IKNN: Informative K-Nearest Neighbor Pattern Classification

Yang Song[1], Jian Huang[2], Ding Zhou[1], Hongyuan Zha[1,2], and C. Lee Giles[1,2]

[1] Department of Computer Science and Engineering
[2] College of Information Sciences and Technology,
The Pennsylvania State University,
University Park, PA 16802, U.S.A.

Abstract. The K-nearest neighbor (KNN) decision rule has been a ubiquitous classification tool with good scalability. Past experience has shown that the optimal choice of K depends upon the data, making it laborious to tune the parameter for different applications. We introduce a new metric that measures the informativeness of objects to be classified. When applied as a query-based distance metric to measure the closeness between objects, two novel KNN procedures, Locally Informative-KNN (LI-KNN) and Globally Informative-KNN (GI-KNN), are proposed. By selecting a subset of most informative objects from neighborhoods, our methods exhibit stability to the change of input parameters, number of neighbors(K) and informative points (I). Experiments on UCI benchmark data and diverse real-world data sets indicate that our approaches are application-independent and can generally outperform several popular KNN extensions, as well as SVM and Boosting methods.

1 Introduction

The K-nearest neighbor (KNN) classifier has been both a workhorse and benchmark classifier [1,2,4,11,14]. Given a query vector x_0 and a set of N labeled instances $\{x_i, y_i\}_1^N$, the task of the classifier is to predict the class label of x_0 on the predefined P classes. The K-nearest neighbor (KNN) classification algorithm tries to find the K nearest neighbors of x_0 and uses a majority vote to determine the class label of x_0. Without prior knowledge, the KNN classifier usually applies Euclidean distances as the distance metric. However, this simple and easy-to-implement method can still yield competitive results even compared to the most sophisticated machine learning methods.

The performance of a KNN classifier is primarily determined by the choice of K as well as the distance metric applied [10]. However, it has been shown in [6] that when the points are not uniformly distributed, predetermining the value of K becomes difficult. Generally, larger values of K are more immune to the noise presented, and make boundaries more smooth between classes. As a result, choosing the same (optimal) K becomes almost impossible for different applications.

Since it is well known that by effectively using prior knowledge such as the distribution of the data and feature selection, KNN classifiers can significantly

J.N. Kok et al. (Eds.): PKDD 2007, LNAI 4702, pp. 248–264, 2007.

improve their performance, researchers have attempted to propose new approaches to augmenting the performance of KNN method. e.g., Discriminant Adaptive NN [9] (DANN), Adaptive Metric NN [6] (ADAMENN), Weight Adjusted KNN [8] (WAKNN), Large Margin NN [13] (LMNN) and etc. Despite the success and rationale of these methods, most have several constraints in practice. Such as the effort to tune numerous parameters (DANN introduces two new parameters, K_M and ϵ; ADAMENN has six input parameters in total that could potentially cause overfitting), the required knowledge in other research fields (LMNN applies semidefinite programming for the optimization problem), the dependency on specific applications (WAKNN is designed specifically for text categorization) and so on. Additionally, in spite of all the aforementioned constraints, choosing the proper value of K is still a crucial task for most KNN extensions, making the problem further compounded.

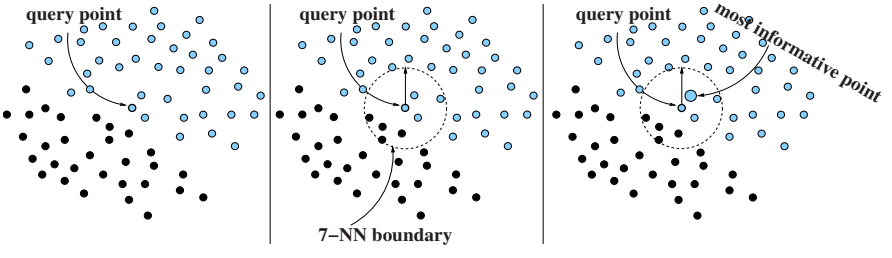


Fig. 1. A toy classification problem. (Left) The original distribution of two classes. (Middle) Results of $k = 7$ NN method where the query point is misclassified. (Right) One of our proposed methods, (LI-KNN), chooses one informative point for prediction.

Therefore, it is desirable to augment the performance of traditional KNN without introducing much overhead to this simple method. We propose two KNN methods that are ubiquitous and their performances are insensitive to the change of input parameters. Figure 1 gives an example that shows the motivation of our approach, in which the traditional KNN method fails to predict the class label of the query point with $K = 7$. One of our proposed method (LI-KNN) takes the same value of K, finds the most *informative* point ($I = 1$) for the query point according to the new distance metric, and makes a correct prediction.

1.1 Our Contribution

In this paper, we propose two novel, effective yet easy-to-implement extensions of KNN method whose performances are relatively insensitive to the change of parameters. Both of our methods are inspired by the idea of *informativeness*. Generally, a point(object) is treated to be *informative* if it is *close to the query point and far away from the points with different class labels*. Specifically, our paper makes the following contributions:

(1) We introduce a new concept named informativeness to measure the importance of points, which can be used as a distance metric for classification. (2)

Based on the new distance metric, we propose an efficient *locally informative* KNN (LI-KNN) method. (3) By learning a weight vector from training data, we propose our second method that finds the *globally informative* points for KNN classification (GI-KNN). (4) We perform a series of experiments on real world different data sets by comparing with several popular classifiers including KNN, DANN, LMNN, SVM and Boosting. (5) We discuss the optimal choice of the input parameters (K and I) for LI-KNN and GI-KNN and demonstrate that our methods are relatively insensitive to the change of parameters.

The rest of the paper is organized as follows: Section 2 presents related work about different approaches to improve KNN pattern classification; section 3 introduces the definition of informativeness and our first algorithm LI-KNN; section 4 continues to propose the second learning method GI-KNN; we apply the proposed methods to both synthetic and real-world data sets in section 5 for evaluation; finally we conclude in section 6.

2 Related Work

The idea of nearest neighbor pattern classification was first introduced by Cover and Hart in [4], in which the decision rule is to assign an unclassified sample point to the classification of the nearest of a collection of predetermined classified points. The authors proved that when the amount of data approaches infinity, the one nearest neighbor classification is bounded by twice the asymptotic error rate as the Bayes rule, independent of the distance metric applied.

Hastie and Tibshirani [9] developed an adaptive method of nearest neighbor classification (DANN) by using local discrimination information to estimate a subspace for global dimension reduction. They estimate between (B) and within (W) the sum-of-squares matrices, and use them as a local metric such as $\sum = W^{-1}BW^{-1}$. They showed that their work can be generalized by applying specialized distance measures for different problems.

Weinberger et al. [13] learned a Mahanalobis distance metric for KNN classification by using semidefinite programming, a method they call large margin nearest neighbor (LMNN) classification. Their method seeks a large margin that separates examples from different classes, while keeping a close distance between nearest neighbors that have the same class labels. The method is novel in the sense that LMNN does not try to minimize the distance between all examples that share the same labels, but only to those that are specified as *target neighbors*. Experimental results exhibit great improvement over KNN and SVM.

By learning locally relevant features from nearest neighbors, Friedman [7] introduced a flexible metric that performs recursively partitioning to learn local relevances, which is defined as $I_i^2(z) = (Ef - E[f|x_i = z])^2$, where Ef denotes the expected value over the joint probability density $p(x)$ of an arbitrary function $f(x)$. The most informative feature is recognized as the one giving the largest deviation from $P(x|x_i = z)$.

Han et al. [8] proposed an application of KNN classification to text categorization by using adjusted weight of neighbors (WAKNN). WAKNN tries to

learn the best weight for vectors by measuring the cosine similarity between documents. Specifically, the similarity function is defined as $cos(X, Y, W) = \frac{\sum_{t \in T}(X_t \times W_t) \times (Y_t \times W_t)}{\sqrt{\sum_{t \in T}(X_t \times W_t)^2} \times \sqrt{\sum_{t \in T}(Y_t \times W_t)^2}}$, where X and Y are two documents, W the weight vector and T the set of features (terms). Optimizations are also performed to speed up WAKNN. The experiments on benchmark data sets indicate that WAKNN consistently outperforms KNN, C4.5 and several other classifiers.

3 Locally Informative KNN (LI-KNN)

Without prior knowledge, most KNN classifiers apply Euclidean distances as the measurement of the *closeness* between examples. Since it has already been shown that treating the neighbors that are of low relevance as the same importance as those of high relevance could possibly degrade the performance of KNN procedures [7], we believe it to be beneficial to further explore the information exhibited by neighbors. In this section, we first propose a new distance metric that assesses the informativeness of points given a specific query point. We then proceed to use it to augment KNN classification and advocate our first method, LI-KNN.

3.1 Definition of Informativeness

We use the following naming conventions. Q denotes the query point, K indicates the K nearest neighbors according to a distance metric, and I denotes most informative points based on equation (1). For each point, x_i denotes the i's feature vector, x_{ij} its j's feature and y_i its class label. Let N represent the total number of training points, where each point has P features.

Definition 1. *Specify a set of training points* $\{x_i, y_i\}_1^N$ *with* $x_i \in \mathbb{R}^P$ *and* $y_i \in \{1, ...m\}$. *For each query point* x_i, *the* ***informativeness*** *of each of the remaining N-1 points* $\{x_j, y_j\}_1^N$ *is defined as:*

$$\mathcal{I}(x_j|Q{=}x_i) = -\log(1 - \mathcal{P}(x_j|Q = x_i)) * \mathcal{P}(x_j|Q = x_i),\ j = 1, ...N, j \neq i \quad (1)$$

where $\mathcal{P}(x_j|Q = x_i)$ *is the probability that point* x_j *is informative (w.r.t. Q), defined as:*

$$\mathcal{P}(x_j|Q{=}x_i){=}\frac{1}{Z_i}\left\{\mathtt{Pr}(x_j|Q{=}x_i)^{\eta}\left(\prod_{n=1}^{N}\left(1 - \mathtt{Pr}(x_j|Q{=}x_n)\mathbb{I}_{[y_j \neq y_n]}\right)\right)^{1-\eta}\right\} \quad (2)$$

The first term $\mathtt{Pr}(x_j|Q = x_i)^{\eta}$ in equation (2) can be interpreted as the likelihood that point x_j is close to the Q, while the second part indicates the probability that x_j far apart from dissimilar points. The indicator $\mathbb{I}[.]$ equals to 1 if the condition is met and 0 otherwise. Z_i is a normalization factor and η is introduced as a balancing factor that determines the emphasis of the first term. Intuitively, η is set to $\frac{N_{x_j}}{N}$, where N_{x_j} represents the number of points in the same class of x_j.

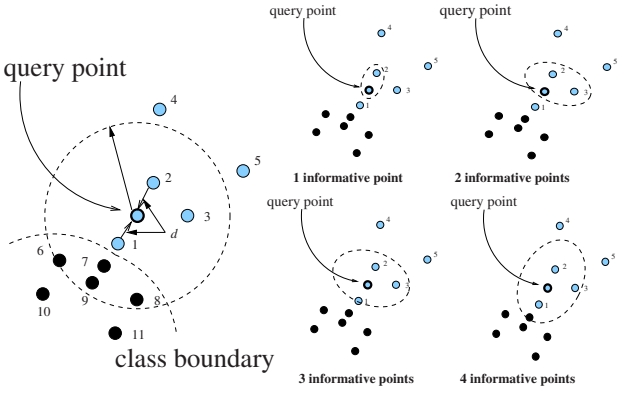


Fig. 2. An illustration of 7-NN and the corresponding i informative points for the query point. (Left) 7-NN classification and the real class boundary. (Right) $i(i = \{1, 2, 3, 4\})$ informative points for the same query.

The rationale of informativeness is that two points are likely to share the same class label when their distance is sufficiently small, assuming the points have a uniform distribution. This idea is the same as KNN classification. On the other hand, compared to traditional KNN classifiers that measures pairwise distances between the query point and neighbors, our metric also calculates the closeness between neighbor points, i.e., the informative points should also have a large distance from dissimilar points. This further guarantees that the locations of other informative points have the same class label maximum likelihood.

Figure 2(left) gives an example for clarification, in which point 1 and point 2 (with the same class label) both have the same distance d from Q but point 1 is closer to the real class boundary. Thus, point 1 is more likely to be closer to the points in other classes. As such we claim that point 1 is less informative than point 2 for Q by DEFINITION 1. Again, assuming the distribution over the concept location is uniform, it is more likely that points (e.g., 3 & 4) have the same label as points 1 & 2 and will more likely distribute around point 2 than point 1.

3.2 Informativeness Implementation

To define $\texttt{Pr}(x_j|Q = x_i)$ in equation (2), we can model the causal probability of an individual point on Q as a function of the distance between them:

$$\texttt{Pr}(x_j|Q = x_i) = f(\|x_i - x_j\|_p) \tag{3}$$

where $\|x_i - x_j\|_p$ denotes the p-norm distance between x_i and x_j. To achieve higher probability when two points are close to each other, we require $f(.)$ to be a function inverse to the distance between two points. The generalized Euclidean distance metric satisfies this requirement. Thus, equation (3) can be defined as

$$\mathtt{Pr}(x_j|Q=x_i)=\exp(-\frac{||x_i-x_j||^2}{\gamma}) \qquad \gamma>0 \tag{4}$$

In practice, it is very likely that the features have different importance, making it desirable to find the best weighting of the features. Specifically, we define $||x_i-x_j||^2=\sum_p w_p(x_{ip}-x_{jp})^2$, where w_p is a scaling that reflects the relative importance of feature p. Although there exists numerous functions for calculating w_p, here we specify it as follows:

$$w_p=\frac{1}{m}\sum_{k=1}^{m}w_{pk}=\frac{1}{m}\sum_{k=1}^{m}\mathtt{Var}_{\mathbf{x}_k}(\mathbf{x}_{pk}) \tag{5}$$

We obtain w_p by averaging over all classes' weights w_{pk}, which is calculated using the variance of all points in each class k at feature p, denoted by $\mathtt{Var}_{\mathbf{x}_k}(\mathbf{x}_{pk})$.

The normalization factor Z_i in equation (2) ensures the well-defined probabilities of neighbors for a given query point x_i. Specifically,

$$Z_i=\sum_{j=1}^{N}\mathtt{Pr}(x_j|Q=x_i), \qquad \sum_{j=1}^{N}\mathcal{P}(x_j|Q=x_i)=1 \tag{6}$$

so that the normalization is guaranteed.

Based on the implementation, we have the following proposition:

Proposition 1. *Given a specific query x_0, $\forall\, x_i, x_j$ that satisfies $\|x_i-x_0\|^2=kd$ and $\|x_j-x_0\|^2=d$ with $d\in\mathbb{R}^+, k>1$, $\mathcal{I}(x_i|x_0)<\exp((1-k)d)^\eta\mathcal{I}(x_j|x_0)$.*

Proof. For simplicity, we only consider the case that x_i and x_j are in the same class, i.e., $y_i=y_j$. Without loss of generality, we let $\gamma=1$ for equation (4). We have

$$\begin{aligned}\frac{\mathcal{P}(x_j|Q=x_0)}{\mathcal{P}(x_i|Q=x_0)}&=\frac{\mathtt{Pr}(x_j|Q=x_0)^\eta H(x_j)^{1-\eta}}{\mathtt{Pr}(x_i|Q=x_0)^\eta H(x_i)^{1-\eta}}\\&=\frac{\exp(-d)^\eta H(x_j)^{1-\eta}}{\exp(-kd)^\eta H(x_i)^{1-\eta}}\\&=\exp((k-1)d)^\eta\frac{H(x_j)^{1-\eta}}{H(x_i)^{1-\eta}}\end{aligned} \tag{7}$$

where $H(\mathtt{x})=\left(\prod_{n=1}^{N}\left(1-\mathtt{Pr}(\mathtt{x}|Q=x_n)\mathbb{I}_{[\mathtt{y}\neq y_n]}\right)\right)$. Since $H(\cdot)$ is independent of the query point, its expected value (taken over $\mathtt{x}$ and each x_n) can be defined as

$$\begin{aligned}E(H(\mathtt{x}))&=E\left(\prod_{n=1}^{N}\left(1-\mathtt{Pr}(\mathtt{x}|Q=x_n)\mathbb{I}_{[\mathtt{y}\neq y_n]}\right)\right)\\&=\prod_{n=1}^{N}\left(E\left(1-\mathtt{Pr}(\mathtt{x}|Q=x_n)\mathbb{I}_{[\mathtt{y}\neq y_n]}\right)\right)\end{aligned}$$

$$= \prod_{n=1}^{N} \left(E(1 - \exp(-\|\mathtt{x} - x_n\|^2)\mathbb{I}_{[\mathtt{y} \neq y_n]}) \right)$$
$$= \prod_{n=1}^{N} \left((1 - E\exp(-\|\mathtt{x} - x_n\|^2)\mathbb{I}_{[\mathtt{y} \neq y_n]}) \right)$$

Recall that x_i and x_j are in the same class, thus the set of dissimilar points (say $\{x'_n, y'_n\}_1^q$) should be the same. The above equation can then be simplified by removing the indicator variables:

$$E(H(\mathtt{x})) = \prod_{n=1}^{q} \left((1 - E\exp(-\|\mathtt{x} - x'_n\|^2)) \right)$$
$$= \prod_{n=1}^{q} \left(1 - \int_1^N \exp(-\|\mathtt{x} - x'_n\|^2) d\mathtt{x} \right)$$

with $N \to \infty$, it is easy to verify that $E(H(x_i)) = E(H(x_j))$. Applying the results to equation (7), we have

$$\frac{\mathcal{P}(x_j|Q = x_0)}{\mathcal{P}(x_i|Q = x_0)} = \exp((k-1)d)^\eta > 1 \quad (\mathtt{with}\ k > 1) \tag{8}$$

Applying equation (8) to equation (1), we finally have:

$$\frac{\mathcal{I}(x_j|Q = x_0)}{\mathcal{I}(x_i|Q = x_0)} = \frac{\log(1 - \mathcal{P}(x_j|Q = x_0))}{\log(1 - \mathcal{P}(x_i|Q = x_0))} \cdot \exp((k-1)d)^\eta$$
$$= \log_{(1-\mathcal{P}(x_i|Q=x_0))}(1 - \mathcal{P}(x_j|Q = x_0)) \cdot \exp((k-1)d)^\eta$$
$$> \exp((k-1)d)^\eta \qquad \square$$

3.3 LI-KNN Classification

So far we have proposed to compute the informativeness of points in the entire data distribution for a specific Q. However, considering the high dimensionality and large number of data points in practice, the computational cost could be prohibitively high. We propose to make use of the new distance metric defined in equation (1) by restricting the computation between the nearest neighbors in an augmented *query-based* KNN classifier.

Algorithm 1 gives the pseudo-code of LI-KNN classification. Instead of finding the informative points for each x_i by going over the entire data set, LI-KNN retrieves I *locally* informative points by first getting the K nearest neighbors (we consider the Euclidean distance here). It then applies equation (1) to the K local points and the majority label between the I points are assigned to x_i. Specifically, when $I = 1$, LI-KNN finds only the most informative point, i.e., $y_i = \arg\max_{y_k, k \in \{1,\ldots,K\}} \mathcal{I}(x_k|Q = x_i)$. In this way the computational cost of finding the most informative points is reduced to a local computation. Noticeably, when

Algorithm 1. LI-KNN Classification

Input: (S, K, I)
target matrix: $S = \{x_i, y_i\}_1^N$
number of neighbors: $K \in \{1, ..., N-1\}$
number of informative points: $I \in \{1, ..., K\}$
Initialize $err \leftarrow 0$
for each query point x_i ($i = 1$ to N) **do**
  find K nearest neighbors $\mathcal{X}_K$ using Euclidean distance
  find I most informative points among K neighbors (equation (1))
  majority vote between the I points to determine the class label of x_i
  if x_i is misclassified
    $err \leftarrow err + 1/N$
  end if
end for
Output: err

K equals to N, the locally informative points are exactly the optimal informative points for the entire data distribution as in DEFINITION 1. Likewise, when I equals to K, LI-KNN performs exactly the same as KNN rule.

At the first glance, it seems that LI-KNN introduces one more parameter I for the KNN method. However, by carefully checking the requirement for points to be informative, it is not hard to figure out that LI-KNN is relatively insensitive to both K and I. (1) Regardless of the choice of K, the points that are closest (in Euclidean distance) to Q are always selected as neighbors, which by equation (2) have a high probability to be informative. (2) On the other hand, given a fixed number of K, the informativeness of the local points are fixed which insures that the most informative ones are always chosen. For example, in Figure 2(left), point 2 & 3 are selected as the neighbors for Q with K increasing from 3 to 7. Meanwhile, when K equals to 7 and I ranges from 1 to 3, the informative sets (Figure 2(right)) are $\{2\}$,$\{2, 3\}$ and $\{2, 3, 1\}$ respectively, which include the most informative points in all cases that ensures Q is classified correctly. In practice, cross-validation is usually used to determine the best value of K and I.

4 GI-KNN Classification

The LI-KNN algorithm classifies each individual query point by learning informative points separately, however, the informativeness of those neighbors are then discarded without being utilized for other query points. Indeed, in most scenarios, different queries Q may yield different informative points. However, it is reasonable to expect that some points are more informative than others, i.e., they could be informative neighbors for several different points. As a result, it would seem reasonable to put more emphasis on those points that are *globally* informative. Since it has been shown that KNN classification [13] can be improved by learning from training examples a distance metric, in this section we enhance the power of the informativeness metric and propose a boosting-like

iterative method, namely a *globally informative* KNN (GI-KNN) that aims to learn the best weighting for points within the entire training set.

4.1 Algorithm and Analysis

The goal of GI-KNN is to obtain an optimum weight vector A from all training points. The algorithm iterates M predefined steps to get the weight vector, which was initially set to be uniform. In each iteration, an individual point is classified in the same way as LI-KNN by finding I informative neighbors, with the only exception that in GI-KNN the distance metric is a weighted Euclidean distance whose weight is determined by A (line 5 & 6 in Algorithm 2, where $D(x_i, \mathbf{x})$ denotes the Euclidean distance between x_i and all the remaining training points, and $D_A(x_i, \mathbf{x})$ is the weighted distance). We use $\epsilon_m^i \in (0, 1)$ to denote the *weighted* expected weight loss of x_i's informative neighbors during step m. The cost function C_m^i is a smooth function of ϵ_m^i, which guarantees it to be in the range of (0,1) and positively related with ϵ_m^i. Here we use tanh function as the cost function, depicted in Figure 3[1]. The weight vector A is updated in the manner that if x_i is classified incorrectly, the weights of its informative neighbors which have different labels from x_i are decreased exponentially to the value of C_m^i (line 9, $e(x_i, x_\ell) = C_m^i$ `if` $y_i \neq y_\ell$; line 13, $A(x_\ell) \leftarrow A(x_\ell) \cdot \exp(-e(x_i, x_\ell))$). Meanwhile, the weights remain the same for neighbors in the same class with x_i even if x_i is misclassified (line 9, $e(x_i, x_\ell) = 0$ `if` $y_i = y_\ell$). Clearly, the greater the weight the query point is, the higher the penalty of misclassification will be for the selected neighbors. The vector A is then normalized before the next iteration.

Instead of rewarding those points that classify Q correctly by increasing their weights, the weights of neighbors remain unchanged if Q is classified correctly. This could potentially cause accumulative effects to points whose weights that once increased will always increase in the following steps, ending up with dominant large weights. As a result, we penalize those points that give the wrong prediction and have different labels with Q. Therefore, by updating the weight vector before the next iteration, they will be less likely to be selected as the neighbors for the same Q.

While GI-KNN has several parallels to Boosting such as the structure of the algorithm, GI-KNN differs from Boosting in the way weights are updated. Specifically, Boosting assigns high weights to points that are misclassified in the current step, so that the weak learner can attempt to fix the errors in future iterations. In GI-KNN classification, the objective is to find globally informative points, thus higher weights are given to the neighbors that seldom makes wrong predictions. Notice that the weight of the query point remains unchanged at that time, because the weight is updated for a specific point if and only if it is chosen to be one of the informative points for Q.

Another difference from Boosting is that the objective of the Boosting training process is to find a committee of discriminant classifiers that combines the

[1] In practice, we did not find much difference in performance for different τ. Therefore, we choose $\tau = 1$ for our implementation.

Algorithm 2. GI-KNN Training

Input: (T, K, I, M)
training set: $T = \{\mathbf{x}, \mathbf{y}\} \in \mathbb{R}^{N \times P}$
number of neighbors: $K \in \{1, ..., N-1\}$
number of informative points: $I \in \{1, ..., K\}$
number of iterations: $M \in \mathbb{R}$
Initialization: $A = \{1, ..., 1\} \in \mathbb{R}^{N \times 1}$ [*the weight vector*]
for $m = 1$ to M **do**
  for each query point x_i ($i = 1$ to N) **do**
    $D_A(x_i, \mathbf{x}) = \frac{D(x_i, \mathbf{x})}{A}$ [*calculate the weighted distance*]
    $\mathcal{N}_m(x_i) \leftarrow I$ most informative points according to $D_A(x_i, \mathbf{x})$
    $\epsilon_m^i = A(x_i) \cdot E_A[\mathcal{N}_m(x_i)] = A(x_i) \cdot \frac{1}{I} \sum_{i=1}^{I} A(\mathcal{N}_m(i))$
    $C_m^i = \frac{1}{2}(1 + tanh(\tau * (\epsilon_m^i - \frac{1}{2})))$

$$e(x_i, x_\ell) = \begin{cases} C_m^i & \text{if } y_i \neq y_\ell; \\ 0 & \text{if } y_i = y_\ell. \end{cases}$$

    if point x_i is classified incorrectly [*update the neighbors' weights*]
      $err_m \leftarrow err_m + \frac{1}{N}$
      for each x_ℓ ($\ell \in \mathcal{N}_m(x_i)$) **do**
        $A(x_\ell) \leftarrow A(x_\ell) \cdot \exp(-e(x_i, x_\ell))$
      end for
    renormalizes A so that $\sum_{i=1}^{N} A(i) = N$
  end for
  $\xi_m \leftarrow err_m - err_{m-1}$
end for
Output: the weight vector A

weak learners, while GI-KNN tries to learn a query-based distance metric by focusing on finding the best weight vector for each training instance so that the misclassification rate of training examples could be minimized.

4.2 Learning the Weight Vector

At completion, the learned vector A can be used along with L_2 distance metric for KNN classification at each testing point $\mathbf{t_0}$. Specifically, given the training set $T = \{x_i, y_i\}_1^N$, the distance between $\mathbf{t_0}$ and each training point x_i is defined as

$$D(\mathbf{t_0}, x_i) = \|\mathbf{t_0} - x_i\|_{A_i} = \frac{\sqrt{(\mathbf{t_0} - x_i)^T (\mathbf{t_0} - x_i)}}{A_i} \tag{9}$$

By adding weights to data points, GI-KNN in essence is similar in effect to learning a Mahalanobis distance metric $D(x_i, x_j)$ for k-nearest neighbor classification. i.e., $D(x_i, x_j) = D_{\mathcal{A}}(x_i, x_j) = \|x_i - x_j\|_{\mathcal{A}} = \sqrt{(x_i - x_j)^T \mathcal{A}(x_i - x_j)}$, where $\mathcal{A}$ determines the similarity between features. In our case, A measures the importance of each training point rather than their features.

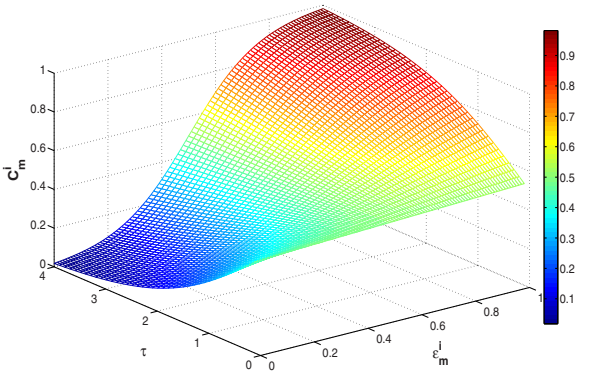


Fig. 3. Cost function (C) used for GI-KNN

In practice, we make two modifications to Algorithm 2 to reduce the computational cost. First, the L_2 distances between all training points are computed and stored (say in matrix D) at the beginning of the program. Then, instead of updating D real-time for each query point (line 5 in Algorithm 2), we do it after each external iteration. In another words, for each point, we update the weight vector if necessary, but use the same A for all points in the same iteration. After each round, D is updated with the new weight vector. Similarly, rather than normalizing A after each classification of Q (line 15 in Algorithm 2), the normalization is performed only after each external iteration. We discover that empirically these two modifications do not degrade the performance in most scenarios.

4.3 Complexity of GI-KNN Training

The major overhead of GI-KNN training phase is the time needed to find the informative neighbors for each point (line 5 of Algorithm 2). Specifically, $\mathcal{N}_m(i)$ keeps the indices of informative neighbors for point x_i, whose length is controlled by the input parameter I. Given K and I, the effort to find the nearest neighbors is bounded by $O(KP)$, where P denotes dimension of the input data. Calculating and ranking the informativeness of K nearest neighbors involves computing the pairwise distances between them and thus costs $O(K^2P)$ time to train. Thus the total time is bounded by $O(KP)+O(K^2P) = O(K^2P)$ for each point. Therefore, the training process requires approximately $O(K^2PMN)$ time for N training examples and M iterations. Remember that the traditional KNN classification costs $O(KPN)$ for the same setting, while LI-KNN requires $O(K^2PN)$. When the K is not very large, the computational complexity is nearly the same for KNN and LI-KNN, both of which are equal to one iteration time for GI-KNN.

5 Experiments

In this section, we present experimental results with benchmark and real-world data that demonstrate the different merits of LI-KNN and GI-KNN. LI-KNN

and GI-KNN are first rigorously tested by several standard UCI data sets. Then our proposed methods are applied to text categorization using the real-world data from CiteSeer Digital Library[2]. Finally, we investigate their performance on images by applying to image categorization on the COIL-20[3] bench-marked data sets.

For performance evaluation, several classifiers are used for comparison. The classic KNN [4] classifier is used as the baseline algorithm. We implemented DANN [9] as an extension of KNN[4]. To be more convincing, we also compare with one of the newest KNN extensions – Large Margin Nearest Neighbor Classification (LMNN)[5]. Two discriminant classifiers are also compared: a Support Vector Machine (SVM) and a Boosting classifier. We use the AdaBoost.MH [12] and the Multi-class SVM [5] software (K.Crammer et al.[6]) for multi-class classification.

5.1 UCI Benchmark Corpus

We evaluate our algorithms by using 10 representative data sets from UCI Machine Learning Repository[7]. The size of the data sets ranges from 150 to 20,000 with dimensionality between 4 and 649, including both two classes and multi-class data ($C = 3, 26$ etc). For evaluation, the data sets are split into training sets and testing sets with a fixed proportion of 4:1. Table 1 reports the best testing error rates for these methods, averaged over ten runs. Our methods on these data sets exhibit competitive results in most scenarios.

Figure 4(a) shows the stability of LI-KNN on the testing errors rates of the Iris data set. KNN always incurs higher error rates than our algorithms. The performance of LI-KNN is depicted for four different values of I. It is obvious that even with different values of I (given the same K), the results are similar, indicating that the performance of LI-KNN does not degrade when the number of informative points changes. In addition, with the change of K, LI-KNN is relatively stable regarding the error rate. The variation of LI-KNN is roughly 3%, meaning that K does not have a large impact on the results of LI-KNN.

Figure 4(b) compares Boosting and GI-KNN on the Breast Cancer data for the first 1,000 iterations. Overall, GI-KNN incurs lower error rates. From 620 to about 780 iterations GI-KNN's error rates are slightly higher than Boosting. However, the error rates of Boosting fluctuate quite a bit from 0.048 to 0.153, while GI-KNN is relatively stable and the performance varies only between (0.043, 0.058). Moreover, our algorithm obtains the optimal results significantly earlier.

[2] http://citeseer.ist.psu.edu

[3] http://www1.cs.columbia.edu/CAVE/software/softlib/coil-20.php

[4] During the experiment, we set $K_M = max(N/5, 50)$ and $\epsilon = 1$ according to the original paper.

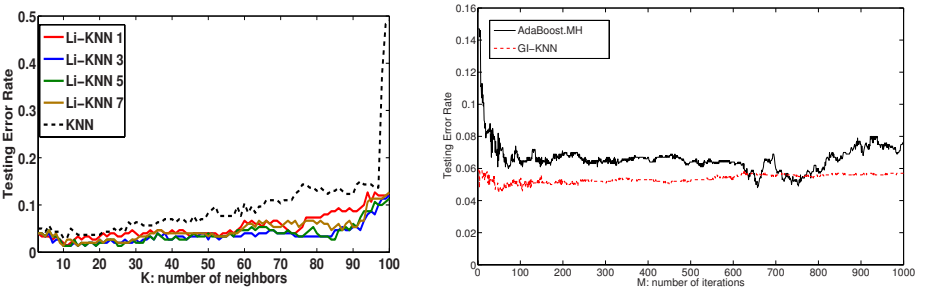
[5] The code is available at http://www.seas.upenn.edu/∼kilianw/lmnn/

[6] See http://www.cis.upenn.edu/∼crammer/code-index.html

[7] http://www.ics.uci.edu/∼mlearn/MLRepository.html

Table 1. Testing error rates for KNN, DANN, LMNN, SVM, Boosting, LI-KNN and GI-KNN of 10 UCI Benchmark data sets. N, D and C denote the number of instances, dimensionality and number of classes respectively. Numbers in the parentheses indicate the optimal neighbors K for KNN, DANN and LMNN, (K, I) for LI-KNN, and number of iterations M for GI-KNN and Boosting.

Data Sets	N	D	C	KNN	DANN	LMNN	**LI-KNN**	**GI-KNN**	SVM	Boosting
Iris	150	4	3	0.044 (9)	0.040 (5)	0.053 (3)	0.013 (9, 5)	**0.010** (25)	0.042	0.038 (45)
Wine	178	13	3	0.263 (3)	0.250 (7)	**0.031** (5)	0.137 (15, 1)	0.137 (13)	0.205	0.192 (135)
Glass	214	10	2	0.372 (5)	0.436 (5)	0.356 (3)	**0.178** (7, 3)	0.198 (202)	0.222	**0.178** (304)
Ionosphere	351	34	2	0.153 (5)	0.175 (7)	0.100 (5)	0.127 (5, 3)	0.127 (8)	**0.090**	0.092 (156)
Breast	699	9	2	0.185 (7)	0.120 (11)	0.927 (5)	0.080 (4, 1)	**0.045** (48)	0.052	0.048 (657)
Heart	779	14	5	0.102 (3)	0.117 (5)	0.092 (5)	**0.078** (7, 1)	**0.078** (192)	**0.078**	0.080 (314)
Digit	2000	649	10	0.013 (3)	0.010 (3)	0.009 (3)	**0.005** (19, 1)	**0.005** (137)	0.010	**0.005** (175)
Isolet	7797	617	26	0.078 (11)	0.082 (11)	0.053 (5)	0.048 (13, 3)	**0.042** (175)	0.044	**0.042** (499)
Pendigits	10992	16	10	0.027 (3)	0.021 (5)	**0.020** (3)	**0.020** (9, 1)	**0.020** (42)	0.033	0.038 (482)
Letter	20000	16	10	0.050 (5)	0.045 (3)	0.042 (5)	0.045 (5, 3)	0.040 (22)	**0.028**	0.031 (562)



(a) (b)

Fig. 4. (a) Results on Iris for K from 1 to 100. LI-KNN chooses the number of informative points (I) to be 1, 3, 5 and 7. **(b)** Results on Breast Cancer for AdaBoost.MH and GI-KNN (with $K = 5$ and $I = 1$). The best result for GI-KNN is slightly better (0.045) than that of AdaBoost.MH (0.048).

5.2 Application to Text Categorization

For text categorization experiments we use the CiteSeer data set consisting of nearly 750,000 documents primarily in the domain of computer science. Several types of data formats are indexed concurrently (*txt, pdf, ps, compressed files, etc.*). For the purpose of text categorization, we only make use of plain text files. For convenience, the metadata of the documents, i.e., the titles, abstracts and keyword fields are used in our experiments.

Document class labels are obtained from the *venue impact page*[8] which lists 1,221 major venues whose titles are named according to the DBLP[9] format. We make use of the top 200 publication venues listed in DBLP in terms of impact

[8] http://citeseer.ist.psu.edu/impact.html
[9] http://dblp.uni-trier.de/

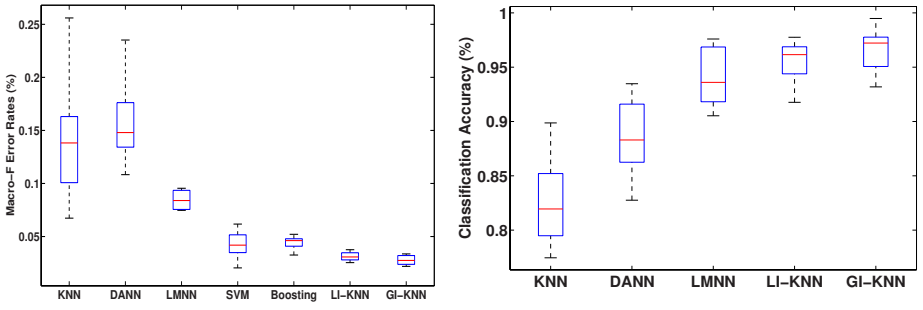


Fig. 5. (a) Box plots of macro-F error rates (100% minus Macro-F scores) on CiteSeer data set summarizes the average F scores on 193 classes. Our algorithms have very low error rates on average, with very small deviations. Plus(+) signs indicate the outliers. **(b)** Classification Accuracies on CiteSeer data set for KNN, LI-KNN and GI-KNN with different number of neighbors ($K = \{1, ..., 50\}$). Our algorithms generally demonstrate more stable results and higher accuracies.

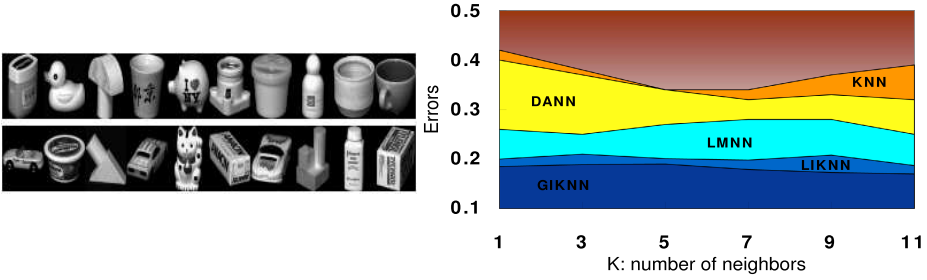
rates, each of which was referred as a class label. Furthermore, we intentionally filtered those classes that contain too few examples (i.e., less than 100 documents). Overall, the total number of documents used for the experiments is 118,058, which are divided into a training set and testing set by 10-fold cross-validation. Meanwhile, we keep the imbalance of the classes, i.e., some classes have more training examples than others. Documents are treated as *bag-of-words* and *tf-idf* weights of terms are used to generate the feature space.

Figure 5(a) shows the box plots *macro-F* error rates. The optimal parameters (e.g., the number of iterations M and so on) are estimated by 10-fold cross-validation on the training set. It is evident that the spread of the error distribution for our algorithms are very close to zero, which clearly indicates that LI-KNN and GI-KNN obtain robust performance over different classes. Meanwhile, our algorithms incur lower error rates even for small classes, making them potentially good choices for imbalanced data set classification.

We further show the stability of our algorithms by box plots of the classification accuracies for different number of neighbors. Figure 5(b) depicts the results of KNN, DANN and our algorithms for K from 1 to 50 with a fixed number of $I = 1$ (i.e., only the most informative neighbor). The mean accuracies are higher for LI-KNN and GI-KNN than KNN, and the variations are almost half as that of KNN and DANN.

5.3 Object Recognition on COIL-20

We use the processed version of COIL-20 database for object recognition. The database is made up with 20 gray-scale objects, each of which consists 72 images with size 128×128. Figure 6(a) shows a sample image of each of the 20 objects.



(a) (b)

Fig. 6. (a) Randomly generated images from each object in the COIL-20 database. **(b)** Results on COIL-20 with different number of neighbors.

We treat each object as one class, spliting the data into training and testing set with the proportion of 3:1. Figure 6(b) shows the classification errors regarding the 5 algorithms, where K ranges from 1 to 11. GI-KNN and LI-KNN generally outperform others with the best parameters, while both show stable results with the change of K.

5.4 Discussion

Our I-KNN procedures introduce two adjustable tuning parameters K and I, it is then desirable to automate the choice of them. Theoretically we did not prove the optimal choices for either K or I, however, empirical studies with different ranges of values on several data sets allow us to draw a rough conclusion. Basically, the value of K should be reasonably big. The bigger K is, the more information can be gathered to estimate the distribution of neighborhood for the query point. However, with the increase of K, the effort to compute the informativeness of neighbors (equation (2)) grows exponentially as well. In practice, we figured out that $K \in (7, 19)$ could be a good trade-off regardless of data size. Meanwhile, a smaller I is preferable to give the best predictions. Experimental results indicate that $I = 1$ and 3 usually achieve the best results, and the performance generally degrades with the increase of I. There is potentially another parameter to tune, i.e., η in equation (2), to balance the contribution of the first term. However, we only use $\eta = \frac{N_{x_j}}{N}$ here.

We have observed that most influential on the running time on both algorithms is the computation cost of the informativeness metric, of which the normalization factor (equation (2) and (6)) takes most of the time. To further improve the performance, we remove the normalization part in our experiments, i.e., equation (2) and (6). This significantly reduced the complexity of our model and did not jeopardize the performance very much.

Regarding the choice of the cost function C_m^i for GI-KNN training (line 8 in Algorithm 2), since GI-KNN has a different objective (to find the best weight vector) than boosting and other machine learning algorithms (to minimize a smooth convex surrogate of the 0-1 loss function), we did not compare the

performance between different loss functions like exponential loss, hinge loss and so on. Since we believe that the performance change will not be significant by exhaustively searching for the best loss function. The choice of different loss functions has already been extensively studied, interested readers can refer to [3] for details.

6 Conclusion and Future Work

This paper presented two approaches namely LI-KNN and GI-KNN to extending KNN method with the goal of improving classification performance. Informativeness was introduced as a new concept that is useful as a query-based distance metric. LI-KNN applied this to select the most informative points and predict the label of a query point based on the most numerous class with the neighbors; GI-KNN found the globally informative points by learning a weight vector from the training points. Rigorous experiments were done to compare the performance between our methods and KNN, DANN, LMNN, SVM and Boosting. The results indicated that our approaches were less sensitive to the change of parameters than KNN and DANN, meanwhile yielded comparable results to SVM and Boosting. Classification performance on UCI benchmark corpus, CiteSeer text data, and images suggests that our algorithms were application-independent and could possibly be improved and extended to diverse machine learning areas.

Questions regarding the GI-KNN algorithm are still open for discussion. Can we possibly prove the convergence of GI-KNN, or is there an upper-bound for this algorithm given specific K and I? More practically, is it possible to stop earlier when the optimum results are achieved? As a boosting-like algorithm, can we replace the 0-1 loss function with a smooth convex cost function to improve the performance? Furthermore, it will be interesting to see whether the informativeness metric can be applied to semi-supervised learning or noisy data sets.

References

1. Athitsos, V., Alon, J., Sclaroff, S.: Efficient nearest neighbor classification using a cascade of approximate similarity measures. In: CVPR '05, pp. 486–493. IEEE Computer Society, Washington, DC, USA (2005)
2. Athitsos, V., Sclaroff, S.: Boosting nearest neighbor classifiers for multiclass recognition. In: CVPR '05, IEEE Computer Society, Washington, DC, USA (2005)
3. Bartlett, P., Jordan, M., McAuliffe, J.: Convexity, classification and risk bounds. J. Amer. Statist. Assoc. 101, 138–156 (2006)
4. Cover, T., Hart, P.: Nearest neighbor pattern classification. IEEE Transactions on Information Theory 13(1), 21–27 (1967)
5. Crammer, K., Singer, Y.: On the algorithmic implementation of multiclass kernel-based vector machines. J. Mach. Learn. Res. 2, 265–292 (2002)
6. Domeniconi, C., Peng, J., Gunopulos, D.: Locally adaptive metric nearest-neighbor classification. IEEE Trans. Pattern Anal. Mach. Intell. 24(9), 1281–1285 (2002)

7. Friedman, J.: Flexible metric nearest neighbor classification. technical report 113, stanford university statistics department (1994)
8. Han, E.-H.S., Karypis, G., Kumar, V.: Text categorization using weight adjusted k -nearest neighbor classification. In: 5th Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD), pp. 53–65 (2001)
9. Hastie, T., Tibshirani, R.: Discriminant adaptive nearest neighbor classification. IEEE Trans. Pattern Anal. Mach. Intell. 18(6), 607–616 (1996)
10. Latourrette, M.: Toward an explanatory similarity measure for nearest-neighbor classification. In: ECML '00: Proceedings of the 11th European Conference on Machine Learning, London, UK, pp. 238–245. Springer-Verlag, Heidelberg (2000)
11. Peng, J., Heisterkamp, D.R., Dai, H.K.: LDA/SVM driven nearest neighbor classification. In: CVPR '01, p. 58. IEEE Computer Society, Los Alamitos, CA, USA (2001)
12. Schapire, R.E., Singer, Y.: Improved boosting algorithms using confidence-rated predictions. In: COLT' 98, pp. 80–91. ACM Press, New York (1998)
13. Weinberger, K.Q., Blitzer, J., Saul, L.K.: Distance metric learning for large margin nearest neighbor classification. In: NIPS (2005)
14. Zhang, H., Berg, A.C., Maire, M., Svm-knn, J.M.: Discriminative nearest neighbor classification for visual category recognition. In: CVPR '06, pp. 2126–2136. IEEE Computer Society, Los Alamitos, CA, USA (2006)