Chapter 5

Evaluation

This chapter contains the evaluation of the methods proposed in the previous part, chapter 4. In section 5.1, we present the data sets and the methodology used for testing. We give details about the parameter choices and set baseline scores obtains by either classical NCA or simple linear projections, such as PCA, LDA or RCA. Results for each individual method are presented in sections 5.3 and 5.4. A comparison of the methods is shown in subsection 5.3.4 using accuracy versus time plots.

We should mention that we did not include all the results in this chapter to prevent cluttering. Further experimentations can be found in appendix B.

5.1 Evaluation setup

The evaluation was done in terms of two metrics: accuracy and speed. An additional and more subjective criterion is to judge a method by visualizing low dimensional representations of various data sets. We provided 2D plots of the projected data where suitable.

The data sets selected for testing are listed in table 5.1. We note that the used data vary both as number of samples N and as dimensionality D. Even if we concentrate on large amounts of data, we need small data sets to assess the performance of the new models. The methods' speed was tested on the large data sets (`usps`, `magic` and `mnist`). However, the diversity in size and complexity made it difficult to find the optimal selection of parameters. We are aware that there is "no free lunch" in accurately solving widely different problems with a fixed model. When concentrating on a single task it is often advised to include

Data set name	Abbrevation	N	D	C
Balance scale	`balance`	625	4	3
Ecoli	`ecoli`	336	7	8
Glass identification	`glass`	214	9	6
Ionosphere	`ionosphere`	351	33	2
Iris	`iris`	150	4	3
Landsat satellite	`landsat`	6435	36	6
MAGIC Gamma telescope	`magic`	19020	10	2
MNIST digits	`mnist`	70000	784	10
Pima Indians diabetes	`pima`	768	8	2
Image segmentation	`segment`	2310	18	7
SPECTF heart	`spectf`	267	44	2
Blood transfusion	`transfusion`	748	4	2
USPS digits	`usps`	11000	256	10
Wine	`wine`	178	13	3
Yeast	`yeast`	1484	8	10

Table 5.1: This table presents the characteristics of the data sets used: number of samples N, dimensionality of the data D and number of classes C. The two digits data sets `mnist` and `usps` were downloaded from the following URL `http://cs.nyu.edu/~roweis/data.html`. All the others data sets are available in the UCI repository `http://archive.ics.uci.edu/ml/datasets.html`.

prior knowledge into the model. Also it is easier to make minor tweaks of the parameters to boost the performance.

For evaluation we used 70% of the data set for training and the rest of 30% was kept for testing. We made exceptions for two data sets: `landsat` and `mnist`. These are commonly already split in training and testing sets. We report standard errors for the mean accuracy averaged over different splits.

The methods we test are: sub-sampling (SS; section 4.1), mini-batches (MB; section 4.2), stochastic learning (SL; section 4.3). For stochastic learning we included the two approximated methods: the fast kernel density estimation idea (SL-KDE; section 4.4) and compact support version of NCA (SL-CS; section 4.5). Each method has particular parameters that we discuss in its corresponding subsection. There are some common parameters for all the methods. These choices are related to the NCA implementation and, for convenience, we remind them here. We experimented with three optimization methods: gradient ascent with "bold driver" heuristic, conjugate gradients and variants of stochastic gradient ascent with early stopping. For initialization we used the techniques described in subsection 3.3.2: random initialization, PCA, LDA or RCA. At test time, we did classification using 1-NN or using an NCA based function as described in subsection 3.3.5. However, we usually present scores using both classification rules.

The experiments were carried in MATLAB and most of the implementations are the authors' own work. There are some exceptions however. We used Carl E. Rasmussen's `minimize.m` for conjugate gradient optimization.[1] The RCA implementation was provided by Noam Shental on his web-page.[2] Also we used functions from Iain Murray's MATLAB toolbox.[3] Finally, we inspected previous implementations of NCA,[4] even if we did not explicitly made use of them.

5.2 Baseline

We started by implementing the standard NCA algorithm (appendix A.1). This consists the main baseline against which we compare new models. For our first series of experiments, we tried to replicate the work in the original article (Goldberger et al., 2004). We encountered some difficulties since no information about their implementation was provided in the paper. Our results are presented in table 5.2 (scores are averaged over 40 runs). We randomly initialized the matrix $\mathbf{A}$ and optimized it using conjugate gradients (CG) method. This was the easiest thing to do since no parameter tuning is need for CG. We note that the results are similar to those of Goldberger et al. (2004); for `ionosphere` data set we obtained slightly worse results.

However, in order to achieve a robust implementation of NCA we had to carry out additional experiments. The learnt lessons were summarized in section 3.3.

[1]This is available for download at <`http://www.gaussianprocess.org/gpml/code/matlab/util/minimize.m`>.

[2]The code was downloaded from <`http://www.openu.ac.il/home/shental/`>.

[3]Iain Murray's toolbox is available at <`http://homepages.inf.ed.ac.uk/imurray2/code/imurray-matlab/`>.

[4]Implementations of NCA are provided by Laurens van der Maaten in his MATLAB Toolbox for Dimensionality Reduction <`http://homepage.tudelft.nl/19j49/Matlab_Toolbox_for_Dimensionality_Reduction.html`> and by Charless C. Fowlkes on his website <`http://www.ics.uci.edu/~fowlkes/software/nca/`>.

		Train score	Test scores		Baseline
Data set	d	$f(\mathbf{A})$	1-NN	NCA	Eucl.
`balance`	2	92.86 ± 0.47	90.78 ± 0.53	90.61 ± 0.55	
	$D = 4$	95.36 ± 0.38	93.40 ± 0.47	93.04 ± 0.51	76.18
`ionosphere`	2	98.31 ± 0.14	79.86 ± 0.75	79.74 ± 0.78	
	$D = 33$	72.07 ± 0.71	86.22 ± 0.64	72.87 ± 0.71	85.38
`iris`	2	99.38 ± 0.11	94.94 ± 0.39	94.72 ± 0.39	
	$D = 4$	99.48 ± 0.10	95.10 ± 0.44	95.15 ± 0.44	95.53
`wine`	2	99.15 ± 0.14	92.4 ± 1.0	92.4 ± 1.0	
	$D = 13$	98.95 ± 0.15	95.36 ± 0.51	95.36 ± 0.51	74.53

Table 5.2: Accuracy of standard NCA on four small data sets. Scores are averaged over 40 runs. The second column presents the dimensionality d the data set is reduced to. The last column shows the leave one out cross validation performance on the data set using Euclidean metric.

Data set	PCA	LDA	RCA
`usps`	73.47 ± 0.13	87.44 ± 0.12	87.42 ± 0.13
`magic`	77.097 ± 0.080	76.17 ± 0.29	77.574 ± 0.078
`mnist`	70.13	9.96	79.11

Table 5.3: Accuracy of three linear transformation techniques applied on the large data sets. We used 1-NN for classification. Scores are averaged over 20 runs, except for `mnist` data set. We reduced each data set dimensionality to $d = 5$.

We will further see the influence of the implementation tricks in the next part (section 5.3). Results were shown in section 3.3 and are also attached at the end of the thesis, appendix B:

- Tables B.3 and B.4 compare two of the optimization methods: conjugate gradients and gradient ascent with "bold driver" heuristic. We observe that the two methods give close scores on most of data sets. However, the gradient ascent takes longer until it reaches convergence.

- Figures B.1, B.2 and B.3 illustrate the initialization effect on three data sets: `iris`, `balance` and `ecoli`. RCA seems to be the best option for initialization and we used it in most of our comparisons. However, random projection can also be sometimes surprisingly good as we see in figure B.1(a).

We could not apply NCA on large data sets (`usps`, `magic`, and `mnist`) since it would have taken far too long. We decided to use as a baseline the linear transformations that we also use for initialization (PCA, LDA and RCA). The results are averaged over different splits of training and testing set and they are listed in table 5.3. For `mnist` data set we have scores of classic NCA (Singh-Miller, 2010). For $d = 5$ the accuracy obtained was 91.1% using kNN classifier and 90.9% using NCA classifier. For the classification procedure Singh-Miller learnt the optimal value for k using cross-validation.

5.3 Mini-batches methods

The mini-batches methods are compared on the larger data sets: `usps`, `magic`, and `mnist`. We chosen to reduce the dimensionality to $d = 5$. The decision is partially motivated by the fact NCA is very effective for reducing the data dimensionality to small values, somewhere around 5 to 15. A more principled approach would have been to develop a model choosing algorithm: start with a projection to $d = 2$ dimensions and then increase the number of dimensions until the score stops improving. The idea is similar to the "early stopping" procedure and it is illustrated for `landsat` data set in figure 5.1.

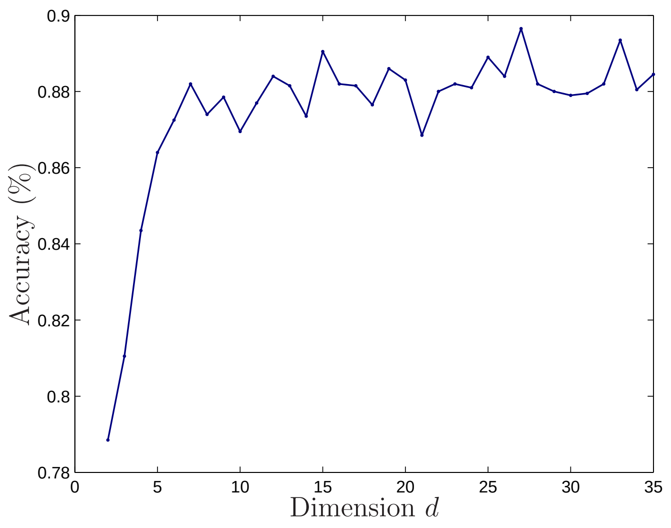


Figure 5.1: Evolution of test accuracy as dimensionality increases on `landsat` data set. We see that NCA operates almost as well in low dimensions ($d = 6, \cdots, 10$) as in high dimensions ($d > 25$). This approach can be used for selecting a suitable dimension to which we project the data.

Data set	Train	1-NN	NCA
`usps`	99.112 ± 0.099	89.30 ± 0.18	89.41 ± 0.16
`magic`	82.71 ± 0.41	79.25 ± 0.45	80.22 ± 0.72
`mnist`	96.867	82.130	82.470

Table 5.4: Accuracy scores for SS method on the larger data sets. We used RCA for initialization and CGs for optimization. We used a subset of $n = 3000$ data points for training and the whole data set for testing.

5.3.1 Sub-sampling

For sub-sampling we trained NCA on a subset of $n = 3000$ samples of the original data. We used conjugate gradients for optimization and RCA linear transformation for initialization. As previously mentioned in section 4.1, the sub-sampled data has a thinner distribution than the original data which helps the method to obtain good scores at training. But the test performance is hindered because we do not use the true distribution. This is especially evident for the digits data `usps` and `mnist` (table 5.3.1).

Data set	Train	1-NN	NCA
`usps`	92.00 ± 0.43	91.26 ± 0.17	92.37 ± 0.17
`magic`	80.04 ± 0.48	79.14 ± 0.52	79.8 ± 1.1
`mnist`	87.47 ± 0.49	87.52 ± 0.36	89.88 ± 0.25

Table 5.5: Accuracy scores for MB method on the larger data sets. We used RCA for initialization and the mini-batches were clustered in the low-dimensional space using RPC. The size of a mini-batch was of maximum $n = 2000$ data points.

5.3.2 Mini-batches

We trained NCA using the gradient ascent variant with clustered mini-batches (section 4.2). For the learning rate, we used an update rule of the form $\frac{\eta}{t+t_0}$. We fixed $\eta = 1$ and tuned the other free parameter t_0 using cross validation across an exponential scale from 0.1 to 1000. After that, a finer tuning was done on a linear scale around the best value of t_0. We used 5% of the training set for cross validation to monitor the accuracy score at each iteration. If the performance on the cross validation set does not increase for 25 iterations, we stop the learning process and return to the previously best parameter.

To get significant gradients we used large mini-batches $n = 2000$. The points in a batch are selected via recursive projection clustering (RPC) algorithm. The clustering was done in the low dimensional space, after projecting the points with the current linear transformation matrix $\mathbf{A}$. The results obtained by mini-batches method can be found in table 5.3.2. We note similar scores on `magic` and better results on the other two data sets.

5.3.3 Stochastic learning

This method was trained using the variant of stochastic gradient ascent presented in section 4.3. We used the same parameters as in the previous section. We considered $n = 50$ neighbours to look at for each iteration and computed their contributions with respect to the whole data set.

Besides the results on the large data sets (table 5.3.3), we also present the performance of this method when used for small data sets (table 5.3.3). We note a considerable improvement on the `magic` data set compared to the previous two methods. For small data sets, we observe similar results as the baseline NCA.

		Train score	Test scores	
Data set	d	$f(A)$	1-NN	NCA
`balance`	2	88.35 ± 0.83	87.37 ± 0.49	90.45 ± 0.38
	$D = 4$	94.70 ± 0.87	95.32 ± 0.34	96.14 ± 0.29
`ionosphere`	2	89.0 ± 1.7	85.71 ± 0.94	87.08 ± 0.95
	$D = 33$	92.6 ± 1.5	84.72 ± 0.57	84.34 ± 0.59
`iris`	2	96.41 ± 0.94	96.33 ± 0.57	97.00 ± 0.46
	$D = 4$	97.5 ± 1.3	95.67 ± 0.71	96.11 ± 0.66
`wine`	2	98.80 ± 0.70	97.22 ± 0.65	97.50 ± 0.49
	$D = 13$	99.25 ± 0.62	96.85 ± 0.41	96.85 ± 0.41

Table 5.6: Accuracy scores for SL method on the small data sets. We used RCA for initialization. The scores are averaged after 20 iterations.

Data set	Train	1-NN	NCA
`usps`	90.23 ± 0.50	90.68 ± 0.22	92.64 ± 0.17
`magic`	78.39 ± 0.25	79.76 ± 0.13	84.49 ± 0.12
`mnist`	85.97 ± 0.37	86.07 ± 0.43	89.35 ± 0.39

Table 5.7: Accuracy scores for SS method on the larger data sets. We used RCA for initialization. At each iteration we consider $n = 50$ data points.

The most important remark is that the classification done using NCA objective function is better than 1-NN classification. This observation also applies for the large data sets results. More results for this method are in appendix, tables B.1 and B.2.

5.3.4 Comparison

To easily compare the 3 presented methods, we provide time-accuracy plots (figure 5.2). We did not compute time for smaller data sets, since the classical NCA was usually fast enough. We also included in the plots the method based on the compact support kernels (we discuss its individual scores in subsection 5.4.2).

In terms of accuracy, all of the NCA variants improve the accuracy over PCA, LDA or RCA. SS gives comparable results to the other two only on `magic` data set. This data set has only two classes and a sub-sampled data set does not remove too much information of the decision boundaries. MB and SL are similar in accuracy. We note the proposed methods obtain a slightly worse score than classic NCA on `mnist`. The performance reported in (Singh-Miller, 2010) was around 91%, while the mean accuracy for MB and SL is about 90%.

The differences in time varied strongly. We used a logarithmic scale on the time to better discriminate the plotted scores. For each method we show the equidensity contours of the Gaussian distribution. The contours are ellipses, but because of the logarithmic axis they look deformed.

The time spent varies even for the same method because the number of iterations until convergence depends on random parameters. There is a certain trade-off between time and accuracy, especially for the digits data sets.

We show also low dimensional representations of data for $d = 2$ for two of the data sets: `usps` (figure 5.3) and `magic` (figure 5.4).

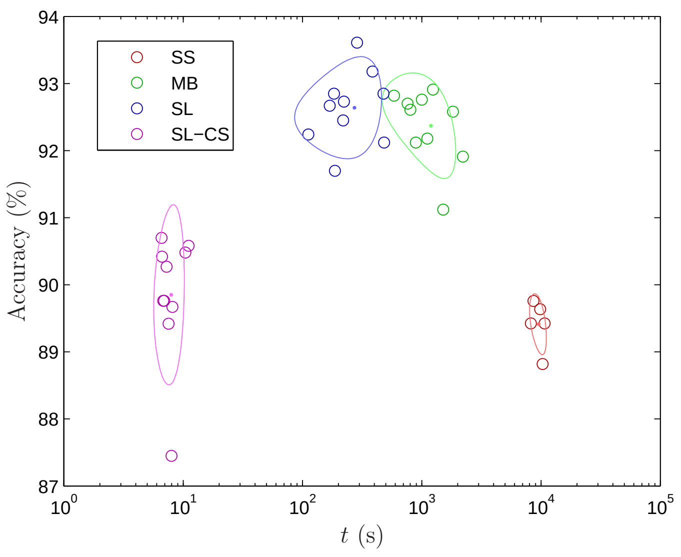
5.4 Approximate computations

The following methods can be applied on classic NCA, but we tested them in conjunction with the stochastic learning procedure to further boost the speed.

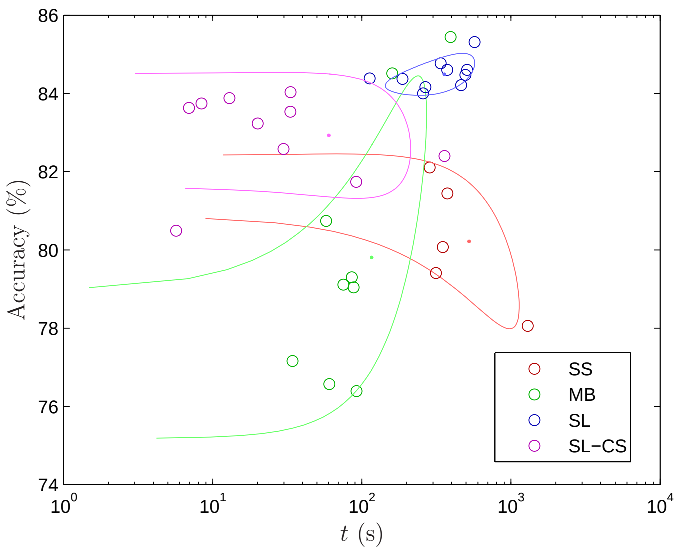
For approximate computations we also tried a more simplistic approach, similar to that of Weinberger and Tesauro (2007): we selected only the first 100 neighbours for each point and discarded those points whose contribution p_{ij} is less than $\exp(-30)$. This simple approach gave surprinsingly good score, although we do not present them here.

5.4.1 NCA with k-d trees

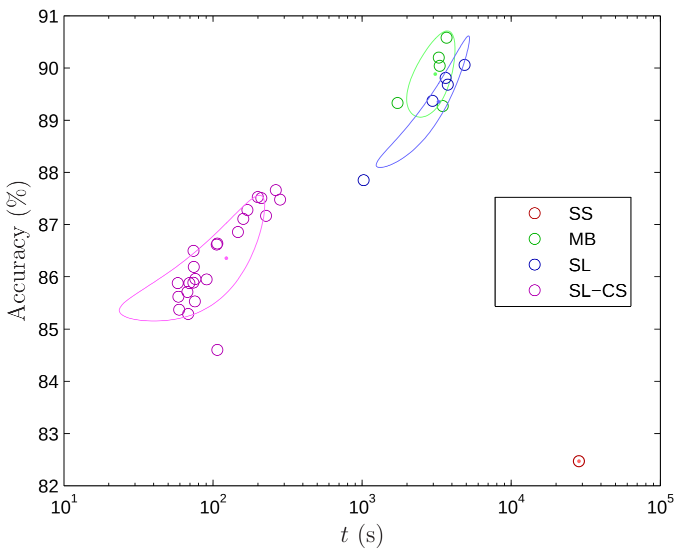
For the class-conditional kernel density estimation idea we used the algorithm described in section 4.4. Our k-d tree implementation was done in Matlab and, for this reason, it was pretty slow. The code was slower than the simple SL version, because the we cannot vectorize the recursive computation of the objective function and its gradient. We experimented with kernel density code written in C and mex-ed in Matlab and we think that such an approach can improve the speed. Doing simple kernel density estimation with a C written code



(a) usps

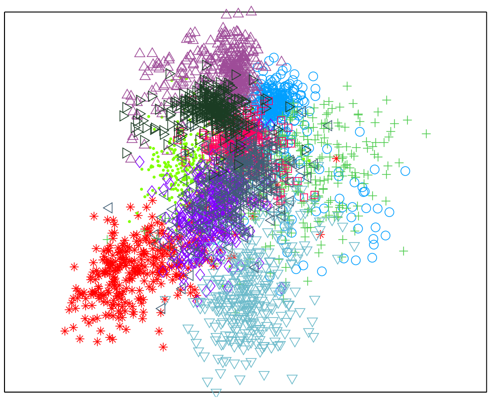


(b) magic

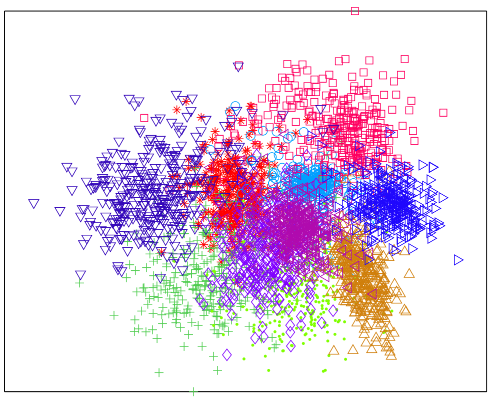


(c) mnist

Figure 5.2: Time *vs.* accuracy plots on larger data sets for four of the proposed methods. For SS we plotted the 1-NN score, while for the other three the points inidicate the NCA score.

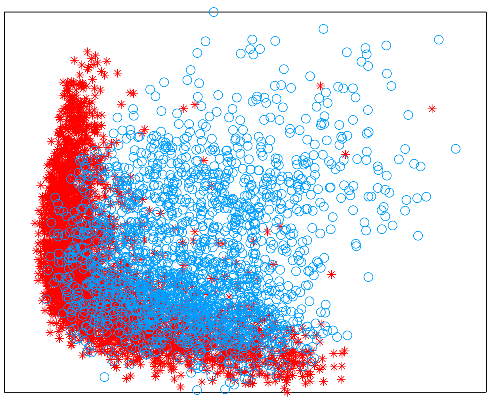


(a) NCA MB

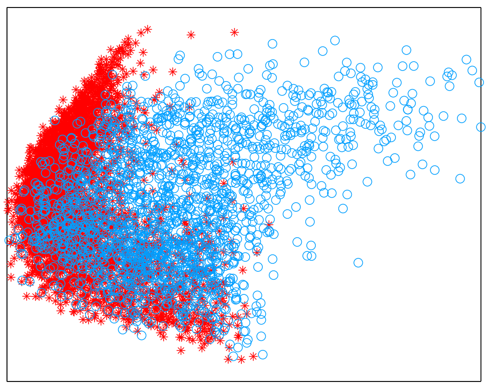


(b) NCA SL

Figure 5.3: Two dimensional projections of `usps` data set using two variants of NCA learning. The linear transformation was learnt on a training set, and here is plotted the projection of a testing set.



(a) NCA MB



(b) NCA SL

Figure 5.4: Two dimensional projections of `magic` data set using two variants of NCA learning. The linear transformation was learnt on a training set, and here is plotted the projection of a testing set.

$\epsilon_{\max}$	Train	1-NN	NCA	Visited points
0	87.05 ± 0.66	86.30 ± 0.32	87.87 ± 0.24	80.0 ± 5.9
10^{-50}	87.50 ± 0.31	86.26 ± 0.24	87.88 ± 0.20	55.1 ± 5.2
10^{-20}	85.49 ± 0.89	86.32 ± 0.28	87.87 ± 0.26	45.7 ± 4.1
10^{-5}	87.29 ± 0.49	85.95 ± 0.19	87.63 ± 0.29	22.8 ± 2.7
0.01	87.34 ± 0.76	86.14 ± 0.29	87.90 ± 0.24	22.1 ± 4.2
0.1	86.70 ± 0.39	86.12 ± 0.29	88.09 ± 0.19	20.0 ± 2.0

Table 5.8: NCA SL + k-d trees on `landsat`. $\epsilon_{\max}$ denotes the maximum error that we accept while approximating the density for a point given a class $p(\mathbf{x}|c)$. Visited points indicates the fraction of points that are used for computing the function and the gradient.

Data set	Train	1-NN	NCA
`usps`	89.68 ± 0.51	90.57 ± 0.20	92.67 ± 0.15
`magic`	78.09 ± 0.37	79.68 ± 0.30	84.50 ± 0.21
`mnist`	84.4 ± 1.4	85.5 ± 1.0	88.98 ± 0.75

Table 5.9: NCA SL + k-d trees on large data sets. For these experiments we set $\epsilon_{\max} = 0.1$.

proved to be even faster than doing it in MATLAB. The short time did not permit us to finalize this approach. We demonstrate that SL with k-d trees achieves good results even if we discard a large number of the points. Table 5.9 shows results for this method on the large data sets. We also present how the accuracy and the average number of prunings depend on the maximum error imposed (table 5.8).

5.4.2 NCA with compact support kernels

The compact support version of NCA is easy to implement and it is very fast as we already saw in section 5.3.4. Usually only a small fraction of the points is inspected. In figure 5.5, we see how this fraction evolves with the learning process. In the first iterations there might be a larger number of points, but soon this drops off and stabilizes. It might be tempting to start with all the points very close together to ensure that no point is outside the compact support of any

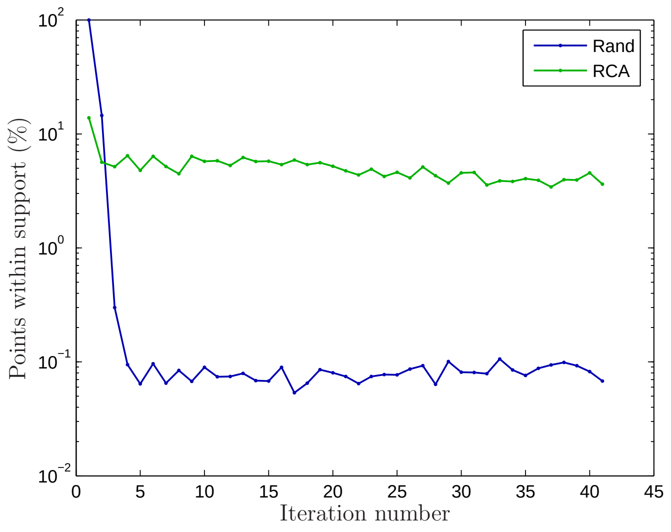
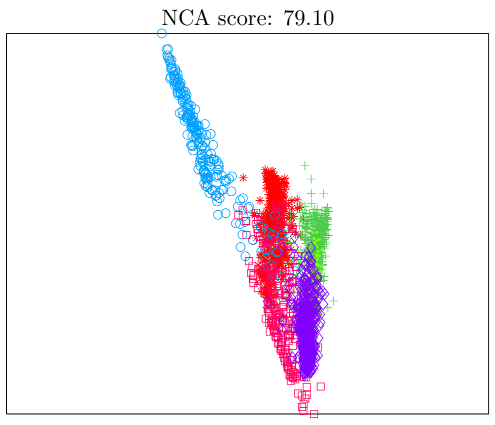
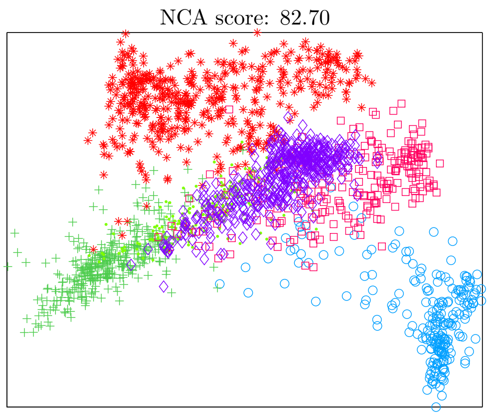


Figure 5.5: This figure illustrates how the fraction of the points inspected varies during the learning procedure. When we use random initialization, there are inspected only 0.1% of the points. If RCA is used to initialize the learning algorithm a fraction of about 10% is used.



(a) NCA SL CS randomly initialized



(b) NCA SL CS with RCA initialization

Figure 5.6: Illustration of final projections using two different initializations for NCA SL CS.

Data set	Train	1-NN	NCA
`usps`	85.16 ± 0.33	87.02 ± 0.44	89.85 ± 0.30
`magic`	76.92 ± 0.56	79.09 ± 0.42	82.92 ± 0.36
`mnist`	80.55 ± 0.28	81.96 ± 0.26	86.36 ± 0.18

Table 5.10: Accuracy scores for NCA SL CS method on the larger data sets.

other point. This approach has however two drawbacks. Since all the points are in the support of all the other points it means the first iteration will not present any gain in speed. In fact, the first iteration can be easily as expensive as the whole learning process. A second issue is that the gradients will be very large and might "throw away" points out of the existing support. More reliable is an initialization with the linear transformation such as RCA. This provides a more stable evolution. We also see that the final projection looks better in the second case (figure 5.6).

Results that demonstrate performance in terms of accuracy on small data sets are attached in appendix, tables B.5 and B.6. The comparison has as baseline the classical NCA and the extended version of NCA with compact support kernels and background distribution (NCA CS BACK). The results on the large data set are available in table 5.10.

5.5 Recommendations

When dealing with a large data set, we suggest to first try NCA SL CS. This method has the advantage of being easy to implement and very fast even for large data sets. However, the speed does come at a cost. We note a slight decrease in accuracy for most experimentations. Nonetheless, applying NCA SL CS first gives us an idea of how well NCA can perform on a given data set. If one is pleased with the score, it can use the classic NCA SL or the more sophisticated NCA SL + k-d tree and hope to get an improvement of a couple of percentages in accuracy. However all the methods offer better scores than the eigendecomposition based methods.

Appendix A

MATLAB code

A.1 NCA objective function

```
function [f, df] = nca_obj(A, X, c)
%NCA_OBJ Neighbourhood Component Analysis objective function.
%Returns function value and the first order derivatives.
%
%     [f, df] = nca_obj(A, X, c)
%
% Inputs:
%       A dxD - projection matrix (d <= D).
%       X DxN - data.
%       c 1xN - class labels.
%
% Outputs:
%       f 1x1   - function value.
%      df 1xD*d - derivative values.

% Dan Oneata, June 2011

  [D N] = size(X);
  A = reshape(A,[],D);

  df = zeros(size(A,1),D);

  AX = A*X;
  dist_all = square_dist(AX,AX);
  kern_all = exp(-dist_all);
  kern_all(1:N+1:end) = 0;

  % Compute distance to the neighbours:
  row_dict = logical(eye(max(c)));
  neigh_mask = row_dict(c,c);
  kern_neigh = sum(kern_all.*neigh_mask, 1);
  kern_sum = sum(kern_all, 1);
  p = kern_neigh ./ kern_sum;
  p = max(p, eps);

  % Compute function value:
  f = - sum(p);

  if nargout > 1,
    K = bsxfun(@rdivide, kern_all, kern_sum);
    K = max(K, eps);
    K = K';
    for i=1:N,
        x_ik = bsxfun(@minus,X(:,i),X);
        Ax_ik = bsxfun(@minus,AX(:,i),AX);
        x_ij = x_ik(:,c==c(i));
        Ax_ij = Ax_ik(:,c==c(i));

        % Update gradient value:
        df = df + p(i) * bsxfun(@times, K(i,:), Ax_ik) * x_ik' ...
            - bsxfun(@times, K(i,c==c(i)), Ax_ij) * x_ij';
    end
    df = -2*df;
    df = df(:);
  end

end
```

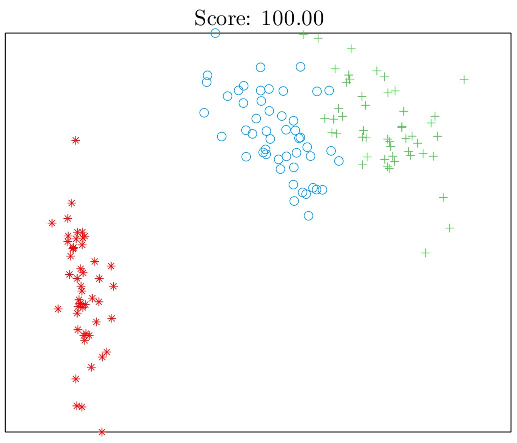
Notes:

1. `square_dist` computes the pairwise distances between two D-dimensional sets of points. The function was written by Iain Murray and it can be found in his MATLAB Toolbox, available at the following URL: `http://homepages.inf.ed.ac.uk/imurray2/code/imurray-matlab/square_dist.m`.

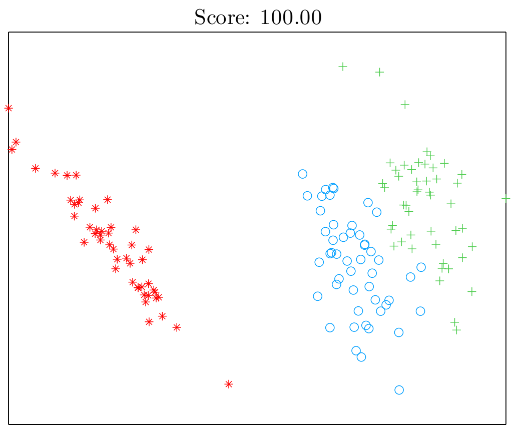
2. This implementation is suitable for rather small data sets whose pairwise distance matrix can fit in the RAM. If you are dealing with large data sets, it is better to iterate through the data set and compute the distances successively: from a point to the rest of the data set.

Appendix B

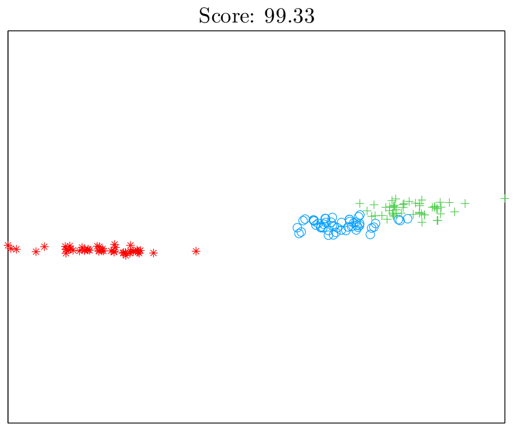
Further results



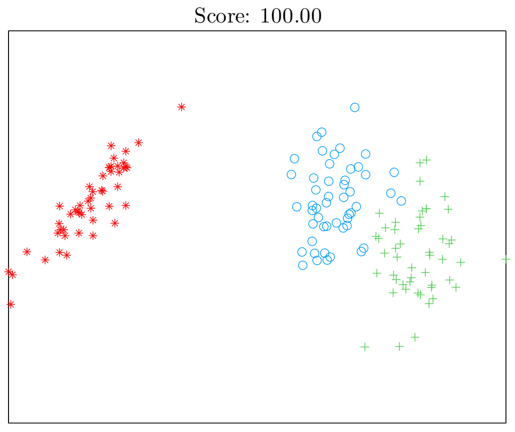
(a) NCA projection after random initialization



(b) NCA projection after PCA initialization

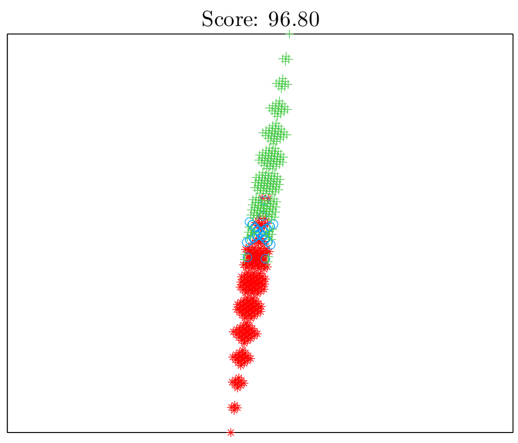


(c) NCA projection after LDA initialization

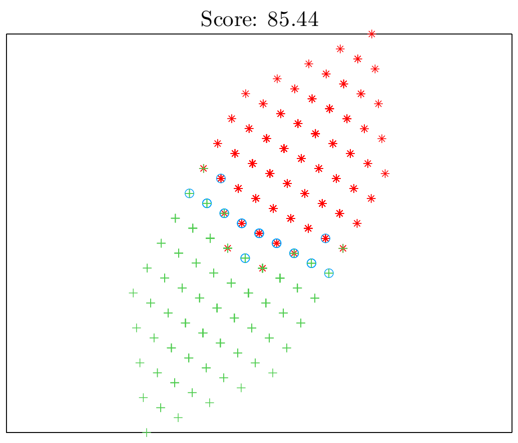


(d) NCA projection after RCA initialization

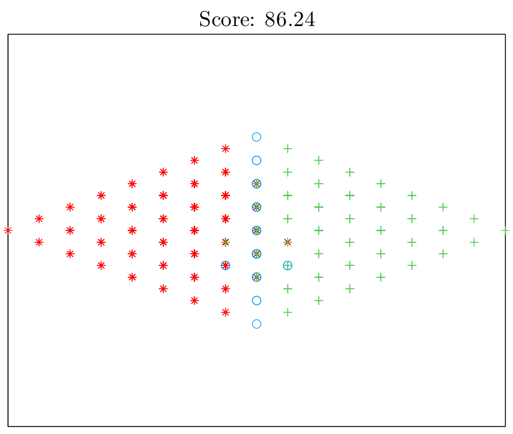
Figure B.1: Results on `iris` data set.



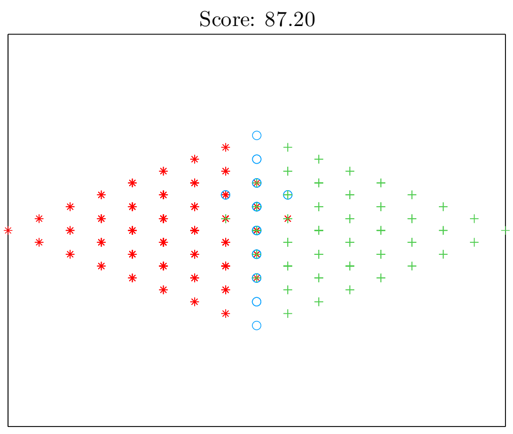
(a) NCA projection after random initialization



(b) NCA projection after PCA initialization

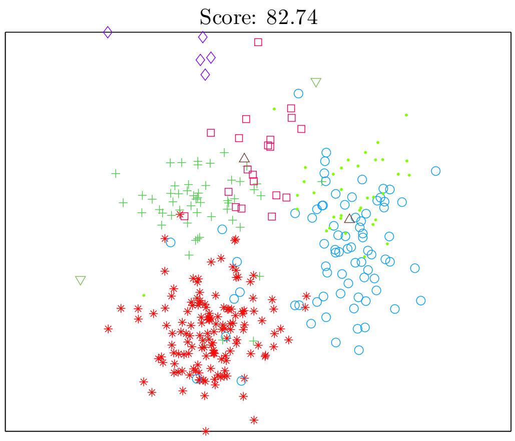


(c) NCA projection after LDA initialization

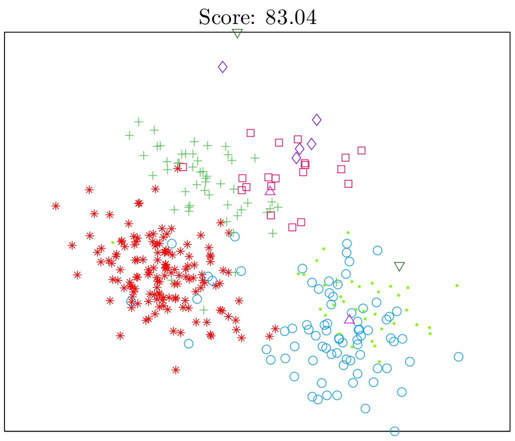


(d) NCA projection after RCA initialization

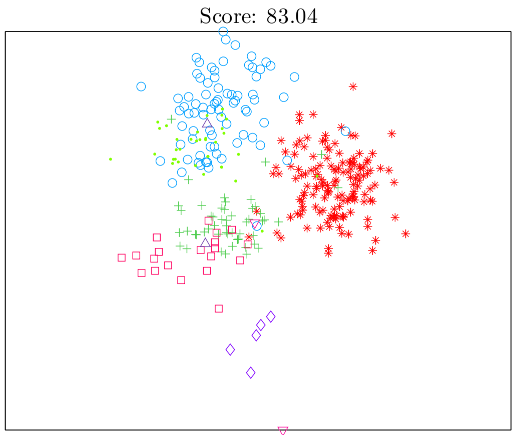
Figure B.2: Results on `balance` data set.



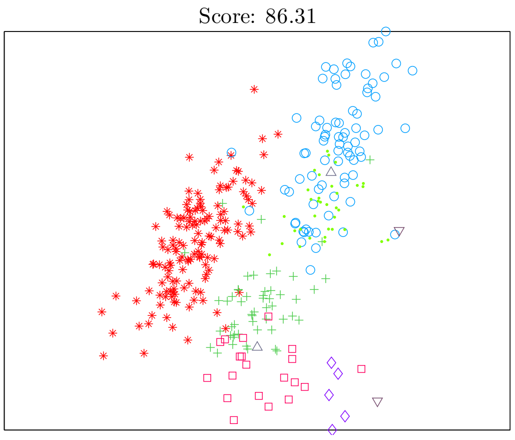
(a) NCA projection after random initialization



(b) NCA projection after PCA initialization



(c) NCA projection after LDA initialization



(d) NCA projection after RCA initialization

Figure B.3: Results on `ecoli` data set.

Data set	d	Train	1-NN	NCA
`balance`	2	88.35 ± 0.83	87.37 ± 0.49	90.45 ± 0.38
	3	94.0 ± 1.0	94.44 ± 0.80	95.11 ± 0.56
	$D = 4$	94.70 ± 0.87	95.32 ± 0.34	96.14 ± 0.29
`glass`	2	55.0 ± 2.5	54.9 ± 1.7	59.0 ± 1.7
	3	65.4 ± 3.3	59.0 ± 1.2	62.5 ± 1.2
	4	69.2 ± 3.0	60.5 ± 1.2	63.77 ± 0.90
	5	68.2 ± 3.0	64.1 ± 1.3	65.7 ± 1.1
	$D = 9$	76.0 ± 2.3	68.0 ± 1.6	69.7 ± 1.6
`ionosphere`	2	89.0 ± 1.7	85.71 ± 0.94	87.08 ± 0.95
	3	91.0 ± 1.8	86.6 ± 1.1	86.8 ± 1.1
	4	89.6 ± 1.4	87.7 ± 1.1	87.74 ± 0.97
	5	91.4 ± 1.7	88.07 ± 0.73	88.35 ± 0.78
	$D = 33$	92.6 ± 1.5	84.72 ± 0.57	84.34 ± 0.59
`iris`	2	96.41 ± 0.94	96.33 ± 0.57	97.00 ± 0.46
	3	97.30 ± 0.82	95.67 ± 0.57	96.00 ± 0.62
	$D = 4$	97.5 ± 1.3	95.67 ± 0.71	96.11 ± 0.66
`wine`	2	98.80 ± 0.70	97.22 ± 0.65	97.50 ± 0.49
	3	98.17 ± 0.87	97.32 ± 0.58	97.69 ± 0.52
	4	98.51 ± 0.86	97.04 ± 0.53	97.32 ± 0.46
	5	99.22 ± 0.40	96.95 ± 0.45	97.13 ± 0.47
	$D = 13$	99.25 ± 0.62	96.85 ± 0.41	96.85 ± 0.41

Table B.1: Accuracy scores of NCA trained using stochastic learning on small and medium sized data sets. The scores are averaged over 20 repeats. RCA was used for initialization.

Data set	d	Train	1-NN	NCA
`yeast`	2	42.83 ± 0.93	45.85 ± 0.41	53.90 ± 0.47
	3	47.30 ± 0.85	46.49 ± 0.48	54.37 ± 0.40
	4	50.1 ± 1.2	48.69 ± 0.68	54.50 ± 0.63
	5	50.41 ± 0.82	50.19 ± 0.45	55.35 ± 0.49
	$D = 8$	52.09 ± 0.84	51.34 ± 0.47	55.44 ± 0.56
`ecoli`	2	74.1 ± 2.2	75.8 ± 1.0	81.98 ± 0.96
	3	81.2 ± 1.8	78.72 ± 0.77	81.49 ± 0.75
	4	79.9 ± 2.1	80.20 ± 0.73	83.12 ± 0.86
	5	80.9 ± 2.4	79.16 ± 0.77	83.62 ± 0.64
	$D = 7$	82.0 ± 2.1	80.89 ± 0.52	84.85 ± 0.48
`pima`	2	70.6 ± 1.1	69.09 ± 0.64	76.65 ± 0.44
	3	70.0 ± 1.3	68.57 ± 0.62	75.56 ± 0.82
	4	71.3 ± 1.2	69.41 ± 0.60	74.37 ± 0.57
	5	73.0 ± 1.2	69.03 ± 0.71	72.47 ± 0.81
	$D = 8$	76.2 ± 1.5	69.29 ± 0.88	70.48 ± 0.81
`segment`	2	84.62 ± 0.64	88.53 ± 0.39	88.80 ± 0.33
	3	92.27 ± 0.57	95.45 ± 0.31	94.46 ± 0.33
	4	94.24 ± 0.46	97.11 ± 0.18	96.09 ± 0.20
	5	94.98 ± 0.35	97.05 ± 0.16	96.64 ± 0.13
	$D = 18$	95.73 ± 0.57	97.01 ± 0.28	96.68 ± 0.30
`transfusion`	2	70.48 ± 0.99	73.60 ± 0.47	78.78 ± 0.42
	3	70.08 ± 0.92	73.49 ± 0.54	77.95 ± 0.39
	$D = 4$	66.0 ± 3.1	71.44 ± 0.56	75.4 ± 2.9

Table B.2: Accuracy scores of NCA trained using stochastic learning on small and medium sized data sets. The scores are averaged over 20 repeats. RCA was used for initialization.

		Conjugate gradients		Bold driver	
Data set	d	1-NN	NCA	1-NN	NCA
`balance`	2	92.74 ± 0.57	92.29 ± 0.56	92.05 ± 0.93	92.45 ± 0.81
	3	94.81 ± 0.52	94.87 ± 0.53	92.79 ± 0.95	92.87 ± 0.97
	4	95.16 ± 0.39	95.32 ± 0.33	94.18 ± 0.76	94.36 ± 0.69
`glass`	2	53.6 ± 1.4	54.2 ± 1.5	58.7 ± 1.5	62.5 ± 1.4
	3	61.8 ± 1.1	60.9 ± 1.2	66.3 ± 1.1	67.1 ± 1.1
	4	62.7 ± 1.4	62.2 ± 1.5	65.8 ± 1.0	66.6 ± 1.3
	5	64.5 ± 1.2	63.46 ± 0.97	66.69 ± 0.93	67.00 ± 0.87
	9	66.5 ± 1.8	65.5 ± 1.8	66.0 ± 1.7	65.6 ± 1.7
`ionosphere`	2	86.23 ± 0.80	86.18 ± 0.75	82.78 ± 0.98	82.78 ± 0.91
	3	87.69 ± 0.62	87.64 ± 0.62	86.04 ± 0.83	86.08 ± 0.85
	4	87.64 ± 0.55	87.64 ± 0.57	86.51 ± 0.73	86.51 ± 0.73
	5	89.39 ± 0.70	89.39 ± 0.70	87.88 ± 0.64	87.88 ± 0.64
	33	87.97 ± 0.68	87.97 ± 0.68	88.44 ± 0.59	88.44 ± 0.59

Table B.3: Comparison in terms of accuracy between two possible optimization methods for NCA: conjugate gradients and gradient ascent with the "bold driver" heuristic

		Conjugate gradient		Bold-driver	
Data set	d	1-NN	NCA	1-NN	NCA
iris	2	94.44 ± 0.51	94.11 ± 0.48	93.67 ± 0.82	93.78 ± 0.84
	3	95.44 ± 0.73	95.00 ± 0.75	95.22 ± 0.72	95.33 ± 0.74
	4	96.56 ± 0.46	96.44 ± 0.46	95.44 ± 0.76	95.56 ± 0.77
wine	2	96.67 ± 0.38	96.57 ± 0.38	96.30 ± 0.54	96.30 ± 0.54
	3	96.39 ± 0.48	96.39 ± 0.48	96.57 ± 0.59	96.57 ± 0.59
	4	96.20 ± 0.40	96.48 ± 0.41	97.13 ± 0.52	97.13 ± 0.52
	5	96.57 ± 0.60	96.57 ± 0.60	96.76 ± 0.55	96.76 ± 0.55
	13	96.20 ± 0.48	96.20 ± 0.48	97.22 ± 0.48	97.22 ± 0.48
yeast	2	42.37 ± 0.81	43.20 ± 0.84	44.65 ± 0.56	46.68 ± 0.57
	3	48.06 ± 0.67	48.30 ± 0.65	47.06 ± 0.53	47.95 ± 0.54
	4	49.06 ± 0.47	49.14 ± 0.47	49.48 ± 0.42	49.83 ± 0.38
	5	50.09 ± 0.39	50.09 ± 0.38	50.16 ± 0.61	50.41 ± 0.62
	8	50.66 ± 0.52	50.70 ± 0.52	50.19 ± 0.58	50.28 ± 0.60

Table B.4: Comparison in terms of accuracy between two possible optimization methods for NCA: conjugate gradients and gradient ascent with the "bold driver" heuristic

		NCA		NCA CS		NCA CS BACK	
Data set	d	1-NN	NCA	1-NN	NCA	1-NN	NCA
`balance`	2	92.74 ± 0.57	92.29 ± 0.56	87.90 ± 0.68	89.04 ± 0.52	89.9 ± 1.3	91.04 ± 0.94
	3	94.81 ± 0.52	94.87 ± 0.53	89.60 ± 0.76	90.08 ± 0.60	65.2 ± 3.8	87.85 ± 0.53
	4	95.16 ± 0.39	95.32 ± 0.33	90.82 ± 0.62	92.07 ± 0.54	67.2 ± 3.4	86.91 ± 0.30
`glass`	2	53.6 ± 1.4	54.2 ± 1.5	55.8 ± 1.1	57.2 ± 1.7	58.5 ± 1.2	59.4 ± 1.5
	3	61.8 ± 1.1	60.9 ± 1.2	63.2 ± 1.4	56.3 ± 1.8	61.5 ± 1.2	63.5 ± 1.1
	4	62.7 ± 1.4	62.2 ± 1.5	62.0 ± 1.4	54.5 ± 1.6	65.2 ± 1.2	65.2 ± 1.0
	5	64.5 ± 1.2	63.46 ± 0.97	64.9 ± 1.1	57.1 ± 1.3	67.5 ± 1.2	67.2 ± 1.3
	9	66.5 ± 1.8	65.5 ± 1.8	68.9 ± 1.5	57.8 ± 1.5	68.2 ± 1.6	64.1 ± 1.3
`ionosphere`	2	86.23 ± 0.80	86.18 ± 0.75	82.83 ± 0.74	85.33 ± 0.93	85.90 ± 0.74	84.20 ± 0.74
	3	87.69 ± 0.62	87.64 ± 0.62	85.42 ± 0.62	83.44 ± 0.69	88.11 ± 0.74	84.91 ± 0.52
	4	87.64 ± 0.55	87.64 ± 0.57	86.08 ± 0.90	83.6 ± 1.1	88.73 ± 0.92	83.7 ± 1.1
	5	89.39 ± 0.70	89.39 ± 0.70	88.87 ± 0.61	81.46 ± 0.76	86.65 ± 0.86	82.08 ± 0.98
	33	87.97 ± 0.68	87.97 ± 0.68	86.56 ± 0.92	66.09 ± 0.95	87.26 ± 0.45	84.76 ± 0.83

Table B.5: Accuracy scores for the compact support versions of NCA.

		NCA		NCA CS		NCA CS BACK	
Data set	d	1-NN	NCA	1-NN	NCA	1-NN	NCA
`iris`	2	94.44 ± 0.51	94.11 ± 0.48	95.22 ± 0.59	94.78 ± 0.72	95.22 ± 0.67	95.11 ± 0.60
	3	95.44 ± 0.73	95.00 ± 0.75	95.33 ± 0.93	95.0 ± 1.0	94.89 ± 0.80	94.2 ± 1.1
	4	96.56 ± 0.46	96.44 ± 0.46	95.56 ± 0.54	95.67 ± 0.53	95.2 ± 1.0	93.11 ± 0.95
`wine`	2	96.67 ± 0.38	96.57 ± 0.38	97.04 ± 0.51	96.94 ± 0.44	97.22 ± 0.48	97.59 ± 0.46
	3	96.39 ± 0.48	96.39 ± 0.48	96.48 ± 0.63	95.93 ± 0.65	97.22 ± 0.53	97.50 ± 0.40
	4	96.20 ± 0.40	96.48 ± 0.41	98.06 ± 0.36	98.06 ± 0.28	98.24 ± 0.33	98.89 ± 0.30
	5	96.57 ± 0.60	96.57 ± 0.60	96.20 ± 0.71	95.00 ± 0.81	97.78 ± 0.43	97.87 ± 0.40
	13	96.20 ± 0.48	96.20 ± 0.48	95.09 ± 0.70	92.59 ± 0.92	94.63 ± 0.81	98.43 ± 0.30
`yeast`	2	42.37 ± 0.81	43.20 ± 0.84	42.57 ± 0.63	48.31 ± 0.90	44.38 ± 0.66	48.24 ± 0.79
	3	48.06 ± 0.67	48.30 ± 0.65	46.63 ± 0.72	51.6 ± 1.0	47.33 ± 0.61	50.67 ± 0.55
	4	49.06 ± 0.47	49.14 ± 0.47	49.26 ± 0.51	53.06 ± 0.94	49.93 ± 0.36	53.08 ± 0.30
	5	50.09 ± 0.39	50.09 ± 0.38	50.16 ± 0.59	52.96 ± 0.93	49.79 ± 0.62	54.54 ± 0.56
	8	50.66 ± 0.52	50.70 ± 0.52	50.93 ± 0.44	54.89 ± 0.43	48.73 ± 0.65	56.93 ± 0.62

Table B.6: Accuracy scores for the compact support versions of NCA.