1 Introduction

Nearly 100 years ago, psychologist Charles Spearman, hypothesized incorrectly all dimensions of mental ability can be explained solely by one underlying factor, which he dubbed as *'g'*. Factor analysis seeks to uncover if there are indeed underlying independent, and unmeasurable, "factors" that affect the observed dependent variables. A typical factor analysis attempts to answer four major questions:

- How many factors are needed to explain relationship within data?
- Do these factors have real life meanings? If so, what?
- Do hypothesized factors explain the data well, and how much can the dimensionality be reduced while still maintaining inter-data relationship?
- Can factors measure the amount of pure random or unique variance within each observed variable?

Factor analysis is often trained using the Expectation-Maximization algorithms to try and identify the mean, variance, and dimensional mapping that corresponds to the lower dimensional factors. Professor Ng hypothesized that this algorithm could both be sped up and improved if ideas from Probabilistic Principal Component Analysis are used. The rest of the 5 page paper will present the new algorithm, discuss the data set and experiential methodology, present conclusions drawn from the results, and then present selected results. Additional material such as derivation of the algorithm, the basic factor analysis model, 4 selected scripts from the 24 code files, and supporting graphs are left for the appendix. We will start by presenting the algorithms.

2 Algorithms

2.1 Factor pPCA

The established algorithms (EM, GDA, and PCA) can be seen in Appendix A see page 7. Here we present the algorithm for factor analysis using probabilistic principal component analysis. The derivation of this algorithm is fully described in Appendix B (see page 9).
Given k and X,

a.) Calculate sample mean μ and sample variance Σ.

b.) While not converged

i.) Calculate the k largest eigenvalues $d = \{\lambda_1, ..., \lambda_k\}$ with corresponding (column) eigenvectors $v = \{v_1, ..., v_k\}$ of $\Psi^{-\frac{1}{2}}\Sigma\Psi^{-\frac{1}{2}}$ sorted in descending order.

ii.) Let $\tilde{d}$ denote the number of elements of $d > 1$. Set $t = \min(k, \tilde{d})$

iii.) Set $\forall i, j \leq k, Ł_{ij} = \begin{cases} (d_{ii} - 1)^{\frac{1}{2}} & i = j \leq t \\ 0, & \text{o.w} \end{cases}$

iv.) Set $U_{:,i} = v_i$.

v.) Set $\forall i, j, \Psi_{ij} = \begin{cases} (\Sigma - \Lambda\Lambda^T)_{ii} & i = j \\ 0, & i \neq j \end{cases}$

vi.) Set $\Lambda = \Psi^{\frac{1}{2}} * U * L$

2.2 Hypothesis

In approaching this analysis, we expected that Factor pPCA would be faster then Factor EM, but not necessarily more accurate, as it may impose the restrictive assumptions of pPCA. Nor did we expect it to maximize likelihood at every iteration, thus no longer guaranteeing convergence. Our test were generated to test this hypothesis.

3 Experimental Methodology

3.1 Optimizing Factor EM

In order to ensure that the pPCA-Factor algorithm was properly tested, a lot of effort was dedicated to optimizing Factor EM and GDA for matlab, by leveraging compiled existing code and reducing the number of for loops necessary for the code to execute. Everything was rewritten so that sums can be replaced with matrix multiplications while maintaining accuracy of result. This provides a stiffer test of accuracy and time efficiency of the pPCA-Factor algorithm.

3.2 Tests

We designed several layers of tests in order to ascertain the various properties of these algorithms. We developed two main tests for time as that was our main criteria:

a.) Every time we executed the algorithm, we tested for how long it ran in order to measure its time. We then plotted for each data set time versus dimensions. This gives us a direct comparison between FactorEM and FactorpPCA. In each test, we ensured that we initialized the algorithm with identical starting points and identical starting criteria.

b.) In order to ascertain why one algorithm took longer then the other, we had each algorithm return the number of iterations it took to converge. Similarly, we plotted the number of iterations versus dimensions, allowing us to determine if one algorithm took longer to converge because of iteration or computational cost.

We also developed three main tests for accuracy:

a.) We trained the algorithm on the entire data set, and then plotted training error as a function of dimension per algorithm. This allows us a direct comparison of error per dimension across algorithms.

b.) We also trained the algorithm on 70% of the data set, and then tested the predicative ability on the remaining 30%. This gives us an idea of the predicative ability of the algorithms.

c.) We also plotted the total log-likelihood per dimension per algorithm, in order to ascertain which algorithm claimed to have a greater likelihood of explaining the data set.

We also developed several tests to ensure the algorithms were correct:

a.) We tested if the algorithm were indeed correct by measuring if the log-likelihood was increasing per iteration, as it should be in any EM algorithm. If so, we can conclude if its correct. This was the problem– our EM algorithm does not monotonically increase.

b.) Wanting to ensure if initialization had any effect on the algorithm, we initialized the algorithms at 3 different points, one where Ψ is the identity matrix I, one where Ψ is 0.1 * I, and the last where Ψ is 10 * I. If the results were unaffected, we can conclude that the matrix is invariant to scaling. Λ was always chosen to be random as its initialization only affects EM.

Finally, in order to ensure that results were indeed at maxima points, we ran each of the above tests on the algorithm initialized with the converged result of the other algorithm. It was suspected that it won't take many iterations before the algorithm converges. The last part of any experimental methodology is of course, the data set selection.

3.3 Data Set

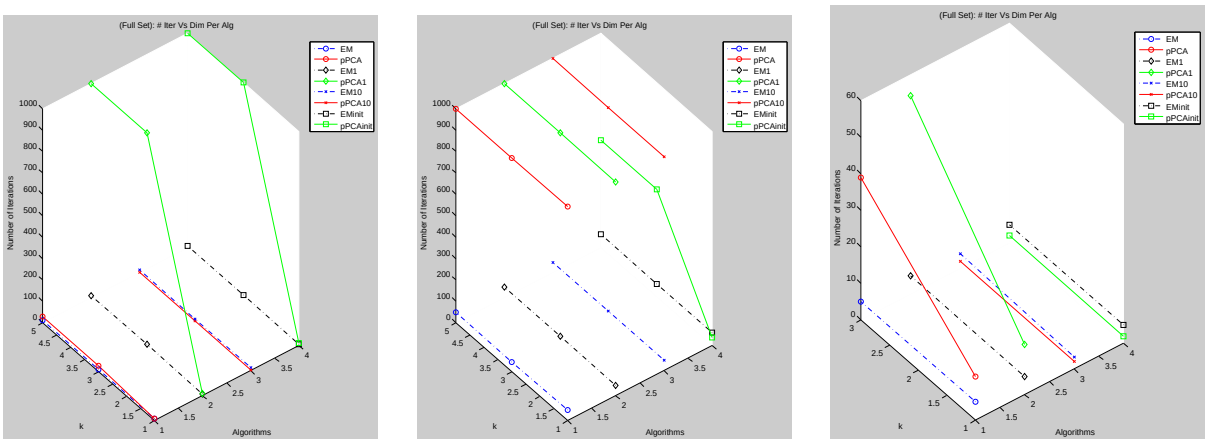
All of our data came from the UCI Machine Repository, see [1]. The data downloaded comprises 3 distinct sets.In "Machine", we attempt to predict if the computer is faster then the average processor speed based on publicly released machine data. In "Balance", we attempt to analyze psychological data, given the relative weighting and distance. Here there exists 3 classes, Left, Right, and Balanced. In "Votes", we attempt to predict the party of the politician given his votes in 16 key issues. Finally, we generated random data that remained consistent with both GDA and Factor Analysis to see how well the algorithms performed. As we present the results, we will go into more detail as to the implementation of the data sets. For complete detail, please see Appendix C, (see page 12).

4 Conclusion

As the results are best interpreted in light of conclusions, we present the conclusions first. It seems that FactorpPCA is both faster *per iteration* and more accurate. It seems to fare better with a convergence test based on likelihood rather then on global maxima on the matrices, as those may be scaled or altered but have similar predictive capabilities. Initialization appears to be important, as indicated by Machine, in that Ψ should be initialized to have similar magnitude to the data sets. Otherwise, subtracting 1 tends to get swamped, and we get 0's in the diagonal. The algorithm as a result no longer guarantees convergence, and likelihood, as predicted, does not increase with every iteration. Nevertheless, this looks to be a promising algorithm and there seems to be several mechanisms that can be used to fix these problems. First, the data set can be normalized. (This wasn't tried, but this may resolve problem of having the data sets of varying magnitudes.) Second, there could be a further refinement of the convergence criteria made. As there is far more plots and data then can fit in 5 pages, the secondary results and graphs are reproduced in the appendix.

5 Results

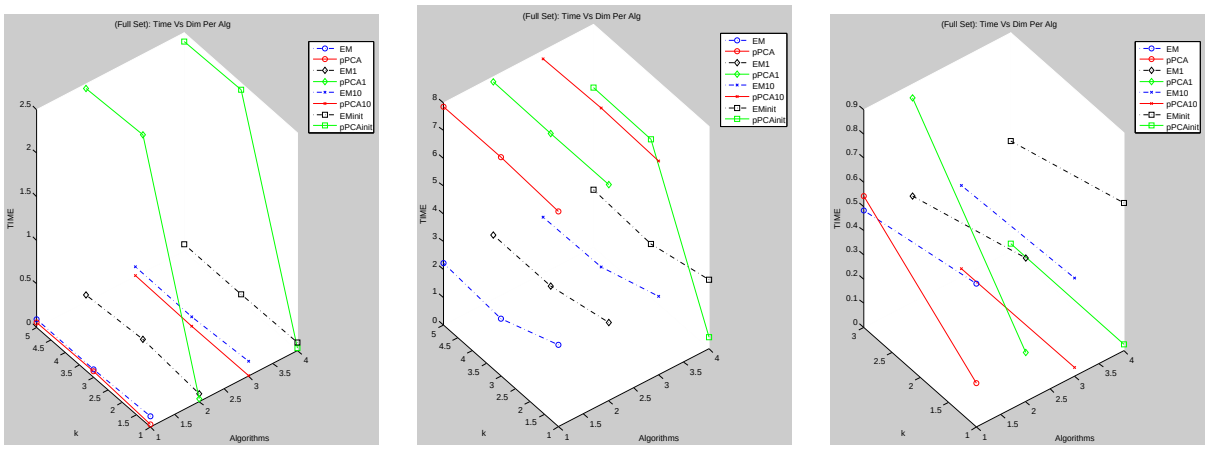
The following graphs are 3-d band graphs. EM or pPCA designates the standard algorithm. If the name is followed by a 1, the algorithm is run with Psi initialized to 0.1 * I. With 10, it means it was multiplied by 10. With "init", this means it was initialized with the results of the other algorithm. The following graphs were generated with a convergence test on Ψ and a cap of number of iterations of 1000.



(a) (b) (c)

Figure 1: Iteration Figures- Random, Machine, Balance

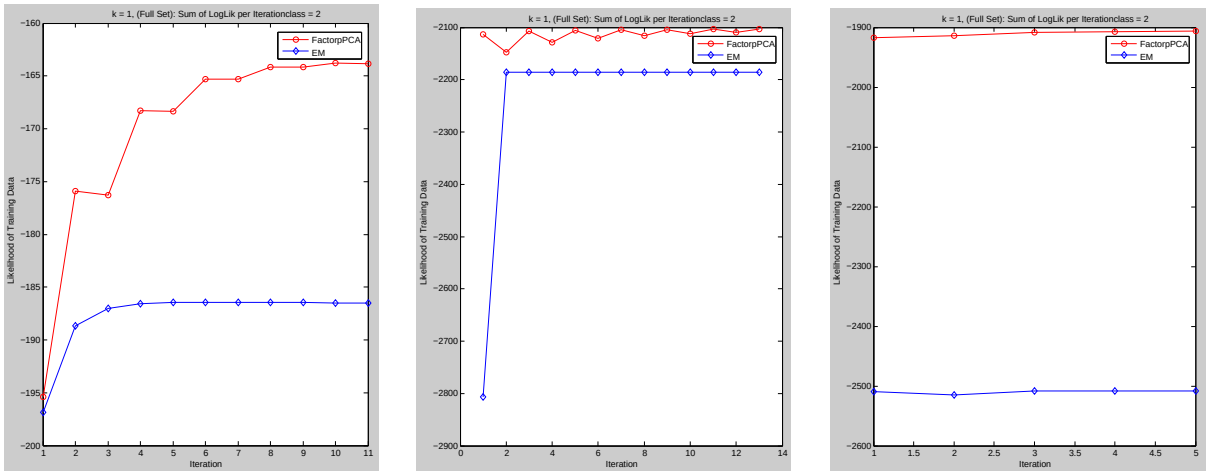
Only three data sets are presented. The results of votes are included in Appendix E (see page 25). Here, we clearly note that machine failed to converge within 1000 iterations, while it converged in all other cases but one.



(a) (b) (c)

Figure 2: Time Figures- Random, Machine, Balance

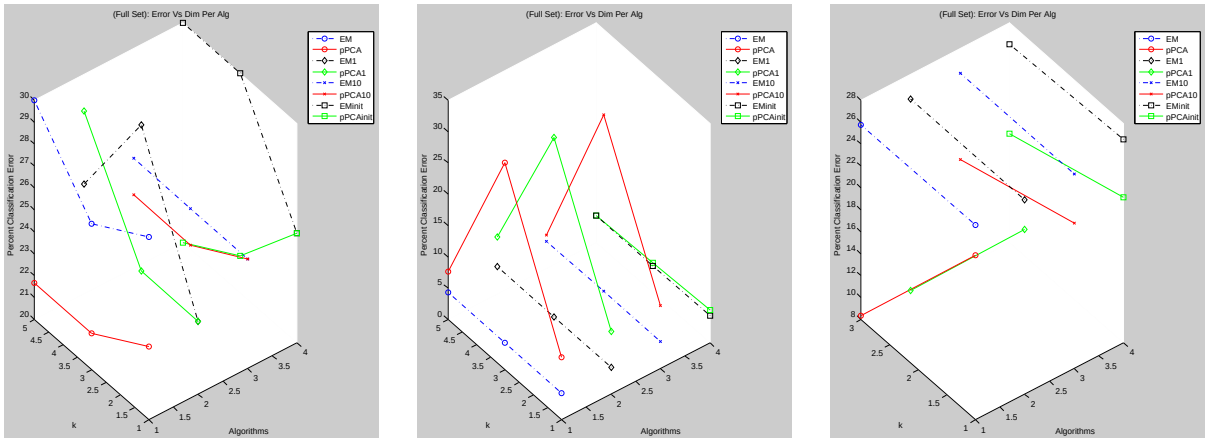
It is clear that FactorpPCA outerperfomed FactorEM except when it failed to converge. Moreover, it is interesting to note that in many cases, FactorpPCA required more iterations to converge, and yet, still managed to outperform FactorEM. Thus, it was clear we needed to analyze our convergence criteria.



(a) (b) (c)

Figure 3: Likelihood Figures-Random, Machine, Balance

Both EM, pPCA stagnates well before we terminate the algorithm and becomes cleared by looking at the graphs in Appendix E. As a result, its possible (and likely) that the convergence on Psi is overrestricting and does not increase predictive ability by any significant amount.



(a) (b) (c)

Figure 4: Error Figures- Random, Machine, Balance

While speed is important, it is important to analyze classification error. Test error is presented in the appendix, as well as comparison against GDA. These graphs are of training error on the full set and indicate that FactorpPCA is clearly the winner here. Thus, FactorpPCA is a promising algorithm, but is susceptible to initialization, magnitude of the data (machine), and may require a more refined convergence test based on likelihood rather then on maxima of Psi. The first figures in the appendix test this convergence criteria. This concludes our project.

6 Appendix A- Basic Models

6.1 Basic Factor Analysis Model

Factor analysis assumes $x \in \Re^n$ is an observed random variable according to the following model:

$$\begin{aligned} z &\sim N(0, I) \\ x|z &\sim N(\mu + \Lambda z, \Psi) \end{aligned}$$

where $z \in \Re^k$ is the latent random variable.[1]

The parameters are: $\mu \in \Re^n$, $\Lambda \in \Re^{n \times k}$, and $\Psi = \text{diag}(\sigma_1^2 \dots \sigma_n^2) \in \Re^{n \times n}$.
The joint distribution for (z, x) is:

$$\begin{bmatrix} z \\ x \end{bmatrix} \sim N\left(\begin{bmatrix} 0 \\ \mu \end{bmatrix}, \begin{bmatrix} I & \Lambda^T \\ \Lambda & \Lambda\Lambda^T + \Psi \end{bmatrix} \right)$$

An important note is that $\Lambda\Lambda^T + \Psi$ must be symmetric positive semidefinite, as it is the sum of the "square" of a matrix and a covariance matrix.

Given a training set $\{x^{(i)}\}_{i=1}^m$, the log-likelihood of the parameters is:

$$\begin{aligned} \ell(\mu, \Lambda, \Psi) &= C - m \log |\Lambda\Lambda^T + \Psi| - \frac{1}{2} \sum_{i=1}^m (x^{(i)} - \mu)^T (\Lambda\Lambda^T + \Psi)^{-1} (x^{(i)} - \mu) \\ &= C - m \log |\Lambda\Lambda^T + \Psi| - \frac{(\Lambda\Lambda^T + \Psi)^{-1}}{2} \frac{1}{m} \sum_{i=1}^m (x^{(i)} - \mu)(x^{(i)} - \mu)^T \end{aligned}$$

where C is constant.

The ML estimator for μ is simply given by the mean of the data and can be calculated once:

$$\mu = \frac{1}{m} \sum_{i=1}^m x^{(i)}$$

Thus, Factor Analysis seeks to solve:

$$\textbf{argmax}_{\Lambda, \Psi} \underbrace{\left(-\log |\Lambda\Lambda^T + \Psi| - \text{tr}((\Lambda\Lambda^T + \Psi)^{-1} \frac{\sum_{i=1}^m (x^{(i)} - \mu)(x^{(i)} - \mu)^T}{m}) \right)}_{L} \tag{1}$$

6.2 Factor EM Model

As this is our primary model to test against, its important to restate the results:
The log likelihood of the model is as follows:

$$l(\mu, \Lambda, \Psi) = \log \prod_{i=1}^m \frac{1}{2\pi^{\frac{n}{2}} |\Lambda\Lambda^T + \Psi|} e^{\left(-\frac{1}{2}(x^{(i)} - \mu)^T (\Lambda\Lambda^T + \Psi)^{-1} (x^{(i)} - \mu)\right)}$$

The E-step is as follows:

[1] Note $k > n$ yields the trivial optimal model by setting $\Lambda = I_{n \times n} | 0_{n \times (k-n)}$.

$$\begin{aligned}\mu_{z^{(i)}|x^{(i)}} &= \Lambda^T(\Lambda\Lambda^T+\Psi)^{-1}(x^{(i)}-\mu),\\ \Sigma_{z^{(i)}|x^{(i)}} &= I-\Lambda^T(\Lambda\Lambda^T+\Psi)^{-1}\Lambda\end{aligned}$$

The M-step is as follows:

$$\begin{aligned}\Lambda &= \left(\sum_{i=1}^m (x^{(i)}-\mu)\mu_{z^{(i)}|x^{(i)}}^T\right)\left(\sum_{i=1}^m \mu_{z^{(i)}|x^{(i)}}\mu_{z^{(i)}|x^{(i)}}^T+\Sigma_{z^{(i)}|x^{(i)}}\right)^{-1}\\ \Phi &= \frac{1}{m}\sum_{i=1}^m x^{(i)}x^{(i)T}-x^{(i)}\mu_{z^{(i)}|x^{(i)}}^T\Lambda^T-\Lambda\mu_{z^{(i)}|x^{(i)}}^T x^{(i)T}+\Lambda\mu_{z^{(i)}|x^{(i)}}\mu_{z^{(i)}|x^{(i)}T}+\Sigma_{z^{(i)}|x^{(i)}}\Lambda^T,\\ \Psi_{ii} &= \Phi_{ii},\\ \Psi_{ij} &= 0, \text{ if } i\neq j\end{aligned}$$

6.3 Gaussian Discriminant Analysis Model

This model has parameters $\phi_j = P(y=j)$, μ_j, and Σ. The MLE estimates are:

$$\begin{aligned}\phi_j &= \frac{1}{m}\sum_{i=1}^m I\{y^{(i)}=j\},\\ \mu_j &= \frac{\sum_{i=1}^m Iy^{(i)}=jx^{(i)}}{\sum_{i=1}^m Iy^{(i)}=j},\\ \Sigma &= \frac{1}{m}\sum_{i=1}^m (x^{(i)}-\mu_{y^{(i)}})(x^{(i)}-\mu_{y^{(i)}})^T\end{aligned}$$

7 Appendix B- Derivation of Factor pPCA

7.1 Maximizing w.r.t. Λ with Ψ fixed

For the sake of brevity, let $\Sigma = \frac{\sum_{i=1}^{m}(x^{(i)}-\mu)(x^{(i)}-\mu)^T}{m}$.

Therefore,

$$\nabla_\Lambda \text{Ł} = -2\Lambda^T(\Lambda\Lambda^T+\Psi)^{-T} + 2\Lambda^T(\Lambda\Lambda^T+\Psi)^{-T}\Sigma(\Lambda\Lambda^T+\Psi)^{-T}$$

Setting the gradient to zero and taking transposes, the extreme values of Λ must satisfy:

$$\Lambda = \Sigma(\Lambda\Lambda^T+\Psi)^{-1}\Lambda \tag{2}$$

Lets only the consider when $\Lambda \neq 0$ and $\Lambda\Lambda^T + \Psi \neq \Sigma$ as those cases trivially satisfy the above constraint. The two other cases are considered at the end.

Assuming Ψ is non-singular[2], its singular value decomposition exists and is given by $\Psi^{\frac{1}{2}}\Lambda = ULV^T$, where $U = (u_1, u_2, \ldots, u_k) \in \Re^{n\times k}$ is an column orthonormal matrix, $L = [l_1, l_2, \ldots, l_k]$ I $\in \Re^{\text{k}\times\text{k}}$ is a diagonal matrix of singular values, and $V \in \Re^{k\times k}$ is an orthogonal matrix. Thus, we have the decomposition $\Lambda = \Psi^{\frac{1}{2}}ULV^T$.

Substituting this into Eqn. 2 for Λ yields:

$$\Sigma\Psi^{-\frac{1}{2}}(UL^2U^T+I)^{-1}UL \;=\; \Psi^{\frac{1}{2}}UL \tag{3}$$

If $l_i = 0$ for some $i \in \{1, \ldots, k\}$, then the solution for U can have an arbitrary column u_i. Thus, reduce the system to only the constrained dimensions and invert L[3].

$$\begin{aligned}
& \Sigma\Psi^{-\frac{1}{2}}(UL^2U^T+I)^{-1}UL = \Psi^{\frac{1}{2}}UL \\
\implies\quad & \Sigma\Psi^{-\frac{1}{2}}(UL^2U^T+I)^{-1}U = \Psi^{\frac{1}{2}}U \\
\implies\quad & \Sigma\Psi^{-\frac{1}{2}}U(I-Q) = \Psi^{\frac{1}{2}}U \\
\implies\quad & \Psi^{-\frac{1}{2}}\Sigma\Psi^{-\frac{1}{2}}U(I-Q) = U \\
\implies\quad & \Psi^{-\frac{1}{2}}\Sigma\Psi^{-\frac{1}{2}}U = U(I+L^2)
\end{aligned}$$

$$\text{where } Q = \text{diag}(q_1, \ldots, q_k) \in \Re^{k\times k} \text{ with each } q_i = l_i^2/(1+l_i^2).$$

Step 2 to 3 uses the matrix inversion lemma to yield: $(UL^2U^T+I)^{-1} = I - UQU^T$. The last step follows from the observation that $q_i = l_i^2/(1+l_i^2)$ the i^{th} diagonal entry of $I-Q$ must be $1/(1+l_i^2)$.

For any $l_i \neq 0$, this equation can be written in terms of the column u_i as:

$$\Psi^{-\frac{1}{2}}\Sigma\Psi^{-\frac{1}{2}}u_i \;=\; (1+l_i^2)u_i$$

Thus, u_i must be an eigenvector of $\Psi^{-\frac{1}{2}}\Sigma\Psi^{-\frac{1}{2}}$.
Denoting the corresponding eigenvalue by λ_i, this implies $l_i = \sqrt{\lambda_i - 1}$ which constrains $\lambda_i > 1$.

[2]The assumption that Ψ is non-singular amounts to, within the factor analysis model, that the diagonal entries are $\neq 0$. That is, the normally distributed error has non-zero variance along every dimension of X.

[3]Note that reducing L did not adversely affect the solution as we can permute the rows and columns back by a linear operator, then use block matrix notation to invert and consider only the segments we are interested in.

Therefore,

- V can be any $k \times k$ orthogonal matrix. So let V = I.
- U and L are chosen together, column by column. Each column u_i and diagonal entry l_i should be picked using one of the following choices:
  Choice 1: $l_i = 0$; Arbitrary $u_i \in \Re^n$
  Choice 2: $l_i = \sqrt{\lambda_i - 1}$; u_i corresponding eigenvector to eigenvalue λ_i.

7.2 The global maximum

Without loss of generality, let $l_1, l_2, \ldots, l_t$ represent the nonzero values picked using Choice 2 above, and let $\lambda_{a_1}, \lambda_{a_2}, \ldots, \lambda_{a_t} > 1$ represent the corresponding eigenvalues of $\Psi^{-\frac{1}{2}}\Sigma\Psi^{-\frac{1}{2}}$. Furthermore, let $U_{1:t} \in \Re^{n\times t}$ contain the t columns of U picked using Choice 2 above and let $L_{1:t} = [l_1, \ldots, l_t]$ I $\in \Re^{\mathrm{t}\times\mathrm{t}}$. Using Choice 1, $l_{t+1}, \ldots, l_k = 0$ with undetermined corresponding eigenvalues.

The eigenvalues of $\Psi^{\frac{1}{2}}$ are simply $\sqrt{\sigma_i^2}$ for $i \in \{1, 2, \ldots, n\}$.
The eigenvalues of $U_{1:t}L_{1:t}^2U_{1:t}^T$ are exactly the diagonal entries of $L_{1:t}^2$. [4]
Then,

$$\begin{aligned}
\Lambda &= \Psi^{\frac{1}{2}} \left(\begin{array}{c|c} U_{1:t}L_{1:t} & 0 \end{array} \right) V^T \quad \text{(Block matrix multiplication)} \\
\implies \quad \Lambda\Lambda^T + \Psi &= \Psi^{\frac{1}{2}}(U_{1:t}L_{1:t}^2U_{1:t}^T + I)\Psi^{\frac{1}{2}} \\
\implies \quad \log|\Lambda\Lambda^T + \Psi| &= \log\left(\prod_{i=1}^{n}\sigma_i^2 \cdot \prod_{i=1}^{t}\lambda_{a_i}\right)
\end{aligned}$$

Since the eigenvalues of $(U_{1:t}L_{1:t}^2U_{1:t} + I)$ are the t values $\{\lambda_{a_1}, \lambda_{a_2}, \ldots, \lambda_{a_t}\}$ and $(n-t)$ 1's,

$$\begin{aligned}
\mathrm{tr}((\Lambda\Lambda^T + \Psi)^{-1}S) &= \mathrm{tr}((U_{1:t}L_{1:t}^2U_{1:t} + I)^{-1}\tilde{S}) \\
&= \sum_{i=1}^{t} 1 + \sum_{i=t+1}^{n} \lambda_{a_i} \\
\implies Ł &= -\log|\Lambda\Lambda^T + \Psi| - \mathrm{tr}(\Lambda\Lambda^T + \Psi^{-1}\Sigma) \\
&= -\left(n\log\sigma_i^2 + \sum_{i=1}^{t}\log\lambda_{a_i} + t + \sum_{i=t+1}^{n}\lambda_{a_i}\right)
\end{aligned}$$

We therefore need to minimize

$$-Ł = n\log\sigma_i^2 + \sum_{i=1}^{t}(1 + \log\lambda_{a_i}) + \sum_{i=t+1}^{n}\lambda_{a_i} \tag{4}$$

Since the following holds,

- $\lambda_{a_i} > 1 \qquad \forall i \in \{1, 2, \ldots, t\}$
- $1 + \log(x) < x \qquad \forall x > 1$
- $f(x) = x - (1 + \log x)$ is an increasing function of x for $x > 1$.

[4] since this is explicitly in the form of an eigenvector decomposition.

Eqn. 4 is minimized over $\{a_i\}$ and t by picking the largest t eigenvalues as $\lambda_{a_1}, \ldots, \lambda_{a_t}$ and also picking t as high as possible:

$$t = \min(k, \text{ number of eigenvalues of } \tilde{S} > 1).$$

7.3 Maximizing w.r.t. Ψ with Λ fixed

Compute the gradient of Ł w.r.t. Ψ:

$$\nabla_\Psi Ł \;=\; -(\Lambda\Lambda^T + \Psi)^{-1} + (\Lambda\Lambda^T + \Psi)^{-1}\Sigma(\Lambda\Lambda^T + \Psi)^{-1}$$

Setting the gradient to zero, the extreme values of Ψ must satisfy $\forall i, j,\ i \neq j$:

$$\begin{aligned} ((\Lambda\Lambda^T + \Psi)^{-1}\Sigma)_{ii} &= 1 \\ \implies \qquad \Psi_{ii} &= (\Sigma - \Lambda\Lambda^T)_{ii} \\ \Psi_{ij} &= 0 \end{aligned}$$

7.4 Cases

Case 1: $\Lambda = 0$. This is a minimum of Ł.
Case 2: $\Lambda\Lambda^T + \Psi = \Sigma$. This means $\Lambda\Lambda^T = \Sigma - \Psi$, which has a solution when up to k eigenvalues of $S - \Psi$ are nonnegative, and the rest are zero (i.e., $S - \Psi$ must be positive semidefinite, meaning all eigenvalues are nonnegative, but further conditions are required to guarantee a rectangular square root. The solution is given in its SVD form as $\Lambda = ULV^T$ where $U = (u_1, \ldots, u_k) \in \Re^{n\times k}$ has eigenvectors of $S - \Psi$ in its columns, $L \in \Re^{k\times k}$ is a diagonal matrix with the square roots of the corresponding eigenvalues on the diagonal, and $V \in \Re^{k\times k}$ is an arbitrary orthogonal matrix.[5] This case seems unlikely in real data because it requires, for example, the sample covariance S to be of the form $S =$ (rank k matrix + diagonal matrix).

[5]For an explanation, refer to the definition of the square-root of a symmetric positive definite matrix. The major difference in our case is that Λ is rectangular, whereas the standard definition refers to a square matrix as the square-root. The bridge follows from the following two facts: first, the regular square matrix square-root can be obtained by appending $(n - k)$ zero columns to the rectangular Λ; second, we can have a rectangular square root (only) because of the condition on zero eigenvalues of $S - \Psi$, namely, that this latter matrix has rank at most k for a rectangular decomposition of this form.

8 Appendix C- Data Sets

8.1 Random

Random Data was generated using a natural dimensions of 2. Various higher dimensions and number of training samples were used, with results being consistent across any such variation. So the results that were presented were with a total training sample of 60, and n = 5. This allowed us to have at least one data set that terminated relatively quickly, allowing us to generate quick results.

8.2 Voting

Below is the voting data set information as taken from the names section of the repository.

```
Relevant Information:
      This data set includes votes for each of the U.S. House of
      Representatives Congressmen on the 16 key votes identified by the
      CQA.  The CQA lists nine different types of votes: voted for, paired
      for, and announced for (these three simplified to yea), voted
      against, paired against, and announced against (these three
      simplified to nay), voted present, voted present to avoid conflict
      of interest, and did not vote or otherwise make a position known
      (these three simplified to an unknown disposition).

5. Number of Instances: 435 (267 democrats, 168 republicans)
```

Nays were treated as -1, Yea's as 1, Abstaining as 0, and the goal was to predict, based on voting record, was someone a democrat or a republican.

8.3 Machine

Below is the machine data set information as taken from the names section of the repository.

```
7. Attribute Information:
   1. vendor name: 30
      (adviser, amdahl,apollo, basf, bti, burroughs, c.r.d, cambex, cdc, dec,
       dg, formation, four-phase, gould, honeywell, hp, ibm, ipl, magnuson,
       microdata, nas, ncr, nixdorf, perkin-elmer, prime, siemens, sperry,
       sratus, wang)
   2. Model Name: many unique symbols
   3. MYCT: machine cycle time in nanoseconds (integer)
   4. MMIN: minimum main memory in kilobytes (integer)
   5. MMAX: maximum main memory in kilobytes (integer)
   6. CACH: cache memory in kilobytes (integer)
   7. CHMIN: minimum channels in units (integer)
   8. CHMAX: maximum channels in units (integer)
   9. PRP: published relative performance (integer)
  10. ERP: estimated relative performance from the original article (integer)
```

As there were 8 different classes formed by binning, and we needed results per class, we decided to combine these into 2 classes– 1 above the mean of the global data set, and 1 below. We then tested if it was above the mean or below.

8.4 Balance

Below is the balance data set information as taken from the names section of the repository.

```
4. Relevant Information:
   This data set was generated to model psychological
   experimental results.  Each example is classified as having the
   balance scale tip to the right, tip to the left, or be
   balanced.  The attributes are the left weight, the left
   distance, the right weight, and the right distance.  The
   correct way to find the class is the greater of
   (left-distance * left-weight) and (right-distance *
   right-weight).  If they are equal, it is balanced.

5. Number of Instances: 625 (49 balanced, 288 left, 288 right)
```

The classes 'L', 'B', and 'R', were converted directly into ascii representation. As the classes are not used in anyway in the calculation except as denotation, it doesn't matter – as long as they are distinct. The goal was to predict the correct class.

9 Appendix D- Code

This section contains only the four relevant scripts used to ascertain our results. Nearly 24 scripts were written that execute 5 different variations of the algorithms discussed. They will be provided if requested.

9.1 Basic Factor EM

Below is the code for the basic (unoptimized) factor EM algorithm.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% CS 229 Factor Analysis / EM
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Note: This problem asssumes each column is a training sample.

function [mu, Lambda, Psi, lik, numiter] = factorEM(X, k, Psi,
Lambda)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Defining Parameters we need for E-M Factor Analysis
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Parameters For E-M Algorithm
[n m] = size(X);                    % Dimensions of X
Mu = zeros(k,m);                    % For each dimension, for each sample
Sigma = eye(k);                     % A single n x n covariance matrix
I = eye(k);                         % K x K identity
error = 0.01;                       % Error theshold

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Defining our predicitions for E-M Factor Analysis
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%What we are trying to predict
mu = mean(X, 2);                     % n x 1

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%E-M Factor Analysis
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
numiter = 0; PsiOld = Psi + 2 * error; while ((max(abs(diag(Psi -
PsiOld))) > error) && (numiter < 1000))
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Storing Old Value of Psi
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    PsiOld = Psi;
    numiter = numiter + 1;

    %%%%%%%%%%%%%%%%%%
```

```
    % E-Step
    %%%%%%%%%%%%%%%%%%
    for i = 1:m
        Mu(:,i) = Lambda' * inv(Lambda * Lambda' + Psi) *(X(:,i) - mu);
    end
    Sigma = I - Lambda' * inv(Lambda * Lambda' + Psi)  * Lambda;

    %%%%%%%%%%%%%%%%%%
    % M-Step
    %%%%%%%%%%%%%%%%%%
    term1 = 0; term2 = 0;
    for i = 1:m
        term1 = term1 + (X(:,i) - mu) * Mu(:,i)';
        term2 = term2 + Mu(:,i) * Mu(:,i)' + Sigma;
    end
    LambdaNew = term1 * inv(term2);

    Phi = 0;
    for i = 1:m
        Phi = Phi + X(:,i) * X(:,i)' - X(:,i) * Mu(:,i)' * Lambda' - Lambda * Mu(:,i) * X(:
    end
    Psi = diag(diag(Phi/m));
    Lambda = LambdaNew;
    [l, lik(numiter)] = factorLik(X, mu, Lambda * Lambda' + Psi);
end
```

9.2 FactorpPCA with Likelihood convergence test

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% CS 229 Factor Analysis / pPCA
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Note: This problem asssumes each column is a training sample.

function [mu, Lambda, Psi, lik, numiter] = factorpPCA(X, k, Psi)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Defining Parameters we need for pPCA Factor Analysis
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Parameters For pPCA
[n m] = size(X);
I = eye(n);                           % n x n identity
error = 0.1;                         % Error theshold

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Defining predictions for pPCA Factor Analysis
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
mu = mean(X, 2);                        % n x 1
Lambda = zeros(n,k);                    %Just so we know we are "predicting it"

%Calculate sample variance and "old Psi"
S = 0; Xminusmu = X - mu*ones(1,m); for i = 1:m
   S = S + Xminusmu(:,i) * Xminusmu(:,i)';
end
S = S / m;              %Sample Variance
PsiOld = Psi - 2 * error;   %To Store old Psi Value

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%pPCA Factor Analysis
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
numiter = 0; likNew = 1; likOld = 0;   %Setting initial paramters
while((max(abs(likNew - likOld)) > error) && (numiter < 1000))

    %To remove unnecesary repeated matrix computation
    numiter = numiter + 1;
    PsiSqrt = Psi^0.5; PsiNegSqrt = inv(PsiSqrt);

    %Get the largest number of valid eigenvectors/values
    [v, d] =  eig(PsiNegSqrt * S * PsiNegSqrt); %Get eig vectors/values
    diagD = diag(d);                            %Convert d into a vector of lambdas
    t = min(k, sum(diagD > 1));                 %Find the valid t
    [sorted, indices] = sort(diagD, 'descend'); %Find indices with largest eig 1st
    validLambda = indices(1:t);                 %Get indices of t largest Lambda's
    L = diag(diagD(validLambda) - 1)^0.5;       %Create L for valid Lambda's
    U = v(:, validLambda);              %Create U for valid Lambda's

    %Storing old value of Psi, and calculating Lambda/Psi
    PsiOld = Psi;
    Psi = diag(diag(S - Lambda * Lambda'));
    Lambda = PsiSqrt * cat(2, U * L, zeros(n, k - t));
    [l, lik(numiter)] = factorLik(X, mu, Lambda * Lambda' + Psi);

    %Convergence Test
    if numiter > 1
        likOld = lik(numiter - 1);
        likNew = lik(numiter);
    end
end
```

9.3 Test Code

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% CS 229 Classification test
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
% Note: This problem places assumes each column is a training sample.
% This also "KNOWS" that factorEM and factorpPCA have a cap of 1000

function [] = test(X, y, t)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% As I didn't want to restrict this to a particular prespecified dimension,
% I go for each possible classifictaion, caluclate the values, and then
% also calculate the likihood of that value, and classify it accordingly.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Comparing FactorEM to FactorpPCA on Real Data Sets
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%General Paramaters
class = unique(y);                          %getting classifications
numClass = length(class);                   %Number of classes
[n m] = size(X);                            %numDimensions, numSamples
k = 1:2:n;                                  %Define dimensions to consider

%Iteration Paramters
pPCAiter = zeros(1, length(k));             %0 zero average iterations to date
pPCA10iter = zeros(1, length(k));             %0 zero average iterations to date
pPCA1iter = zeros(1, length(k));            %0 zero average iterations to date
pPCAinititer = zeros(1, length(k));            %0 zero average iterations to date
EMiter = zeros(1,length(k));                %0 zero average iterations to date
EM10iter = zeros(1,length(k));                %0 zero average iterations to date
EM1iter = zeros(1,length(k));               %0 zero average iterations to date
EMinititer = zeros(1,length(k));               %0 zero average iterations to date

%Time Paramters
pPCAtime = zeros(1, length(k));             %0 average time used so far
pPCA10time = zeros(1, length(k));             %0 average time used so far
pPCA1time = zeros(1, length(k));            %0 average time used so far
pPCAinittime = zeros(1, length(k));            %0 average time used so far
EMtime = zeros(1, length(k));               %0 average time used so far
EM10time = zeros(1, length(k));               %0 average time used so far
EM1time = zeros(1, length(k));              %0 average time used so far
EMinittime = zeros(1, length(k));              %0 average time used so far

%Likelihood Parameters
pPCAlik = [];                               %Initializing to empty
pPCA10lik = [];                               %Initializing to empty
pPCA1lik = [];                              %Initializing to empty
pPCAinitlik = [];                              %Initializing to empty
pPCATestlik = [];                              %Initializing to empty
EMlik = [];                                 %Initializing to empty
```

```
EM10lik = [];                                  %Initializing to empty
EM1lik = [];                                  %Initializing to empty
EMinitlik = [];                                  %Initializing to empty
EMTestlik = [];                                  %Initializing to empty

%Likelihood Array Parameters
PlikIter = ones(length(k), 1001, numClass);  %Likelihood array
ElikIter = ones(length(k), 1001, numClass);  %Lkelihood array

%Parameters for Test Error
indices = randperm(m);                          %Random Permutation of Indices
numTrain = round(m * 0.7);                      %Number in Training Sample
XTrain = X(:, indices(1:numTrain));             %Pick 1st 70% of random index
yTrain = y(indices(1:numTrain));
XTest = X(:, indices((numTrain+1):m));           %Pick last 30% of random index
yTest = y(indices((numTrain+1):m));

%Executing Test for each dimension, classification, test/training, etc.
for i = 1:length(k)                           %For each dimension
    for j = 1:numClass                        %For each potential classification

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % Training Error
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

        %FactorpPCA
        t = cputime;                                         %Get Current time and execute
        [mu, Lambda, Psi, lik, iter] = factorpPCA(X(:, y == class(j)),k(i), eye(n));
        pPCAtime(i) = pPCAtime(i) + (cputime - t)/numClass;  %Getting time difference
        pPCAiter(i) = pPCAiter(i) + iter/numClass;           %Summing number of iterations
        [pPCAlik(:,j), l] = factorLik(X, mu, Lambda * Lambda' + Psi);  %lik(i,j) = loglik o
        PlikIter(i, 1:iter, j) = lik;                        %Storing likelihood

        %Factor EM with FactopPCA initialization
        t = cputime;                                         %Get Current time and execute
        [mu, Lambda, Psi, lik, iter] = factorEM(X(:, y == class(j)),k(i), Psi, Lambda);
        EMinittime(i) = EMinittime(i) + (cputime - t);               %Getting time differen
        EMinititer(i) = EMinititer(i) + iter/numClass;               %Summing number of ite
        [EMinitlik(:,j), l] = factorLik(X, mu, Lambda * Lambda' + Psi); %lik(i,j) = prob of

        %FactorppCA with Psi * 10
        t = cputime;                                         %Get Current time and execute
        [mu, Lambda, Psi, lik, iter] = factorpPCA(X(:, y == class(j)),k(i), eye(n) * 10);
        pPCA10time(i) = pPCA10time(i) + (cputime - t)/numClass;  %Getting time difference
        pPCA10iter(i) = pPCA10iter(i) + iter/numClass;           %Summing number of iterati
        [pPCA10lik(:,j), l] = factorLik(X, mu, Lambda * Lambda' + Psi);  %lik(i,j) = loglik

        %FactorppCA with Psi * 0.1
```

```
    t = cputime;                                                    %Get Current time and execute
    [mu, Lambda, Psi, lik, iter] = factorpPCA(X(:, y == class(j)),k(i), eye(n) * 0.1);
    pPCA1time(i) = pPCA1time(i) + (cputime - t)/numClass;  %Getting time difference
    pPCA1iter(i) = pPCA1iter(i) + iter/numClass;           %Summing number of iterations
    [pPCA1lik(:,j), l] = factorLik(X, mu, Lambda * Lambda' + Psi);  %lik(i,j) = loglik o

    %FactorEM
    t = cputime;                                                    %Get Current time and execute
    [mu, Lambda, Psi, lik, iter] = factorEM(X(:, y == class(j)),k(i), eye(n), rand(n,k(i
    EMtime(i) = EMtime(i) + (cputime - t);                         %Getting time difference
    EMiter(i) = EMiter(i) + iter/numClass;                         %Summing number of iterations
    [EMlik(:,j), l] = factorLik(X, mu, Lambda * Lambda' + Psi); %lik(i,j) = prob of ith
    ElikIter(i, 1:iter, j) = lik;                                  %Storing likelihood

    %FactorpPCA with EM initilization
    t = cputime;                                                    %Get Current time and execute
    [mu, Lambda, Psi, lik, iter] = factorpPCA(X(:, y == class(j)),k(i), Psi);
    pPCAinittime(i) = pPCAinittime(i) + (cputime - t)/numClass;  %Getting time differen
    pPCAinititer(i) = pPCAinititer(i) + iter/numClass;           %Summing number of iter
    [pPCAinitlik(:,j), l] = factorLik(X, mu, Lambda * Lambda' + Psi);  %lik(i,j) = logli

    %FactorEM with Psi * 10
    t = cputime;                                                    %Get Current time and execute
    [mu, Lambda, Psi, lik, iter] = factorEM(X(:, y == class(j)),k(i), eye(n) * 10, rand
    EM10time(i) = EM10time(i) + (cputime - t);                     %Getting time difference
    EM10iter(i) = EM10iter(i) + iter/numClass;                     %Summing number of iteratio
    [EM10lik(:,j), l] = factorLik(X, mu, Lambda * Lambda' + Psi); %lik(i,j) = prob of it

    %FactorEM with Psi * 0.1
    t = cputime;                                                    %Get Current time and execute
    [mu, Lambda, Psi, lik, iter] = factorEM(X(:, y == class(j)),k(i), eye(n) * 0.1, ranc
    EM1time(i) = EM1time(i) + (cputime - t);                       %Getting time difference
    EM1iter(i) = EM1iter(i) + iter/numClass;                       %Summing number of iterations
    [EM1lik(:,j), l] = factorLik(X, mu, Lambda * Lambda' + Psi); %lik(i,j) = prob of ith

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Training Versus Test
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    [mu, Lambda, Psi, lik, iter] = factorEM(XTrain(:, yTrain == class(j)),k(i), eye(n),
    [EMTestlik(:,j), l] = factorLik(XTest, mu, Lambda * Lambda' + Psi); %lik(i,j) = prob
    [mu, Lambda, Psi, lik, iter] = factorpPCA(XTrain(:, yTrain == class(j)),k(i), eye(n)
    [pPCATestlik(:,j), l] = factorLik(XTest, mu, Lambda * Lambda' + Psi); %lik(i,j) = pr
end

%Classification error on training samples
pPCAerror(i) = classifyError(pPCAlik, y);
pPCA1error(i) = classifyError(pPCA1lik, y);
pPCA10error(i) = classifyError(pPCA10lik, y);
```

```
    pPCAiniterror(i) = classifyError(pPCAinitlik, y);
    pPCATesterror(i) = classifyError(pPCATestlik, yTest);
    EMerror(i) = classifyError(EMlik, y);
    EM1error(i) = classifyError(EM1lik, y);
    EM10error(i) = classifyError(EM10lik, y);
    EMiniterror(i) = classifyError(EMinitlik, y);
    EMTesterror(i) = classifyError(EMTestlik, yTest);
end

%Computing GDA
[GDAerror, mu, Sigma] = GDA(XTrain, yTrain, XTest, yTest); GDAerror
= GDAerror * ones(1, length(k));

%Plots
X = ones(1,length(k)); %Defining X axess of plots

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% UnNormalized
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%Plotting Time it takes
hold off figure(1) plot3(X, k, EMtime, '-.bo', X, k, pPCAtime,
'-ro',  X + 1, k, EM1time, '-.kd', X + 1, k, pPCA1time, '-gd'); hold
on plot3(X + 2, k, EM10time, '-.bx', X + 2, k, pPCA10time, '-rx', X
+ 3, k, EMinittime, '-.ks', X + 3, k, pPCAinittime, '-gs');
xlabel('Algorithms'); ylabel('k'); zlabel('TIME'); title('(Full
Set): Time Vs Dim Per Alg'); legend('EM', 'pPCA', 'EM1', 'pPCA1',
'EM10', 'pPCA10', 'EMinit', 'pPCAinit'); view(3);

%Plotting number of iterations
hold off figure(2) plot3(X, k, EMiter, '-.bo', X, k, pPCAiter,
'-ro', X + 1, k, EM1iter, '-.kd', X + 1, k, pPCA1iter, '-gd'); hold
on plot3(X + 2, k, EM10iter,  '-.bx', X + 2, k, pPCA10iter, '-rx', X
+ 3, k, EMinititer, '-.ks', X + 3, k, pPCAinititer, '-gs');
xlabel('Algorithms'); ylabel('k'); zlabel('Number of Iterations');
title('(Full Set): # Iter Vs Dim Per Alg'); legend('EM', 'pPCA',
'EM1', 'pPCA1', 'EM10', 'pPCA10', 'EMinit', 'pPCAinit'); view(3);

%Plotting error per dimension
hold off figure(3) plot3(X, k, EMerror, '-.bo', X, k, pPCAerror,
'-ro', X + 1, k, EM1error, '-.kd',  X + 1, k, pPCA1error, '-gd');
hold on plot3(X + 2, k, EM10error, '-.bx', X + 2, k, pPCA10error,
'-rx',  X + 3, k, EMiniterror, '-.ks', X + 3, k, pPCAiniterror,
'-gs'); xlabel('Algorithms'); ylabel('k'); zlabel('Percent
Classification Error'); title('(Full Set): Error Vs Dim Per Alg');
legend('EM', 'pPCA', 'EM1', 'pPCA1', 'EM10', 'pPCA10', 'EMinit',
'pPCAinit'); view(3);
```

```
%Plotting Test Error
hold off figure(4) plot(k, GDAerror, '--kx', k, EMTesterror, '-.bo',
k, pPCATesterror, '-rd'); xlabel('k'); ylabel('Percent
Classification Error');
title('Test Set- 30%');
legend('GDA', 'EM', 'pPCA');

%Plotting likelihood per dimension per class
for i = 1:length(k)
    for j = 1:numClass
        figure((i-1) * numClass + j + 4)                %Figure N + 4
        %Finding data points for pPCA/EM
        Pmax = find(PlikIter(i, :, j) > 0);             %Find when it ends
        Emax = find(ElikIter(i, :, j) > 0);             %Find when it ends
        X = 1:(min(Pmax(1), Emax(1)) -1);                %Find the minimum
        PY = PlikIter(i, X, j);                        %Pull out values
        EY = ElikIter(i, X, j);                        %Pull out Values

        %Plotting them
        plot(X, PY, '-ro', X, EY, '-bd');
        xlabel('Iteration');
        ylabel('Likelihood of Training Data');
        title(['k = ', num2str(k(i)), ', (Full Set): Sum of LogLik per Iteration', 'class =
        legend('FactorpPCA', 'EM');
    end
end
```

9.4 Data Format

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% CS 229 Real Data Sets
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Note: This problem places in each column a training sample.

%%%%%%%%%%%%%%%%%%%%%%%%%%
%Turning off warnings
%%%%%%%%%%%%%%%%%%%%%%%%%%
s = warning('off', 'all');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Preprocess Data for Badges
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear                                   %Clear Variables
MAX = 30;                               %Maximum dimension to consider
X = [];                                 %Defining X
fid = fopen('badges.data', 'rt');       %open file
```

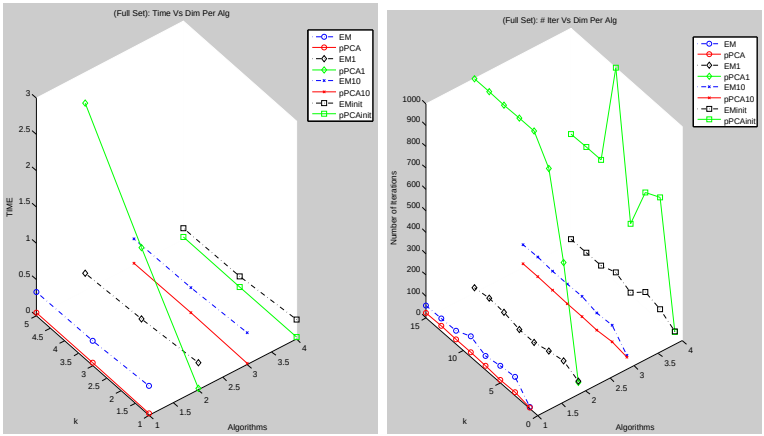
```
fgetl(fid);                                %skip first whie space
i = 0;                                     %currently at 0th element
maxDim = 0;                                %Currently max dimension is 0
while feof(fid) == 0                       %until end of file
   i = i + 1;                              %go to next element
   tline = fgetl(fid);                     %read 1 line
   y(i, 1) = (tline(1) == '+');               %Set y = 1 if +, 0 if -
   dim = min(length(tline) - 2, MAX);      %Whats this dimension (if < MAX)?
   maxDim = max(maxDim, dim);              %Whats largest size dimension?
   X = [X , [double(tline((1:dim) + 2))'; zeros(MAX - dim,1)]];
            %Concatenate X with new input
end
X = X(1:maxDim, :);                        %Shorten X to valid dimensions
fclose(fid);                               %Close file
test(X, y, 'Badges');                      %Testing it on read data

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Preprocess Data for Flags
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clear                                      %Clear Variables
X = [];                                    %Defining X
i = 0;                                     %currently at 0th element
fid = fopen('flag.data', 'rt');            %open file
while feof(fid) == 0                       %until end of file
   i = i + 1;                              %Go to next element
   tline = fgetl(fid);                     %read 1 line
   mat = regexp(tline, '(?<=,\s*)\w*', 'match');
   y(i, 1) = mat(6);                          %Storing Classification
   X(:,i) = str2double(mat([1:5,7:16,18:27])); %Dropping Class, text -> numeric
end
fclose(fid);                               %Close file
test(X, str2double(y), 'flags');           %Testing it on read data

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Preprocess Data for Balance Scale
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clear                                      %Clear Variables
X = [];                                    %Defining X
i = 0;                                     %currently at 0th element
fid = fopen('balance-scale.data', 'rt');        %open file
while feof(fid) == 0                       %until end of file
   i = i + 1;                              %Go to next element
   tline = fgetl(fid);                     %read 1 line
   mat = regexp(tline, '(?<=,\s*)\w*', 'match');
   y(i, 1) = double(tline(1));                %Storing Classification
   X(:,i) = str2double(mat);               %text -> numeric
end
fclose(fid);                               %Close file
```

```
test(X, y, 'Balance Scale');           %Testing it on read data

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Preprocess Data for Machine
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clear                                  %Clear Variables
X = [];                                %Defining X
i = 0;                                 %currently at 0th element
fid = fopen('machine.data', 'rt');     %open file
while feof(fid) == 0                   %until end of file
   i = i + 1;                          %Go to next element
   tline = fgetl(fid);                 %read 1 line
   mat  = regexp(tline, '(?<=,\s*)\w*', 'match');
   y(i, 1) = mat(9);                   %Storing Classification
   X(:,i) = str2double(mat(2:7));      %text -> numeric
end
fclose(fid);                           %Close file
y = str2double(y);                     %Converting to numeric
test(X, y > mean(y), 'Machine');       %Testing it on read data

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Preprocess Data for House Votes
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clear                                  %Clear Variables
X = [];                                %Defining X
j = 0;                                 %currently at 0th element
fid = fopen('house-votes-84.data', 'rt');     %open file
while feof(fid) == 0                   %until end of file
   j = j + 1;                          %Go to next element
   tline = fgetl(fid);                 %read 1 line
   mat = regexp(tline, '(?<=,\s*)\S', 'match');
   for i = 1:length(mat)                   %for each dimension
        if strcmp(mat(i),'n')              %if voted no
            X(i,j) = -1;                   %store -1
        else                               %otherwise either ? (0) or Y (1)
            X(i,j) = strcmp(mat(i),'y');   %store 1 if yes  or 0 o.w
        end
   end
   y(j,1) = strcmp(tline(1),'r');      %1 for republican, 0 democrat
end
fclose(fid);                           %Close file
test(X, y, 'Votes');                   %Testing it on read data

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Preprocess Data for Random test Sample
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clear                                  %Clear Variables
n = 5;                                 %Number of dimensions
```

```
m = 30;                                  %Number of samples
k = 2;                                   %Smaller dimensions
Psi = diag(rand(n,1));                   %Random Psi
Lambda = rand(n,k);                      %Random Lambda
mu = rand(n,1);                          %Mean for positive class
X = mu * ones(1,m) + Lambda * randn(k, m) + Psi^0.5* randn(n, m);
mu = -1 * rand(n,1);                     %Mean for negative class
X = [X, mu * ones(1,m) + Lambda * randn(k, m) + Psi^0.5* randn(n,
m)];
y = [ones(m,1); zeros(m,1)];             %Inputting Classification
test(X, y, 'Random');                    %Testing it on read data
```

10 Appendix E - More Results

These results are the plots (similar to above of likelihoods)

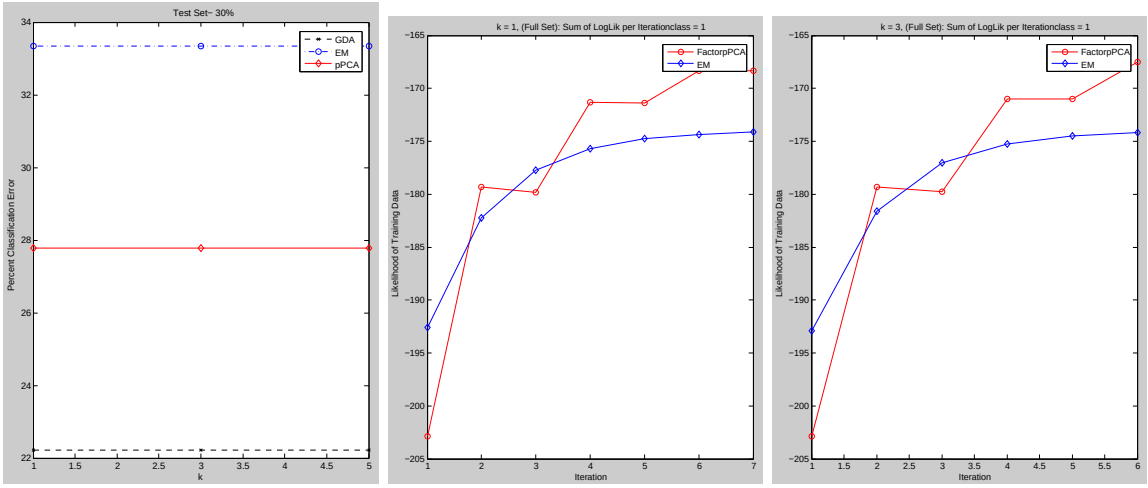


(a) (b)

Figure 5: Time convergence- Random, Machine

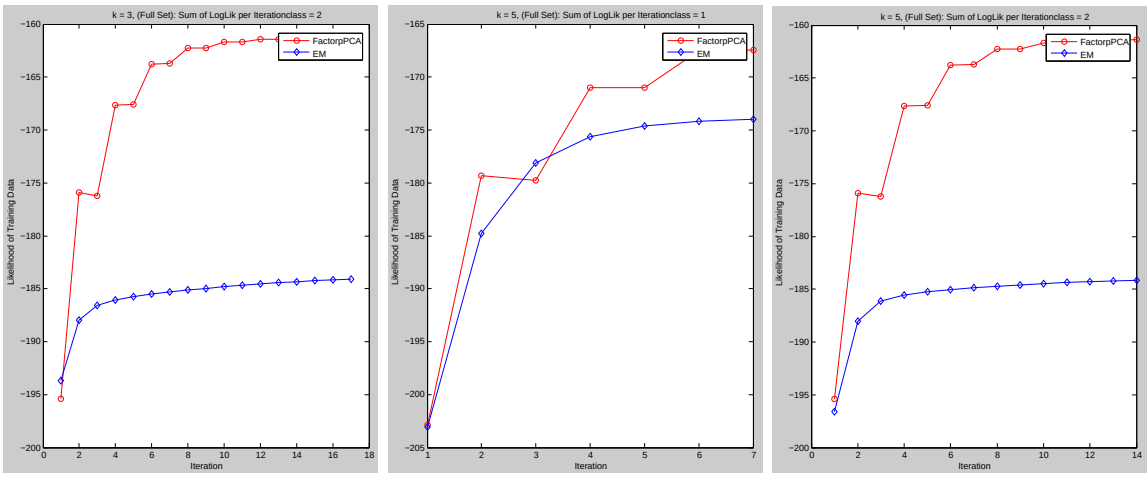
The rest of the plots associated with this test are not replicated below for sake of alleviating redundancy as they are similar to the plots below. They are available upon request however as they reflect the benefits of the new convergence test.

The plots below are the results of the other tests we executed on these algorithms. They are reproduced here for readers critique.



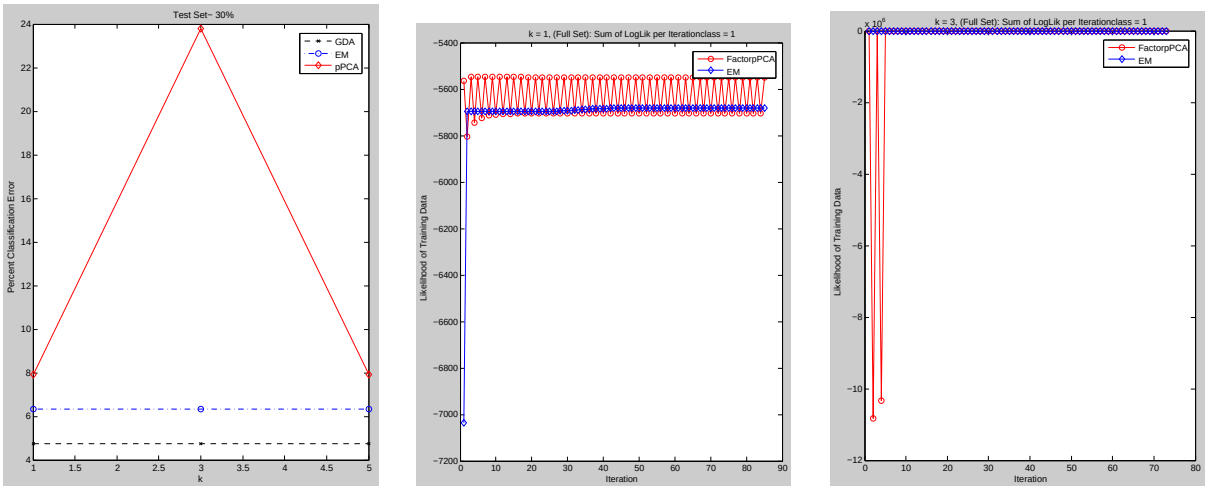
(a) (b) (c)

Figure 6: Tested on random data set



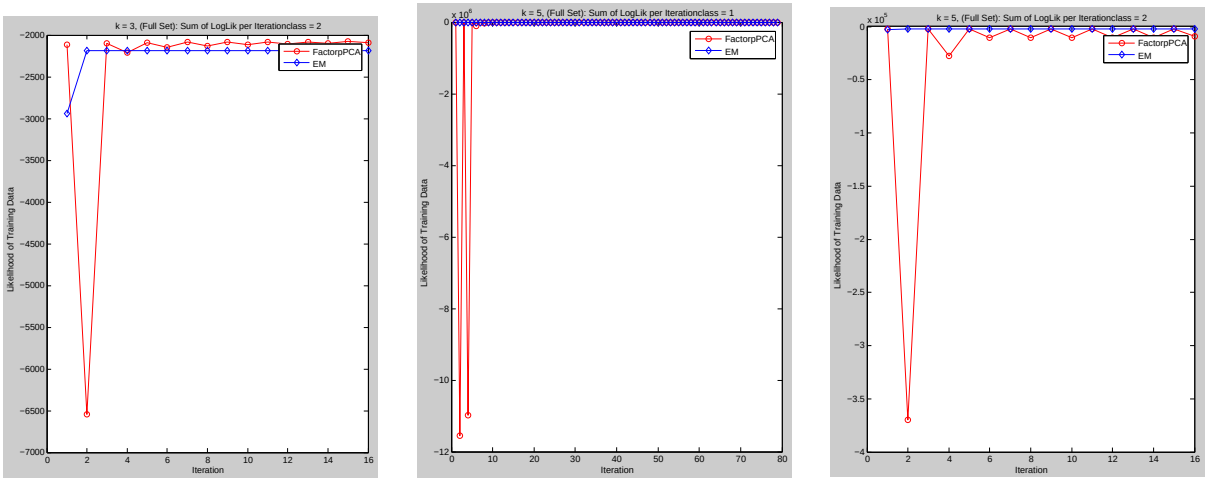
(a) (b) (c)

Figure 7: Tested on random data set



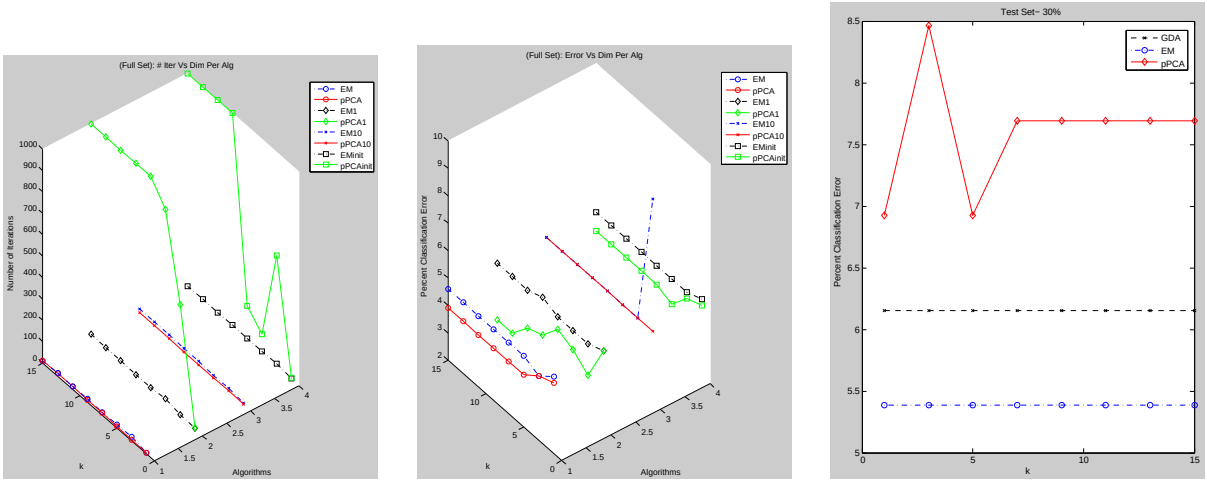
(a) (b) (c)

Figure 8: Tested on machine data sample



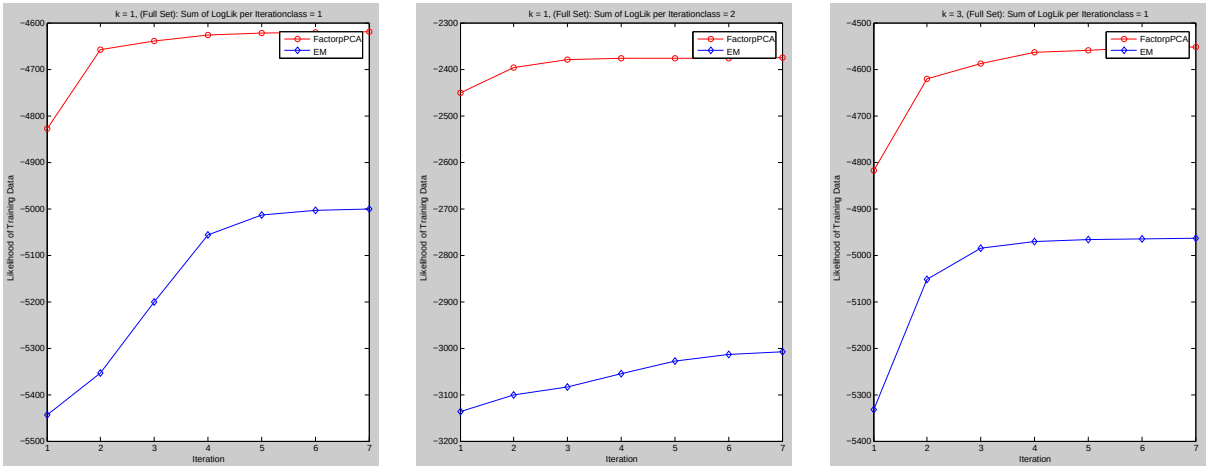
(a) (b) (c)

Figure 9: Tested on machine data sample



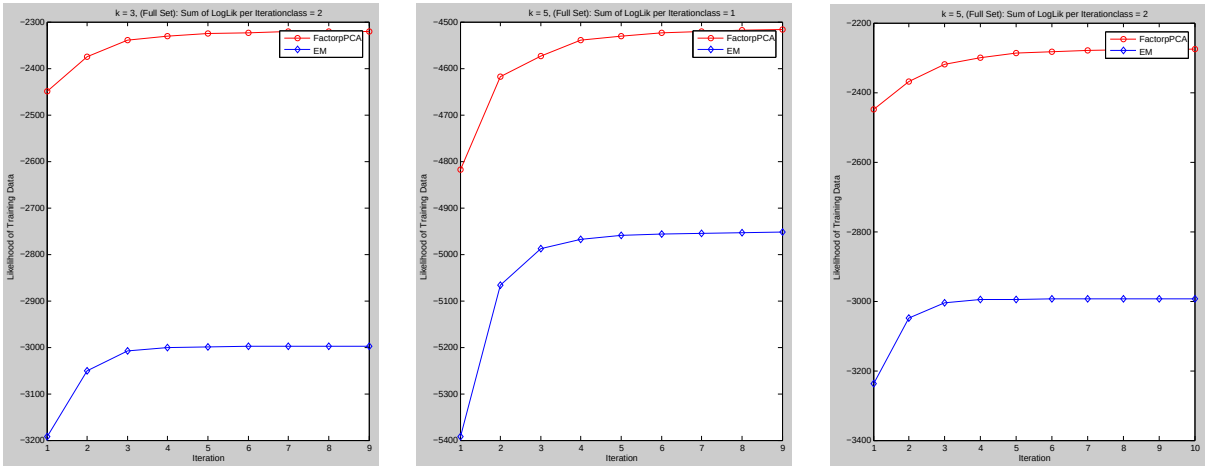
(a) (b) (c)

Figure 10: Tested on votes data sample



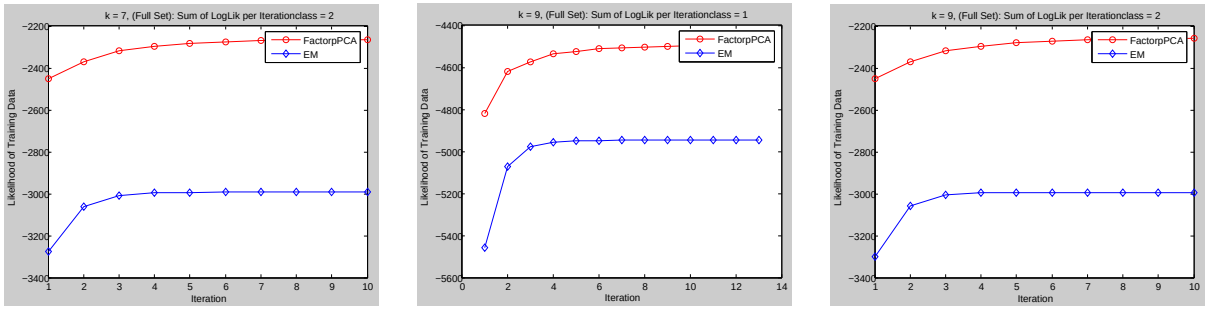
(a) (b) (c)

Figure 11: Tested on votes data sample



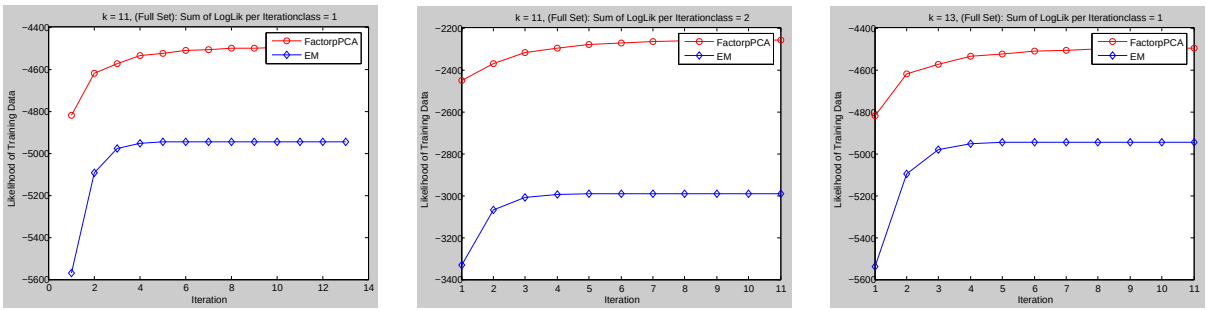
(a) (b) (c)

Figure 12: Tested on votes data sample



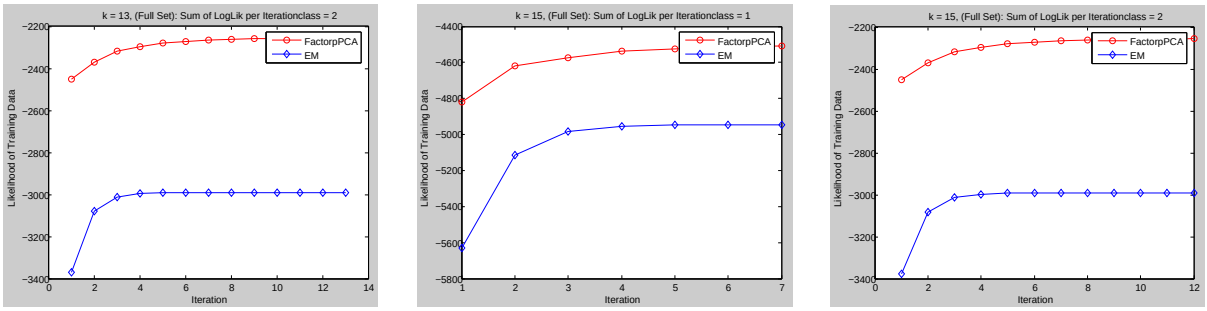
(a) (b) (c)

Figure 13: Tested on votes data sample



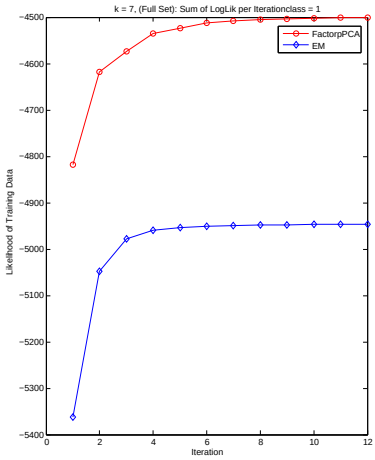
(a) (b) (c)

Figure 14: Tested on votes data sample



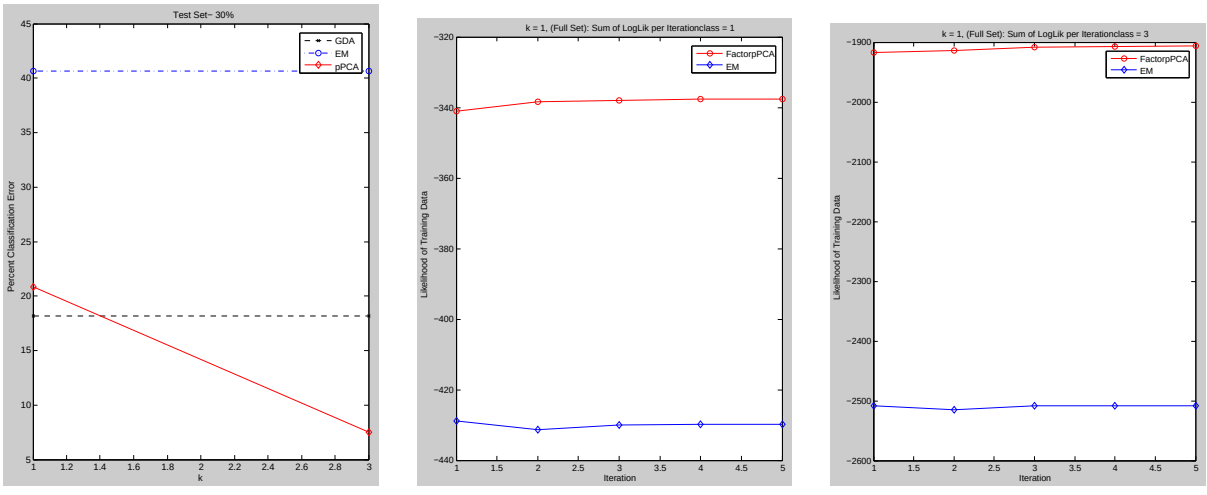
(a) (b) (c)

Figure 15: Tested on votes data sample



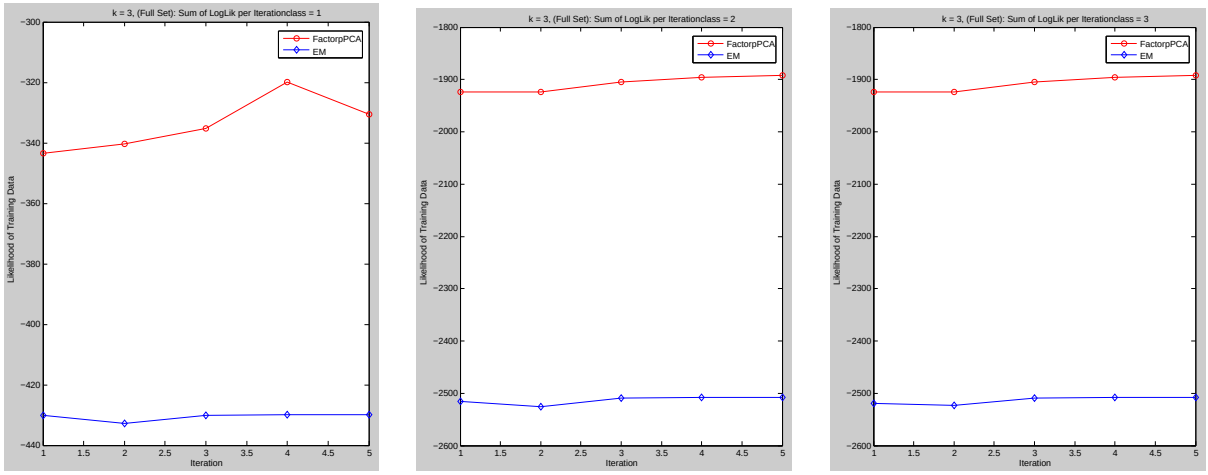
(a)

Figure 16: Tested on votes data sample



(a) (b) (c)

Figure 17: Tested on balance data sample



(a) (b) (c)

Figure 18: Tested on balance data sample

References

[1] Uci machine learning repository.

[2] Gene H. Golub and Charles F. Van Loan. *Matrix Computations*. The Johns Hopkins University Press, Baltimore, MD, USA, 1989.

[3] Roger A. Horn and Charles R. Johnson. *Matrix analysis*. Cambridge University Press, New York, NY, USA, 1986.

[4] Andrew Y. Ng. Factor analysis cs229 lecture notes.

[5] M. Tipping and C. Bishop. Probabilistic principal component analysis. *Journal of the Royal Statistical Society, Series B*, 61(Part 3):611–622, 1997.