The Thirty-First International Florida
Artificial Intelligence Research Society Conference (FLAIRS-31)

The Enemy of My Enemy Is My Friend: Class-to-Class Weighting in K-Nearest Neighbors Algorithm

Xiaomeng Ye
Indiana University Bloomington
xiaye@umail.iu.edu

Abstract

The K-nearest neighbors algorithm (k-NN) is widely used in instance-based learning and case-based reasoning. The basic k-NN approach has been refined and augmented in many ways, including the use of local weighting, asymmetric metrics, and class-specific weighting, which enables the use of different similarity criteria for each class. This paper extends class-specific weighting with a method we call class-to-class (C2C) weighting. Beyond class-specific weighting, which learns feature weightings to identify the most similar cases to a class, C2C weighting also focuses on learning differences between classes to potentially apply those differences to classification. Once C2C weighting has learned how class C_1 is different from class C_2, given a new case is different from a C_1 case in a way similar to the way C_2 cases are different from C_1 cases, then the new case is assigned to class C_2.

C2C offers two potential advantages: First, unlike global weighting, it is robust to deletion of the cases in a given class, because non-native class weightings can still make relatively good predictions. We demonstrate experimentally that this can be true even when a whole class of cases is dropped. Additionally, C2C might provide a new potential form of explainability, in explaining classifications based on pattern of differences. Preliminary results suggest that in normal settings C2C offers accuracy comparable to standard methods, though slightly lower. However, with our initial learning method, the native class weightings of C2C weighting are easily skewed and can lead to worse performance than traditional global weightings. We argue this is not an intrinsic flaw in C2C weighting, but rather an issue in the combination of C2C weighting with global weighting, and propose an approach to address this issue.

Background

The K-nearest neighbors algorithm (Fix and Hodges 1951; Cover and Hart 1967) is a widely used case classification algorithm that has received considerable attention in case-based reasoning (Aha, Kibler, and Albert 1991). For classification, given an input query, k-NN retrieves the most similar k cases. Based on the classes of the k retrieved cases, it suggests the class of the input case, often by majority vote. The presumption of k-NN is that cases with similar features are likely to share the same class.

The standard method for assessing similarity between the input query and stored cases is to measure the difference between every feature and combine the individual distances, e.g., by Euclidean distance or a sum of differences (Wettschereck, Aha, and Mohri 1997). As different features have different importance, each feature is given a weighting (Wettschereck, Aha, and Mohri 1997). Normally, these weightings are global. However, this method is susceptible when the local landscape has its own norm which is different from that of the global landscape. For this reason, local weightings are developed to assign weightings to features in smaller scopes (Aha and Goldstone 1992; Friedman 1994; Ricci and Avesani 1995). The most fine-grained local weighting method is instance-specific, namely, every instance has its own set of feature weightings.

Another avenue of effort assigns feature weightings in regard to the classes (Marchiori 2013). In this method, each class has its own feature weightings. Although the work shows that class-specific weighting is "useful to unravel interesting properties of features with respect to a class of interest", it also admits that class-specific weighting is not better at predicting new cases because ultimately it is splitting the traditional weighting into several parts.

This paper continues the idea of class-specific weighting. We developed the class-to-class (C2C) weighting and combined it with global weighting. We tested it on various data sets as well as synthetic data sets, and compared the prediction accuracy with that of a global weighting method.

Theory of C2C Weighting

Existing weighting methods all focus on the fact that among cases of the same class, certain features tend to be similar. By assigning higher feature weightings to these features, the weighting methods reward this norm in similarities.

We suggest a novel weighting method, named class-to-class (C2C) weighting. C2C weighting looks at the pattern of differences from one class to another. The assumption is that cases of class C_1 are different from cases of class C_2 in a systematic way. Given a new input case, if C_1 cases are different from the new case in the same fashion, then this new case is likely to be of class C_2. Metaphorically speaking, the enemy (the input case) of my enemy (C_1 cases) is

my friend (C_2 cases).

Note that C2C weighting does not reject traditional weighting. In other words, C2C weighting can reward both similarity pattern and difference pattern between classes.

We use the notation C_1-C_2 to represent a set of C2C feature weightings that reflect the trend of similarity/difference from C_1 to C_2.

C2C weighting functions as an extension to existing weighting methods. It can be attached to any of the weighting methods described in the Background section. For our purpose here, we only study extending traditional global weighting with C2C weightings.

Implementation

A global weighting offers only one set of feature weightings. Each feature weighting represents the importance of a corresponding feature. Let the number of classes be M, a C2C weighting would offer in total $M \times M$ sets of feature weightings. The memory requirement is thus much higher than global weighting. If the difference patterns are symmetric, namely weighting C_1-C_2 is the same as C_2-C_1, then this memory requirement can be reduced by half. Additionally, the additional memory usage is not bad at all when M is a small number.

Next we present a few settings needed for implementation, followed by the pseudo-code for training and using C2C weighting.

Feature Metric

We built a case-based reasoning (CBR) system using the feature metric directly from ISAC (Bonzano, Cunningham, and Meckiff 1996; Bonzano, Cunningham, and Smyth 1997). Other feature metrics or similarity measures (Richter 1993; Weinberger, Blitzer, and Saul 2006) could also be used but the decision would influence subsequent implementation details. While ISAC actually uses local weighting, for the purpose of this paper, our CBR system uses global weighting.

The feature metric is used to measure the distance between two values of a certain feature. Given a case base, we first find out the maximum value v_{max} and minimum value v_{min} of the feature. So the range of the feature is $v_{max} - v_{min}$. Then the feature difference score (fd) between two values of that feature, v_t and v_b, is calculated as

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

A fd is a number between -1 and 1. We consider a positive fd as an indication of similarity between the two feature values, and a negative fd as an indication of difference.

In ISAC, a feature weighting w_v is multiplied with the feature difference score fd for feature v to produce an activation value. The similarity between two cases is measured by the activation sum of all features. For a case, the other case with the highest activation sum is the most similar case to this case. In C2C weighting, this is slightly different and will be explained in the Case Metric section.

Updating Feature Weighting

As the case with feature metric, we use the same updating policies as ISAC(Bonzano, Cunningham, and Smyth 1997) to change the feature weightings in the training process. The updating policies are:

- Good matching up: if the retrieved case is of the same class as the input case (good), we increase the weightings of features with a positive fd (matching).
- Good unmatching down: if the retrieved case is of the same class as the input case (good), we decrease the weightings of features with a negative fd (unmatching).
- Bad matching down: if the retrieved case is of a different class from the input case (bad), we decrease the weightings of features with a positive fd (matching).
- Bad unmatching up: if the retrieved case is of a different class from the input case (bad), we increase the weightings of features with a negative fd (unmatching).

Following the example of (Bonzano, Cunningham, and Smyth 1997) We alter the weightings by adding/subtracting some number. The formula for increasing weighting is:

$$w_i(t+1) = w_i(t) + \delta * \frac{F_c}{K_c} \tag{2}$$

The formula for decreasing weighting is:

$$w_i(t+1) = w_i(t) - \delta * \frac{F_c}{K_c} \tag{3}$$

$w_i(t)$ is the weighting of the i-th feature 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.

Learning and Rewarding Differences

In ISAC, a negative fd score indicates difference, therefore the feature weightings are strictly positive. To understand this, imagine the weighting w_v for feature v is negative. If two cases are vastly different in feature v, then the fd score is negative. When multiplied with a negative weighting, it positively contributes to the activation sum. This is not desired since k-NN wants to only reward similarity.

Therefore, in the training process of ISAC, when updating policies produce a negative feature weighting, it is reset back to zero or a very small positive value.

To allow the learning and rewarding of difference patterns in C2C, a key implementation is to allow negative feature weightings in C_i-C_j, when $C_i \neq C_j$.

When $C_i \neq C_j$, a negative feature weighting multiplied with a negative feature difference score will positively influence the activation sum, thus rewarding difference for the corresponding feature.

Note that when $C_i = C_j$, negative feature weightings are still not allowed for the same reason as in ISAC.

Case Metric

In C2C weighting, cases of class C_1 actually has M sets of feature weightings, where M is the number of classes. These weightings are C_1-C_1, C_1-C_2, ... C_1-C_M. A weighting C_i-C_j indicates the pattern of similarities and differences from cases of C_i to cases of C_j. Lastly C_i-C_j are not necessarily symmetric, namely C_i-$C_j \neq C_j$-C_i.

For the weighting C_i-C_j, we call C_i the original class and C_j the projected class. If $C_i = C_j$, then C_i-C_j is called the native class weighting. If $C_i \neq C_j$, then C_i-C_j is called the non-native class weighting. These terms are useful in later sections.

Continuing from the Feature Metric section, the activation sum between a case A of class C_i and a case B of class C_j is calculated using:

$$w = C_i - C_j \tag{4}$$

$$fd(v_t, v_b) = 1 - 2\frac{|v_t - v_b|}{v_{max} - v_{min}} \tag{5}$$

$$activation(A, B, w) = \sum_v w_v * fd(v_A, v_B) \tag{6}$$

where w is a set of feature weightings, v is the index of a feature, w_v is the weighting for v in w, and v_A is the value of feature v in case A. The term $activation(A, B, w)$ reads as: the activation score between case A and case B under weighting w.

It might be a little counter-intuitive to use this case metric in testing. After all, given a new input case whose class is unknown, we won't be able to decide which weighting w to use. We tackle this by suggesting all possible classes for the input case, finding the weightings, and calculating the activation score for each possible class. The highest k activation scores win. Each of these activation scores is calculated by some w (or C_i-C_j), which casts a vote for the projected class C_j in w. This will be further detailed in the pseudo-code section.

Pseudo-code

We implemented C2C weighting as an extension for global weighting.

The pseudo-code for training is listed in Algorithm 1. It is different from the training of traditional weightings in several aspects:

- There are multiple weightings C_i-C_j
- Cases are divided into pools based on their classes, and the retrieval takes place in every pool. The motivation behind this design is that, even if we know that the training case A is of class C_j, we still want to retrieve cases from some other class C_i, so that we can update C_i-C_j to learn the pattern of similarities and differences from C_i to C_j.
- If $C_i = C_j$, then no negative value is allowed in the set of weightings C_i-C_j. Otherwise, negative values are allowed.

Following the example of ISAC, we normalize the weightings after every round of feature weighting updating, to prevent the risk that certain weightings become too large and overshadow all other feature weightings.

Algorithm 1 training algorithm

Require: $case\_base$, the cases to be learned.
initialize feature metric
store all possible classes in $all\_classes$
for each $C_i \in all\_classes$ **do**
  for each $C_j \in all\_classes$ **do**
    initialize C_i-C_j weighting
  end for
end for
initialize $pool$ to store cases in each class
for each $case \in case\_base$ **do**
  C_i = the class of $case$
  add $case$ to $pool_i$
end for
repeat
  for each $case\ A \in case\_base$ **do**
    C_j = the class of A
    for each $pool_i \in pool$ **do**
      retrieve k cases B_1 ... B_k from $pool_i$
      such that $activation(B_p, A, w)$ are the max
      w is of the form $(C_i$-$C_j)$
      for each $B_p \in B_1$ ... B_k **do**
        for each $f_q \in features$ **do**
          $oldW_q = (C_i$-$C_j)_q$
          $newW_q = oldW_q \pm \delta * \frac{F_{B_p}}{K_{B_p}}$
          if $C_i = C_j$ and $newW_q < 0$ **then**
            set $newW_q$ to 0.001
          end if
          set $(C_i$-$C_j)_q$ to $newW_q$
        end for
        Normalize $(C_i$-$C_j)$
      end for
    end for
  end for
until training converge / max # of iterations reached

The code for testing is listed in Algorithm 2. It is different from the testing of traditional weightings in that there is a class suggestion stage before the retrieval of stored cases.

Following the pseudo-code, the computational complexity for the training and testing of C2C weighting is M times the respective complexity of global weighting, where M is the number of classes.

Performance

In this section, we present the prediction accuracies of our modified k-NN algorithm and compare them with the accuracies of a k-NN algorithm using global weighting. We tested on three data sets: Iris, Cleveland, and Breast Cancer (Lichman 2013). We divided the cases into training and testing set by the ratio of 7:3. We tried setting k to 1 and 3. As the results are similar, we only show the results for when k is 1. For each setting, we shuffled the cases in every iteration in the training process and trained for 10 iterations. We ran

Algorithm 2 testing algorithm

Require: $test\_case\ A$
Initialize $score$
for each $C_j \in all\_classes$ **do**
  Suggest the class of A as C_j
  for each $case\ B \in case\_base$ **do**
    C_i = the class of B
    $w = C_i$-C_j
    $score_{B,A,C_i\text{-}C_j} = activation(B, A, w)$
  end for
end for
Get the highest k $score_{C_i\text{-}C_j}$
extract C_js from C_i-C_js
Return the major vote of C_js

	Iris	Cleveland	Breast Cancer
C2C	0.889	0.611	0.828
Global	0.938	0.762	0.947

Table 1: C2C vs Global weighting, no cases removed

training + testing for each k-NN algorithm 100 times and took the average of accuracies in testing.

We notice C2C weighting struggles with data sets with certain characteristics, but shines under some other circumstances. We present an analysis of the phenomena. Lastly we use a synthetic data set to confirm this observation.

The Bad

Although C2C weighting sounds enticing, our results in Table 1 show that C2C weighting combined with a traditional global weighting performs slightly worse than global weighting.

After closer examination of the feature weightings produced, we notice that when C2C weighting produces bad accuracies, it is normally because some native class weighting is heavily skewed. For example, an example of bad weighting C_1-C_1 for the iris data set is:

[0.001, 3.997, 0.001, 0.001]

Clearly this weighting heavily skews toward the second feature and ignores similarity in other features.

In the future iterations of training: if a new input case of C_1 retrieved a stored case of C_1 using this weighting, then it is highly likely that their second feature values match, while other feature values may not (due to the weighting only rewarding similarity in the second feature). Because of the "good matching up" policy, the second feature weighting will increase, while others may change to more or less. After normalizing, the second feature weighting would still dominate.

The only way to regulate such skewed weighting is to retrieve a stored case of C_1 when the input case is of some other class C_2. Because of the weightings, the two cases' second feature values match, thus triggering the "bad matching down" policy to drive down the second feature weighting. However, there remain two obstacles: 1) The retrieval is unlikely to happen because the input case of C_2 is more likely to retrieve a C_2 case, especially if the weighting for C_2-C_2 is not skewed. 2) Even if the retrieval does happen, other feature values have to be unmatched to trigger "bad unmatching up", thus increasing other feature weightings. However, there is no guarantee that this will happen.

In short, once a native class weighting is skewed in C2C weighting, our current design does not offer a way to restore it. Such skewed weighting is the direct reason for a lower accuracy.

	Iris	Cleveland	Breast Cancer
C2C	0.722	0.562	0.627
Global	0.612	0.452	0.624

Table 2: C2C vs Global weighting, a whole class of cases removed

The Good

Knowing what could go wrong, we manually check C2C weighting results of multiple runs. When the weightings are not skewed, the accuracy is comparable to global weighting.

In these runs, the native class weighting is correctly trained and produce good predictions. The scores obtained using native class weightings are generally higher than the scores obtained using non-native class weightings. As a result, the cases are almost always retrieved using the native class weightings. Here C2C weighting shows the same effect as a class-specific weighting (Marchiori 2013). This phenomenon is also partially due to completeness and cleanness of our data sets.

We then carried out an experiment where we removed a whole class of cases from the case base before the testing process. Even though in the real world no one would simply drop a whole class of cases, but data compression and case base maintenance are practical issues. Our experiment is crude but meaningful. As we disrupted the completeness of our case base. we rendered certain native class weightings useless, and forced the CBR system to utilize non-native class weightings. We deterministically chose the class of the first stored case, and removed all cases of that class from the case base. We then ran the same experiment as the last section for both C2C weighting and global weighting. The prediction accuracies are listed in Table 2.

From this experiment we see that in certain situations C2C excels traditional weighting. When the whole class C_1 of cases is dropped, global weighting would simply fail for any input case of C_1. For C2C weighting, even though the native class weighting C_1-C_1 can no longer be used, non-negative class weightings such as C_2-C_1 can still work with cases of class C_2 and project an input case as C_1.

In the Iris data set and the Cleveland data set, cases of different classes are nicely separated from each other. If the difference between C_2 and C_1 follows a pattern, then C_2-C_1 can be distinctively established. Cases in C_2 will be able to make a good suggestion about whether an input case is of C_1.

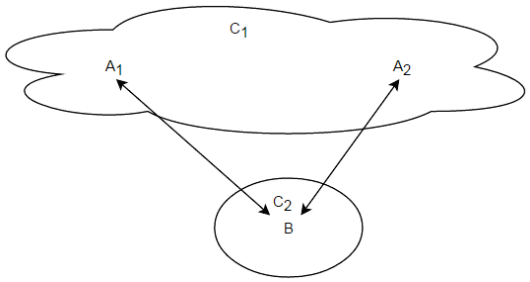

Figure 1: Scattered data set leads to unclear difference pattern

In the Breast Cancer data set, cases of different classes are widely scattered and mixed with each other. A simplified example of such data sets is shown in Figure 1. In this figure, because the cases of C_1 are spread out, the difference pattern between A_1 and B is not the same as that between A_2 and B. Therefore, non-native class weightings are hard to learn, or there might not exist one. As shown by our result for the Breast Cancer data set, C2C weighting performs about the same as global weighting. In such scenarios, feature selection and feature transformation might be needed to establish the pattern of differences in features between classes.

The Synthetic Data Experiment

To further illustrate the point above and demonstrate when C2C weighting can be of great interest, we created three data sets of points on a 2-dimensional plane. Every point belongs to one of four classes. Each class of points is generated using a Gaussian distribution.

There are two parameters to generate the four groups: μ and σ. The four Gaussian distributions are: $\mathcal{N}((\mu, \mu), \sigma^2)$, $\mathcal{N}((\mu, -\mu), \sigma^2)$, $\mathcal{N}((-\mu, \mu), \sigma^2)$, $\mathcal{N}((-\mu, -\mu), \sigma^2)$. We used $\sigma = 2$ and various $\mu = 2, 4, 8$ to generate the three data sets.

With these data sets, we ran both C2C weighting and global weighting on it for 100 times to get their average accuracies. We also re-ran the same experiment after removing one class of cases and two classes of cases from the data sets. The results are shown in Table 3, 4, and 5.

The results confirm our earlier analysis. When μ is small, different classes are mixed together and it is hard to learn a distinguished pattern of differences between two classes. Because the native class weightings can be skewed and the non-native class weightings are hard to learn, in these situations, C2C weighting performs worse than global weighting.

As μ becomes larger, C2C weighting is less skewed in native class weightings, and its accuracy gradually approaches that of its counterpart. When μ is large, the classes are well separated and C2C weighting easily learns the correct pattern of differences. In these situations, C2C weighting performs great. Even when we drop one or multiple classes, the prediction accuracy remains almost intact (0.946, 0.945, 0.941). This is because non-native class weightings are making good suggestions when the native class weightings cannot.

	$\mu = 2$	$\mu = 4$	$\mu = 8$
C2C	0.575	0.881	0.946
Global	0.588	0.897	1.0

Table 3: C2C vs Global weighting, no cases removed

	$\mu = 2$	$\mu = 4$	$\mu = 8$
C2C	0.459	0.693	0.945
Global	0.483	0.701	0.753

Table 4: C2C vs Global weighting, 1 class of cases removed

	$\mu = 2$	$\mu = 4$	$\mu = 8$
C2C	0.443	0.495	0.941
Global	0.452	0.503	0.498

Table 5: C2C vs Global weighting, 2 classes of cases removed

Discussion

By implementing and testing C2C on different data sets, we learned that even though the accuracy of C2C combined with global weighting actually degenerates the accuracy of global weighting, C2C weighting can achieve certain advantages over global weighting.

When no case is removed, global weighting functions better than C2C weighting. We argue that this is not an intrinsic flaw in C2C weighting. Instead, it is a problem in our integration of C2C weighting with global weighting. We almost directly reuse the training process of global weighting on C2C weighting, and native class weightings can be skewed by the training process. Non-native class weightings can offer useful insight, especially when native class weightings are not confident (low activation score) or cannot be used (whole class of cases removed). In our experiment, non-native class weighting can still make good predictions of a class, even if the case base does not contain a single case of that class.

Another unshown advantage of C2C weighting is that, it opens a brand-new avenue of explainability. When traditional k-NN suggests an input case is of class C_1, the explanation is always that the input case is similar to some existing case of C_1. Using C2C weighting, k-NN can make a different kind of suggestion that the input case is of C_1, because it is different from C_i cases in a way that existing C_1 cases are different from C_i cases.

Future Work

Better Integration of C2C with Existing Weighting Methods

As discussed, we should not directly reuse the training procedure of global weighting on C2C weighting. One possible way to solve this is to train non-native and native class weightings separately, possibly even using different procedures.

In the testing process, non-native and native class weighting can also be used simultaneously, suggesting the class of a case both from within the class and outside the class. When multiple non-native class weightings are taken into account, the votes from different non-native class suggestions can triangulate the projected class.

Integration of C2C with Instance-specific Weighting

Instance-specific weighting (Aha and Goldstone 1992) is a weighting method naturally fit for C2C extension. Compared to global weighting, Instance-specific weighting is less likely to be skewed, because it focuses on the local landscape and the weighting updates do not influence globally.

If combined with instance-specific weighting, C2C will be able to learn the pattern of differences between classes in a local region, even if cases of different classes are scattered and mixed globally. To reuse the example of Figure 1, C2C weighting will learn the a C_1-C_2 weighting for cases near A_1 and another C_1-C_2 weighting for cases near A_2. The two weightings are useful in their own local regions and do not interfere each other.

Therefore, we envision that the integration of C2C with instance-specific weighting naturally evades the issues we encountered when integrating C2C with global weighting.

Summary

After studying existing weighting methods used in k-NN, we invented class-to-class weighting. C2C weighting is a weighting method that learns the pattern of similarities and differences between classes. By using this pattern, k-NN can make predictions of an input case in a novel way. We carried out experiments and compared the performance of C2C weighting with global weighting. We acknowledged the flaw in current C2C weighting implementation but also found situations when C2C weighting is beneficial. We will push further the study by better integrating C2C weighting with existing weighting methods and by presenting some practical usage of C2C weighting.

Acknowledgement

We thank Dr. David Leake for the continuous support and guidance. We thank the insightful comments from anonymous reviewers. We did our best to address these comments. Some comments also point out interesting future directions, including: experimenting with challenging data sets such as heterogeneous data sets; and improving the weighting updating policies by formulating the problem with a cost function and carrying out standard optimization.

References

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

Aha, D.; Kibler, D.; and Albert, M. 1991. Instance-based learning algorithms. *Machine Learning* 6(1):37–66.

Bonzano, A.; Cunningham, P.; and Meckiff, C. 1996. *ISAC: A CBR system for decision support in air traffic control*. Berlin, Heidelberg: Springer Berlin Heidelberg. 44–57.

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.

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

Fix, E., and Hodges, J. L. 1951. Discriminatory analysis, nonparametric discrimination: Consistency properties. *US Air Force School of Aviation Medicine* Technical Report 4(3):477+.

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

Lichman, M. 2013. UCI machine learning repository.

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

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

Richter, M. M. 1993. Classification and learning of similarity measures. In Opitz, O.; Lausen, B.; and Klar, R., eds., *Information and Classification*, 323–334. Berlin, Heidelberg: Springer Berlin Heidelberg.

Weinberger, K. Q.; Blitzer, J.; and Saul, L. K. 2006. Distance metric learning for large margin nearest neighbor classification. In Weiss, Y.; Schölkopf, B.; and Platt, J. C., eds., *Advances in Neural Information Processing Systems 18*. MIT Press. 1473–1480.

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.