possível gerar uma lista com um número arbitrário de documentos a serem rotulados. Dessa maneira, as iterações do algoritmo podem ser espaçadas conforme a necessidade do usuário. Tal fato possibilita que uma lista de instâncias suficientemente grande (para um dia completo de trabalho do analista) possa ser gerada fora do horário comercial. Além da definição formal deste problema, uma solução é apresentada, utilizando um framework que constrói a lista iterativamente ponderando a utilidade da instância para o classificador (incerteza) e a diversidade trazida ao modelo em relação as instâncias já selecionadas. A avaliação experimental demonstra que o uso do Batch

Ordenado provê uma redução do número de execuções do algoritmo, mantendo a qualidade das instâncias selecionadas. Em alguns casos, utilizando somente o conteúdo não rotulado disponível, os resultados obtidos são melhores que os obtidos utilizando métodos tradicionais. Em outras palavras, uma lista ordenada de instâncias, gerada a partir do conteúdo não analisado, foi responsável por um processo de rotulação com resultados estatisticamente melhores ou iguais ao de algoritmos tradicionais de aprendizado ativo mas sem suas limitações.

Abstract

With the large amount of information generated every day on the internet, it is getting harder, if not impossible, to manually administrate and process such data. In order to overcome this problem, Machine Learning algorithms are becoming widely used in different domains. The Active Learning field arose as a way to optimize the training phase of some machine learning methods. The main idea is that algorithms can achieve better results with smaller training sets if they are allowed to select which instances should be labeled. This assumption is specially interesting in scenarios in which unlabeled data is abundant and there is a cost, not negligible, associated with instance labeling. In its original form, one instance is labeled and incorporated by the Active Learning algorithm at each iteration, thus making it impossible to use multiple oracles in parallel. In order to solve this problem, Batch-Mode Active Learning methods arose by being able to select more than one instance at each iteration. Although this methods allow the use of multiple oracles, it is still necessary to run the algorithm at each annotated batch. With the increasing use of these methods in business environments, it is important to reduce the necessity of constant iterations, and consequently, the analysts' idle time when waiting for a new batch to be created. In this dissertation, the Ranked Batch-Mode Active Learning problem is described. It relaxes traditional Batch-Mode Active Learning methods by generating a query that is an ordered list of instances, thus allowing batches to be of arbitrarily large sizes. In this way, the algorithm iterations can be spaced according to the user needs. This characteristics allow that a sufficiently large instance list (for a full work day) be generated outside working hours, then avoiding frequent stops for batch construction. In addition to the formal definition of this problem, one solution is presented that consists of a framework which iteratively builds the instance list by weighting the instance utility for the classifier (uncertainty) and the diversity brought to the model regarding already labeled and selected instances. The experimental evaluation shows that the Ranked Batch allows the reduction of the algorithm executions while maintaining the quality of the selected instances. In some cases, using only unlabeled data, the results obtained are better to

the ones of traditional methods. In other words, an ordered list of instances, generated using unlabeled data, was able to guide a labeling process with results statistically better or equal to traditional active learning algorithms without their limitations.

List of Figures

List of Tables

Contents

Chapter 1

Introduction

Machine Learning algorithms have become increasingly popular for a number of reasons. In particular, the volume of data generated every day is huge and impracticable to be manually read and processed by human analysts. For example, Twitter, a popular micro-blogging tool in which users are allowed to post short messages (with up to 140 characters), may produce up to thousands posts per second[1]. In days of popular events, *e.g.* in the MTV Video Music Awards, the number of posts-per-second can be as high as 8,686. Analyzing such big volume of data is impractical without automated systems.

One category of Machine Learning algorithms consists of *supervised* algorithms, that is, algorithms that are able to execute specific tasks (e.g. document classification) given that a training set containing labeled examples is provided. These algorithms are also called *learners* because they are able to increase their performance on the task execution as new training examples are provided. However labeling the training examples can be an expensive task because of the expert knowledge necessary and the time-consuming nature of the process. Moreover, some applications may need a substantial number of labeled instances in order to achieve an acceptable error rate. Finally, some applications may also need constant training data to cope with changes on patterns of use.

In order to reduce the effort of creating such training set, a new sub-field of Machine Learning arose: Active Learning. This type of algorithm is able to select and present to the analyst (considered as an oracle) instances that should be labeled first. This group of instances is called a *query* since it requires "answers" (i.e, labels) from the oracle[2]. After labeling, these instances are incorporated into the training

[1] http://yearinreview.twitter.com/en/tps.html

[2] Note that this notion of query is very different from the one commonly used in Information

set with an expectation of rapidly increasing the classifier model quality. In general, traditional Active Learning 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 is an expensive process implying in a long waiting time. For instance, creating a labeled set using a crowdsourcing service like Amazon Mechanical Turk[3] is impratical with such limitations. Since these services allow users to create labeling tasks that are outsourced to a large group of freelance workers, querying one instance at a time would not take advantage of this highly parallel environment.

To alleviate the aforementioned problems, a Batch-Mode Active Learning algorithm can be used. Such class of algorithms can query multiple instances at once that, then, can be independently labeled in parallel. Although this solves the problem of using multiple analysts, the algorithm still has to execute frequently, thus making the analysts idle once per iteration. This happens because after the batch is labeled and incorporated in the training set, a new query must be constructed. Increasing the queried batch size can reduce the idle time since more instances are labeled at each iteration. However this may come at the cost of labeling instances in an ineffective order. This happens because instances inside a batch are usually not ordered. In other words, selecting an arbitrarily large number of instances can lead to a sampling strategy close to the random one.

To overcome these issues we introduce the Ranked Batch-Mode Active Learning problem. The main idea is to relax some of the aforementioned limitations, such that, the query is a ranked list instead of an unordered set. This problem is more connected to real world scenarios, in which it is very common to have analysts paid hourly to build a training set for a given problem, or there is a limited budget and we want to maximize the gains with this budget. This 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 without having the necessity to wait for the learner update and query construction.

In addition to the definition of this new problem, we also propose a method for ranking an arbitrarly large set of unlabeled instances in descending order of informativeness, that is, the order in which they should be labeled by the oracle. After one or more instances are labeled, the acquired knowledge can be incorporated by the method

Retrieval and similar fields.

[3]https://www.mturk.com/

and used to generate a new, more accurate, ranking of the (still) unlabeled instances.

In summary, each iteration of the method happens in three phases: *Uncertainty Estimation*, *Ranking Construction* and *Labeling by Oracle*. In the *Uncertainty Estimation* step, a score is assigned to each candidate instance given by the confidence of a classifier, trained with the available labeled data, in the prediction. This score is responsible for ranking higher instances in which the classifier is least confident. The second step, *Ranking Construction* is where the ranking itself is built. Instances are selected based on the previously calculated uncertainty scores and on a similarity score that represents the new knowledge that will be brought to the classifier if a given instance is selected. By weighting these two factors, it is possible to prioritize diversity in the first iterations and uncertainty in the later ones, based on the intuition that one should start by having a macro vision of the instance space and should end having a refined vision of the class boundary regions. The resultant query, differently from other Active Learning methods, is a ranking of instances in the order they should be labeled by the oracle. Finally the ranking is presented for *Labeling by the Oracle* that annotates one or more instances. This knowledge can then be used to update the model and this process can re-start.

We evaluated our method using datasets from the UCI Machine Learning repository, belonging to different domains. Our experimental results demonstrate that the proposed method is able to select instances in a better way than random selection, with results comparable to traditional Active Learning algorithms. In other words, the proposed method can be a drop-in replacement for traditional methods without their limitations. Our framework is able to achieve the same accuracy of these methods with only one ranking built, that is, the initial ranking generated without any labeled data had similar performance to traditional models iteratively updated with labeled instances.

1.1 Main Contributions

The main contributions of this work are summarized as follows.

The introduction of a new problem: Ranked Batch-Mode Active Learning This formulation relaxes the traditional Batch-Mode Active Learning by querying a ranked list of instances.

The proposal of a framework for solving the proposed problem This new method constructs a ranked query even when no labeled data is provided. We

conducted an experimental evaluation using datasets of different domains from the UCI Machine Learning repository.

An analysis of the framework components We conducted a $2^k r$ factorial design experiment and evaluated the impact of each of the framework's main components on the final result.

1.2 Text Organization

This work is organized as follows. Chapter 2 overviews the Active Learning field as well as some directly related work. Chapter 3 introduces the Ranked Batch-Mode Active Learning problem and presents our method for solving it. Chapter 4 presents the experimental evaluation and discusses the main results. Finally, Chapter 5 concludes this dissertation, reviewing our main contributions and proposing future work.

Chapter 2

Active Learning

2.1 Machine Learning Basics

This section presents an overview of the Active Learning problem. Specifically it contains distilled core ideas, methods and applications, as well as notational conventions, which will be used throughout this work.

2.1.1 Classification Problem

Computer programs that are able to improve their performance at some task through experience are said to be *learners*. As presented in Mitchell [1997]:

Definition 1 *A computer program is said to* ***learn*** *from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.*

In the Classification Problem, the experience E is the training data provided which consists of multi-attribute records (features) along with a special variable called *class*. The class is a categorical attribute belonging to a finite set. The provided data is then used to construct a model that relates given features to the class variable (task T). Afterwards this model can be used to predict the class of test instances in which the features are provided and the classes are unknown.

Let $\mathcal{C}$ be the finite set of all possible classes, and $\mathcal{D}$ a training set consisting of instances $d_i = (x_i, c_i)$, where x_i is a vector containing the corresponding value for each feature, and $c_i \in \mathcal{C}$ is a categorical attribute indicating its class. The main goal of a classification algorithm is to construct a mapping function $f : \mathcal{D} \to \mathcal{C}$ that is a good approximation of the real mapping function $g : \mathcal{D}_U \to \mathcal{C}$, being $\mathcal{D}_U$ the instance

universe, in order to predict the unknown classes of instances, $t_i = (x_i)$, of the test set $\mathcal{T}$.

As stated by Definition 1, in order to improve the task execution quality, it is necessary to provide more experience to the algorithm (increasing the training set with new instances). The amount of training that should be provided to a learning algorithm is the focus of different machine learning theory studies and can vary based on: the application (different problems can have different class boundaries), the algorithm and the quality of the training set (e.g. presence/absence of noise, disagreement between multiple annotators).

The Machine Learning Theory field provides a framework [Mitchell, 1997] to calculate a lower bound for the size m of a randomly drawn $\mathcal{D}$ given: δ, which is the complement of the probability that the learner will be able to learn a model hypothesis that is approximately correct; a ϵ, which represents the upper bound for the expected error; and the algorithm's Vapnik-Chervonenkis dimension (VC) [Blumer et al., 1989]. This is defined in Equation 2.1.

$$m \geq \frac{1}{\epsilon}\left(4\log_2\left(\frac{2}{\delta}\right) + 8 \times \mathrm{VC}_{\mathrm{example}} \times \log_2\left(\frac{13}{\epsilon}\right)\right) \tag{2.1}$$

Given ϵ, δ and VC, this framework allows us to estimate the number of training instances for a classification application. For example, consider a Neural Network that is being used for classifying images as a young corn or a weed (classes) in order to automate herbicide spraying in agricultural fields [Yang et al., 2000]. If a simple Perceptron with 10,000 units (one unit per pixel, 100x100 images) in the input layer is used, the VC dimension, as presented in [Sontag, 1998] is $VC_{\mathrm{example}} = \mathrm{input\_units}{+}1 = 10,001$. Let m be the number of training instances needed for a $\epsilon = 0.5$ and $\delta = 0.5$, then the sample complexity of this example can be given by Equation 2.2.

$$m \geq \frac{1}{\epsilon}\left(4\log_2\left(\frac{2}{\delta}\right) + 8 \times \mathrm{VC}_{\mathrm{example}} \times \log_2\left(\frac{13}{\epsilon}\right)\right)$$

$$m \geq 8\log_2(4) + 16 \times 10,001 \times \log_2(26)$$

$$m \geq 752,162 \tag{2.2}$$

As Equation 2.2 shows, the number of training instances needed to achieve a good classifier can be large. The difficulty of obtaining these labels can vary according to the domain. It can be as easy as retrieving user data or it can require expert knowledge, being an expensive task.

2.1.2 When Labels Are Costly to Obtain

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 usually by clicking the "Spam" or "Not Spam" buttons[1]. Other scenarios include ratings for movies [Bennett and Lanning, 2007] (used in new movies recommendation); assigning priority to emails [Aberdeen and Slater, 2010] (used to improve the quality of the priority inbox system); "Like"[2] and "+1"[3] social markers (used in recommendation).

On the other hand, there are other scenarios in which such labeling task can be difficult, time-consuming or expensive. Examples include:

Speech recognition: Accurate transcription of speech utterance at phonetic level is extremely time-consuming (taking 400 times longer than the utterance duration) and requires linguistic expertise. Transcription at word level is still time consuming, requiring about ten times longer than the audio duration. [Zhu, 2005].

Hypothesis generation and experimentation: The process of labeling a given instance can be a biological experiment [King et al., 2004], which implies in time and money costs.

Information extraction: Highlighting entities (e.g. person and organization names) and relations (e.g. person works for given entity) in documents can take more than 30 minutes, even for a simple newswire story [Settles et al., 2008]. Other fields may require experts.

Classification and filtering: A training set consisting of annotated documents (e.g. articles, web pages) or other kinds of media (e.g. images) can be a slow, tedious and even redundant task. There are fields in which additional expertise is required, for example, in medical image categorization [Lehmann et al., 2005].

Active learning systems aim to overcome this labeling bottleneck as explained in the following sections.

[1] What happens when I click the "Spam" or "Not Spam" button for an email I received? `http://help.yahoo.com/l/pt-pt/yahoo/mail/ymail/abuse/abuse-59.html`

[2] Facebook's like button: `http://developers.facebook.com/docs/reference/plugins/like/`

[3] Google's +1 button: `http://www.google.com/+1/button/`

2.2 Query Strategy

Active Learning is the sub-field of Machine Learning in which the learning algorithm is responsible for querying data that should be annotated by an oracle, such as a human analyst or an experiment. This labeled set is then used in the model construction.

The main idea is that, by querying instances, the algorithm will be able to learn better classification models with fewer examples, thus reducing labeling costs. In this way Active Learning systems are well-motivated in different Machine Learning problems in which data may be abundant and easily obtained whereas annotated data is expensive to obtain.

In order to Active Learning to fulfill its objective of learning a good classification model with few examples, it is necessary to choose the most informative queries, that is, the instances that, when labeled, will better improve the classification model. In this section an overview of different strategies for measuring the informativeness is presented.

2.2.1 Uncertainty Sampling

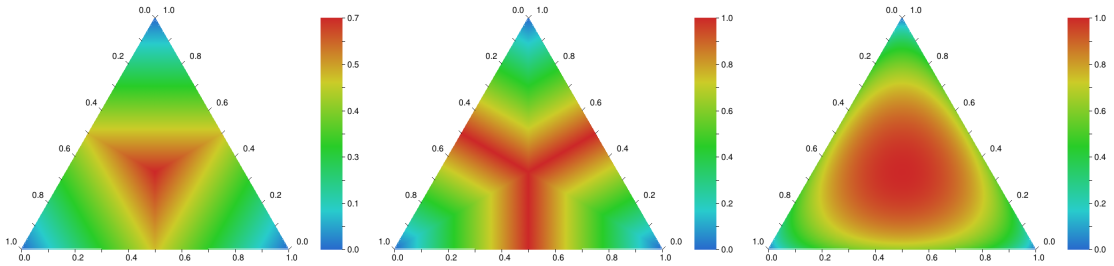
Uncertainty sampling [Lewis and Gale, 1994] is a simple and vastly used query selection framework. The main idea is to query instances in which the learner is least certain about its prediction. This is usually accomplished by using the probabilities that are output by the classifier. For example, in a binary problem, the learner will select instances predicted with probability close to 0.5.

For multiclass problems, there are multiple ways of using this probabilities. The trivial way of selecting one instance in the multiclass scenario is by querying the instance in which a prediction is made with the smallest confidence. Let $p_t(c_i)$ be the probability of instance t belonging to class c_i and $c_{\text{PREDICTED}}$ the class with biggest probability. Then q, the queried instance, can be selected by Equation 2.3.

$$q = \arg\max_t 1.0 - p_t(c_{\text{PREDICTED}}) \tag{2.3}$$

One problem of this strategy is that it only considers the probability of the predicted class, thus, ignoring the remaining distribution. Despite this limitation, such a strategy is widely used in extraction tasks [Culotta and McCallum, 2005] given that the probability of the most likely label sequence can be efficiently computed.

There are two main methods that consider probabilities of classes other than the predicted one: the *margin sampling* and the *entropy sampling*. In the *margin sampling* [Schapire et al., 1998], the objective is to select instances in which the margin between



(a) Least confident (b) Margin (c) Entropy

Figure 2.1: Ternary heatmap illustrating the weight given by each uncertainty measure in a problem with three classes

the first and second most probable classes is narrow. The intuition is that instances with a big difference in these two probabilities are easily distinguishable by the learner, whereas instances in which these probabilities are close (usually) lay near the class boundaries. Given that c_{FIRST} is the class with biggest probability and c_{SECOND} the second most probable class, then the *margin sampling* can be defined by Equation 2.4.

$$q = \arg\max_t p_t(c_{\text{FIRST}}) - p_t(c_{\text{SECOND}}) \tag{2.4}$$

This alleviates the problem of losing probabilities information, but important information can still be discarded in datasets with many classes. The *entropy sampling* [Shannon, 1948] is one way of considering all probabilities output by the learner. It is widely used in different problem domains [Jain and Kapoor, 2009] like object recognition [Holub et al., 2008], parser acquisition [Hwa, 2001], image classification [Joshi et al., 2009] and others. The idea is to measure how many options, or uncertainty, exist within the outcome of a probability distribution. The highest entropy value is achieved when every component has the same value, that is, every class has equal probability. Equation 2.5 presents the entropy sampling strategy.

$$q = \arg\max_t \sum_i -p_t(c_i) * \log(p_t(c_i)) \tag{2.5}$$

Figure 2.1 presents an example of a dataset with three classes and the weight associated by each uncertainty measure to an instance given the probabilities output by the learner. As expected, in all plots, the center is the point where the most informative instance lies, because this is where the probability distribution is most uniform ($p_1 \approx p_2 \approx p_3$). The entropy measure, differently from the other two, does not favor instances in which only one class is highly unlikely (region near the triangle

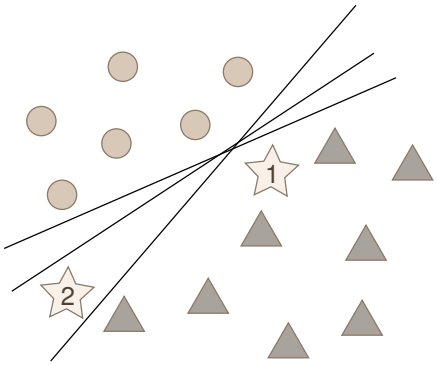


Figure 2.2: Query by Committee sampling example with two classes and committee of size three

sides). This happens because the entropy considers all probability values, prioritizing instances that have a non-zero probability of belonging to each class.

In this work, this is the query framework that will be used, given the simple model and the low computational cost.

2.2.2 Query By Committee

The query-by-committee [Seung et al., 1992] sampling consists of maintaining a committee of models that represents competing hypotheses regarding the instance space. The queried instance is the one that the models most disagree. The idea is that by choosing instances in which the models disagree, the learner aims to discard the biggest number of hypotheses, regardless of the instance label. This can be seen as a simplification of the hypothesis space search proposed by Angluin [1988], in which just a subset of hypotheses (represented by the committee) is maintained.

Figure 2.2 shows an example of the query-by-committee sampling strategy in a binary problem. The hypothesis of each classifier is represented by a class boundary line. Unlabeled instances are depicted as numbered starts. The star number 1 is classified as a triangle by all hypothesis, differently from star 2 that is predicted as circle by one hypothesis. In this scenario, star number 2 is selected for labeling due to the disagreement in the committee.

The committee can be constructed in different ways according to the classifier used. Seung et al. [1992] sample a committee of random hypotheses. In generative model classes, different models can be created by sampling a given probability distribution [McCallum and Nigam, 1998; Dagan and Engelson, 1995]. Other methods use ensemble techniques like bagging, boosting [Abe and Mamitsuka, 1998] or stacking [Purpura et al., 2008]. Finally, the committee can also be generated by partitioning the

feature space [Muslea et al., 2000] or by creating artificial data [Melville and Mooney, 2004, 2003].

The measure of the committee disagreement is also a research topic and, besides the simple vote count, two other approaches are used. The first one is *vote entropy* [Dagan and Engelson, 1995] that can be seen as a generalization of the entropy uncertainty sampling. Let $V(c_i)$ be the number of votes cast by the committee to class c_i, and B the size of the committee. Then, the *vote entropy* is defined by Equation 2.6.

$$q = \arg\max_t \sum_i^{|\mathcal{C}|} -\frac{V(c_i)}{B} * \log \frac{V(c_i)}{B} \tag{2.6}$$

The other disagreement metric is the *Kullback-Leibler* (KL) divergence to the mean [McCallum and Nigam, 1998]. The main idea is to calculate the average of the KL between each committee member distribution and the "consensus" distribution (average probability), that is, the resultant distribution after combining the whole committee. Let $P_b(c_i)$ be the probability of instance t belonging to class c_i given the model b, $P_B(c_i)$ be the probability of instance t belonging to class c_i given the consensus (average probability). Then the KL disagreement metric is defined by Equations 2.7 and 2.8.

$$q = \arg\max_t \frac{1}{B} \sum_b^B D(P_b||P_B) \tag{2.7}$$

$$D(P_b||P_B) = \sum_i^{|\mathcal{C}|} P_b(c_i) \log \frac{P_b(c_i)}{P_B(c_i)} \tag{2.8}$$

2.2.3 Expected Error Reduction

The Expected Error Reduction query framework aims to select instances in which the generalization error (error in the test set) is more likely to be reduced. This is accomplished by estimating the expected error of a model trained with $\mathcal{D}$ and a selected instance i of the unlabeled set $\mathcal{U}$. The instance with smallest expected error, or risk, is then queried.

Since the true label of the instances in $\mathcal{U}$ are not known, the expected error is approximated by calculating the expectation over all possible labels. There are different ways of computing this error, each one with its special meaning. Let $P(c_i|t)$ be the probability of instance t belonging to class c_i in the model trained using $\mathcal{D}$; $P_{t_{c_j}}(c_i|t)$ be the analogous probability with model trained with $\mathcal{D}$ and instance t belonging to class c_j; and $c_{\text{PREDICTED}}$ be the predicted class for a given instance. Then Equation

2.9 presents the Expected Error Reduction framework for the 0/1-loss function, and Equation 2.10 presents this query strategy for the log-loss error function.

$$q = \arg\min_{t} \sum_{i}^{|\mathcal{C}|} P(c_i|t) \left(\sum_{u}^{\mathcal{U}} 1 - P_{t_{c_i}}(c_{\text{PREDICTED}}|u) \right) \tag{2.9}$$

$$q = \arg\min_{t} \sum_{i}^{|\mathcal{C}|} P(c_i|t) \left(\sum_{u}^{\mathcal{U}} \sum_{j}^{|\mathcal{C}|} P_{t_{c_i}}(c_j|u) \times \log P_{t_{c_i}}(c_j|u) \right) \tag{2.10}$$

The 0/1-loss equation can be seen as a minimization of the total number of incorrect predictions, whereas the log-loss equation can be seen as a minimization of the expected entropy.

2.2.4 Variance Reduction

It is usually expensive to directly minimize the expectation of a loss function. However, it is possible to reduce the generalization error indirectly by minimizing the output variance. Consider a regression problem in which the objective is to minimize a squared-loss error. The learner's expected future error for a given instance can be decomposed as in Equation 2.11 [Geman et al., 1992].

$$\begin{aligned} E_{\mathcal{T}} = \quad & E[(y - E[y|x])^2] + \\ & (E_{\mathcal{D}}[f(x)] - E[y|x])^2 + \\ & E_{\mathcal{D}}[(f(x) - E_{\mathcal{D}}[f(x)])^2] \end{aligned} \tag{2.11}$$

where $E_{\mathcal{D}}$ is the expectation over the labeled set $\mathcal{D}$, E is the expectation over the conditional density, that is, $P(y|x)$, and $E_{\mathcal{T}}$ is the expectation over both. $f(x)$ is the model prediction for an instance x and y indicates its true label.

The first term of the decomposition is called noise, and represents the error that is conditioned to the training set. The second term is the bias and represents the difference between the model constructed with $\mathcal{D}$ and the distribution that actually generates y from x. The third term, variance, captures the variability of the model under resampling data sets of fixed size. Minimizing error can then be achieved by minimizing bias, variance or both [Schein and Ungar, 2007].

In [Cohn et al., 1994, 1996] a statistical analysis of the variance estimation is presented. The authors show that it is possible to calculate this value in closed-form for neural networks, Gaussian mixture models and locally-weighted linear regression. In the context of neural networks the output variance for an instance x can be approx-

imated by Equation 2.12 [MacKay, 1992].

$$\sigma^2 \approx \left[\frac{\partial f(x)}{\partial \theta}\right]^T \left[\frac{\partial^2}{\partial \theta^2} S_\theta(\mathcal{D})\right]^{-1} \left[\frac{\partial f(x)}{\partial \theta}\right] \approx \nabla x^T F^{-1} \nabla x \tag{2.12}$$

where $S_\theta(\mathcal{D})$ is the squared error of the current model, with parameters θ, on the labeled set $\mathcal{D}$. The first and third terms represents the gradient of the model's predicted output. The second term is the inverse of a covariance matrix representing a second-order expansion of the objective function with respect to θ, written in shorthand as F. This is also known as *Fisher Information Matrix*, or $I(\theta)$, and represents the amount of information that an observable random variable X (instance) carries about an unknown parameter θ (classifier parameters) upon which the probability of X depends. In this way, to minimize the variance an active learner should select data that maximizes its Fisher Information (or minimizes the inverse thereof).

When the model have K parameters the *Fisher Information Matrix* takes the form of a $K \times K$ covariance matrix. Therefore its minimization is not straightforward and can happen in different ways:

A-optimality: minimizes the *trace* of the inverse information matrix.

D-optimality: minimizes the *determinant* of the inverse information matrix.

E-optimality: minimizes the maximum *eigenvalue* of the inverse information matrix.

A-optimal designs aim to reduce the average variance of parameter estimates and are the most common choice in Active Learning. Other way to build queries using *Fisher Information Matrix* consists in using the Fisher information ratio. Let $I_{\mathcal{U}}(\theta)$ be the Fisher information matrix of the unlabeled pool, and $I_q(\theta)$ the fisher information matrix of a given query q, then the fisher information ratio [Zhang and Oles, 2000] can be defined by Equation 2.13.

$$q = \arg\min_q tr(I_{\mathcal{U}}(\theta) I_q(\theta)^{-1}) \tag{2.13}$$

Querying the set that minimizes this ratio is analogous to minimizing the future output variance once q has been labeled. Thus, reducing the generalization error (with respect to $\mathcal{U}$. [Schein and Ungar, 2007] and [Zhang and Oles, 2000] used this strategy for text classification using a logistic regression algorithm. Hoi et al. [2006a] used this concept in Batch-Mode Active Learning as is further explained in Section 2.3.4.

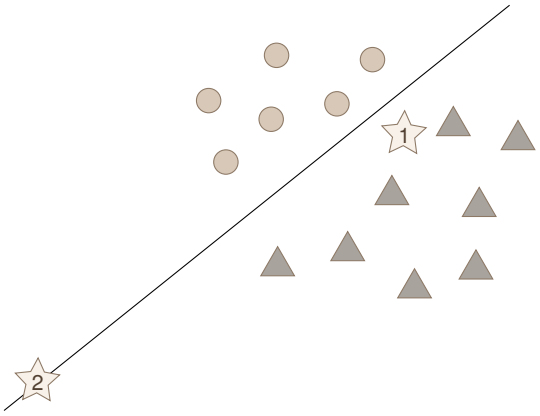


Figure 2.3: Example of scenario in which uncertainty sampling would select a poor query

2.2.5 Density Weighted Methods

Techniques that minimize the estimated error or the variance consider the entire input space when choosing queries. Thus, these strategies are less prone to selecting outliers, that is, instances that lies in sparsely populated regions of the instance space.

Figure 2.3 represents a binary classification problem. The line represents the class boundary estimated by a given classifier. Unlabeled instances are represented by stars. An uncertainty sampling strategy would select instance 2 for labeling since it lies on the classification boundary, although this instance is not a good representative of the data. In a qualitative analysis, selecting instance 1 would lead to better results since it is in a denser region. By utilizing the unlabeled pool when choosing the query, the error reduction and variance estimation techniques implicitly avoid this problem. For other query strategies the input distribution can be modeled explicitly.

In [McCallum and Nigam, 1998], a density-weighted query by committee approach is used. The density is estimated by calculating the average distance of one instance to all other instances. Similarly, Settles and Craven [2008] presents an information density framework to help selecting instances that are not only informative, but also good representatives of the underlying distribution. Let $\phi(q)$ represent the informativeness of q according to a given query strategy, for example, uncertainty sampling, and $sim(a, b)$ be the similarity between instances a and b. Then the density framework can be defined by Equation 2.14.

$$q = \arg\max_{q} \phi(q) \times \left(\frac{1}{|\mathcal{U}|} \sum_{u}^{\mathcal{U}} \text{sim}(q, u) \right)^{\beta} \tag{2.14}$$

In this framework the density is calculated by the average similarity between a

candidate instance and the unlabeled instances. The parameter β is used to control the importance of the density term. Variations of this strategy computes the similarity between the instance and its computed cluster.

In [Fujii et al., 1998], instances that are queried are similar to the unlabeled set and dissimilar to the already labeled set. The main idea is to obtain instances that are ambiguous to a great number of unlabeled instances and, at the same time, different from the already obtained knowledge.

In [Nguyen and Smeulders, 2004], the unlabeled data is clustered and highly populated clusters' instances are prioritized. Similarly, in [Shen and Zhai, 2005] selected instances belong to different clusters. Xu et al. [2007] explicitly calculates a density function by measuring the J-Divergence [Lin, 1991] between instances.

2.3 Active Learning Scenarios

There are three main settings for active learning application. In the **Membership Query Synthesis** the learner may request label for an instance in the unlabeled set or a synthesized query within the application domain (Section 2.3.1). 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 (Section 2.3.2. The last setting is the **Pool-Based Sampling** in which a large pool of unlabeled data is obtained at once (Section 2.3.3). One variation of the Pool-Based Sampling is known as Batch-Mode Active Learning (Section 2.3.4), which is the setting more similar to the one discussed in this dissertation.

2.3.1 Membership Query Synthesis

Firstly, [Angluin, 1988, 2004] studied the problem of using different kinds of queries in order to learn an unknown concept. In a membership query, the learner asks whether a particular domain element belongs, or not, to the unknown concept. In this way, the learner is able to query instances that are not present in the input space, that is, instances that are synthesized *de novo* (i.e. from the beginning). One advantage of membership queries is the theoretical guarantees on the bounds on the number of examples that need to be asked.

Figure 2.4 provides an overview of the Membership Query Synthesis strategy. The model generates queries, based on the data source universal set, that are then presented to the oracle for labeling. The knowledge from the annotated instance is incorporated, and the learner can repeat the process.

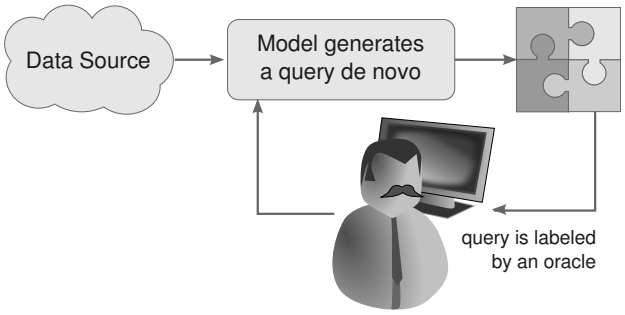


Figure 2.4: Membership Query Synthesis Overview

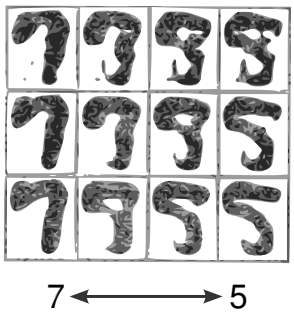


Figure 2.5: Membership Query Synthesis applied to handwritten character recognition

This methodology was successfully applied to different problems. In [Cohn et al., 1996] queries are generated from the input space in order to train a locally weighted regression algorithm. The objective is to learn to predict the position of a two-degree-of-freedom robot arm given both joint angles. The proposed method is able to reduce significantly the Mean Squared Error in comparison to a random sampling strategy.

Another real world application of Membership Query Synthesis is presented in [King et al., 2004, 2009]. A robot scientist, that is a physically implemented laboratory automation system, automatically originates a hypothesis (query) to explain observations, devises experiments to test these hypothesis, physically runs the experiment, interprets the results and then repeats this cycle. This workflow was applied in the investigation of genes encoding one kind of enzymes in the yeast *Saccharomyces cerevisiae*. The method was responsible for a three-fold decrease in material cost compared to the naïve strategy of running the least expensive experiment, and a 100-fold cost decrease compared to randomly generated experiments.

Although Membership Query Synthesis can work very well for applications in which the oracle is an automated process, when the oracle is a human annotator the generated queries can be confusing or impossible to be labeled. For example, Lang and Baum [1992] faced a problem when applying membership query learning for training a neural network for handwritten characters classification. The characters generated for

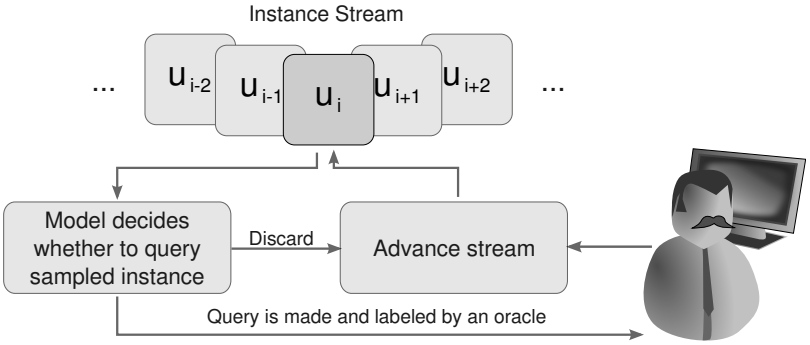


Figure 2.6: Stream-Based Selective Sampling Overview

labeling often were unrecognizable symbols without natural semantic meaning. Figure 2.5 shows possible queries for characters between numbers seven and five.

2.3.2 Stream-Based Selective Sampling

The main idea of Stream-Based Selective Sampling is to filter instances drawn from the input distribution. In other words, it selects, classifies and trains using only those instances that show promise of improving the current model [Atlas et al., 1990]. Instances are sampled one at a time, then this approach can be called *stream-based* or *sequential active learning*. Although Stream-Based Selective Sampling is similar to membership learning, in cases of unknown or non-uniform distributions, it is guaranteed that queries will be reasonable for human annotators given that they are sampled from the real input distribution.

Figure 2.6 presents an overview of the Stream-Based Selective Sampling. Instances are sampled from an input distribution and presented one at a time. The learner should then decide whether to query the oracle for the label or to discard the given instance. The next instance is sampled and the process repeats.

Mitchell [1982] presents a method in which annotated instances are used by the learner in order to generalize the current best hypothesis that models the data. This method can be extended to the stream-based scenario by querying the instance that comes close to matching half of the generalizations currently in the search space. In this way, annotating this instance will allow to discard approximately half of the possible generalizations regardless of its label. Keeping this *candidate set of hypotheses* (sometimes called a *version space*) can be intractable for some predictors. Its size could also grow exponentially with the training set [Haussler, 1989], which motivated the advent of approximate approaches. In Cohn et al. [1994], an approximation for neural networks was presented and applied in power system security analysis, showing

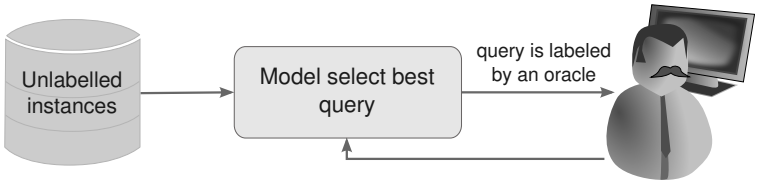


Figure 2.7: Pool-Based Sampling Overview

improvement compared to random sampling. This method was then extended for an agnostic setting in [Dasgupta et al., 2008].

The method proposed in [Beygelzimer et al., 2010] uses an oracle that returns an empirical risk minimizing hypothesis in order to avoid the version space approach. Queries are made in a safe way, in the sense that the hypothesis will eventually converge to the same solution as a passive learning algorithm. This method showed substantial improvement over the passive learner in an intrusion detection dataset.

Dagan and Engelson [1995] used the Query by Committee paradigm [Seung et al., 1992] in its selective sampling strategy. A group of classifiers are generated from a random sample of the training data. Each of these classifiers predict the label of a given candidate instance, which can be queried by the learner with a probability proportional to the degree of disagreement among the committee members. This method was successfully applied to part-of-speech tagging.

2.3.3 Pool-Based Sampling

The Pool-Based Sampling assumes that a large collection of unlabeled data can be gathered at once. The data present in the unlabeled set helps to selectively draw informative instances from the pool [Lewis and Gale, 1994].

Figure 2.7 presents an overview of the Pool-Based Sampling. An instance belonging to the unlabeled set is chosen by the learner in order to be annotated. After this instance is labeled by the oracle, this knowledge is incorporated by the learner, a new query is selected from the unlabeled pool, and the process repeats.

Lewis and Gale [1994] used an uncertainty metric to calculate the value of each instance in the unlabeled pool. The instance with biggest uncertainty is then selected for labeling. This method provided a 500-fold reduction on the training size needed to achieve a specific level of effectiveness in a textual dataset. Other studies were made in the text classification domain, for example, a query-by-committee strategy is applied in [McCallum and Nigam, 1998] and an approximation of the version space method discussed previously is presented for Support Vector Machines [Vapnik, 1995] in [Tong and Koller, 2002]. This idea was also applied for image retrieval [Tong and Chang,

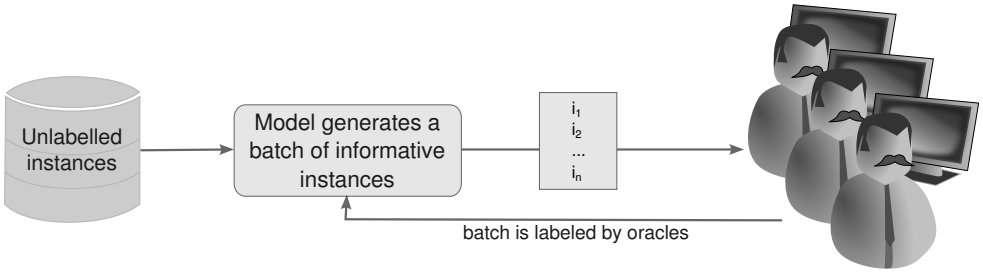


Figure 2.8: Batch-Mode Active Learning Overview

2001].

As aforementioned, Thompson et al. [1999] applied the pool-based scenario in semantic parsing and information extraction. In these domains, the output is structured and the algorithm exploits such a structure. In the semantic parsing, for example, the instances that are queried are the ones that were not successfully parsed or parsed with a low certainty. Settles and Craven [2008] also applied pool-based learning in a structured scenario, the sequence labeling problem.

2.3.4 Batch-Mode Active Learning

In most Pool-Based Sampling scenarios the querying and labeling process occurs one instance at a time. This process is not suitable for scenarios in which the model update is slow or multiple oracles are used in parallel. Batch-Mode Active Learning avoids this issues by allowing the learner to query instances in groups.

Figure 2.8 presents the typical workflow of a Batch-Mode Active Learning algorithm. First, the learner builds a query set $\mathcal{Q}$ containing unlabeled instances. Second, the oracle labels the presented batch. Finally, the new labeled instances are incorporated by the learner. A new batch can be generated and the process repeats.

The challenge in the Batch-Mode Active Learning is how to assemble an optimal query set. Selecting the best instances according to a traditional method is not enough 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.

Brinker [2003] presents a method for incorporating diversity in batches generated using Support Vector Machines. This is achieved by explicitly weighting the instance informativeness (closeness to SVM's decision boundary) with its similarity with instances already selected for the batch. This method could be adapted to be used in

the scenario discussed in this dissertation. In Section 4.7.2 we present a discussion about similarities and divergences between this and our method. In [Xu et al., 2007] instances are selected not only by weighting relevance and similarity but also a density score. The density, as discussed before, aims to avoid querying documents that lie in unimportant, sparsely populated regions.

In [Hoi et al., 2006a], the batch is created by selecting the instances that maximize the Fisher Information of a binary logistic regression algorithm. Let $p(x)$ be the distribution of all unlabeled examples, and $q(x)$ the distribution of unlabeled instances that are selected for labeling. Let I_p and I_q denote the Fisher's Information Matrices of the classification model for the distribution $p(x)$ and $q(x)$ respectively. Then the selected query q is the one that minimizes the ratio between $I_p(\theta)$ and $I_q(\theta)$ as presented in Equation 2.15. Note that θ represents the model parameters of the logistic regression given a labeled set $\mathcal{D}$.

$$q = \arg\min_{q} \operatorname{tr}(I_q(\theta)^{-1} I_p(\theta)). \tag{2.15}$$

The challenge of using this method is the number of possible queries (q) that is exponential with the number of unlabeled examples. This is solved by a greedy near-optimal algorithm [Hoi et al., 2006b] based on the idea of submodular function. Using this method the instances selected to query q will have three important properties: 1) uncertain to the current classification model, 2) dissimilar to the other selected examples, and 3) similar to most of the unselected examples.

Similarly, in [Guo and Schuurmans, 2008], the batch is selected as a whole and not by scoring instances. The batch selection is treated as an optimization problem that aims to learn a good classifier directly. The best set of instances are selected in a way that the generated classifier will attain maximum likelihood with already labeled instances while attaining minimum uncertainty on the unlabeled set. Since it is intractable to conduct an exhaustive search, an approximation is used.

In [Laws et al., 2011; Haertel et al., 2010], a parallel approach is used for applying Active Learning to data being annotated using crowd-sourcing. While the oracle is labeling one query, the learner is able to prepare another batch. In order to reduce idle time, the query can be constructed using stale data. In our work, the batch is not constructed in parallel, however a large enough batch can be generated in order to supply a crowd-sourcing demand.

Xu et al. [2003] used clustering in order do obtain a diverse batch. A Support Vector Machine is used in order to select an initial set of unlabeled instances that lies close to the decision boundary. This set is then clustered and the most representative

(medoid) instance of each cluster is added to the batch.

In [Shi et al., 2012], the Batch-Mode Active Learning setting is expanded to the networked data scenario. Instances are nodes in a graph that can be related through edges. In this way a query batch is constructed aiming to select instances with high uncertainty, high impact (isolated nodes would not have great impact on unlabeled data) and minimum redundancy (instances in a batch should be distinct). These characteristics are weighted in a formula and used to construct the query batch iteratively.

Although the discussed methods allow the use of multiple oracles in parallel, it is still mandatory to continuously update the model and generate a new batch. In a corporative scenario, in which analysts are paid hourly to label instances, continuously updating the model means that analysts' time is being wasted on waiting. In order to adequate the algorithm execution to the corporative schedule a *novel* Active Learning mode is presented in this dissertation. If instances are queried in the form of an arbitrarily large ranking of instances, the model could be updated, for example, once a day, avoiding analysts' idle time.

Batch-Mode Active Learning methods can be divided into two major groups. Methods of the first one construct the batch iteratively, that is, instances are selected one at a time. Methods in the second group construct a batch given its informativeness as a whole, that is, there are no selections of isolated instances. The first group is the one that is closer to the scenario proposed in this dissertation, being easily adaptable for querying a ranking of instances. In this dissertation, a general framework is proposed for weighting diversity and informativeness of instances in order to build a query ranking. This framework can be used for adapting some of the discussed methods to the Ranked Batch-Mode Active Learning scenario. One advantage of our framework is that the parameter that weights factors is explicitly defined, avoiding the necessity of tuning its value.

Methods that choose the batch as a whole are not suitable for the proposed scenario because there is no relative ordering of the instances of the batch. For example, selecting a batch with size equal to the number of unlabeled instances would return the unlabeled set as a query. Since there is no pre-defined labeling order, the oracle would label instances in a random fashion.

2.4 Concluding Remarks

In this chapter we presented a review on Active Learning, as well as a comparison among Batch-Mode Active Learning methods. As discussed, Batch-Mode Active Learning methods are not suitable for generating large batches due to the absence of ordering within the query. This makes these methods not suitable for scenarios with time and budget constraints.

In this dissertation, a general framework for solving the Ranked Batch-Mode Active Learning is proposed and possibilities of using this framework with current methods is later discussed.

Chapter 3

Ranked Batch-Mode Active Learning

In this chapter we present the Ranked Batch-Mode Active Learning problem and compare it to the traditional Batch-Mode Active Learning. We also introduce the framework for solving this problem along with its main components.

3.1 Problem Definition

It is usual for enterprises and research groups that employ Machine Learning algorithms in their workflow, to have a limited budget for content labeling. Using Active Learning can lead to a better use of the available resources, but this could 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:

- **The impossibility to use multiple oracles in parallel.** Only one instance is selected for labeling at each iteration.
- **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 analyzing the Batch-Mode Active Learning 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 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. This can be seen as a tradeoff between the batch quality and its size.

In this context, our work's goal is to create a workflow of document labeling that is more suitable for several business models. For doing so, we present a new Active Learning strategy that generates an ordered batch of instances, in order to make 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 dissertation is the first formalization of this problem.

The Ranked Batch-Mode Active Learning problem relaxes the Batch-Mode Active Learning 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 her own pace.

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

In Figure 3.1, we present the workflow of Ranked Batch-Mode Active Learning 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).

More formally, the Ranked Batch-Mode Active Learning problem is given by Definition 2.

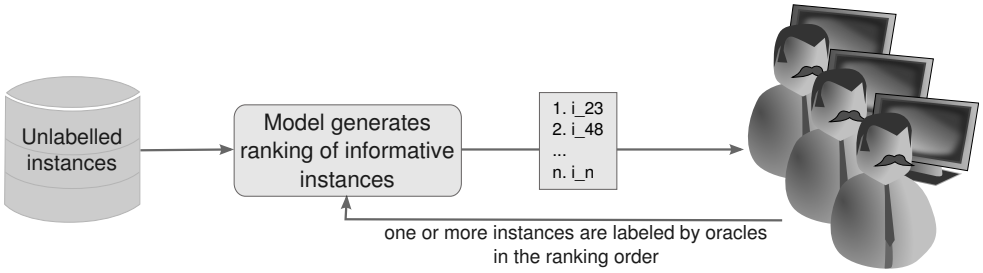


Figure 3.1: Ranked Batch-Mode Active Learning overview

Definition 2 *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 $\mathcal{Q}$ 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 this work, the Ranked Batch-Mode Active Learning is solved by ranking instances using a framework that weights the classifier uncertainty in predicting each instance along with a diversity score. This score is responsible for measuring the difference of feature values between a given instance and the currently selected training set. The next sections contain details about our ranking framework.

3.2 Overview

The proposed method aims to give more flexibility to the Batch-Mode Active Learning scenarios by relaxing the necessity of frequently executing the algorithm. As presented in Figure 3.2, the technique is employed continuously in three steps.

In the first step, *Uncertainty Estimation*, a classifier is trained with all labeled instances 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. The information obtained in the previous step is then used in the *Ranked Batch Construction* stage in which the instances are ranked. The resultant rank is forwarded to the *Labeling by an Oracle*. The oracle labels one ore more instances, in the ranking order, and requests another iteration of the method. The amount of content that must be labeled is not fixed and can be adjusted. Also, at any time, the cycle can restart and incorporate the feedback provided.

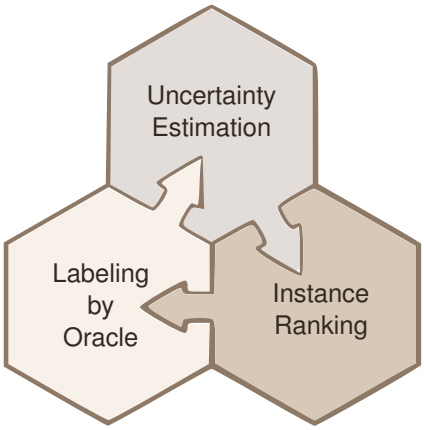


Figure 3.2: Overview of Ranked Batch-Mode Active Learning Framework

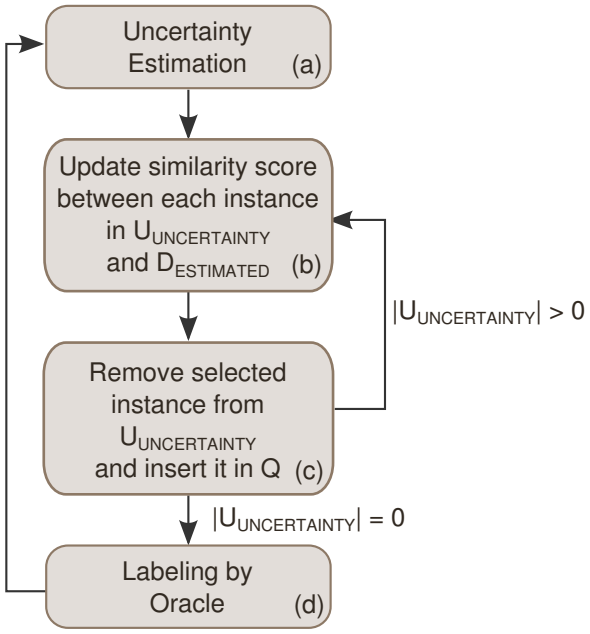


Figure 3.3: Detailed overview of Ranked Batch-Mode Active Learning Framework

Figure 3.3 illustrates this overview. 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 $\mathcal{D}$ at each iteration. During the ranking construction this set represents the expected training set given that all instances already in $\mathcal{Q}$ are annotated. 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 Equation 3.1 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 number of instances is selected.

$$\text{finalScore} = \alpha \times \textit{similarityScore} + (1.0 - \alpha) \times \text{uncertaintyScore}. \tag{3.1}$$

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. 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 in $\mathcal{D}$, 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 3.3. As shown in Chapter 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 3.4.

3.3 Uncertainty Estimation

At each iteration, an uncertainty score is calculated for each instance in the current test 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 test 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. The parts that comprise the *Uncertainty Estimator* are discussed next: the classifier in Section 3.3.1 and the generator of the uncertainty score in Section 3.3.2.

3.3.1 Classifiers

The classifier is the core of the uncertainty estimator. It is expected to provide the probabilities of a given instance belonging to each known class. A classifier that is a good fit is the one that outputs probabilities that are close to the real hit rate. 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 times.

In this work, three different classifiers were considered as the core of the uncertainty estimator: K-Nearest Neighbors (*Knn*), Gaussian Naïve Bayes (*NbGauss*) and *Rocchio*. They were chosen based on the execution speed and the possibility to work with real-valued attributes without the necessity of discretizing its values[1]. This way we avoid the impact of discretization in the obtained results. Each of these classifiers are explained next.

Knn

k-nearest neighbors (or simply, *Knn*) is a type of *instance based learning* (or *lazy learning*) algorithm in which the classification model is approximated locally, that is, the decision boundary is defined considering each training document independently [Cover and Hart, 1967]. The main calculations of the *Knn* algorithm is deferred until

[1]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.

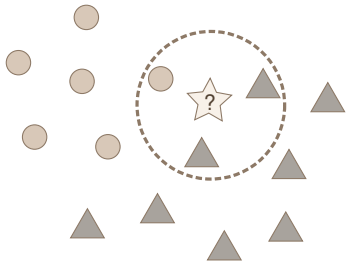


Figure 3.4: Knn classifier example

a test instance is provided. The document with unknown label is then assigned to the majority class among those of its k neighbors.

Figure 3.4 presents an example of classification with $k = 3$. This example is a two-dimensional binary classification problem, that is, the instances are mapped to a two-dimensional plane and there are two classes in $\mathcal{C}$: triangles and circles. Given an unknown instance (star) its nearest neighbors are fetched and its class is predicted as triangle.

The *knn* classifier can output the number of neighbors that were responsible for a given class selection, but the uncertainty estimator needs probabilities for each class. Obtaining the necessary probabilities is a matter of dividing the number of neighbors by the k parameters. In the previous example, the probability of the class being predicted as triangle is the number of nearest neighbors that are triangle divided by the total number of neighbors, $p = 2/3 = 0.66$.

In order to select the nearest neighbors, it is necessary to set the k parameter and use a similarity function that is meaningful to the specific domain. Throughout this work, the *Euclidean*[2] is the similarity function for the *knn* uncertainty estimator and the k value is set to 10. This parameter is set to a fixed value and not defined in a cross-validation process because of the small size of the evaluated datasets. Evaluating the impact of the k parameter in the uncertainty estimation is beyond the scope of this work and instructions for choosing a good value are presented in Section 4.4.1.

The *Knn* algorithm for predicting a class of a test instance is given by Algorithm 2.

Gaussian Naïve Bayes

The Naïve Bayes classifier is a simple probabilistic classifier that applies the Bayes' theorem [Bayes and Price, 1763] for estimating the probability of an instance belonging to a given class, that is, $P(c|t)$ being c a given known class and t a document with

[2]The Euclidean similarity function in discussed in Section 3.4.1

Algorithm 2 K-nearest neighbors prediction

Procedure KnnPredict
Input: A set with manually labeled instances $\mathcal{D}$
Input: A test instance t with unknown label
Input: The number k of nearest neighbors that should be considered
1. sim = EuclideanSimilarity($\mathcal{D}$, t) {compute similarity between t and every instance in $\mathcal{D}$}
2. sim = SortDescending(sim)
3. c, count = CountClassFirstKInstances(sim, k) {determine the most frequent class and its count among top-k instances}
4. p = count / k {calculate confidence in prediction}
5. **return** c, p

unknown class. With the Bayes' theorem, this probability is rewritten as Equation 3.2.

$$P(c|t) = \frac{P(c) \times P(t|c)}{P(t)} \tag{3.2}$$

In order to reduce the complexity of the $P(t|c)$ calculation, the different features (f_i) of t are assumed to be independent, resulting in Equation 3.3.

$$P(t|c) = P(f_0, f_1, \ldots, f_n|c) = P(f_0|c) \times P(f_1|c) \times \ldots \times P(f_n|c) \tag{3.3}$$

This is a rather strong assumption that does not always hold true, thus the name "naïve". Although, in practice, this classifier presents good results [Zhang, 2004].

The estimation of $P(f_i|c)$ can vary in different implementations of the algorithm. In this work, each feature is considered to have a normal distribution with mean (μ) and variance (σ) estimated from the training set [John and Langley, 1995]. Equation 3.4 presents the final estimation formula for $P(f_i|c)$.

$$P(f_i|c) = \frac{1}{\sigma_i \times \sqrt{2\pi}} \times \exp\left(\frac{-(f_i - \mu_i)^2}{2\sigma_i}\right) \tag{3.4}$$

The class that is assigned to t is the one with biggest probability $P(c|t)$. Because the comparison between probabilities defines the class, and the denominator is constant for a given instance ($P(t)$), such denominator can be ignored. Note that this version of the Naïve Bayes classifier assumes that features are normally distributed, which can hinder the performance of the algorithm if the dataset does not hold this assumption.

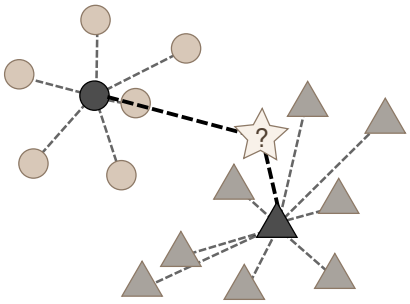


Figure 3.5: Rocchio classifier example

Rocchio

Finally, Rocchio is a relevance feedback process that allows a user query to be modified given its feedback and it aims to provide results that better approximate his interests. This algorithm was later adapted to the classification scenario by Hull [1994].

This classifier uses all information of class membership provided in $\mathcal{D}$ to summarize each known class. This summarization happens in the form of a class centroid that can be, for example, an average of the instance vectors. The prediction process is accomplished by assigning to the test instance the same class of the closest centroid. In order to measure the similarity between a given centroid and an instance, similar to the *Knn* classifier, it is necessary to use a similarity function. Because of its popularity, the *Euclidean* distance is used throughout this work.

Figure 3.5 illustrates the *Rocchio* execution. The example is a binary problem: two centroids are calculated, each being the average position of the instances belonging to its class. The centroids are represented by a dark circle and a dark triangle. Given an instance with unknown class, represented by the star, the algorithm compares the distance between the instance and each centroid. The class of the closest centroid is then assigned, in this case, triangle.

In order to provide the necessary probabilities, the similarity to a centroid can be divided by the sum of similarities with all centroids. This way, the perfect probability is achieved when the instance is in the same position of a given centroid, sharing no similarities with other ones.

The *Rocchio* training procedure used in this dissertation is defined by Algorithm 3. After this model is constructed, an instance with unknown label can be predicted by using Algorithm 4.

Algorithm 3 Rocchio training procedure

Procedure RocchioTrain
Input: A set with manually labeled instances $\mathcal{D}$
1. {compute the average position of instances for each class}
2. centroids = CalculateCentroids($\mathcal{D}$)
3. **return** centroids

Algorithm 4 Rocchio classification

Procedure RocchioPredict
Input: Centroids of each known class
Input: A test instance t with unknown label
1. bestSim = 0.0
2. bestClass = -1
3. simSum = 0.0
4. **for all** $c \in$ centroids **do**
5. sim = EuclideanSimilarity(c, t)
6. simSum = simSum + sim
7. **if** sim > bestSim **then**
8. bestSim = sim
9. bestClass = c
10. **end if**
11. **end for**
12. p = bestSim/simSum {calculate confidence in prediction}
13. **return** c, p

3.3.2 Uncertainty Score

After the probabilities are output by the chosen classifier, they must be converted to an uncertainty score. In this dissertation the Least Confident uncertainty score is used.

This uncertainty score aims to prioritize instances in which the class is predicted with a small confidence. In other words, this score is given by 1.0 minus the probability of the predicted class. Let p_{c_i} be the probability of an instance belonging to class c_i, then the Smallest Probability score can be formally described by Equation 3.5.

$$s = 1.0 - \max_i p_{c_i} \tag{3.5}$$

This is the chosen method for generating the uncertainty score due to three characteristics as follows.

Simplicity: It is straightforward to interpret the metric result, making it easier to identify trends and explain results.

Datasets with different number of classes: Each dataset used in the experimen-

tal evaluation has a different number of classes. It is easier to compare results across datasets when the metric has a similar behavior independent of the number of classes. Entropy, for instance, sums up a probability component for each class available in the dataset.

Minimization of classification error: This metric prefers instances that would help the model to better discriminate among specific classes.

3.4 Instance Ranking

After the Uncertainty Estimation step, the uncertainty score of every instance in $\mathcal{U}$ is calculated. This information can then be 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. This method for computing the similarity between an instance and a given set is described by Algorithm 5.

Algorithm 5 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** (1.0 − biggestSim)

The *SimilarityFunction* used by Algorithm 5 can be any function able to map a

pair of entities from the instance domain ($\mathcal{I}$) to a real number $v : 0 \leq v \leq 1$. 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: $\forall x, y \in \mathcal{I}, f(x, y) \geq 0$
3. Maximality: $\forall x, y \in \mathcal{I}, f(x, x) \geq f(x, y)$
4. Symmetry: $\forall x, y \in \mathcal{I}, f(x, y) = f(y, x)$

Four similarity functions are used in this dissertation: Chebyshev, Cosine, Euclidean and Manhattan. Some of these metrics are actually distance metrics, thus adapted versions were used in order to hold the discussed requirements. The functions used throughout this work are further discussed in Section 3.4.1.

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 a real value that is a weighting between the uncertainty score and current $\mathcal{D}_{\text{ESTIMATED}}$ similarity. The weighting, 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 few instances are labeled, it is better to explore the unknown instance space whereas when a larger training set is available, the uncertainty estimation increases its importance providing better choices.

Algorithm 6 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 α parameter

```
1.  bestScore = −1.0
2.  bestInstance = nil
3.  for all u ∈ U_UNCERTAINTY do
4.     sim = SetSimilarity(D_ESTIMATED, u)
5.     uncertaintyScore = UncertaintyScore(u)
6.     score = α × similarity + (1.0 − α) × uncertaintyScore
7.     if score > bestScore then
8.        bestScore = score
9.        bestInstance = u
10.    end if
11. end for
12. return bestInstance
```

In Algorithm 6, the α parameter is dynamically set based on the training and test set sizes. 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, as defined in Equation 3.6. This parameter is updated every time new labeled content is provided by the oracle.

$$\alpha = \frac{|\mathcal{U}|}{|\mathcal{D}| + |\mathcal{U}|} \tag{3.6}$$

3.4.1 Similarity Functions

The Similarity Function along with the Uncertainty Estimator are the main components of the proposed framework. Different similarity functions are used in this dissertation: Chebyshev, Cosine, Euclidean and Manhattan. Note that the similarity function, as stated before, can vary given the problem domain. For example, in text classification, the *Tf-Idf* Cosine similarity [Baeza-Yates and Ribeiro-Neto, 2011] may be the best choice.

Chebyshev

The 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. In order to obtain a similarity metric with the properties discussed in Section 3.4 the complement of the normalized distance must be used. Let m_i be the maximum difference possible in feature i in a dataset. p_i is the value of feature i in vector p and q_i the value of feature i in vector q. Then the Chebyshev similarity function is defined by Equation 3.7.

$$\text{ChebyshevSim}(p, q) = 1.0 - \max_i \frac{|p_i - q_i|}{m_i} \tag{3.7}$$

Cosine

The Cosine similarity measures the cosine of the angle between two vectors. Since the resultant value can be negative, the function was adapted in order to provide only values between 0 and 1. Let p_i be the value of feature i in vector p and q_i the value of

feature i in vector q. Then the Cosine similarity function is defined by Equation 3.8.

$$\text{CosineSim}(p,q) = \frac{|p.q|}{\parallel p \parallel \parallel q \parallel} = \frac{|\sum_i p_i \times q_i|}{\sqrt{\sum_i p_i^2} \times \sqrt{\sum_i q_i^2}} \tag{3.8}$$

Euclidean

The Euclidean distance is defined as the length of the line segment connecting two points in the Euclidean n-space. Since this is a distance metric it must be adapted to comply with the properties discussed in Section 3.4. The complement of the normalized distance is used as a similarity function. Let p_i be the value of feature i in vector p, q_i the value of feature i in vector q and m the greatest Euclidean distance between two vectors of the dataset. Then the Euclidean similarity function is defined by Equation 3.9.

$$\text{EuclideanSim}(p,q) = 1.0 - \frac{\sqrt{\sum_i (p_i - q_i)^2}}{m} \tag{3.9}$$

Manhattan

The Manhattan distance is the sum of the absolute difference of coordinates. Since this is a distance metric it should be adapted to be used as a similarity function. The complement of the normalized metric is used. Let p_i be the value of feature i in vector p, q_i the value of feature i in vector q and m the greatest Manhattan distance possible in the dataset. Then the Manhatttan similarity function is defined by Equation 3.10.

$$\text{ManhattanSim}(p,q) = 1.0 - \frac{\sum_i |p_i - q_i|}{m} \tag{3.10}$$

3.5 Cold Start

When there is no training data available, that is, $\mathcal{D} = \varnothing$, 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 play 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* [Schütze et al., 2006] 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 can 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 Tomanek et al. [2009].

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 $\mathcal{D} = \varnothing$ is the one that has the highest average similarity with instances in $\mathcal{U}$. The intuition is: the instance that shares more 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 7 presents the method used for choosing the initial seed when no initial training is provided.

In Chapter 4 the presented heuristic for selecting the first instance of the query when no training is provided is evaluated and compared to a random strategy.

3.6 Concluding Remarks

In this Chapter we introduced a new problem: Ranked Batch-Mode Active Learning which aims to relax some of the Batch-Mode Active Learning assumptions: 1) the batch size can be indefinitely large since the query is presented not as an unordered batch but as an instance ranking organized in the order that should be labeled by an oracle; 2) the oracle may also choose when to update the learner's model, making its execution less frequent. The main objective is to allow the use of Active Learning

Algorithm 7 Seed selection

Procedure ChooseSeed
Input: The set of unlabeled instances $\mathcal{U}$

```
1.  bestAverageSim = −1.0
2.  bestInstance = nil
3.  for all u1 ∈ U do
4.    averageSim = 0.0
5.    for all u2 ∈ U do
6.      if u1 ≠ u2 then
7.        sim = SimilarityFunction(u1, u2)
8.        averageSim = averageSim + sim
9.      end if
10.   end for
11.   averageSim = averageSim/(|U| − 1)
12.   if averageSim > bestAverageSim then
13.     bestAverageSim = averageSim
14.     bestInstance = u1
15.   end if
16. end for
17. return bestInstance
```

methods in real world scenarios in which analysts cannot wait for multiple executions of the algorithm in order to annotate unlabeled instances.

The proposed method works by building the query one instance at a time. The appended instance is the one that has a good uncertainty score and brings diversity to the group of already selected or labeled instances. By weighting these factors, it is possible to prioritize diversity on the first iterations and uncertainty in the later ones, which is consistent with the intuition that one should start by having a macro vision of the instance space and should end having a refined vision of the class boundaries regions.

Two main components of the proposed method were also presented: the uncertainty score is calculated given the output of a core classifier, and the diversity score is calculated given a similarity function. There are different combinations of both and it is crucial to choose one that is meaningful to the dataset domain.

Finally, the cold start problem was presented as well as a workaround, in which the first instance is selected by a heuristic that is expected to provide a good initial direction for the instance space search.

Chapter 4

Experiments

This chapter presents the experimental validation of the proposed method along with studies regarding Uncertainty Estimators, Similarity Functions and the strategy used to tackle the cold start

The datasets used in the experimental evaluation are described in Section 4.1. The evaluation process is described in Section 4.2 along with the utilized metrics. An evaluation of Uncertainty Estimators and Similarity Functions is presented in Section 4.4. This section also presents an initial discussion about how to choose these components along with the impact of each factor calculated by a factorial experimental design. Section 4.5 presents a discussion on the proposed strategy for solving the cold start problem. The instance ranking generated by the proposed method is compared to a random ranking strategy in Section 4.6. Section 4.7 presents a comparison between the proposed method and traditional Active Learning algorithms, including a Pool-Based Active Learning and a Batch-Mode Active Learning method.

4.1 Datasets

Datasets from different domains are used in order to assess the performance of the proposed framework. The chosen datasets were selected from the UCI Machine Learning Repository [Frank and Asuncion, 2010] because of its variety of domains and for being well known by the Machine Learning community. These datasets were chosen based on the diversity of applications and on the class and feature values. More specifically, the 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 4.1 details these datasets while Figure 4.1[1] shows the

[1]Note that the scales on the y-axis are different for each dataset

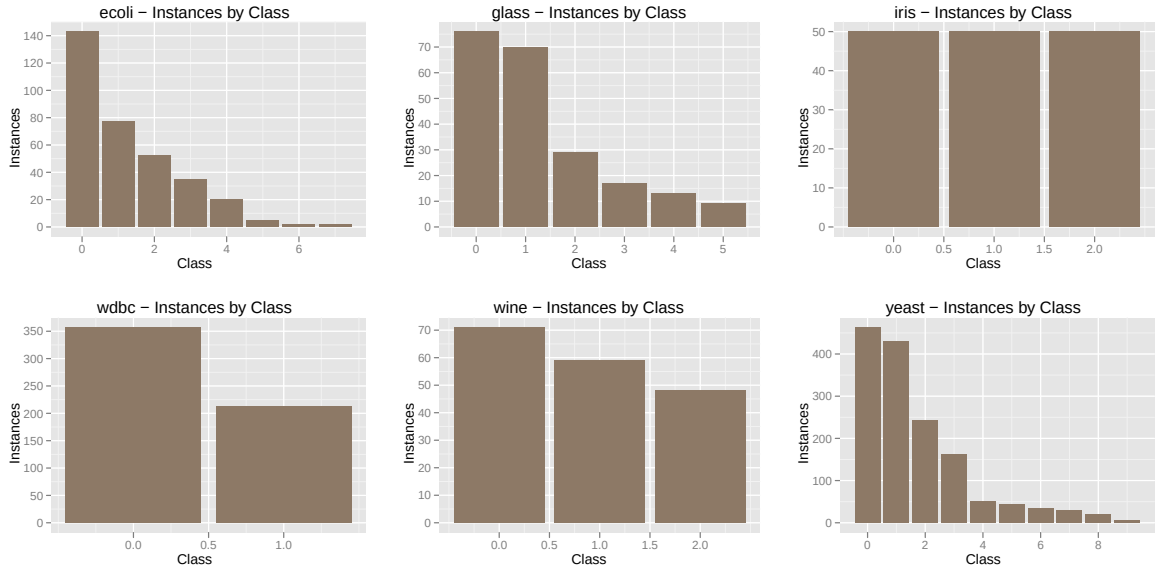


Figure 4.1: UCI Datasets' instances by class

number of instances in each class for all selected datasets. Note that the class distribution in some datasets are very skewed. For example in the *ecoli* dataset, there are two classes (6 and 7) that have only two instances each. Furthermore, the majority of the datasets have less than six hundred instances. These characteristics impact on the way that experiments are conducted as we further explain in Section 4.2.

Table 4.1: UCI Datasets' characteristics.

Dataset	**Description**	**# classes**	**# features**	**# instances**
ecoli	Protein site localization	8	7	336
glass	Glass type identification	6	9	214
iris	Iris plants identification	3	4	150
wdbc	Breast cancer diagnosis	2	30	569
wine	Wine recognition	3	13	178
yeast	Protein site localization	10	8	1484

4.2 Experiment Setup

For assessing the quality of the output ranking, the datasets were randomly split in half multiple times. Each split pair is then used in the following way. The first set is

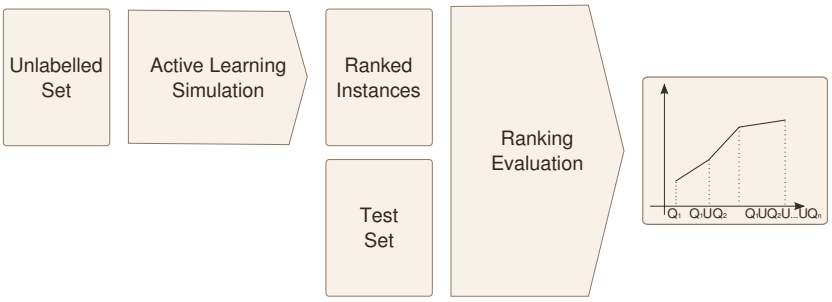


Figure 4.2: Experimental setup overview

the unlabeled set ($\mathcal{U}$) that contains the instances to be ranked or selected. In some of the executed experiments, a few instances were labeled as a seed. In this case, these instances were extracted from this same set. The other set is the test one ($\mathcal{T}$), containing instances to be used only in the evaluation. 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.

Figure 4.2 presents an overview of the general execution of one split. The unlabeled data is fed to an Active Learning algorithm. This algorithm selects which instances should be queried and retrieves their classes. After running one or more times, it returns an instance list with the instances originally in $\mathcal{U}$ ranked in the order that labels should be obtained. 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. The k parameter is selected by a cross-validation on the current portion of the ranking being used as train. This was the chosen classifier, among the ones we tested, due to performance issues and mainly its good effectiveness in the evaluated datasets. Note that this classifier is used in the Ranking evaluation and it is independent of the one used in the Uncertainty Estimator.

Before presenting the actual experiments and results, an overview of the adopted metrics is presented in Section 4.3.

4.3 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 evaluated by calculating its Accuracy and MacroF1. In this work the classification algorithm used for the evaluation is the *KNN* algorithm [Cover and Hart, 1967] with k defined by a cross-validation [Kohavi, 1995] on the set $\mathcal{Q}@x$. This algorithm is used because of its good results and efficiency on the used datasets.

Accuracy

The *accuracy* measures the ratio of instances in $\mathcal{T}$ that were correctly predicted by the classification algorithm. Equation 4.1 presents the *accuracy* formula in the Machine Learning domain.

$$accuracy = \frac{n_{\text{correct}}}{|\mathcal{T}|} \tag{4.1}$$

where n_{correct} is the number of instances in $\mathcal{T}$ correctly predicted by the used classifier. This metric represents how good (in general) the classification is, but it can lead to false conclusions when used in datasets with skewed class distribution. For example, considering a two class dataset with 99 instances of class c_0 and one instance of class c_1. If the classifier predicts the whole set as belonging to class c_0 the resultant accuracy will be 0.99. Although this is a very good value, the problem objective may be to correctly identify instances of class c_1. In this situation the *Accuracy* is not a good indicator of the quality. One way to weight every class equally, despite its size, is by using the *MacF1* metric, described next.

MacF1

Some Information Retrieval ranking measures [Baeza-Yates and Ribeiro-Neto, 2011] were adapted to the Machine Learning context. In this work, specifically, *precision*, *recall* and *F1* are used. In the multi-class classification problem, these metrics are defined for each class in $\mathcal{C}$. Let TP_c be the true positive ratio of a given class c, that is, the number of instances predicted as, and really being, from class c; FP_c be the false positive ratio, that is, the number of instances erroneously predicted as being from class c; and FN_c be the false negative count, in other words, the number of instances belonging to class c that are incorrectly predicted as belonging to another class. This sets are illustrated by Figure 4.3.

Precision is defined as the ratio of instances correctly predicted as belonging to

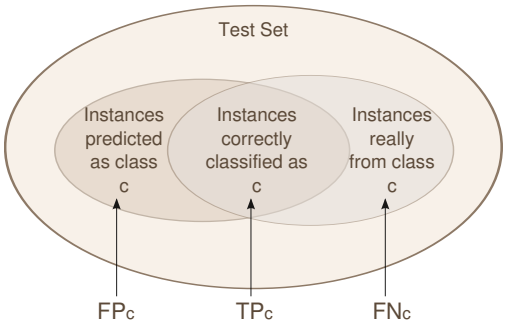


Figure 4.3: Venn diagram of TP_c, FP_c and FN_c

class c by all instances predicted as being of class c, as defined in Equation 4.2.

$$precision_c = \frac{TP_c}{TP_c + FP_c} \tag{4.2}$$

Recall is defined as the ratio of instances belonging to class c that are correctly predicted as being from class c, as defined in Equation 4.3.

$$recall_c = \frac{TP_c}{TP_c + FN_c} \tag{4.3}$$

In order to summarize the *precision* and *recall* values, the metric *F1* can be used. *F1* is defined as the harmonic mean between *precision* and *recall*, as in Equation 4.4

$$F1_c = \frac{2 \times precision_c \times recall_c}{precision_c + recall_c} \tag{4.4}$$

All these metrics are defined for a binary problem (problem with two classes) or for one class in a multi-class problem. In order to summarize the *F1* scores in a single number, the *MacF1* is defined as the average of *F1*s as Equation 4.5.

$$MacF1 = \frac{\sum_{c \in \mathcal{C}} F1_c}{|\mathcal{C}|} \tag{4.5}$$

The *MacF1* gives equal weight to every class, that is, differently from the *accuracy*, misclassification of uncommon classes do heavily impact the final result. In this way, *accuracy* and *MacF1* can be seen as complementary metrics.

Area Under the Curve

The *accuracy* and *MacF1* equations 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. That is, in order to make the method evaluation easier, a quality metric must be associated with the whole ranking Q.

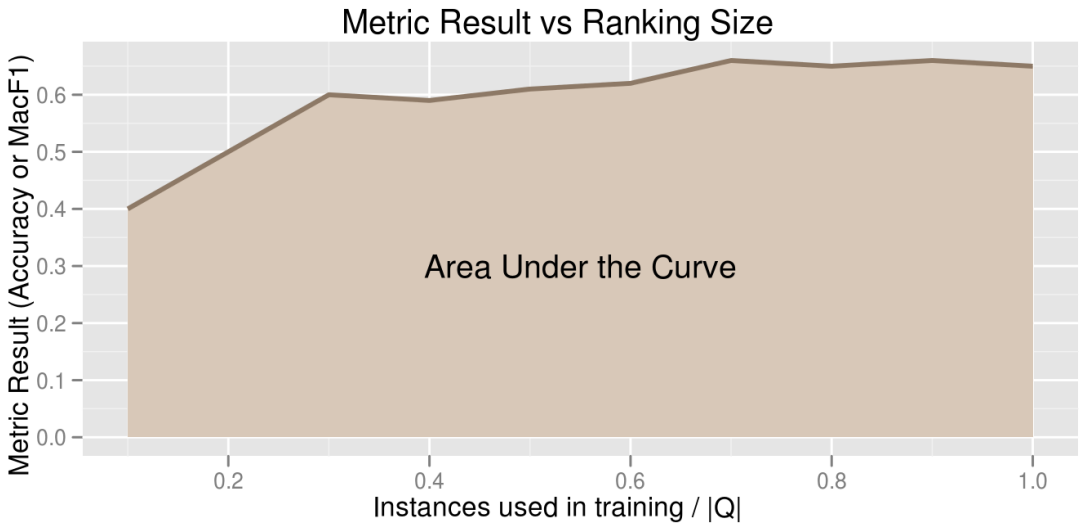


Figure 4.4: Area Under the Curve

A typical result of evaluating $\mathcal{Q}$ can be seen in Figure 4.4. One way to obtain a single number for the execution is to measure the area under the metric curve (the colored area). This method is used for obtaining a single number from other types of curves [Bradley, 1997] 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. Algorithm 8 is used for calculating the Area Under the Curve (AUC).

Algorithm 8 Area Under the Curve

Procedure AreaUnderCurve
Input: The vector x containing the ratio of the ranking used in each metric (x-axis)
Input: The vector y containing the metric result of each measure (y-axis)
Input: The number of observations n

1. auc $= 0.0$
2. $i = 1$
3. **for** $i < n$ **do**
4. base1 $= y[i-1]$
5. base2 $= y[i]$
6. $h = x[i] - x[i-1]$
7. trapezoidArea $= \frac{\text{base1} + \text{base2}}{2} \times h$
8. auc $=$ auc $+$ trapezoidArea
9. **end for**
10. **return** auc

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 $\mathcal{Q}$.

4.4 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 problem domains and possible complexity constraints. In this section, we present how to choose good components for a given problem and analyze the impact of such a choice.

4.4.1 Uncertainty estimator

A good uncertainty estimator should provide a probability for each prediction that is 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 in bins. In this work, three estimators are compared by the mean squared error (*MSE*) across each bin. The *MSE* is defined in Equation 4.6.

$$MSE = \frac{\sum_{b=0}^{n-1}(e(b) - p(b))^2}{n} \tag{4.6}$$

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. This way, 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 in new training and a validation sets. The new training set is used to train a given estimator, and the validation set is used to calculate the correct predictions ratio. The test set is used in order to verify the relation between the calculated *MSE* and the result of the proposed method execution.

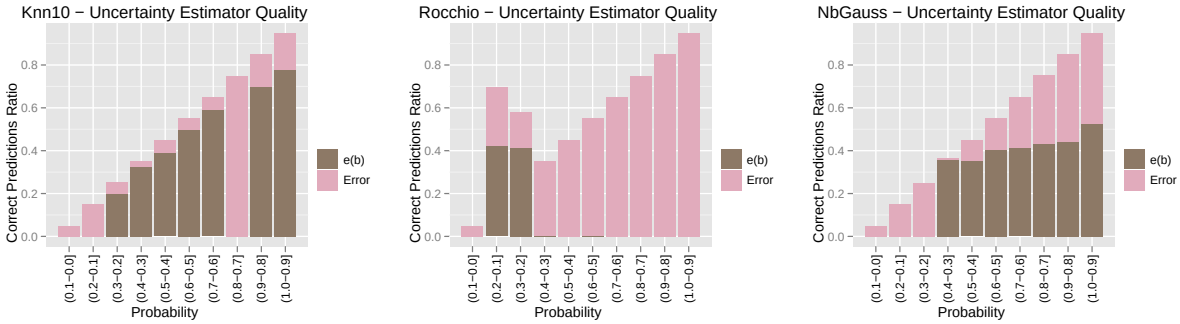


Figure 4.5: Uncertainty estimator quality for the *glass* dataset

In this work, three 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; and a Gaussian Naïve Bayes (*NB Gauss*). In a real world scenario, k can be selected by considering the classifiers trained with different parameters as different uncertainty estimators.

Figure 4.5 presents one sample of how the *MSE* is calculated. The bar plot ($e(b)$) represents the probability that the classifier made a correct prediction, and the Error is the distance to the perfect model, that is $|e(b) - p(b)|$. In the dataset *glass*, the *Knn* is the one that gets closer to the ideal model whereas the *Rocchio* classifier tends to provide probabilities that are not really meaningful. This qualitative evaluation is confirmed by the calculated *MSE*.

Table 4.2: Mean Squared Errors of uncertainty estimators

Dataset	**Knn10**	**Rocchio**	**NBGauss**
ecoli	0.003184 ± 0.002294 ▲	0.371932 ± 0.001193	0.064247 ± 0.019010
glass	0.009501 ± 0.000391 ▲	0.101414 ± 0.045563	0.075931 ± 0.000874
iris	0.012740 ± 0.004541	0.276062 ± 0.005645	0.005315 ± 0.000343 ▲
wdbc	0.004390 ± 0.000300 ▲	0.144769 ± 0.002631	0.045327 ± 0.001610
wine	0.014470 ± 0.000551 ▲	0.341014 ± 0.009549	0.023330 ± 0.012135
yeast	0.007522 ± 0.000088 ▲	0.124583 ± 0.000296	0.038520 ± 0.001047

Table 4.2 presents the calculated *MSE* between the obtained probabilities and the perfect model as well as the confidence intervals with 95% of confidence. The estimators that are significantly better are marked with a ▲. In order to evaluate if the *MSE* is a good measure of the 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 4.3: AUC-MacF1 of different uncertainty estimators

Dataset	Knn10	Rocchio	NBGauss
ecoli	58.04345 ± 2.03743 ●	54.14583 ± 1.77268	58.12891 ± 1.96621 ●
glass	46.07579 ± 1.76127 ●	44.24015 ± 2.34076 ●	36.82012 ± 1.70224
iris	91.05253 ± 1.17847 ▲	69.59344 ± 1.93440	87.14806 ± 2.25009
wdbc	90.01343 ± 1.20737 ▲	87.38219 ± 1.79822	88.92472 ± 1.24292
wine	91.00987 ± 0.80863 ●	91.66246 ± 0.80805 ●	91.32646 ± 0.96706 ●
yeast	43.30552 ± 0.83688 ▲	30.53398 ± 1.33456	39.54735 ± 0.76823

Table 4.3 presents the *Area Under the Curve* of the *MacF1* metric and the 95% confidence intervals. The estimator with significantly better result is marked with a ▲, when more than one algorithm does not show statistical difference, with 95% of confidence, then the bullet mark (●) is used. As can be seen, the *MSE* results are endorsed by the *AUC-MacF1*, being *Knn10* the best uncertainty estimator in most datasets.

Two datasets, though, are outliers in this experiment. The first one is the *iris* dataset, in which the *NbGauss* provides the smaller *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. By calculating the ratio between the biggest and the smallest *MSE* it is possible to clearly see such a difference. Equations 4.7 and 4.8 present the ratio for the *ecoli* and the *iris* dataset.

$$MSERatio_{ecoli} = \frac{MSE_{NbGauss}}{MSE_{Knn10}} = \frac{0.064247}{0.003184} = 20.178077 \tag{4.7}$$

$$MSERatio_{iris} = \frac{MSE_{Knn10}}{MSE_{NbGauss}} = \frac{0.012740}{0.005315} = 2.396989 \tag{4.8}$$

The other interesting case occurs in the *wine* dataset. Whereas there are big differences on the estimators' *MSE*s, 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 quickly (with few instances the metric achieves a stationary point). In the first scenario a bad estimator, that assign scores randomly,

will lead to selections close to the random sampling which may be a good choice given the dataset. In the second scenario the quick stabilization of the metric shows that adding more instances of the given dataset will not greatly impact the final result. If this point is achieved with few instances than the resultant AUC will not be very different. The latter is the case of the *wine* dataset. Throughout this work, we use *Knn10* as the uncertainty estimator due to its small average *MSE*.

4.4.2 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 the 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) [Manning et al., 2008] in different datasets.

The 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 $\{d_1, d_2...d_m\}$ and n_i the total number of documents retrieved until document d_i. Then the Average Precision (AP) is defined by Equation 4.9.

$$AP = \frac{\sum_{i=1}^{m} p_i}{m} \tag{4.9}$$

where p_i is defined by Equation 4.10.

$$p_i = \frac{i}{n_i} \tag{4.10}$$

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 [Heuser et al., 2007].

Although the 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 good MAP value for queries of all classes, thus a signal-to-noise ratio (SNR = μ/σ) 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 Equation 4.11

presents the formula for the AP_{SNR}.

$$AP_{\text{SNR}} = \frac{\mu(AP)}{\sigma(AP)} = \frac{MAP}{\sigma(AP)} \tag{4.11}$$

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. This way 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 4.4: 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 ▲

Table 4.4 presents the results of the AP_{SNR} for different similarity metrics as well as the confidence intervals with 95% of confidence. The similarity function with significantly better AP_{SNR} is marked with a ▲. In order to evaluate the relation between this quality metric and the method result, the *AUC-MacF1* of the generated ranking was calculated using the test split.

Table 4.5: 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 ●

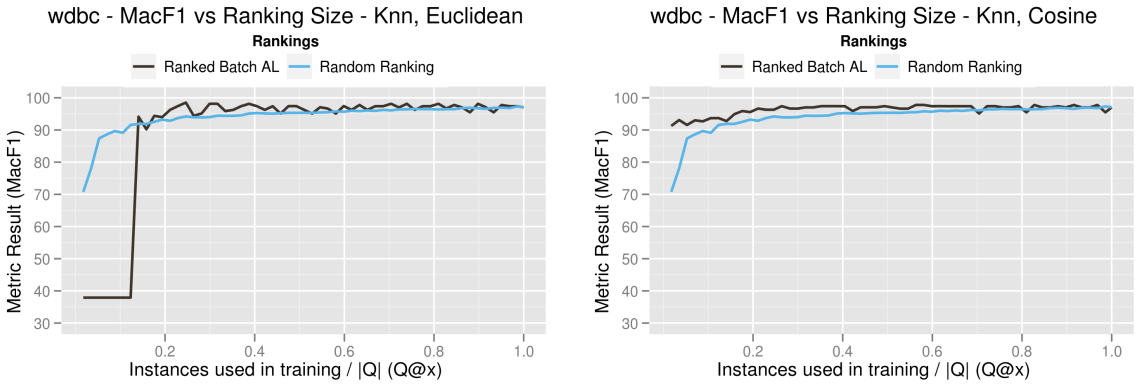
Table 4.5 presents the *Area Under the Curve* of the *MacF1* metric and the 95% confidence intervals. The similarity function with significantly better result is marked with a ▲, when more than one algorithm does not show statistical difference, with 95% of confidence, then the bullet mark (●) is used. 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, although in order for these results to be conclusive, further experiments are necessary.

Table 4.6: 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

Another sign of the relation between the AP_{SNR} and the method result is the Pearson correlation between these two values presented in Table 4.6. 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 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 known, thus hindering the classification performance. In this scenario, a naïve similarity function can make a better choice, for example, a random choice would lead to the knowledge of both classes sooner.

This characteristic can be better visualized in Figure 4.6. In the beginning of the ranking constructed using the *Euclidean* similarity (Figure 4.6a), 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 (Figure 4.6b) is able to discover both classes quicker leading to better classification models early on.



(a) Euclidean similarity function

(b) Cosine similarity function

Figure 4.6: Comparison of the *wdbc* dataset result when using different similarity functions

Given the presented results, throughout this work, unless noted otherwise, the *Cosine* similarity function will be used due to its small average AP_{SNR}.

4.4.3 Impact of each component

One way to evaluate the impact of different components on the final method result is by executing a $2^k r$ factorial design [Jain, 1991]. In the experimental design, 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*).

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 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 in this work, the $2^2 r$ factorial design can be summarized as follows.

1. Each factor is associated to a variable (x_A and x_B) representing the lower and higher level:

$$x_k = \begin{cases} -1 & \text{if factor k assumes its lower level,} \\ +1 & \text{if factor k assumes its higher level.} \end{cases}$$

2. The response variable y is regressed on x_A and x_B values using a nonlinear regression model of the form:

$$y = q_0 + q_A x_A + q_B x_B + q_{AB} x_A x_B + e$$

where e is the experimental error and the q's are the effects of given factor or factor combination.

3. The effects q_0, q_A, q_B and q_{AB} are determined by expressions called *contrasts*, which are linear combinations of average responses $\overline{y}_i$ calculated based on multiple observations of each possible combination of the variables. Being x_{Ai} and x_{Bi} levels of x_A and x_B respectively, the observation is modeled as:

$$\overline{y} = q_0 + q_A x_{Ai} + q_B x_{Bi} + q_{AB} x_{Ai} x_{Bi}$$

4. Once the effects have been computed, the model can be used to estimate the response for any given factor values. The difference between the estimate ($\hat{y}_{ij}$) and measured value (y_{ij}) in the jth replication of the ith experiment represents the experimental error: $e_{ij} = y_{ij} - \hat{y}_{ij}$. The sum of the squared errors (SSE) can then be used to estimate the variance of the errors and also the confidence intervals for the effects:

$$\text{SSE} = \sum_{i=1}^{2^2} \sum_{j=1}^{r} e_{ij}^2$$

5. The importance of a factor is measured by the proportion of the total variation in the response that is explained by the factor. In order to calculate this proportion, it is first, necessary to calculate the total variation of y, or the Total Sum of Squares (SST):

$$\text{SST} = \sum_{i,j} (y_{ij} - \overline{y})^2 = 2^2 r q_A^2 + 2^2 r q_B^2 + 2^2 r q_{AB}^2 + \sum_{i,j} e_{ij}^2$$

where $\overline{y}$ is the mean of responses from all replications of all experiments. This expression can then be rewritten, replacing each sum of square block with a different notation:

$$\text{SST} = \text{SSA} + \text{SSB} + \text{SSAB} + \text{SSE}$$

6. Each of the presented SS (sum of square) represents the variation explained by a given effect of experimental error. Thus, the *fraction of variation* explained by factor k is given by:

$$k = \frac{\text{SS}k}{\text{SST}}$$

7. Assuming that errors are normally distributed with zero mean, then the variance of errors can be estimated from the SSE as follows:

$$s_e^2 = \frac{\text{SSE}}{2^2(r-1)}$$

Finally, the confidence intervals can be obtained by calculating the variance of each effect as follows:

$$s_{q_0} = s_{q_A} = s_{q_B} = s_{q_{AB}} = \frac{s_e}{\sqrt{2^2 r}}$$

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. This way, 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. This process is used in each dataset.

For the uncertainty estimator, the high level is considered to be the one with smallest *MSE* (Section 4.4.1) and the low level is the strategy that assigns a random score for each instance. The second factor has the high level set as the similarity function with the highest AP_{SNR} (Section 4.4.2) with the low level being, similarly to the uncertainty estimator, a random strategy that returns a random value for each instance pair. In this way, it is possible to assess the impact of the proposed components in the final result in comparison to a naïve strategy that outputs random scores. Given

the small *MSE*, the high level of the Uncertainty Estimator will be fixed as being the *Knn10* classifier and the high level of the Similarity Function is fixed as the *Cosine* one given its AP_{SNR} average value.

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 4.7 presents the amount of the variation that is explained by each factor, being factor A the uncertainty estimator and B the similarity function.

Table 4.7: 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

The factorial experiment results confirm the intuition in the previous experiments. The final impact of the uncertainty estimator is usually higher than the impact of the similarity function. This means that it is usually better to invest in improving the estimation of the uncertainty. Two datasets show peculiar results: First, the *iris* dataset that presents high error because the classification quality, in the ranking evaluation, is highly dependent of the estimated parameter k. Second, the *wine* dataset in which a big part of the final result is explained by a combination of both factors. This happens because the response variable have similar values regardless of the component used.

4.5 Evaluation of the Initial Selection

The method proposed in this dissertation 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 3.5). In this work, this problem is overcome by selecting the first instance as the one that is the best representative of the dataset, that is, the one that has more similarity with the instances 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. This set of experiments is divided as follows. Section 4.5.1 compares the quality of a single ranking generated with a random initial selection versus the proposed heuristic. Section 4.5.2 analyzes a similar scenario, but the generated ranking is initiated with 20% of random instances. Finally, in Section 4.5.3 the impacts of the random and the heuristic initial selection strategies are studied when instances are labeled by the oracle and the model is iteratively updated.

4.5.1 One Random Instance as Initial Selection

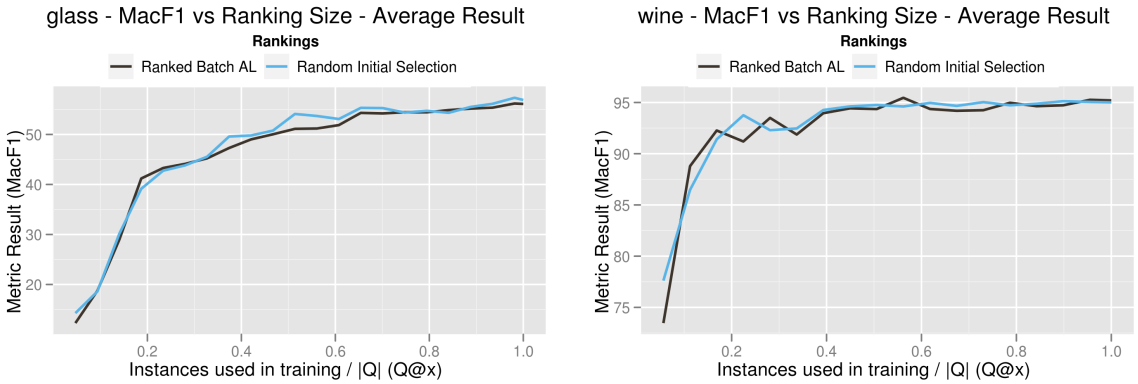
In this experiment, the proposed heuristic for initial selection is compared to a random strategy. In the random strategy, the ranking is started with a random instance. As discussed before, a bad start can hinder the performance of the whole generated ranking.

Table 4.8: 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

Table 4.8 presents the comparison between the proposed heuristic for selecting the first instance versus selecting one random instance. Results indicate that there is



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

Figure 4.7: *MacF1* comparison of the proposed heuristic for initial selection versus starting with one random instance

no substantial difference between the strategies. The random strategy has, though, the advantage of not requiring a big computational cost like the "most common" instance selection method. Such similar results of the discussed strategies can be better visualized in Figure 4.7. The results for the generated ranking are presented for two datasets: *glass* and *ecoli*. 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.

4.5.2 20% of the Unlabeled Instances as Initial Selection

Another discussed strategy for building the beginning of the ranking when no labeled instances are provided is the random selection of a group of instances. In this 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.

Table 4.9 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 significative improvement).

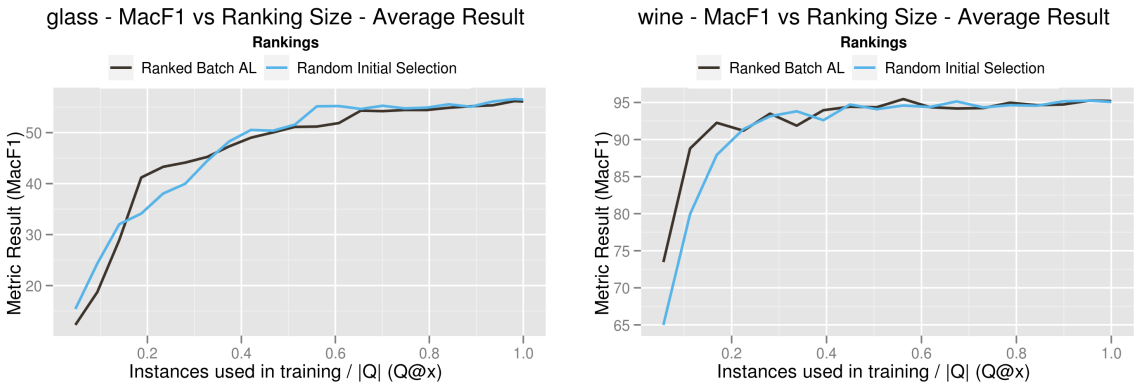
In general, the heuristic strategy presents results that are statistically equal or better than the random initial selection. This indicates that the initial selection should

Table 4.9: 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

be small (for example, one instance), and the proposed method should be used for constructing the ranking normally.



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

Figure 4.8: *MacF1* comparison of the proposed heuristic for initial selection versus starting with 20% of random instances

Figure 4.8 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 Figure 4.7 when selecting an initial random set.

4.5.3 Impact of the Instance Selection Strategy Throughout the Ranking

Although the proposed heuristic and random strategy show similar results on the first generated ranking, it is important to evaluate the impact throughout the ranking. As discussed before, a bad start can hinder the performance of Active Learning algorithms in subsequent choices.

In this experiment, the first instance is selected by one of the discussed techniques, its label is queried and labeled. 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.

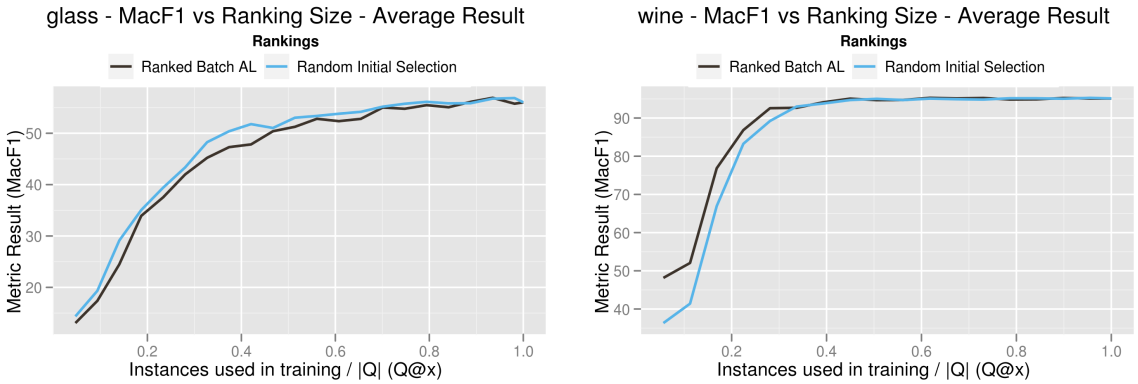
Table 4.10: Comparison between the proposed heuristic for initial selection versus choosing a random instance with one instance being labeled per iteration

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

Table 4.10 presents the results for the ranking generated with one instance labeled per iteration. As can be seen, usually, there is no difference between the two strategies. The proposed heuristic though was able to achieve a 22% improvement in the *AUC-MacF1@20%* on the *wine* dataset. Although the differences in averages of the different methods are not statistically significant with 95% confidence, the proposed method achieves 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.

Figure 4.9 presents the comparison between the strategies on the *glass* and *wine* datasets. In the dataset with the worst performance of the heuristic (Figure 4.9a), the behavior of both curves are very similar. On the other hand, in the *wine* dataset it



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

Figure 4.9: *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.

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.

Since the proposed heuristic shows a higher average for every metric, it will be used for choosing the first instance of the ranking when no training set is provided. Although, in a real world scenario this can be substituted by the random selection strategy in order to avoid the additional computational cost.

4.6 Rank Quality Evaluation

In this section the quality of the generated ranking is compared to the random selection strategy. Two different scenarios are studied. First, the full instance ranking is generated without a provided seed and considering that no label is provided by the oracle (Section 4.6.1). Second, the proposed framework is used to generate a full ranking when half instances are provided as seed, but no further instances are fed by the oracle (Section 4.6.2). Both scenarios capture how the classification quality increases, considering that only one ranking is generated and no feedback is given to the algorithm. In other words, the idea is to evaluate the quality of $\mathcal{Q}$ given $\mathcal{D}$ and $\mathcal{U}$.

4.6.1 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. Although counter-intuitive, the random sampling strategy can be a strong baseline given the dataset characteristics [Settles, 2009]. This happens because the random strategy provides an unbiased sample that obeys the underlying probability distribution.

A ranking $\mathcal{Q}$ is generated using the whole unlabeled set $\mathcal{U}$. This ranking is then used for evaluating 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.

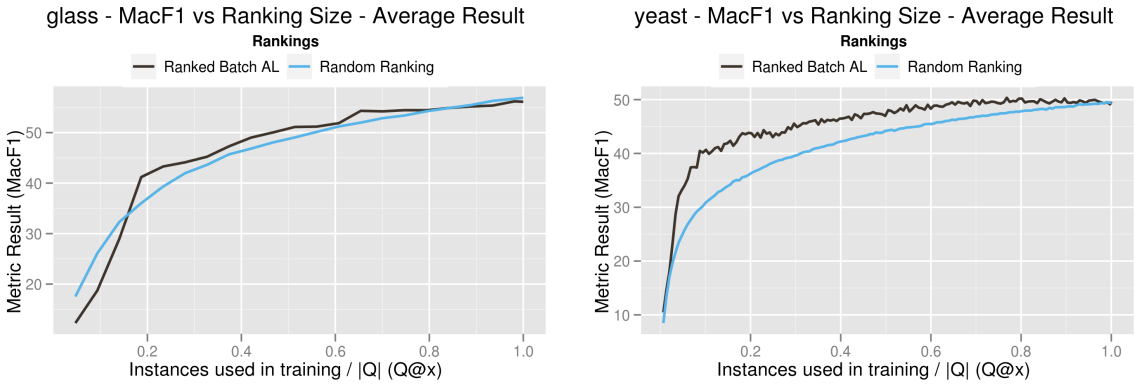
Table 4.11: Resultant ranking quality when no training is provided Knn Cosine

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

Table 4.11 presents the summary for each dataset. Four metrics are presented for each dataset. The first two are the areas under the *Accuracy* and *MacF1* curves. Usually the objective is to achieve a good classifier with few instances as possible, thus two other metrics are used consisting of the *AUC* on the top 20% of the final ranking.

The proposed method presents gains in the *AUC-MacF1* and *AUC-Accuracy* for most datasets when compared to the random strategy. In the *yeast* dataset, the improvement of the *AUC-MacF1* comes at the expense of reducing the *Accuracy* in the beginning of ranking, although this difference is compensated throughout the ranking. More precisely, the gain in the *AUC-MacF1* is approximately 9.6% whereas the *AUC-Accuracy@20%* loss is approximately 3.1%.



(a) *glass* (b) *yeast*

Figure 4.10: *MacF1* results for ranking generated without training data

The generated ranking has a bad start in the *glass* dataset which it is not able to compensate. The rest of the ranking stays a little above the random result, but this start is enough to hinder the *AUC* values. The quality of the rest of the ranking can be assessed by the *AUC-MacF1* that is statistically equivalent to the random one. This hypothesis is endorsed by the resultant chart of *MacF1* versus the number of instances in $\mathcal{Q}@x$. As shown in Figure 4.10a the curve displays the expected behavior.

In the *yeast* dataset, our method provided significant gains over the random curve. Figure 4.10b shows the resultant curve in these situations. 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.

Finally, without any training data the ranking generated by our method is able to achieve a statistically significant improvement in the *AUC-MacF1* datasets average while having results no statistically different from the baseline in the other metrics.

4.6.2 Ranking With Training Data

Despite the possibility to generate rankings even when no training data is provided, it is expected that the query ranking $\mathcal{Q}$ increase its quality given the increase of knowledge contained in $\mathcal{D}$. This experiment aims to compare the ranking generated with the proposed method with the randomly generated ranking when labeled data is provided. Specifically, half of the instances are provided with theirs labels ($\mathcal{D}$). This information is then used to build a ranking, and the result is compared to the random sampling strategy.

The initial train split is divided once again. Half instances are then used as the set

$\mathcal{D}$, that is, these documents are provided to the algorithm with the respective 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 train 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 4.12: Resultant ranking quality when training is provided Knn Cosine

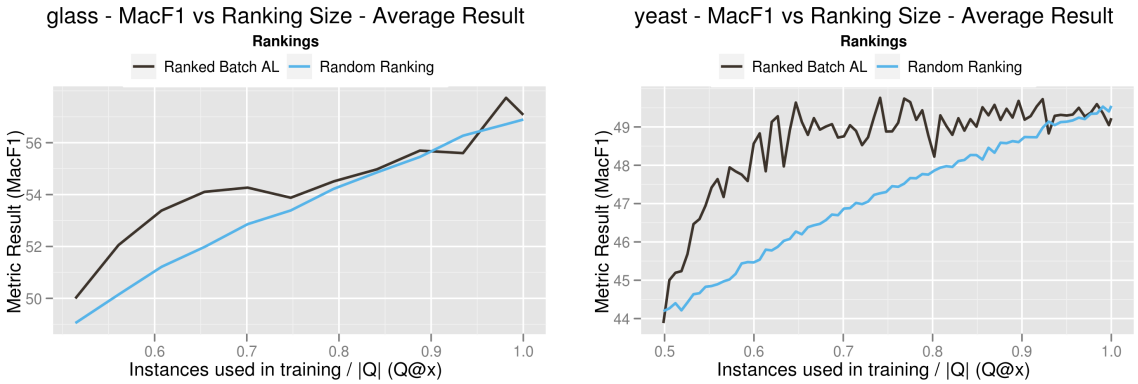
Dataset	**AUC-Acc**		**AUC-Acc@20%**		**AUC-MacF1%**		**AUC-MacF1@20%**	
	Random	RBMAL	Random	RBMAL	Random	RBMAL	Random	RBMAL
ecoli	82.51	83.24 ▲	81.39	81.48 ●	62.88	64.58 ▲	60.44	62.54 ▲
glass	61.69	62.16 ●	59.49	60.43 ●	53.46	54.12 ●	50.13	51.87 ▲
iris	94.48	94.92 ▲	93.70	94.01 ●	94.38	94.83 ▲	93.59	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

As before table 4.12 presents the results summary for each dataset. Four metrics were calculated: *AUC-Accuracy*, *AUC-Accuracy@20%*, *AUC-MacF1* and *AUC-MacF1@20%*.

As the number of labeled instances increase, the random baseline gets harder to overcome because it provides an unbiased sample of the underlying probability distribution of the data. The difference between the calculated and the random ranking also gets narrower because the metrics tend to the values obtained when all available data is used. As show in Table 4.12, even with these hindrances, the proposed method is able to achieve statistically significant improvements in almost all datasets. This is reflected in the average of each metric, that is statistically better than the baseline with 95% confidence. In the *glass* dataset, although there is no statistically significant difference, the calculated method has higher average in all metrics. This indicates that the proposed method continues to create rankings better than the random selection as new labels are supplied.

Figure 4.11 shows the average curves for the *glass* and *yeast* datasets for the



(a) *glass* (b) *yeast*

Figure 4.11: *MacF1* results for ranking generated with training data

MacF1. As can be seen, the calculated ranking usually leads to *MacF1* results that are close or better than the random one.

4.7 Traditional Active Learning Scenarios

Considering the presented method is flexible enough for working in traditional Active Learning scenarios, a deeper study on the behavior of the method is presented in the following sections. Section 4.7.1 compares our strategy with traditional Active Learning methods in which one instance is queried and labeled at each iteration. A comparison with traditional Batch-Mode sampling strategies is presented in Section 4.7.2, in this set-up methods query k instances at each iteration. In both cases the idea is to simulate the use of the proposed method in these well known scenarios.

4.7.1 Pool-Based Sampling Comparison

In this section, our method is compared to a traditional Active Learning algorithm. Two different experiments are executed. First, the proposed method generates a full instance ranking without any provided label while the baseline method is updated with labels at each iteration. Second, the proposed method, similarly to the baseline, is fed with one label at each iteration.

In both sections 4.7.1.1 and 4.7.1.2 the baseline method uses uncertainty sampling for choosing which instance to query and can be described by Algorithm 9.

The *TrainClassifier* method builds a *Knn10* classifier, which is the same algorithm that composes the proposed method's uncertainty estimator in this experiment.

Algorithm 9 Uncertainty sampling Active Learning

Procedure TraditionalActiveLearning
Input: The set with manually labeled instances $\mathcal{D}$
Input: The set of unlabeled instances $\mathcal{U}$

```
1. mostUncertain = 0
2. mostUncertainScore = 0.0
3. classifier = TrainClassifier(D)
4. for all u ∈ U do
5.    uncertainty = ClassifierUncertainty(classifier, u)
6.    if uncertainty ≥ mostUncertainScore then
7.       mostUncertainScore = uncertainty
8.       mostUncertain = u
9.    end if
10. end for
11. L = WaitForOracleLabel(mostUncertain)
12. D = D + L
13. U = U − L
14. return (D,U)
```

Considering that the same algorithm is used, it is possible to derive insights about the impact of adding the diversity and the α parameter to the sampling strategy.

4.7.1.1 Ranking Generated Without Labels

In this experiment, the first ranking generated by the proposed method (without training data) is compared to the ranking generated by the baseline method considering that the whole unlabeled set is labeled by an oracle iteratively, one instance at a time.

Table 4.13 presents the results of the baseline and the proposed method. Our method without any labels provided is capable of ranking instances in a way that the resultant classifiers are close to the ones trained with the active learning generated rank. When the beginning of the ranking is considered (*AUC-MacF1@20%*) the results can be even better then the baseline. For instance, the *glass* dataset presents the worst performance, with 15% decrease in the *AUC-MacF1@20%* while the *ecoli* dataset shows a 15% increase in the same metric.

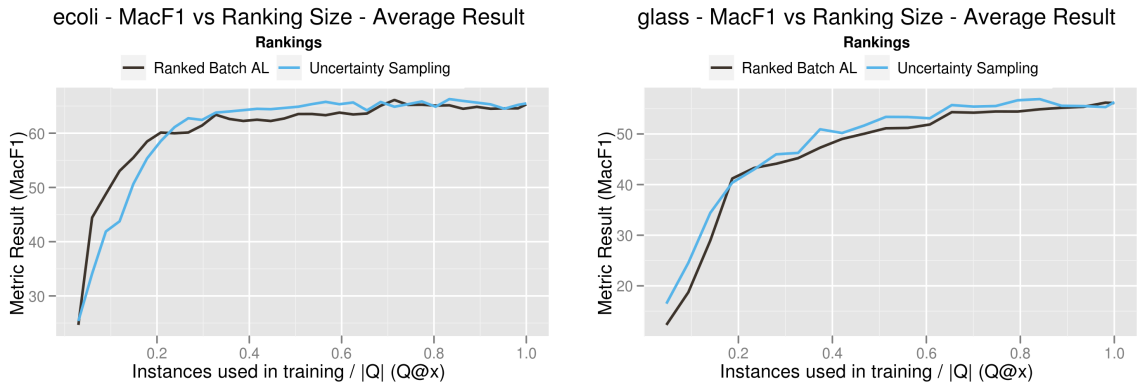
Note that the baseline model is updated with one additional label at each step, while the proposed method's ranking is generated only using the unlabeled set. This means that with one iteration, without any labeled data, the proposed method is able to generate a ranking that has competitive results with the baseline, but being much more effective.

It is clearer to observe the proximity between both ranks by analyzing Figure

Table 4.13: Comparison between the proposed method (no labels provided) and the uncertainty sample strategy

Dataset	AUC-Acc		AUC-Acc@20%		AUC-MacF1%		AUC-MacF1@20%	
	Baseline	RBMAL	Baseline	RBMAL	Baseline	RBMAL	Baseline	RBMAL
ecoli	80.78	80.26 ●	70.02	71.01 ●	60.93	61.12 ●	42.14	48.66 ▲
glass	57.43	55.10 ▼	43.02	35.59 ▼	48.85	47.13 ▼	29.12	24.80 ▼
iris	93.06	92.09 ▼	84.25	85.87 ●	92.71	91.81 ▼	82.40	84.86 ●
wdbc	94.84	94.31 ▼	90.93	90.31 ●	94.39	93.83 ▼	90.01	89.36 ●
wine	92.90	93.22 ●	82.75	87.21 ▲	92.64	93.06 ●	80.57	85.82 ▲
yeast	49.40	48.98 ●	43.76	42.91 ●	46.21	45.71 ▼	34.88	36.52 ▲
average	78.06	77.32 ●	69.12	68.81 ●	72.62	72.11 ●	59.85	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) *ecoli* (b) *glass*

Figure 4.12: Comparison between the proposed method's generated ranking without labeled data and the uncertainty sample strategy with one label provided at each iteration

4.12. Figure 4.12a presents the *ecoli* dataset and the good initial ranking, while Figure 4.12b shows the generated ranking with worse results. Even on the *glass* dataset both curves are close.

4.7.1.2 Ranking Generated Iteratively

In this experiment, the proposed method is used exactly the same way as the baseline strategy, that is, at each iteration one instance is queried and its label is fed to the

algorithm. The model is updated, and another label is queried until the unlabeled set is empty. The objective is to assess the quality of the resultant classifiers when the proposed method is used as a traditional active learning method in which one instance is queried at a time.

Table 4.14: Comparison between the proposed method and the uncertainty sample strategy

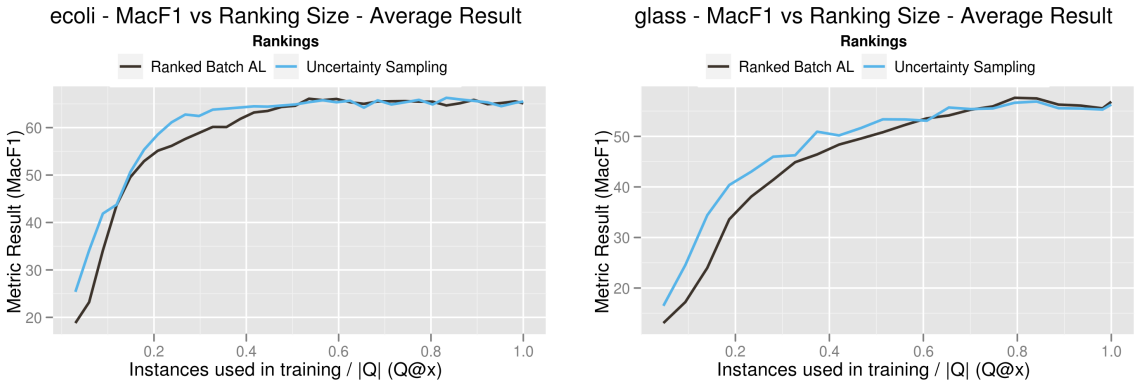
Dataset	**AUC-Acc**		**AUC-Acc@20%**		**AUC-MacF1%**		**AUC-MacF1@20%**	
	Baseline	RBMAL	Baseline	RBMAL	Baseline	RBMAL	Baseline	RBMAL
ecoli	80.78	80.37 ●	70.02	66.74 ▼	60.93	59.46 ▼	42.14	38.07 ▼
glass	57.43	55.91 ▼	43.02	40.13 ●	48.85	46.33 ▼	29.12	21.77 ▼
iris	93.06	92.30 ●	84.25	82.45 ●	92.71	91.69 ●	82.40	79.44 ●
wdbc	94.84	94.77 ●	90.93	90.41 ●	94.39	94.24 ●	90.01	88.96 ●
wine	92.90	90.27 ▼	82.75	65.76 ▼	92.64	89.16 ▼	80.57	57.27 ▼
yeast	49.40	49.16 ●	43.76	42.85 ●	46.21	43.28 ▼	34.88	27.82 ▼
average	78.06	77.13 ●	69.12	64.72 ●	72.62	70.69 ▼	59.85	52.22 ●

▲ 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 4.14 presents the results of the baseline and the proposed method. While feeding the model with labeled data more frequently leads to an improvement in *Accuracy*, the performance of the *MacF1* is, surprisingly, hindered by the model update. This can be seen as a bad value choice for the α parameter, because setting it to consider only uncertainty would lead to the same results as the baseline. This happens because the method was designed to select multiple instances at one, sometimes at the expense of not making the best selection at one given point. One way to avoid this problem is to select the α value according not only to the size of the training set but also to the average speed of labeling, that is, the number of instances labeled by the oracle at each iteration. With this information, it is possible to adjust the α value considering that, in this case, the oracle always labels one instance between iterations.

Figure 4.13 presents the results for the proposed method and the baseline. Note that the curves are more divergent usually when few instances are selected. This happens because the alpha parameter, when the unlabeled set approaches the end, starts to prioritize the uncertainty estimation, that is, the selection strategy tends to be the same of the one used by the baseline.

Additionally, these results can be seen as another indicative of the conclusion



(a) *ecoli* (b) *glass*

Figure 4.13: Comparison between the proposed method's generated ranking and the uncertainty sample strategy

obtained with the factorial design: having a good uncertainty estimator is fundamental to achieve a good ranking because, in some circumstances, it can be the upper bound of the proposed method.

4.7.2 Pool-Based Batch-Mode Sampling Comparison

In this section, the presented method is compared to a Batch-Mode Active Learning algorithm in two scenarios. First, the proposed method is used to generate a full instance ranking while the baseline method is updated with its selected batch at each iteration. Second the proposed method is fed with the same number of instances.

The method used in Sections 4.7.2.1 and 4.7.2.2 is a Batch-Mode Active Learning method based on Support Vector Machines [Brinker, 2003]. 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 one instance receives to be selected for a given batch b is given by Equation 4.12

$$s_i = \lambda \times \text{distance}(i) + (1 - \lambda) \times \text{batchSimilarity}(i, b) \tag{4.12}$$

This method shares some similarities with the proposed method: the λ parameter can, for example, be seen as the α value in this work, although the α parameter assumes pre-determined values and no tuning is needed. The main difference between the two methods is in the use of λ, α and the use of distinct uncertainty estimators. Although this method does not return a ranked batch, it could be adapted to do so.

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

4.7.2.1 Ranking Generated Without Labels

In this experiment the first ranking generated by the proposed method (without any labeled data) is compared to the final ranking generated by the baseline method when labeled data is provided at each iteration.

Table 4.15 presents the comparison results for a batch of size five. Other values for the batch size were also tested and showed similar results, thus, they are not presented.

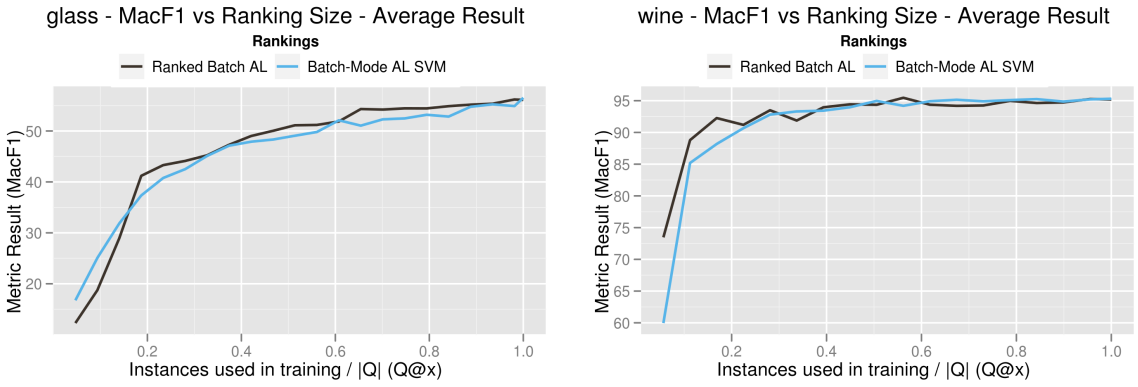
Table 4.15: Comparison between the proposed method, with no labeled data provided, and a batch-mode active learning strategy with the model iteratively updated with labeled instances

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

As can be seen, even with this setup that benefits the baseline, the proposed method is able to achieve results comparable to the baseline method even when no labeled data is fed to the method of this dissertation. In different datasets, there is also an increase of the ranking quality (usually in the *MacF1*). For example, in the *ecoli* dataset there is a 5% increase in the *MacF1* result. This implies that it is possible to avoid a big computational effort maintaining or improving performance.

Figure 4.14 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 the proposed method's ranking is generated with one iteration and given the necessities, it can be a better approach to lose some accuracy in order to



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

Figure 4.14: Comparison between the proposed method's generated ranking and the Batch-Mode Active Learning strategy

avoid iterating at each labeling phase. The second case is the *wine* dataset in which results are significantly better than the baseline, being able to provide a head-start.

This is an important result, because it indicates that the proposed method, with only one iteration, may be able to compete with similar Batch-Mode Active Learning methods.

4.7.2.2 Ranking Generated Iteratively

In this experiment the objective is to compare the proposed method with the Batch-Mode Active Learning baseline in a similar scenario, 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.

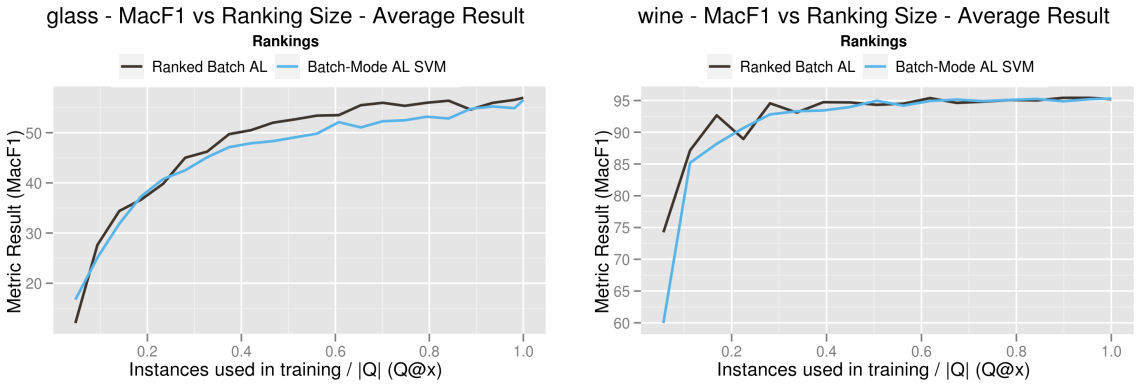
Table 4.16 presents the comparison results for a batch of size five. Other values were also tested and showed similar results, thus, they are not presented.

The results show that when the model is updated, differently from the uncertainty sampling comparison (Table 4.14), there is a quality increase in the selected batch. This result indicates 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, the proposed 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 the imporvement in the average that is close to 8%.

Table 4.16: Comparison between the proposed method and a batch-mode active learning 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*

Figure 4.15: Comparison between the proposed method's generated ranking and the Batch-Mode Active Learning strategy

For comparison purposes, Figure 4.15 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.

4.8 Concluding Remarks

This Chapter addressed the experimental evaluation of the proposed method for solving the Ranked Batch-Mode Active Learning problem. The diversity of domains, number of classes, features and instances enriches the evaluation of the method, making it possible to observe trends and outliers.

The evaluation process used consisted in splitting the datasets in half-and-half multiple times. The first half was used as the unlabeled set and, when necessary, training set, while the other half was used as the test set. Incremental portions of the generated instance ranking were used as training in a given classifier that had its quality assessed in the test split. These multiple evaluations were then used to plot a chart of the classifier quality versus the amount of the ranking used for training. In order to compare rankings the *Area Under the Curve* of this plot is calculated. Since each dataset is split multiple times, it is possible to present confidence intervals and statistically significant results.

The first batch of experiments aimed to answer three questions: How to choose a good Uncertainty Estimator? How to choose a good Similarity Function? and Which one is more important? One method was proposed for choosing the Uncertainty Estimator and one to choose the Similarity Function given the dataset. While the presented results indicate that the proposed method can be effectively used, further investigation, with more datasets, is necessary in order to obtain conclusive results. A $2^k r$ factorial design was used to investigate the impacts of these two components on the final result and one important conclusion is that, usually, most of the result variation is explained by the Uncertainty Estimator. This means that it is usually better to invest in choosing a good estimator in detriment of the Similarity Function.

In Section 3.5 an heuristic for tackling the Cold Start problem was presented. This strategy was compared to starting the ranking with a random instance. Results showed that both strategies lead to similar results, although the proposed heuristic can induce better results of *MacF1*. The proposed strategy can be used when the objective is optimizing the *MacF1*, otherwise the random strategy should be used since it also implies in the reduction of computational costs.

Finally the method was compared to different ways of choosing instances for labeling. In the first scenario the proposed method was compared to the random strategy, that is, the oracle labels instances in a random fashion, and annotated instances doesn't impact future choices. The method proposed in this dissertation was able to achieve gains, specially in *MacF1* for different datasets, even when no labeled data was provided. This means that the proposed method even when generating a full ranking

only with the unlabeled set is able to achieve better quality than the random selection. The second scenario consists in selecting one instance at a time and updating the model accordingly. In this set of experiments the baseline used is a traditional Uncertainty Sample strategy. The proposed method was able to achieve results close to the baseline, but failed in different datasets. Since the Uncertainty Estimators used in both methods is the same, this is possibly a reflex of the way the α parameter is updated. One way to circumvent this problem would be to update α based not only on the size of the training and unlabeled sets but also on the average speed of labeling, that is, the number of instances labeled at each iteration. The last scenario is the typical Batch-Mode Active Learning in which multiple instances are queried and labeled at each iteration. In this experiment the proposed method was compared to a Batch-Mode Active Learning method based on Support Vector Machines. The proposed method was able to achieve better results and this is an indicative that it could be a drop-in replacement for Batch-Mode Active Learning algorithms.

The framework for ranking instances presented in this dissertation was able to provide gains over the random selection even when no labeled data is provided. With the increase of annotated instances the quality of the ranking increases and lead to better classifier models. These experiments show that it is possible to relax the initial assumptions of Batch-Mode Active Learning in a way that the machine suits the man's schedule and not otherwise.

Chapter 5

Conclusion

This dissertation introduced and addressed the Ranked Batch-Mode Active Learning. 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 (in $\mathcal{D}_{\text{ESTIMATED}}$).

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 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 baseline methods.

Concluding this dissertation we present our main contributions and what we plan to study as future work.

5.1 Contributions

The main contributions of this dissertation include:

Ranked Batch-Mode Active Learning description The problem of Ranked Batch-Mode Active Learning was introduced in this dissertation. It aims to make Active Learning algorithms more suitable to use in companies and research groups. In this way, the algorithm is better suited to the user schedule and not otherwise.

Framework for generating ranked queries We presented a framework for solving the Ranked Batch-Mode Active Learning. It generates a query by ranking instances according to its informativeness and dissimilarities with the labeled set. These metrics are obtained through the use of two main components: the uncertainty estimator and a similarity function. The proposed method was evaluated and presented results better than random sampling and comparable to traditional Active Learning methods, in some cases, achieving the same accuracy without any labeled data.

Metrics for evaluating the framework components One metric for evaluating each main component of the framework was presented and evaluated. The experimental results indicate that these metrics can be used to assist in choosing the Uncertainty Estimator and Similarity Function for a given dataset.

Experimental evaluation of the framework components A $2^k r$ factorial design was used in order to measure the impact of each component on the final result. As a general rule the uncertainty estimator is the component that explains most of the result variation.

5.2 Future Work

Other topics that may generate interesting future work include:

Further study metrics for evaluating components Although this dissertation is an indicative that the proposed metrics can be used to assess the component quality, there are datasets in which the general rule may not apply. Further studies with more datasets is needed in order to obtain a conclusion.

Test new uncertainty estimators As discussed in Chapter 2, there is a wide range of sampling strategies that can be a drop-in replacement for the uncertainty estimator, for example, error estimation, variance reduction and density estimation. Since this component is responsible for most of the ranking outcome, we plan to further experiment with different estimators of informativeness.

Further study the impact of the α parameter The α parameter is used to weight the impact of each component on the ranking construction. The α values throughout the ranking should be further studied, including the possibility of updating α according to the oracle speed, that is, the number of analyzed instances at each iteration.

Expand the Ranked query to the Stream scenario In the Stream-based selective sampling (Section 2.3.2) one instance is queried at a time. One way would be to choose one or more instances from the stream in order to create a ranked buffer for labeling.

Bibliography

Naoki Abe and Hiroshi Mamitsuka. Query Learning Strategies Using Boosting and Bagging. In *ICML '98: Proceedings of the Fifteenth International Conference on Machine Learning*, pages 1–9, San Francisco, CA, USA, 1998.

Douglas Aberdeen and Andrew Slater. The learning behind gmail priority inbox. *Production*, pages 3–6, 2010. URL `http://static.googleusercontent.com/external_content/untrusted_dlcp/research.google.com/en/en/pubs/archive/36955.pdf`.

Dana Angluin. Queries and Concept Learning. *Machine Learning*, 2(4):319–342, April 1988.

Dana Angluin. Queries revisited. *Theor. Comput. Sci.*, 313(2):175–194, 2004.

Les Atlas, David Cohn, Richard Ladner, M. A. El-Sharkawi, and R. J. Marks, II. Training connectionist networks with queries and selective sampling. In *Advances in neural information processing systems*, pages 566–573. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1990.

Ricardo Baeza-Yates and Berthier Ribeiro-Neto. *Modern Information Retrieval*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2011.

M. Bayes and M. Price. An essay towards solving a problem in the doctrine of chances. by the late rev. mr. bayes, communicated by mr. price, in a letter to john canton, m. a. and f. r. s. *Philosophical Transactions (1683-1775)*, 1763.

James Bennett and Stan Lanning. The netflix prize. In *KDD Cup and Workshop in conjunction with KDD*, 2007.

Alina Beygelzimer, Daniel Hsu, John Langford, and Tong Zhang. Agnostic active learning without constraints. In *Advances in Neural Information Processing Systems*

23: 24th Annual Conference on Neural Information Processing Systems, pages 199–207, 2010.

Anselm Blumer, A. Ehrenfeucht, David Haussler, and Manfred K. Warmuth. Learnability and the vapnik-chervonenkis dimension. *J. ACM*, 36:929–965, October 1989. ISSN 0004-5411.

Andrew P. Bradley. The use of the area under the roc curve in the evaluation of machine learning algorithms. *Pattern Recognition*, 30(7):1145 – 1159, 1997.

Klaus Brinker. Incorporating diversity in active learning with support vector machines. In *Proceedings of the 20th International Conference on Machine Learning*, pages 59–66. AAAI Press, 2003.

David Cohn, Les Atlas, and Richard Ladner. Improving Generalization with Active Learning. *Mach. Learn.*, 15(2):201–221, May 1994.

David A. Cohn, Zoubin Ghahramani, and Michael I. Jordan. Active learning with statistical models. *J. Artif. Int. Res.*, 4(1):129–145, March 1996.

T. Cover and P. Hart. Nearest neighbor pattern classification. *Information Theory, IEEE Transactions on*, 13(1):21–27, January 1967.

Aron Culotta and Andrew McCallum. Reducing labeling effort for structured prediction tasks. In *Proceedings of the 20th national conference on Artificial intelligence - Volume 2*, AAAI'05, pages 746–751. AAAI Press, 2005.

Ido Dagan and Sean Engelson. Committee-Based Sampling for Training Probabilistic Classifiers. In *Proceedings of the International Conference on Machine Learning*, July 1995.

Sanjoy Dasgupta, Daniel Hsu, and Claire Monteleoni. A general agnostic active learning algorithm. In *International Symposium on Artificial Intelligence and Mathematics*, 2008.

A. Frank and A. Asuncion. UCI machine learning repository, 2010. URL `http://archive.ics.uci.edu/ml`.

Atsushi Fujii, Takenobu Tokunaga, Kentaro Inui, and Hozumi Tanaka. Selective sampling for example-based word sense disambiguation. *Comput. Linguist.*, 24(4):573–597, December 1998.

Stuart Geman, Elie Bienenstock, and René Doursat. Neural networks and the bias/variance dilemma. *Neural Comput.*, 4(1):1–58, January 1992.

Yuhong Guo and Dale Schuurmans. Discriminative batch mode active learning. In *Advances in Neural Information Processing Systems 20, Proceedings of the Twenty-First Annual Conference on Neural Information Processing Systems*, 2008.

Robbie Haertel, Paul Felt, Eric Ringger, and Kevin Seppi. Parallel active learning: eliminating wait time with minimal staleness. In *Proceedings of the NAACL HLT 2010 Workshop on Active Learning for Natural Language Processing*, pages 33–41. Association for Computational Linguistics, 2010.

David Haussler. Learning conjunctive concepts in structural domains. *Mach. Learn.*, 4(1):7–40, 1989.

Carlos A. Heuser, Francisco N. A. Krieser, and Viviane Moreira 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 - Volume 83*, ER '07, pages 71–76, Darlinghurst, Australia, Australia, 2007.

Steven C. H. Hoi, Rong Jin, and Michael R. Lyu. Large-scale text categorization by batch mode active learning. In *Proceedings of the 15th international conference on World Wide Web*, WWW '06, pages 633–642, New York, NY, USA, 2006a. ACM.

Steven C. H. Hoi, Rong Jin, Jianke Zhu, and Michael R. Lyu. Batch mode active learning and its application to medical image classification. In *Proceedings of the 23rd international conference on Machine learning*, ICML '06, pages 417–424. ACM, 2006b.

A. Holub, P. Perona, and M. C. Burl. Entropy-based active learning for object recognition. In *Computer Vision and Pattern Recognition Workshops, 2008. CVPRW '08. IEEE Computer Society Conference on*, pages 1–8, 2008.

David Hull. Improving text retrieval for the routing problem using latent semantic indexing. In *Proceedings of the 17th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '94, pages 282–291, New York, NY, USA, 1994. Springer-Verlag New York, Inc.

Rebecca Hwa. On minimizing training corpus for parser acquisition. In *Proceedings of the Fifth Computational Natural Language Learning Workshop*, pages 84–89, 2001.

Prateek Jain and Ashish Kapoor. Active learning for large multi-class problems. In *2009 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 20-25 June 2009, Miami, Florida, USA*, pages 762–769. IEEE, 2009.

R. K. Jain. *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling*. Wiley, 1 edition, April 1991.

George H. John and Pat Langley. Estimating continuous distributions in bayesian classifiers. In *Proceedings of the Eleventh conference on Uncertainty in artificial intelligence*, UAI'95, pages 338–345, San Francisco, CA, USA, 1995. Morgan Kaufmann Publishers Inc.

Ajay J. Joshi, Fatih Porikli, and Nikolaos Papanikolopoulos. Multi-class active learning for image classification. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 2372–2379. IEEE, 2009.

Ross D. King, Kenneth E. Whelan, Ffion M. Jones, Philip G. K. Reiser, Christopher H. Bryant, Stephen H. Muggleton, Douglas B. Kell, and Stephen G. Oliver. Functional genomic hypothesis generation and experimentation by a robot scientist. *Nature*, 427(6971):247–252, January 2004.

Ross D. King, Jem Rowland, Stephen G. Oliver, Michael Young, Wayne Aubrey, Emma Byrne, Maria Liakata, Magdalena Markham, Pinar Pir, Larisa N. Soldatova, Andrew Sparkes, Kenneth E. Whelan, and Amanda Clare. The Automation of Science. *Science*, 324(5923):85–89, April 2009. ISSN 1095-9203.

Ron Kohavi. A study of Cross-Validation and bootstrap for accuracy estimation and model selection. In *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence*, pages 1137–1145, 1995.

K. J. Lang and E. B. Baum. Query learning can work poorly when a human oracle is used. 1992.

Florian Laws, Christian Scheible, and Hinrich Schütze. Active learning with amazon mechanical turk. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 1546–1556. Association for Computational Linguistics, 2011.

Thomas M. Lehmann, Mark O. Guld, Thomas Deselaers, Daniel Keysers, Henning Schubert, Klaus Spitzer, Hermann Ney, and Berthold B. Wein. Automatic catego-

rization of medical images for content-based retrieval and data mining. *Computerized Medical Imaging and Graphics*, 29(2-3):143–155, 2005.

David D. Lewis and William A. Gale. A sequential algorithm for training text classifiers. In *Proceedings of the 17th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 3–12, New York, NY, USA, 1994. Springer-Verlag New York, Inc.

Jianhua Lin. Divergence measures based on the shannon entropy. *IEEE Transactions on Information theory*, 37:145–151, 1991.

David J. C. MacKay. Information-based objective functions for active data selection. *Neural Comput.*, 4(4):590–604, July 1992.

Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schtze. *Introduction to Information Retrieval*. Cambridge University Press, New York, NY, USA, 2008.

A. McCallum and K. Nigam. Employing EM in pool-based active learning for text classification. In *Proceedings of ICML-98, 15th International Conference on Machine Learning*, pages 350–358, 1998.

Prem Melville and Raymond J. Mooney. Constructing diverse classifier ensembles using artificial training examples. In *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence*, pages 505–510, Acapulco, Mexico, August 2003.

Prem Melville and Raymond J. Mooney. Diverse ensembles for active learning, 2004.

T. M. Mitchell. *Machine learning*. McGraw Hill, New York, 1997.

Tom M. Mitchell. Generalization as search. *Artificial Intelligence*, 18(2):203 – 226, 1982.

Ion Muslea, Steven Minton, and Craig A. Knoblock. Selective sampling with redundant views. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on Innovative Applications of Artificial Intelligence*, pages 621–626, 2000.

Hieu T. Nguyen and Arnold Smeulders. Active learning using pre-clustering. In *Proceedings of the twenty-first international conference on Machine learning*, pages 79–. ACM, 2004.

Stephen Purpura, Claire Cardie, and Jesse Simons. Active learning for e-rulemaking: public comment categorization. In *Proceedings of the 2008 international conference on Digital government research*, dg.o '08, pages 234–243, 2008.

Robert E. Schapire, Yoav Freund, Peter Bartlett, and Wee S. Lee. Boosting the Margin: A New Explanation for the Effectiveness of Voting Methods. *The Annals of Statistics*, 26(5):1651–1686, 1998.

Andrew I. Schein and Lyle H. Ungar. Active learning for logistic regression: an evaluation. *Mach. Learn.*, 68(3):235–265, 2007.

Hinrich Schütze, Emre Velipasaoglu, and Jan O. Pedersen. Performance thresholding in practical text classification. In *Proceedings of the 15th ACM international conference on Information and knowledge management*, CIKM '06, pages 662–671, New York, NY, USA, 2006. ACM.

B. Settles, M. Craven, and L. Friedland. Active learning with real annotation costs. In *Proceedings of the NIPS Workshop on Cost-Sensitive Learning*, pages 1069–1078, 2008.

Burr Settles. Active learning literature survey. Computer Sciences Technical Report 1648, University of Wisconsin–Madison, 2009.

Burr Settles and Mark Craven. An analysis of active learning strategies for sequence labeling tasks. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, EMNLP '08, pages 1070–1079, Stroudsburg, PA, USA, 2008. Association for Computational Linguistics.

H. Sebastian Seung, Manfred Opper, and Haim Sompolinsky. Query by committee. In *Proceedings of the Fifth Annual ACM Conference on Computational Learning Theory*, pages 287–294, 1992.

C. E. Shannon. A mathematical theory of communication. *Bell system technical journal*, 27, 1948.

Xuehua Shen and ChengXiang Zhai. Active feedback in ad hoc information retrieval. In *Proceedings of the 28th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 59–66. ACM, 2005.

Lixin Shi, Yuhang Zhao, and Jie Tang. Batch mode active learning for networked data. *ACM Trans. Intell. Syst. Technol.*, 3(2):33:1–33:25, 2012.

Eduardo D. Sontag. Vc dimension of neural networks. In *Neural Networks and Machine Learning*, pages 69–95. Springer, 1998.

Cynthia A. Thompson, Mary E. Califf, and Raymond J. Mooney. Active learning for natural language parsing and information extraction. In *Proc. 16th International Conf. on Machine Learning*, pages 406–414. Morgan Kaufmann, San Francisco, CA, 1999.

Katrin Tomanek, Florian Laws, Udo Hahn, and Hinrich Schütze. On proper unit selection in active learning: co-selection effects for named entity recognition. In *Proceedings of the NAACL HLT 2009 Workshop on Active Learning for Natural Language Processing*, pages 9–17, Stroudsburg, PA, USA, 2009. Association for Computational Linguistics.

Simon Tong and Edward Chang. Support vector machine active learning for image retrieval. In *Proceedings of the ninth ACM international conference on Multimedia*, MULTIMEDIA '01, pages 107–118, New York, NY, USA, 2001. ACM.

Simon Tong and Daphne Koller. Support vector machine active learning with applications to text classification. *J. Mach. Learn. Res.*, 2:45–66, March 2002.

Vladimir N. Vapnik. *The nature of statistical learning theory*. Springer-Verlag New York, Inc., New York, NY, USA, 1995.

Zhao Xu, Kai Yu, Volker Tresp, Xiaowei Xu, and Jizhi Wang. Representative sampling for text classification using support vector machines. In *Proceedings of the 25th European conference on IR research*, pages 393–407, Berlin, Heidelberg, 2003. Springer-Verlag.

Zuobing Xu, Ram Akella, and Yi Zhang. Incorporating Diversity and Density in Active Learning for Relevance Feedback. In Giambattista Amati, Claudio Carpineto, and Giovanni Romano, editors, *Advances in Information Retrieval*, volume 4425 of *Lecture Notes in Computer Science*, chapter 24, pages 246–257. Springer Berlin / Heidelberg, Berlin, Heidelberg, 2007.

C Yang, S O Prasher, J Landry, H S Ramaswamy, and A Ditommaso. Application of artificial neural networks in image recognition and classification of crop and weeds. *Canadian Agricultural Engineering*, 42(September):147–152, 2000.

Harry Zhang. The Optimality of Naive Bayes. In *FLAIRS Conference*. AAAI Press, 2004.

Tong Zhang and Frank J. Oles. A probability analysis on the value of unlabeled data for classification problems. In *17th International Conference on Machine Learning*, 2000.

Xiaojin Zhu. *Semi-supervised learning with graphs.* PhD thesis, Carnegie Mellon University, Pittsburgh, PA, USA, 2005.