April 2018

Feature-Tree Labeling for Case Base Maintenance

Nariman NAKHJIRI [a,1], Maria SALAMÓ [a] and Miquel SÀNCHEZ-MARRÈ [b]
[a] *Facultat de Matemàtiques i Informàtica,*
Universitat de Barcelona Institute of Complex Systems (UBICS),
Universitat de Barcelona (UB)
[b] *Knowledge Engineering and Machine Learning Group (KEMLG-UPC)*
Intelligent Data Science and Artificial Intelligence Research Centre (IDEAI-UPC)
Dept. of Computer Science
Universitat Politècnica de Catalunya (UPC)

Abstract. Case Base Maintenance (CBM) algorithms update the content of the case base with the aim of improving the case-based reasoner performance. In this paper, we introduce a novel CBM method called Feature-Tree Labeling (FTL) with the focus on increasing the general accuracy of a Case-Based Reasoning (CBR) system. The proposed FTL algorithm is designed to detect and remove noisy cases from the case base, based on value distribution of individual features in the available data. The competence of the FTL method has been compared with well-known state-of-the-art CBM algorithms. The tests have been done on 25 datasets selected from the UCI repository. The results show that FTL obtains higher accuracy than some of the state-of-the-art methods and CBR, with a statistically significant degree.

Keywords. Case-Based Reasoning, Case Base Maintenance, Decision Tree,

1. Introduction

The competence of a Case-Based Reasoning (CBR) [12] system can be measured according to two aspects. First the general accuracy and second the size of the case base. In the field of Case Base Maintenance (CBM) [9], methods that aimed to increase the accuracy of prediction are referred to *Competence Enhancement* [3] algorithms, and methods that are focused on reducing the size of the case base while keeping the accuracy at the same level are called *Competence Preservation* [15] algorithms.

As the classification of new instances in a CBR system heavily depends on individual instances in the case base, in order to increase the average classification accuracy of data, it is important to define a model to distinguish and remove noisy cases from the case base. The model can be built based on inter-relations among instances in the case base. In this paper, we propose the Feature-Tree Labeling (FTL) algorithm. It is a Competence Enhancement Case Base Maintenance algorithm that looks into the value distribution of each feature in the training data. The FTL competence model contains a decision tree

[1] Corresponding Author: University of Barcelona, Spain; E-mail: nnakhjna21@alumnes.ub.edu

classifier for each feature, which has been built with the CART [2] technique. The evaluation of FTL algorithm demonstrates that it is able to achieve higher average accuracy in comparison to well-known state-of-the-art methods. The basic model of CBR along with four state-of-the-art CBM methods have been tested to put the FTL performance into perspective. To analyze its performance, we have chosen 25 datasets from the UCI [7] repository. The results show a statistical significant improvement in accuracy made by FTL compared to the baseline CBR and some of the state-of-the-art methods.

The structure for the rest of this paper is as follows. First, Section 2 presents the related work in Case Base Maintenance and use of decision trees with a CBR system. Section 3 details the proposed method and Section 4 discusses its performance. Finally, Section 5 is devoted to the conclusions and future work.

2. Related work

In the literature, Case Base Maintenance refers to algorithms applied on the case base of a Case-Based Reasoning system, to remove harmful content and improve its performance. One of the early attempts for CBM in Competence Enhancement category was *Wilson*'s Edited Nearest Neighbors (ENN) [14]. ENN is a decremental algorithm that considers cases from case base which are misclassified by their k-nearest neighbors as noise. Thus it removes them. ENN was an inspiration for many later research. *Tomek*'s method of Repeated Edited Nearest Neighbors (RENN) [13], takes ENN strategy and applies it multiple times on the case base until no further noises could be detected. A branch of CBM algorithms build their competence models on property sets of cases in case base. Blame-Based Noise Reduction (BBNR) [6] technique introduced by *Delany*, uses a *Liability* set to determine the competence of instances. *Delany* later expands her work with four property sets of Reachability, Dissimilarity, Coverage, and Liability with RDCL [5] method. RDCL categorizes cases in the case base into eight profiles based on their performance in a leave-one-out test of the case base.

Tree classifiers are one of the common tools for classification. Although they tend to not have the highest accuracy among other methods, but there are several reasons for their popularity such as an easy interpretation of structure, low time consumption for train/test and ability to handle both numeric and nominal attributes. Numerous studies have tried to synthesize decision trees (DT) with CBR from different aspects, but none of them had been done on CBM of the system. We can mention *Richardson and Warren*'s work [1] on using decision trees to add weights to the features and *Chang and Fan*'s implementation [4] of a hybrid method of CBR, decision trees and genetic algorithms on a medical domain, as some of the approaches with successful results. Several experiments have also done with combination of DT and CBR with a Data Mining (DM) structure. With increased interest in DM, many researchers turned their focus to embed those techniques in a CBR system in order to retrieve better information and improve performance of a case based reasoner. Among these studies, experimenting DTs with a CBR system, it is interesting, the work of *Huang, Chen and Lee* [8] on a CBR data-mining system for disease diagnosis. And finally, we can mention *Perner*'s work [11], that has focused on data-mining for CBR on sparse data with a decision tree. As mentioned, the area of using decision trees for maintenance of a CBR system has not been fully covered and the proposed FTL method can opens a new path for Case Base Maintenance.

3. Method

The proposed *Feature-Tree Labeling* (*FTL*) is a CBM algorithm that uses decision tree classifiers to create a reformed version of the case base, which will be used in detection of noisy cases.

3.1. Introduction to Feature-Tree Labeling

Noisy and harmful cases detection requires a competence model. On general data without expert advice or specific rules to point them out, our only source is the data itself along its distribution and interrelations. The goal of FTL is to discover additional information from the distribution of each feature. This extra knowledge helps the proposed method to distinguish between regular and irregular cases in case base. The *FTL* labels features of cases in case base with the common class of their locality. Labeled representation of data help us to detect suspicious cases and to decide which ones should be removed.

In a dataset with n attributes, the process of *FTL* can be summarized into four steps:

1. *Constructor*: Train a decision tree on every feature with a total of n trees. The values of the trees come from the cases in the case base.
2. *Labeler*: Build a reformed version of the case base with the n trained decision trees.
3. *Hamming Nearest Neighbors*: Run a modified version of the Nearest Neighbor classifier on the reformed case base.
4. *Removal*: Remove those cases that do not have the same class as their neighborhood's majority.

These four steps of the FTL algorithm can also be categorized into two phases of the procedure. The first two steps belong to the *data preparation* phase, where we create a reformed case base, and two later steps define the *maintenance cycle*, in which FTL detects and removes noisy cases. Algorithm and functions presented in this paper use CB as the notation for the case base with n attributes, which consists of cases like $C \in CB$: $C = \{c_1, c_2, ..., c_n, L_c\}$, where c_i is the value of ith attribute and L_c is the class label in the case C.

Algorithm 1 FTL

Input: The case base CB with n attributes.
Output: The maintained version of the case base, CB_{edited}.

```
// Phase 1: Data preparation
T = Constructor(CB)                                          ▷ Step 1.
CB_lbd = Labeler(CB, T)                                      ▷ Step 2.
// Phase 2: Maintenance cycle
CB_edited = ∅
for C_lbd ∈ CB_lbd do
    CB_test = CB_lbd − {C_lbd}
    neighbors = Hamming_Nearest_Neighbours(CB_test, C_lbd)   ▷ Step 3.
    predicted_class = Majority_Vote(neighbors)
    if predicted_class == actual class of C_lbd then
        CB_edited.append(C)                                  ▷ Step 4.
return CB_edited
```

The Algorithm 1 is the pseudo-code of the FTL method. The first phase of the algorithm consists of the *Constructor* and the *Labeler* functions. These functions are detailed in Section 3.2. The second phase of the FTL which includes step 3 and step 4, takes place in a loop for leave-one-out test of the cases in the reformed case base (CB_{lbd}). A case C in step 4, which is added to the CB_{edited}, is the corresponding case in raw data to C_{lbd} from CB_{lbd}. Further explanations of this second phase of FTL are described in Section 3.3.

3.2. Data preparation

The first phase of the FTL method is *data preparation*. In this phase a reformed version of the case base will be produced (CB_{lbd}). This phase consists of step 1 the *Constructor*, explained in Section 3.2.1, and step 2 the *Labeler*, detailed in Section 3.2.2.

Data preparation in FTL uses decision tree classifiers built on *Breiman*'s Classification and Regression Tree (CART) [2] technique. CART allows us to create a tree classifier on every feature of data independently of being a numeric or a nominal attribute. *Gini impurity* is used as the measure to split the nodes of the trees. It is worth mentioning that, as tree classifiers in FTL are built on one feature of the data, there is no need for the dependent attribute selection for the splits. This improves the construction time of each tree in the algorithm compared to more complex trees built on several features.

3.2.1. Constructor

The *Constructor* function presented in Algorithm 2 is devoted to train n tree classifiers with feature values of the cases in case base. The output of Constructor function is the set $T = \{t_1, t_2, t_3, ..., t_n\}$, where t_i is a decision tree classifier trained on values of ith attribute from the cases in the case base. According to Algorithm 2, the training set for the decision tree classifier of t_i contains partial cases of instances in the case base. Each partial case of an instance C, consists of a single feature value of c_i and the class label of C.

Algorithm 2 CONSTRUCTOR

Input: CB: The case base with n attributes.
Output: Set $T = \{t_1, t_2, t_3, ..., t_n\}$.

```
T = ∅
for i ∈ {1,2,...,n} do
    train_set = ∅
    for C ∈ CB do
        partial_case = {c_i, L_c}
        train_set.append(partial_case)
    t_i = DecisionTree.train(train_set)
    T.append(t_i)
return T
```

The *DecisionTree.train* used in Algorithm 2, calls the training routine of the decision tree in CART with a *train_set* data and returns a decision tree classifier.

3.2.2. Labeler

The next step after training the decision trees in *Constructor*, is to use these classifiers to build a reformed case base. *Labeler* function is designed to replace the feature values of the cases in the case base with their classification results produced by their attribute tree. According to *Algorithm 3*, the feature value of c_i for every case in the case base, calls the prediction routine of the tree classifiers ($t_i.predict$). The result would be a *label* in the domain of data class labels, *L*. This *label* value would replace c_i in the labeled case base (CB_{lbd}).

Algorithm 3 LABELER

Input: *CB*: The case base with *n* attributes, *T*: the set *T* produced by *Constructor* function. *A* is the set of attributes in CB.
Output: CB_{lbd}: The labeled version of the case base.

```
CB_lbd = ∅                                   ▷ labeled case base
for C ∈ CB do
    C_lbd = ∅                                ▷ labeled case
    for i ∈ A do
        label = t_i.predict(c_i)
        C_lbd.append(label)
    CB_lbd.append(C_lbd)
return CB_lbd
```

To give a better understanding we show an example of FTL data transformation in *iris* dataset. Instances in this small dataset are the variations of iris flower. In *iris* dataset, $A = \{sepal_{length}, sepal_{width}, petal_{length}, petal_{width}\}$ and $L =$ {Setosa ,Versicolour, Virginica}. The case $C = \{4.7, 3.2, 1.3, 0.2\}$ with class label of $L_C = \{Setosa\}$, under FTL process will be transformed into $C_{lbd} = \{Versicolour, Setosa, Virginica, Setosa\}$.

3.3. Maintenance cycle

The second phase of the FTL method uses a leave-one-out test on the labeled case base produced in *data preparation*. Leave-one-out test allows us to retrieve neighbors of every case (*step 3*) and test for their competence (*step 4*). See Section 3.3.1 and Section 3.3.2, respectively.

3.3.1. Hamming Nearest Neighbors

Algorithm 4 implements a modified version of the k-NN classifier, suitable for CB_{lbd}, to retrieve the nearest neighbors of a given case. There are two differences between the k-NN classifier and the Hamming Nearest Neighbors (HNN) used in the FTL. The first difference lies in the metrics that HNN uses. As the name suggests *Hamming distance* is the chosen metric for the FTL.Duo to the fact that CB_{lbd} contains only nominal features, the Hamming distance is a reasonable option for the distance comparison.

The second difference between HNN and k-NN is between the number of neighbors that each method retrieve. While k-NN classifier returns the constant number of cases (k) on each cycle, HNN retrieves all the cases in the CB_{lbd} with the minimum distance to the target instance. With only nominal features in CB_{lbd} and applying the Hamming metric,

possible outcomes for distances between cases can range only between $\{1,2,3,...,n\}$, where n represents the number of features in data. The limited outcome for the distances situates many cases in the same position in a nearest neighbors classification procedure. Also some cases that are different in their original feature values, may have similar representation in the CB_{lbd}. With having many cases with a same distance to another choosing a constant number of 'k' from them will be undependable as we have to choose randomly if cases with similar distance exceed the number of cases we want to retrieve. Algorithm 4 details the procedure of *Hamming Nearest Neighbors*.

Algorithm 4 HAMMING NEAREST NEIGHBORS

Input: CB_{lbd}: The reformed case base, C_{test}: the labeled case under the test, $C_{test} = \{c'_1, c'_2, c'_3, ..., c'_n, L_{c'}\}$
Output: *Neighbors*: The set of closest instances in CB_{lbd} to C_{test}

```
Distances = ∅
 for C ∈ CB do
    distance_C = 0
     for i ∈ A do
        if c'_i == c_i then
            distance_C += 0
        else
            distance_C += 1
    Distances.append(distance_C)
distance_min = minimum(Distances)
 neighbors = ∅
 for C ∈ CB do
    if distance_C == distance_min then
        neighbors.append(C)
return neighbors
```

3.3.2. Removal

The rationale behind the removal policy of FTL is that irregular cases that do not have common values in their features according to their class membership can not represent their class well in future classifications. The final step of the FTL method as described in Algorithm 1, computes the majority class of the retrieved neighbors from HNN for each case. If the predicted class for the labeled version of a case C is different from its actual class, then the case C had some irregular feature values and would be categorized as noise. Cases with the same predicted and actual class will find their way to the maintained case base and will be returned as the final output of the FTL algorithm.

4. Evaluation

For the evaluation of FTL method, 25 datasets from the UCI [7] repository have been selected for testing. For the purpose of comparison, four well-known state of the art algorithms are also implemented in tests. These algorithms are, ENN [14], RENN [13], BBNR [6], and RDCL [5]. The RDCL method used in the evaluation is set to remove

cases from case base with 'DL' and 'DCL' profiles. All the methods in the evaluation use a 10-fold cross validation, a 1NN classifier and a heterogeneous metric (Euclidean for numerical and Hamming for nominal values) in the CBR system.

4.1. Datasets

Table 1 shows the details of these datasets where the first column represents the acronym for the dataset, the second column reflects their full name. The columns tagged with *#num* and *#nom* represent the number of numeric and nominal attributes of the dataset. Presence of missing values in datasets is shown in fifth column of Table 1, and the sixth one is dedicated to the number of classes. The last column shows the population percentage of the most crowded class to the rarest one.

	Dataset	# num	# nom	missing values	# class	class ratio (most/least)
BL	bal	4	0	No	3	46.1% /7.8%
BI	biopsies	24	0	No	2	51.6% / 48.4%
BR	breast-w	9	0	Yes	2	65.5% / 34.5%
BP	bupa	6	0	No	2	58.0% / 42.0%
CM	cmc	2	7	No	3	42.7% / 22.6%
CO	colic	7	15	Yes	2	63.0% / 37.0%
CR	credit-a	6	9	Yes	2	55.5% / 44.5%
FI	fis	21	0	No	2	56.0% / 44.0%
GL	glass	9	0	No	6	35.5% / 4.2%
GR	grid	2	0	No	2	50.0% / 50.0%
HC	heart-c	6	7	Yes	2	54.5% / 45.5%
HH	heart-h	6	7	Yes	2	63.9% / 36.1%
HS	heart-statlog	13	0	No	2	55.6% / 44.4%
HP	hepatitis	6	13	Yes	2	79.4% / 20.6%
IO	ionosphere	34	0	No	2	64.1% / 35.9%
IR	iris	4	0	No	3	33.3% / 33.3%
LB	labor	8	8	Yes	2	64.9% / 35.1%
MX	mx	0	11	No	2	50.0% / 50.0%
PI	pima-indians	8	0	No	2	65.1% / 34.9%
SG	segment	19	0	No	7	14.3% / 14.3%
SN	sonar	60	0	No	2	53.4% / 46.6%
SB	soybean	0	35	Yes	19	13.5% / 1.2%
VE	vehicle	18	0	No	4	25.8% / 23.5%
WI	wine	13	0	No	3	39.9% / 27.0%
ZO	zoo	1	16	No	7	40.6% / 4.0%

Table 1. Datasets details

4.2. Results

Table 2 presents the average accuracy of 10-fold cross validation obtained by the FTL and four well-known state-of-the-art methods on every dataset. The last two rows of the Table 2 show the overall average accuracy and case base reduction rate of each method.

According to Table 2 the average accuracy obtained by FTL method on 25 datasets is the highest value among its peers and way above the CBR. Furthermore, greatest improvements happened in *colic (CO)*, *bal (BL)*, *heart-h (HH)*, *heart-statlog (HS)* and *soybean (SB)* datasets with more than 5 percent improvement in their accuracy. In comparison to baseline CBR, the largest increase in accuracy belongs to *colic (CO)* dataset with about 11.16%.

Data	CBR	ENN	RENN	BBNR	RDCL	FTL
BL	76.16	85.13	85.28	84.01	82.9	**84.64**
BI	83.18	81.43	81.61	82.8	82.4	83.08
BR	95.86	96.11	96.11	96.41	96.12	96.84
BP	62.93	60.27	61.21	63.52	63.2	61.21
CM	44.4	45.61	44.52	46.97	46.9	45.95
CO	73.36	81.77	82.61	83.98	84.22	**84.52**
CR	81.76	85.66	85.36	85.06	85.36	85.37
FI	63.93	64.37	62.54	63.98	65.37	63.93
GL	66.31	69.22	68.7	67.6	67.26	66.05
GR	96.13	95.76	95.92	96.5	96.34	92.8
HC	74.2	80.44	80.43	76.5	76.13	**79.13**
HH	72.83	79	80.72	76.61	76.28	*80.28*
HS	74.07	77.78	77.04	78.15	76.3	**79.63**
HP	78	82.48	82.55	81.26	80.05	81.33
IO	86.93	84.93	84.93	88.6	87.46	86.93
IR	95.33	96	96	94	95.33	96
LB	83.38	79.71	79.71	85.38	85.38	86.48
MX	78.61	76.21	75.87	72.26	76.85	78.51
PI	70.73	75.56	75.8	72.94	74.11	71.13
SG	97.36	95.76	95.54	97.14	97.23	95.32
SN	86.84	81.57	80.71	81.68	84.56	86.84
SB	82.15	88.2	86.88	89.21	89.5	**87.79**
VE	69.44	67.55	67.3	67.17	67.16	68.36
WI	95.64	95.64	94.46	95.64	95.64	95.64
ZO	94.63	92.52	89.85	92.79	94.63	92.52
Acc	79.37	80.75	80.47	80.81	81.07	**81.21**
CB red.	0.00	20.40	**22.51**	15.67	9.29	**15.45**

Table 2. Average results of each method

The average case base reduction of the methods is shown in Table 2 (CB red.). FTL method reduces the size of the case base by 15.45% on average. This value is in the same range of the BBNR method and overall stands as an acceptable case base reduction considering that FTL is a competence enhancement method.

Another aspect of CBR competence is the amount of time a case base maintenance process would take. Table 3 reflects the computational cost of the FTL and compared state-of-the-art methods. To compare the time aspect, two periods have been measured: average time for preprocessing ($T_{Preprocessing}$) and the time for the classification of new cases ($T_{Classification}$) for 25 datasets, are shown in Table 3.

Method	$T_{Preprocessing}$	$T_{Classification}$
CBR	0.0000	0.0051
ENN	1.4849	0.0040
RENN	1.8855	0.0037
BBNR	1.5832	0.0040
RDCL(DL,DCL)	1.8785	0.0043
FTL	3.5635	0.0040

Table 3. Average recorded time

As Table 3 shows, while *RENN*, *BBNR* and *RDCL* preprocessing times are about the same, *FTL* takes twice that long. This leads the *FTL* to a disadvantage. However, at the time for classification, due to case base reduction aspect of *FTL* it stands with the second best time along with *BBNR* and *ENN* algorithms. Note that, all of the methods reduced the classification time as they reduce the size of the case base and the amount of computations required to find and retrieve neighboring cases from the case base. Methods with the largest case base reduction have scored lowered and have better time.

Additionally, to analyze significance of the results from FTL performance, a Friedman [**?**]-Nemenyi [10] test is applied. Friedman test is a statistical test of multiple comparisons that gives a rank to the methods in the evaluation. The lower mean rank means a

better performance of an algorithm. The Friedman test ranks the FTL as the best method. Figure 1 illustrates the Friedman's test result of the FTL and state-of-the-art algorithms along with their significance threshold of each method provided by the Nemenyi test.

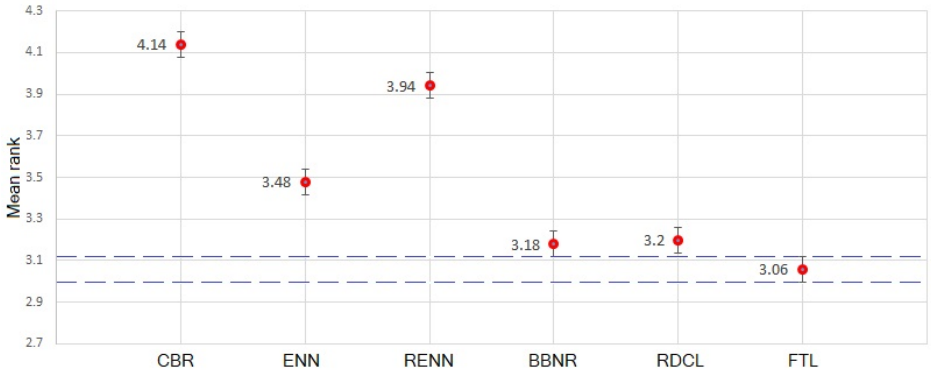

Figure 1. Friedman-Nemenyi test results for methods under analysis

The significance threshold of Nemenyi test shown in Figure 1 is set with a 95% confidence. Two methods that do not overlap their boundaries, are different with a statistically significant degree. The dashed line in the Figure 1, draws the boundaries of significance threshold for FTL method. As the blue dashed lines show, average accuracies obtained by FTL, are significantly better than CBR, ENN, RENN, and RDCL algorithms. The improvements made by FTL compared to BBNR is also pretty close to significance threshold.

5. Conclusions and future work

Feature-Tree Labeling is a case base maintenance algorithm that builds a decision tree on each of the data attributes. These tree classifiers are used to transform the raw case base into a new format. Under the new format every attribute of the cases in the case base is replaced by the class label of its classification with the relative decision tree. After labeling the instances, FTL uses the labeled case base to detect and remove instances that can not be correctly classified by their neighbors. Labeled version of cases that are misclassified by their labeled neighbors tend to have feature values that does not represent their class well according to the other members in the case base. An analysis of FTL performance shows its competence. The average accuracy obtained by FTL on 25 datasets is not only better than the baseline CBR but also above all the well-known state-of-the-art methods that it has been compared to. The Friedman-Nemenyi test shows a statistically significant better performance of the FTL in comparison to baseline CBR, ENN, RENN, and RDCL algorithms.

Feature-Tree Labeling succeeds on improving the general accuracy of a CBR system, but there is room for improving further its competence model. Here we point out two possible future works. First, to experiment with more robust pruning policies for the decision trees in FTL. The Second path for future work would be using the data preparation phase of the FTL with other maintenance cycle strategies. Transformation of data

under FTL can be a model to extract more knowledge from the data space and the quality of individual cases in future classifications.

Acknowledgements

This work has been partially supported by Spanish Ministry of Science and Innovation (grant number TIN2015-71147-C2-2), by the Catalan Agency of University and Research Grants Management (AGAUR) (grants number 2017 SGR 341 and 2017 SGR 574), and by Spanish Network "Learning Machines for Singular Problems and Applications (MAPAS)" (TIN2017-90567-REDT, MINECO/FEDER EU).

References

[1] J. G. Bazan. Hierarchical classifiers for complex spatio-temporal concepts. In *Transactions on Rough Sets IX*, pages 474–750. Springer, 2008.

[2] L. Breiman. *Classification and Regression Trees*. Routledge, 1984.

[3] H. Brighton and C. Mellish. Advances in instance selection for instance-based learning algorithms. *Data mining and knowledge discovery*, 6(2):153–172, 2002.

[4] P.-C. Chang, C.-Y. Fan, and W.-Y. Dzan. A cbr-based fuzzy decision tree approach for database classification. *Expert Systems with Applications*, 37(1):214–225, 2010.

[5] S. J. Delany. The good, the bad and the incorrectly classified: Profiling cases for case-base editing. In *International Conference on Case-Based Reasoning*, pages 135–149. Springer, 2009.

[6] S. J. Delany and P. Cunningham. An analysis of case-base editing in a spam filtering system. In *European Conference on Case-Based Reasoning*, pages 128–141. Springer, 2004.

[7] D. Dheeru and E. Karra Taniskidou. UCI machine learning repository, 2017.

[8] M.-J. Huang, M.-Y. Chen, and S.-C. Lee. Integrating data mining with case-based reasoning for chronic diseases prognosis and diagnosis. *Expert systems with applications*, 32(3):856–867, 2007.

[9] D. B. Leake and D. C. Wilson. Categorizing case-base maintenance: Dimensions and directions. In *European Workshop on Advances in Case-Based Reasoning*, pages 196–207. Springer, 1998.

[10] P. Nemenyi. Distribution-free multiple comparisons. In *Biometrics*, volume 18, page 263. INTERNATIONAL BIOMETRIC SOC 1441 I ST, NW, SUITE 700, WASHINGTON, DC 20005-2210, 1962.

[11] P. Perner. Mining sparse and big data by case-based reasoning. *Procedia Computer Science*, 35:19–33, 2014.

[12] M. M. Richter and R. O. Weber. *Case-Based Reasoning: A Textbook*. Springer, 2013.

[13] I. Tomek. An experiment with the edited nearest-neighbor rule. *IEEE Transactions on systems, Man, and Cybernetics*, (6):448–452, 1976.

[14] D. L. Wilson. Asymptotic properties of nearest neighbor rules using edited data. *IEEE Transactions on Systems, Man, and Cybernetics*, (3):408–421, 1972.

[15] D. R. Wilson and T. R. Martinez. Reduction techniques for instance-based learning algorithms. *Machine learning*, 38(3):257–286, 2000.