Stochastic Neighbor Compression

Matt J. Kusner MKUSNER@WUSTL.EDU
Stephen Tyree SWTYREE@WUSTL.EDU
Kilian Weinberger KILIAN@WUSTL.EDU
Kunal Agrawal KUNAL@WUSTL.EDU
Washington University in St. Louis, 1 Brookings Dr., St. Louis, MO 63130

Abstract

We present Stochastic Neighbor Compression (SNC), an algorithm to *compress* a dataset for the purpose of k-nearest neighbor (kNN) classification. Given training data, SNC learns a much smaller synthetic data set, that minimizes the stochastic 1-nearest neighbor classification error on the training data. This approach has several appealing properties: due to its small size, the compressed set speeds up kNN testing drastically (up to several orders of magnitude, in our experiments); it makes the kNN classifier substantially more robust to label noise; on 4 of 7 data sets it yields *lower* test error than kNN on the entire training set, even at compression ratios as low as 2%; finally, the SNC compression leads to impressive speed ups over kNN even when kNN and SNC are both used with ball-tree data structures, hashing, and LMNN dimensionality reduction—demonstrating that it is complementary to existing state-of-the-art algorithms to speed up kNN classification and leads to substantial further improvements.

1. Introduction

The k-nearest neighbors (kNN) decision rule classifies an unlabeled input by the majority label of its k nearest training inputs. It is one of the oldest and most intuitive classification algorithms (Cover & Hart, 1967). Nevertheless, when paired with domain knowledge (Belongie et al., 2002; Simard et al., 1992) or learned distance metrics (Goldberger et al., 2004; Davis et al., 2007; Weinberger & Saul, 2009), it is highly competitive in many machine learning applications (Tran & Sorokin, 2008). As machine learning algorithms are increasingly used in application settings, *e.g.* recommender systems (Sarwar et al., 2000), the kNN rule is particularly attractive because its predictions are easily explained.

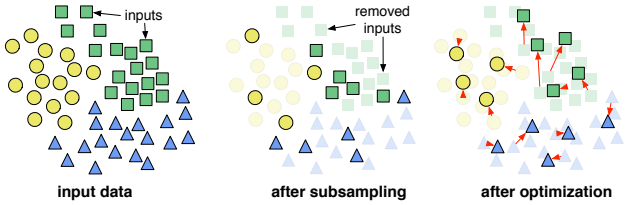

Figure 1. An illustration of the individual stages of SNC. The input data (*left*) is first subsampled uniformly (*middle*) and then optimized to minimize leave-one-out nearest neighbor error (*right*).

An important drawback of kNN is its slow test-time performance. Since it must compute the distances between the test input and all elements in the training set, it takes $O(dn)$ with respect to the data dimensionality d and the training set size n. Similarly, space requirements are also $O(dn)$, as the entire training set needs to be stored. This high time and space complexity makes computing the decision rule impracticable for time critical applications and large-scale datasets—a problem that is likely to remain relevant as datasets continue to grow.

There are three high-level approaches for speeding up the testing. First is to reduce the number of distance computations to some polylogarithmic function in n, through clever tree data structures, such as cover/ball trees (Beygelzimer et al., 2006; Omohundro, 1989), or hashing functions (Gionis et al., 1999; Andoni & Indyk, 2006). Although they often yield impressive speed ups, these methods still store the entire training set and their performance tends to deteriorate with increasing (intrinsic) data dimensionality. The second approach is to reduce the data dimensionality d through supervised dimensionality reduction, *e.g.* large margin nearest neighbors (LMNN) (Weinberger & Saul, 2008), which is particularly effective in combination with tree data structures. The third approach is to compress the

Proceedings of the 31^{st} International Conference on Machine Learning, Beijing, China, 2014. JMLR: W&CP volume 32.

training set by reducing the number of data inputs n. Prior works often involve data set *condensing* (or *thinning*) (Hart, 1968; Angiulli, 2005), which subsample the training data according to clever rules and remove redundant inputs. Alternative algorithms shrink the data to few cluster centers (Chang, 1974; Kohonen, 1990a), which can be optimized with multi-phase initialization procedures (Decaestecker, 1997; Liu & Nakagawa, 1999). Others learn prototypes by 'softening' the kNN decision rule at test-time (Bermejo & Cabestany, 1999), preventing the use of tree data structures.

In this paper, we introduce a novel approach for data set compression, *Stochastic Neighbor Compression (SNC)*, which falls into the third category of algorithms. SNC compresses the training data by learning a new set of m synthetic reference vectors, where $m \ll n$. Figure 1 illustrates our algorithm schematically. We initialize our compressed set with a small subset of the training set, sampled uniformly at random. We then optimize the position of these inputs directly to minimize the classification error on the training set. To this end, we relax the kNN rule into a stochastic neighborhood framework (Goldberger et al., 2004; Hinton & Roweis, 2002), which allows us to approximate the classification error of the training set with a continuous and differentiable function.

We are making four novel contributions: 1. we introduce and derive SNC, a novel data compression algorithm for kNN; 2. we demonstrate the efficacy of SNC on seven real world data sets and show that on all tasks it outperforms existing algorithms for data set reduction and on 4/7 data sets kNN on the full training set obtains even *higher* error rates than kNN with SNC—at a staggeringly low compression ratio of only 4%; 3. We conjecture and observe empirically that SNC substantially increases robustness of kNN to (label) noise; 4. We demonstrate that SNC works well alongside existing algorithms — such as ball trees, hashing, and dimensionality reduction — that speed up nearest neighbor classification. In fact, it adds impressive speed ups of one order of magnitude on top of the existing state-of-the-art.

2. Background

We denote the training data to be a set of input vectors $\{\mathbf{x}_1, \ldots, \mathbf{x}_n\} \subset \mathcal{R}^d$, arranged as columns in matrix $\mathbf{X} \in \mathcal{R}^{d \times n}$, and corresponding labels $\{y_1, \ldots, y_n\} \subseteq \mathcal{Y}$, where $\mathcal{Y}$ contains some finite number of classes.[1]

Our approach draws from two ideas in machine learning that use stochastic neighborhood distributions: stochastic neighborhood embeddings (Hinton & Roweis, 2002), and neighborhood components analysis (Goldberger et al., 2004). Here, we describe both in some detail.

Stochastic Neighborhood Embedding (SNE). Hinton & Roweis (2002) introduced SNE, an algorithm to visualize a given data set by learning a low-dimensional embedding in 2d or 3d. For two points $\mathbf{x}_i$ and $\mathbf{x}_j$, we define the *dissimilarity measure* d_{ij}^2; it is commonly an element of the Gaussian kernel $d_{ij}^2 = \gamma_i^2 \|\mathbf{x}_i - \mathbf{x}_j\|^2$, where γ_i^2 is the precision of the Gaussian distribution. The authors define a stochastic neighborhood, which captures the neighborhood relation between inputs $\mathbf{x}_i$ and $\mathbf{x}_j$ through probability p_{ij} of the event that $\mathbf{x}_i$ is assigned $\mathbf{x}_j$ as its nearest neighbor,

$$p_{ij} = \frac{\exp(-d_{ij}^2)}{\sum_{k=1}^{n} \exp\left(-d_{ik}^2\right)}. \tag{1}$$

The low dimensional embedding is optimized to approximately preserve the stochastic neighborhood distribution of the input data. More precisely, SNE minimizes the KL-divergence between the original (high dimensional) stochastic neighborhoods and the induced neighborhoods in the low dimensional space. This approach was recently further refined by Van der Maaten & Hinton (2008) to yield improved visualizations by substituting the local Gaussian distributions with Student t-distributions in the input space.

Neighborhood Components Analysis (NCA). Goldberger et al. (2004) introduced NCA, an algorithm that uses stochastic neighborhoods to learn a Mahalanobis pseudometric, $d_{ij} = \|\mathbf{A}(\mathbf{x}_i - \mathbf{x}_j)\|$. This metric is parameterized by a matrix $\mathbf{A}$ and is incorporated into the stochastic neighborhood in (1). In contrast to SNE, NCA is a supervised learning algorithm and optimizes this metric explicitly for kNN. To improve the kNN accuracy, it maximizes an approximation of the leave-one-out (LOO) training accuracy of the 1 stochastic neighbor rule. Under this rule, an input $\mathbf{x}_i$ with label y_i is classified correctly if its nearest neighbor is any $\mathbf{x}_j \neq \mathbf{x}_i$ from the same class $(y_j = y_i)$. The probability of this event can be stated as

$$p_i = \sum_{j:y_j=y_i} p_{ij}, \tag{2}$$

where we define $p_{ii} = 0$. NCA learns $\mathbf{A}$ by maximizing (2) over all inputs $\mathbf{x}_i \in \mathbf{X}$.

3. Stochastic Neighbor Compression

In this section, we describe our approach, called *Stochastic Neighbor Compression (SNC)*. SNC is inspired by the seminal works from Section 2 and uses a stochastic neighborhood distribution to reduce the training set, with n data inputs, into a compressed set with m vectors, where $m \ll n$. This much smaller compressed set is then used as a reference set during kNN testing. For standard kNN implementations, the test time and space complexity reduce from $\mathrm{O}(nd)$ to only $\mathrm{O}(md)$.

[1] Throughout this manuscript we will abuse notation slightly and treat $\mathbf{X}$ as a set of column vectors or matrix interchangeably (*i.e.* we allow the notation $\mathbf{x}_i \in \mathbf{X}$ but also $\mathbf{X}^\top \mathbf{y}$).

Stochastic Reference. We learn a new compressed set of reference vectors $\mathbf{Z} = [\mathbf{z}_1, \ldots, \mathbf{z}_m]$ with labels $\hat{y}_1, \ldots, \hat{y}_m$. This data set is initialized by uniformly subsampling m vectors from $\mathbf{X}$, while maintaining their exact labels. The labels $\hat{y}_i$ will be fixed throughout, whereas the vectors $\mathbf{Z}$ will be optimized. Let us define the probability that input $\mathbf{x}_i$ is assigned $\mathbf{z}_j$ as nearest reference vector as

$$p_{ij} = \frac{\exp(-\gamma^2 \|\mathbf{x}_i - \mathbf{z}_j\|^2)}{\sum_{k=1}^{m} \exp(-\gamma^2 \|\mathbf{x}_i - \mathbf{z}_k\|^2)}. \tag{3}$$

Input $\mathbf{x}_i$ is classified correctly if and only if it is paired with a reference vector from the same class $\hat{y}_j = y_i$. The probability of this event is precisely given by (2).

Objective. Ideally, we want $p_i = 1$ for all $\mathbf{x}_i \in \mathbf{X}$, corresponding to 100% classification accuracy of $\mathbf{X}$ on $\mathbf{Z}$. It is straight forward to see that the KL-divergence (Goldberger et al., 2004) between this "perfect" distribution and p_i is

$$KL(1||p_i) = -\log(p_i). \tag{4}$$

Our goal is to position the compressed set $\mathbf{Z}$ such that as many training inputs as possible are classified correctly. In other words, we need p_i to be close to 1 for all inputs $\mathbf{x}_i \in \mathbf{X}$. Hence, we define our loss function to sum over the KL-divergences (4) for all inputs in $\mathbf{X}$,

$$\mathcal{L}(\mathbf{Z}) = -\sum_{i=1}^{n} \log(p_i) \tag{5}$$

Gradient with respect to Z. We minimize the objective (5) with conjugate gradient descent. In order to state the gradients in simpler form, we first define two additional matrices $\mathbf{Q}, \mathbf{P} \in \mathcal{R}^{n \times m}$ as

$$[\mathbf{Q}]_{ij} = (\delta_{y_i, y_j} - p_i), \quad [\mathbf{P}]_{ij} = \frac{p_{ij}}{p_i}.$$

Here, $\delta_{y_i, y_j} \in \{0, 1\}$ denotes the Dirac Delta function and takes on value 1 if and only if $y_i = y_j$. Although we omit the details of the derivation, this notation allows us to state the gradient of $\mathcal{L}$ with respect to the compressed set $\mathbf{Z}$ entirely in matrix operations,

$$\frac{\partial \mathcal{L}}{\partial \mathbf{Z}} = -2\left(\mathbf{X}\Big(\mathbf{Q} \circ \mathbf{P}\Big) - \mathbf{Z}\Delta\Big(\Big(\mathbf{Q} \circ \mathbf{P}\Big)^\top \mathbf{1}_n\Big)\right), \tag{6}$$

where $\circ$ is the Hadamard (element-wise) product, $\mathbf{1}_n$ is the $n \times 1$ vector of all ones, and $\Delta(\cdot)$ signifies placing a vector along the diagonal of an otherwise 0 matrix.

Computational complexity. The computational complexity of each gradient descent iteration with respect to $\mathbf{Z}$ costs $\mathrm{O}\big(nm\big)$ to compute $(\mathbf{Q} \circ \mathbf{P})$, $\mathrm{O}\big(dnm\big)$ to compute $\mathbf{X}(\mathbf{Q} \circ \mathbf{P})$, and $\mathrm{O}\big(dm^2\big)$ to compute $\mathbf{Z}\Delta((\mathbf{Q} \circ \mathbf{P})^\top \mathbf{1}_n)$, resulting in $\mathrm{O}\big(dmn\big)$ overall complexity.

Algorithm 1 SNC in pseudo-code.

1: Inputs: $\{\mathbf{X}, \mathbf{y}\}$; new (compressed) data set size m
2: Initialize $\mathbf{Z}$ by class-based sampling m inputs from $\mathbf{X}$
3: Learn $\mathbf{Z}$ with conj. gradient descent, eq. (6)
4: Return $\mathbf{Z}$

Implementation. We optimize $\mathbf{Z}$ by minimizing (5) with conjugate gradient descent (we use a freely-available Matlab implementation [1]) and provide our implementation of SNC as open source available for download at `http://tinyurl.com/msovcfu`. The individual steps of the SNC approach are described in Algorithm 1.

3.1. Metric Learning Extension

Drawing directly on ideas proposed in Goldberger et al. (2004), for additional flexibility, we can extend (3) with an affine feature transformation matrix $\mathbf{A}$,

$$p_i = \sum_{j:\hat{y}_j = y_i} \frac{\exp\left(-\|\mathbf{A}(\mathbf{x}_i - \mathbf{z}_j)\|^2\right)}{\sum_{k=1}^{m} \exp(-\|\mathbf{A}(\mathbf{x}_i - \mathbf{z}_k)\|^2)}. \tag{7}$$

Let us denote the corresponding loss function as $\mathcal{L}^{\mathbf{A}}$. The resulting objective can be minimized with respect to $\mathbf{A}$ and $\mathbf{Z}$. This extension allows us to automatically optimize the feature scale γ^2 by setting $\mathbf{A} = \gamma^2 \mathbf{I}$; rescale features with a diagonal matrix $\mathbf{A} = \Delta$; or induce dimensionality reduction with a rectangular matrix, *i.e.* $\mathbf{A} \in \mathcal{R}^{r \times d}$.

Gradients w.r.t. A and Z. The gradient of $\mathcal{L}$ w.r.t. $\mathbf{A}$ is similar to the NCA gradient in Goldberger et al. (2004),

$$\frac{\partial \mathcal{L}^{\mathbf{A}}}{\partial \mathbf{A}} = -2\mathbf{A} \sum_{i=1}^{n} \sum_{j=1}^{m} \frac{p_{ij}}{p_i} q_{ij} \mathbf{v}_{ij} \mathbf{v}_{ij}^\top, \tag{8}$$

where we abbreviate $\mathbf{v}_{ij} = (\mathbf{x}_i - \mathbf{z}_j)$ and $q_{ij} = [\mathbf{Q}]_{ij}$. The gradient of $\mathcal{L}^{\mathbf{A}}$ w.r.t. $\mathbf{Z}$ results in a modification of (6):

$$\frac{\partial \mathcal{L}^{\mathbf{A}}}{\partial \mathbf{Z}} = 2\mathbf{A}^\top \mathbf{A} \frac{\partial \mathcal{L}}{\partial \mathbf{Z}}. \tag{9}$$

Due to additional multiplications by $\mathbf{A}$, the time complexity of each gradient iteration increases to $\mathrm{O}\big(d^2 mn + d^3\big)$. (The cubic term drops if $\mathbf{A}$ is diagonal or of the form $\gamma^2 \mathbf{I}$.)

Practical aspects. We find that the form $\mathbf{A} = \gamma^2 \mathbf{I}$ leads to comparable results as the diagonal or full matrix. It has the added advantage that it is substantially faster and that it alleviates the need to multiply the test data with $\mathbf{A}$, as the kNN decision rule is invariant to uniform feature scaling. Optimizing the scaling factor γ^2 does however affect the compressed set $\mathbf{Z}$. Also, optimizing γ^2 with conjugate

[1] `http://tinyurl.com/minimize-m`

Table 1. Characteristics of datasets used in evaluation.

DATASET STATISTICS			
NAME	n	$\lvert\mathcal{Y}\rvert$	d (d_L)
YALE-FACES	1961	38	8064 (100)
ISOLET	3898	26	617 (172)
LETTERS	16000	26	16 (16)
ADULT	32562	2	123 (50)
W8A	49749	2	300 (100)
MNIST	60000	10	784 (164)
FOREST	100000	7	54 (54)

gradient prior to optimizing $\mathbf{Z}$ (instead of jointly) leads to similar results and may be preferred in practice due to its improved running time. In our experiments, we initialize γ^2 with cross-validation and optimize it prior to learning. We pick the initialization that yields minimal training error.

4. Results

We evaluate the efficacy of SNC on seven benchmark data sets. We begin with a brief description of the individual learning tasks and then evaluate the compression ratio and test error, training time, sensitivity to noise and finally visualize the SNC decision boundary and reference vectors.

Dataset descriptions. We evaluate SNC and other training set reduction baselines on seven classification datasets detailed in Table 1. *YaleFaces* (Georghiades et al., 2001) consists of gray-scale face images of 38 individuals under varying (label invariant) illumination conditions. The task is to identify the individual from the image pixel values. *Isolet*[1] is a collection of audio feature vectors of spoken letters from the English alphabet. The task is to identify which letter is spoken based on the recorded (and pre-processed) audio signal. *Letters*[1] is derived from images of English capital letters, the learning task is to identify the letter type based on font specific features. *Adult*[1] contains U.S. census income and personal statistics, the task is to predict if a household has an income over $\$50,000$. *W8a*[2] contains keyword attributes extracted from web pages and the task is to categorize a web page into a one of a set of predefined categories. *MNIST*[3] is a set of gray-scale handwritten digit images; the task is to identify the digit value from the image pixels. *Forest*[1] contains geological and map-based data, and the task is to identify the type of ground cover (e.g. tree type) in a given area of a map.

In addition to these data sets, we also used the *USPS*[4] handwritten digits data set as a development set. As we evaluated SNC multiple times on its test portion, and we want to clearly separate development and evaluation data, we are not including it in this result section. The results are comparable to the benchmark sets included in this section.

[1] http://tinyurl.com/uci-ml-data
[2] http://tinyurl.com/libsvm-data
[3] http://tinyurl.com/mnist-data
[4] http://tinyurl.com/usps-data

Preprocessing. Weinberger & Saul (2008) show that Large Margin Nearest Neighbors (LMNN) is an effective method to speed up kNN search through dimensionality reduction, that is, by reducing the parameter d in the running time $O(nd)$. LMNN learns a projection into a lower dimensional space that speeds up kNN while maintaining (or improving) the classification error. We can validate this observation on our benchmark tasks and therefore pre-process all datasets with LMNN, which improves the kNN speed and accuracy for nearly all datasets.

For Isolet and MNIST, the dimensionality is reduced as described in Weinberger & Saul (2009). For the remaining datasets, if the input dimensionality d is ≥ 200 it is reduced to 100 with LMNN, and if it is between 100 and 200, it is reduced to 50. For the YaleFaces data set, we follow Weinberger & Saul (2009) and first rescale the images to 48x42 pixels, then reduce the dimensionality with PCA (to 200) while omitting the leading 5 principal components (which capture large variations in image brightness). Finally, we apply LMNN to reduce the dimensionality further to $d = 100$. Table 1 lists the dimensionality of each dataset before (d) and after (d_L) LMNN preprocessing. For Forest, we follow Angiulli (2005) who subsample uniformly.

Implementation. For purposes of this evaluation, SNC is initialized by subsampling inputs based on the class distribution up to the desired compression rate. For testing, we use the 1NN (1 Nearest Neighbor) rule for all of the algorithms. Results of SNC and of any initialization-dependent baselines are reported with the average and standard deviation over 5 runs. Neither YaleFaces nor Forest have predefined test sets and so we report the average and standard deviations in performance over 5 and 10 splits, respectively.

Baselines. Figure 2 shows the test error of kNN evaluated on a compressed training set generated by SNC (solid blue line) with $\mathbf{A} = \gamma^2\mathbf{I}$. We depict varying rates of compression, and compare against the following related baselines: 1. kNN without compression both before (brown dotted line) and after LMNN dimensionality reduction (red dotted line), 2. kNN on a reference set subsampled uniformly from the training set based on the class balance (pink dotted line), 3. Approximate kNN via locality-sensitive hashing *LSH* (Gionis et al., 1999), using the implementation by Aly et al. (2011) (purple dotted line) 4. *CNN* (Hart, 1968) (orange line), and 5. *FCNN* (Angiulli, 2005) (green line). CNN and FCNN select training-consistent subsets and are arguably the most popular training set reduction algorithms for kNN. Both methods are briefly described in Section 5.

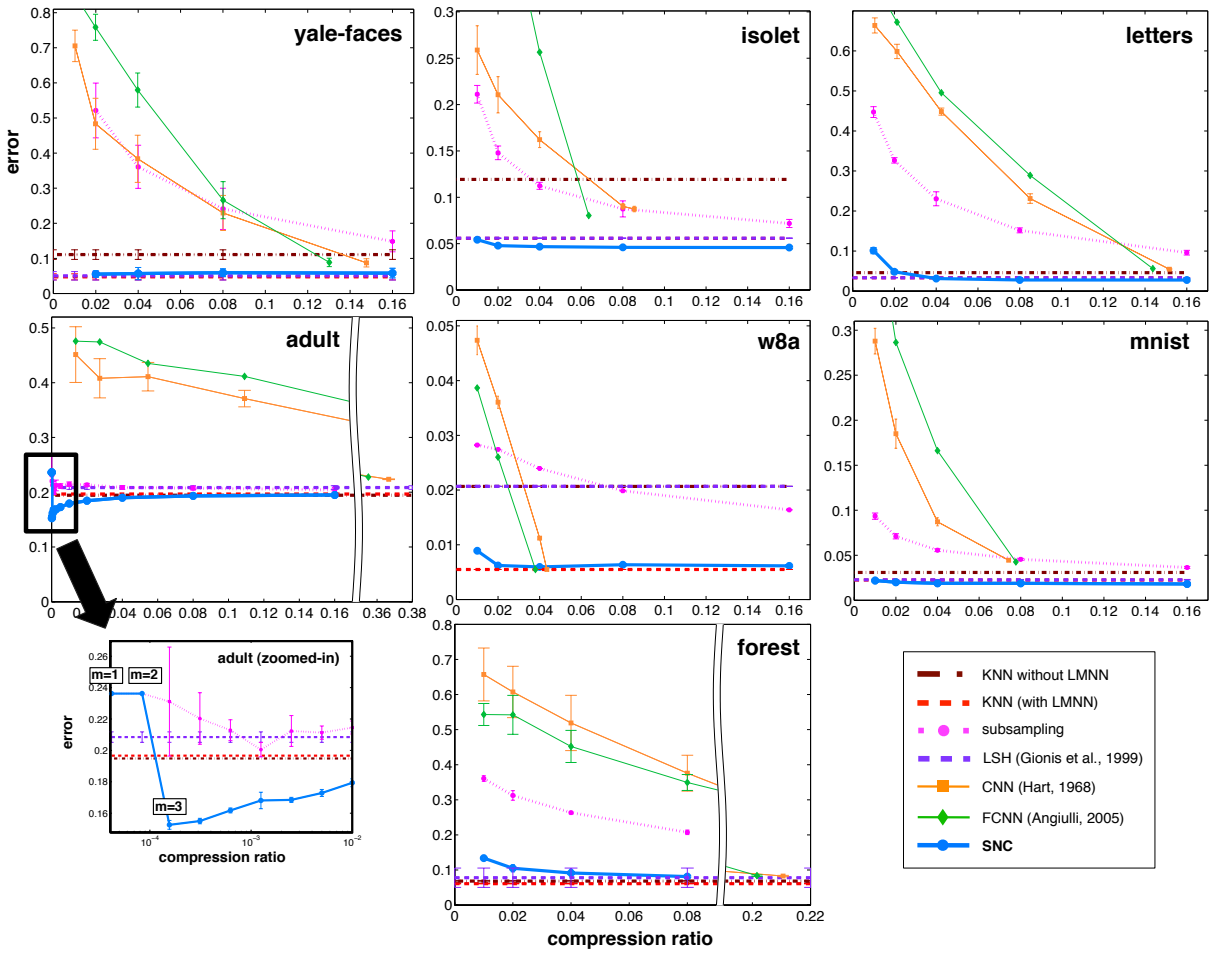

Figure 2. kNN test error rates after training set compression obtained by various algorithms. See text for details.

Both SNC and subsampling can be performed at varying rates of compression and are plotted at compression ratios $\{1\%, 2\%, 4\%, 8\%, 16\%\}$. (We omit the 16% compression rate for forest, due to its large size.) kNN with and without LMNN does not do any compression and for better readability both methods are depicted as horizontal lines. For LSH we cross-validate over the number of tables and hash functions and select the fastest setting that has equal or less leave-one-out error compared to kNN without LSH (for larger datasets, we performed the LSH cross-validation on class-balanced subsamples of the training set: 10% subsamples of Adult, W8a and MNIST, and 5% of Forest). Identical to SNC, we plot average LSH test error and standard deviation for multiple random initializations. CNN and FCNN do not have a parameter for compression ratio. However both algorithms incrementally add inputs to the reference set and for comparison to our method with variable compression rate we have depicted the errors of partial compressed sets. For CNN, we also plot standard deviations as the algorithm is order dependent. In full disclosure, we want to point out that both CNN and FCNN as intended by the authors would only output a single compressed set (the rightmost point of the respective plot lines).

Error and Compression. We observe several general trends from the results in Figure 2. Simply subsampling the training set yields high error rates, showing that *optimization* of the compressed data set is crucial to obtain good compression/error trade-offs. SNC performs extremely well on all data sets even with a compression ratio as low as 2%. In fact, SNC clearly outperforms all other compression methods in terms of compression/error trade-off across *all* data sets—often yielding significantly lower test error rates than CNN and FCNN under only a fraction of their final compression ratio. SNC at $\geq 4\%$ matches (up to significance) or outperforms LSH error on every dataset. Further, on almost all data sets (except W8a and Forest), kNN with SNC can match the test error rates (with and without LMNN) using the full training data even at very high compression ratios ($2 - 4\%$). In fact, on 4/7 data sets,

Table 2. SNC training times.

TRAINING TIMES					
DATASET	COMPRESSION RATIO				
	1%	2%	4%	8%	16%
YALE-FACES	–	4s	6s	9s	15s
ISOLET	11s	17s	28s	50s	1m 26s
LETTERS	41s	1m 18s	2m 44s	4m 34s	8m 13s
ADULT	2m 27s	4m 1s	7m 39s	12m 51s	23m 18s
W8A	6m 5s	10m 19s	19m 26s	39m 12s	1h 12m
MNIST	17m 18s	36m 43s	1h 13m	2h 17m	4h 57m
FOREST	17m 38s	33m	55m 44s	1h 45m	–

kNN with SNC at a compression ratio of 4% achieves even *lower* test error than kNN using the full training set.

The last observation is particularly surprising, as one would expect an increase in error due to compression, rather than a reduction. However, one explanation for this effect is that SNC optimizes the compressed data especially to do well with kNN classification. A good example is the Adult data set, which has a strong class imbalance with 78% of the data belonging to one class. In other words, this is a data set in which kNN barely outperforms predicting the most common label. With high compression, SNC can position its learned reference vectors in a way to learn a simpler decision boundary and outperform kNN drastically with 0.15 vs. 0.20 error (zoomed in portion of the graph).

Time complexity and training time. Training times for SNC, averaged across 5 runs, are given in Table 2. SNC (with $\mathbf{A} = \gamma^2\mathbf{I}$) is expected to scale with complexity $O(dmn)$ per iteration. As the size of the compressed set m doubles between columns, training times roughly double as well. Variations in dimensionality and training set sizes among datasets make comparisons along the columns less precise, but training times do not seem to exceed theoretical expectations. All experiments were performed on an 8-core Intel L5520 CPU with 2.27GHz clock frequency.

Speed-up at test time. At testing time, standard implementations of kNN testing will compute the distances between each test point and *all* reference set points. However, dimensionality reduction (Weinberger & Saul, 2008) or clever structures, such as ball trees (Omohundro, 1989) or hash tables (Gionis et al., 1999), can vastly reduce test time. Table 3 (*Left*) shows the test time speed-up obtained through kNN with an SNC compressed reference set versus kNN with the full training set (after dimensionality reduction with LMNN). The table depicts the speed-up with the standard exhaustive neighbor search (in black), and accelerated versions with ball-trees (in teal) and LSH (in purple), each applied before and after SNC compression.

With a compression ratio $\geq 4\%$ the error rates on all data sets are lower or very close to those obtained with kNN without compression. However, we highlight settings that match or outperform the uncompressed kNN error in bold. For the standard exhaustive implementation in Table 3, speed-ups achieved by SNC generally exceed those expected at the given compression ratio (e.g $> 100\times$ speed-up at 1% compression). This may be due to favorable cache effects from using a smaller reference set. This table shows that SNC compression can lead to notable speed-ups even when using ball-trees and hashing, demonstrating that SNC can be used in conjunction both methods for even greater speed-ups. The results with ball-trees are particularly impressive, as all inputs have undergone dimensionality reduction, which is known to significantly improve ball-tree speed-up itself (Weinberger & Saul, 2008).

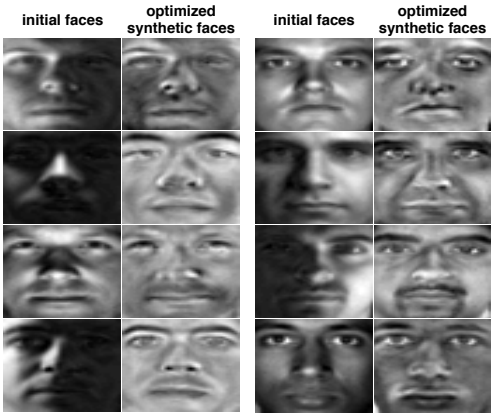

Figure 3. YaleFaces before and after compression.

Table 3 (*Right*) compares the number of distance computations required for kNN search with SNC at 4% compression versus kNN search with ball-trees or LSH using the full training set. The implementations of kNN, ball-trees, and LSH may not be directly comparable, so we use distance computations as a proxy for speed. SNC requires fewer distance computations than either method on all datasets except Adult (with LSH) and Forest.

In summary, our results give strong indication that 1. SNC obtains drastic speed-ups during test-time while only marginally increasing or, at times, decreasing kNN error rates; and 2. it is an effective complement and competitor to existing state-of-the-art strategies for speeding up kNN.

Compressed synthetic faces. Figure 3 visualizes synthetic SNC reference vectors learned on the YaleFaces data. Here, we preprocess the data using PCA and learn a compressed data set using the first 100 principal components. The figure shows (reconstructed) input faces that are initially subsampled to be in the compressed set (left columns) and the resulting optimized (synthetic) faces after SNC. It is interesting to observe that SNC is easily able to identify and emphasize distinguishing characteristics (*e.g.* mustaches) while ignoring noisy qualities (*e.g.* lighting).

Table 3. Left: Speed-up of kNN testing through SNC compression without a data structure (in black) *on top* of ball-trees (in teal) and LSH (in purple). Results where SNC matches or exceeds the accuracy of full kNN (up to statistical significance) are in **bold**. *Right:* Speed-up of SNC at 4% compression *versus* ball-trees and LSH on the full dataset. **Bold** text indicates matched or exceeded accuracy.

	SPEED-UP												SNC 4% COMPARISON	
	COMPRESSION RATIO												DISTANCE COMPS.	
DATASET	1%			2%			4%			8%			16%			BALL-TREES	LSH
YALE-FACES	–	–	–	**28**	**17**	**3.6**	**19**	**11**	**3.5**	**12**	**7.3**	**3.2**	**6.5**	**4.2**	**2.8**	**7.1**	**21**
ISOLET	**76**	**23**	**13**	**47**	**13**	**13**	**26**	**6.8**	**13**	**14**	**3.7**	**13**	**7.0**	**2.0**	**13**	**13**	**14**
LETTERS	143	9.3	100	73	6.3	61	**34**	**3.6**	**34**	**16**	**2.0**	**17**	**7.6**	**1.1**	**8.4**	**3.3**	**23**
ADULT	**156**	**56**	**3.5**	**75**	**28**	**3.4**	**36**	**15**	**3.3**	**17**	**7.3**	**3.1**	**7.8**	**3.8**	**3.0**	**17**	**0.7**
W8A	146	68	39	71	36	35	33	19	26	15	10	18	7.3	5.5	11	13	**2.1**
MNIST	**136**	**54**	**84**	**66**	**29**	**75**	**32**	**16**	**57**	**15**	**8.4**	**37**	**7.1**	**3.6**	**17**	**11**	**8.5**
FOREST	146	3.1	12	70	1.6	11	32	0.90	10	15	1.1	7.0	–	–	–	0.15	**0.35**

Label noise. An interesting observation from the results in Figure 2 is that SNC compression at times *improves* the kNN test error. We conjectured earlier that one explanation may be that kNN with SNC can yield a smoother decision boundary. This effect may be particularly beneficial in scenarios with label noise. We test this conjecture in Figure 4 (*Right*), where we examine the kNN error on the Letters dataset under increasing random label corruption (for $k=1$ and $k=3$). The figure shows clearly that the kNN error increases approximately linearly with label noise. SNC with 2%, 4%, 8% compression seems to smooth out mislabeled inputs and yields a significantly more robust kNN classifier. In contrast, CNN, FCNN and also subsampling (not shown in the figure to reduce clutter) do not mitigate the effect of label noise and at times tend to even amplify the test error. It is worth noting out that, for this experiment, CNN and FCNN were run to convergence and had significantly higher compression ratios than SNC's fixed ratios. For instance, at 0.32 label noise, CNN and FCNN both use more than 65% of the data in their compressed set.

Visualized decision boundary. Figure 4 (*Left*) shows the reference set (white circles) and decision rule (colored shading) before and after SNC optimization. The data set consists of USPS handwritten digits $\{0, 1, 2, 3, 4\}$ after projection onto 2D with LMNN (class membership is indicated by color). The left plot shows the (randomly subsampled) initialization of the reference set and the decision boundaries generated by this set. The SNC vectors are learned with 4% compression ratio and different scaling factors ($\gamma^2 = 1/2$ and $\gamma^2 = 8$, respectively). The decision regions for each class are notably erroneous in several regions prior to reference set optimization. For both scale factors γ^2, optimizing the reference set with SNC improves the decision boundary over the random sampling.

With a small γ^2 (middle pane) the corresponding large variance of the stochastic neighborhood encourages reference set vectors to produce results resembling a mixture model (Bishop, 2006), where groups of compressed vectors act as mixture component centers. The cluster centers are not at the expected locations (i.e. nested within a dense set of vectors), but are pushed outwards to accommodate the (possibly too) small γ^2. Larger values of γ^2 (right pane), converge to the decision boundaries between classes. Indeed, for every compressed input close to the boundary there are one or more representing the neighboring classes.

It is interesting to observe that by controlling γ^2, SNC can learn very different compressed sets. For naturally clustered data sets, larger values of γ^2 may be preferred, to make reference vectors represent dense regions in the data distribution. For data without such structure, smaller values of γ^2 may result in lower errors, as SNC can model the decision boundary more accurately. As γ^2 is an important hyperparameter that changes the characteristics of the compressed set, its initial value should be set via cross-validation prior to potential further optimization.

5. Related Work

Research on speeding up kNN is almost as old as the kNN rule itself. A big fraction concentrates on developing clever data structures in order to reduce the number of test time comparisons; examples include KD trees (Bentley, 1975), cover- and ball-trees (Beygelzimer et al., 2006; Omohundro, 1989) and hashing (Gionis et al., 1999). In this section we review prior research that takes the complementary approach of reducing the size of training data. We group these approaches under three general categories: *training set consistent sampling*, *prototype generation*, and *prototype positioning*. (A complete survey is Toussaint (2002).)

Training Set Consistent Sampling. The earliest work, called Condensed Nearest Neighbors (CNN) (Hart, 1968), starts by randomly selecting a single input and creating a 'reference' set, which it will use to classify the training data. It adds misclassified training inputs sequentially to this reference set until the full training set is correctly classified. There have been multiple extensions to CNN including post-processing methods (Gates, 1972) and stricter selection rules (Tomek, 1976; Devi & Murty, 2002). Recently Fast CNN (Angiulli, 2005) makes CNN sub-quadratic in n to train (as opposed to O(n^3) naïvely for CNN), with empirically better test generalization. All these methods retain a set of inputs that are a subset of the original training set.

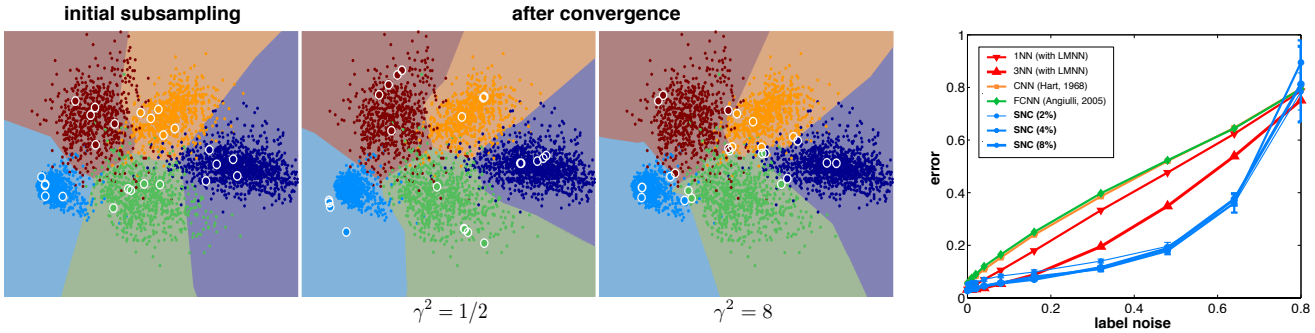

Figure 4. Left: The decision rule and SNC data set (white circles) learned from 2d USPS digits under varying γ^2. *Right:* kNN test error rates with various data set reduction methods on the Letters dataset under artificial label noise.

Prototype Generation. Prototype generation methods create 'prototypes' for the training data, which allow the inclusion of new, artificial instances in the reduced data set (Bandyopadhyay & Maulik, 2002). These generated inputs are typically found via clustering. Chang (1974) propose repeatedly merging nearest neighbors within a class while the training error is unaffected. Mollineda et al. (2002) merge clusters of inputs, until the LOO training error increases over a pre-determined threshold.

Prototype Positioning. There has also been work on learning the positions of this prototype subset. Toussaint (2002) describe using proximity graphs to generate a reduced set. Salzberg et al. (1995) determine the best set of prototypes to exactly reproduce any decision boundary requested. There has been a body of work on using learning vector quantization (LVQ) (Kohonen, 1990b) for designing kNN prototypes as well. In general, all of these methods consider local properties to optimize the reference set whereas SNC incorporates global information from the entire dataset through the stochastic neighborhoods.

Most similar to SNC are methods which optimize prototypes to maximize Gaussian mixtures (which can be interpreted as a stochastic neighborhood). The stochastic neighborhood, described in Section 2, smoothly models the probability that each prototype is the nearest neighbor of a given training point using a Gaussian likelihood. The primary differences between SNC and these methods in initial prototype selection and how inputs are classified during test-time. Decaestecker (1997) and Liu & Nakagawa (1999) use a three-phase search for initial prototypes that involves k-means and two different elimination rules. Bermejo & Cabestany (1999) develop a variation of kNN called 'soft'-kNN that uses the R nearest prototypes to classify inputs. They then maximize the probability of a correct prediction using the R closest prototypes. In effect, if $R = n$ their objective is similar to ours. However, because soft-kNN always considers all of the R prototypes to make a classification this setting cannot be sped up with ball trees, so the authors cross validate the soft-kNN error over R. This cross-validation does not consider the objective of reducing the size of R, which may result in a model that requires significantly more computation than SNC.

6. Conclusions

We have introduced SNC, which is a simple and efficient algorithm to compress the training set for kNN. Our experiments indicate that SNC reference set almost always provides comparable or lower test errors than the training set with compression rates of 2-4%, leading to an order of magnitude improvement in testing time.

One important direction for future work is extending SNC to neighborhoods of higher cardinality (i.e. $k = 3, 5$). Tarlow et al. (2013) provide a rigorous and efficient approach for extending the stochastic neighborhood framework to larger neighborhoods. Another consideration is that, as nearest-neighbor search in non-Euclidean spaces becomes more popular (e.g. covariance matrices (Chandrasekhar et al., 2009)) it is valuable to consider if SNC can be extended to this setting. Another interesting direction is if one can learn compressed reference inputs while simultaneously optimizing ball-tree structures.

In summary, we believe that SNC is a robust and highly effective algorithm that is based on straight-forward gradient descent optimization. As it (a) seems to consistently improve kNN *speed, accuracy* and *robustness*, and (b) can be combined with existing algorithms to improve kNN, we hope it will be useful to researchers and practitioners in machine learning and its application domains.

Acknowledgements. KQW, MK, ST are supported by NSF grants 1149882, 1137211. ST and KA are supported by NSF grants 1150036, 1218017. ST is supported by an NVIDIA Graduate Fellowship. The authors thank Laurens van der Maaten for helpful discussions. Computations were performed via the Washington University Center for High Performance Computing, partially provided through grant NCRR 1S10RR022984-01A1.

References

Aly, Mohamed, Munich, Mario, and Perona, Pietro. Indexing in large scale image collections: Scaling properties and benchmark. In *WACV*, pp. 418–425. IEEE, 2011.

Andoni, Alexandr and Indyk, Piotr. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. In *FOCS*, pp. 459–468. IEEE, 2006.

Angiulli, F. Fast condensed nearest neighbor rule. In *ICML*, pp. 25–32, 2005.

Bandyopadhyay, S. and Maulik, U. Efficient prototype reordering in nearest neighbor classification. *Pattern Recognition*, 35(12): 2791–2799, 2002.

Belongie, S., Malik, J., and Puzicha, J. Shape matching and object recognition using shape contexts. *PAMI*, 24(4):509–522, 2002.

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

Bermejo, S. and Cabestany, J. Adaptive soft k-nearest-neighbor classifiers. *Pattern Recognition*, 32:2077–2979, 1999.

Beygelzimer, A., Kakade, S., and Langford, J. Cover trees for nearest neighbor. In *ICML*, pp. 97–104, 2006.

Bishop, C.M. *Pattern recognition and machine learning*. Springer New York, 2006.

Chandrasekhar, Vijay, Takacs, Gabriel, Chen, David, Tsai, Sam S, Singh, Jatinder, and Girod, Bernd. Transform coding of image feature descriptors. In *IS&T/SPIE Electronic Imaging*, 2009.

Chang, C.L. Finding prototypes for nearest neighbor classifiers. *IEEE Transactions on Computers*, 100(11):1179–1184, 1974.

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

Davis, J.V., Kulis, B., Jain, P., Sra, S., and Dhillon, I.S. Information-theoretic metric learning. In *ICML*, pp. 209–216, 2007.

Decaestecker, C. Finding prototypes for nearest neighbour classification by means of gradient descent and deterministic annealing. *Pattern Recognition*, 30(2):281–288, 1997.

Devi, V.S. and Murty, M.N. An incremental prototype set building technique. *Pattern Recognition*, 35(2):505–513, 2002.

Gates, G.W. The reduced nearest neighbor rule. *IEEE Transactions on Information Theory*, 18:431–433, 1972.

Georghiades, A.S., Belhumeur, P.N., and Kriegman, D.J. From few to many: Illumination cone models for face recognition under variable lighting and pose. *PAMI*, 23(6):643–660, 2001.

Gionis, Aristides, Indyk, Piotr, Motwani, Rajeev, et al. Similarity search in high dimensions via hashing. In *VLDB*, volume 99, pp. 518–529, 1999.

Goldberger, J., Hinton, G.E., Roweis, S.T., and Salakhutdinov, R. Neighbourhood components analysis. In *NIPS*, pp. 513–520. 2004.

Hart, P.E. The condensed nearest neighbor rule. *IEEE Transactions on Information Theory*, 14:515–516, 1968.

Hinton, G.E. and Roweis, S.T. Stochastic neighbor embedding. In *NIPS*, pp. 833–840. MIT Press, 2002.

Kohonen, T. Improved versions of learning vector quantization. In *IJCNN*, pp. 545–550. IEEE, 1990a.

Kohonen, T. The self-organizing map. *Proceedings of the IEEE*, 78(9):1464–1480, 1990b.

Liu, C. L. and Nakagawa, M. Prototype learning algorithms for nearest neighbor classifier with application to handwritten character recognition. In *ICDAR*, pp. 378–381. IEEE, 1999.

Mollineda, R.A., Ferri, F.J., and Vidal, E. An efficient prototype merging strategy for the condensed 1-nn rule through class-conditional hierarchical clustering. *Pattern Recognition*, 35 (12):2771–2782, 2002.

Omohundro, S.M. *Five balltree construction algorithms*. International Computer Science Institute, Berkeley, 1989.

Salzberg, S., Delcher, A.L., Heath, D., and Kasif, S. Best-case results for nearest-neighbor learning. *PAMI*, 17(6):599–608, 1995.

Sarwar, B., Karypis, G., Konstan, J., and Riedl, J. Application of dimensionality reduction in recommender system-a case study. Technical report, DTIC Document, 2000.

Simard, P., LeCun, Y., and Denker, J.S. Efficient pattern recognition using a new transformation distance. In *NIPS*, pp. 50–58, 1992.

Tarlow, D., Swersky, K., Charlin, L., Sutskever, I., and Zemel, R. Stochastic k-neighborhood selection for supervised and unsupervised learning. In *ICML*, pp. 199–207, 2013.

Tomek, I. Two modifications of cnn. *IEEE Trans. on Systems, Man, and Cybernetics*, (11):769–772, 1976.

Toussaint, G.T. Proximity graphs for nearest neighbor decision rules: recent progress. *Interface*, 34, 2002.

Tran, D. and Sorokin, A. Human activity recognition with metric learning. In *ECCV*, pp. 548–561. Springer, 2008.

Van der Maaten, L. and Hinton, G. Visualizing data using t-sne. *JMLR*, 9(2579-2605):85, 2008.

Weinberger, K.Q. and Saul, L.K. Fast solvers and efficient implementations for distance metric learning. In *ICML*, pp. 1160–1167, 2008.

Weinberger, K.Q. and Saul, L.K. Distance metric learning for large margin nearest neighbor classification. *JMLR*, 10:207–244, 2009.