Learning Case Feature Weights from Relevance and Ranking Feedback

Luc Lamontagne[1] **and Alexandre Bergeron-Guyard**[2]

[1] Department of Computer Science and Software Enginering, Université Laval, Québec, Canada
Luc.Lamontagne@ift.ulaval.ca
[2] Defence Research and Development Canada, Valcartier, Canada
Alexandre.BergeronGuyard@drdc-rddc.gc.ca

Abstract

We study in this paper how explicit user feedback can be used by a case-based reasoning system to improve the quality of its retrieval phase. More specifically, we explore how both ranking feedback and relevance feedback can be exploited to modify the weights of case features. We propose some options to cope with each type of feedback. We also evaluate, in an interactive setting, their impact on a travel scenario where some user provides feedback on a series of queries. Our results indicate that the combined weight-learning scheme proposed in this paper succeeds, on average, to assign more weights to the features used to formulate relevance and ranking feedback.

Introduction

To be useful, a case-based reasoning (CBR) system must provide recommendations that meet the information needs of its users. In most CBR systems, the quality of the recommendations strongly depends on how case similarity is evaluated. However, it is difficult to determine at design time a generic similarity configuration that will be satisfying for most situations encountered by the system. Moreover, this configuration might have to be personalized for multiple users having different needs, interests and preferences.

Our objective is to study how CBR systems can improve the quality of their content-based recommendations by learning from their interactions with human users during online retrieval sessions. Given some preliminary retrieval results made by the system, we would like the CBR system to self-adapt its retrieval knowledge to the on-line feedback of a specific user.

In this paper, we address the problem of learning the weights of case features when the system is provided with mixed feedback. More specifically, we propose an optimization scheme that combines techniques to cope with both relevance feedback and ranking feedback.

This approach is useful for CBR applications where case solutions are complex objects. For instance, one of our applications is the support of intelligence operations where products such as likely events and courses of actions are recommended based on the similarity of past operational situations. In this scenario, human feedback is required to assess the retrieval results. As this revision process is usually tedious, we assume that CBR self-adaptation has to be conducted on a limited amount of feedback.

Learning Weights Based on User Feedback

We assume that some user is responsible to train the CBR system according to its own needs and preferences. As we do not assume that the user is familiar with CBR technology, training of the system is solely accomplished by critiquing the recommendations made by the CBR retrieval component. A training episode would proceed as follows (Figure 1):

a) The trainer (i.e. the user) submits a query to the CBR system and gets some recommendations (retrieval results). We assume that case similarity is estimated as a weighted average of the local similarity measures of the case features;
b) The trainer analyzes the recommendations and, if not satisfied, provides some feedback on the validity and/or the relative importance of the cases presented in the retrieval results;
c) A learning component then takes the recommendations of the system and the feedback of the user to determine how the similarity configuration of the system should be modified. A similarity configuration contains all the functions and parameters necessary to estimate the similarity between two cases.

Once the similarity configuration of the system is updated, the user can submit new queries and proceed again with steps a) to c) to further refine the similarity

configuration. Such a process would be repeated until the user is satisfied with the quality of the recommendations provided by the CBR system.

As the user might use have strict criteria and soft preferences for evaluating retrieval results, we want the system to handle two types of explicit feedback: ranking feedback and relevance feedback.

Ranking feedback allow the user to reorder the results as a function of his perceived similarity to the current situation. In our experiments, the user is able to perform re-ranking by using some graphical user interface to move a selected retrieval result up or down. Relevance feedback is done by tagging results as either "*relevant*", "*irrelevant*", or as "*unknown*" (the latter indicating that the user is uncertain about the relevance of a case).

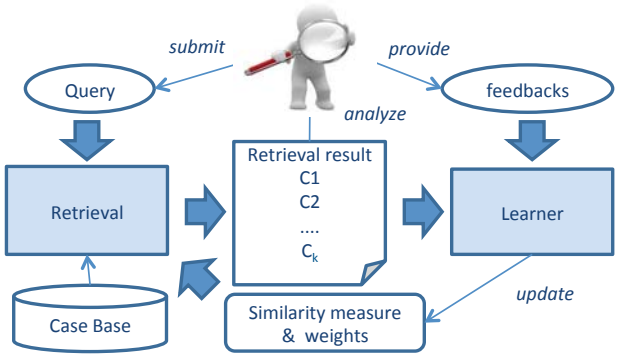

Figure 1: Learning weights from user feedback

The objective of the system is to modify the weights of the similarity measure to obtain a new ordering of cases that reflects the specified feedback. This is accomplished by using the techniques discussed in the next sections.

Learning from Ranking Feedback

Given a list of cases retrieved by the CBR system that are presented in decreasing value of similarity, a user can provide a ranking feedback describing the ordering of cases he would have liked to get from the system. An example of ranking feedback is presented in Figure **2**.

The new ordering of cases provided by the user indicates that, for any two consecutive cases, the first should either have a higher rank or be ranked equal to the other. The latter occurs either when two cases are considered equivalent (e.g. having similar feature values) or when the user has no specific preference over two cases. Hence a ranking feedback does not necessarily require a strict total order of the cases. It is also important to mention that the feedback of the user only applies to the cases returned as retrieval results by the CBR system.

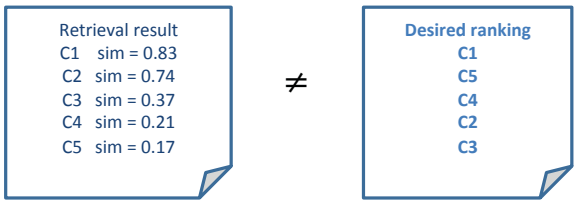

Figure 2: Example of Ranking Feedback

To modify the ranking of some cases, we need to modify the global similarity $sim(q,c)$ of each case c with the query q such that the ranking of the system becomes identical to the ranking of the user. In our work, this similarity is estimated as the weighted sum of the local similarities sim_f, i.e.

$$sim(q,c) = \sum_{f=1}^{n} w_f \times sim_f(q,c) \ .$$

Given a ranking feedback from the user, we want to determine a new set of weights w'_f such that if $rank(c_1) < rank(c_2)$ then $sim'(q,c_1) \geq sim'(q,c_2)$ where

$$sim'(q,c) = \sum_{f=1}^{n} w'_f \times sim_f(q,c) \ .$$

The general algorithm to update the weights is described in Figure **3**. The LEARN-RANKING-WEIGHTS algorithm is a heuristic search that tries to adjust feature weights so that the cases returned by a CBR component have the same order as those in the ranking feedback. This algorithm relies on three functions:

- UPDATE-WEIGHTS modifies the weights according to the differences between the retrieval results and the ranking feedback provided by the user.
- RERANK-CASES evaluates, using a set of weights, the similarity of cases with the query problem. In practice, this function simply invokes the retrieval phase of the CBR component with the cases that were contained in the initial retrieval results.
- ESTIMATE-ERROR estimates to what extent the ranking in the retrieval results diverges from the ranking feedback.

```
function LEARN- RANKING-WEIGHTS(query,
        retrievalResults, weights, feedback) returns weights
    cases ← the k nearest cases contained in retrievalResults.
    repeat until some stopping criterion is satisfied
        weights ← UPDATE-WEIGHTS(weights, retrievalResults,
                                 feedback);
        results ← RERANK-CASES(query, cases, weights);
        error ← ESTIMATE-ERROR(results, feedback);
    return weights
```

Figure 3: General algorithm for learning weights from ranking feedback

LEARN-RANKING-WEIGHTS iteratively tries to find a promising set of weights, determine the new case ranking obtained with these weights and determine if this new ranking still contains errors. These were repeated steps

until a combination of the following stopping criteria is satisfied:

- There is no more ranking error (as estimated by the ESTIMATE-ERROR function);
- The algorithm has completed *N* iterations (*N* being a threshold value defined by the system designer);
- The ranking error is small and does not get further reduced over multiple iterations.

Estimation of the Learning Error

A ranking error occurs when two cases are not respectively ranked as recommended by the ranking feedback. For instance, as illustrated in Figure **4**, the user would have liked that the ranking of cases c_2 and c_4 to be inversed in the retrieval result.

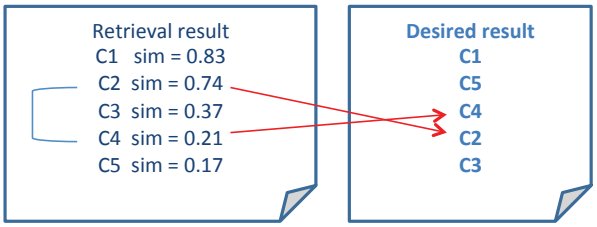

Figure 4: Ranking error between cases c_2 and c_4

The estimation of the ranking error for two wrongly ranked cases c_i and c_j could be estimated using the following measures:

- The difference of similarity between the two cases. In our example, the similarity difference is 0.53.
- The number of ranks separating the pair of cases. In our example, c_2 and c_4 are separated by 2 ranks.
- A combination of both errors. For instance:

$$error = \left(sim(q, c_i) - sim(q, c_j)\right) \times \left(rank(c_j) - rank(c_i)\right)$$

```
function ESTIMATE-ERROR(results, feedback) returns weights
    cases ← the k nearest cases contained in the results.
    totalError ← 0.0;
    for each case1 in cases
        for each case2 following case1 in cases
            if (WRONG-RANKING (case1, case2, results, feedback))
                simError = sim(case1) – sim(case2)
                rankError = rank(case2) – rank(case1)
                totalError ← totalError + (simError * rankError)
    return totalError
```

Figure 5: Error estimation function

In our work, we use the latter error definition and the total error over all the wrong rankings in the retrieval results of the CBR system as estimated by the ESTIMATE-ERROR function described in Figure **5**.

Learning to Rerank

To learn new weights from ranking feedback, we adopted a gradient descent approach [Stahl 2001]. Gradient descent is an optimization method that search for a local optimum by modifying a solution step by step. The modifications made at each step are proportional to some gradient values, which is a quantity estimating to what extent a feature contributes to the total ranking error.

```
function LEARN-WEIGHTS-GRADIENT(query, retrievalResults,
weights, feedback) returns weights
  bestWeights ← 0
  lowestError ← ESTIMATE-ERROR(retrievalResults, feedback);
  repeat until (error → 0) or (nbIterations > MAX_STEPS)
    newWeights ← UPDATE-WEIGHTS -GRADIENT(bestWeights,
                    retrievalResults, feedback, learningRate);
    results ← RERANK-CASES(query, cases, newWeights);
    newError ← ESTIMATE-ERROR(results, feedback);
    if (newError < lowestError)
        bestWeights ← newWeights
        lowestError ← newError
    else
        learningRate ← learningRate * scalingFactor
    increment nbIterations
  return bestWeights
```

Figure 6: Learning weights from ranking feedback using gradient descent

```
function UPDATE-WEIGHTS -GRADIENT(weights,
      retrievalResults, feedback, learningRate) returns weights
    for each weight in weights
        gradient ← COMPUTE-GRADIENT(retrievalResults,
                                    feedback, feature);
        newWeight ← weight + (learningRate × gradient)
        add newWeight to newWeights
    return NORMALIZE(newWeights)

function COMPUTE-GRADIENT(retrievalResults, feedback,
                                    feature) returns double
    cases ← the list of cases from retrievalResults
    gradient ← 0.0;
    for each case1 in cases
        for each case2 following case1 in cases
            if (WRONG-RANKING (case1, case2, results, feedback))
                simDiff = sim_feature(case1) – sim_feature(case2)
                rankDiff = rank(case1) – rank(case2)
                gradient ← gradient + (simDiff * rankDiff)
    return gradient
```

Figure 7: Updating weights using gradient descent

More formally, the gradient is obtained from the first derivative of the total ranking error function with respect to each weight variable. To minimize the ranking error, we update each feature weight w_f by making a step corresponding to the inverse of the gradient (i.e. we follow the opposite direction of the slope of the error function to reach some minimum point). The resulting update function corresponds to the sum of the local errors for each inverted pair of cases, i.e.

$$w_f' = w_f - \alpha \sum_{c_i, c_j \in WR} simError_f(q, c_i, c_j) \times rankError(c_i, c_j)$$

where WR is the set of wrongly ranked pairs of cases and

$$simError_f(q, c_i, c_j) = sim_f(q, c_i) - sim_f(q, c_j)$$
$$rankError(c_i, c_j) = rank(c_i) - rank(c_j).$$

A pseudo-code description of the algorithm is provided in Figure **6** and Figure **7**. The learning rate α is a small positive value that regulates the changes made to the weights. Usually the learning rate is adjusted empirically by conducting trials on a representative domain case base. We found in our experiments that a small value should be selected to prevent the algorithm from bouncing back and forth over a local optimal set of weights. A scaling factor, between 0 and 1, is used to decrease the learning rate when weight updates fails to reduce the error rate.

Learning from Relevance Feedback

We describe in this section how to adjust the weights of the CBR system when a user provides some feedback on the relevance of some cases.

Relevance Feedback

The idea of relevance feedback is to get the user to indicate that some cases should either be present or not present in the retrieval results. In general, relevance should indicate if a case meets some user's information needs. However, in practice, a case might be judged irrelevant for one of the following reasons:

a) The value of one important feature is not acceptable.
b) Some combinations of feature values do not go well together.
c) The solution of a case is not useful.

With respect to weight optimization, the reasons a) and b) indicate that more weight should be assigned to the features used to determine the relevance of a case. However reason c) is more complex as it relates to the utility of a case. While we expect case utility to be proportional to case similarity, it cannot be fully guaranteed in practice. Hence, relevance feedback should work well if the feedback given by user is solely based on the features of the case problems.

Rocchio Update

The field of Information Retrieval (IR) [Manning et al. 2009] has studied for many years how to cope with relevance in retrieval systems. Their main intuition was that the keywords of a query should be assigned a higher weight if they appear in relevant documents. Conversely, they should be penalized if they appear frequently in non-relevant documents.

We adapted this intuition to case-based reasoning as follows: features assigning higher similarity values to relevant cases should see their weight increased while weights of other features should be decreased. Extending the Rocchio equation proposed in the IR literature, we propose the following update function to adjust weights based on relevance feedback:

$$w_f' = w_f + \frac{\beta}{|cases_{REL}|} \sum_{c_i \in cases_{REL}} sim_f(q, c_i) - \frac{\gamma}{|cases_{NR}|} \sum_{c_j \in cases_{NR}} sim_f(q, c_j)$$

where $cases_{REL}$ is the group of relevant cases and $cases_{NR}$ is the group of non-relevant cases. This relevance feedback update function has the effect of modifying the feature weights so that the query problem q is moved closer to the centroid of the relevant cases and moved away from the centroid of the non-relevant cases.

The parameters β and γ determine the relative important of relevant cases to irrelevant ones. In practice, values such as $\beta = 0.75$ and $\gamma = 0.25$ are frequently used. But we recommend determining them empirically on a representative data set.

It is important to mention that this approach does not require an iterative algorithm as the learning algorithm presented for coping with ranking feedback. Given some retrieval results and a relevance feedback, the update function is applied only once to each feature weight.

Pseudo-ranking Approach

A relevance feedback suggests some observations on the desired ranking of the cases. First, we would expect the group of relevant cases to be returned at the top of the case ranking. For instance, in our example presented in Figure **8**, relevant cases c_1 and c_5 should be the most highly ranked cases. Even if ranked first, it is impossible to determine a strict total ordering among the relevant cases solely based on their relevance value. Hence we considered that they should all have the same rank (rank 1).

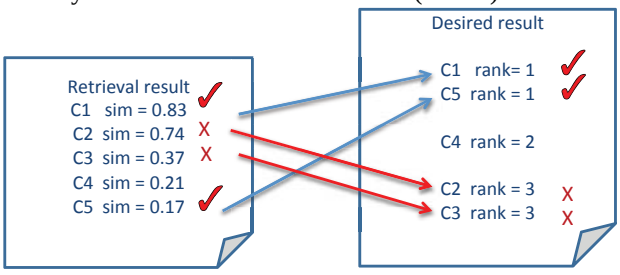

Figure 8: Relevance feedback as a pseudo-ranking

In contrast, non-relevant cases should have lower similarities than those of the relevant group and be located at the bottom of the case ranking. For instance, c_2 and c_3 should not precede c_1 and c_5.

This leaves the remaining group of cases for which we do not have a relevance feedback (c_4 in our example). One option would be to ignore these "*unknown*" cases during the learning process. However we believe that some

valuable information resides in the fact that these cases are neither relevant nor non-relevant. We assume that the similarity of the "*unknown*" cases should be lower than those of the relevant group and higher than those of the non-relevant. This implies that they should be ranked as an intermediate group in-between relevant and irrelevant case.

Given that a relevance feedback can be interpreted as a pseudo-ranking of three groups of cases (*relevant*, *unknown* and *irrelevant*), it becomes possible to apply a gradient descent algorithm to learn weights from relevance feedback. However, to do so, we need to refine the notion of error. A *pseudo-ranking error* occurs if:

- Two cases do not have the same relevance feedback;
- And the case with lower relevance has a higher similarity value than the other.

Learning with Combined Feedback

To learn weights when both types of feedback are provided by the user, we simply adopt a cascade of learners where:

- Cases are ranked first based on the ranking feedback. Gradient descent is applied to establish this ranking.
- The resulting weights are then updated using the modified Rocchio update function to take into account the relevance of the cases.

This first step of this scheme ensures that a suitable set of weights is selected to respect the relative ranking desired by the user. This approach also offers the advantage that a pseudo-ranking optimization can be conducted if the user only provides some relevance feedback.

The second step, where a relevance feedback update is applied to the ranking weights, aims to reduce further the global similarity of the non-relevant cases so that it would leave some opportunity to insert new cases in the retrieval results during successive search over the entire case base.

Empirical Study

To study the behavior of the proposed weight-learning scheme, we conducted experiments in an interactive setting to evaluate how user feedback were translated into feature weights. The training sessions made use of the travel case base[1], a data set containing an interesting variety of feature types (numerical, categorical, ontological…). The training scheme and the learning algorithms described in the paper were implemented in Java. We also made use of jCOLIBRI [Recio-Garcia et al. 2008] as a CBR framework to perform case retrieval and to obtain the retrieval results needed by the weight learning algorithms.

[1] The travel case base contains 1024 cases defined with 7 problem features (holiday type, number of persons, region, transportation mean, duration, season, accommodation) and 2 solution features (hotel & price). This case base is available at http://www.cs.auckland.ac.nz/~ian/CBR/.

Each of our training sessions followed this procedure:

1. The human trainer selects some features to determine the relevance and ranking of the cases.
2. The trainer submits a query to the CBR system and analyzes the retrieval results. In our experiments, the 5 top-ranked cases were presented.
3. The trainer formulates relevance and ranking feedback based on the features selected in step 1.
4. The learning scheme updates the weights and stored them in the similarity configuration to be reused for other training episodes. In our experiments, we used a learning rate of 0.1, a scaling factor of 0.5, $\beta = 0.8$ and $\gamma = 0.1$.
5. Steps 2-4 are repeated with new queries until the ranking and relevance of the results are deemed satisfactory by the trainer.

The results presented in this section are the average values obtained over 5 different training sessions.

In our first experiment, we use a single feature (*holiday type*) to determine the relevance of a case. To be relevant, a case must have the same feature value as the query. We also use another feature (*season*) to establish the relative ranking of the cases. A ranking feedback corresponds to the cases ranked in decreasing order of similarity for this feature. The training results, presented in Figure **9**, clearly indicates that the learning scheme recognizes, within 2 or 3 user queries, the two features used to formulate the feedback. It is interesting to observe that most of the weight is assigned to the feature associated to relevance (*holiday type*). We also notice that the feature assigned to ranking feedback (*season*), is assigned sufficient weight by the learning scheme to act as a soft constraint.

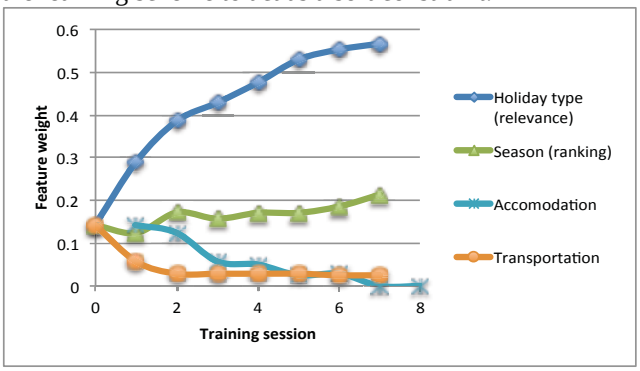

Figure 9: Weight learning results with relevance and ranking feedback determined using one feature each.

We repeated the same experiment using two features to determine the relevance of a case (*holiday type* and *season*) and one feature to establish case ranking (*accommodation*). Again the features used to establish case relevance were quickly recognized through learning and are assigned most of the global weight (see Figure **10**). In practice, most of the recommendations become relevant after a few training

episodes. We also observed that the learning scheme assigns sufficient weight to the ranking feature (*accommodation*) to get a consistent case ordering in most situations. However we noticed during our training sessions that the gradient learner sometimes had difficulty to perform its optimization satisfactorily.

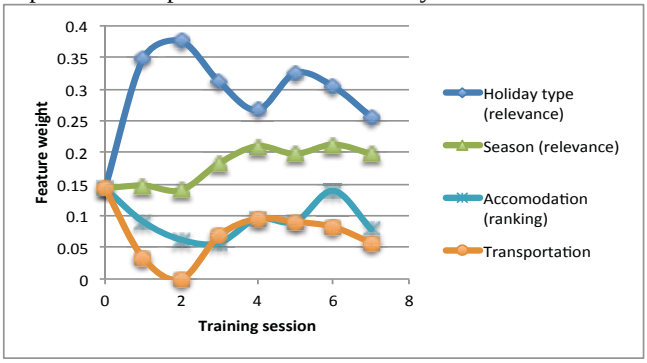

Figure 10: Weight learning results for relevance and ranking feedback established from 3 features.

Part of this problem is due to the error estimation function. If two cases have inverted ranks but also have the same global similarity score, the training scheme estimates that there is no error as the difference of similarity is null. In these conditions, weight learning does not get applied. This relates to the fact that ranking feedback corresponds to non-strict partial rankings. Other criteria to force optimization in these situations should be studied.

Another problem occurs when some features get assigned a weight of 0. These features have no more influence on the ranking of the cases. It then becomes difficult in the following training episodes to reassign them some weight. As fewer features contribute to similarity estimation, the likelihood of having multiple cases with the same similarity score is increased - which brings us back to the problem with the error estimation function.

Related Work

Early work in this research direction was dedicated to the modification of feature weights when CBR is applied to classification tasks. [Wettschereck & Aha 1997] compared various learning algorithms based on a leave-one out evaluation of the cases to estimate the classification accuracy of a CBR retrieval component. In [Bonzano et al. 1997], learning is used to update feature weights in order to optimize the problem-solving performance of a CBR system. [Branting 2003] proposed an approach to determine feature weights by looking at selections from a set of items. Feature weights are modified by adding/subtracting fixed values so that selected items are moved higher in the recommendation list. Our presentation of the gradient descent approach to learn weights from ranking feedback is inspired from the work of Armin Stahl [Stahl 2001]. The topic of user preferences has also been studied for recommender systems [Bridge et al. 2005] and conversational CBR [Aha et al., 2005]. Our research effort differs from previous work as it incorporates both relevance and ranking feedback to learn CBR feature weights. We also make use a modified formulation of Rocchio relevance feedback to update CBR feature weights.

Conclusion

In this paper, we explored how CBR weight learning can be conducted for both ranking and relevance feedback. We proposed to combine techniques to consider both types of feedback simultaneously. Experiments conducted to evaluate the algorithms clearly indicate that the learning scheme can recognize within a few sessions the features used to formulate both types of feedback.

As future work, we recommend to experiment with more complex learning schemes to estimate their benefits. For instance, stochastic gradient descent seems to be an interesting candidate. Other error estimation functions should be studied to force weight optimization when multiple cases have the same global similarity. Finally, to improve the robustness of the solutions, we would like to integrate past feedback into the learning scheme using some decay functions for preferences change over time.

References

Aha, D.W. ; McSherry, D. ; & Yang, Q. 2005. Advances in conversational case-based reasoning. *Knowledge Engineering Review*, 20(3), pp. 247-254.

Bonzano, A.; Cunningham, P.; and Smyth, B. 1997. Using Introspective Learning to Improve Retrieval in CBR: A Case Study in Air Traffic Control, *In Proceedings of the Second International Conference on Case-Based Reasoning Research and Development*, Springer, pp. 291-302.

Branting, K. 2004. Learning Feature Weights from Customer Return-Set Selections, *Knowledge and Information Systems*, vol. 6, issue 2, Springer, pp. 188-202.

Bridge, D.G. ; Göker, M.H. ; McGinty, L. ; Smyth, B. 2005. Case-based recommender systems. *Knowledge Engineering Review*, 20(3), pp. 315-320.

Manning, C.; Raghavan, P.; and Shutze, H. 2009. *Introduction to Information Retrieval*, Cambridge University Press.

Recio-Garcia, J. A.; Diaz-Agudo, B.; and Gonzalez-Calero, P. 2008. *jCOLIBRI: Tutorial*, Technical report, Facultad de Informatica, Universidad Complutense de Madrid, Spain.

Stahl, A. 2001. Learning Feature Weights from Case Order Feedback, *Case-Based Reasoning Research and Development*, Springer. pp. 502-516.

Wettschereck, D.; Aha, D.; and Mohri, T. 1997. A Review and Empirical Evaluation of Feature Weighting Methods for a Class of Lazy Learning Algorithms. *Artificial Intelligence Review*, 11 (1-5), pp. 273-314.