

Modelling Handwritten Digit Data using Probabilistic Principal Component Analysis

Mohamed E.M. Musa, Robert P.W. Duin and Dick de Ridder

Pattern Recognition Group,
Department of Applied Physics
Delft University of Technology,
Lorentzweg 1, 2628 CJ Delft The Netherlands
e-mail: musa@ph.tn.tudelft.nl

Keywords: Principal Component Analysis, mixture models, model selection, handwritten digit recognition

Abstract

Principal Component Analysis (PCA) is one century old now. Nevertheless, it still undergoes research and new extensions are found. Probabilistic Principal Component Analysis (PPCA, proposed by Tipping and Bishop) is one of these recent PCA extensions. PPCA defines a probabilistic generative model for PCA. It can easily be extended to mixture models. Among recent mixture density theoretical developments is Dasgupta's algorithm for learning mixtures of Gaussians. We propose enhancing PPCA's EM training algorithm by increasing the number of submodels iteratively, together with using a version of Dasgupta's algorithm for parameter initialization. Handwritten digit classification is an extensively studied problem. Therefore, it is considered a popular benchmark for model comparison. Experimental results show remarkable improvement when using our extensions.

1 Probabilistic Principal Component Analysis

Principal Component Analysis (PCA) emerged at the turn of the twentieth century. Nevertheless, it is still one of the most popular techniques for dimensionality reduction. If we project an n -dimensional data set (assumed, without loss of generality, to have zero mean) into an m -dimensional subspace ($m < n$) and the resulting variance of the projected data is measured, then the principal components define a subspace such that this "captured" variance is highest. These m principal components are found to be the dominant m eigenvectors (a proof can be found in almost all multivariate statistics books). Karl Pearson (1901) is the first to describe the technique. A descrip-

tion of practical computing methods came much later from Hotelling (1935). Later, with the advent of electronic computers, the technique achieved widespread use as the machine computational power increases the dimensions of the data that can be manipulated by many order of magnitudes.

Despite its popularity, PCA has some limitations. Two of its main limitations are:

- (I) A lack of a probabilistic or generative model.
- (II) The technique is globally linear.

To overcome these two limitations Tipping and Bishop [1] propose a mixture model for probabilistic PCA. Their aim is to overcome the first limitation by defining a probabilistic model for PCA, and to model the data globally nonlinear by a mixture of local linear submodels. From here on, we will call this model "PPCA". A number of implementations of "mixtures of PCA" have been proposed in the literature, before PPCA. Each defines a different algorithm or a variation [2,3], but none defines a probability density.

1.1 Probabilistic Principal Component Analysis

PCA is merely a rotation of an n -dimensional data space and selection of m dimensions in the rotated space as the new m -dimensional linear subspace. If the data in the original space is Gaussian then the data in the rotated subspace is also Gaussian. Therefore, PPCA is a Gaussian modeller that defines the relation between the Gaussians in the original space and the subspace. The generative model:

$$t = Wx + \mu + \epsilon \quad (1)$$

specifies the relation between these two Gaussians where t (n -dimensional) is the data vector, x (m -dimensional) is the subspace vector, W are the m dominant eigenvectors (principal components, or PCs), μ is the data mean, and ϵ is a noise model which is assumed isotropic Gaussian (i.e. $\epsilon \sim N(0, \sigma I)$) approximating the average of the minor eigenvalues. This allows the following definitions of probability distributions over t -space and x -space:

$$p(t) = (2\pi)^{-n/2} |C|^{-1/2} \exp\left(-\frac{1}{2}(t - \mu)^T C^{-1}(t - \mu)\right) \quad (2)$$

$$p(t|x) = (2\pi\sigma^2)^{-n/2} \exp\left(-\frac{1}{2}\sigma^2\|t - Wx - \mu\|^2\right) \quad (3)$$

$$p(x) = (2\pi)^{-n/2} \exp\left(-\frac{1}{2}x^T x\right) \quad (4)$$

Formula (2) is the general Gaussian formula with the covariance matrix approximated as:

$$C = \sigma^2 I + WW^T \quad (5)$$

1.2 Mixtures of Probabilistic Principal Component Analyzers

The Gaussian density model has proved to be the most exceedingly popular density model, perhaps mainly due to its simplicity and general applicability. This popularity leads density modelers to build mixtures of Gaussians when it seems that one Gaussian does not fit the data well, or when the data naturally emerges from a mixture of Gaussians. A multivariate mixture of Gaussians is given by the weighted sum :

$$F_k(t) = \sum_j^k q_j p_j(t, \theta_j) \quad (6)$$

where q_j are the mixing weights satisfying $\sum q_j = 1$ and $q_j > 0$. $p_j(t, \theta_j)$ is the j^{th} Gaussian, parameterized by θ_j . As PPCA is a Gaussian model, formula (6) can also be used to define a mixture of PPCA models. The only difference to be mentioned here is in the parameter set, θ_j . For the general Gaussian model, θ_j contains the mean and the covariance matrix. For the PPCA Gaussian model, θ_j contains the mean μ_j , the m -dominant eigenvectors W_j and the average of the minor eigenvalues σ_j .

2 Training

Methods that can be used for distributing the training data among submodels are categorized as *hard clustering* or *soft clustering*. Hard clustering necessitates a two-stage procedure: partitioning of the data space followed by estimation of the parameters of each partition. The other method, EM algorithm, carries out the partitioning process and submodel parameter estimation as one global process. During the E-step, responsibility for each data point is assigned to the submodels; during the M-step PCA is performed, altering the parameters of a submodel to fit the assigned data better. The fitting measurement – i.e. how well the assigned data fit their model – is one of the main differences between PPCA and previous work. PPCA is a probabilistic model, therefore the fitting is measured by the probability density function and the maximum likelihood framework is the proper framework for the M-step. As pointed out by the authors of PPCA, all previous work suffer the limitation of the absence of a probability. This limitation is the main reason for finding different algorithms in the literature for measuring how the data fits in the model [2,3]. One of the measurements that have been used extensively is the reconstruction error. In reconstruction error the data point is projected into the subspace, and then back to the data space. The distance between the reconstructed data point and the original one is the reconstruction error. However, the reconstruction error by itself is unfit for defining a probability measure.

2.1 Hard Clustering and Initialization for the EM Algorithm

EM in a maximum likelihood framework may be the most appealing framework that can be used for estimating the PPCA mixture model. Yet some difficulties of EM should be considered. While EM is guaranteed not to decrease the likelihood, there is no guarantee that it may not get stuck in a local maximum. For mixture learning, the algorithm has no proper extensions: neither for finding the optimal number of submodels, nor a generally accepted method for parameter initialization.

If we need h data points to estimate a mean of a *one*-dimensional data set, then we need h^n data points to estimate the mean of n -dimensional data sets. In other words, a plausible data size for an n -dimensional data mean estimation is around $2^{\mathcal{O}(n)}$. This is an astronomical figure for high dimensional data and no practical training data set is expected to fulfil this requirement. Is it possible to reduce the dimension of the data so dramatically that this requirement actually becomes reasonable? Dasgupta showed that to estimate the mean of a mixture of Gaussians with common covariance matrix, we can map the data to a random subspace of size $O(\log k)$ dimension,

where k is the number of Gaussians, without collapsing the Gaussians together [4]. This makes the number of data points needed only polynomial in k . The reader may wonder how can we preserve the relative distance between data points with this high reduction of dimensionality by just random projection.

However, in our situation we do not want this. We want most of the pairwise distances to contract significantly, so that the fraction of points within the expected distance from any Gaussian center in the reduced space \mathbb{R}^m is exponentially greater than the fraction of points within the expected distance from the same center in the original space \mathbb{R}^n . At the same time we do not want the distance between different Gaussians to contract. These conflicting requirements are accommodated by a projection to just $\mathcal{O}(\log k)$. This method of projection has another tremendous benefit: even if the original Gaussians are highly skewed, their projected counterparts will be more spherical and there by easier to learn. A proof for these arguments would be found in [4].

The clustering method described above which we will refer to as ‘‘Dasgupta’s algorithm’’ can be used as a hard clustering method for training a PPCA model. However, it can also be used for calculating an initial estimation for the EM algorithm (i.e. soft clustering). In this project we test a version of Dasgupta’s algorithm as a hard clustering method for PPCA. Finding good initializations as a starting estimation may cut down the training time and (most importantly) may help in escaping some local maxima.

Increasing the number of submodels iteratively during training may help in finding the optimal number of submodels. So in addition to testing Dasgupta’s hard clustering we also test the efficiency of an EM algorithm initialized by clusters found by Dasgupta’s algorithm, as well as the efficiency of increasing the number of submodels iteratively. This latter idea was investigated by Li and Barron [5]. The next section defines these techniques algorithmically.

2.2 Algorithms

The two algorithms in this paper will be called algorithm A, for initialization, and algorithm B, for optimising the number of submodels iteratively.

2.2.1 Algorithm A

1. Project the whole data into the space of the m dominant principal components.
2. $S =$ projected data
3. For $i = 1 \dots k$
 - (a) For all $x \in S$, let r_x be the smallest radius such that there are $\geq p$ points within distance r_x from x .

- (b) Let μ^* be the point x with the lowest radius r_x
- (c) let $S' =$ the q closest points to μ^*
- (d) From the points corresponding to S' in the data space estimate θ_j (i.e. μ^*, W_j, σ_j)
- e) $S = S - S'$

For this algorithm, the modeler should determine the number of Gaussians, k ; the subspace dimensionality, m ; the number of points that are expected to be close to it i -th Gaussian center, p ; and the number of points expected to be from the same i^{th} Gaussian, q . These parameters depend upon the problem at hand. However, two guidelines for the relation between these parameters, are: (I) $m \cong \mathcal{O}(\log k)$, i.e. the dimension of the projection subspace should be in the order of the number of Gaussians; (II) $p < q$, i.e. the number of points used for finding the centers must be a subset of the points used for the estimation of the entire Gaussians.

After projecting the data into the subspace the algorithm-control iterates k times between the subspace and the original space: find a cluster in the subspace (steps (a)-(c)); go back to the original space and estimate one submodel parameters using the points corresponding to the recently found cluster (step (d)); remove the cluster (step (e)).

2.2.2 Algorithm B

1. Run EM on k submodels initialized by algorithm A.
2. Find a new cluster (using algorithm A) using $\frac{1}{k}$ th part of the data. The data points chosen are the points that have the lowest probability for the current k submodels.
3. Run EM on $k + 1$ submodels
4. If $\text{likelihood}(k) \cong \text{likelihood}(k + 1)$ or $k = l$, stop (l is the upper limit for the number of Gaussians); otherwise go to step 2.

3 Application

Handwritten digit recognition is a popular classification problem that is used extensively in testing relative density classification approaches as well as discriminative approaches [2]. The popularity and the availability of large data sets enable it to stand as a good benchmark for testing and comparing different classification methods. Especially for our problem, there are some publications on handwritten digit classification using ‘‘mixtures of PCA’’ [1,2]. In this section we describe our experiments for testing the proposed algorithms on modeling handwritten digits using mixtures of PPCA.

The data set used in our experiments was extracted from the well-known NIST handwritten digit database [6]. The original data set consisted of 128x128 pixel binary images. In pre-processing, these images were normalised for position, size, slant and stroke width, resulting in 16x16 pixel grey-value images [7]. Furthermore, for the experiments described in this paper PCA was used on the entire data set to reduce the number of dimensions from 256 to 64. The resulting data set was used to construct training and test sets: all experiments reported here were repeated three times on randomly selected training and test sets of 1000 samples per class each.

3.1 Experiments description

We have trained a mixture of PPCA models for handwritten digit recognition and tested its classification performance. Each digit is modeled by ten submodels at most and each submodel has ten PCs. During the training phase, only images of one class are presented to the model generator program, i.e. each digit model is built separately. 1000 patterns (images) per class (digit) have been used for training. Each pattern is an 8x8 image taken as one 64-dimensional vector.

We have designed four experiments:

- I. Training ten submodels per class starting with random initializations.
- II. Training ten submodels per class using algorithm A – hard clustering only.
- III. Training ten submodels per class starting with initialization from algorithm A.
- IV. Training at most ten submodels per class starting with initialization from algorithm A and iteration controlled by Algorithm B.

All experiments are repeated three times with differently drawn train and test sets. Experiment (I) is repeated three times for each pair of sets.

3.2 Clustering Parameters

The parameters p and q in algorithm A are initialized with the values 50 and 100 respectively. We have set q to 100 to divide one digit space into ten subspaces. However, experiment (IV) indicates that it could have been chosen higher as there are only 5 submodels per digit on average. The choice of p follows from that, as a demand is that $p < q$.

4 Results

For all experiments we have used the same program for testing. The testing data set consists of 1000

patterns per class. Table 1 summarizes the first testing results. The error in this table is the percentage of the misclassified patterns. We added a fourth column to the table to show the average number of submodels found by experiment IV.

As an illustration, figure 1 shows the cluster centers found in experiment II by algorithm A.

The data set has been classified before using a number of methods [7,8]. Table 2 gives an overview of the results obtained thus far on a training set of 1000 samples per class.

4.1 Discussion

The results of this first set of experiments show that performances of all methods of initialisation and model fitting are nearly equal. Especially the fact that the hard clustering method, algorithm A, performs as well as the EM algorithm is curious. To investigate what caused this, we inspected the models found by each algorithm. It became obvious that for some models, problems in estimating σ , the noise level, caused poor performance.

The technique, adopted by PPCA, of approximating the average of the minor eigenvalues corresponding to the variance not explained by PCA, as a noise parameter σ gives insight into this problem. Eqn. 5 shows that the diagonal components of the model covariance matrix C are dependent on σ in the minor eigenvector directions. This structure makes the covariance matrix very sensitive to the actual value of σ . For very small values, C will become singular and the whole model becomes undefined. However, even when the matrix is non-singular and σ is very small, the model will become prone to overfitting. This can be seen by realizing that for small σ , some elements of C^{-1} will become very large. Now, normally the image elements which are multiplied by the large values in C^{-1} (see eqn. 2) will be very small, as their variance is very low. However, if in the data set an image occurs which has some noise present in pixel positions which normally have low variance, this noise will be blown up. It will have a large effect on the estimate of the probability of the image (eqn. 2) and both training (specifically, the E-step) and recognition will suffer.

This is the main reason that makes the clustering-only experiment (experiment II) give results comparable to the EM-based experiments (experiment I, III and IV), as the noise has less influence on algorithm A than on the EM algorithm. Table 3 shows the average σ (over all submodels) and the recognition error for each class for one of the experiments I. It is obvious from the table that digit “1” has one of the worst results and the lowest value of σ . This gives the insight that σ can be used as a clue for deciding on the optimal number of PCs for the PPCA model in gen-

Experiment	Initialization	Model fitting	Error (%)	Std. dev.	No. submodels
I	random	EM	2.63	0.13	100
II	algorithm A	-	2.70	0.22	100
III	algorithm A	EM	2.48	0.25	100
IV	algorithm A	algorithm B / EM	2.59	0.16	54

Table 1: Test results for the four experiments using different initialisation and model fitting algorithms.

Type	Classifier	Error (%)
Bayes plug-in	Nearest mean	15.88
	Linear	9.84
	Quadratic	4.70
Neural network	LeNotre	4.87
	LeNet	3.43
	LeCun	2.32
	1 hidden layer @ 256 units	2.44
Support Vector Classifier	1 hidden layer @ 512 units	1.99
	Polynomial, 5 th degree	1.29
	Radial basis, $\sigma = 10$	1.38

Table 2: Results for various classifiers on the NIST data set.

Class	Avg. σ	Error (%)
0	0.18	1.99
1	0.06	4.99
2	0.25	0.79
3	0.22	2.79
4	0.20	1.48
5	0.24	1.75
6	0.16	1.92
7	0.14	3.00
8	0.22	2.36
9	0.13	4.10

Table 3: The average σ and test error for each of the 10 digit classes.

Experiment	Initialization	Model fitting	Error (%)	Std. dev.	No. submodels
I	random	EM	2.08	0.18	100
II	algorithm A	-	2.60	0.23	100
III	algorithm A	EM	1.86	0.25	100
IV	algorithm A	algorithm B / EM	2.20	0.13	53

Table 4: Test results for the four experiments using different initialisation and model fitting algorithms. In these experiments, σ was regularised by adding a constant value of 0.1 throughout all iterations.

eral and most importantly that regularising σ , e.g. by adding a regularisation constant, could improve performance.

To verify this, we re-ran the experiments using regularization. In each iteration of the EM algorithm, σ was calculated by averaging the minor eigenvalues, as in normal PPCA, and adding a fixed value of 0.1. For completeness, the σ 's estimated by algorithm A (experiment II) were also regularized in the same way before testing. Performance increased for each experiment. The results of these experiments, shown in table 4, show that randomly initialised PPCA now performs quite good compared to the results previously obtained using other classifiers (table 2). It is also obvious that EM is better than our hard clustering method, algorithm A alone. Using algorithm A as an initialisation of the EM algorithm improves results somewhat, but not significantly. However, the most interesting result is the fact the the number of submodels found in experiment IV, by algorithm B, is significantly reduced at only a small increase in test error. Algorithm B found 53 submodels on average, where the other models used 100; it gave a test error of 2.20% on average, vs. 2.08% on average for the standard PPCA algorithm.

5 Conclusion

We have applied Tipping and Bishop's mixture-of-PPCA model to handwritten digit recognition. In a first set of experiments, it was shown that some of the model's assumptions (e.g., equal noise variance in all directions) sometimes cause training problems or poor final performance. To remedy this, simple regularization was shown to improve results considerably. The experiments with the regularized algorithms show that the mixture-of-PPCA model performs quite well, compared to results obtained earlier using (often dedicated) classifiers.

A drawback of this method is that the number of dimensions per subspace, and the number of subspaces to be estimated per digit, have to be specified beforehand. We proposed a new way of training PPCA mixture models, inspired by the work of Dasgupta and Li and Barron, which automatically determines the optimal number of subspaces to be used per digit. This training method was shown experimentally to decrease the number of subspaces needed significantly, at only small cost to the performance.

As future work, we plan on investigating better ways of setting the parameters p and q of algorithm A (section 2.2). If we improve initialization, it is to be expected that the final performance of our new training method will improve as well. Furthermore, it would be interesting to see whether an automatic way of finding the optimal number of PCs could be incorporated into the algorithm as well, perhaps based on

the value of σ for each submodel. This would make the mixture-model virtually free of parameters.

References

- [1] Tipping, M.E. and Bishop, C.M. Mixtures of Principal Component Analyzers. *Neural Computation*, 11(2):443-482,,1999
- [2] Hinton, G.E., Dayan, P. and Revow, M. Modeling the manifolds of images of handwritten digits. *IEEE Transaction on Neural Networks* , 10(3):65-74, 1997.
- [3] Kambhatla, N. and Leen T. K. Dimension reduction by local principal Component analysis. *Neural Computation*, 9(7):1493-1516, 1997.
- [4] Dasgupta, S. Learning Mixtures of Gaussians. *Proc. IEEE Symposium on Foundation of Computer Science*, 1999.
- [5] Li, J.Q. and Barron, A.R., Mixture density estimation. In Solla, S.A., Leen, T.K. and Müller, K.-R., editors, *Advances in Neural Information Processing Systems 12*, MIT Press, Cambridge, MA. 2000.
- [6] Wilson, C.L. and Garris, M.D. Handprinted character database 3, february 1992. National Institute of Standards and Technology; Advanced Systems Division. URL: <http://www.nist.gov/srd/niststd19.htm>
- [7] de Ridder, D., Hoekstra, A. and Duin, R.P.W., Feature extraction in shared weights neural networks. In Kerckhoffs, E.J.H., Sloot, P.M.A., Tonino, J.F.M. and Vossepel, A.M., editors, *Proceedings of the 2nd annual conference of the Advanced School for Computing and Image Processing*, pp. 289-294, Delft, The Netherlands, 1996. ASCI, ASCI.
- [8] de Ridder, D., Adaptive methods of image processing, PhD thesis, to appear.

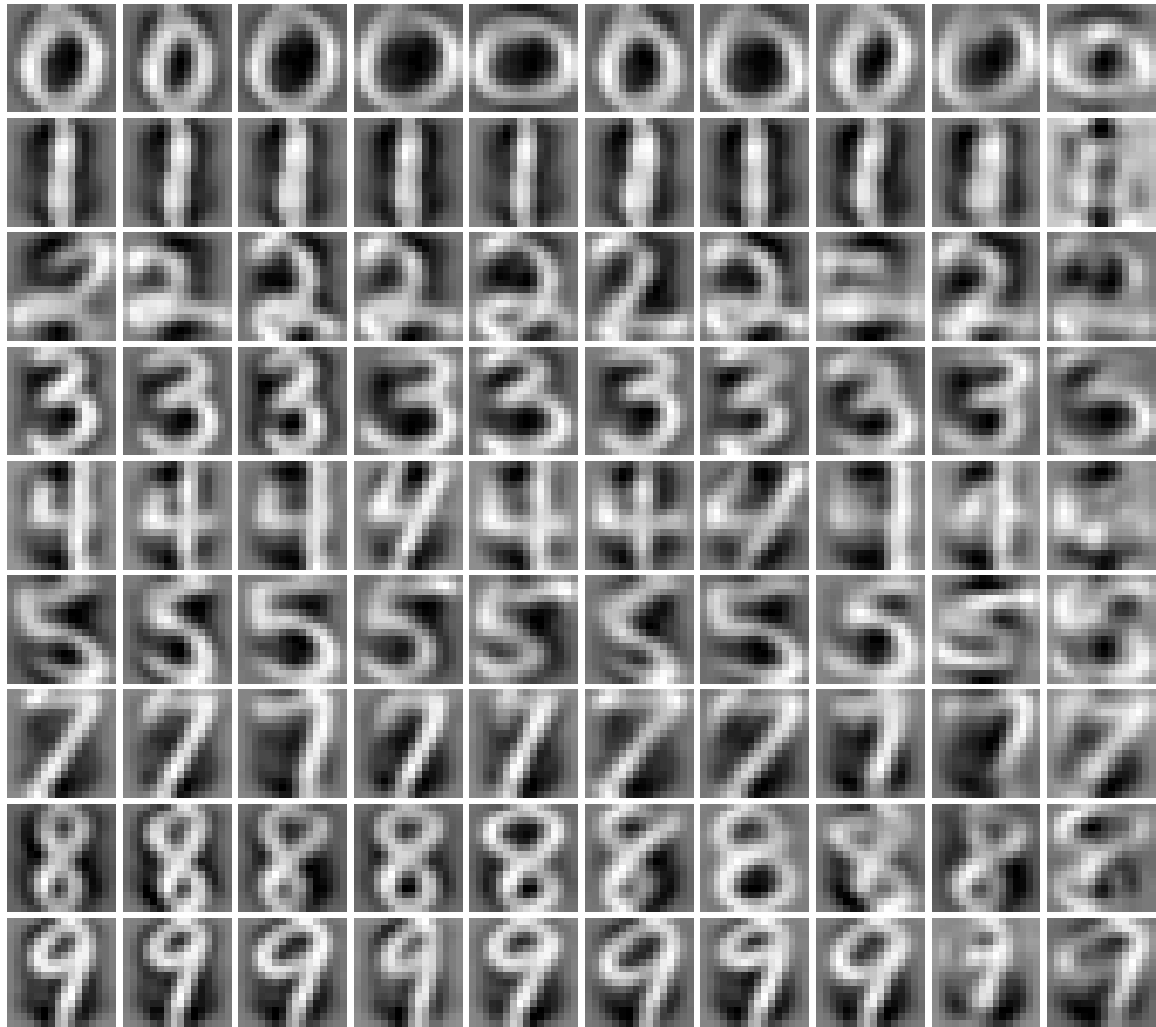


Figure 1: Submodel origins found in experiment II, by algorithm A only.