Project Assignment 1 - Winter 2011
k-Nearest Neighbor Algorithm

Andrew
Computer Science and Enginering
University of California San Diego
aheiberg@ucsd.edu

André Christoffer Andersen
Computer Science and Enginering
University of California San Diego
andre@andersen.im

Fabian
Computer Science and Enginering
University of California San Diego
fsiddiqi@ucsd.edu

January 20, 2011

Abstract

The goal of this project is to implement and benchmark various k nearest-neighbor (kNN) classifiers in order to do optical character recognition (OCR) on handwritten digits from 0 to 9. Two main distance functions are used: Minkowski and generalized similarity, with and without weights learned by linear regression. kNN algorithms utilizing the former yielded slightly more accurate and significantly faster results. Random subsets of both the training and test data were repeatedly taken throughout the experiments to avoid over-fitting in parameter selection. The distance functions are also used against a data set sporting a 90% reduction in dimensionality, with the only Minkowski function giving similar accuracy to the original set.

1 Introduction

The task assigned to the kNN algorithm is to correctly label handwritten digits.The training and test data sets are drawn from a total of 9298 images from the standardized USPS data set. The handwritten digits are each represented by a 16 by 16 pixel gray-scale image with associated label. Internally, however, the digits are 256 dimensional vectors where each element is represented by a 32-bit float point ranging from -1 to +1.

The kNN algorithm itself is uncomplicated. Its success or failure depends on an intelligently chosen distance function and tie-breaking strategy. Per the assignments suggestion, the Minkowski distance (of which Euclidean distance is a special case) and weighted similarity are used.

Although the whole data set only occupies just over 9MB of memory, the nature of some of the algorithms explode this number in far excess of available computational and memory resources at hand. We have therefore put special effort in to dimensional pruning in order to reduce each data points dimensionality. In general we have conducted principal component analysis (PCA) to successfully reduce the data set with over 90% without loss of accuracy using Minkowski distances. For the generalized similarity function, effort was directed at reducing the all-to-all 256^2 pixel comparisons.

Best case results mirror what is found in literature: a best case of 94.52% accuracy. This was independent of whether we used the dimensionally reduced or original data set. The different approaches to generalized similarity yielded discrepant accuracies, whereas the performance with the Minkowski was more consistent. The different approaches and posited reasons for the discrepancies are presented below.

2 Design of Algorithms

2.1 Principles of the k-Nearest Neighbor Algorithm

The overarching algorithm for the general kNN classifier is as follows: a test point is presented to the classifier whereby the k closest training points are located according some distance function. The k selected training points are now represented by a vector of labels and distances. A plurality function is then applied to the k labels yielding which label to return. If there is a tie between two or more labels the algorithm will determine what to return by using a tie-breaker. Two typical tie-breaking algorithms are tie-breaking by selecting the closest tied training point or just by uniform random selection.

2.2 Minkowski Distance Function

The Minkowski distance function is a generalization of the Euclidean and Manhattan distances. It can be stated mathematically as

$$\left(\sum_{i=1}^{n} |x_i - y_i|^p\right)^{1/p}$$

By assigning $p = 1$ or $p = 2$ we can get the more special Manhattan and Euclidean cases, respectively. However, the naive MATLAB implementation of this code is inherently inefficient for these cases. We therefore implemented the polynomial expansion for $p = 1$ and $p = 2$.

Finally, the most intuitive of these distance functions is the Euclidean case. In practical terms it is an expansion of the Pythagorean theorem which yield the shortest path between points in Cartesian coordinate systems. We will therefore pay special attention to this distance function.

2.3 Similarity Function

2.3.1 Sampling

The function `randomSubset()` takes in data ($m \times n$), their labels ($m \times 1$), and a fraction that exists in $[0, 1]$ and randomly returns $fraction * m$ rows from data, along with the appropriate labels. Binornd, a built-in MATLAB function, is used to return *yes/no*, a ($m \times 1$) array. Each entry in *yes/no* is either a zero or one, with the probability of any number being a one equal to $fraction$. The function returns `data(logical(yes/no))` and `labels(logical(yes/no))`.

2.3.2 Learned Weights

The generalized similarity function is: $\sum_{i=1}^{p}\sum_{j=1}^{p} b_{ij}x_iy_j$, the implementation of which is straightforward (for performance considerations, see Optimizations). The interesting decisions come when generating b, the weights. As suggested in the assignment description, linear regression was used.

To learn weights, it is necessary to take two data points x_1 and x_2 from the training data, calculate $x_1^T \times x_2$, and associate a regression label (+1 or -1) with it depending on whether the labels of x_1 and x_2 are the same. Then, using linear regression, the weights b_{ij} are calculated. The first order of business is to split the training data into groups according to label so the regression label of +1 or -1 can be accurately assigned. This is achieved by the function `sortIntoBins()`, which returns a cell array of length 10, where $bins_i$ is the test data labeled as the digit $i-1$. The next question is how many different pairs of test data should be generated to feed to the linear regression model. In the case of this training set, there are $C(7291, 2) = 26,575,695$ possible pairs. However, $x_1^T \times x_2$, has $256^2 = 65,536$ entries, meaning 2GB of memory will only allow around 2,000 rows. Later, special cases of the generalized similarity function will be considered which reduce the dimensionality from 65,536 to 2,116 and 256, allowing for more rows in the regression model.

With 10 bins (labels), there are $\frac{10\times11}{2} = 55$ different bin-pairs from which to draw regression data. The function `generateLabels()` takes as parameters the output of `sortIntoBins()` as well as the integer *samplesperbinpair* and outputs W and *reglabels*. Two for loops run through each of the 55 bin-pair possibilities. For each bin pair i and j, a random subset of each bin is taken, $rsbin_i$ and $rsbin_j$. A matrix in which every element of $rsbin_i$ is paired with every element of $rsbin_j$ is generated. The cross product is computed for the two training examples present in every row. Finally, the first *samplesperbinpair* are taken from this list and appended to the W. If $i = j$, *samplesperbinpair* +1's are appended to the reglabels, -1 if $i\ neqj$. Using W and *reglabels*, b (the weights) can be computed by doing $b = W\ reglabels$. The b vector can then be passed to a distance function.

It is important to note that an equal number of data pairs are taken from each bin-pair. This does not preserve the proportionality of 'same-label' to 'different-label' pairs present in the full collection of pairs. Assume all bins have the same cardinality, BP. If $bin_i = bin_j$, there are $\frac{BP*(BP+1)}{2}$ unique pairs. If $bin_i \neq bin_j$, there are BP^2 unique pairs. Therefore, taking an equal number from each bin-pair over-represents the 'same-label' examples.

Another alternative method used for selecting pairs was to simply take the first *samplesperbinpair* from each bin and compute the `bin`$_i$`(r,:)' * bin`$_j$`(r,:)`, where r ranges from 1: *samplesperbinpair*.

2.3.3 Special Cases

As mentioned before, computing weights for all $65,536$ pixel pairings comes with a significant computational burden. Intuitively, one would expect that the relationship between every pair of pixels is not equally important. For example, the relationship between the top-left and bottom-right pixel is most likely of very little help in determining the true distance between two training examples. This idea should be captured by the b weights after sufficient training (e.g non-helpful pairs are weighted to 0), but perhaps human intelligence can guide which pixel relationships are important.

The first intuition is to only compare each pixels to itself. More formally, $b_{ij} = 0$ when $i \neq j$. b_{ij} when $i = j$ can be set to 1, or the weight can be learned using regression. This computation can be done by $x.{*}y$, point-wise multiplication, much more efficiently than the generalized computation.

Building on this idea, consider the case where each pixel in test datum x is compared to only its immediate neighborhood in test datum y. The relationship between these pixels should be important, because slight variations (one pixel in any direction) in the way a particular number was formed will be captured in the sum. Unfortunately, x'*y must still be computed. However, all 65,536 entries in the matrix do not need to be stored as we are only interested in those where i and j can be said to be neighboring. The function `neighborMask(m,n)` returns *nmask*, an (m*n X m*n) matrix, where $nmask_{ij} = 1$ iff i and j are neighbors. For example, $i = 1$ and $j = 2$ are clearly neighbors; j is one pixel to the right of i. In this case of a 256 dimensional array representing a 16x16 matrix, $i = 20$ and $j = (20 - 16) = 4$ are neighbors as well, j being the pixel directly one top of i. Masking x'*y with `neighborMask(16,16)` yields only 2,116 entries. While this does not save any time computationally, in fact increasing it due to the masking, it greatly increases the number of entries that can be stored in W and *reglabels*, which might yield more accurate b weights after regression.

Since both generalized and neighbor similarity both use different masks of x'*y, `ones(256,256)` and `neighborMask(16,16)` respectively, it makes sense to compute the two distances simultaneously, rather than computing x'*y once for the generalize version, then all over again for neighbor version:

```
cross = x'*y;}
distances_gen = sum(sum( cross(mask_gen) .* b_gen ));
distances_neighb = sum(sum( cross(mask_neighb) .* b_neighb ));
```

2.4 Tie-Breakers

2.4.1 Uniform Random Selection and Most Frequent Value

Intuitively we we could use the integrated MATLAB function `mode()` which returns the most frequent value among the k nearest-neighbours. Unfortunately, "*when there are multiple values occurring equally frequently, mode returns the smallest of those values*, which makes the classification task skewed toward smaller valued labels. To remedy this we select randomly among the labels with equal frequency instead.

2.4.2 Nearest Tied Training Point

The tie-breaking algorithm for nearest tied training point takes a vector with the k nearest-neighbours and corresponding distances. Firstly, we iterate over all possible labels, 0 to 9, counting labels and storing the results in a vector with same dimension as the number of digits. For every iteration we also keep track of the smallest distance found among each set of labels. These are stored in a vector of the same dimension as the label count. The final step is to again minimize over all the tied labels locally minimized distances.

2.5 Modifications to Original Sets

2.5.1 Dimensionality Reduction by Principal Component Analysis

The dimensionality of the original training set can be reduced by determining its principal components. The method used was proposed by Erkki Oja [1]. By using *Oja's Learning Rule*, we can perform a SVD and determine the singular values and principal components of the data set. This method is used due to the high-dimensionality of the data, since it simply *approximates* these values. To generate this new, reduced set, all but the first 25 dimensions are discarded (after applying the transformation): when using a test example, it is first mapped onto the new space using the first 25 principal components and then the standard k-Nearest Neighbors algorithm is executed.

To determine the amount of principal components that should be discarded, 256 new data sets are generated: the first one will contain the original training and testing sets after having been transformed using one principal component; for the second, two principal components will be used, and so on. The accuracy of the system using each data set is measured and the appropriate amount of dimensions are cut off.

2.5.2 Standardization of Data Set

Another data set has been created by standardizing the original set. To do so, each element $x_{i,j}$ has been transformed as follows:

$$z_{i,j} = \frac{x_{i,j} - \mu_j}{\sigma_j} \tag{1}$$

In the previous equation, j denotes the feature (in this case, $j \in [1, 256]$), i represents the example and μ_j and σ_j are the mean value and standard deviation of the j^{th} feature, respectively.

3 Design of Experiments

3.1 Similarity Function

3.1.1 Point-wise Unit-weights

The computationally simplest option, point-wise unit-weights, was chosen as a jumping off point. To investigate potential overfitting, the parameter TR is introduced, varying from 0.1 to 1. Next, the variable k, or the number of nearest neighbors to examine, is varied from 1 to 10. The inner-most loop iterates from 1 to 20, testing the accuracy of the other two parameters 20 times against a randomly chosen subset 0.05 the size of the test data. This inner-most loop was intuitively added to combat over-fitting in parameter selection; this was before cross-validation was known to be a possibility.

3.1.2 Point-wise Regressed-weights

To test point-wise with regressed weights, the TR parameter was removed. It was rather unsurprising, revealing as it did that using more test data improves accuracy. However, a new parameter, `samples per binpair`, was added. This parameter controls the size of the W matrix used in regression. The variable k was varied from 1 to 4 and `samples per binpair` from 100 to 400 in steps of 100. The two methods for generating pairs (random and in-order) were both used in regression. Additionally, L2 normalization was also tested on in-order pair generation regression.

3.1.3 Neighbor and Generalized Unit Weights

Next is generalized and neighbor unit-weights. As discussed in the 'Algorithms' section, it was more efficient to run both of these test simultaneously. k was varied from 1:5, and due to time/processing constraints only three test iterations were performed, each on a random samples of the test data 1/12 the size of the full test set.

3.1.4 Neighbor and Generalized Regressed Weights

The generalized and neighbor regressed-weights used a full training set and k=4. The only parameter being determined in this case was *samplesperbinpair*, varying from 500:500:2000. The *samplesperbinpair* for the generalized version were 50 times smaller than the neighbor version in order to not overflow memory. Only two test iterations were run. L2 regularization was also introduced where possible.

3.2 Accuracy Testing

Three tests were conducted using the PCA-reduced, the standardized and the original training sets. In each test, one parameter is varied while the others remain constant; the considered parameters are the amount of nearest neighbors which are computed, k, the value of p when computing the Minkowski distance and the number of training samples, T_r, which are used.

To determine the optimal value of k, the parameter is varied from 1 to 100, p from 1 to 30 and T_r from 500 to 7200 in steps of 100. The default values for these parameters (i.e., when they are not being swept) are $k = 10$, $p = 2$ and $T_r = 7291$ (the whole set). All experiments are performed 50 times and the average is computed. A sample of 100 test examples were used (due to time constraints), though the set was regenerated every time a parameter was varied and for every of the 50 iterations.

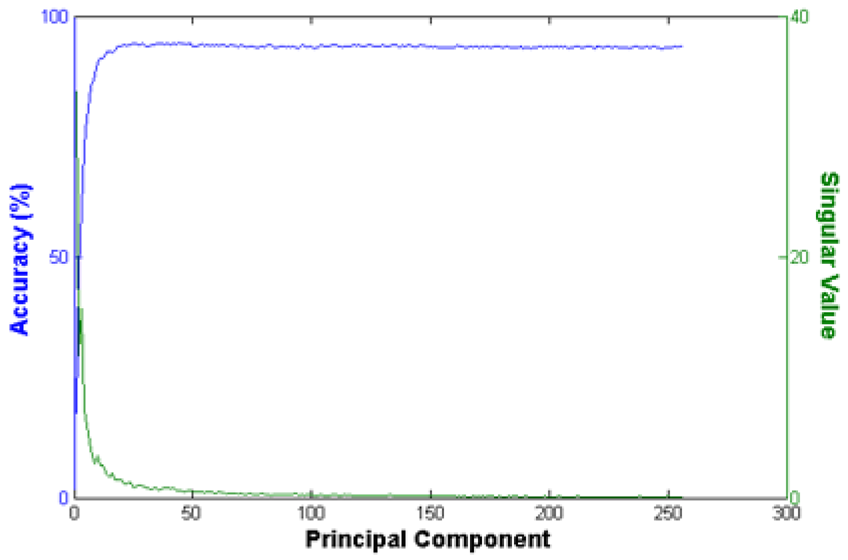

Figure 1: An accuracy of 94 percent is achieved with 25 principal components , which is equal to that achieved by the original data set. The whole set of testing examples is used, and the process repeated 50 times (the average is shown here). The singular values for each principal component are superimposed. Once 25 principal components or more have been used, the average accuracy of the system is 93.7 percent and has a variance of 0.067 percent.

4 Results of Experiments

4.1 Minkowski Distance Function

4.1.1 Reduction in Dimensionality

The accuracy of the system has been determined as a function of the amount of principal components used to encode the data. Refer to Figure 1 for the results. This measure of accuracy is in agreement with Figure **??**: the information content of the first 25 principal components is much larger than the rest. We can therefore discard these last 231 dimensions from the transformed set, thus exponentially increasing speed while essentially maintaining accuracy rate unchanged.

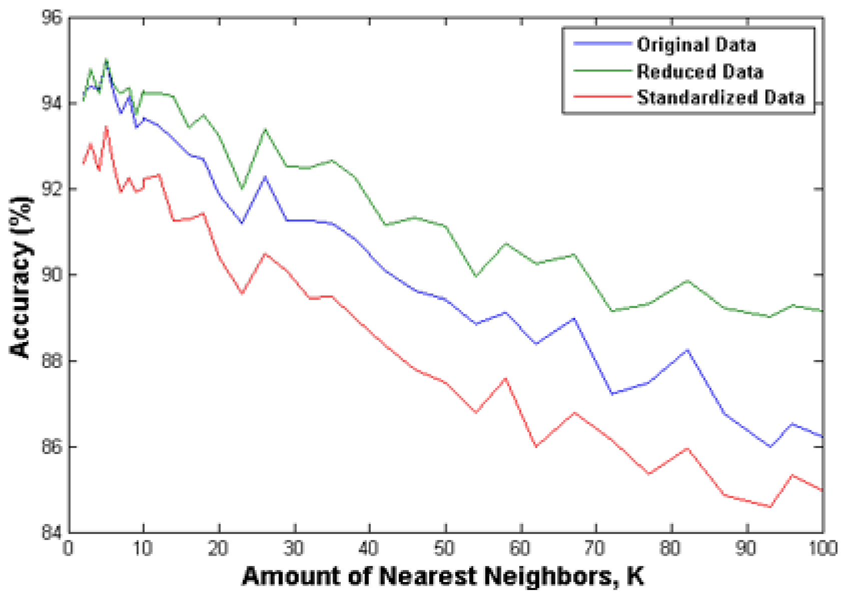

Figure 2: The k parameter was varied from 1 to 30 in steps of 2. Accuracy decreases as k increases.

4.1.2 Optimal Number of Nearest Neighbors

Figure 2 shows that as we increase the number of nearest neighbors past a value of 7, the accuracy of the system decreases linearly for all three data sets. This is most probably due to the fact that as we increase k, a series of data points which lie very far away from our test point are being considered. These distant digits do not contain relevant information and should therefore be ignored.

4.1.3 Optimal Number of Training Examples

When presenting a test example to the k-NN algorithm, correct classification is more probable as the number of training examples is increased. This can be seen in Figure 3, where the system accuracy increases logarithmically with number of training samples. The size of the training set is 7291, and experiments could not be conducted with a greater amount of samples. However, it is expected that the accuracy should decrease if too many training samples are considered. This phenomenon is called *overfitting* and should be avoided. This, however, is not the case and the whole training set should be used.

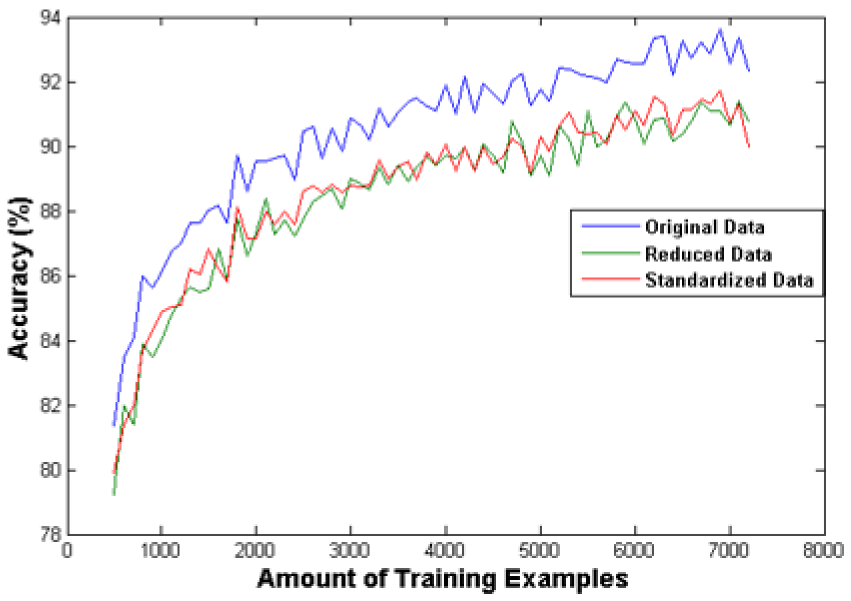

Figure 3: As expected, accuracy increases with the number of training samples.

4.1.4 Optimal p-Value for Minkowski Distance Function

Accuracy of the system does not vary with the p-Value used when implmeneting the Minkowski distance function with the original and reduced dimensionality data sets. However, as can be seen in Figure 4, the performance is severely compromised when using the standardized data set as p increases. This is due to the fact that standardization eliminates any information contained within the variance of the features (something that PCA emphasizes).

4.2 Similarity Functions

4.2.1 Point-wise Unit Weights

The maximum accuracy achieved by this approach is .9409 at a .95 subset of the training data with $k = 1$. Unsurprisingly, in Figure 1 there is a clear upward trend in accuracy as the percentage of training data used increases. For a 0.7-1.0 subset of the data, the optimal value of $k = 4$. Looking at Figure 5, it appears that k=4 has the smoothest gradient, which suggests its high average accuracy is not influence by any anomalous over-fit peaks.

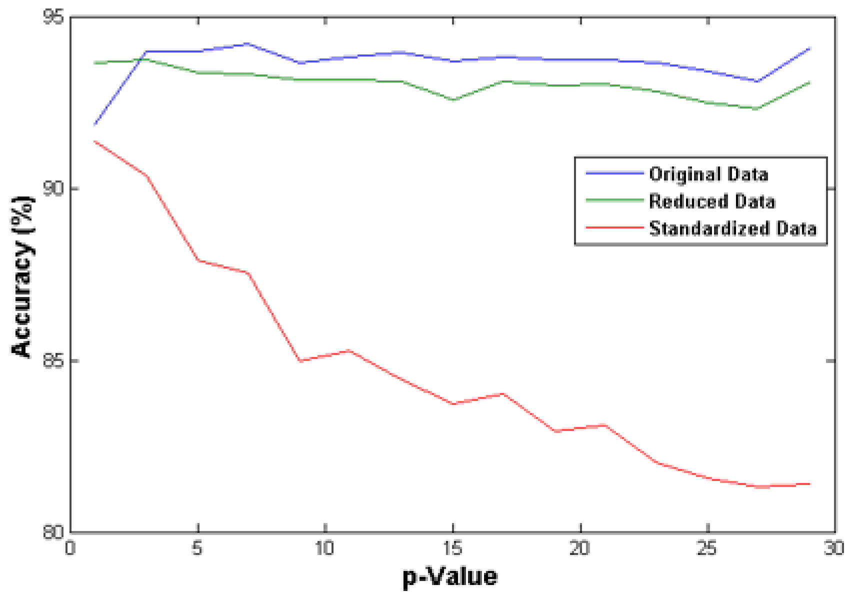

Figure 4: The p-value does not seem to have a large effect on reduced and original sets.

Though at 60% training data there is a peak at $k = 3$, this is most likely a case of over-fitting rather than a true, general-purpose optimal k value. For such a simple point-wise approach, the accuracy is surprising, especially considering the abundance of negative values in the data that should tend to confuse the dot product.

Using the reduced-dimensionality dataset yielded poor accuracy (Figure 6) so this approach was abandoned in further experiments.

4.2.2 Point-weise Regressed Weights

Here the training fraction parameter was dropped in lieu of the *samplesperbinpair* parameter, which controls how big the W matrix is for generating weights. As discussed in the 'Algorithms' section, there were two ways of selecting training pairs. It was though that by randomizing the pair generation by using `randomSubset()` that the resultant b weights would offer a better solution than simply taking the first *samplesperbinpair* rows from each bin. This turned out to be incorrect, however. When using the in-order version, when $bin_i = bin_j$, each training example gets paired with itself. This seems to have

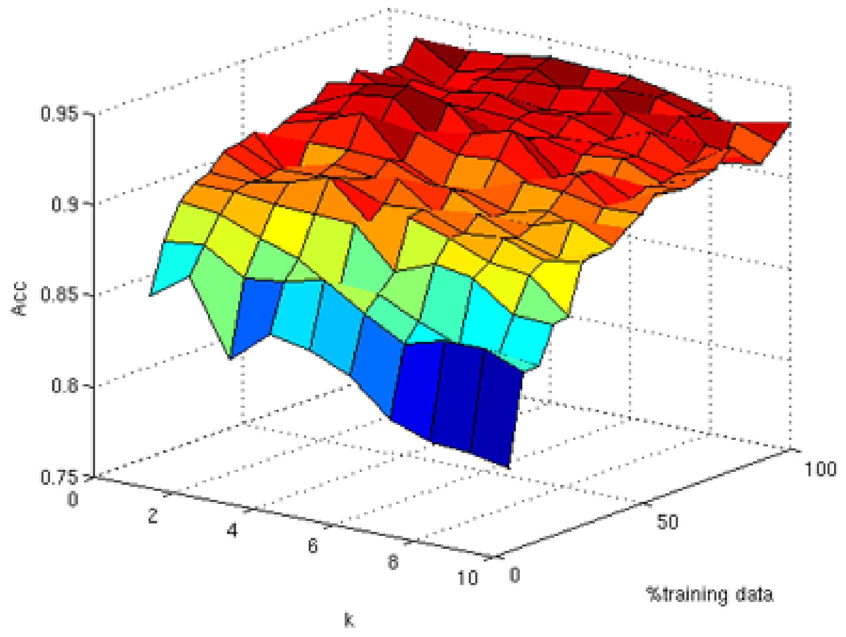

Figure 5: Accuracy of point-wise similarity distance, unit weights.

unilaterally increased accuracy, as can be seen from comparing Figures 7 and 8.

One would also expect the regressed weights to offer better accuracy than the unit weights, because if the unit weights are optimal the regression should find them. However, this assumes two things: 1) There is a linear relationship in the data and 2) the W matrix is large enough to be representative of all the possible pairs. In the test cases above, $samplesperbinpair$ peaked at 400, meaning there were $55 * 400 = 22,000$ pairs vs $7291 * 7290/2 = 26,575,695$, only .08%.

In-order pair generation and with L2 regularization with $lambda = 0.000001$ yielded the best results (Figure 9). On a full data set with k=1:4, this variation of regressed weights performed, on average, 0.5% better than the unit weights.

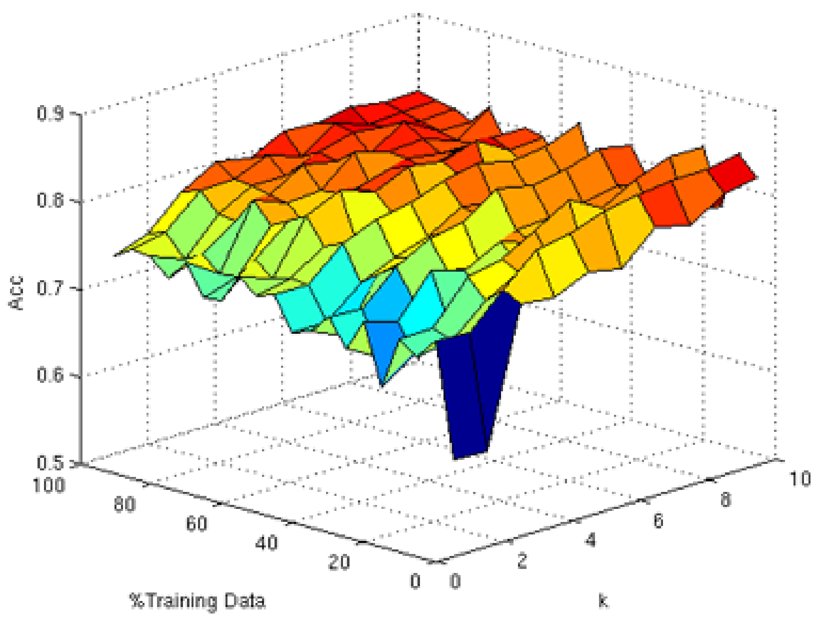

Figure 6: Accuracy of point-wise similarity distance, unit weights using reduced data set $p = 25$.

4.2.3 Generalized and Neighbor Unit Weights

It was a little surprising to see how poorly the generalized similarity distance function fared (Figure 10), but given the intuition that most of the ij pairs will bear little to no relation to one another perhaps this should not be such a shock. These ij pairs whose relationship to one another convey no useful information (e.g the bottom middle and top left pixel) serve only to convolute the sum.

On the other hand, the neighbor masking of the all the ij pairs performed admirably (Figure 11). Using the full dataset, $k = 1 : 5$, neighbor unit weights performed 1.7% better on average than the point-wise unit weight version.

4.2.4 Generalized and Neighbor Regressed Weights

All accuracies in this section are based on the full dataset, $k = 4$, and two iterations testing a 1/15-sized random subset of the test data.

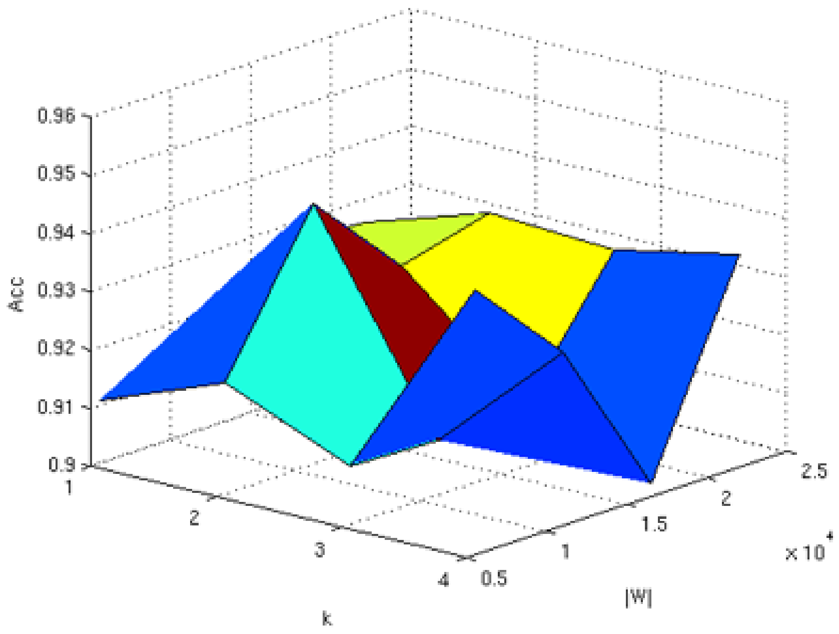

Figure 7: Accuracy of point-wise similarity distance, regressed weights. Random pair generation.

- Accuracy of generalized regressed weights without L2 regularization: 0.30
- Accuracy of neighbor regressed weights without L2 regularization: 0.16
- Accuracy of neighbor regressed weights with L2 regularization: 0.66

t was expected that the regression would yield weights that filtered out the ij pairs that contributed no useful information. This expectation was met, with around 98% of labels being equal to zero. However, the non-zero b weights generated had unexpectedly large deviations from 0, typically around the +-30 mark. Unfortunately, MATLAB refused the attempt at bringing these values closer to 0 through L2 regularization. The code:

```
[xty_g; lambda*eye(256^2)] \ [reglabels_g; zeros(256^2,1)];
```

particularly the `eye`(256^2) portion, was refused with a: `Maximum variable size allowed by the program is exceeded.`

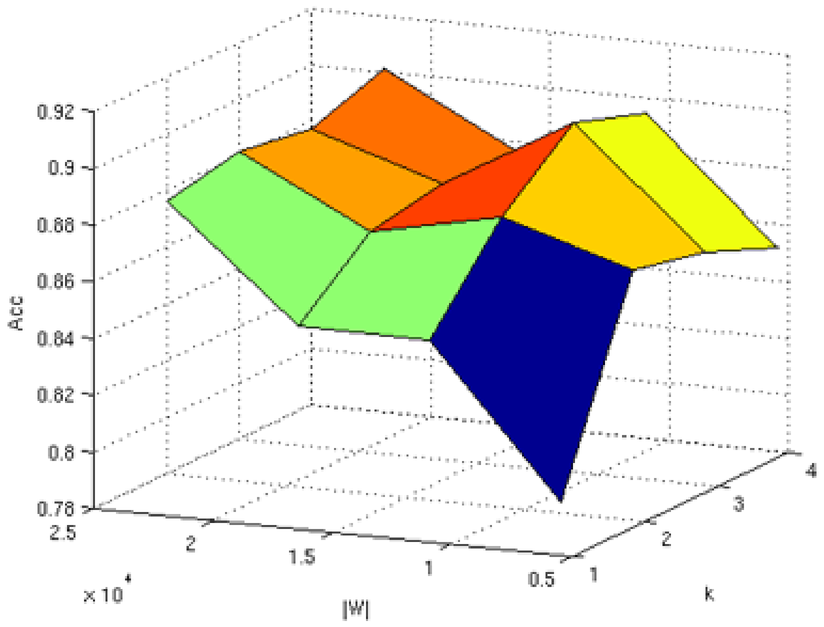

Figure 8: Accuracy of point-wise similarity distance, regressed weights. In-order pair generation.

Also frustrating were the weighted neighborhood b values and, consequently, accuracy. Here regularization could be applied, yielding an accuracy of .6643. Perhaps linear regression is not suited for this problem.

5 Findings and Lessons Learned

5.1 Similarity Function Results

The generalized similarity function performed poorly. Even under regression, the weights that were generated did not capture the point-wise or neighbor intuitions that led to higher accuracy. L2 regularization helped, but accuracy was still far below the unit weighted version. Regression only gave slight improvement in the point-wise case, suggesting that there is no linear relationship to find in the more complicated generalized and neighbor cases. It is also possible there is a mistake in the code for generalized W construction, which was separate from the more helpful point-wise W construction.

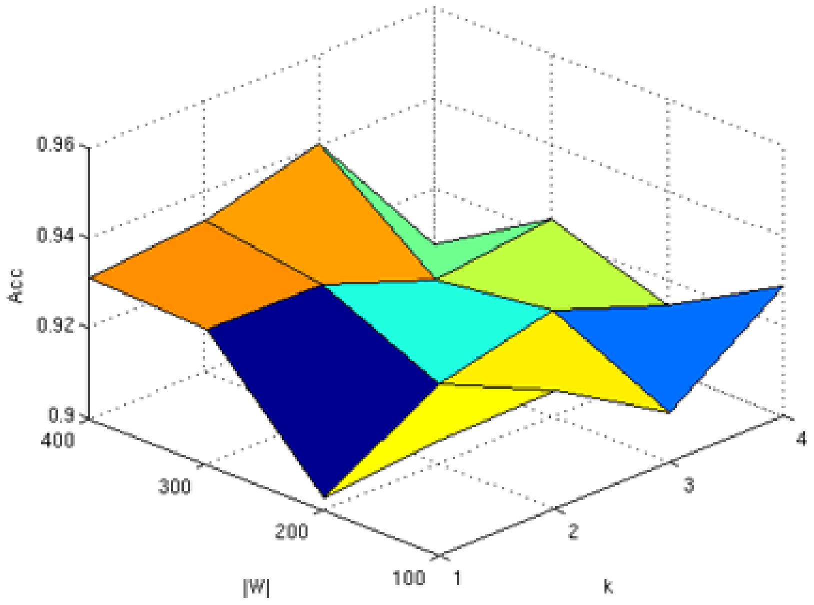

Figure 9: Accuracy of point-wise similarity distance, regressed weights. In-order pair generation with L2 regularization.

Finally, given that the accuracy of the point-wise unit weight case was on par with the best Euclidean distance, the huge computational overhead involved with the generalized or neighbor cases is complete unnecessary.

5.2 Minkowski Function Results

When using a standard Euclidean distance function with a reduced dimensionality set, accuracy ranges from 93 to 95 percent, while significantly increasing speed. When standardizing the data set, a lot of information is lost. The principal component analysis of the set retains this information and exaggerates variances within a feature, a characteristic which in general holds the most amount of information.

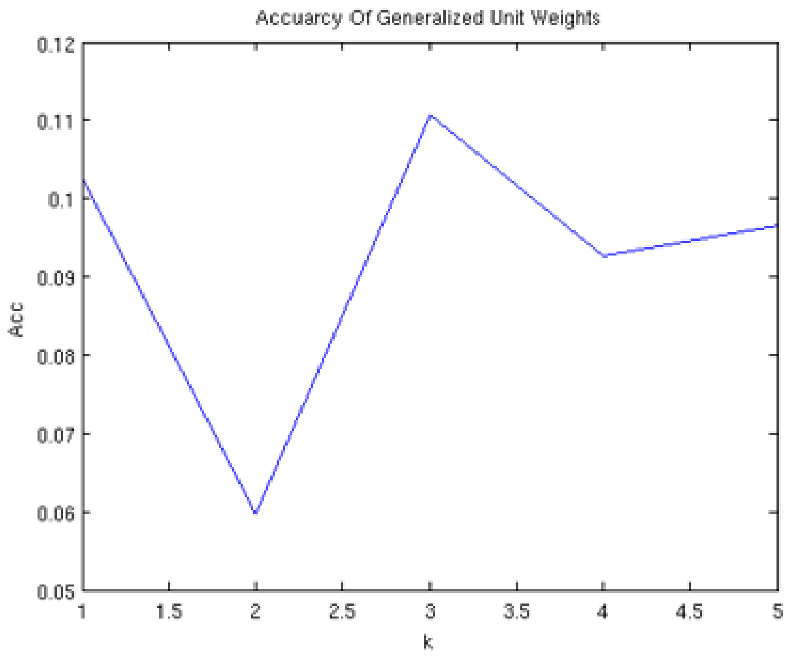

Figure 10: Accuracy of generalized unit weights.

5.3 Comparison to Literature

Abraham, Jain and Von der Zwaag achived accuracies on the full USPS data set with $k = 5$ of up to 94.469 percent [2]. For other classifiers V. Vapnik declares varying accuracies. For a decision tree classifier he got 83.8 percent and for a two-layer neural network he achieved 94.1 percent accuracy. Also, he states an interesting benchmark, human performance, yielding 97.5 percent [3].

6 Limitations

- Cross validation was not employed, testing was done by repeatedly taking random samples of the test data. This does not guarantee that all testing and training data is used to accurately evaluate the algorithms.
- Assigning labels of 0 and 1, rather than -1 and 1, produced more accurate regression weights. Why this occurs and what variations or

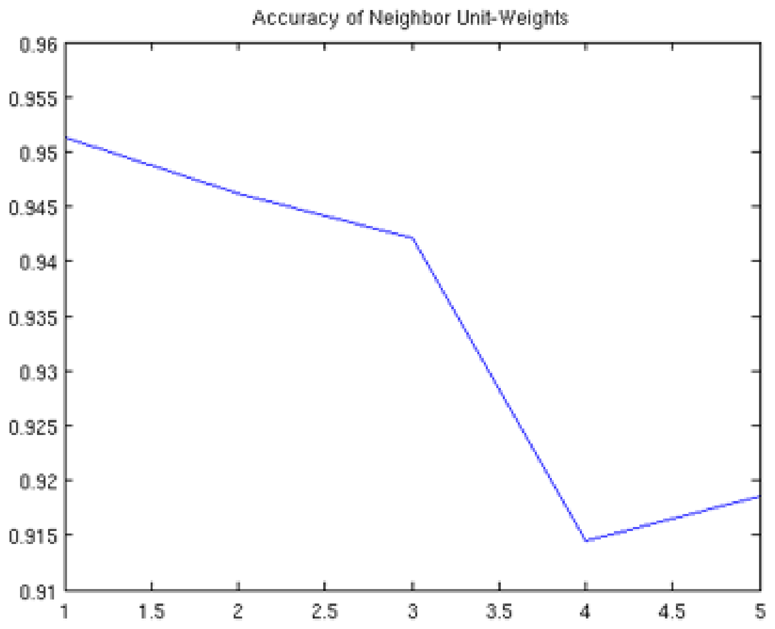

Figure 11: Accuracy of neighbor unit weights.

improvements could be made were not explored.

- Quantitative timing data is almost completely absent from the analysis.
- A confusion matrix might have suggested beneficial tweaks to the used algorithms. For example, if it was seen that 4 and 9 were being confused disproportionately, more (4,9) pairs could be added to W, hopefully increasing the regressions sensitivity to this more difficult distinction.
- Seeking out a machine with more memory would have helped generate bigger W matrices. As mention, only 0.08% of all possible pairs were used, which could very well account for the poor performance of generalized regression.
- No sensitivity analysis in were computed between tie-breaking functions.
- Our kNN results both weighted and unweighted are on par with AJZ and V. Vapniks two-layer neural network, but fail to achieve human

standards landing 3 to 4 percentage points short.

References

[1] Oja E (1982) *A simplified neuron model as a principal component analyzer.* J Math Biol 15(3):267273

[2] Ajith Abraham, L. C. Jain, Berend J. Von der Zwaag, Innovations in intelligent systems, 2004

[3] V. Vapnik, The Nature of Statistical Learning Theory, Springer-Verlag, 1995.