The Thirty-Second International Florida
Artificial Intelligence Research Society Conference (FLAIRS-32)

C2C Trace Retrieval: Fast Classification Using Class-to-Class Weighting

Xiaomeng Ye
Indiana University Bloomington
xiaye@iu.edu

Abstract

Traditional case-based classification methods are based on feature similarity. In contrast, class-to-class (C2C) weighting also considers whether the difference between two cases has been seen before. Combined with instance-specific weighting, C2C weighting learns the local *patterns of both similarities and differences* (shortened as *patterns*). Once C2C weightings has learned the pattern between case A of class C_1 and some set of cases R of class C_2, given a query Q whose difference from A matches the pattern between A and R, then we can skip cases around A and continue the search for near neighbors around R.

Based on this, we developed an algorithm, C2C trace retrieval, which quickly traverses promising cases, retrieves relevant cases from different classes, and provides an informed hypothesis of the query's class. C2C trace retrieval achieves great efficiency at a reasonable cost of accuracy. Therefore, C2C trace retrieval can be used as a fast classification method or as the first pass for a more sophisticated method.

Introduction

The focus of k-nearest neighbor algorithms (k-NN) is to find cases with similar feature values for a query case (Wettschereck, Aha, and Mohri 1997). The presumption of k-NN is that cases of the same class share similar (important) features. Along this line of study, class-to-class (C2C) weighting (Ye 2018) learns the patterns of similarities and differences between classes, and uses the patterns in classification tasks. Based on the assumption that cases of class C_1 are different from cases of class C_2 in a systematic way, C2C weighting opens up a brand-new avenue of case retrieval and explanability. However, preliminary experiments show that C2C weighting performs slightly worse than global weighting because the weightings can be skewed.

The goal of this study is two-fold: 1) we improve C2C weighting by combining it with instance-specific weighting. We also train the native and non-native class weightings of C2C weighting separately. In the improved C2C weighting, a comparison between a query and a case identifies not only whether they match, but also where else to look if their difference has been seen before. 2) Using the novel local knowledge provided by C2C weighting, we develop an agent-based case retrieval method, C2C trace retrieval. We use C2C trace retrieval to find relevant and yet diverse cases, the majority vote of which provides an informed guess of the query's class.

C2C trace retrieval is a retrieval method based on patterns of both similarities and differences (shortened as patterns). At inference time, its time complexity does not hinge on the size of the case base. We compare C2C trace retrieval with other case retrieval methods and locate its uniqueness in the spectrum of case retrieval methods. We evaluated C2C trace retrieval on multiple data sets and compare its classification accuracy and number of similarity calls made with k-NN, k-dimensional tree and locality-sensitive hashing. Our results show that C2C trace retrieval achieves superior efficiency at a reasonable cost of accuracy.

Background

Case Retrieval Methods in CBR

For the purpose of this study, we present a few assumptions before discussing case retrieval methods: 1) Each case is represented by a vector of nominal and numerical features along with a class label; 2) Each case has only one class or label, and the training cases include cases from all classes; 3) The similarity measure for each feature is given. A detailed description of case retrieval methods can be found in (Richter and Weber 2013). Most relevant methods are listed below.

Two-stage retrieval methods filter for promising cases in the first stage before conducting a more costly case comparison. These methods reduce the search space to reduce the cost of case retrieval. In MAC/FAC ("many are called but few are chosen"), the MAC stage narrows down the search space by roughly searching among probe cases, and the FAC stage further confirms the match of the found cases using a more sophisticated similarity measure (e.g. structural mapping engine) (Gentner and Forbus 1991). Protos uses a prototypical case base along with the real case base. Given an input query, it first finds the closest prototypical case, which then redirects to a set of real cases (Ray Bareiss, W. Porter, and Wier 1990). Fish and Shrink is a two-stage method where the Fish stage computes the distances between the query and some cases, and the Shrink stage uses triangle inequality to filter out cases (Schaaf 1996).

A k-dimensional tree (KD-tree) divides points in a k-dimensional space into a binary tree (Bentley 1975). When building the tree, an attribute and a critical value are selected for every node. Cases with corresponding attribute values lower than the critical value are assigned to one side of the node, while the rest of cases are assigned to the other side. Search is done by traversing the tree.

Locality-sensitive hashing (LSH) is a good representative of a family of approximate nearest neighbors algorithms (Indyk and Motwani 1998). LSH reduces case retrieval time by utilizing hash functions to hash similar cases into the same bins. The nearest neighbors of an input query can be approximated by applying the hash functions. Different hashing algorithm suits the need of different similarity measures.

Feature Weighting Methods in k-NN

In k-NN, the similarity between two cases is measured by the combination of differences of all features, e.g. Euclidean distance. The difference of a feature can be weighted to (de)emphasize its importance in the overall measure (Wettschereck, Aha, and Mohri 1997). Weightings can also be assigned based on the classes of cases (Marchiori 2013). For situations when the importance of features varies across regions of the case space, local weightings are developed to assign weightings in each region (Aha and Goldstone 1992; Friedman 1994; Ricci and Avesani 1995).

Instance-specific weighting is a local weighting method where each case has its own set of feature weightings. The ISAC system (Bonzano, Cunningham, and Smyth 1997) modifies the instance-specific feature weightings using introspective learning based on classification performance. If the retrieved case correctly predicts the class of the query case, ISAC increases weightings of matching features and decreases weightings of unmatching features. Similarity scores between cases of the same class are thus driven higher. If the class of the retrieved case does not match that of the query case, ISAC decreases weightings of matching features and increases weightings of unmatching features, to make the two cases more distant.

Class-to-class (C2C) Weighting

The weighting methods described in the above section rely on the presumption that cases of the same class share similar features. k-NN algorithms using such weightings are similarity-based retrieval methods (López de Mántaras et al. 2005). C2C weighting adds the additional assumption that cases of two different classes have certain features that differ consistently. In addition to the similarity patterns within each class, which traditional weighting methods do, C2C weighting aims to find the patterns between classes to potentially apply these patterns in classification (Ye 2018).

C2C weighting as it is used by k-NN can be described by the following equations:

$$A = \{A_1, A_2, ..., A_d, C_i\} \in CB, \tag{1}$$

$$W = \{\sum_{i=1}^{m}\sum_{j=1}^{m} w_{ij}\}, \tag{2}$$

$$w_{ij} = [w_{ij1}, w_{ij2}, ..., w_{ijd}], \tag{3}$$

$$fd(v_t, v_b) = 1 - 2\frac{|v_t - v_b|}{v_{max} - v_{min}}, \tag{4}$$

$$activation(A, B, w_{ij}) = \sum_{v=1}^{d} w_{ijv} * fd(A_v, B_v). \tag{5}$$

The equations are explained as: (1) Let there be m classes, a case A from the case base CB is defined as the composite of d features, A_1 to A_d, and a class label C_i. C_i is the class label representing the ith class. (2) W is the set of all class weightings, where (3) each class weighting w_{ij} is a vector of d feature weightings describing the pattern from C_i to C_j. w_{ijv} is the vth feature weighting in the class weighting wij. (4) $fd(v_t, v_b)$ measures the feature distance between two features v_t and v_b with range $[-1, 1]$. (5) $activation(A, B, w)$ is the activation score between two cases A and B under class weighting w_{ij}.

We refer to w_{ij} as a native class weighting if $i = j$, and as a non-native class weighting if $i \neq j$. Native class weightings capture the pattern of similarity within the same class, which traditional feature weightings also do. Non-native class weightings capture the patterns between classes. Most importantly, non-native class weightings can have negative feature weightings.

In (4), if two features are similar then the feature distance $fd(v_t, v_b)$ is positive; if they are different then the feature distance is negative. In (5), a negative feature distance multiplied by a negative feature weighting positively contributes to the total activation score. This mechanism rewards patterns of differences in the presence of negative feature weightings.

In $activation(A, B, w)$, the choice of w depends on the classes of A and B. In testing mode, case A is a case from the case base CB, and case B is the query whose class (C_j) is unknown. We suggest all possible classes of B, and use the class with the highest activation score(s). This is how k-NN may use C2C weighting to predict the class of a query:

$$B = \{B_1, B_2, ..., B_d, C_j\}, \tag{6}$$

where C_j is unknown. In 1-NN (when $k = 1$), C_j is determined by:

$$C_j = \arg\max_{1 \leq j \leq m} activation(A, B, w_{ij}) \forall A \in CB \tag{7}$$

and this can be extended to k-NN by a majority vote on C_j from the k highest activation scores.

The implementation and training of C2C weighting directly follows the example of the ISAC system (Ye 2018). Weightings for matching features in correct predictions and unmatching features in incorrect predictions are increased, while other weightings are decreased. We modify the weightings by:

$$w_{ijv}(t+1) = w_{ijv}(t) \pm \delta * \frac{F_c}{K_c}, \tag{8}$$

where $w_{ijv}(t)$ is the v-th feature weighting at time step t. δ is a fixed value. F_c is the number of times the case has been falsely retrieved and K_c is the number of times the case has been correctly retrieved. In short, the feature weightings are changed so that frequently seen patterns between classes can leads to high activation scores in testing mode.

Improved C2C Weighting

There are two flaws in the original design of C2C weighting: 1) Cases of class C_i are used to train the native class weighting w_{ii}, and cases of a different class C_j to train the non-native class weighting w_{ij}. However, the training of traditional feature weightings involves both positive and negative examples. Therefore, native class weightings are often not properly trained and skewed. 2) If cases of different classes are spread widely, each case may have its own patterns with other classes instead of a single pattern defined between the classes.

Corresponding to each flaw, we implement two improvements for C2C weighting: 1) We train native class weightings with both positive and negative examples; 2) We integrate C2C weighting with instance-specific weighting (shortened as IS weighting). In other words, equation 2, 3, and 5 are changed into the following:

$$W = \{ \sum_{A \in CB} \sum_{i=1}^{m} \sum_{j=1}^{m} w_{Aij} \}, \tag{9}$$

$$w_{Aij} = [w_{Aij1}, w_{Aij2}, ..., w_{Aijd}], \tag{10}$$

$$activation(A, B, w_{Aij}) = \sum_{v=1}^{d} w_{Aijv} * fd(A_v, B_v). \tag{11}$$

In the resulting instance-specific C2C weighting (shortened as IS-C2C weighting), native class weightings inherit the benefits of IS weighting and learn the patterns of similarity *in local regions*, while non-native class weightings learn the patterns *between local regions in different classes*.

The new training scheme is described in Algorithm 1. Native class weightings are initialized with feature weightings equal to 1, while non-native class weightings are initialized with feature weightings equal to -1.

Understanding C2C Weighting

C2C weighting was previously shown to perform slightly worse than global weighting (Ye 2018). However, for the improved IS-C2C weighting, accuracy matches that of its counterpart, IS weighting. To understand this, we need to clarify a few concepts:

The foundation of IS-C2C weighting is IS weighting. The native class weightings in IS-C2C weighting are essentially IS weighting. When using IS-C2C weighting, the majority of best matched cases in k-NN are actually suggested by native class weightings. Additionally, a non-native class weighting is trained only if the weighting leads to some case retrievals in the training phase. In some scenarios, not many non-native class weightings are trained, thus reducing the overhead of IS-C2C weighting.

Non-native class weightings are local and directional. In IS-C2C weighting, a non-native class weighting is trained for the pattern from one case to a local region in some other class. The knowledge is local knowledge anchored to one case. The suggestive power of a non-native class weighting is best used along with other native and non-native class weightings. For example, both New York and Philadelphia are northeast of Washington, so further information is needed to differentiate between New York and Philadelphia.

Algorithm 1 IS-C2C Weighting Training Algorithm

Require: CB, the case base.
for each $w_{Aij} \in W$ **do**
  set $w_{Aijv} = 1$ where $1 \leq v \leq d$
  if $i \neq j$ **then**
    set $w_{Aijv} = -1$ where $1 \leq v \leq d$
  end if
end for
repeat
  for each $B = \{B_1, B_2, ..., B_h, C_{j_1}\} \in CB$ **do**
    for each C_i where $1 \leq i \leq m$ **do**
      retrieve k cases $A1$ ... Ak of class C_i
      with the highest $activation(A, B, w_{Aij_2})$
      where $A \in \{A1, ..., Ak\}$
      for each $A \in \{A1, ..., Ak\}$ **do**
        update F_A, K_A based on j_1, j_2
        for each v where $1 \leq v \leq d$ **do**
          $new\ w = w_{Aij_2v} \pm \delta * \frac{F_A}{K_A}$
          if $i = j_2$ and $new\ w < 0$ **then**
            set $new\ w$ to 0.001
          end if
          $w_{Aij_2v} = new\ w$
        end for
        Normalize w_{Aij_2}
      end for
    end for
  end for
until training converge / max # of iterations reached

C2C Trace Retrieval

Traditional weighting methods only show how well a stored case A matches a query Q. On the other hand, IS-C2C weighting can suggest whether the difference is similar to one of its learned patterns when the match fails. If the difference between Q and A is similar to the learned difference between some region R and A, it suggests Q and R share similar features. C2C trace retrieval repeatedly uses the learned difference patterns to suggest promising search areas, thus skipping vast majority of the irrelevant cases.

Assuming case A is of class C_i, the weighting w_{Aij} tells us about the pattern between the case A and a region R in class C_j. To further take advantage of this knowledge, we can also record cases $(R1, R2...)$ in that region R. This is done by keeping a log and recording cases $(R1, R2...)$ of class C_j that leads to the training of weighting w_{Aij}. We refer to this log as the training history of weighting w_{Aij}. The length of a training history shows the number of times a weighting is trained. Notice that cases that are not of class C_j may also influence w_{Aij} (as negative examples) but they are not recorded in this training history.

C2C trace retrieval in its current design is an agent-based retrieval method where each agent is composite of an extendable list of cases, or a trace. It functions as following:

1. Given an input query, the agents starts out by randomly pick t cases from every class. Cases with multiple well-trained non-native class weightings are preferred.
2. Each case chosen in Step 1 functions as the initial case of a trace. As there are m classes, there will be $m \cdot t$ traces .
3. For each trace, the agent compares the last case A on the trace with the query, using each of the native and non-native class weightings anchored on A.
4. Assume the highest activation score is yielded by weighting w_{Aij}. The agent retrieves the most recent case S from the training history of weighting w_{Aij}, and appends case S to the tail of the current trace.
5. If case S already appears on this trace, or if the training history of w_{Aij} is empty, the current trace is stable.
6. Repeat step 3-5 until all traces are stable.
7. The agents combine all unique cases in traces into one pool, and carry out a k-NN using IS-C2C weighting in this pool. The k cases with the highest scores are retrieved.

The number of traces t is a trade-off decision between efficiency and accuracy. If $t = 1$, C2C trace retrieval is just a crude guess. If t is set to the number of all cases then C2C trace retrieval produces the same prediction as a k-NN using IS-C2C weighting. For best performance, the number of traces starting in a class should be proportional to the number of cases in the class.

The choice of initial cases in Step 1 is crucial. Picking cases whose training history is short or empty will lead to short and uninsightful traces. A long training history indicates that the weightings of a case are trained multiple times to reflect strong patterns across the case space. In our implementation, we order cases by the lengths of their training history and use a Poisson random number generator ($\lambda = 1$) to pick initial cases.

Computational Complexity

Assuming the size of case base is n, number of attributes d, number of classes m, number of training iterations I, this section discusses computational complexities. IS weighting works with one set of feature weightings for each case while IS-C2C weighting does the same with at most m sets.

In k-NN, the training complexity of IS weighting or global weighting is $O(dIn^2)$. The training complexity of IS-C2C weighting is $O(dImn^2)$. The retrieval time complexity of IS weighting is $O(dn)$, and that of IS-C2C weighting is $O(dmn)$.

As for the retrieval time complexity of C2C trace retrieval, there are $O(mt)$ traces in total, each trace is of an expected length L, every case requires $O(m)$ similarity measure calls and each call is $O(d)$. Therefore the time complexity for building traces is $O(dm^2tL)$. After gathering all traces, we have in total $O(mtL)$ cases. To select k most similar cases, we can reuse similarity scores from the previous stage.

In sum, the retrieval time complexity of C2C trace retrieval is $O(dm^2tL)$. In retrospect, L is often less than m, as a trace normally would not re-enter a class after leaving it. t is a user-set value, and a small t is good enough when a class that is not wide-spread or when m is big.

	iris	echo	WDBC	Cleveland	MNIST(1)	MNIST(2)	letter	red	white
d	4	6	30	13	196	196	16	11	11
n	150	97	569	303	1478	706	2000	1599	4898
m	3	2	2	2	2	10	26	6	6

Table 1: Data sets information

For KD-tree, the training complexity is $O(n \log n)$. For 1-NN, the testing complexity is usually $O(\log n)$, but $O(n)$ in the worst scenario (Bentley 1975). The testing complexity varies because the search for nearest neighbors might require backtracking in a KD-tree. The cost of backtracking hinges on the dimensionality of the domain. In practice, the testing complexity is $O(2^d + logn)$.

For LSH, the training complexity is shown to be polynomial and the testing complexity polynomial in d and $\log n$. In practice, it offers orders of magnitude improvement in efficiency over KD-tree (Indyk and Motwani 1998). In testing, the LSH implementation we use is $O(dHL)$ in hashing and $O(bLN_c)$ in collision resolution, where H is the number of dot products per hash (or the number of indices for a hash code), b is the average bin size, L is the number of projections (or hashes), N_c is the expected number of collisions per projection (Slaney and Casey 2008). However, LSH poses some complexity in design as it requires a suitable choice of a family of hashing functions for a given problem domain.

Performance

We examine the performance of C2C trace retrieval on classification tasks, and compare it with k-NN using global/C2C/IS-C2C weighting, KD-tree, and LSH. We examine the algorithms under eight data sets: Wisconsin diagnostic breast cancer (WDBC), Heart Disease Cleveland, echocardiogram (echo), iris, red and white wine, letter recognition (Dheeru and Karra Taniskidou 2017), and MNIST (LeCun and Cortes 2010). We modified the letter and MNIST data set to make the experiment runnable on computers with 8GB RAM. 10% of the letter data are used. MNIST images are blurred from 28*28 to 14*14 and part of all cases are used in two experiments: (1) only two of digits (0 and 1) about 700 samples each; (2) all ten digits, about 70 samples each. For white wine, we merge 5 cases of class 9 to class 8, to make white wine more comparable with red wine where there are no class 9 cases. Details about the data sets are listed in Table 1.

We use the implementation of k-NN and KD-tree from the Java-ML library (Abeel, de Peer, and Saeys 2009). The C2C feature weightings are trained until they converge (all within 20 iterations). For all experiments, the performance of each algorithm is evaluated with 10-fold cross validation. Lastly, we use $k = 5$ and $t = 5$ (except for red and white wine, where we use $t = 1$ because some of the classes are very small).

As Euclidean distance applies to all our data sets, we use the package TarsosLSH (Six 2013) because this LSH implementation supports the Euclidean hash family, which is based upon (Slaney and Casey 2008). We used $H = 10, L = 30$. H and L are explained before. To determine the width of a projection, w, the algorithm first searches for the clos-

est neighbor within a time limit for 30 trials, and then the product of the average distance to the closest neighbor and a constant multiple (we use 9) serves as the width. In reality, researchers can tune these parameters for maximal efficiency or for a probabilistic guarantee of finding the true nearest neighbor. For the purpose of this study, fine tuning is not needed, as we use LSH as a classifier and we use the majority vote of the candidates in found bins after hashing. We want to advise the readers that the evaluation here is by no means a definitive comparison between the algorithms. Each algorithm may be fine tuned to achieve better performance in certain domains, for example, by careful choices of k in k-NN, the balancing of KD-tree, different choices of H, L, and w in LSH, and the number of traces per class t in C2C trace retrieval.

Accuracy is measured by the average accuracy across all classes, weighted by the size of each class. The results are listed in Table 2. C2C trace retrieval is generally worse when compared to other methods. While the gain in efficiency is small, the loss in accuracy is also small, as shown in the experiments with Cleveland, echo, and iris. When C2C trace is greatly faster, it loses more accuracy, as shown in the experiments with WDBC, MNIST(1), red and white wine.

Efficiencies of the algorithms are loosely comparable since they all require similarity measure calls between cases (activation measure call in the case of C2C), which constitute the most expensive operations of each algorithm. More specifically, we measure: the number of similarity measure calls made in C2C trace retrieval and k-NN, the number of nodes inspected in KD-tree (traversing a node requires at least one similarity measure call between the node and the query) (Moore 1991), and the number of items in the retrieved bins after hashing (ranking the items requires similarity measure calls). These numbers implies the numbers of similarity measure calls made in the algorithms. We measure the sum of these numbers for all cases in the 10-fold cross validation. The results are listed in Table 3 and the costs per query case are in Table 4. Note that LSH requires additional $H \cdot L = 300$ dot products per query case during the hashing phase, and a similarity measure call is essentially a dot product. The cost of hashing for LSH is included in the Tables 4.

In our experiments with WDBC, MNIST(1), red and white wine, C2C trace retrieval greatly reduces the number of similarity measure calls. This effect is not obvious when n or d is small, as in iris and echo.

However, C2C trace retrieval has an unusually high number of similarity measure calls in MNIST(2) and letter. Using letter as an example, the C2C parameters are $m = 26$, $t = 5$, $L \simeq 2.2$. The total number of calls for every query is $m^2tL = 7436$, multiplying this with the number of cases $n = 2000$, we get 14872000. This matches our observation and computational complexity analysis where the complexity of C2C trace retrieval is $O(dm^2tL)$. When m is large and n is small, m^2 catches up to n and the efficiency advantage of C2C trace retrieval is diminished. In these scenarios, we can reduce t to reduce the cost of C2C trace retrieval.

In summary, C2C trace retrieval excels in terms of efficiency when n or d is huge. When m is huge, we can avoid creating too many traces by reducing t.

	iris	echo	WDBC	Cleveland	MNIST(1)	MNIST(2)	letter	red	white
k-NN	0.973	0.494	0.929	0.667	0.993	0.928	0.983	0.629	0.633
C2C	0.906	0.463	0.913	0.646	0.477	0.818	0.976	0.711	0.722
IS-C2C	0.964	0.515	0.966	0.828	0.903	0.830	0.982	0.712	0.733
KD-tree	0.973	0.494	0.929	0.667	0.993	0.928	0.983	0.667	0.673
LSH	0.951	0.505	0.927	0.650	0.989	0.927	0.983	0.659	0.684
C2C trace	0.916	0.526	0.759	0.769	0.885	0.884	0.953	0.634	0.601

Table 2: Weighted Average Accuracy

	iris	echo	WDBC	Cleveland	MNIST(1)	MNIST(2)	letter	red	white
k-NN	20250	8466	291384	82626	2022300	448590	3600000	2301120	21591362
C2C	60750	16932	582768	165252	3727296	3968040	93600000	13806720	129548172
IS-C2C	38985	14711	350334	165252	3852660	3716635	41398200	4404926	34161235
KD-tree	11433	10480	357745	121350	2577812	843482	3105318	1778233	11206435
LSH	2551	2814	37307	27142	546162	304615	1116846	108689	213548
C2C trace	21491	5104	28358	26174	94820	746205	14943012	76820	294804

Table 3: Number of Distance Measure Calls

	iris	echo	WDBC	Cleveland	MNIST(1)	MNIST(2)	letter	red	white	
k-NN	135	87	512	273	1368	**635**	1800	1439	4408	
C2C	405	174	1024	545	2521	5620	46800	8634	26449	
IS-C2C	259	151	615	545	2606	5264	20699	2754	6974	
KD-tree	**76**	108	629	400	1744	1195	1552	1112	2288	
LSH	17	29	65	90	370	431	**558**	68	44	(+300)
C2C trace	143	**52**	**50**	**86**	**64**	1057	7472	**48**	**60**	

Table 4: Number of Distance Measure Calls Per Query Case (Best Numbers Highlighted)

Discussion

Drawbacks

Nondeterministic Nature The agent-based design of C2C trace retrieval is nondeterministic. We randomly choose cases that are promising in starting informative traces by choosing cases with the longest training history. Because of the nondeterministic nature of C2C trace retrieval, we lack theoretical discussion about the error rate bound of the classification.

Memory Usage and Training Time When n or d is huge, IS-C2C weighting can become too cumbersome to train and store. This is a direct issue inherited from IS weighting, forcing us to down sample some of the data sets in experiments.

There are two ways to alleviate the problem: 1) Since the weighting w_{Aij} associated with a case A can be trained independently from other weightings, we may complete training some feature weightings and store them in a hard-drive-based database before training more feature weightings. In testing, C2C trace retrieval starts with t cases from every class, so it is only necessary to load the feature weightings of these cases. 2) condensing the case base so we do not need to store feature weightings for all cases.

Condensing the Case Base

IS weighting and IS-C2C weighting store some feature weightings for every case, which can be problematic when the case base is huge. Condensed nearest neighbors (Hart 2006) can be used to reduce the case bases under IS weighting. IS-C2C weighting, inheriting the drawbacks of IS weighting, also inherits from it improvement techniques, such as CNN. Interestingly, when using C2C weighting, a whole class might be removed if the non-native class weighting of another class can correctly predict this class. This phenomenon is explained in (Ye 2018).

Relation with Index-based Retrieval Methods

An index-based retrieval method builds an index to look up cases in the training stage. Some examples are Protos, KD-tree, and LSH. In C2C trace retrieval, cases with the longest training histories function as the initial indices. The training of IS-C2C weighting automatically identifies these indices. Different from traditional indices, which suggest "whether to look into a neighborhood", IS-C2C weighting indices also suggest "where else to look". IS-C2C weighting indices might also be useful for other index-based retrieval methods. In fact, in Protos, one "indexing mechanism is based on the differences between pairs of 'neighboring' exemplars in the category structure" (Ray Bareiss, W. Porter, and Wier 1990).

Future Directions

C2C trace retrieval retrieves cases from multiple classes. The heterogeneous cases offers a novel explanation for the final classification. These cases also serve as interesting alternatives to the query, allowing the user to explore similar cases of different classes.

C2C weighting can also be used in detecting anomalies or unseen classes. C2C weighting learns and stores difference patterns in the training data. Given a query, C2C weighting is able to detect if the query conforms to difference patterns learned from the training data. If the query shows signs of new difference patterns while not following the already learned ones, the query is possibly an anomaly or of an unseen class.

Summary

Taking advantage of the local knowledge offered by IS-C2C weighting, C2C trace retrieval provides a quick approximate solution for the task of classification. Case retrieval methods often suffer when the size or dimensionality of the case base is huge, rendering C2C trace retrieval an attractive alternative. Multiple experiments demonstrated that C2C trace retrieval achieves great efficiency at a reasonable cost of accuracy.

References

Abeel, T.; de Peer, Y. V.; and Saeys, Y. 2009. Java-ml: A machine learning library. *The Journal of Machine Learning Research* 10(931–934).

Aha, D. W., and Goldstone, R. L. 1992. Concept learning and flexible weighting. In *Proceedings of the Fourteenth Annual Conference of the Cognitive Science Society*, 534–539. Erlbaum.

Bentley, J. L. 1975. Multidimensional binary search trees used for associative searching. *Commun. ACM* 18(9):509–517.

Bonzano, A.; Cunningham, P.; and Smyth, B. 1997. Using introspective learning to improve retrieval in CBR: A case study in air traffic control. In Leake, D., and Plaza, E., eds., *Proceedings of the 2nd International Conference on Case-Based Reasoning (ICCBR-97)*, volume 1266 of *LNAI*, 291–302. Berlin: Springer.

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

Friedman, J. H. 1994. Flexible metric nearest neighbor classification. Technical report, Stanford University.

Gentner, D., and Forbus, K. 1991. MAC/FAC: A model of similarity-based retrieval. In *Proceedings of the Thirteenth Annual Conference of the Cognitive Science Society*, 504–509. Chicago, IL: Cognitive Science Society.

Hart, P. 2006. The condensed nearest neighbor rule (corresp.). *IEEE Trans. Inf. Theor.* 14(3):515–516.

Indyk, P., and Motwani, R. 1998. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*, STOC '98, 604–613. New York, NY, USA: ACM.

LeCun, Y., and Cortes, C. 2010. MNIST handwritten digit database.

López de Mántaras, R.; McSherry, D.; Bridge, D.; Leake, D.; Smyth, B.; Craw, S.; Faltings, B.; Maher, M.; Cox, M.; Forbus, K.; Keane, M.; Aamodt, A.; and Watson, I. 2005. Retrieval, reuse, revision, and retention in CBR. *Knowledge Engineering Review* 20(3).

Marchiori, E. 2013. *Class Dependent Feature Weighting and K-Nearest Neighbor Classification*. Berlin, Heidelberg: Springer Berlin Heidelberg. 69–78.

Moore, A. W. 1991. An intoductory tutorial on kd-trees.

Ray Bareiss, E.; W. Porter, B.; and Wier, C. 1990. Protos: An exemplar-based learning apprentice. 29:549–561.

Ricci, F., and Avesani, P. 1995. *Learning a local similarity metric for case-based reasoning*. Berlin, Heidelberg: Springer Berlin Heidelberg. 301–312.

Richter, M., and Weber, R. 2013. *Case-Based Reasoning: A Textbook*. Berlin: Springer.

Schaaf, J. 1996. Fish and shrink. a next step towards efficient case retrieval in large scaled case bases. In Smith, I., and Faltings, B., eds., *Advances in case-based reasoning*, 362–376. Berlin: Springer Verlag.

Six, J. 2013. Tarsoslsh. https://github.com/JorenSix/TarsosLSH.

Slaney, M., and Casey, M. 2008. Locality-sensitive hashing for finding nearest neighbors [lecture notes]. *IEEE Signal Processing Magazine* 25(2):128–131.

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):273–314.

Ye, X. 2018. The enemy of my enemy is my friend: Class-to-class weighting in k-nearest neighbors algorithm. In *Proceedings of the Thirty-First Florida Artificial Intelligence Research Society Conference*, 389–394.