Cross - Validation for Principal Component Analysis

Lucas Reymond[a], Pranav Krishna[a]

[a]*EPFL, Chemin du Barrage, Lausanne, Lausanne, 1015, Switzerland*

Keywords: PCA, Cross Validation, Matrix Completion, Procrustes PCA, Repaired CV for PCA

1. Introduction

In this day and age, large data-sets are very common but it is difficult to interpret and visualise high dimensional data. It is in this use case that Dimensionality reduction becomes important. Principal Components analysis, commonly called PCA, is a technique for reducing the dimensionality of the data while preserving or minimising the information loss. The main idea behind PCA is to successively create new variables in such a way as to capture a larger portion of the variation in the data.

It must be noted that PCA is an old technique and has been rediscovered many times throughout the last century in Signal Processing, Mechanical Engineering and Meteorological Science. The earliest description was given by Pearson (1901).

For PCA, the new variables are necessarily a linear combination of the existing variables, which aids the interpretability. This reduces the problem to finding the singular value decomposition of the data-set and taking successively larger truncations of the diagonal matrix to include more-and-more of the variability in the data. Because of the simplicity of the underlying procedure, PCA can be thought of in multiple ways. One mental model of PCA is trying to fit ellipsoids to the data. If there is not a lot variability along one of the axes, it will be small in length and in the other case, it will be larger. Clearly, this means that PCA is an unsupervised problem as we do not have anything akin to a response in the problem setup. Another equivalent way of visualising the PCA is to compute the best low-rank approximation of the data-set while deviations from the affine space are considered as the random error.

This is a problem for practitioners of PCA as it complicates the task of the selection the optimal number of components. We will explore a few methods of doing cross-validation for PCA. They are -

- Repaired CV for PCA
- EM Algorithm
- Procrustes CV mentioned in Kucheryavskiy et al. (2020)

Lastly, we look at a Matrix Completion which is similar in flavour to doing a PCA with missing data.

2. Methods

First, we provide a brief description of PCA. As discussed in class Masak (2022a), there are multiple ways of thinking about the PCA. One of them is that the Principal Components can be thought of as the directions (mutually orthogonal) along which the variance of the data is

maximised. Thus, if we observe that the variance of the data is encoded in the Covariance matrix, finding the Principal components becomes the following series of problems -

$$\begin{aligned} v_1 &= \arg\min_{\|v\|=1} v^\top \widehat{\mathbf{\Sigma}} v \\ v_2 &= \arg\min_{\|v\|=1, v^\top v_1 = 0} v^\top \widehat{\mathbf{\Sigma}} v \\ &\vdots \end{aligned}$$

This idea is the intuition behind the Procrustes algorithm. A holdout-portion of the data is rotated so that the Principal Components calculated from the rest of the data align with the Principal Components of the entire data set. Doing this on various holdout-data sets gives us the Procrustes data-set.

This report is mainly about the Cross-Validation procedure for PCA. Thus, it is of interest to understand what is a common method to find the best rank for the PCA. As mentioned in the notes, it is as follows Masak (2022a) -

- Split data into K folds $J_1, \ldots, J_K$
- **for** $k = 1, \ldots, K$
- Solve $\widehat{\mathbf{L}} = \arg\min_{\text{rank}(\mathbf{L})=r} \|\mathbf{X}[J_k^c,] - \mathbf{L}\|_2^2$
- Calculate $Err_k(r) = \sum_{j \in J_k} \|x_j - P_{\widehat{L}} x_j\|_2^2$
- **end for**

In the end, we choose

$$\widehat{r} = \arg\min_r \sum_{k=1}^{K} Err_k(r).$$

We observe that this procedure utilises all the data in the K^{th} fold to make predictions, which is perhaps, not a good idea on second thoughts. The issue becomes clear on comparing it with linear regression. In the case of Linear Regression, we observe that the formula for Leave one Out cross validation utilises the response and fitted value to calculate the error and the contribution of the co-variate is through the fitted value. In this case, no such procedure is possible as the problem is unsupervised, i.e., there are no response variables.

Another way to see the sub-optimality of this approach is to observe the lack of a trade-off between taking more components and fitting to the error. As we take more components, the fit tho the error improves and the overall sum decreases. Thus, one has to become dependent on human observations such as observing the deepest drop while explaining some fixed proportion of the variability in he data-set.

One way to overcome this is to make this problem a supervised problem in an artificial manner. But, as discussed in the course, this requires distributional assumptions on the data-set, which is a priori, not a requirement for the algebraic manipulations which give rise to the PCA.

Taking the normality assumptions allows us to use the property of the multivariate Normal Distribution that all of its marginals are also distributed normally. This will allow us to do cross validation in a more sophisticated manner.

To formalise the problem we write it as : Given $x_n \in \mathbb{R}^p$ are i.i.d. realizations of $X \sim N(\mu, \Sigma)$ where Rank(Σ) = r and r is not necessarily p.

The objective is to find r.

2.1. Repaired PCA

The first algorithm implemented uses the Gaussianity (the fact that conditionals of a multivariate normal distribution are normal) to turn an unsupervised problem to a supervised problem. We can find the algorithm and explanation on how to use Gaussianity here Masak (2022a).

2.2. EM Algorithm

The EM algorithm is a type of MM-algorithm wherein we minorise by taking the Expectation. Then, we take the maximum of this minorising function to get an estimate of the parameters. Iterating these steps until convergence gives us the parameters.

It is a straightforward choice for checking the optimal number of components for the PCA. We assume that the data was generated in an i.i.d. manner from a Normal distribution with a rank-deficient covariance matrix. Now, the objective is to find out the rank of this covariance matrix. To do this, we make an initial guess about the mean and covariance of the Normal distribution and keep on updating the estimates by the EM algorithm. For the updates, we assume that we did not see a portion of the data and impute it. It is in here that the Normality assumption is crucial as it allows us to impute the missing data using the conditional distribution of the Normal Distribution. The details of the proof are on the course GitHub repository at Masak (2022b).

2.3. Procrustes Cross Validation

This is a novel method of cross validation coming from the Analytical Chemistry literature Kucheryavskiy et al. (2020) wherein we create a validation data-set from the original data-set. First, a PCA is performed on the entire data set and then the data is split into various folds. After this, a PCA is performed on the data in all but the first fold and then, the angles are measured between the original PCA and this new PCA. Finally, the first hold-out data set is rotated by these angles to make the first portion of the Procrustes data-set. This process is then repeated for all other folds to finally give the Procrustes data set, which is similar but not identical to the original data-set.

In our opinion, this method is the most useful when we do not have a large amount of data, which would mean that separating the data into folds and ignoring some portion of the data set would lead to a significant loss in the accuracy of the PCA. Thus, using the Procrustes data-set, christened the pseudo-validation data set avoids the issue of of decreased accuracy by allowing us to use the entire data-set for the estimation of the optimal number of components. Another thing we observe is that the algebraic manipulations required to create this pseudo-validation set do not depend on the Normality assumption that is crucial for the EM algorithm.

2.4. PCA with missing data

The matrix completion also known as PCA with missing data Masak (2022a) is an algorithm that solves an optimization problem to provide the best least squares fit to the partially observed data matrix of a given rank. It find a rank r that minimize

$$\sum_{(i,j)\in\Omega} (X_{ij} - M_{ij})^2$$

s.t. rank(M)=r and Ω indicates which entries of X are missing. M is computed iteratively using SVD for dimension reduction until convergence.

3. Main results

Here, we run our implementations of the algorithms mentioned in 2. he way we checked the efficacy of Repaired CV for PCA was different than the other two methods because of the difference in the time it took to return the results. As expected, the EM algorithm was very slow and we had to modify the procedure, which is detailed in the relevant subsection. Similarly, the Procrustes PCA was slow in the implementation at Kucheryavskiy (2022). It should be noted that the implementation of the algorithm has been supposedly updated but we were not able to find it in the reference available to us.

Another point to note is that the matrix completion algorithm, even though similar to PCA, is not cross-validation. Thus, we checked its effectiveness in a different manner. The details are mentioned in its relevant subsection.

3.1. PCA with missing data

We test our algorithm by giving it a matrix X 10×10 whose rows are normal distributed with distribution $\mathcal{N}(0, \Sigma)$ with Σ of rank r. Then we remove randomly a given proportion of the entries of X and we choose the real rank r of Σ and for each case we run 100 simulations and report how many times the algorithm found the real given rank of Σ. We remove between 10 to 90 % of the entries of X which has a rank between 2 to 10.
In this setup, the algorithm is not very good at finding the real rank (except for 10) and overestimates it. The Table 1 shows the number of times (out of 100 runs) than the algorithm found the correct rank, where the real rank is indicated in the first row and the proportion of entries deleted is indicated in the first column. For example, we can read on this table that when we removed 20% of the entries of a matrix 10×10 generated using a matrix Σ of rank 5, the algorithm determined that the rank is 5 only 4 times (out of 100). It's efficient only when the matrix is full rank and independent of the proportion of data deleted.

	2	3	4	5	6	7	8	9	10
10%	0	0	0	1	2	7	6	32	100
20%	0	0	0	4	1	1	6	24	100
30%	0	0	0	1	1	2	5	32	100
40%	0	0	0	1	1	4	5	29	100
50%	0	0	0	1	2	3	7	38	100
60%	0	0	0	1	3	4	12	20	100
70%	0	0	0	0	2	3	9	19	100
80%	0	0	0	1	1	7	4	28	100
90%	0	0	0	0	1	1	10	26	100

Table 1: Number of times the algorithm found the correct rank when the data to be imputed was the value in the row and the real rank was the number in the column

We see on Figure 1 (that is just one example) that the value we want to minimize drops at the value of the real rank and then we try to use this to get better results. Saying that the value of interest becomes 0 if smaller than 10^{-10} we get better results: the algorithm is almost always correct (at least 99% cases are correct).

Now that this algorithm is working efficiently for normal, we try it by giving it a matrix X 10×10 whose rows have a t-multivariate distribution with $\mu = 0, \Sigma$ of a given rank r and one degree of freedom. We get very good results: the algorithm always find correct rank and never fail for any rank or proportion of data removed.

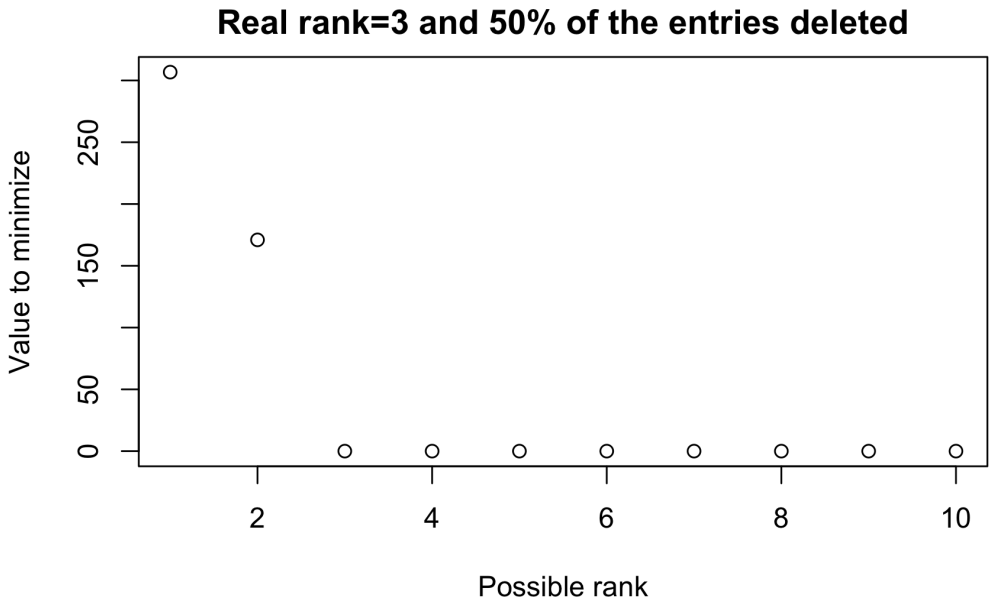

Figure 1: Plot of error vs the assumed rank for imputing 50% of the data when the real rank is 3

3.2. Repaired PCA

We test our algorithm by giving it a matrix X $n \times 10$ whose rows are Normally distributed with distribution $\mathcal{N}(0, \Sigma)$ with Σ of rank r. But this time we don't remove anything and add noise (i.e. we add a normally distributed random variable $\mathcal{N}(0, 0.1)$ to each component of X). We choose the real rank r of Σ and for each case we run 100 simulations and report how many times the algorithm found the real given rank of Σ. We use $n = 100$ to 500 as the number of rows of X which has a rank between 1 to 10. We choose the number of folds to be 5.

	1	2	3	4	5	6	7	8	9	10
100	31	28	27	28	16	29	32	42	41	30
200	46	29	25	14	20	23	27	40	37	29
300	46	39	30	18	16	23	26	37	40	26
400	58	44	25	14	21	25	39	38	44	34
500	40	40	34	26	21	30	30	36	52	29

Table 2: Number of times the algorithm found the correct rank when the number of observations is on the left and the real rank is on the top

The results shown in Table 2 are not too bad but still not perfect. To get better results we again use the drop of the value to minimize at the real rank using the same trick as before and get results as shown in Table 4. The results are very good when the real rank is between 2 and 4 but in other cases the algorithm find the correct rank less than one time out of 3.

Now we try this algorithm for non-normal distributed data by giving it a matrix X $n \times 10$ whose rows have a t-multivariate distribution with $\mu = 0, \Sigma$ of a given rank r and one degree of freedom. The results are shown in Table 5. We get quite good results when the real rank is between 2 and 5 and bad results otherwise. We see that the correctness of the algorithm does not depends on the number of rows but only the real rank of Σ.

	1	2	3	4	5	6	7	8	9	10
100	31	100	99	94	41	18	14	24	27	32
200	40	100	98	97	34	22	25	26	32	18
300	36	100	100	100	34	20	35	30	36	32
400	27	100	100	96	32	18	30	21	31	26
500	42	100	100	100	27	23	29	38	33	32

Table 3: Number of times the algorithm found the correct rank when the number of observations is on the left and the real rank is on the top when we supply additional information as mentioned above

	1	2	3	4	5	6	7	8	9	10
1	35	100	78	66	53	10	12	9	6	2
2	41	99	82	62	41	18	13	11	3	14
3	26	100	86	58	48	11	6	20	6	2
4	33	100	78	62	54	22	15	0	11	2
5	38	100	82	72	40	19	6	4	4	5

Table 4: Number of times the algorithm found the correct rank when the number of observations is on the left and the real rank is on the top when we supply non-normal data

3.3. EM Algorithm

We test our algorithm by giving it a matrix X 200×10 whose rows are normal distributed with distribution $\mathcal{N}(\mu, \Sigma)$ with μ being generated randomly and Σ of rank r. Then we remove randomly some of the entries of X and we choose the real rank r of Σ and for each case we run 100 simulations and report rank of Σ as chosen by the algorithm. We remove between 10 to 90% of the entries of X which are then imputed. To put it in other words, we are doing 2-fold cross validation wherein the proportion of the data to be imputed is said to be removed. We chose to implement cross-validation this way because in our experience during the course, 2-fold cross validation provided a sort of a lower bound on the performance of the algorithms and taking more folds decidedly improves the performance.

	1	2	3	4	5	6	7	8	9	10
10%	1	2	3	4	5	6	7	8	10	8
20%	1	2	3	4	5	10	10	10	5	5
30%	1	2	3	4	10	4	10	2	2	1
40%	1	2	3	10	2	10	1	10	1	1
50%	1	2	10	10	1	1	1	1	1	1
60%	1	10	1	1	10	10	1	10	1	1
70%	10	10	1	10	10	1	10	10	10	10
80%	10	10	10	10	10	10	10	10	10	10
90%	1	1	1	1	1	1	1	1	1	1

Table 5: The best rank according to the algorithm when we hide the proportion of data on the left and the real rank of the Covariance matrix is the number on the top

As mentioned in the notes, we observed that even though the EM algorithm was reliably able to predict the rank with much less data than the other two methods, it is the slowest of all even if we implemented the early-stopping variant by implementing only 20 implementations of the EM-cycle.

We also observe a diagonal of 10's wherein the algorithm is definitely useful and after which, the results become more or less extreme. This is an artifact of the contorted way of cross-validation that we have implemented and the fact that we allow the rank of Σ to be 10. To make a comment on the efficacy of this algorithm in real life, we would only look at the ranks 1 through 5 or 6 and the hidden/removed data proportions between 10 and 50%. In our opinion, these results are because we should not take a full rank 4×4 matrix to make predictions about a 10×10 matrix of rank 6.

Thus, in the scenario which is plausible in real-life, the algorithm works perfectly.

3.4. Procrustes PCA

We test our algorithm by giving it a matrix X 500×10 whose rows are normal distributed with distribution $\mathcal{N}(\mu, \Sigma)$ with μ being generated randomly and Σ of rank r. Then we remove randomly some of the entries of X and we choose the real rank r of Σ and for each case we run 100 simulations and report rank of Σ as chosen by the algorithm. We remove between 10 to 90% of the entries of X which are then imputed. To put it in other words, we are doing 2-fold cross validation wherein the proportion of the data to be imputed is said to be removed. We chose to implement cross-validation this way because in our experience during the course, 2-fold cross validation provided a sort of a lower bound on the performance of the algorithms and taking more folds decidedly improves the performance.

	1	2	3	4	5	6	7	8	9	10
10%	10	2	3	4	5	6	7	8	8	8
20%	10	3	3	4	5	6	7	8	9	9
30%	10	2	4	4	6	6	7	9	9	9
40%	10	10	3	5	5	6	7	8	9	8
50%	10	2	3	7	6	7	8	8	9	10
60%	10	2	4	4	5	7	9	8	9	10
70%	10	10	3	10	10	6	9	9	10	10
80%	10	10	4	10	6	6	7	9	10	10
90%	10	6	8	4	6	7	10	10	10	9

Table 6: The best rank according to the algorithm when we hide the proportion of data on the left and the real rank of the Covariance matrix is the number on the top

We also observe a row of 10's wherein the algorithm is definitely wrong at rank 1. We cannot explain this odd-result and just telling the algorithm that the rank was not 10 did not improve the results significantly. Another odd behaviour we observed was that the algorithm was very incorrect when we ran the simulations with the seed set to 0 but any other seed led to useful results. Again, based on the information we have about the random-number generation covered in the course, we do not have any ideas about what had lead to this erroneous performance.

Furthermore, we observe that the performance of the algorithm is reasonable even when we have a high rank of Σ and a larger proportion of the data is being imputed when compared to the EM algorithm. This suggests that the algorithm is quite efficient and further study would be fruitful to see why it doesn't perform well when the rank of the Covariance matrix is 1.

Other than that, we think this algorithm is slower compared to the Repaired PCA because the algorithm tried to generate an entire new data set which requires many computations of the PCA for various folds. In our opinion, this is not an issue because the algorithm is supposed to be useful

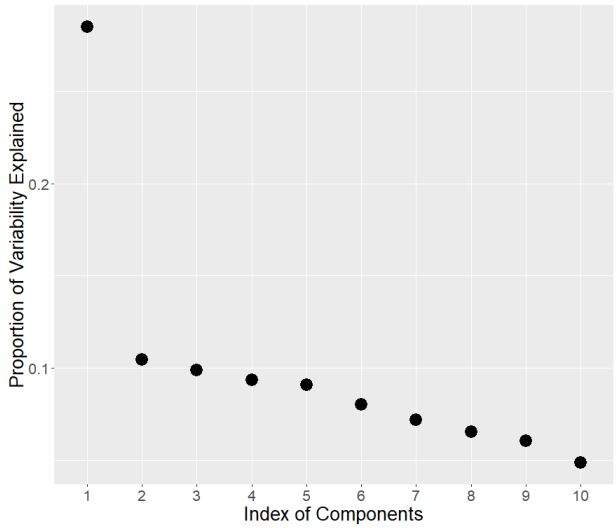

Figure 2: Plot showing the Proportion of variability explained by each Component

when we do not have a large number of observations and thus, the computation time would not be such an important factor in real-life situations.

4. Application

To check the applicability of the methods, we applied the cross-validation algorithms discussed above to some real-world data. It is commonly believed that the log-returns of stocks are Normally distributed, i.e., if P_k is the Price of some stock at the opening time on some day and Q_t is the Price of the same stock at the closing time on the same day, then $log(\frac{P_k}{Q_k})$ is widely believed to be Normally distributed. We selected a few stocks listed on the NASDAQ stock exchange some of which were American and some Swiss. We ran the discussed algorithms for PCA and found out that for all the algorithms, the best rank was 1.

On looking at Figure 2, we observe that the proportion of variability explained by one component is significantly higher than the other components and it seems plausible that the best rank would be 1.

This suggests that there is no significant relationship between the country of origin being USA or Switzerland and the daily return of a stock if we compare the particular subset of companies listed on the NASDAQ stock exchange.

5. Conclusion

We observe that the improved methods of doing Cross validation are efficient. The trade-off between the number of components and the error is more visible in EM-algorithm and Procrustes PCA but if we supply some additional constraints, the trade-off is visible for the repaired PCA too. In conclusion, we see that the Repaired CV for PCA performed the best. The EM algorithm, though efficient, is very slow compared to the rest. If we had more time, we would have tried to optimise the code and tried to implement a more standard version of Cross-Validation wherein we take some 5 or 10 folds and try to compare them. The Procrustes algorithm also works as expected and is better than the EM algorithm in real life even when a larger proportion of the data had to be imputed in our implementation.

Regarding matrix completion, we found an algorithm that can determine the matrix that approximate well a matrix of any given rank with missing entries and the result does not depends on the proportion of the matrix removed, which is a bit surprising.

References

Sergey Kucheryavskiy. Cross Validation kernel description, 2022. URL https://mdatools.com/docs/pca--pcv.html.

Sergey Kucheryavskiy, Sergei Zhilin, Oxana Rodionova, and Alexey Pomerantsev. Procrustes cross-validation—a bridge between cross-validation and independent validation sets. Analytical Chemistry, 92(17):11842–11850, 2020.

Tomas Masak. Cross Validation kernel description, 2022a. URL https://htmlpreview.github.io/?https://github.com/TMasak/StatComp/blob/master/Notes/06_CV.html.

Tomas Masak. Cross Validation kernel description, 2022b. URL https://htmlpreview.github.io/?https://github.com/TMasak/StatComp/blob/master/Notes/07_EM.html.

Karl Pearson. Liii. on lines and planes of closest fit to systems of points in space. The London, Edinburgh, and Dublin philosophical magazine and journal of science, 2(11):559–572, 1901.