An Efficient Deep Learning Model for Recommender Systems

Kourosh Modarresi(✉) and Jamie Diner

Adobe Inc., San Jose, CA, USA
kouroshm@alumni.stanford.edu, diner@adobe.com

Abstract. Recommending the best and optimal content to user is the essential part of digital space activities and online user interactions. For example, we like to know what items should be sent to a user, what promotion is the best one for a user, what web design would fit a specific user, what ad a user would be more susceptible to or what creative cloud package is more suitable to a specific user.

In this work, we use deep learning (autoencoders) to create a new model for this purpose. The previous art includes using Autoencoders for numerical features only and we extend the application of autoencoders to non-numerical features.

Our approach in coming up with recommendation is using "matrix completion" approach which is the most efficient and direct way of finding and evaluating content recommendation.

Keywords: Recommender systems · Artificial intelligence · Deep learning

1 Introduction

1.1 An Overview of Matrix Completion Approach

With the advancements in data collection and the increased availability of data, the problem of missing values will only intensify. Traditional approaches to treating this problem just remove rows and/or column that have missing values but, especially in online applications, this will mean removing most of the rows and columns as most data collected is sparse. Naïve approaches impute missing values with the mean or median of the column, which changes the distribution of the variables and increases the bias in the model. More complex approaches create one model for each column based on the other variables; our test show that this work well for small matrices but the computational time increases exponentially as more columns are added. For only numerical datasets, matrix factorization using SVD-based models proved to work on the Netflix Prize but has the drawback of inferring a linear combination between variables and not working well with mixed datasets (continuous and categorical). For sequential data, researches have been done using Recurrent Neural Networks (RNN). However, the purpose of this paper is to create a general matrix completion algorithm that does not depend on the data being sequential and works with both continuous and categorical variables that would be the founding block of a Recommendation System. A novel model is proposed using an autoencoder to reconstruct each row and impute the unknown values based on the known values, with a cost function that optimizes separately the continuous and categorical variables. Tests show that this method outperforms the performance of more complex models with a fraction of the execution time.

Matrix Completion is a problem that's been around for decades but took prominence in 2006 with the Netflix Price, where the first model to beat Netflix's baseline recommender system by more than 10% would win 1 million dollars. In such a dataset, each row represented a different user and each column a different movie. When a user i rated movie j, the position ij of the matrix would reflect the rating, otherwise it would be a missing value. This is a very particular type of dataset, as every column represented a movie from which a limited number of ratings was possible (1–5). It is fair to say that the difference between the values in the columns reflect the taste of the user but, in a general sense, each column represents the same concept i.e., a movie. Most of the research in matrix completion and recommendation systems have been done on datasets of this type, predicting the rating that a user will give on a movie, song, book, or any other content. However, most of the datasets, created in the real world, are not of this type as each column may represent a different type of data. Thus, the data could be demographical (age, income, etc.), geographical (city, state, etc.), medical (temperature, blood pressure, etc.), just to name a few. Any dataset may have missing values, and the purpose of this work is to create a general model that imputes these missing values and recommends contents in the face of having all possible type of data.

1.2 The State of the Art

Naïve Approaches

The most basic approach is to fill the missing values with the mean or median (for continuous variables) or the mode (for categorical variables). This method presents two clear problems: the first is that it is changing the distribution of the variable by giving more prominence and over-representation to the imputed variable than it really has in the data, and the second is that bias is introduced to the model, as the output is the same for all the missing values in a specific column. This is specially a problem for highly sparse datasets. It is important to notice that a variation of this method exists where the mean or median of the *row* (instead of the column) is imputed, but only works for continuous variables. The mode could be used for both continuous and categorical but will still present the problems described earlier. Some more models can be found in [1, 6, 48, 66–68].

Collaborative Filtering and Content-Based Filtering

Collaborative filtering is one of the main methods for completing *Netflix*-style datasets. In collaborative filtering, a similarity between rows (or columns) is calculated and used to compute a weighted average of the known values to impute the missing values. This method only works for numerical datasets, and is not scalable as similarity must be computed for all pairs (which is very computationally expensive).

Content-Based filtering uses attributes of the columns to find the similarity between them and then calculate the weighted average to impute. This method only works for numerical datasets.

SVD Based
The Singular Value Decomposition works by finding the latent factors of the matrix by factorizing it into 3 matrices:

$$X = U\Sigma V^T$$

Where U is an m x m unitary matrix, Σ is a diagonal matrix of dimensions m x n and V is an n x n unitary matrix. The matrix Σ represent the singular values of matrix X, and the columns of U and V are orthonormal. It reconstructs the matrix X by finding its low-rank approximation. A preprocessing step for this method is pre-imputing the missing values, usually with the mean of the column, as missing values are not permitted. This method is one of the most popular one as it was the winning solution of the Netflix Prize, but has the drawback of only working on numerical datasets, inferring a linear combination of the columns, and usually are fit for *Netflix*-style datasets.

More Complex Approaches
More complex approaches create one model for each variable with missing values, using the rows with known values in a column as the training set. A model is trained using all the variables, except the one column, as the input, and that column as the output. After a model is trained, the missing values are estimated by predicting the output of the other rows. The principal drawback of these methods is that the number of models that have to be trained increase with the number of columns of the dataset, therefore it is very computationally expensive for large datasets. This framework can work for mixed datasets or for numerical only datasets, depending on the model used. Pre-imputing missing values is needed for this framework as missing values are not permitted, usually with the mean of the column.

Some implementations of these models use Random Forest (missForest, works for mixed datasets), chained equations (mice, works for numerical only), EMB (Amelia, works for mixed datasets in theory but in this paper only the numerical part worked), FAMD (missMDA, works for mixed datasets).

2 Our Deep Learning Model

2.1 The General Framework

When designing the model, three main objectives were considered:

- Minimize reconstruction error for continuous variables
- Minimize reconstruction error for categorical variables
- Eliminate the effect of missing values in the model

Our proposed method uses autoencoders to reconstruct the dataset and impute the missing values. The concept originates from idea of SVD method through using deep learning model. Autoencoders are an unsupervised method that tries to reconstruct the input in the output using a neural network that is trained using backpropagation.

A general overview of the model is shown in Fig. 1.

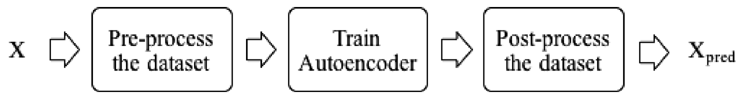

Fig. 1. The general overview of the model.

2.2 The Step of Pre-process the Dataset

The dataset can be of three types: all continuous, all categorical, or mixed (some columns are continuous and some categorical). Therefore, the first step of pre-processing the data is finding out which columns are numerical and which are categorical. The procedure followed in this work, to achieve this, is shown in Fig. 2, below.

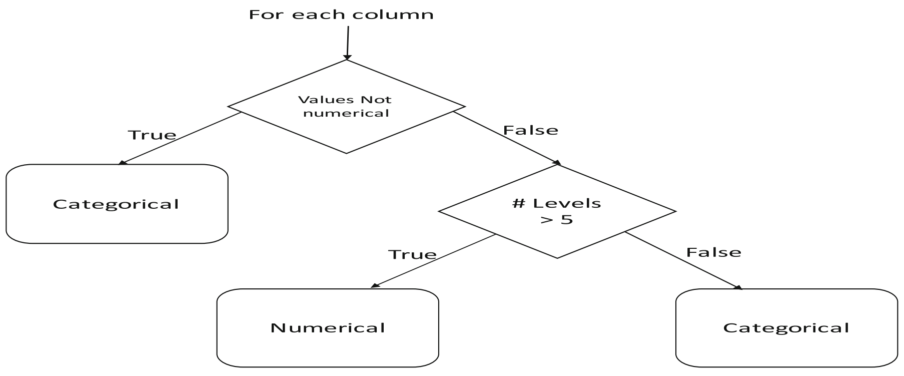

Fig. 2. The column type definition.

Once the column type is known, each of the continuous columns (if they exist) are normalized using Min Max Scaling. This way, every numerical column is scaled between 0 and 1. This step of normalization of data is a necessary step in the application of Neural Networks. The minimum and maximum values for each column are saved to be able to rescale the reconstructed matrix to the original scale.

After normalizing the continuous columns, the next step is encoding the categorical columns. For simplicity purposes, and because the order of the columns is not relevant in the model, all the continuous columns are moved to the beginning of the matrix and the categorical columns to the end. Then, each categorical column is encoded using One-Hot encoding, where one new column is created for each level of each categorical variable. The column with the label has a value of 1 and the rest a value of 0.

At this step, the matrix is all numerical and every column is between 0 and 1. For the reasons that will be explained in Sect. 2.3, three masks will be extracted from the encoded dataset:

- Missing Value Mask: same shape as the encoded matrix, where the missing values are encoded as 0 and the non-missing values as 1.
- Numerical Mask: a vector of the same length as the number of columns, where the continuous columns (if exist) are encoded as 1 and the categorical columns (if exist) are encoded as 0.
- Categorical Mask: the complement of the numerical mask, where the continuous columns are encoded as 0 and the categorical as 1.

The last step in encoding the matrix is converting all missing values to 0. This serves two purposes: the first is that neural networks can't handle missing values, and the other is to remove the effect of these missing nodes in the neural network. Once the encoded matrix and the three masks are created, the training step can begin.

2.3 Training the Autoencoder

To train the autoencoder, each row of the encoded matrix is treated as the input and output at the same time. Therefore, the number of nodes in the input (n_input) and output layer are equal to the number of columns in the encoded matrix.

The architecture that was defined consists of 3 hidden layers. The design is symmetrical with the number of nodes of each of the hidden layers as follows:

- Hidden Layer 1: *n_input/2*
- Hidden Layer 2: *n_input/4*
- Hidden Layer 3: *n_input/2*

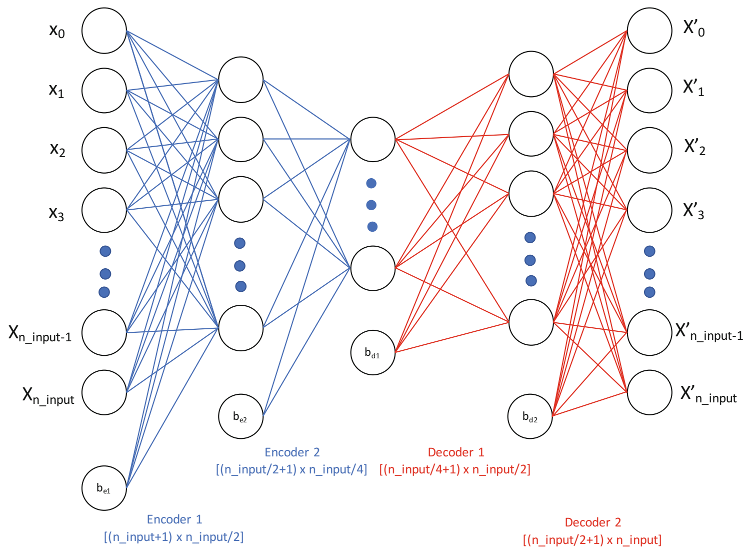

Fig. 3. The network architecture.

There are two encoding layers and two decoding layers. The reason why the number of nodes for the hidden layers is smaller than the input layer is due to the idea of projecting the data onto a lower dimension and find the latent factors to reconstruct the data set from there.

Figure 3 shows the autoencoder neural network architecture, with the dimensions of each encoding/decoding layer. The "+1" in the first dimension of each encoder/decoder is the bias term that was added.

The activation function that was used for each of the nodes is the sigmoid given as,

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

The output of each encoder and decoder are computed as follows:

$$Encoder\,1 = \sigma(X * W_{E1} + B_{E1})$$

Where * denotes matrix multiplication, W_{E1} are the weights for encoder 1 learned from the network (initialized randomly) and B_{E1} is the bias of the encoder 1 learned from the network (initialized randomly). This result is fed to the second encoder,

$$Encoder\,2 = \sigma(Encoder\,1 * W_{E2} + B_{E2})$$

Similarly, for the Decoders:

$$Decoder\,1 = \sigma(Encoder\,2 * W_{D1} + B_{D1})$$

$$X' = Decoder\,2 = \sigma(Decoder\,1 * W_{D2} + B_{D2})$$

The output of decoder 2 has the same dimensions as the input and is the output from which the weights will be trained.

2.4 The Cost Functions

As stated previously, there are three main objectives in this work; to minimize reconstruction error for both continuous and categorical variables, and to eliminate the effect of missing values in the model.

Continuous and categorical variables are different in nature, and therefore should be treated differently when used in any model. In most neural networks applications, there is only one type of output variable (either continuous or categorical) but in this case, there may be mixed nodes. This work proposes using a mixed cost function that is the sum of two separate cost functions, one for continuous variables and one for categorical variables.

$$cost_{total} = \underset{W,B}{\operatorname{argmin}}(cost_{continuous} + cost_{categorical})$$

To be able to distinguish between continuous and categorical variables, the numerical and categorical masks, that are created earlier, will be used.

For the purpose of the third objective, the missing values mask will be used to only consider the error of values that are *not* missing. By using this approach, there is no need to pre-impute missing values as they will have no effect on the overall cost function.

Mathematically, the continuous cost function is as follows:

$$cost_{continuous} = \sum\nolimits_{i,j} \left(\left(X'_{ij} - X_{ij} \right) \delta_{num_j} \delta_{miss_{ij}} \right)^2$$

Where X'_{ij} is the output of Decoder 2 for position *ij*, X_{ij} is the same value in the original encoded matrix, δ_{num_j} is the value in the numerical mask for column *j*, and $\delta_{miss_{ij}}$ is the value in the missing value mask for position *ij*. It is clear that this cost will only consider values that are in columns that are numerical $(\delta_{num_j} = 1)$ and that are not missing in the original matrix $(\delta_{miss_{ij}} = 1)$.

The categorical cost function is given by the cross entropy:

$$cost_{categorical} = -\sum_{i,j} \left(X_{ij} \ln\left(X'_{ij}\right) + (1 - X_{ij}) \ln\left(1 - X'_{ij}\right) \right) \delta_{cat_j} \delta_{miss_{ij}}$$

Similarly, X'_{ij} is the output of Decoder 2 for position *ij*, X_{ij} is the same value in the original encoded matrix, δ_{cat_j} is the value in the categorical mask for column *j*, and $\delta_{miss_{ij}}$ is the value in the missing value mask for position *ij*. It is clear that this cost will only consider values that are in columns that are categorical $(\delta_{cat_j} = 1)$ and that are not missing in the original matrix $(\delta_{miss_{ij}} = 1)$. The total cost function is minimized using Gradient Descent. The learning rate for these tests was set at a default of 0.01.

2.5 The Post-processing of the Dataset

The output of the Autoencoder is a matrix where all the numerical columns are at the beginning, and all the categorical columns are split among different columns, with a value between 0 and 1, at the end. The goal is to reconstruct the original matrix, with the columns in the same order and each categorical variable as one column with different levels.

The first step is computing the "prediction" for the categorical variables, that is, the level of the categorical variables that obtained the highest score after the decoder 2. Once the category is found, the name of the column is assigned as the category or level for that variable. This is repeated for all categorical variables.

Once each categorical column is decoded to its original form and levels, the columns are reordered using the order of the original dataset. Then, the numerical variables are scaled back using the minimum and maximum values saved during the pre-processing step for each column.

At this point, the matrix is in the same shape and scale as the original matrix; with all the missing values imputed.

The model in this work is based on a deep learning model using autoencoder for content recommendation based on the solution of the matrix completion problem. The main idea that this work proposes is extending the state of the art to impute missing values of any type of dataset, and not just numerical. One of the principal idea of this work is the application of a new cost function, a mixed cost function, that has not been done before. This function detects which columns are continuous and which are categorical, and computes the proper error depending on the type of the data. This improves considerably the performance of the model and can be extended to any neural network application that requires output nodes of mixed types.

3 The Results and Conclusion

3.1 The Data Set and the Results

For this analysis, 15 publicly available datasets [12–26] were used. The dataset was selected such that the data set would be diverse with respect to sparsity level, domain or application, amount of numerical vs categorical data, and the number of rows and columns.

To create a more varied selection of data, 100 bootstrap samples were created from each of the datasets by selecting a random number of rows, a random number of columns, and a random number of missing values.

To measure the performance of continuous variables, the Normalized Root Mean Squared Error (NRMSE) measure is used. The reason this metric is used is that we could compare the performance of difference datasets regardless of the range or variance it has. The lower the NRMSE score, the better.

$$NRMSE = \sqrt{\frac{mean\left(\left(x_{true} - x_{pred}\right)^2\right)}{var\left(x_{true}\right)}}$$

To measure the performance of categorical variables, the Accuracy is used. The higher the accuracy score, the better.

$$Accuracy = mean\left(x_{true} = x_{pred}\right)$$

The execution time is measured in seconds. The lower the execution time, the better.

To compare the performance of our model vs other state of the art models, seven packages in R were used as baselines models: *Amelia* [51], *impute* [49], *mice* [72], *missForest* [70], *missMDA* [59], *rrecsys* [11], and *softImpute* [48]. The models in these packages are state of the art solutions for the matrix completion problem and cover all the models described in the introduction.

The number of missing values ranged from 0 to 100%, but limitations on other packages only allowed only up to 80% on most models, and 20% on Amelia package model.

Figure 4 shows the performance of the models with 1500 bootstrap samples (100 per dataset) measured by the NRMSE. It can be seen that the model proposed in this paper outperforms all of the models, with less variation in the results. The closest model, Amelia, was only tested with up to 20% sparsity but our autoencoder still improves the median NRMSE by 11% (0.09293 vs 0.10395).

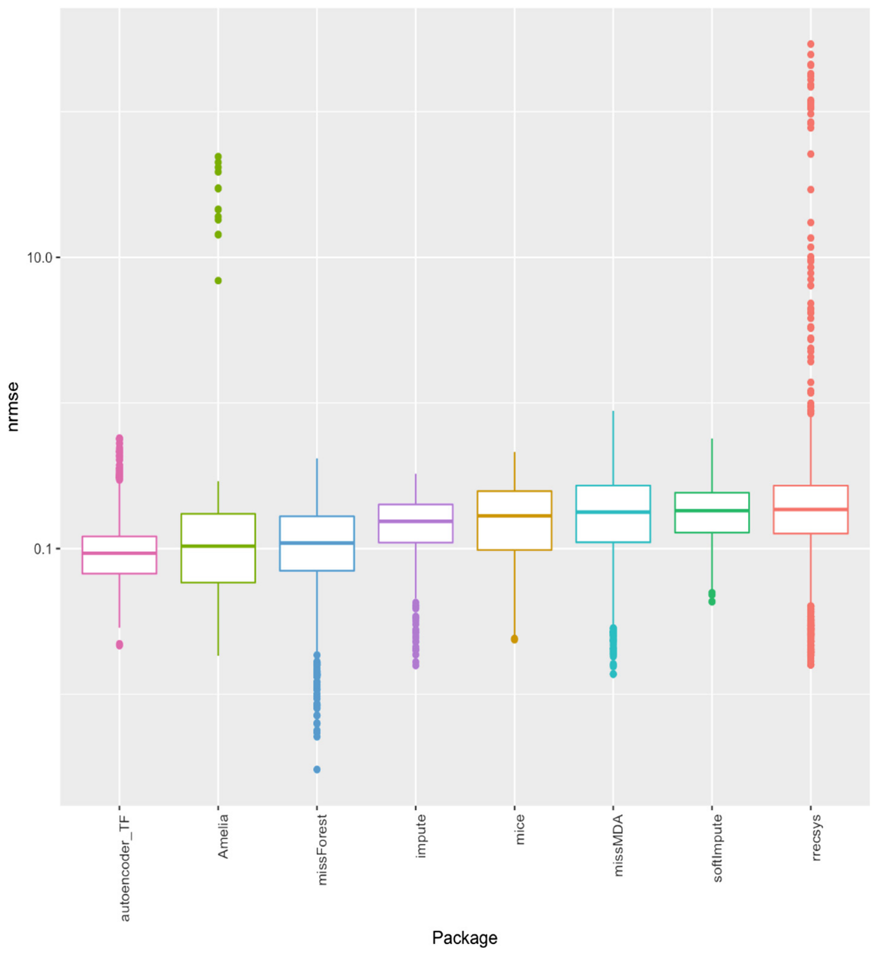

Fig. 4. Comparing the performance using NRMSE.

Figure 5 shows the accuracy of categorical variables for all packages that are able to handle them. Out of the seven packages that were tested to compare, only four are able to impute categorical variables. The model proposed in this paper sits right in the middle in terms of median performance with large variation in the results.

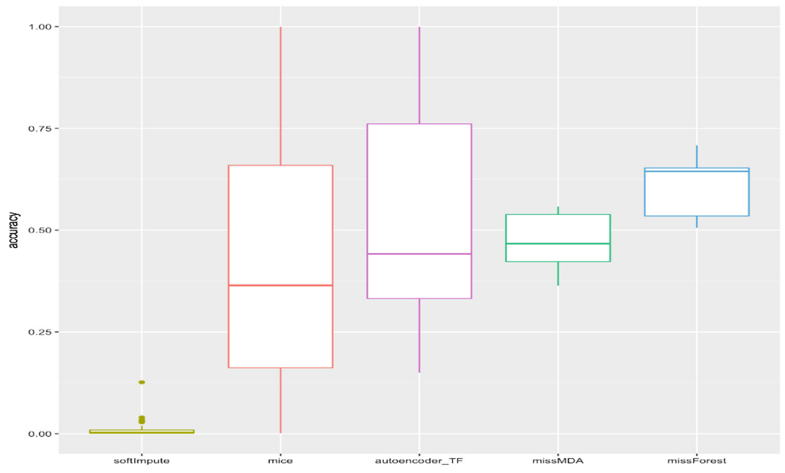

Fig. 5. Comparing the accuracy of different models.

Figure 6 shows the execution time in seconds for all the packages. The tests were run in a MacBook Pro with a 2.5 GHz Intel Core i7 processor. It can be seen that the autoencoder model is the third slowest, however the median computational cost is still reasonable at about 0.5 s per model. Comparing the execution time to models that can handle categorical values, the two models that outperform in accuracy take about 5 times as long to execute as the autoencoder indicating our model has the best

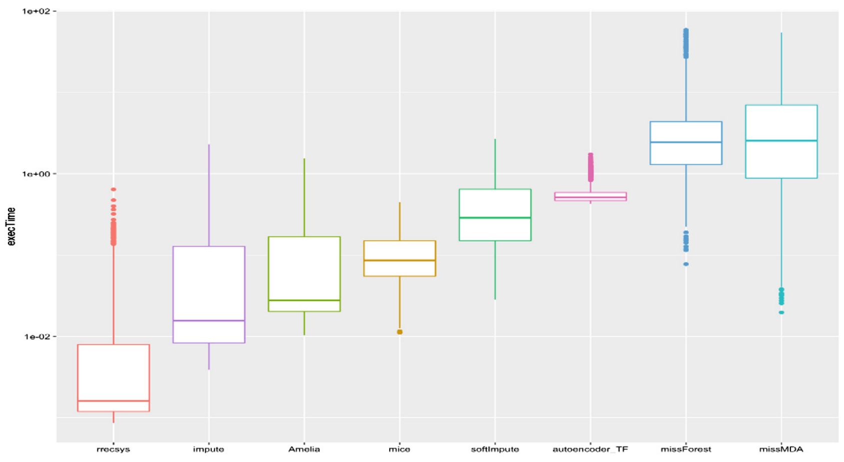

Fig. 6. Comparing the execution time of different models.

performance for NRMSE for all models tested. Thus, for the models that can handle mixed datasets, our model has the best tradeoff between accuracy and execution time.

The results indicate our model outperforms existing models. It has the best NRMSE of all models, and has the best trade-off accuracy and computational complexity as the two models.

References

1. Becker, S., Bobin, J., Candès, E.J.: NESTA, a fast and accurate first-order method for sparse recovery. SIAM J. Imaging Sci. **4**(1), 1–39 (2009)
2. Bjorck, A.: Numerical Methods for Least Squares Problems. SIAM, Philadelphia (1996)
3. Boyd, S., Vandenberghe, L.: Convex Optimization. Cambridge University Press, Cambridge (2004)
4. Breese, J.S., Heckerman, D., Kadie, C.: Empirical analysis of predictive algorithms for collaborative filtering. In: Proceedings of Fourteenth Conference on Uncertainty in Artificial Intelligence. Morgan Kaufmann (1998)
5. Cai, J.-F., Candès, E.J., Shen, Z.: A singular value thresholding algorithm for matrix completion. SIAM J. Optim. **20**(4), 1956–1982 (2008)
6. Candès, E.J., Recht, B.: Exact matrix completion via convex optimization. Found. Comput. Math. **9**, 717–772 (2008)
7. Candès, E.J.: Compressive sampling. In: Proceedings of the International Congress of Mathematicians, Madrid, Spain (2006)
8. Chen, P.-Y., Wu, S.-Y., Yoon, J.: The impact of online recommendations and consumer feedback on sales. In: Proceedings of the 25th International Conference on Information Systems, pp. 711–724 (2004)
9. Cho, Y.H., Kim, J.K., Kim, S.H.: A personalized recommender system based on web usage mining and decision tree induction. Expert Syst. Appl. **23**, 329–342 (2002)
10. Claypool, M., Gokhale, A., Miranda, T., Murnikov, P., Netes, D., Sartin M.: Combining content-based and collaborative filters in an online newspaper. In: Proceedings of the ACM SIGIR 1999 Workshop on Recommender Systems (1999)
11. Çoba, L., Zanker, M.: rrecsys: an R-package for prototyping recommendation algorithms. In: RecSys 2016 Poster Proceedings (2016)
12. Data, Abalone. https://archive.ics.uci.edu/ml/datasets/abalone
13. Data, Air Quality. https://archive.ics.uci.edu/ml/datasets/Air+Quality
14. Data, Batting. http://www.tgfantasybaseball.com/baseball/stats.cfm
15. Data, Bike. https://archive.ics.uci.edu/ml/datasets/bike+sharing+dataset
16. Data, Boston. https://archive.ics.uci.edu/ml/datasets/housing
17. Data, CASP. https://archive.ics.uci.edu/ml/datasets/Physicochemical+Properties+of+Protein+Tertiary+Structure
18. Data, Census: Click on the “Compare Large Cities and Towns for Population, Housing, Area, and Density” link on Census 2000. https://factfinder.census.gov/faces/nav/jsf/pages/community_facts.xhtml
19. Data, Concrete. https://archive.ics.uci.edu/ml/datasets/Concrete+Compressive+Strength
20. Data, Data_akb. https://archive.ics.uci.edu/ml/dtasets/ISTANBUL+STOCK+EXCHANGE#
21. Data, Parkinsons. https://archive.ics.uci.edu/ml/datasets/parkinsons
22. Data, S&P. http://www.cboe.com/products/stock-index-options-spx-rut-msci-ftse/s-p-500-index-options/s-p-500-index/spx-historical-data
23. Data, Seeds. http://archive.ics.uci.edu/ml/datasets/seeds

24. Data, Waveform. https://archive.ics.uci.edu/ml/datasets/Waveform+Database+Generator+(Version+2)
25. Data, Wdbc. https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+%28Prognostic%29
26. Data, Yacht. http://archive.ics.uci.edu/ml/datasets/yacht+hydrodynamics
27. d'Aspremont, A., El Ghaoui, L., Jordan, M.I., Lanckriet, G.R.G.: A direct formulation for sparse PCA using semidefinite programming. SIAM Rev. **49**(3), 434–448 (2007)
28. Davies, A.R., Hassan, M.F.: Optimality in the regularization of ill-posed inverse problems. In: Sabatier, P.C. (ed.) Inverse Problems: An Interdisciplinary Study. Academic Press, London (1987)
29. DeMoor, B., Golub, G.H.: The restricted singular value decomposition: properties and applications. SIAM J. Matrix Anal. Appl. **12**(3), 401–425 (1991)
30. Donoho, D.L., Tanner, J.: Sparse nonnegative solutions of underdetermined linear equations by linear programming. Proc. Natl. Acad. Sci. **102**(27), 9446–9451 (2005)
31. Efron, B., Hastie, T., Johnstone, I., Tibshirani, R.: Least angle regression. Ann. Stat. **32**, 407–499 (2004)
32. Elden, L.: Algorithms for the regularization of ill-conditioned least squares problems. BIT **17**, 134–145 (1977)
33. Elden, L.: A note on the computation of the generalized cross-validation function for ill-conditioned least squares problems. BIT **24**, 467–472 (1984)
34. Engl, H.W., Hanke, M., Neubauer, A.: Regularization methods for the stable solution of inverse problems. Surv. Math. Ind. **3**, 71–143 (1993)
35. Engl, H.W., Hanke, M., Neubauer, A.: Regularization of Inverse Problems. Kluwer, Dordrecht (1996)
36. Engl, H.W., Kunisch, K., Neubauer, A.: Convergence rates for Tikhonov regularisation of non-linear ill-posed problems. Inverse Prob. **5**, 523–540 (1998)
37. Engl, H.W., Groetsch, C.W. (eds.): Inverse and Ill-Posed Problems. Academic Press, London (1987)
38. Gander, W.: On the linear least squares problem with a quadratic Constraint. Technical report STAN-CS-78–697, Stanford University (1978)
39. Golub, G.H., Van Loan, C.F.: Matrix Computations. Computer Assisted Mechanics and Engineering Sciences, 4th edn. Johns Hopkins University Press, US, (2013)
40. Golub, G.H., Van Loan, C.F.: An analysis of the total least squares problem. SIAM J. Numer. Anal. **17**, 883–893 (1980)
41. Golub, G.H., Kahan, W.: Calculating the singular values and pseudo-inverse of a matrix. SIAM J. Numer. Anal. Ser. B **2**, 205–224 (1965)
42. Golub, G.H., Heath, M., Wahba, G.: Generalized cross-validation as a method for choosing a good ridge parameter. Technometrics **21**, 215–223 (1979)
43. Guo, S., Wang, M., Leskovec, J.: The role of social networks in online shopping: information passing, price of trust, and consumer choice. In: ACM Conference on Electronic Commerce (EC) (2011)
44. Häubl, G., Trifts, V.: Consumer decision making in online shopping environments: the effectsof interactive decision aids **19**, 4–21 (2000)
45. Hastie, T., Tibshirani, R., Friedman, J.: The Elements of Statistical Learning; Data mining, Inference and Prediction. Springer, New York (2001). https://doi.org/10.1007/978-0-387-84858-7
46. Hastie, T.J., Tibshirani, R.: Handwritten Digit Recognition via Deformable Prototypes. AT&T Bell Laboratories Technical report (1994)
47. Hastie, T., Tibshirani, R., Eisen, M., Brown, P., Ross, D., Scherf, U., Weinstein, J., Alizadeh, A., Staudt, L., Botstein, D.: 'Gene Shaving' as a method for identifying distinct sets of genes with similar expression patterns. Genome Biol. **1**, 1–21 (2000)

48. Hastie, T., Mazumder, R.: Matrix Completion via Iterative Soft-Thresholded SVD (2015)
49. Hastie, T., Tibshirani, R., Narasimhan, B., Chu, G.: Package 'impute'. CRAN (2017)
50. Hofmann, B.: Regularization for Applied Inverse and Ill-Posed problems. Teubner, Stuttgart, Germany (1986)
51. Honaker, J., King, G., Blackwell, M.: Amelia II: A program for Missing Data (2012)
52. Anger, G., Gorenflo, R., Jochum, H., Moritz, H., Webers, W. (eds.): Inverse Problems: principles and Applications in Geophysics, Technology, and Medicine. Akademic Verlag, Berlin (1993)
53. Hua, T.A., Gunst, R.F.: Generalized ridge regression: a note on negative ridge parameters. Commun. Stat. Theory Methods **12**, 37–45 (1983)
54. Iyengar, V.S., Zhang, T.: Empirical study of recommender systems using linear classifiers. In: Cheung, D., Williams, G.J., Li, Q. (eds.) PAKDD 2001. LNCS (LNAI), vol. 2035, pp. 16–27. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-45357-1_5
55. Jeffers, J.: Two case studies in the application of principal component. Appl. Stat. **16**, 225–236 (1967)
56. Jolliffe, I.: Principal Component Analysis. Springer, New York (1986). https://doi.org/10.1007/978-1-4757-1904-8
57. Jolliffe, I.T.: Rotation of principal components: choice of normalization constraints. J. Appl. Stat. **22**, 29–35 (1995)
58. Jolliffe, I.T., Trendafilov, N.T., Uddin, M.: A modified principal component technique based on the LASSO. J. Comput. Graph. Stat. **12**(3), 531–547 (2003)
59. Josse, J., Husson, F.: missMDA: a package for handling missing values in multivariate data analysis. J. Stat. Softw. **70**(1) (2016)
60. Linden, G., Smith, B., York, J.: Amazon.com recommendations: item-to-item collaborative filtering. Internet Comput. **7**(1), 76–80 (2003)
61. Mazumder, R., Hastie, T., Tibshirani, R.: Spectral regularization algorithms for learning large incomplete matrices. JMLR **2010**(11), 2287–2322 (2010)
62. McCabe, G.: Principal variables. Technometrics **26**, 137–144 (1984)
63. Modarresi, K., Golub, G.H.: An adaptive solution of linear inverse problems. In: Proceedings of Inverse Problems Design and Optimization Symposium (IPDO2007), 16–18 April 2007, Miami Beach, Florida, pp. 333–340 (2007)
64. Modarresi, K.: A Local Regularization Method Using Multiple Regularization Levels, Stanford, April 2007
65. Modarresi, K., Golub, G.H.: An efficient algorithm for the determination of multiple regularization parameters. In: Proceedings of Inverse Problems Design and Optimization Symposium (IPDO), 16–18 April 2007, Miami Beach, Florida, pp. 395–402 (2007)
66. Modarresi, K.: Recommendation system based on complete personalization. Procedia Comput. Sci. **80C** (2016)
67. Modarresi, K.: Computation of recommender system using localized regularization. Procedia Comput. Sci. **51C** (2015)
68. Modarresi, K.: Algorithmic Approach for Learning a Comprehensive View of Online Users. Procedia Comput. Sci. **80C** (2016)
69. Sedhain, S., Menon, A.K., Sanner, S., Xie, L.: AutoRec: autoencoders meet collaborative. In: WWW 2015 (2015)
70. Stekhoven, D.: Using the missForest Package. CRAN (2012)
71. Strub, F., Mary, J., Gaudel, R.: Hybrid Collaborative Filtering with Autoencoders (2016)
72. Van Buuren, S., Groothuis-Oudshoorn, K.: MICE: multivariate imputation by chained equations in R. J. Stat. Softw. **45**(3), 1–67 (2011)