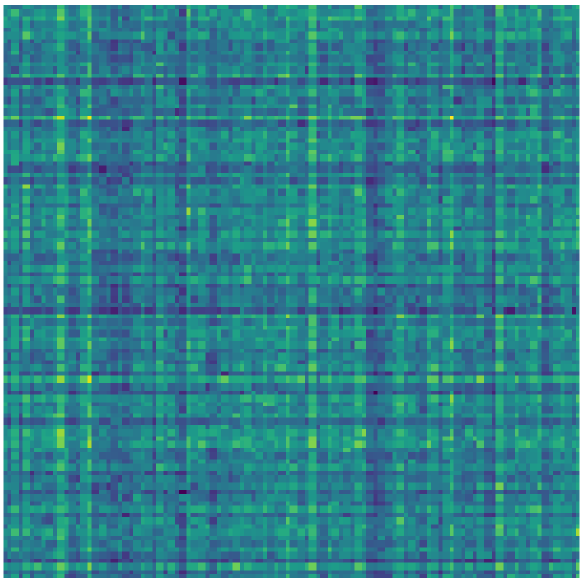
CHALMERS



Deep Learning Models for Data Integration and Surrogate Models for Interpretable Predictions with Applications in Integromics and Recommender Systems

Master's thesis in Mathematical Sciences

OSKAR LIEW
PER EDWARDSSON

Department of Mathematical Sciences
CHALMERS UNIVERSITY OF TECHNOLOGY
Göteborg, Sweden 2020

MASTER'S THESIS IN MATHEMATICAL SCIENCES

Deep Learning Models for Data Integration and Surrogate Models for Interpretable Predictions with Applications in Integromics and Recommender Systems

OSKAR LIEW
PER EDWARDSSON

Department of Mathematical Sciences
Division of Applied Mathematics and Statistics
CHALMERS UNIVERSITY OF TECHNOLOGY

Göteborg, Sweden 2020

Deep Learning Models for Data Integration and Surrogate Models for Interpretable Predictions with Applications in Integromics and Recommender Systems
OSKAR LIEW
PER EDWARDSSON

© OSKAR LIEW , PER EDWARDSSON, 2020

ISSN 1652-8557
Department of Mathematical Sciences
Division of Applied Mathematics and Statistics
Chalmers University of Technology
SE-412 96 Göteborg
Sweden
Telephone: +46 (0)31-772 1000

Cover:
Raw data on the left and the reconstructed matrix using the latent factors learned by dCMF on the right.

Chalmers Reproservice
Göteborg, Sweden 2020

Deep Learning Models for Data Integration and Surrogate Models for Interpretable Predictions with Applications in Integromics and Recommender Systems
OSKAR LIEW
PER EDWARDSSON
Department of Mathematical Sciences
Division of Applied Mathematics and Statistics
Chalmers University of Technology

Abstract

Many tasks require the simultaneous analysis of multiple heterogeneous data sets, also known as integrative data analysis. In the past, most data integration methods made linear assumptions in the shared latent representations between the data sets. Recently, Deep Collective Matrix Factorization (dCMF) was proposed as a matrix completion algorithm that can utilize auxiliary data sources without making any assumptions about the data, by modelling non-linearities using deep learning. In this thesis, we examine the performance and versatility of dCMF and propose a framework to interpret the predictions of the model, based on Linear Interpretable Model-agnostic Explanations (LIME), that we call dCMF-LIME. The explanations give variable importance measures for an individual prediction and can be used to gain trust or to troubleshoot a model. We also propose a method for unsupervised data translation that we call a Data Translation Network (DTN) that can learn to transform data from one set of data to another by first encoding them to a shared latent domain and then reconstructing any of the learned data from said latent domain. We saw that dCMF outperformed our baseline methods on simulated data and a recommendation task, but it showed poor performance on our gene-disease association test, where it was outclassed by all other methods. DTN displayed the third best performance in the same test and shows promise for future work.

Keywords: Integrative data analysis, Deep learning, dCMF, CMF, Integromics

Acknowledgements

We wish to express our deepest gratitude for our supervisors, Rebecka Jörnsten and Felix Held, for your incredible support during our thesis work, despite the unfortunate circumstances. It has been greatly appreciated and the thesis is better for it.

Thank you Erik Kristiansson, for being our examiner. Without you this master thesis could not have happened.

Thank you Martin Raum, for allowing us to use Gantenbein for running our simulations. It has been instrumental in keeping running times down.

We also want to thank to Oskar Allerbo, for his assistance with deep learning specific questions.

Lastly we want to pay special regards to the teachers and professors that we have had the fortune of meeting during our studies, as well as the program directors Jana Madjarova for *Engineering Physics*, Julie Rowlett for *MPENM* and Mats Granath for *MPCAS*. We have learned incredibly much these last five years and we have you to thank for it.

Acronyms

aSDAE Additional Stacked Denoising Autoencoder. 1

CMF Collective Matrix Factorization. 1–4, 8–11, 15–17, 23, 24, 26, 27

dCMF Deep Collective Matrix Factorization. i, 1–5, 7–11, 13–20, 23–27

DTN Data Translation Network. i, 2, 3, 7, 8, 10, 11, 16–18, 24–27

FPR False Positive Rate. 20

GAN Generative Adversarial Network. 27

gCMF Group-wise sparse Collective Matrix Factorization. 1, 2, 8–11, 15–17, 23, 24, 27

GSEA Gene Set Enrichment Analysis. 27

LIME Linear Interpretable Model-agnostic Explanations. i, 2, 3, 5, 13, 14

MM-PCA Multi-group Multi-view Principal Component Analysis. 1, 2, 8–11, 15–17, 23, 24

NMF Non-negative Matrix Factorization. 1, 3

pRMSE predicted Root Mean Square Error. 9, 10, 23

RMSE Root Mean Square Error. 4, 5

SHAP Shapely Additive Explanations. 5

SVD Singular Value Decomposition. 1, 3

TCGA The Cancer Genome Atlas. 11, 12

TPR True Positive Rate. 20

Contents

1 Introduction

In large scale studies it has become more common to jointly analyze several data sets; public databases, in-house experiments, data from different technological platforms or molecular level data. These so called data integration tasks pose challenges due to the heterogeneous nature of the data sets, large quantities of missing values and a costly overhead of preprocessing, aligning and normalizing the data. Data is often pairwise relational and can therefore be represented as matrices. Unknown entries in the data can then be predicted using matrix completion methods by assuming that sufficient information may be hidden in latent substructures, that can be found through factorization of the data. However, classical linear methods, e.g. Non-negative Matrix Factorization (NMF) or SVD-based methods, only handle a single matrix at a time. Recently, Deep Collective Matrix Factorization (dCMF) [MR19] was proposed as a method to enable more flexible data integration. dCMF utilizes deep neural network autoencoders to find shared latent information in multi-view data.

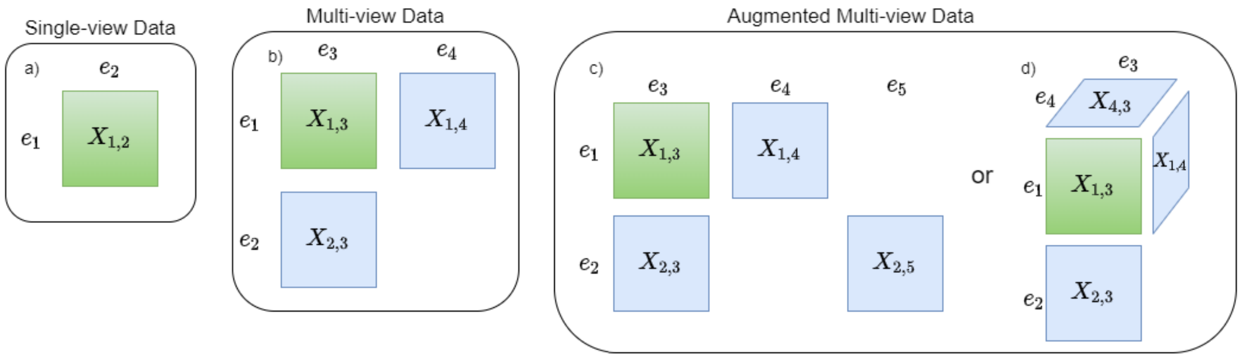


Figure 1.1: Different data settings are visualized above. Each matrix X is a *view* and each e is a data *entity*. The green matrix ($X_{1,2}$ in the single-view data a), $X_{1,3}$ in the others) is considered the *main* matrix which is of interest to supplement. In multi-view data b), additional side information matrices are available that overlap with each other on at least one axis. In augmented multi-view data, the additional side information matrices can have side information matrices of their own, as in c), and there can be matrices that share both of its entities with two other matrices, creating a three dimensional relation between matrices, as in d).

When multiple partially related data sets are available, predictive models can benefit by using the information in the shared structure of the data. This data setting is commonly referred to as multi-view data. Each matrix represents a *view* and the relationship between two *entities* along the rows and columns of the matrix. The first example, Figure 1.1 a) depicts a single view $X_{1,2}$ with row entity e_1 and column entity e_2. The second example, Figure 1.1 b) shows a multi-view data setup with three views and four entities. This setting is very common for recommender systems where $X_{1,3}$ are user ratings for the different items and the side information matrices $X_{2,3}$ and $X_{1,4}$ are item features, e.g. genres, titles and run-time for movies, and user features, like age and gender, respectively. The entities are users e_1, item features e_2, items e_3 and user features e_4. The third example, Figure 1.1 c) is called an *Augmented Multi-View* setup and is often recognized by the side information matrices having side information matrices of their own, but it can really be any arbitrary collection of matrices where all matrices share at least one axis with another matrix in the setup [KBT14], such as in Figure 1.1 d) where $X_{4,3}$ share both its column and row entity with other views. Continuing the recommender setup example using c), the matrix $X_{2,5}$ could include information about the genres of movies, including average length, common actors, and such. Another example of an augmented multi-view setup could include an additional matrix relating e_3 to e_4, resulting in a three dimensional matrix structure.

In the past, algorithms for integrative data analysis on augmented multi-view data were commonly linear methods, e.g. Collective Matrix Factorization (CMF) and Group-wise sparse Collective Matrix Factorization (gCMF), that were limited to finding linear interactions of the latent factors. These restrictions made the model lack in flexibility compared to modern non-linear methods like Additional Stacked Denoising Autoencoder (aSDAE) that uses neural networks to improve multi-view (not augmented) learning. Other recent models, like Multi-group Multi-view Principal Component Analysis (MM-PCA) [Kal+19] can find linear latent factors but with regularization to make the factors interpretable. dCMF combines the benefit of analysing augmented multi-view data as well as being able to model non-linear relations through the use of autoencoders, at the cost of becoming more opaque than the linear models. A common concern when working with neural networks is over- and under-fitting, which dCMF is able to minimize by employing automated hyperparameter selection. It does, however encounter issues when confronted with data of mixed sparsity levels and it is therefore not guaranteed to outperform the older methods.

The opaque nature of dCMF and other deep learning based methods can be a large deterring factor for some applications, e.g. in medicine and automotive industry. To help humans understand the models better, we implement a framework for interpreting the predictions of dCMF based on Linear Interpretable Model-agnostic Explanations (LIME) [RSG16], that we call dCMF-LIME. By using dCMF-LIME it is possible to get variable importance measures for predictions from a trained dCMF model, allowing the user to gain intuition for the model's decisions.

We also examine the situations where dCMF excels and fails by comparing its performance to the similar, linear methods, CMF, gCMF and MM-PCA. Three different data sets are used; A simulated dataset, the MovieLens100k [HK15] recommender dataset, and medical data collected from The Cancer Genome Atlas [Wei+13] and DisGeNET [Piñ+15], that we refer to as the omics dataset. These datasets were chosen because they were similar to those used in Mariappan, Rajan (2019) [MR19] and are therefore easy to compare to their results. We also propose a model that we call a Data Translation Network (DTN), that is based on the work of [LBK17] on unsupervised image-to-image translation. DTN uses neural networks to translate data between two or more data entities by first encoding the views into a shared latent domain. This method should theoretically be more general than any of the other methods mentioned here, because it would learn what information is shared between the views more directly. How well two data entities can be translated can be found by observing the distance between points in the latent domain. An example application of this method could be to translate between gene expression profiles collected using traditional bulk-analysis techniques and modern single-cell techniques allowing their joint analysis, saving resources for medical laboratories.

The aims for this thesis is to i) compare the importance of preprocessing of data for linear and neural network based data integration methods; ii) implement an interpretability framework for the dCMF [MR19] method; and iii) introduce a new data integration method based on deep learning.

2 Background

In this section we aim to supply the reader with a sufficient theoretical background to understand the methods discussed in this thesis. The section starts with a definition of matrix factorization. After that we give a description of Collective Matrix Factorization (CMF), to give a deeper understanding of Deep Collective Matrix Factorization (dCMF), that is described shortly thereafter. The reasoning behind explainable models in machine learning is discussed and a few different approaches are highlighted, as well as the Linear Interpretable Model-agnostic Explanations (LIME) method for interpretable machine learning, which is used in this thesis. Lastly, the Data Translation Network (DTN) neural network architecture is described.

2.1 Matrix Factorization

In linear algebra, a matrix factorization aims to find a factorization of a matrix $X \in \mathbb{R}^{m\times n}$ into a product of lower ranked matrices $U \in \mathbb{R}^{m\times k}$ and $V \in \mathbb{R}^{n\times k}$ such that $X \approx f(UV^T)$ for some function f. Singh and Gordon [SG08] define a matrix factorization algorithm as consisting of a set of choices:

1. A link function $f : \mathbb{R}^{m\times n} \to \mathbb{R}^{m\times n}$, that enables transformations in the factorization.
2. A loss function $\mathcal{L}(X, f(UV^T)) \geq 0$, as a dissimilarity measure between X and its prediction $f(UV^T)$.
3. An optional matrix of data weights $W \in \mathbb{R}_+^{m\times n}$, that are arguments to the loss.
4. Strict constraints on the latent factors $U, V \in \mathcal{A}$, where $\mathcal{A}$ is some closed set.
5. And a regularization term $\mathcal{R}(U, V) \geq 0$.

As such, a matrix factorization algorithm finds a factorization $X \approx f(UV^T)$ by solving

$$\underset{U,V\in\mathcal{A}}{\operatorname{argmin}} \left(\mathcal{L}(X, W, f(UV^T)) + \mathcal{R}(U, V)\right). \tag{2.1}$$

2.2 Deep Collective Matrix Factorization

Most common matrix factorization methods such as Singular Value Decomposition (SVD) and Non-negative Matrix Factorization (NMF) find factorizations of a single matrix and are unable to generalize to a multi-view setting. Unlike these methods, CMF proposed by [SG08], aims to find joint low-rank factorizations of a set of M matrices $(X^{(1)}, \ldots, X^{(M)})$ relating to each other through N_E entities $(e_1, \ldots, e_{N_E})$ where each entity has dimension d_{e_i}.

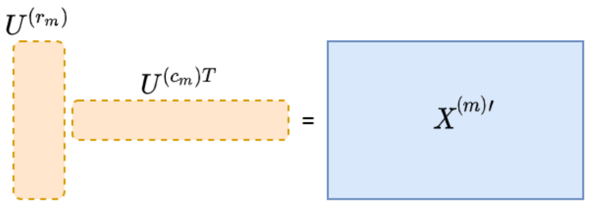


Figure 2.1: Illustration of the reconstruction of a matrix $X^{(m)}$ from the latent factors of its row and column entities r_m and c_m.

Let r_m be the row entity of the m:th matrix and c_m be its column entity. Then the m:th matrix can be approximated by $X^{(m)} \approx U^{(r_m)}U^{(c_m)^T}$ where $U^{(e_i)} \in \mathbb{R}^{d_{e_i}\times k}$ is a latent low-rank representation for entity e_i, where k is the rank of the latent space. These low rank representations are found by solving the optimization problem

$$\underset{\left\{U^{(e_i)}\in\mathbb{R}^{d_{e_i}\times k}\right\}_{e_i\in\{1,\ldots,N_E\}}}{\operatorname{argmin}} \sum_{m=1}^{M} \mathcal{L}\left(X^{(m)}, U^{(r_m)}U^{(c_m)^T}\right) + \sum_{i=1}^{N_E} \mathcal{R}\left(U^{(e_i)}\right), \tag{2.2}$$

where $\mathcal{L}$ is some loss function and $\mathcal{R}$ is a regularization function. There are many choices of loss and regularization functions, e.g. Bouchard, Guo and Yin [BGY13] have successfully used a negative log-likelihood loss and a squared Frobenius norm for the regularization.

dCMF [MR19] aims to extend the CMF framework by including non-linearity in the low rank representations such that $U^{(e_i)} = f_\theta^{e_i}([X]_{e_i})$. Here $f_\theta^{e_i}$ denotes a non-linear transformation unique to the entity e_i, parameterized by θ and $[X]_{e_i}$ corresponds to all the matrices related to entity e_i. Similar to CMF, the entity-specific latent representations are found by solving the optimization problem

$$\underset{\theta, \left\{U^{(e_i)} \in \mathbb{R}^{d_{e_i} \times k}\right\}_{i \in \{1,\dots,N_E\}}}{\operatorname{argmin}} \sum_{m=1}^{M} \mathcal{L}\left(X^{(m)}, f_\theta^{r_m}([X]_{r_m}) \left(f_\theta^{c_m}([X]_{c_m})\right)^T\right) + \sum_{i=1}^{N_E} \mathcal{R}\left(U^{(e_i)}\right). \tag{2.3}$$

In dCMF, f is modelled using fully-connected autoencoder neural networks [Kra91]. A neural network consists of layers of neurons that are connected to the neurons of the adjacent layers. Each neuron performs a weighted sum of the outputs of the previous layer and applies an activation function. Each weight is associated with a connection and is a parameter that can be modified to change the behaviour of the network in order to minimize some objective function. An autoencoder network, visualized in Figure 2.3, consists of two parts; an encoder E and a decoder D. The network tries to learn the unit function $D(E(\mathbf{x})) = \mathbf{x}$ but is restricted by an hourglass shape in the layer sizes, meaning it has to learn to encode the data $E(\mathbf{x})$ into a lower dimension and then restore it to its original dimension using the decoder $D(E(\mathbf{x}))$.

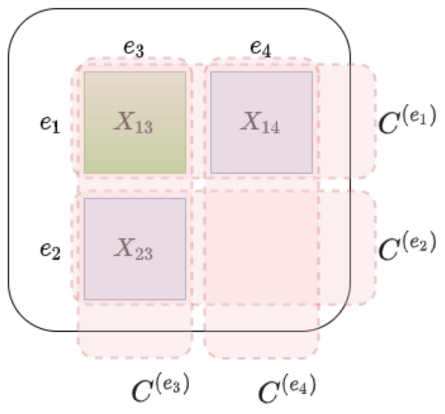


Figure 2.2: Illustration of how the concatenated views are constructed from a multi-view data grid consisting of three matrices. These concatenated views $C^{(e_i)}$ are then used as input for the autoencoders used in dCMF.

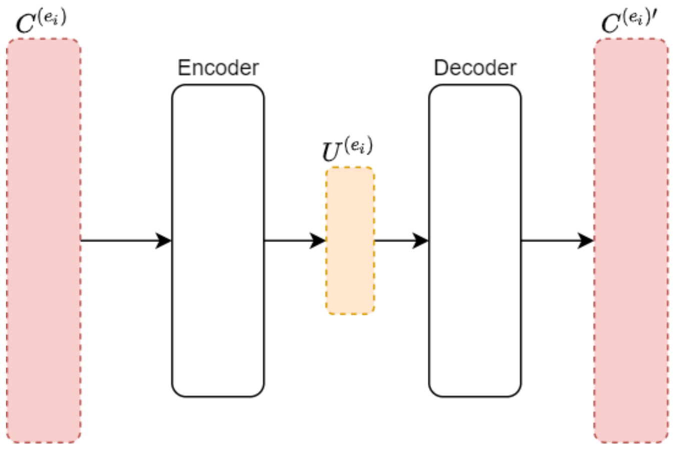


Figure 2.3: In order to extract latent features, the input view $C^{(e_i)}$ is compressed into a lower dimensional latent representation $U^{(e_i)}$ by way of an encoder. Conversely, translating latent features back into the data e_i is done in a decoder. This produces a re-creation of $C^{(e_i)}$ called $C^{(e_i)\prime}$, denoted by the prime. This type of network is commonly referred to as an autoencoder.

Entities can be shared over multiple matrices and to enable the autoencoders to share information between matrices, a set of N_E concatenated matrices $C^{(e_i)}$ are created, one for each entity e_i, by concatenating all matrices containing e_i along their shared dimension, $i = 1, ..., N_E$. This process is visualized in Figure 2.2. Then N_E autoencoders are used to extract the latent substructures $U^{(e_i)}$ from each concatenated matrix $C^{(e_i)}$. For the autoencoder whose input is $C^{(e_i)}$ the decoding is $D(E(C^{(e_i)})) = C^{(e_i)\prime}$, and the middle layer, or encoding, becomes the latent factor $E(C^{(e_i)}) = U^{(e_i)}$ after training, which is equivalent to $f_\theta^{e_i}([X]_{e_i})$ in Equation (2.3). The autoencoders are trained together by solving the optimization problem

$$\underset{\left\{U^{(e_i)} \in \mathbb{R}^{d_{e_i} \times k}\right\}_{i \in \{1,\dots,N_E\}}}{\operatorname{argmin}} \sum_{m=1}^{M} l_R\left(X^{(m)}, X^{(m)\prime}\right) + \sum_{i=1}^{N_E} l_E\left(C^{(e_i)}, C^{(e_i)\prime}\right) \tag{2.4}$$

Here l_R is the matrix reconstruction loss measured by the Root Mean Square Error (RMSE) between the view $X^{(m)}$ and the predicted matrix $X^{(m)\prime} = U^{(r_m)} U^{(c_m)^T}$ and l_E is the autoencoder reconstruction loss measured by the RMSE between the input $C^{(e_i)}$ and the decoding $C^{(e_i)\prime}$. Hyperparameters for training and autoencoder depth are iteratively selected using a special variant of Bayesian optimisation [Jas12] called multi-task Bayesian optimization [SSA13], that is able to optimize multiple tasks (training of autoencoders, that are dependent due to the concatenation, in the case of dCMF) simultaneously and transfer knowledge gained from one task to another.

2.3 Explainable Models

While modern deep learning methods can achieve great performance in machine learning tasks such as classification and clustering, their answers can never achieve 100% accuracy in all situations. These models are mostly considered black boxes and it can be hard to know if the perceived performance is real even for truly new data. When an eventual failure occurs, it can lead to catastrophical consequences, in the worst case loss of life in industries such as medicine and automotive, where the uncertainty of black box models has been a barrier for field deployment. Therefore, we applied an explainable framework to dCMF which provides explanations as to why the model arrived at its conclusions, increasing trust or finding shortcomings in the model's predictions. There are two main approaches to interpretability and explainability of machine learning models; integrated (transparent) and post-hoc [DBH18].

The best model explanations are of course the model itself, but that requires a model that is transparent, meaning it is easy for a human to understand its inner workings [LL17]. These models are usually interpretable by their mathematical structure such as linear regression, decision trees or general additive models [TG19]. Another way of creating transparent models is through pruning, through the use of l_1-regularization. An example of such a method is the group lasso [YL06] that has successfully been used to create sparse and interpretable neural network models [Tan+17]. The downside with integrated explainable models is that the transparency of a model and its predictive performance often are conflicting objectives and there exists a trade-off between the two [WFM15].

Post-hoc interpretability methods instead extract information from an already trained model and commonly treat the original model as a black-box. Because this approach does not depend on how the model works, the performance of the model is not affected [DBH18]. Examples of post-hoc interpretation methods include variable importance measures through perturbation of input data, such as in [ZF14] or through surrogate models as in Shapely Additive Explanations (SHAP) [LL17] or LIME [RSG16]. The explainable framework for dCMF that we propose in this thesis is based on LIME.

2.3.1 Local Interpretable Model-agnostic Explanations

LIME [RSG16], tries to find an interpretable model that is locally faithful to a supervised learning model. The supervised learning model is treated as a black box and can be trained using any kind of data. This is achieved in four steps that are described in more detail in the following paragraphs.

1. Creation of an interpretable representation of a data point.
2. Local exploration through perturbation of the interpretable representation.
3. Computation of black box model response on the original representation of the perturbed samples.
4. Training of the explainer model using the black box model response as the dependent variable and the interpretable samples as the independent variables.

The model of interest $f : \mathcal{X} \to \mathcal{Y}$ might be trained on data in a domain $\mathcal{X}$ that is hard or impossible for a human to understand, such as word embeddings for text classification or three color channels per pixel for images. Interpretable representations of these two cases could be a binary vector that indicates whether a word exists in the input or the presence of a contiguous patch of similar pixels for image recognition. Given an instance $x \in \mathcal{X}$ that we want to explain, we create an interpretable representation of x by transforming the data with a function $h : \mathcal{X} \to \{0,1\}^{d'}$ such that $h(x) = x'$, where the dimension of the interpretable representation d' is usually lower than the dimension of $\mathcal{X}$. There should exist an approximate inverse transformation $h^{-1}(x') \approx x$ for h so that the data can be restored to the original representation. The function h has to be designed by the user according to their own sense of what is an interpretable representation of the data. Because of the existence of a fidelity and interpretability trade-off, the interpretable representation $h(x) = x'$ has to be selected with care.

The instance x with interpretable representation x' is explored by sampling a random number non-zero elements from x' and setting the rest of the elements of x' to zero, resulting in a perturbed sample on the interpretable representation $\hat{x}' \in \{0,1\}^{d'}$. This is repeated N times to create a training set for the exploration model. Then, for each perturbed sample $\hat{x}'_i$, $i = 1, \dots, N$ the original representation of that sample $h^{-1}(\hat{x}'_i) = \hat{x}_i \in \mathcal{X}$ is recovered. The samples are now in a domain where the model output $f(\hat{x}_i)$ can be computed and its output used as a target values for the explanation model.

The explanation model $g : \{0,1\}^{d'} \to \mathcal{Y}$ is trained using the target values $f(\hat{x}_i)$ as the dependent variable and the corresponding interpretable representations $\hat{x}'_i$ as independent variables. This way, the model g becomes an approximation of f defined on the interpretable domain $\{0,1\}^{d'}$. The models are usually sparse linear models, like a regularized linear model [HK70] where the coefficients can be used as variable importance measures. The explainer models are then trained on the locally weighted loss

$$\mathcal{L}\left(f, g, \pi_x\right) = \sum_{i=1}^{N} \pi_x(\hat{x}_i)\left(f(\hat{x}_i) - g(\hat{x}'_i)\right)^2. \tag{2.5}$$

where $\pi_x(\hat{x}_i) = \exp\left(-D(x, \hat{x}_i)^2/\sigma^2\right)$ is an exponential kernel defined on some distance measure $D(x, \hat{x})$ e.g. l_2-distance or cosine distance. This loss ensures that samples closer in proximity to the local point x are more important for the training which makes the explainer model g more locally faithful to f.

3 Method

In this section we outline the experimental setups of this thesis starting with an introduction to the Data Translation Network (DTN) and a description of the ideas behind it. After that we give a summary of the baseline methods that Deep Collective Matrix Factorization (dCMF) and DTN are compared to. Following that comes an explanation of the benchmarking datasets, how they were preprocessed and/or simulated and which evaluation metrics were used. Lastly, the dCMF-LIME interpretability is described as well as the two test case setups with the MovieLens100k and omics data.

3.1 Using Neural Networks for Data Integration

By using embedding networks to compress data into latent representations, the problem of data integration can be tackled without using matrix factorization. The network introduced in this thesis is the Data Translation Network that has been designed to integrate data by translating between different data cohorts. It is inspired by the work of Liu et al. on unsupervised image to image translation [LBK17].

DTN, illustated in Figure 3.1, is constructed by one set of networks for each input data matrix. For a set of M input matrices $X_i \in \mathbb{R}^{n_i \times m_i}$, $i = 1, ..., M$, one set of embedding networks E_i and one set of generating networks G_i are created. E_i is a fully connected feed forward neural network with input size m_i and output size $k < m_i \,\forall\, i$, with an adjustable number of layers. The generator G_i is also a fully connected feed forward neural network which has the same number of layers, but the reverse order, taking an input of size k and producing an output of size m_i. The number of neurons on each layer is interpolated between input and output sizes, creating a cone-shaped layer structure like an autoencoder.

The role of E_i is to embed the rows of X_i into a latent dimension k, creating a new matrix $Z_i \in \mathbb{R}^{n_i \times k}$. The rows in Z_i represent the rows in X_i, but notably, each row in Z_i is of the same size as the rows of $Z_j, \forall\, j = 1, ..., M$. The role of generator G_i is to take a latent representation as input and produce an output that matches the distribution of X_i. However, since all latent representations have the same size, G_i can take any such representation as input.

Consider a sample row x_i taken from matrix X_i. When translated into the latent domain, we represent it by z_i. Then, we can pass it through a generator G_j. We expect that if $j = i$, $G_j(z_i) \approx x_i$, that is we can reconstruct the input data using the generator of the same data type like an autoencoder would[WYZ16]. If we use another generator, $j \neq i$, we now expect to receive something that looks like the distribution of the whole matrix X_j, but has the characteristics of the specific row x_i. We call this act data translation, akin to what is called image translation in [LBK17].

Data that can be translated with minimal loss are data that we call integratable. If X_i and X_j are integratable, we expect that for such data, the latent representations Z_i, Z_j are located close to each other in the latent domain, ideally overlapping. If one region in the latent domain contain many points from both X_i and X_j, this means that the autoencoder finds that location to be a good space to represent their respective data entities, and that data can be translated accurately between the data entities.

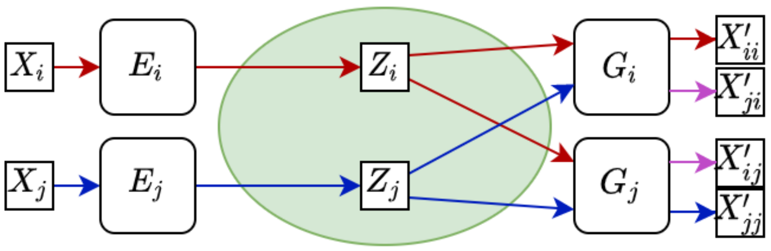


Figure 3.1: An example of a Data Translation Network sketched above. X constitutes input matrices, E are embedding networks, and G are generative networks. The latent domain $Z = \mathbb{R}^k$ is shared for all data, and any latent representation Z_i can be used with any generator G_j to produce an output matrix X'_{ij}. If $i = j$, this is a reconstruction. Otherwise, this is a data translation.

Consider Figure 3.1. The input data sets X_i, X_j are compressed into the latent domain by their respective embedding network. The latent representations can then be used as input in either generator. Using the generator corresponding to the input matrix produces a recreation of the input matrix, matching the top and bottom matrices of the right hand side. The two middle outputs X'_{ij}, X'_{ji} follow the distribution of their last index while retaining characteristics of their first index. Consider an example with two different results of an experiment conducted in two different laboratories, one small scale with low resources and one large scale with exact measurements. It is expected that if the result of the experiment are strong enough, the result from the small scale can be translated to the large scale laboratory, creating an output that produces the same experimental result, but as if it had been done in the large scale laboratory.

Since our goal for X'_{ij} is different from that of X'_{ii}, different loss functions have been used depending on which output is being evaluated. In our implementation, the loss functions of the DTN is given by

$$\mathfrak{L}_{i=j}(X_i, X'_{ij}) = \alpha\sqrt{\frac{1}{D}\sum_{n,m}(x^{(i)}_{nm} - x^{(ij)\prime}_{nm})^2}$$

$$\mathfrak{L}_{i\neq j}(X_i, X'_{ij}) = (1-\alpha)\left[\sum_{n,m} x^{(i)}_{nm} \ln \frac{x^{(i)}_{nm}}{x^{(ij)}_{nm}{}'}\right]^2$$

$$\mathfrak{L} = \sum_{i=1}^{N}\sum_{j=1}^{N}\mathfrak{L}_{ij} + \beta\sum_{w_i}|w_i|^2$$

where D is the total number of elements contained in x and x', α is a parameter that regulates the relative importance of the two losses, β is a regularization factor and x_{nm} contains the element on row n and column m of matrix X. The first loss variant is a root mean squared error and the second is a Kullback-Leibler divergence. The parameter α can be tuned to weigh data translation versus data reconstruction, or to make sure that the losses are of equal magnitude, which will assist in data integration. The final loss function $\mathfrak{L}$ is then iterated through via back propagation until convergence, defined by when n_p epochs have been iterated through without improvement. The parameter n_p is referred to as *patience*. The performance of DTN was measured using the gene-disease test on the omics data.

3.2 Baselines

The dCMF and DTN algorithms were compared to two established multi-view matrix factorization methods; Collective Matrix Factorization (CMF) [SG08] and Group-wise sparse Collective Matrix Factorization (gCMF) [KBT14] as well as one recent algorithm; Multi-group Multi-view Principal Component Analysis (MM-PCA) [Kal+19]. This was done to give perspective to the results of the new models.

Collective Matrix Factorization

Collective Matrix Factorization is described briefly in the beginning of Section 2.2. It finds low-rank factorizations of a set of data matrices $(X^{(1)}, \ldots, X^{(M)})$ by solving the optimization problem in Equation (2.2).

Group-wise Sparse Collective Matrix Factorization

For CMF, if the individual matrices do not share their low-rank structure with each other, the algorithm will mostly pick up noise and hardly any joint signal. To remedy this problem, gCMF [KBT14] has the ability to learn separate private factors that are unique to a single entity type in the setup. If no such private factors can be found, it is identical to regular CMF. These factors are learned using Variational Bayesian Inference, similar to what was done by [IR10] and [KS12].

Multi-Group Multi-View Principal Component Analysis

The method MM-PCA, was proposed by [Kal+19]. It finds sparse low-rank factors that can be unique to a single matrix or shared between two or more matrices by minimizing the loss function

$$\min_{D^{(e.)},V^{(e.)}} \sum_{m=1}^{M} ||\mathcal{M}^{(m)} \odot \left(X^{(m)} - V^{(r_m)} D^{(r_m)} D^{(c_m)} V^{(c_m)^T} \right) ||_F^2 + \lambda_1 \sum_{i=1}^{E} ||D^{(e_i)}||_1 + \tag{3.1}$$

$$+\lambda_2 \sum_{c=1}^{k} \sqrt{\sum_{i=1}^{E} (D^{(e_i)})^2_{cc}} + \frac{\lambda_3}{E} \sum_{i=1}^{E} ||V^{(e_i)} D^{(e_i)}||_1 + \frac{\lambda_4}{E} \sum_{i=1}^{E} \sum_{d=1}^{d_i} ||V^{(e_i)} D^{(e_i)}_{d\cdot}||_2, \tag{3.2}$$

where $V^{(e_i)}$ are orthogonal and $D^{(e_i)}$ are diagonal $K \times K$ matrices for all $e_i \in r_m \cup c_m$, $m \in 1, \ldots, M$, d_i is the dimension of entity e_i and $|| \cdot ||_F$ is the Frobenius norm. $\mathcal{M}^{(m)}$ is a matrix where an element $\mathcal{M}^{(m)}_{ij} = 0$ if the corresponding element $X^{(m)}_{ij}$ is unobserved and $\odot$ is element wise multiplication. MM-PCA has four regularization terms weighted by $\lambda_1, \ldots, \lambda_4$. These aim to achieve data integration by identifying shared factors, rank selection, sparse loadings and variable selection. The strength of MM-PCA compared to the other methods explored in this thesis is that it is explainable by design, at the cost of being more expensive to compute.

3.3 Simulated data

An augmented multi-view data setting was simulated with the purpose of comparing the performance of the four models dCMF, CMF, gCMF and MM-PCA with and without preprocessing of the data. Linear methods typically require the data to be centered and scaled to function at their best. The hypothesis is that dCMF is not as dependent on preprocessing as the other, linear methods, because of its more flexible design. Two models of each type, with the same hyperparameters, were trained on the data. For the first model of each type, no preprocessing was applied and for the second model the data was translated to zero mean and scaled to unit variance. A full list of hyperparameters is available in Appendix A.

3.3.1 Data simulation

The simulated data was structured like the augmented multi-view data example in Figure 1.1 c). The data setting was created by generating a latent factor matrix $U^{(e_i)}$ for each e_i, where $i = 1, \ldots, 5$ and e_1, e_2 are the row entities and $e_3, \ldots, e_5$ are the column entities. The data matrices were then created using the outer matrix product $X^{(m)} = U^{(r_m)} U^{(c_m)^T}$, where r_m and c_m are the row and column entities of matrix m. Similar to what was done in [MR19], each element in $U^{(e_i)}$ was simulated like $U^{(e_i)}_{ij} \sim U(0,1)$, where $U(0,1)$ is the uniform distribution between 0 and 1. Each latent factor $U^{(e_i)}$ had the same rank of 100 and row dimensions 800, 200, 300, 500 and 300 respectively. Then to impart sparsity in the matrices $X^{(m)}$, 30% of the elements in $U^{(r_m)}$ and $U^{(c_m)}$ were set to zero.

3.3.2 Evaluation

To test each model's performance on both raw and normalized data, a normalized variant of each matrix was created using $X^{(m)}_{\text{norm}} = (X^{(m)} - \mathbb{E}(X^{(m)}))/\sqrt{\text{Var}(X^{(m)})}$, where $\mathbb{E}(X^{(m)})$ and $\text{Var}(X^{(m)})$ is the empirical mean and variance of matrix $X^{(m)}$. A test set T was created by extracting 5% of the element indices (i,j) from $X^{(1)}$. A test and training matrix for $X^{(1)}$ and $X^{(1)}_{\text{norm}}$ were then created the same way, using the same T, as follows:

$$\begin{cases} X^{(1)}_{ij,\text{ test}} \leftarrow X^{(1)}_{ij}, & (i,j) \in T \\ X^{(1)}_{ij,\text{ test}} \leftarrow 0, & (i,j) \notin T \end{cases}, \quad \begin{cases} X^{(1)}_{ij,\text{ train}} \leftarrow X^{(1)}_{ij}, & (i,j) \notin T \\ X^{(1)}_{ij,\text{ train}} \leftarrow 0, & (i,j) \in T \end{cases}. \tag{3.3}$$

The models were trained using $X^{(1)}_{\text{train}}$ for both the raw and preprocessed data. The matrix $X^{(1)}$ was approximated by $X^{(1)\prime} = U^{(r_1)\prime} U^{(c_1)\prime^T}$. The errors were measured by their predicted Root Mean Square Error (pRMSE):

$$\text{pRMSE} = \sqrt{\frac{1}{|T|} \sum_{(i,j) \in T} \left(X^{(1)}_{ij,\text{ test}} - X^{(1)\prime}_{ij} \right)^2} \tag{3.4}$$

The training was repeated five times for each model and raw/preprocessed data, using the same training and test set and the same settings. Different local minima in the model loss would be found each time and the pRMSE was averaged over the five runs. To improve the interpretability of the results, the pRMSE values were scaled by the empirical standard deviation of $X^{(1)}$.

3.3.3 Data Integration Analysis

To investigate the ability to translate one data type to another, a data set of two matrices from the simulated data was created. The two matrices were both $X^{(1)}$, so even without any constraints to be identical, the two encoders should create overlap in the latent representations, which indicates translatability. This means that DTN finds them to be similar in the latent domain and can therefore be translated with minimal loss.

A DTN that used a latent domain of rank $k = 5$ was trained until convergence on the dataset described in the previous paragraph. The latent representations of the dataset of the encodings, $Z, Z' \in \mathrm{R}^{n \times k}$, were then observed. If the points in Z' could be moved closer to the points in Z by multiplying one of the columns with -1, this was done. This could be done because the networks are sign invariant.

3.4 MovieLens100k

The dataset MovieLens100k [HK15] is a common benchmarking dataset for recommender systems, consisting of three matrices and four entities. The main feature of MovieLens100k are the 100000 ratings from 943 users on 1682 movies. The ratings are complemented by two side information matrices. The first with information about the age, gender, zip code and occupation of the users and the second with information about the titles, release dates and genres of the different movies. Table 3.1 contains a summary of the dataset.

3.4.1 Preprocessing

The ratings are given on a scale of 1 to 5, and to simplify the problem of finding new movies that a user would like, the ratings were dichotomized such that ratings of 4 and above were set to 1 and ratings lower than 4 were set to 0. Unrated entries were set to "NA" to differentiate between unrated movies and disliked movies, which could be utilized by CMF, gCMF and MM-PCA. For dCMF, unrated entries were set to 0.

The user information matrix was encoded by creating seven age groups, with roughly the same number of users in each, and creating separate columns for the genders, age groups, zip codes and occupations. All categories were one-hot encoded to create a matrix of binary values with 943 users and 808 user features. The movie information matrix was encoded in a similar way by one-hot encoding the genres and release dates of the movies. Resulting in a matrix of binary values with 1682 movies and 259 movie features. This encoding is similar to that of [MR19] and [Don+17], but with lighter encoding of the movie features.

Table 3.1: Movielens100k information. Sparsity is the proportion of zeros in the data

Matrix	Row entity	Column entity	Row dim	Col dim	Sparsity	Data type
$X^{(1)}$	User	User features	943	808	0.9951	Binary
$X^{(2)}$	User	Movies	943	1682	0.9652	Binary
$X^{(3)}$	Movie features	Movies	259	1682	0.9895	Binary

3.4.2 Evaluation metric

Predictions are obtained through matrix completion by multiplying the latent representations learned by the models $X^{(2)\prime} = U^{(\text{User})} \cdot U^{(\text{Movies})^T} \in \mathbb{R}^{m \times n}$. The most important predictions from a recommender system are those with the highest values, which correspond to the top recommended items for a certain user. A full list of hyperparameters of the four models can be viewed in appendix Appendix A. Training and test data was randomly sampled for each run such that 95% of the positive ratings appeared in the training set and the remaining 5% in the test set, just like for the simulated data Equation (3.3).

The performance of a recommender system can be evaluated by comparing the top k predicted items to a user's known liked items. We used recall as the evaluation metric, which is good for implicit feedback. Other metrics, such as precision, are not suited because a zero rating can occur either because a user is uninterested in that item or they are unaware of it [MR19]. The matrix $X^{(2)\prime}$ contains the predicted ratings and $X_{i\cdot}^{(2)\prime}$ are the predicted ratings for user i, corresponding to the i-th row of $X^{(2)\prime}$. We sort $X_{i\cdot}^{(2)\prime}$ and call the indices of the k largest elements S_i^k and let S_i^{test} be the indices of the liked items in the test set for user i. Then the *recall at k* $R_i(k)$ for a user i can be defined as

$$R_i(k) = \frac{|S_i^k \cap S_i^{\text{test}}|}{|S_i^{\text{test}}|} = \frac{\text{Number of items in test set user } i \text{ likes of the top } k \text{ predictions}}{\text{Total number of items user } i \text{ likes in the test set}}.$$

The $R_i(k)$ was then averaged over all users i with liked items in the test set $|S_i^T| > 0$, to get the mean recall $R(k)$ for the model at a certain k. Like [MR19] and [Don+17], we measured the average performance over 5 runs (except for MM-PCA which was only trained once because of its slow computation time).

3.5 Omics data

The performance of dCMF, DTN and dCMF-LIME on omics data was explored using data collected from the global collaborative project The Cancer Genome Atlas (TCGA) [Wei+13] and the DisGeNET [Piñ+15] curated gene-disease associations. The TCGA data that was used consisted of 3599 patients with clinical features and RNA sequencing values for over 16000 genes. The clinical features consisted of mixed data types like patient cancer cohort (Table 3.2), gender, age at diagnosis and a number of pre-computed summary statistics such as clusters for copy numbers, gene mutations, and the like, see Appendix C for the full list. To connect genes to a set of diseases, the DisGeNET data set was used to extract correlations between genes and diseases. Through it, a binary table that indicates interactions between 9411 genes and 10370 diseases - not only cancer diseases - was attained. The full list of diseases can be found at the DisGeNET web page https://www.disgenet.org.

This data was used for two tasks. The first task was to benchmark the ability of dCMF and DTN to predict gene-disease association pairs in the DisGeNET matrix, compared to the baseline methods CMF and gCMF. MM-PCA was omitted because of its unfeasibly long computation time for the size of this problem. Similar to what was done for the previous benchmarks, 5% of the gene-disease associations were removed from the training data and placed in a test set, and replaced with 0's. For evaluation we used the metric *probability at k*, denoted $P(k)$ defined below in Section 3.5.2. In the past, dCMF has shown poor performance when the sparsity of two views deviate too greatly [MR19], since this creates a concatenated matrix with regions of greatly varying sparsity. To test if this makes any difference, a sparse variant of the omics data setting was used in addition to the one described in the previous paragraph. For this modification, the RNA sequencing data was replaced by the much sparser gene mutation matrix, with 2828 patients as the row entity and mutations of 2000 genes as the column entity. This matrix is also taken from TCGA and contains binary values if a mutation was observed for a certain gene for each patient.

The second task was instead to use the data to predict the cancer cohort of the patients and explain the predictions using dCMF-LIME. To accomplish this, the patients were split into a 90% training group and a 10% test group, where a dCMF model was used to predict the cancer cohort for each patient in the test group. After which explanations were computed for every prediction. The procedure is explained in more depth in Section 3.6.3.

Table 3.2: Types, counts and proportion of cancer cohorts in the TCGA data.

Cancer cohort	counts	Proportion
Acute Myeloid Leukemia	173	0.0481
Bladder Cancer	122	0.0339
Breast Cancer	840	0.2334
Colon Cancer	190	0.0528
Endometrioid Cancer	370	0.1028
Formalin Fixed Paraffin-Embedded Pilot Phase II	12	0.0033
Glioblastoma	166	0.0461
Head and Neck Cancer	303	0.0842
Kidney Clear Cell Carcinoma	476	0.1323
Lung Adenocarcinoma	352	0.0978
Lung Squamous Cell Carcinoma	258	0.0717
Ovarian Cancer	265	0.0736
Rectal Cancer	72	0.0200

3.5.1 Preprocessing

For the clinical features, any duplicate columns were removed and categorical features were replaced with their one-hot encodings. The patients that were not present in the RNAseq matrix were filtered out. To reduce the dimensionality of the gene expression data, the 2000 genes with the highest variance in their RNA sequencing values were selected and missing values were replaced by the mean for that specific gene. The DisGeNET gene-disease associations were filtered out such that only diseases associated to any of the selected genes were included, reducing the total number of diseases to 4796. Lastly, the columns of each matrix were scaled by their maximum absolute value to bring the data into the range $[-1, 1]$. This resulted in three matrices, described in Table 3.3.

The sparse variant of the data setting with the mutation data instead of the RNAseq, used the same 2000 genes with the highest variance in their RNA sequence expressions and the gene-disease matrix was therefore left unchanged. The patients were once again filtered out so only patients present in both $X^{(1)}$ and $X^{(2)}$ were included. This data is described in Table 3.4.

Table 3.3: Omics data features. Although the data set includes few matrices, they are rather heterogeneous.

Matrix	Row entity	Column entity	Row dim	Col dim	Data Type	Sparsity
$X^{(1)}$	Patient	Clinical features	2828	118	Mixed	0.8835
$X^{(2)}$	Patient	Gene (RNA)	2828	2000	Continuous	0
$X^{(3)}$	Disease	Gene	4796	2000	Binary	0.9975

Table 3.4: A more homogeneous sparse data set.

Matrix	Row entity	Column entity	Row dim	Col dim	Data Type	Sparsity
$X^{(1)}$	Patient	Clinical features	2828	118	Mixed	0.8835
$X^{(2)}$	Patient	Gene (Mutation)	2828	2000	Binary	0.9904
$X^{(3)}$	Disease	Gene	4796	2000	Binary	0.9975

3.5.2 Evaluation metric

A suitable evaluation metric for the gene-disease association task is *probability at* k, $P(k)$, as discussed in [ND14]. To compute it, each gene is ranked by its model predicated score for every disease. The empirical cumulative probability that a gene-disease pair is found below a threshold k is referred to as $P(k)$. It is averaged over all diseases to get a single value for the model at each k.

Consider a binary matrix $X \in \{0,1\}^{m \times n}$ that is predicted by a matrix $X' \in \mathbb{R}^{m \times n}$ from a matrix completion algorithm. To compute $P(k)$, determine the necessary permutations to sort the rows of X' in descending order and place them in the matrix R. Iterating row-wise through the index pairs (i, j), where $X_{ij} = 1$, record the corresponding values R_{ij} in a vector r of length n_{nz}, where n_{nz} is the number of non-zero entries in X. Finally, we let

$$P_i(R) = \text{Number of } i\text{'s in } R$$
$$P(k) = \frac{1}{n_{\text{nz}}} \sum_{i=1}^{k} P_i,$$

where $P(k)$ is the average *probability at* k over all columns in R. The algorithm is implemented in pseudo code below.

$P(k)$ reveals a few things. First, we note that a good algorithm would produce a high $P(k)$ for low values of k, and that all probabilities converge to 1 for high values of k. Since k denotes the genes, it is possible that not all genes are correlated to diseases to begin with, so we do not expect any algorithm to perfectly predict diseases for low k. Regardless, having a high value of $P(k)$ for low k means fewer incorrect predictions.

Algorithm 1 Algorithm to produce the *probability at* k metric

```
1: $X$ binary matrix, $X'$ real valued matrix
2: Let $r \in \mathbb{R}^{n_{\text{nz}}}$
3: $l \leftarrow 1$
4: for each $(i, j)$ such that $X_{ij} = 1$ do
5:     sort row $i$ of $X'$ in descending order, store the indices in $R$
6:     $r_l \leftarrow$ the index at which $l$ is found in $R$
7:     $l \leftarrow l + 1$
8: end for
9: Let $p \in \mathbb{R}^k$
10: for $i = 1...k$ do
11:     $p_i \leftarrow$ the number of $i$'s in $r$ normalized by $n_z$
12: end for
13: Return cumulative sum of $p$
```

Note that $P(k)$ requires an matrix prediction from an algorithm. In this thesis, when calculating $P(k)$, the algorithms have been trained using a training set, as described in section 3.5. When $P(k)$ was calculated, the reconstructed training set (X' in the description above) was compared to the test set (X above). This meant that the model had to learn what the missing elements in the training set were supposed to be.

3.6 Interpretability

To create explanations for the predictions of the dCMF algorithm we have used the Linear Interpretable Model-agnostic Explanations (LIME) [RSG16] framework, described briefly in Section 2.3.1. Because dCMF is not a classification or regression algorithm, for which LIME was originally intended, some special considerations are required. Only a single value in the output matrix can be explained at once, so in order to explain multiple predictions the same number of sparse linear models need to be trained. For this task we determined that the simulated data would not be as intuitive as the movie recommendation task, as most people have a fairly good grasp on why one movie would be recommended based on another. In addition, the data is already in an interpretable format, meaning it consists only of binary features that correspond to a user's features, liked movies or movie ratings, which are easy to interpret. This relieves the need to manipulate the data further to make it intuitive for a human. Once the LIME framework was implemented it was also tested using the Omics data to see if it could give any insights into that more complex problem.

3.6.1 dCMF-LIME

Once a dCMF model had been trained and the networks extracted we could put new data in conjunction with the old training data through the networks to get new predictions. This was used to create factorizations without having to retrain the networks. To create explanations for the prediction of an instance x, that instance is fed to the dCMF-LIME algorithm together with a function that creates interpretable samples $\hat{x}'$ and the same samples in the original data representation $\hat{x}$. An example of such a function is given in Section 3.6.3. Predictions y that are used as target values for the explainer model are computed using the extracted networks and the samples $\hat{x}$. The explainer is a linear ridge regression model [HK70] that is trained using the targets y as the dependent variable and the interpretable samples $\hat{x}'$ as the independent variables.

3.6.2 Movielens100k

The MovieLens100k data set, described more in detail in Section 3.4, had been encoded such that all features were binary and indicated simple things such as if a person belongs to a certain age group or if they have watched and liked a certain movie. Therefore the data was already in an interpretable format and LIME could be easily applied. Hyperparameters of the dCMF model can be viewed in Appendix A.

The input data for the explanations consisted of three persons filling in the movies they themselves have liked as well as their their age group, gender, occupation and a random zip-code. These three people are referred to as person A, B and C. See Appendix B for a complete list of their features. The data was already easily interpretable in its original representation, meaning no special manipulation was required to make it interpretable and we could simply say that $x = x'$. The top ten recommendations were computed for each person and the predicted rating of the top recommended item was explained.

To do this, two hundred samples $\hat{x}$ were drawn by setting a uniformly random number of non-zero elements to zero and targets y were computed by forwarding the $\hat{x}$ vectors through the respective networks and multiplying with the corresponding latent matrices. A ridge regression model was trained with regularization parameter $\alpha = 1$ and the locality was implemented by weighted sampling using the kernel

$$\pi_x(\hat{x}_i) = \exp\left(-D(x, \hat{x}_i)^2\right), \tag{3.5}$$

where $D(x, \hat{x}_i)$ is the cosine distance between x and $\hat{x}_i$, $i = 1, ..., 200$.

3.6.3 Omics data

The omics dataset and the RNA sequence data in particular, was not in a suitable format for LIME, unlike MovieLens100k and some special measures were required in order to format the data. The original data is described in more detail in Section 3.5. The input data consisted of rows from the patient features and RNA-sequence data that were withheld from the training data set, each row corresponding to one patient. The patient features were then reconstructed using the latent representations of the input data and explanations were made for the top predicted cancer cohort of that patient.

The main issue laid within RNA-sequence data. For the interpretable representation x' of x, the the highest expressed 100 and lowest expressed 100 genes of the patient, with respect to the RNAseq values, were picked out and labeled as 1 and 0 respectively resulting in a binary list of 200 genes. This was done to reduce the dimensionality of the data. The new data x' was perturbed in the same way as for the MovieLens100k data set by selecting a random number of positive elements and setting them to 0 to get $\hat{x}'$. The perturbed interpretable data was transformed back into the original data representation $\hat{x}$ by first returning all ones to their original value and setting the value of the zeros to 0 if their original value was larger than 0 and setting them to their original value otherwise. This was done in order to ensure that $\hat{x}$ would have both high and low gene expression values. The genes that were removed during the selection were set to 0. The target values y were computed using the data $\hat{x}$. Then the targets y and the interpretable representation of the perturbed data $\hat{x}'$ were used to train the linear ridge regression explainer model with regularization coefficient $\alpha = 1$. The data was sampled during training with weighted probability using the same kernel as for MovieLens100k (Equation (3.5)) and the same cosine distance measure.

4 Results

This section contains the results for the experiments in this thesis. First the benchmarking results from three data sets, the simulated data, MovieLens100k and the omics data are shown, followed by the explanations extracted from the Deep Collective Matrix Factorization (dCMF) models.

4.1 Simulated data

Figure 4.1 shows the benchmarking results on the simulated data. The first, most obvious thing to note, is that the prediction errors for Collective Matrix Factorization (CMF) and Group-wise sparse Collective Matrix Factorization (gCMF) extend outside the borders of the plot. From Table 4.1 we see that their normalized errors were 5435 and 108 for CMF and gCMF, respectively. Before limiting the maximum gradient magnitude (see Table A.3 for hyperparameters), their errors would be astronomically larger, especially for CMF ($\sim 10^{150}$), suggesting that the algorithms run into numerical difficulties without preprocessing on this data. On the normalized data, CMF and gCMF performed similarly, with CMF performing slightly better. Multi-group Multi-view Principal Component Analysis (MM-PCA) performed better than both CMF and gCMF even on the raw data (relative to the standard deviation of the normalized and raw data) and dCMF performed better still, with lower pRMSE/σ than all the other models. The last row of table 4.1 shows the relative improvement of the pRSME/σ. Of the two algorithms that did not fail without preprocessing, dCMF and MM-PCA, we see that dCMF benefited the most from normalizing the data, with a relative reduction of around 17%, where the error for MM-PCA decreased less, with around 8%.

Table 4.1: Errors for the four algorithms on the raw data with and without scaling of the standard deviation ($\sigma \approx 1.972$ for the raw data and $\sigma \approx 1$ for the normalized data) and error on the normalized data. The last row shows the relative change in pRMSE/σ.

	gCMF	CMF	dCMF	MM-PCA
pRMSE	212.24	10715	1.2934	1.6078
pRMSE/σ	107.64	5434.6	0.6560	0.8155
pRMSE/σ (normed data)	1.0488	0.9672	0.5447	0.7487
Relative pRMSE/σ change	0.0097	0.0001	0.8303	0.9181

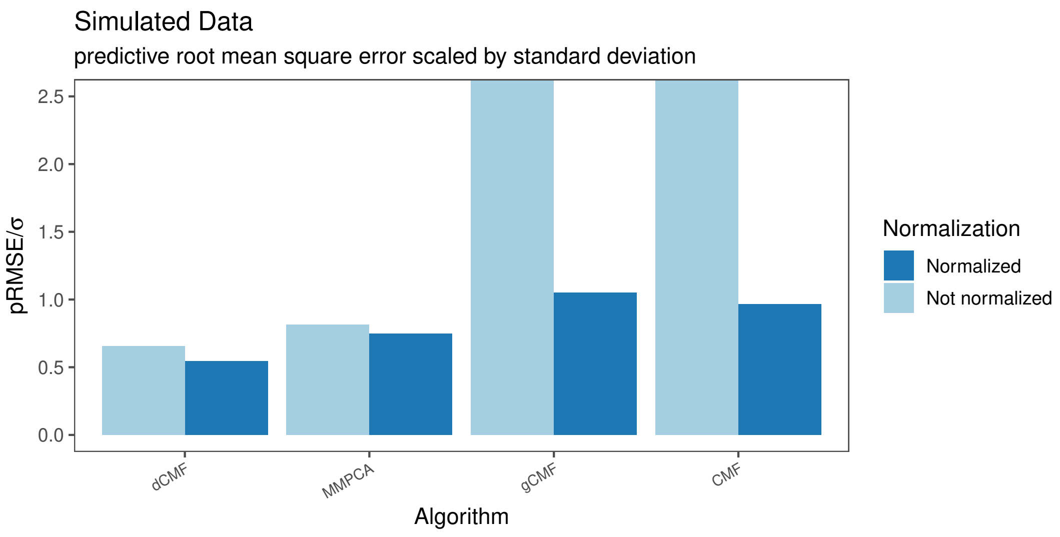


Figure 4.1: The results from the benchmark of the simulated data. The error of the algorithms on the raw data has been scaled by its empirical standard deviation, to show the relative improvement after preprocessing. Errors for CMF and gCMF on the raw data are of higher orders of magnitude than the rest of the errors. The plot is zoomed in to better show the more interesting results.

4.1.1 Data Integration Analysis

A Data Translation Network (DTN) that used a latent domain ($k = 5$) with a data set containing two identical matrices was trained until convergence. The latent representation of the data set according to each network was then considered. The resulting two groups of latent representations can be found in Figure 4.2.

Do note that this training was done with a latent dimension of 5, and the data was simulated using a higher dimension than that. Regardless, we expect that any data integration can be equally easy to do regardless of chosen latent dimension size.

If the peaks of the histogram could be moved closer by way of switching the sign of a column, that was done. The overlap in most - but not all - dimensions is considerable. It is also notable that the different encoders produced point clouds of similar shapes, even if they do not completely overlap each other. E.g the plot with horizontal axis z_4 and vertical axis z_3 overlaps less than most other pairs, with both clouds having a crescent shape with different tilts.

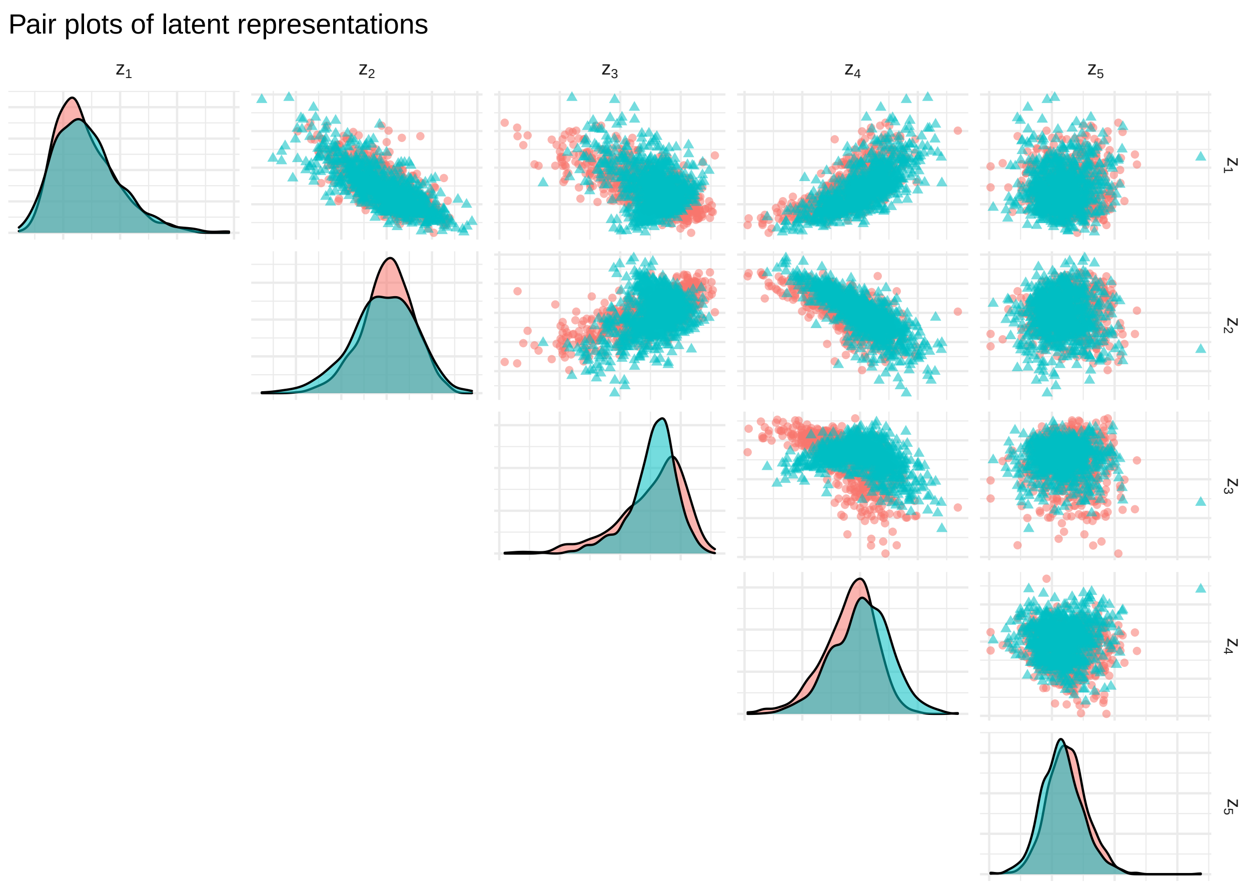


Figure 4.2: Pair plots of the latent representations produced by a DTN trained on the same data for both encoders with a 5 dimensional latent domain. In the off-diagonal plots, each point represents one of these dimensions, and color and shape differentiate the encoders. Diagonal plots are density distributions of the dimension for the two networks. If the peaks of the histogram could be moved closed by way of switching the sign of a column, that was done.

4.2 MovieLens100k

We see from the benchmarking results on the MovieLens100k dataset in Figure 4.3 that dCMF outclasses all the other algorithms by a large margin. It is interesting to see that CMF outperforms gCMF even though Klami et. al. [KBT14] vouch that gCMF should always be better or equal to CMF. These results are very much in line with those of [MR19] so they can be believed to be accurate. MM-PCA performed very well in this task as well, outperforming both CMF and gCMF.

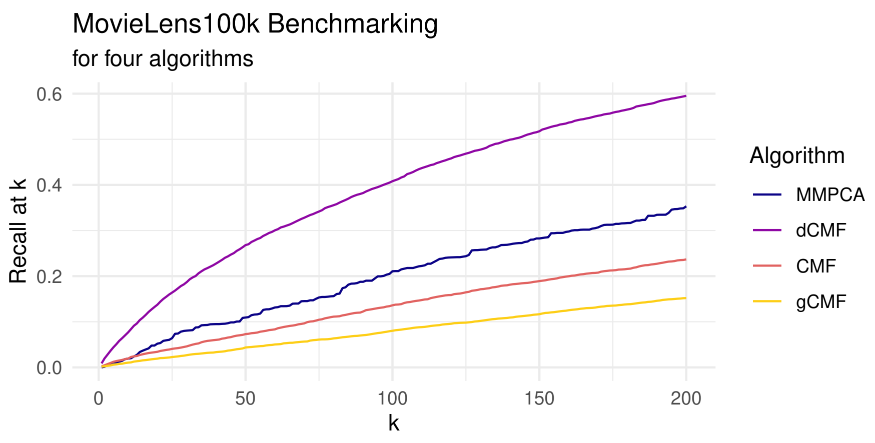


Figure 4.3: Benchmarking results of the four algorithms MM-PCA, dCMF, gCMF and CMF. The *recall at k* $R(k)$ is a measure of how correct the algorithm was in the top k results. It always increases with k and it should ideally be as large as possible even for small k. dCMF performs very well in this run, nearly doubling the next best alternative at $k = 200$.

4.3 Omics data

In order to evaluate the performance of DTN, we attempted to recreate the disease-gene matrix $X^{(3)}$ from Table 3.3, using the matrices from the same table as supplementary matrices. The evaluation used the $P(k)$ evaluation metric, described in Algorithm 1. For this test it is best if the $P(k)$ is large for small k, which means that a gene can be associated with a disease with a high probability. When k gets larger it becomes harder to say if there exists a strong correlation between that gene-disease pair. DTN was compared to the previously established algorithms CMF, gCMF and dCMF. The result can be found in Figure 4.4.

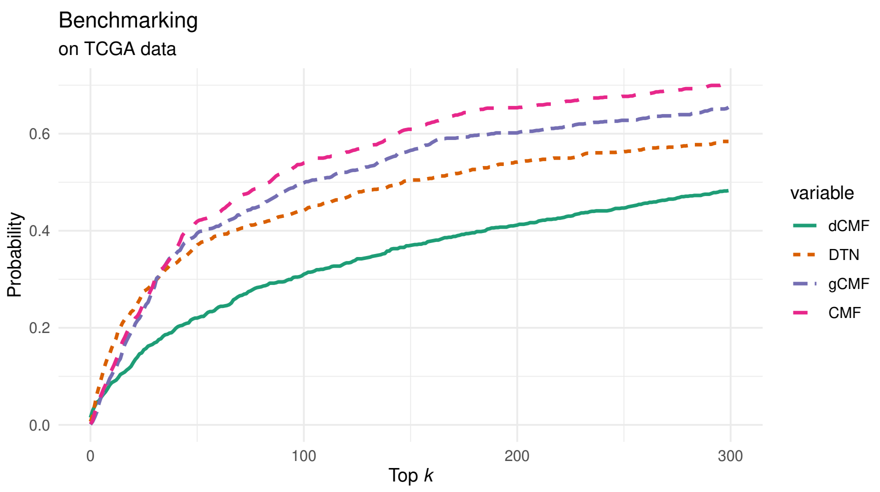


Figure 4.4: The $P(k)$ metric is plotted for 4 different algorithms. In this plot, the algorithms rank which genes correlate with a given disease. Since most genes do not correlate with most diseases, and some genes are not correlated with any disease at all, we do not expect any algorithm to reach high probabilities at low values of k. The max k allowed here is 2000, at which all probabilities converge at 1. However, it is much more important to have a high probability at low k. CMF is the top contender, followed closely by gCMF, then DTN, at the bottom we find dCMF.

To examine the lower performance of dCMF more closely, a modified data set was used, described in Table 3.4. For this comparison, only DTN and dCMF were included because they are the most interesting in the context of this thesis. The result can be found in Figure 4.5.

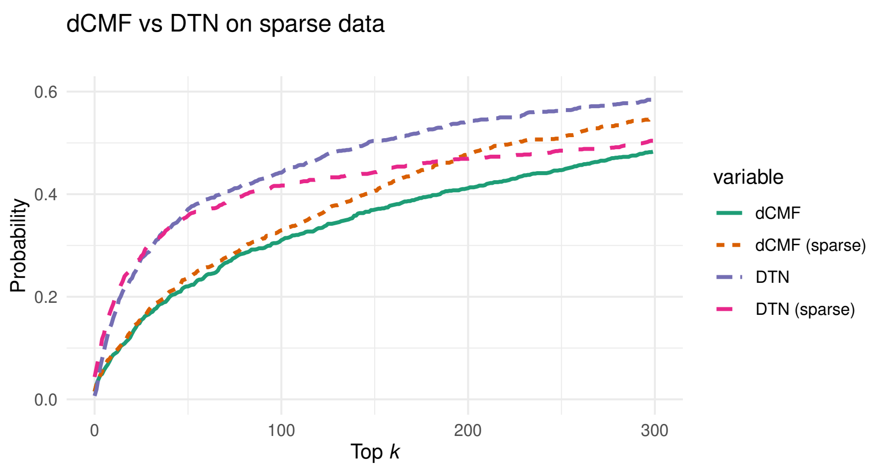


Figure 4.5: DTN and dCMF are compared more directly, both with the heterogeneous data set from Table 3.3 and from the more homogeneous sparse data set from Table 3.4. Some improvement can be seen for dCMF, while DTN seems to have lost some accuracy for larger values of k.

4.4 Interpretability

In this section we showcase the performance of dCMF-LIME on two problems; The MovieLens100k recommender problem and a cancer prediction problem using the omics data. The top ten explanations are shown in bar charts where blue bars correspond to features that increase prediction scores and red bars correspond to features that decrease the prediction scores.

4.4.1 MovieLens100k

The tables 4.2, 4.3 and 4.4 show the top ten recommendations for person A, B and C respectively and the corresponding figures 4.6, 4.7 and 4.8 show explanations for the top recommendation for each person. The recommended items are somewhat similar for all three test subjects. Table 4.5 was created by getting the recommendations for an input consisting of only zeros, meaning no watched movies or user features. This is often referred to as the cold-start problem. The table can be compared to the other three to find which items are biased by the model and which items are user specific. We see that most of the top recommended items for each person are present in Table 4.5, showing the presence of a strong bias in the model. This can also be seen by comparing the magnitude of the explanations to the recommendation scores. The explainer model coefficients are mostly in the range of 0.0005 to 0.002 and the recommendations for the zero-input are around 0.13 to 0.10. This implies that a movie that is not naturally favourably scored by the model would require the presence of many contributing features to be moved to the top of the recommendations.

Recommendations, person A	Score
Silence of the Lambs, The (1991)	0.1487332
Psycho (1960)	0.1441530
Return of the Jedi (1983)	0.1376552
Rear Window (1954)	0.1193775
Graduate, The (1967)	0.1181024
When Harry Met Sally... (1989)	0.1161001
Dances with Wolves (1990)	0.1160568
Fugitive, The (1993)	0.1144019
Forrest Gump (1994)	0.1134828
GoodFellas (1990)	0.1118130

Table 4.2: Top 10 recommended movies for person A. Viewed movies are filtered out from the recommendations.

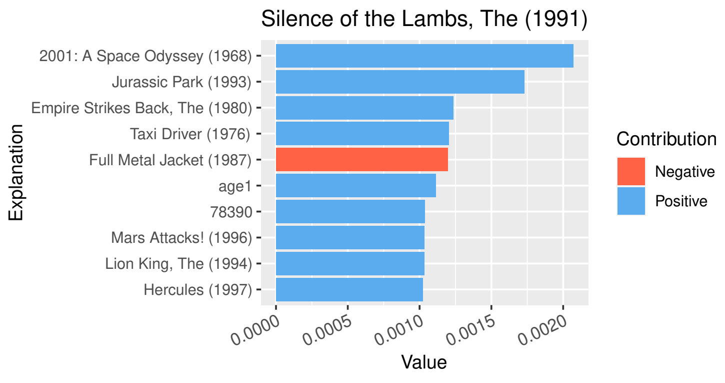


Figure 4.6: Explanations for the top recommendation, The Silence of the Lambs (1991), for person A. Blue bars indicate that the feature has a positive contribution to the prediction and the red bars indicate the opposite.

Recommendations, person B	Score
Star Wars (1977)	0.1325764
Silence of the Lambs, The (1991)	0.1322694
Psycho (1960)	0.1296078
Return of the Jedi (1983)	0.1256587
Raiders of the Lost Ark (1981)	0.1208767
Empire Strikes Back, The (1980)	0.1122722
Rear Window (1954)	0.1118609
2001: A Space Odyssey (1968)	0.1116455
Graduate, The (1967)	0.1089866
Fugitive, The (1993)	0.1053607

Table 4.3: Top 10 recommended movies for person B. Viewed movies are filtered out from the recommendations.

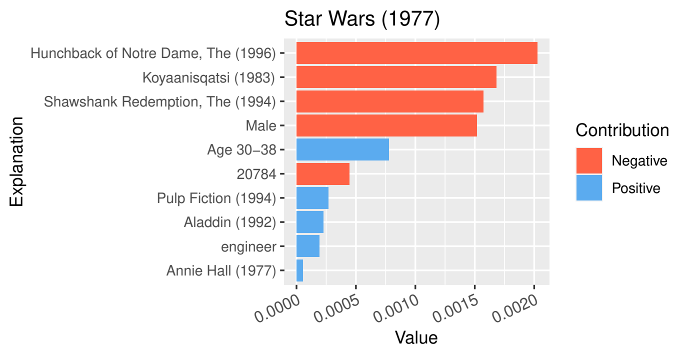


Figure 4.7: Explanations for the top recommendation, Star Wars (1977), for person B. Blue bars indicate that the feature has a positive contribution to the prediction and the red bars indicate the opposite.

Recommendations, person C	Score
Psycho (1960)	0.1484594
Silence of the Lambs, The (1991)	0.1473105
Raiders of the Lost Ark (1981)	0.1338543
Dances with Wolves (1990)	0.1206117
Rear Window (1954)	0.1199470
Graduate, The (1967)	0.1165255
When Harry Met Sally... (1989)	0.1156397
Fugitive, The (1993)	0.1150441
Shawshank Redemption, The (1994)	0.1133415
North by Northwest (1959)	0.1131875

Table 4.4: Top 10 recommended movies for person C. Viewed movies are filtered out from the recommendations.

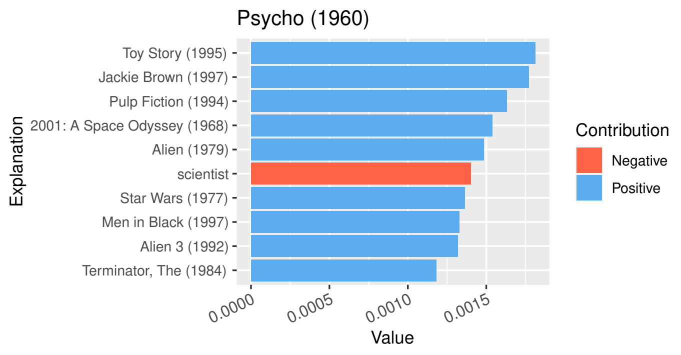


Figure 4.8: Explanations for the top recommendation, Psycho (1960), for person C. Blue bars indicate that the feature has a positive contribution to the prediction and the red bars indicate the opposite.

Table 4.5: Top 10 recommended movies for an input of only zeros, i.e no liked movies and no user features. These recommendations can gives an idea about which items are biased by the models and which items are user specific by comparing to previous tables.

Recommendations, zeros	Score
Silence of the Lambs, The (1991)	0.1315258
Psycho (1960)	0.1287898
Raiders of the Lost Ark (1981)	0.1213705
Rear Window (1954)	0.1089659
Fugitive, The (1993)	0.1082732
Graduate, The (1967)	0.1056070
Shawshank Redemption, The (1994)	0.1040959
Dances with Wolves (1990)	0.1037529
GoodFellas (1990)	0.1019981
When Harry Met Sally... (1989)	0.1018460

Because the recommended items are so similar for the three persons, another dCMF model was trained on the same data but with fewer epochs to prevent overfitting. Besides The English Patient from 1996 being the top recommended item for all three persons (by a large margin) the predicted recommendations deviate more from each other than than those given by the first model. Some items are present in the recommendations for more than one person, but the order they appear in differs more than for the first model, even though this model shows signs of the same problem of recommending similar items to different people.

Table 4.6: Recommendations for the three test subjects on a model that was trained for fewer epochs to reduce overfitting. The recommendations deviate more from one another with this model than for the first one. The items are still similar but their order differentiates more for the different persons.

Recommendations, person A	Recommendations, person B	Recommendations, person C
English Patient, The (1996)	English Patient, The (1996)	English Patient, The (1996)
Back to the Future (1985)	Good Will Hunting (1997)	Raiders of the Lost Ark (1981)
Silence of the Lambs, The (1991)	Raiders of the Lost Ark (1981)	Silence of the Lambs, The (1991)
Monty Python and the Holy Grail (1974)	Schindler's List (1993)	Godfather, The (1972)
Schindler's List (1993)	Casablanca (1942)	One Flew Over the Cuckoo's Nest (1975)
Jerry Maguire (1996)	Star Wars (1977)	Schindler's List (1993)
One Flew Over the Cuckoo's Nest (1975)	Monty Python and the Holy Grail (1974)	Casablanca (1942)
Groundhog Day (1993)	Godfather, The (1972)	Back to the Future (1985)
Blues Brothers, The (1980)	Citizen Kane (1941)	Gandhi (1982)
Casablanca (1942)	Gandhi (1982)	Citizen Kane (1941)

4.4.2 Omics data

For the cancer prediction problem using the omics data, the dCMF algorithm resulted in a 84.4 % prediction accuracy on our test data. Table 4.7 shows statistics over the predictions of the model. Over all, the algorithm is biased towards predicting the more common cancer cohorts and it never predicts the less common *bladder cancer*, *formalin fixed paraffin-embedded pilot phase II*, *lung squamous cell carcinoma* and *rectal cancer.* The exceptions to this are *head and neck cancer* which is predicted twice as often as it occurs in the data, even though it is only a little more common than *lung squamous cell carcinoma*, and *acute myeloid leukemia*, which is correctly predicted every time even though it only occurs 15 times in the test set. Almost all cases of *bladder cancer* and *lung squamous cell carcinoma* are falsely predicted to be *head and neck cancer* and *rectal cancer* is always predicted to be *colon cancer* which intuitively could share similarities. In Figure 4.9 the explanations for *colon cancer* and *rectal cancer* are compared and it becomes apparent why the model predicts *colon cancer* over *rectal cancer*, because most of the same features are present for both cohorts, but with much larger magnitudes for *colon cancer.* The explanations in the figure are only from clinical features of the patients and the RNAseq-values did not appear after around the top 20 predictions and that the different classes in the clinical features are very strong predictors. We also notice how the boxes in the box-plot are very narrow for the clinical features (they are significantly larger for individual genes RNAseq-values), again reinforcing that the clinical features are powerful predictors.

Table 4.7: The table shows number of predictions the model made of each cohort, as well as the number of occurrences in the test data that the predictions were made on (very similar distribution in the training data). It also shows the True Positive Rate (TPR) and False Positive Rate (FPR) of the predictions, computed on the test set. The most common mistake is also given for each cohort in the cases where such mistakes were made by the model.

	Acute Myeloid Leukemia	Bladder Cancer	Breast Cancer	Colon Cancer	Endometrioid Cancer	Formalin Fixed Paraffin-Embedded Pilot Phase II
# predictions	15	0	85	23	32	0
# in test set	15	14	82	15	31	1
TPR	1	0	0.9878	1	0.9677	0
FPR	0	0	0.1220	0.3478	0.0625	0
Commonly mistaken for	-	Head and Neck Cancer	Head and Neck Cancer	-	Ovarian Cancer	Kidney Clear Cell Carcinoma

Glioblastoma	Head and Neck Cancer	Kidney Clear Cell Carcinoma	Lung Adenocarcinoma	Lung Squamous Cell Carcinoma	Ovarian Cancer	Rectal Cancer
22	64	46	47	0	25	0
22	32	44	42	28	25	8
1	1	1	0.9524	0	0.96	0
0	0.5	0.0435	0.1490	0	0.04	0
-	-	-	Endometrioid Cancer	Head and Neck Cancer	Endometrioid Cancer	Colon Cancer

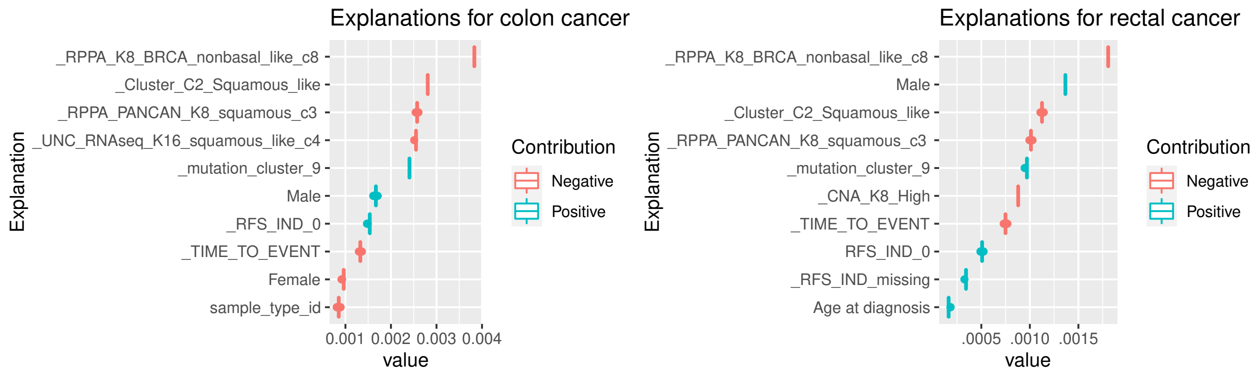


Figure 4.9: Comparison between the explanations for colon cancer and rectal cancer for all patients in the test set, with 0s excluded. RNAseq values of genes do not appear in the explanations until around top 20. The model mistook rectal cancer to be colon cancer in every case. From these explanations we can see that most of the important features are the same for both cohorts, but with larger values for colon cancer. Zeros are excluded from the explanations. Note that the x-axes are different for the two figures.

Below, in Figure 4.10 and Figure 4.11, the predictions for patients A and B are explained using the dCMF-LIME framework described in Section 3.6.1. Patient A was correctly predicted to have *ovarian cancer* and from Figure 4.10 we can see an approximation of what features the model most strongly correlates with *ovarian cancer*. Some features such as the patient being female and the probability increasing with age are intuitive contributors, but it is also possible to see less obvious contributing factors, such as a missing *RFS_IND*-value for the patient or *OV-like c16* methylation. A few genes also appear in the top ten contributing factors but with a negative contribution, meaning these genes suggest that ovarian cancer is less likely.

From Table 4.7 we saw that *head and neck cancer* had a high number of false positives. Patient B was wrongly predicted to have *head and neck cancer* by the model when they actually had *lung adenocarcinoma*. Figure 4.11 shows explanations to why the model gave the prediction it did. Unlike for patient A there are no sanity check features such as *ovarian cancer* being correlated with female patients and the results are therefore harder to analyze without field expertise. A method for analyzing gene feature importance is suggested in the discussion chapter Section 5.4.2 but it was deemed outside the scope of the thesis.

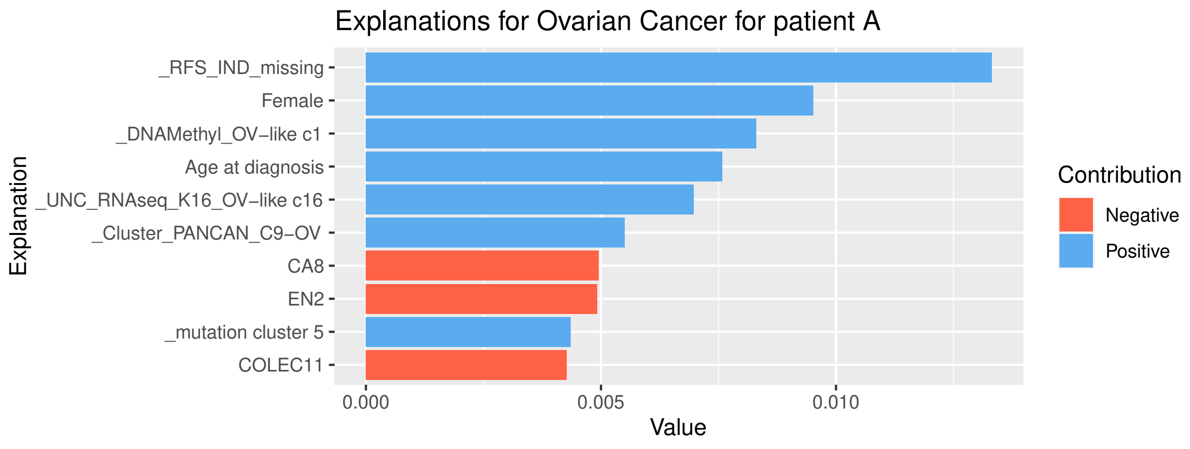


Figure 4.10: Patient A was correctly predicted to have *ovarian cancer*. The explanation for that prediction are described by the figure. Blue bars indicate that the feature has a positive contribution to the prediction and the red bars indicate the opposite. Features with a leading underscore indicate clinical features.

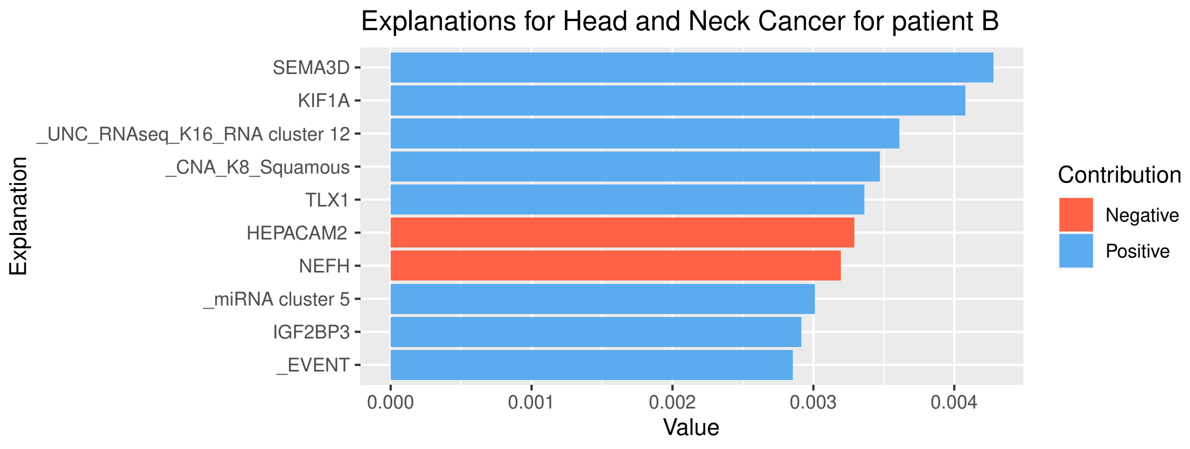


Figure 4.11: Patient B was erroneously predicted to have *head and neck cancer*, when they actually had *lung adenocarcinoma*. The explanation to that prediction are described in the figure. Blue bars indicate that the feature has a positive contribution to the prediction and the red bars indicate the opposite. Features with a leading underscore indicate clinical features.

5 Discussion

In this section we will discuss the results from the previous section, why the results came out as they did, remaining questions as well as possible improvements for future work. First the benchmarking results will be discussed for the three benchmarking datasets, the simulated data, MovieLens100k and the omics data. Following that, the interpretability of the Deep Collective Matrix Factorization (dCMF) models is discussed and the section concludes with a discussion about the data translation networks that were introduced in this thesis.

5.1 Simulated data

The results we saw for the effects of preprocessing in Figure 4.1, in this case a normalization of the data to zero mean and unit variance, were quite aligned with our initial predictions. The expectation was that the linear methods would perform significantly better when the data was normalized, which we saw was the case, especially for Collective Matrix Factorization (CMF) and Group-wise sparse Collective Matrix Factorization (gCMF), that failed to find a good factorization at all when the data was not preprocessed. What we did not expect however, was that dCMF would benefit more from preprocessing than Multi-group Multi-view Principal Component Analysis (MM-PCA), which is a linear model. One hypothesis as to why dCMF performed worse when the data was not centered and normalized, relates to the use of neural networks and the *vanishing gradient problem* that is an effect of the chain-rule, that occurs when training deep neural networks using gradient methods. Because the activation functions in each layer in the autoencoders are all tanh, their derivatives become very small for inputs far from zero. This makes the networks take longer to train, than if the gradients were larger, which eventually happens with time. But this time can be significantly shortened by normalizing and centering the input data before training the network. This could mean that, when the data was not preprocessed, the autoencoders were underfitted and the performance was worse because of that.

CMF and gCMF failed when the data was not normalized, as pointed out earlier. The errors showed in Table 4.1 were after a hard limit had been set on the maximum magnitude of the gradients during training. Without it, the errors were even larger, suggesting a numerical failure in the case of CMF, where the predicted Root Mean Square Error (pRMSE) reached around 10^{150}. gCMF, which is supposed to be more stable [KBT14], still had issues during training, but with significantly smaller errors around 10^5. The gradient limit improved the performance significantly for both algorithms, but it was still evident that no good factorization could be found. According to [KBT14], CMF breaks down when there exists latent factors that are not shared by all matrices. Which is the case here with $U^{(e_4)}$ and $U^{(e_5)}$ only being present in one matrix respectively. gCMF should not have this same problem because it can find individual or partially shared latent features, which could explain why it works better when the data is not normalized. The reason it does not find a very good factorization either, despite being much better, could be because the variational Bayesian inference it uses to find the separate factorizations can get stuck in a local optima during training. On the normalized data, both CMF and gCMF performed fairly well, although worse than the other methods analyzed in this thesis. It is curious that CMF slightly outperformed gCMF, however, when gCMF should have the advantage of being able to find separate low-rank structures. This is odd since the preprocessing should not change the fact that the data is constructed using some private factors which CMF is not supposed to handle well.

MM-PCA performed around 25% better than CMF and gCMF on the normalized data. This is despite being restricted to five principal components, due to computational constraints, when the other methods were set to the real rank of the latent factors of 100. MM-PCA could have an advantage because it attempts to find sparse latent structures and the factors we simulated had around 30% sparsity.

5.2 MovieLens100k

In this section we discuss the benchmarking results on the MovieLens100k data results from Figure 4.3. It is noteworthy how CMF performs better than gCMF when the authors of gCMF [KBT14] say it should always be better than CMF or equal in the worst case, since if no group-wise sparse data is found it should be identical. The fact that they perform differently indicates that group-wise sparse structures could be found in the data, but that it did not make for any improvement in the recall of the movie recommendations. These results, including the $R(k)$ for dCMF fall in line with those from [MR19], which indicates that they are trustworthy.

In this test we saw that MM-PCA performed very well once again. The MovieLens100k data is very sparse, and because MM-PCA has an additional sparsity assumption, that neither CMF or gCMF have, it might be a better fit for this problem. When $R(k)$ is calculated, we simply sort the predictions in descending order, which means that sparsity in the predicted ratings is not a problem. It might even improve the performance by suppressing items that the model deem unimportant.

dCMF on the other hand is leagues ahead of the competition for this problem. All three matrices, ratings, user features and movie features, are also similarly sparse, so no problems arise due to that either.

5.3 Omics data

In the low k domain the Data Translation Network (DTN) algorithm is outperforming its competitors, suggesting that it can find the diseases that are strongly correlated with genetics. However, increasing k gives the linear methods CMF and gCMF an advantage, suggesting perhaps that they can better harness the full data set.

It is surprising that dCMF does not perform better, although we suspect that it has to do with its difficulty to deal with data sets that include different types of data. The data in this data set is quite heterogeneous, with some fully dense continuous data and some sparse binary data. It is a known weakness of dCMF [MR19].

We also see once again that gCMF performed worse than CMF, likely for similar reasons as discussed in the above two sections.

5.4 Interpretability

The models were interpreted using the dCMF-LIME method outlined in section Section 3.6.1 on the MovieLens100k and omics datasets. In this section, the explanations from the dCMF-LIME models as well as the explained models' perceived rationale are discussed.

5.4.1 MovieLens100k

For the MovieLens100k data, it was immediately evident that the dCMF model did not generalize well for new data. The predictions for all three subject were similar with only a few unique recommended items and the differences were mostly because items already rated had been filtered out of the results. It should be noted that the all three test subjects are quite similar: young men with occupations in engineering/academia and a somewhat similar taste in movies. When observing the recommendations for the training data these tendencies are not as evident, and our hypothesis is that this could be due to overfitting in the autoencoders. Another dCMF model was trained for 100 epochs (as opposed to up to 2000 or until convergence of validation data) to test if the effect was indeed from overfitting. The resulting predictions did differ more from each other, even though most movies were recommended for more than one of the test subjects. This suggests that overfitting likely plays a role, but that it is not the only problem, since the model gives more varied recommendations for the data it was trained on. This raises the question if this way of using an already trained model is unsuitable for dCMF and if it is necessary to train a new network if one wants to factorize new data. There is also the possibility that the recommendations are similar due to the relatively small number of rated movies in the test persons compared to that of the training data (where the minimum was 20 items).

When it comes to the explanations themselves it is actually surprisingly hard to determine if they are reasonable. At a first glance they look fairly good, but at the same time we can see that family films, like Toy Story giving a positive contribution to horror movies like Psycho for person C, and for person B most explanations are negative for why Star Wars (1977) was recommended. For most of the top recommended items for each person, the explanations were quite similar and by comparing the magnitude of the prediction coefficients to the recommendation scores it is apparent that even by removing several of the contributing factors, the recommendation would not change much. This once again reinforces that there is a large bias for which movies are recommended and that that bias is hard to sway.

5.4.2 Omics Data

For the omics data, we saw a surprisingly high prediction accuracy of 84.4% and several cancer cohorts could be detected without error. However, a few cancer cohorts were never predicted and *head and neck cancer* was predicted twice as often as it occurred in the test data. Some of the cancer types that were wrongly predicted did share some similarities to other cohorts, that the model often mixed them up with, such as *rectal cancer* being mistaken for *colon cancer* and *ovarian cancer* being mistaken for *endometrioid cancer* and vice versa. It is understandable that the model could have issues distinguishing between these cohorts and Figure 4.9 showed what features were the most important for the model in order to diagnose a patient with *colon-* and *rectal cancer*. From that figure it was evident that the model did in fact increase its prediction score for *rectal cancer* but the same predictors were stronger for *colon cancer*, which led to *rectal cancer* never being predicted.

The explanations for the different cohorts were quite useful for giving a sanity check to the model performance, i.e. did the model look at sensible features when making its predictions? For certain cohorts, like breast- and ovarian cancer it was obvious that the gender of the patient should be an important predictor, which was seen in Figure 4.10. In this sense the explanations of the more complex dCMF model were able to give an intuition into how the model works, without having prior field knowledge. The explanations (especially those in Figure 4.9 that included all patients in the test set) show that the clinical features are very important for what cohort is predicted. Many of these clinical features are classes and clusters that had been found in the RNA-sequencing, methylation, mutation, copy number, etc. of the patients genes, by the curators of the dataset. Individual gene RNAseq-values did not appear in the explanations until around the top 20.

5.5 Data Translation Network

In this section, we discuss the implementation of DTN, its ability to integrate the simulated data and its performance on the omics gene-disease prediction test.

5.5.1 Implementation

Implementing a more sophisticated network architecture than the fully connected feed forward model used in dCMF was an ambition from the start during our thesis work. Finding inspiration in the networks used in natural language processing was an early attempt at improving the network. However, the architectures can not be transferred without thought. In natural language processing, data is homogeneous by design. However, within data integration, data types can be heterogeneous. One matrix might be fully dense with continuous values, whereas the next matrix might be binary with a high sparsity. The algorithm must work in all these scenarios simultaneously. It seems plausible that a fully connected forward propagation model is the best choice considering the need for flexibility, as any architecture that relies on remembering the last element for a number of steps, like a recurrent neural network, would likely lose any advantage in very sparse matrices that can appear in the data integration domain.

The role of DTN can be compared to the role of a STYLE-GAN[KLA18], which also attempts to recreate the distribution of one data entity but with the features of another. A high level explanation of their architecture also seems similar, in that both networks embed data in a latent domain and generate new data using latent representations. However, like above, the heterogeneous matrices demand more flexibility in our domain.

The loss function defines what the goal of the network is, and it is likely that a more complex loss function could be useful for the DTN. The more detailed specification of what is expected for the network, the better loss function can be defined, and considering that it is not obvious what the translated data should look like, a general distribution measure is perhaps a useful starting point. It is possible that adding a shared output layer between the latent domain and the generators could help with integrating, like previous work [LBK17][KLA18].

5.5.2 Data Integration Analysis

Considering the latent representations of observations from the simulated dataset, shown in Figure 4.2, a few observations can be made. Our initial hypothesis was that if the latent representation of two points from two different datasets lie close to each other, then a generator should be able to reconstruct observations that look like the original data corresponding to this generator regardless of which dataset the point originated from. In turn, if each generator is good at turning a latent sample into a realistic one, two nearby latent representations should produce similar but not equal reconstructed observations. That means that two points in close proximity in the latent domain should be points that can be well translated from one domain to another. The findings in Figure 4.2 agree with this sentiment, as we would expect that a network that takes two identical matrices as input should be able to translate from one dataset to another.

As for why the latent representations do not overlap fully, we consider the problem of encoding an observation of dimension n into a smaller dimension k a problem of compression with minimal loss. In particular, the distribution of observations in the latent domain should be such that the the most prominent, if not each, unique feature in each observation can be mapped to a region in latent space, creating feature-region pairs. The encoder then attempts to learn that mapping. However, due to the high degree of freedom in a neural network, there is bound to exist multiple satisfactory solutions that each compress the data with near equally low loss. In our implementation, there are no restrictions on the encoders or decoders, so they will each learn unique but similar mappings due to chance.

5.5.3 Omics data performance

The results presented in Section 4.3 show that the $P(k)$ for low values of k is very much comparable to CMF, which performed best in this task. It is slightly surprising that DTN performs as well as it does in this area, as the algorithm has not explicitly been trained at performing matrix completion. However, it is not unreasonable that the latent information can be useful in filling out the empty spots of a matrix.

Considering that not all diseases have a genetic origin, it is not obvious how high the highest probability should be for low k, but it is not immediately obvious that any of the algorithms produce a particularly strong result. Perhaps it is due to the side matrices not providing enough information to be of significant help. It is encouraging, at least, that the relative performance of DTN looks to be comparable to the other algorithms.

The performance decrease for dCMF is notable, and it matches the hypothesis that the variable sparsity remains problematic for the algorithm. Utilizing side information matrices makes the data harder to work with and in its current state DTN can only translate from one view to another without using information present in any of the other matrices. The model could be changed to translate between concatenated entity matrices, like those used in dCMF, but that would increase the number of datasets to train and consequently the training time and complexity of the model. In our implementation we prioritized simplicity over generality.

6 Conclusion and future work

Our initial hypothesis was that dCMF would outperform the linear baseline algorithms both with and without preprocessing. We saw that this was indeed the case using a simulated multi-view data set and we also saw that CMF and gCMF encountered numerical errors when the data was not preprocessed. On the MovieLens100k benchmark dCMF once again significantly outperformed the other methods. On the omics data on the other hand, dCMF performed the worst out of the four algorithms. The data for this test was highly heterogeneous in sparsity, which is a known weakness for dCMF so to test this another omics dataset with more homogeneous sparsity was created. On this data dCMF performed better, but still worse than the baseline methods.

We were able to create satisfactory explanations for the predictions of the dCMF algorithm for both the MovieLens100k dataset and the omics data using dCMF-LIME. We saw that there was a strong bias toward certain movies for the MovieLens100k model and we saw that the clinical features were very powerful predictors in the cancer prediction task. Some explanations do however require field knowledge to effectively take advantage of such as the RNAseq-values of genes in the omics data. The biological effect of changing a single gene can be hard to predict, therefore certain tools can be very useful when analysing a set of genes. Gene Set Enrichment Analysis (GSEA) [Sub+05], given a large set of genes, aims to find over-represented genes that might correlate with diseases. GSEA could be used to gain a macro-understanding of the explanations by supplying the method with the genes with the highest explaining power. Another way to analyze the explanations could be to attempt to cluster them, to see if there are different types of gene-compositions that lead to the same types of cancer.

The network that we put forward in this thesis, the DTN, performed well in terms of matrix completion, comparable to the baseline methods of CMF, gCMF and dCMF. We showed that unlike for dCMF, views with different sparsity do not affect the result. We showed that given two matrices of the same data, it could produce latent representations that overlapped almost entirely, which we interpret as the two matrices being integratable, and that any two matrices that match this behaviour should be too. Future improvements in this area could include a hyperparameter optimization scheme or finding parts of the Generative Adversarial Network (GAN)-type architectures that can be included in this model. Overall, it seems that solutions based on AI holds some promise in the domain of solving matrix factorization problems, and it is possible that it can bypass the need of finding matrix factors to begin with.

References

[BGY13] G. Bouchard, S. Guo, and D. Yin. Convex collective matrix factorization. *Journal of Machine Learning Research* **31** (2013), 144–152. ISSN: 15337928.

[DBH18] F. K. Dosilovic, M. Brcic, and N. Hlupic. Explainable artificial intelligence: A survey. *2018 41st International Convention on Information and Communication Technology, Electronics and Microelectronics, MIPRO 2018 - Proceedings* (2018), 210–215. DOI: `10.23919/MIPRO.2018.8400040`.

[Don+17] X. Dong et al. A hybrid collaborative filtering model with deep structure for recommender systems. *31st AAAI Conference on Artificial Intelligence, AAAI 2017* (2017), 1309–1315.

[HK15] F. M. Harper and J. A. Konstan. The movielens datasets: History and context. *ACM Transactions on Interactive Intelligent Systems* **5**.4 (2015), 1–19. ISSN: 21606463. DOI: `10.1145/2827872`.

[HK70] A. E. Hoerl and R. W. Kennard. Ridge Regression: Biased Estimation for Nonorthogonal Problems. *Technometrics* **12**.1 (1970), 55–67. ISSN: 15372723. DOI: `10.1080/00401706.1970.10488634`.

[IR10] A. Ilin and T. Raiko. Practical approaches to principal component analysis in the presence of missing values. *Journal of Machine Learning Research* **11** (July 2010), 1957–2000. ISSN: 15324435.

[Jas12] {Ryan Jasper Snoek and Hugo Larochelle and Adams. P. "Practical Bayesian optimization of machine learning algorithms". 2012, pp. 2951–2959.

[Kal+19] J. Kallus et al. MM-PCA: Integrative Analysis of Multi-group and Multi-view Data (2019). arXiv: `1911.04927`. URL: `http://arxiv.org/abs/1911.04927`.

[KBT14] A. Klami, G. Bouchard, and A. Tripathi. "Group-sparse embeddings in collective matrix factorization". *2nd International Conference on Learning Representations, ICLR 2014 - Conference Track Proceedings.* Dec. 2014. arXiv: `1312.5921`. URL: `http://arxiv.org/abs/1312.5921`.

[KLA18] T. Karras, S. Laine, and T. Aila. A Style-Based Generator Architecture for Generative Adversarial Networks. *CoRR* **abs/1812.04948** (2018). arXiv: `1812.04948`. URL: `http://arxiv.org/abs/1812.04948`.

[Kra91] M. A. Kramer. Nonlinear principal component analysis using autoassociative neural networks. *AIChE Journal* **37**.2 (1991), 233–243. ISSN: 15475905. DOI: `10.1002/aic.690370209`.

[KS12] Y. J. Ko and M. Seeger. "Large scale variational Bayesian inference for structured scale mixture models". *Proceedings of the 29th International Conference on Machine Learning, ICML 2012.* Vol. 2. 2012, pp. 1567–1574. ISBN: 9781450312851.

[LBK17] M. Y. Liu, T. Breuel, and J. Kautz. "Unsupervised image-to-image translation networks". *Advances in Neural Information Processing Systems.* 2017. arXiv: `1703.00848`.

[LL17] S. M. Lundberg and S. I. Lee. "A unified approach to interpreting model predictions". *Advances in Neural Information Processing Systems.* Vol. 2017-Decem. May 2017, pp. 4766–4775. arXiv: `1705.07874`. URL: `http://arxiv.org/abs/1705.07874`.

[MR19] R. Mariappan and V. Rajan. Deep collective matrix factorization for augmented multi-view learning. *Machine Learning* **108**.8 (2019), 1395–1420. ISSN: 1573-0565. DOI: `10.1007/s10994-019-05801-6`. URL: `https://doi.org/10.1007/s10994-019-05801-6`.

[ND14] N. Natarajan and I. S. Dhillon. Inductive matrix completion for predicting gene-disease associations. *Bioinformatics* **30**.12 (2014), 60–68. ISSN: 14602059. DOI: `10.1093/bioinformatics/btu269`.

[Piñ+15] J. Piñero et al. DisGeNET: A discovery platform for the dynamical exploration of human diseases and their genes. *Database* **2015** (2015). ISSN: 17580463. DOI: `10.1093/database/bav028`.

[RSG16] M. T. Ribeiro, S. Singh, and C. Guestrin. ""Why should i trust you?" Explaining the predictions of any classifier". *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.* Vol. 13-17-Augu. Association for Computing Machinery, Aug. 2016, pp. 1135–1144. ISBN: 9781450342322. DOI: `10.1145/2939672.2939778`. arXiv: `1602.04938`.

[SG08] A. P. Singh and G. J. Gordon. Relational learning via collective matrix factorization. *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (2008), 650–658. DOI: `10.1145/1401890.1401969`.

[SSA13] K. Swersky, J. Snoek, and R. P. Adams. "Multi-task Bayesian optimization". *Advances in Neural Information Processing Systems.* 2013, pp. 1–9.

[Sub+05] A. Subramanian et al. Gene set enrichment analysis: A knowledge-based approach for interpreting genome-wide expression profiles. *Proceedings of the National Academy of Sciences of the United States of America* (2005). ISSN: 00278424. DOI: `10.1073/pnas.0506580102`.

[Tan+17] A. Tank et al. An Interpretable and Sparse Neural Network Model for Nonlinear Granger Causality Discovery. Nips (2017). arXiv: 1711.08160. URL: http://arxiv.org/abs/1711.08160.

[TG19] E. Tjoa and C. Guan. *A Survey on Explainable Artificial Intelligence (XAI): Towards Medical XAI*. 2019. arXiv: 1907.07374 [cs.LG].

[Wei+13] J. N. Weinstein et al. *The cancer genome atlas pan-cancer analysis project*. 2013. DOI: 10.1038/ng.2764.

[WFM15] J. Wang, R. Fujimaki, and Y. Motohashi. Trading interpretability for accuracy: Oblique treed sparse additive models. *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* **2015-Augus** (2015), 1245–1254. DOI: 10.1145/2783258.2783407.

[WYZ16] Y. Wang, H. Yao, and S. Zhao. Auto-encoder based dimensionality reduction. *Neurocomputing* **184** (2016). RoLoD: Robust Local Descriptors for Computer Vision 2014, 232–242. ISSN: 0925-2312. DOI: https://doi.org/10.1016/j.neucom.2015.08.104. URL: http://www.sciencedirect.com/science/article/pii/S0925231215017671.

[YL06] M. Yuan and Y. Lin. Model selection and estimation in regression with grouped variables. *Journal of the Royal Statistical Society. Series B: Statistical Methodology* **68**.1 (2006), 49–67. ISSN: 13697412. DOI: 10.1111/j.1467-9868.2005.00532.x.

[ZF14] M. D. Zeiler and R. Fergus. Visualizing and understanding convolutional networks. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* **8689 LNCS**.PART 1 (2014), 818–833. ISSN: 16113349. DOI: 10.1007/978-3-319-10590-1_53. arXiv: 1311.2901.

A Model Hyperparameters

A.1 dCMF

Table A.1: Hyperparameters of the dCMF models. Parameters learned through Bayesian optimization left out. PT means pretraining and BO is Bayesian optimization.

Dataset	Activation	Last layer	Epochs	PT threshold	PT epochs	# BO steps	initial design size
Simulated data	tanh	False	6000	0.1	2	5	5
Movielens100k	tanh	False	2000	0.1	2	20	5
Omic data	sigmoid	True	2000	0.1	2	20	5

Table A.2: Hyperparameters ranges for Bayesian optimization.

Hyperparameter	Type	Domain
leanring rate	Continuous	[1e-6, 1e-4]
convergation threshold	Continuous	[1e-5, 1e-4]
weigth decay	Continuous	[0.05, 0.5]
kf	Continuous	[0.1, 0.5]
k	Discrete	{10, 100, 200}
# chunks	Discrete	{1, 2}

A.2 CMF and gCMF

Table A.3: Hyperparameters of the CMF and gCMF models. Anything not mentioned in the table is set to the standard value of the `R` `CMF` package.

Dataset	Likelihood	Iterations	k	Gradient max
Simulated data	Gaussian	100	100	1000
Movielens100k	Gaussian	100	200	∞
Omic data	Gaussian	200	80	∞

A.3 MM-PCA

Table A.4: Hyperparameters of the MM-PCA models. Anything not mentioned in the table is set to the standard value of the `R` `mmpca` package.

Dataset	k
Simulated data	5
Movielens100k	5

A.4 DTN

Table A.5: Hyperparameters of the data translation networks.

Dataset	k	# Layers in autoencoder	Patience	Learning Rate	α
TCGA	200	5	30	1×10^{-3}	0.3
TCGA sparse	80	3	30	1×10^{-3}	0.3
Simulated	5	3	50	5×10^{-4}	0.3

B LIME-dCMF Movielens Test Subjects

Person A

- Male
- Age 18-24
- Programmer
- 78390
- Toy Story (1995)
- Taxi Driver (1976)
- Star Wars (1977)
- Pulp Fiction (1994)
- Shawshank Redemption, The (1994)
- Lion King, The (1994)
- Jurassic Park (1993)
- Blade Runner (1982)
- Home Alone (1990)
- Fargo (1996)
- Godfather, The (1972)
- 2001: A Space Odyssey (1968)
- Empire Strikes Back, The (1980)
- Raiders of the Lost Ark (1981)
- Alien (1979)
- Full Metal Jacket (1987)
- Indiana Jones and the Last Crusade (1989)
- Mars Attacks! (1996)
- Hercules (1997)

Person B

- Male
- Age 30-38
- Engineer
- 20784
- Pulp Fiction (1994)
- Shawshank Redemption, The (1994)
- Aladdin (1992)
- Annie Hall (1977)
- Hunchback of Notre Dame, The (1996)
- Koyaanisqatsi (1983)

Person C

- Male
- Age 30-38
- Scientist
- 09645
- Toy Story (1995)
- GoldenEye (1995)
- Twelve Monkeys (1995)
- Seven (Se7en) (1995)
- From Dusk Till Dawn (1996)
- Braveheart (1995)
- Star Wars (1977)
- Pulp Fiction (1994)
- Forrest Gump (1994)
- Jurassic Park (1993)
- Blade Runner (1982)
- Terminator 2: Judgment Day (1991)
- Fargo (1996)
- 2001: A Space Odyssey (1968)
- Die Hard (1988)
- Willy Wonka and the Chocolate Factory (1971)
- Monty Python's Life of Brian (1979)
- Reservoir Dogs (1992)
- Monty Python and the Holy Grail (1974)
- Empire Strikes Back, The (1980)
- Return of the Jedi (1983)
- Alien (1979)
- Full Metal Jacket (1987)
- Terminator, The (1984)

- Nikita (La Femme Nikita) (1990)
- Shining, The (1980)
- Fifth Element, The (1997)
- Lost World: Jurassic Park, The (1997)
- Men in Black (1997)
- Donnie Brasco (1997)
- Alien: Resurrection (1997)
- Jackie Brown (1997)
- Mission: Impossible (1996)
- Die Hard: With a Vengeance (1995)
- Alien 3 (1992)
- Jumanji (1995)
- Big Lebowski, The (1998)
- Underworld (1997)
- Big Bang Theory, The (1994)

C Patient features

- Patient cancer cohort
- Gender
- Age at diagnosis
- Event
- Time to event
- RFS
- RFS_ind
- Sample type id
- CNA K8 class
- DNA-Methyl cluster
- RPPA class
- RNAseq cluster
- miRNA cluster
- Mutation cluster
- Gene set enrichment cluster