Gene Regulatory Network Inference by Reducing Dimensions for Single-cell transcriptome profiles.

by
Ankur Gajendra Meshram

Under the supervision of
Dr. Vibhor Kumar

Submitted in partial fulfillment of the requirements for the degree of Master of Technology, in Computational Biology

Department of Computational Biology Indraprastha Institute of Information Technology - Delhi
May, 2025

Certificate

This is to certify that the thesis titled *"Gene Regulatory Network Inference by Reducing Dimensions for Single-cell transcriptome profiles"* being submitted by **Ankur Gajendra Meshram** to the Indraprastha Institute of Information Technology, Delhi, for the award of the Master of Technology in Department of Computational Biology, is an original research work carried out by her under my supervision. In my opinion, the thesis has reached the standards fulfilling the requirements of the regulations relating to the degree.

The results contained in this thesis have not been submitted in part or full to any other university or institute for the award of any degree/diploma.

May,2025

Dr. Vibhor Kumar

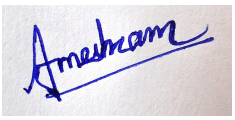
Department of Computational
Biology, Indraprastha Institute of Information
Technology Delhi
New Delhi 110020

Acknowledgements

I extend my heartfelt appreciation to Dr. Vibhor Kumar for his invaluable guidance and support during my M.Tech thesis. His expertise, encouragement, and constructive feedback have been crucial in shaping my work, inspiring me to strive for excellence and overcome challenges. Dr. Vibhor Kumar's commitment to creating a supportive research atmosphere and his confidence in my capabilities have been incredibly inspiring. It has been a privilege to work under his guidance, and I am grateful for his significant time and contributions to my academic journey.

Furthermore, I extend my thanks to my friends for their continuous encouragement and support. Additionally, I acknowledge the contributions of the broader research community, whose insights shared through publications, conferences, and online platforms have been indispensable in shaping my research.

Lastly, I am profoundly thankful to my family for their unwavering love and support, which has been a constant source of strength throughout.



Ankur Gajendra Meshram
MT23241

Abstract

Understanding the complex regulatory relationships between genes is central to conducting cellular functionality and disease. However, the inference of Gene Regulatory Networks (GRNs) from sparse single-cell RNA sequencing (scRNA-seq) data is difficult from a computational perspective, especially when nonlinear dependence is a common feature. To address this, we developed NIRD: Network Inference by Reduced Dimension, a new Linux command line tool for large-scale GRN inference. NIRD relies on applying matrix-factoring strategies to obtain a more manageable representation of scRNA-seq profiles that still preserves critical biological signals. Each matrix-factoring strategy was utilized to target linear versus complex, non-linear relationships that standard linear methods may not capture, and performance for each matrix-factoring strategy was evaluated for GRN inference using the Area Under the Curve (AUC) score. NIRD outputs a complete gene-by-gene matrix representative of inferred regulatory interactions for the cellular population. We expect it will provide an insightful and efficient process of ultimately extracting more informative knowledge from complex regulatory interactions within single-cell transcriptome data.

Contents

List of Figures

List of Algorithms

1. Singular Value Decomposition (SVD)
2. Non-negative Matrix Factorisation (NMF)
3. Bayesian Decomposition (BD)
4. Binary Matrix Factorisation (BMF)
5. Iterated Conditional Modes (ICM)
6. Fisher Non-negative Matrix Factorisation for learning Local features (LFNMF)
7. Alternating Non-negative Least Squares Matrix Factorisation (LSNMF)
8. Probabilistic Non-negative Matrix Factorisation (PMF)
9. Kullback-Leibler Divergence (KLD-NMF)
10. Euclidean Non-negative Matrix Factorisation (ENMF)
11. Sparse Non-negative Matrix Factorisation (SparseNMF or SNMF)
12. Penalised Matrix Factorisation for Constrained Clustering (PMFCC)
13. Separable Non-negative Matrix Factorisation (SepNMF)
14. Principal Component Analysis (PCA)
15. Kernel Principal Component Analysis (KPCA)
16. Edge Overlapping

Chapter 1

Introduction

The use of single-cell RNA sequencing (scRNA-seq) technologies has fundamentally changed our capacity to characterize cellular heterogeneity and better resolve complex biological processes at this scale [1]. scRNA-seq data allow us to survey the transcriptome of individual cells, enabling us to (i) characterize different cell populations, (ii) study distinct cellular states that arise as cells change their expression patterns and input samples change over time, and (iii) predict gene regulatory networks (GRNs) that govern these cellular states and outputs of different tissue and biological systems [2][3]. GRNs - sets of interactions between a set of genes, transcription factors, and other regulators (e.g. metabolites) - are central to understanding the mechanisms that underpin an organism's development and differentiation, generation of diseases, and ultimately, the responses to therapeutic inputs [4].

Yet, inferring GRNs accurately and comprehensively from single cell RNA-seq presents several large challenges. First, scRNA-seq data are high-dimensional, with expression of thousands of genes measured for each individual cell. The high-degree of dimensionality will require employing effective dimensionality reduction to amplify relevant biological signals and down-weight irrelevant features: a feat that becomes difficult due in part to the "curse of dimensionality" [5]. Second, when working with scRNA-seq data, researchers often encounter sparse counts as a large number of genes are measured and in many cases, there is a substantial number of gene counts equal to zero due to (i) technical limitations (e.g. dropout, etc.) and (ii) stochasticity in gene expression [6]. Consequently, zero counts can mask biologically relevant relationships and make it difficult to employ network inference methods designed to analyze bulk RNA-seq count data.

In addition, gene regulatory interactions are context-dependent and may not be linear [7]. The use of linear approaches as a basis for GRN inference, including correlation-based methods and linear regression models, may oversimplify and ultimately yield incorrect regulatory architecture [8]. Because of this, it is becoming increasingly necessary to develop advanced computational methods that accommodate the high dimensionality and sparsity of scRNA-seq data and can also accommodate non-linear dependencies between genes.

To overcome these obstacles we have created **NIRD: Network Inference from Reduced Dimensions**, a new Linux command line algorithm that is able to conduct large-scale GRN inference from scRNA-seq data. NIRD pursues a systematic approach to GRN inference by

combining dimensionality reduction through matrix factorization methods in a computationally efficient manner with a strong approach to evaluating the performance of the inferred networks. NIRD is able to explore a diverse range of matrix factorization methods, including Principal Component Analysis (PCA) [42], Singular Value Decomposition (SVD) [9], Non-negative Matrix Factorization (NMF) [10], Iterated Conditional Modes NMF (ICM), Bayesian Decomposition (BD), Binary Matrix Factorisation (BMF), Fisher NMF, Least Squares NMF (LSNMF), Kullback Leibler Divergence NMF (KLD-NMF), Euclidean NMF (ENMF), Probabilistic NMF (PMF), Sparse NMF (SNMF), Penalized Matrix Factorisation for Constrained Clustering (PMFCC), Separable NMF (SepNMF) and Sparse NMF (SNMF).

The efficacy of both dimensionality reduction methods in producing accurate GRN inference is quantitatively determined by computing the area under the curve (AUC) score in relation to a gold standard or through defined evaluation metrics [e.g., DREAM challenges]. This comparison serves to establish the relative efficacy of dimensionality reduction approaches for the biological context or dataset under consideration. Wondrously, NIRD outputs the inferred GRN as a complete gene-by-gene matrix, representing strength or likelihood of regulatory interactions in the corresponding entries between the gene pairs.

NIRD represents a powerful and efficient command-line tool for researchers looking to explore the complex regulatory landscape of large-scale single-cell transcriptome data. NIRD increases researchers' ability to explore and reconstruct cellular identity and function by considering data sparsity and uncovering non-linear dependencies using matrix factorization.

Motivation :-

With the availability of large-scale single-cell RNA sequencing (scRNA-seq) datasets growing exponentially, there has never been a greater opportunity to explore cellular variation and understand gene regulatory networks (GRNs). Unfortunately, the high dimensionality and sparsity of scRNA-seq data, combined with the non-linear nature of gene regulation, creates a significant obstacle for accurately inferring GRNs. Existing approaches often do not account for all different types of regulatory relationships because they struggle with the high dimensionality and sparsity of scRNA-seq data. Because of these limitations, we decided to develop NIRD: Network Inference by Reduced Dimension. Our goal with NIRD was to develop a command-line tool that is reliable and efficient to use; and take advantage of matrix factoring methods effectively using only a subset of the original features to preserve non-linear dependencies to achieve more accurate and reliable GRN inference from large-scale scRNA-seq datasets. With NIRD we aimed to give scientists a straightforward and scalable platform to zoom in on their mind-bending desires, by helping them uncover the regulatory characteristics of cellular behaviors.

Related Work :-

The approach of inferring gene regulatory networks (GRNs) through various types of high-throughput data, including bulk and single-cell RNA sequencing data, has been a focus of research for more than 20 years [11]. Methods used for GRN inference from bulk RNA-seq data typically relied on correlation [12], mutual information [13], or linear models (those using expectations of the regulatory relationship) by ARACNE [14] or CLR [15]. These methods were intended to model the statistical dependencies between the gene expression profiles to infer regulatory connections.

With the introduction of scRNA-seq data, there are new challenges and opportunities for inferring GRNs. The challenges associated with inferring GRNs from scRNA-seq data are based on the very high dimension, sparsity (due to dropout), and cellular heterogeneity. As a result scRNA-seq GRN inference methods must be tailored for these data. While these challenges are legitimate, various methods have been introduced to combat these challenges. Some scRNA-seq GRN inference methods seek to adapt existing models of network inference (e.g. the previously mentioned statistical dependency based methods) but adapted to the single-cell setting (it often incorporated imputation to account for the sparsity) [16]. Others utilize the properties of the scRNA-seq data itself (e.g., modeling pseudo-time trajectories and developmental/temporal processes) to infer dynamic GRNs from scRNA-seq data [17][18].

A great deal of research has focused on the use of dimensionality reduction to improve upon GRN inference from scRNA-seq data. Various methods, such as PCA [42] and NMF [10], have been investigated to reduce the dimensionality of the gene expression matrix prior to utilizing network inference methods. The reasoning for this involves limiting the number of individual measurements to reduce noise while highlighting important biological signals by selecting the principal components or latent factors that capture the greatest sources of variation in the gene expression matrix. A better taxonomic GRN can theoretically be generated by reducing unnecessary variation or noise within the data [19].

In addition to considering the limitations of linear models to represent complex regulatory relationships, researchers have begun to investigate various non-linear methods for GRN inference from scRNA-seq data. These have included non-linear methods based on machine learning techniques, such as random forests [20], neural networks [21] and Gaussian processes [22], all of which potentially represent more complex relationships between genes than linear approaches but often come with greater computational costs as well as the need for larger datasets.

How NIRD Differs and Contributes :-

NIRD is different from other approaches because:

1) It is the **systematic evaluation of various matrix factorization methods**; previously documented evaluations have limited evaluation and/or only explored how matrix factorization methods could improve non-linear GRN inference from sparse scRNA-seq data.
2) It uses several different methods with several different factors to explicitly **capture more than linear dependencies** (in many ways, non-linear GRNs can capture much more complex regulatory patterns, and it is hypothesized that the dimensionality reduction of a matrix factorization will improve the non-linear inference).
3) It is provided in **command-line form**, so it can be accessed by researchers on a large scale, and hopefully, it is integrated into existing bioinformatics pipelines used to manage large-scale datasets.
4) It provides a **readily interpretable gene-by-gene interaction matrix** intended for use in downstream network analyses.

Our variation and contributions serve the community, by providing a new, evaluated approach for GRN inference from generating scRNA-seq data that directly addresses dimensionality, sparsity, and non-linearity challenges, and a pipeline evaluation of matrix factorization techniques.

Chapter 2

Datasets Used

1) DREAM5 Challenge Data (link):

We tested our NIRD tool on benchmark datasets provided by the DREAM5 (Dialogue for Reverse Engineering Assessments and Methods) Network Inference Challenge [23]. DREAM5 was a way to evaluate the accuracy and robustness of network inference methods on in silico and real in vivo biological datasets; we focused on the four in silico gene expression datasets, which represent transcriptional regulatory networks for different organisms:

A. **Network 1 (in silico)**: is a simulated gene regulatory network. It has expression data for 1643 genes and 195 transcription factors (TFs) across 805 microarray chips. Since we know what the regulatory interactions are and the underlying network structure, we can directly assess inference accuracy.
B. **Network 2 (S. aureus)**: is based on Staphylococcus aureus and has expression data for 2810 genes and 99 TFs across 160 microarray chips. While the data was based on a real organism, the 'gold standard' also came from the underlying regulatory network that was computed and simulated for this challenge.
C. **Network 3 (E. coli)**: is based on a simulated gene regulatory network of Escherichia coli. This is the largest network of the four in silico networks with expression data from 4511 genes, 334 TFs across 805 microarray chips.
D. **Network 4 (S. cerevisiae)**: is a simulated network for Saccharomyces cerevisiae (yeast) that contains gene expression data for 5950 genes and 533 transcription factors (TFs) taken from 536 microarray chips. This network has the most genes among the four datasets used in the DREAM5 challenge.

In addition to gene expression data, the DREAM5 challenge dataset consisted of some extra descriptors for each microarray experiment including, for example, temporal information (for those experiments that were time series) or gene deletion information (for those knockout experiments). The participating teams of DREAM5 were tasked with predicting the complete (genome-scale) transcriptional regulatory networks for each of these four compendia. With the availability of the true underlying networks for these in silico datasets these datasets can be used for benchmarking and comparing the performance of GRN inference methods including our

2) mESC (link):

In order to evaluate the performance of our NIRD algorithm on a real biological dataset, we used a publicly available gene expression mESC (mouse embryonic stem cell) dataset. The source of the gene expression dataset can be found in the benchmark_gene_regulation_mesc GitHub repository. The mESC dataset provides a good resource for evaluating gene regulatory network inference methods in an established biological system.

3) Pancreas (link):

We also relied on a publicly available single-cell RNA sequencing (scRNA-seq) dataset of human pancreas from the Gene Expression Omnibus (GEO) (accession number: GSE81547; [25]). The study provides a comprehensive transcriptional analysis of 2544 human pancreatic cells collected from donors over a span of six decades of life, including healthy and type 2 diabetic individuals.

The aim of the study was to understand mechanisms of aging and somatic mutations that affect the transcriptome of the human pancreas, especially the islet cells. They performed a single cell transcriptome analysis via high-throughput sequencing to characterize the various human pancreas cell populations and identify age-related and disease-related transcriptional changes.

4) Knee Cartilage (link):

To evaluate the suitability of our NIRD algorithm for human clinical data, we used a publicly available single-cell RNA sequencing (scRNA-seq) dataset of human knee cartilage, which can be found at the Gene Expression Omnibus (GEO) with the accession number GSE255460 [24]. This study characterized the transcriptional landscape of human knee cartilage from patients with osteoarthritis (OA) and non-OA controls to identify cellular mechanisms leading to cartilage degeneration.

The study performed 19 cartilage samples from eight OA donors and three non-OA control donors using Illumina NovaSeq 6000 high throughput sequencing. The raw sequencing data includes transcriptomes from tens of thousands of individual cells, thus providing spatial measure of cellular heterogeneity in knee cartilage tissue.

When analyzing the data using the NIRD tool, we limited our scope to two individual samples in this dataset - GSM8072834 and GSM8072837. After performing quality control, an individual sample included raw sequencing data, organized into barcodes, features (genes), and a matrix of counts. To work with this data to run through NIRD, we preprocessed the data as follows:

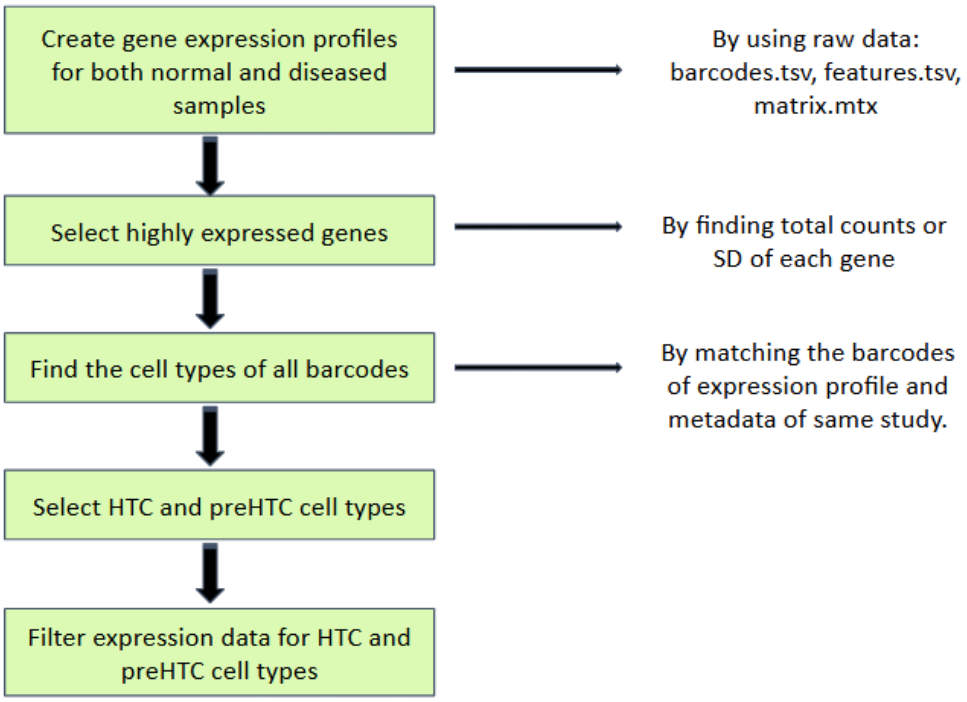


Figure 2.1: Workflow for preprocessing of raw human knee cartilage data

5) hESC (link):

In exploring the dynamic evolution of gene regulatory networks involved in cellular differentiation, we chose to utilize a published single-cell RNA sequencing (scRNA-seq) dataset (GSE75748) that is publicly available through Gene Expression Omnibus (GEO) [26]. This paper contains both a snapshot and temporal characterization of progenitor cells from human embryonic stem cells (hESC) as they undergo differentiation into definitive endoderm progenitors.

The dataset includes 1018 single-cell transcriptome profiles from progenitor snapshot populations, and an additional 758 cells that were captured throughout the time course of differentiation. Using scRNA-seq, the researchers examined variability in the heterogeneity of these specific cell populations and reconstructed the differentiation trajectory with a novel statistical pipeline called Wave-Crest. They also functionally validated candidate regulatory elements using CRISPR/Cas9 gene editing.

Chapter 3

Methods implemented in NIRD

3.1 Matrix Factorisation (MF) :-

Matrix factorization (MF) is a dimensionality reduction method that attempts to factor a high-dimensional data matrix into the product of two (or more) lower-dimensional matrices. The goal is to learn the underlying latent factors or features that govern the relationships within the original data. [27, 28]

In mathematical terms, let us say we have an original data matrix R of dimensions m×n (m could be the number of cells and n could be the number of genes in single-cell data), the matrix factorization method tries to find the two matrices, U of dimensions $m \times k$ and V of dimensions $k \times n$ (where k is a chosen lower dimension $k < m \; and \; k < n$) so that their product approximates the original data matrix R:

$$R \approx U * V$$

In this model:

- Each row in U represents a data point (e.g., a cell) in the lower dimensional space while each column in U represents a latent factor.
- Each column in V represents a feature (e.g., a gene) in the lower dimensional space while each row represents a latent factor.
- The value at the $i^{th} \; row \; and \; j^{th} \; column$ in the reconstructed matrix UV approximates the original value R_{ij}.

From the mathematical statements above, the objective of the factorization process is to find matrices U and V that minimize the distance (error) between R and UV (in this case). The minimization of the distance (error) is typically done through some optimization algorithms or methods including gradient descent or alternating least squares (ALS).

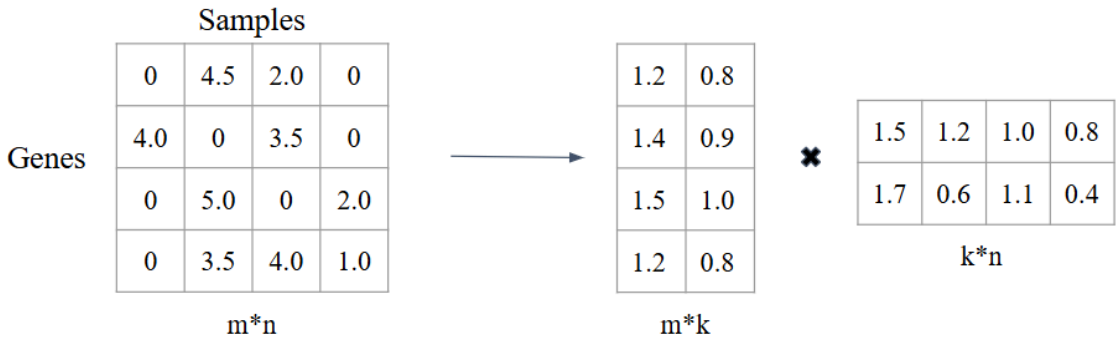


Figure 3.1: Factorisation of initial matrix into two lower dimensional matrices

Why matrix factorisation ?

Conventional dimensionality reduction methods are popular methods in a variety of contexts in single-cell transcriptomic analysis, such as PCA and t-SNE [47]. However, these methods can restrict biological interpretations due to the nature of the methods themselves. Although PCA is categorized as a linear method, it is primarily focused on capturing directions of variability, while being computationally efficient, and deterministic [48]. PCA's linearity can lead to distortion in the representation of the underlying structure of complex, non-linear data; which is often seen in biological data [49, 50]. Examples where PCA performance deteriorates is in cases when local neighborhood information is critical to identifying subtle changes in gene expression (e.g., domains of closely related cell types or states).

Nonetheless, it is important to note that t-SNE was designed to retain information about the local neighborhood structures, therefore, it is possible to visualize the clustering of the data using t-SNE [51]. t-SNE methods are useful when considering a comparison of local matters in a more extensive data with the detriment of loss of a global structure and potentially incorrect representation of the actual cluster sizes and densities among other things introduced from the user defined hyperparameters [52]. Therefore, interpretation is limited with t-SNE in comparison to PCA, notably both t-SNE and PCA provided further hurdles in reproducibility if the data being compared was new or even the same data but separate t-SNE plots were made with the same accuracy criteria [53].

Matrix factorization methods present an interesting alternative since they provide solutions to many of the considerations discussed above, particularly in regards to inferring gene regulatory networks [54]. Matrix factorization methods, unlike PCA and t-SNE, can also represent both linear and nonlinear relationships in the data (depending on the factorization method used, e.g.,

NMF for parts-based representation or more complex kernel methods for non-linearity) and can tackle high-dimensional and high-sparsity gene expression matrices which, along with clustering, is one of the biggest hurdles in single-cell transcriptomics [55,56]. When we decompose gene expression matrices we can interpret these as a collection of latent features or components which arguably represent biological programs or regulatory mechanisms that govern observed variability from cell-to-cell [57]. Matrix factorization methods are, therefore, more than a tool for dimensionality reduction, but a tool for biological discovery [58].

One of the key advantages of matrix factorization is interpretability. By reviewing the genes with high weights for each latent feature, a researcher can discover putative marker genes and understand the molecular basis of cellular identity or processes [57]. These latent features can often be linked back to co-regulated gene modules, regulatory pathways, or distinguishable cellular states, which are useful when reconstructing gene regulatory networks [58]. Because of its benefits, matrix factorization provides a powerful foundation for making biologically relevant inferences about networks, and we plan to utilize it as part of the NIRD framework in the process of GRN reconstruction.

Different MF algorithms implemented in NIRD :-

We implemented 13 different matrix factorisation algorithms in NIRD software which are enlisted as follows -

1) Singular Value Decomposition (SVD)
2) Non-negative Matrix Factorisation (NMF)
3) Bayesian Decomposition (BD)
4) Binary Matrix Factorisation (BMF)
5) Iterated Conditional Modes (ICM)
6) Fisher Non-negative Matrix Factorisation for Learning Local Features (LFNMF)
7) Least Squares NMF (LSNMF)
8) Probabilistic Non-negative Matrix Factorisation (PMF)
9) Kullback-Leibler Divergence (KLD-NMF)
10) Euclidean Non-negative Matrix Factorisation (ENMF)
11) Sparse NMF (SNMF)
12) Separable NMF (SepNMF)
13) Penalized Matrix Factorisation for Constrained Clustering (PMFCC)

Note: For in-depth theory of NMF and its variants, look at Appendix.

3.1.1 Singular Value Decomposition (SVD) :

Singular Value Decomposition (SVD) is an important matrix factorization technique that decomposes the matrix X (for example, a gene expression matrix generated via scRNA-seq) into three matrices U, Σ, and V^T, so we get: $X = U\Sigma V^T$. Here, U and V are orthogonal matrices that represent the singular vectors and are composed of columns of singular vectors, while Σ is a diagonal matrix containing singular values. The singular values tell us how much variance is explained in the singular vector they are associated with and they represent orthogonal directions of maximum variance in the dataset. SVD does not restrict the components to be non-negative as some factorization methods do, which can lead to less biologically interpretable factors at times.[9]

Algorithm 1: Singular Value Decomposition (SVD)

Begin:

1. Initialize parameters:
   - set **n_components** = k
   - get gene names, TF names
   - set regulators = **Intersection**(gene names, TF names) assuming all genes if TFs not provided
   - create the Network which is 0 matrix of (len(regulators) × len(regulators))

2. Get the expression matrix:
   - **Get** the expression data E for regulatory genes only (shape = samples × regulators)

3. Preprocess expression data to prepare it; (optional scaling):
   - **scaled_E** = E

4. Factorization (Truncated SVD):
   - Apply **TruncatedSVD** to fit to scaled_E
   - From the above you get **low_rank_E** = transformed expression (shape = samples × k)
   - and get **mixture_matrix** = |SVD components| (shape = k × regulators)

5. Estimating Feature Importance:
   For each gene (i) in regulators:
   a. Set target: **y** = scaled_E[:, i]
   b. Set inputs: **X** = low_rank_E
   c. Fit **RandomForestRegressor** on y from X
   d. Store out feature importance **fi** (shape = k)
   e. Update **Feature_Importance_Matrix**[i, :] = fi

6: Reverse Factorization:

- compute **Network** = Feature_Importance_Matrix × mixture_matrix
- **Normalize** Network with sum of mixture_matrix as denominator + ε
- (3 or 4): set self.network = Network

7. Remove self loops from Network:
   - make all diagonal elements of Network=0

8. Return:
   - the fitted Network matrix
   - the list of regulators

End.

3.1.2 Non-negative Matrix Factorisation (NMF) :

Non-negative Matrix Factorization (NMF) is an effective method for dimensionality reduction, which factorizes a non-negative data matrix X (for example, a gene expression matrix where all counts of non-negative), into two non-negative matrices, W and H subjects to X≈WH. The first non-negative matrix, W, represents the basis vectors (or 'metagenes' in an example from biology), and the second non-negative matrix, H, contains the coefficients (or 'cell-specific activities' of these metagenes). Non-negativity is crucial, because it means all parts are additive and therefore interpretable, which fits well with most biological contexts, where gene expression levels must be additive, and that biological modules combine to define overall states of the cell.[10]

Algorithm 2: Non-negative Matrix Factorisation (NMF)

Begin:
1. Initialize:
   - Set **n_components** = k
   - Gather gene names and find regulators = **intersection**(gene names, TFs) or pick all genes to use.
   - Create zero matrix Network of size len(regulators) × len(regulators)

2. Preprocess the expression data:
   - Apply **MinMaxScaler** to expression matrix E to transform the entire expression matrix to be non-negative
   - Store **scaled_E** = the scaled version of E

3. Factorization (**NMF**):
   - **Fit** NMF with k components to the scaled_E
   - Get **low_rank_E** = transformed matrix (samples × k), where there are k components
   - Get **mixture_matrix** = |components| matrix (k × regulators) obtained during NMF fit

4. Feature Importance Estimation:
   For each gene i in regulators:
      a. Specify target **y** = scaled_E[:, i]
      b. Specify input **X** = low_rank_E
      c. Fit **RandomForestRegressor** as if predicting y from X
      d. Extract feature importances **fi** (length k)
      e. Store fi in **Feature_Importance_Matrix**[i, :] = fi

5. Reverse Factorization (Project feature importance back):
   - Compute **Network** = Feature_Importance_Matrix × mixture_matrix
   - **Normalize** Network by dividing with sum of mixture_matrix + ε
   - Store self.network = Network

6. Remove self-loops, if needed:
   - Set diagonal of Network = 0

7. Return:
   - Final Network matrix
   - List of regulators

End.

3.1.3 Bayesian Decomposition (BD) :

Bayesian Decomposition (BD) methods for matrix factorization proceed in a way that incorporates probabilistic models, inferring latent factors and determining uncertainties of latent factors, rather than point estimates of factors. The BD approach doesn't just find the best fit, but rather, has a probabilistic model of how data are generated according to some given prior distribution over the latent factors and their weights. Inference requires the maximum likelihood of the observed data given the prior distribution, which usually infers posterior distributions of latent factors, providing a richer model of factorization, as well as confidence intervals of the inferred components and contributions.[30]

Algorithm 3: Bayesian Decomposition (BD)

Begin:

1. **Initialization**:
   a. inherit from NMF base (**NonNegativeMatrixFactorization_NIMFA**)
   b. Number of components = k
   c. Choose regulators from gene names
   d. initialize feature importance matrix (regulators × k)

2. Preprocessing:
   - set **scaled_exp** = E # no scaling, nor normalisation, nor log-transformation

3. Bayesian Factorization using Nimfa:
   a. run **nimfa.Bd**() with:
      - input matrix: V = scaled_exp,
      - Rank = k
      - optional seed = **'nndsvd'**
   b. run the decomposition: **Bd_model**()
   c. extract:
   - W = **low_rank_exp** = basis matrix (samples × k)
   - H = **mixture_matrix** = coefficient matrix (k × genes)

4. Feature Importance Estimation:
   - **for** each regulator i do:
      a. set target **y** = original expression values of gene i
      b. set input **X** = low_rank_exp (i.e. W)
      c. **fit** Random Forest Regressor on y from X
      d. extract feature importances **fi** (length k)
      e. append: **Feature_Importances**[i, :] = fi

5. Reverse Factorization:
   a. Project feature importances from latent space to original space:
      Network = Feature_Importances × mixture_matrix
   b. Normalize by total importance:
      Network /= sum(Feature_Importances + epsilon)

6. Output:
   - Final Network Matrix
   - Mixture and Basis matrices
   - Feature importance scores

End.

3.1.4 Binary Matrix Factorisation (BMF) :

Binary Matrix Factorization (BMF) is a type of matrix factorization that works with a binary matrix that only contains the values 0's and 1's, which are then factorized into the product of binary (or nearly binary) factors. This is in contrast to matrix factorization techniques that allow for the decomposition of real-valued data. Generally, the underlying assumption behind most applications of BMF is Boolean algebra (OR or AND) rather than linear combinations. Given this structure, BMF is particularly appropriate for datasets that have binary relationships (i.e. presence/absence) rather than meaningful variations within a continuous quantity. [31]

Algorithm 4: Binary Matrix Factorisation (BMF)

Begin:
1. Initialization:
   a. Inherit from NMF base (**NonNegativeMatrixFactorization_NIMFA**)
   b. Set number of components = k
   c. Select regulators from gene names
   d. Initialize feature importance matrix (regulators × k)

2. Preprocessing:
   a. Apply **log-normalization**:
      For each element x in E:
         - If x > 0, keep it
         - Else, set x = -10 (to preserve log space)
         - Apply log(x)
   b. **For** each gene (column i):
      - Compute threshold = 0.25 * (min(temp[:, i]) + max(temp[:, i]))
      - Binarize each value in the gene column:
         value = 1 if >= threshold else 0
   c. Transpose and store binarized matrix as **scaled_exp**

3. Binary Matrix Factorization using Nimfa:
   a. Apply **nimfa.Bmf**() with:
      - Input matrix V = scaled_exp
      - Rank = k
   b. Run the decomposition: **Bmf_model**()
   c. Extract:
      - W = **low_rank_exp** = binary basis matrix (samples × k)
      - H = **mixture_matrix** = coefficient matrix (k × genes)

---

4. Feature Importance Estimation:
   For each regulator i:
      a. Set target **y** = original expression values of gene i
      b. Set input **X** = low_rank_exp (W)
      c. Fit Random Forest Regressor to predict y from X
      d. Extract feature importances fi (length k)
      e. Store in **Feature_Importances**[i, :] = fi

5. Reverse Factorization:
   a. Project feature importances from latent space to original space:
      Network = Feature_Importances × mixture_matrix
   b. **Normalize** by total importance:
      Network /= sum(Feature_Importances) + ε

6. Output:
   - Final network matrix
   - Mixture and basis matrices
   - Feature importance scores

End.

3.1.5 Iterated Conditional Modes (ICM) :

Iterated Conditional Modes (ICM) is an optimization algorithm typically used in a probabilistic modelling framework (e.g. graphical models or factor models) when direct joint optimization is not feasible. In matrix factorization, ICM works by iteratively updating one variable or a small collection of variables at a time, while holding all other variables fixed (i.e. other parameter spaces) until a local posterior probability or likelihood is maximized and continues until a local maximum is reached. It is greedy, computationally cheap, but may get stuck in local optima.[31]

Algorithm 5: Iterated Conditional Modes (ICM)

Begin:
1. Initialization:
   a. Inherit **NonNegativeMatrixFactorization_NIMFA** base class
   b. Set **k** = number of components
   c. Get expression matrix **E**

d. Initialize feature importance matrix (shape: regulators × k)

2. Preprocessing:
   a. Do no scaling or transformation
   b. Store raw expression matrix E as **scaled_exp**

3. Iterated Conditional Modes Matrix Factorization (ICM-NMF):
   a. Use **nimfa.Icm**() with parameters,
      - **V** = scaled_exp
      - **rank** = k
      - **seed** = 'nndsvd'
   b. Matrix factorization
   c. Extract:
      - **W** = low_rank_exp = basis matrix (samples × k)
      - **H** = mixture_matrix = coefficient matrix (k × genes)

4. Feature Importance Estimation:
   For each regulator, i:
      a. **Set** target **y** = expression profile of gene i (from original E)
      b. **Set** predictors **X** = W (low_rank_exp)
      c. **Fit** Random Forest Regressor to predict y from X
      d. Extract feature importances (length = k)
      e. Store **Feature_Importances**[i, :] = importances

5. Reverse Factorization (mapping back into full space):
   a. Multiply:
      true_rank_feature_importance = Feature_Importances × mixture_matrix
   b. **Normalize** (by the sum of all Feature_Importances + ε where ε = small number to avoid divide by zero):
      true_rank_feature_importance /= (sum of all Feature_Importances + ε)

6. Output:
   - Final inferred gene regulatory network
   - Low rank matrices W and H
   - Feature importance matrix

End.

3.1.6 Fisher Non-negative Matrix Factorisation for learning Local features :

Fisher NMF for Learning Local Features (LFNMF) builds on the principles of Non-negative Matrix Factorization by introducing a discriminative term, typically in the form of Fisher's Linear Discriminant Analysis (FLDA) or a graph Laplacian. The central objective of LFNMF is to achieve not only data reconstruction, but also to ensure that the features (or metagenes) that are learned are discriminative features that distinguish classes or clusters in the data. This is especially applicable where the discriminative features are desired for classification or clustering.[32]

Algorithm 6: Fisher Non-negative Matrix Factorisation for learning Local Features (LFNMF)

Begin:

1. Initialization:
   a. Derive from **NonNegativeMatrixFactorization**
   b. Set number components = **k**
   c. Extract expression matrix **E** (n positive)
   d. Initialise Zeros matrix for **feature_importances** [len(regulators) × k]

2. Preprocessing:
   a. **No** normalisation or transformation
   b. Store raw expression matrix as **scaled_exp**
3. Fisher NMF Factorization (LFNMF):
   a. Call **nimfa.Lfnmf**() to run:
      - V = scaled_exp
      - rank = k
      - max_iter = 12
      - alpha = 0.01
   b. Perform matrix factorisation
   c. Extract:
      - **W** = low_rank_exp (samples × k)
      - **H** = mixture_matrix (k × genes)

4. Feature Importances Estimation:
   for each regulator i in input index:
      a. Set **y** = expression profile of gene i (from E)
      b. Set **X** = W (low_rank_exp)
      c. **Fit** Random Forest Regressor to predict y from X

---

d. Extract feature importances (length = k)
e. Store in **feature_importances**[i, :] = importances

5. Reverse Factorization (Project back to Gene Space):
   a. Compute:
   true_rank_feature_importance = feature_importances × H
   b. Normalise:
   true_rank_feature_importance /= (sum of all feature_importances + ε)

6. Output:
   - Final Gene Regulatory Network (true_rank_feature_importance)
   - Low-rank matrices (W, H)
   - Feature importance matrix

End.

3.1.7 Alternating Non-negative Least Squares Matrix Factorisation(LSNMF) :

Least Squares NMF (LSNMF), is the version of Non-negative Matrix Factorization that minimizes and has as its reconstruction error the squared Frobenius norm. For a data matrix X, LSNMF finds non-negative matrices W and H by minimizing $||X - WH||_F^2$. The quantity $||.||_F$ denotes the Frobenius norm. This objective function captures the sum of squared differences of the factorization from the original data and is appropriate for data expected to have Gaussian noise. It is a very widely used and computationally very well characterized version of NMF.[33]

Algorithm 7: Alternating Non-negative Least Squares Matrix Factorisation (LSNMF)

Begin:
1. Initialization:
   a. Inherit from **NonNegativeMatrixFactorization_NIMFA**
   b. Set number of components = **k**
   c. Extract expression matrix **E** from the data
   d. Initialize an empty matrix for the feature_importances. Size: [num_regulators × k]

2. Preprocessing:
   a. If no custom preprocessing is detected, use the inherited default:

- Log normalize expression matrix E
- Set matrix as **scaled_exp**

3. Least Squares NMF Factorization:
   a. Use **nimfa.Lsnmf**() with:
      - **V** = scaled_exp
      - **rank** = k
   b. Decompose the factorization:
      - Factor V into W and H such that V ≈ W * H, assigning:
      - **W**: low_rank_exp (samples × k)
      - **H**: mixture_matrix (k × genes)

4. Feature Importance Estimation (if implemented)
   For each regulator i in the input index:
      a. Set **y** = expression profile of gene i from E
      b. Set **X** = W (low_rank_exp)
      c. **Fit** a Random Forest Regressor to predict y from X
      d. Extract the importance scores (length = k)
      e. Store in **feature_importances**[i, :] = importances

5. Reverse Projection (Back into gene space):
   a. **Compute**:
      true_rank_feature_importance = feature_importances * H
   b. **Normalize**:
      true_rank_feature_importance/=(sum of all feature_importances + ε)

6. Outputs:
      - true_rank_feature_importance
      - W and H
      - feature_importances

End.

3.1.8 Probabilistic Matrix Factorisation (PMF) :

Probabilistic NMF (PMF) is a Bayesian approach to Non-negative Matrix Factorization (NMF). Rather than simply estimating the factor matrices W and H, PMF formulates W, H, and the measured data in terms of probability distributions. Typically a generative model is assumed in which the observed data (e.g., gene expression counts) is generated from a combination of latent factors (W and H), plus noise, which is often modeled with a Gaussian distribution when continuous data is being analysed, or as Poisson for count data, or Negative Binomial when appropriate for the data type. Essentially, we have prior distributions (such as Gamma priors for non-negativity) on W and H; Bayesian inference (by variational inference or Gibbs sampling) is then used to infer W and H in terms of the posterior distributions. This gives us not only the factor matrices, but also some measure of uncertainty for each entry in the factor matrices, which is more information than deterministic NMF provides. [35]

Algorithm 8: Probabilistic Matrix Factorisation (PMF)

Start:
1. Initialization:
   a. Inherit from **NonNegativeMatrixFactorization_NIMFA**
   b. Number of components = **k**
   c. Store input expression matrix as **self.exp**
   d. Initialize feature_importances + other containers

2. Preprocessing:
   a. Set **scaled_exp** = original expression matrix (no scaling, no normalizing)
      scaled_exp ← **E**

3. Factorization using Probabilistic NMF:
   a. To perform NMF factorization create **nimfa.Pmf** factorizer with the following:
      - **V** = scaled_exp
      - **rank** = k
      - **seed** = 'nndsvd' for smart initialization
   b. Run the factorizer:
      - Will factor the matrix V into W and H respectively so that:
        $V \approx W \times H$
      - **W**: sample loadings (samples × k)
      - **H**: basis vectors (k × genes)

4. (Optional) Feature Importance Estimation:

For each gene i in list of regulators:
a. Get target vector **y** = expression of gene i
b. **W** will be the input features (**low_rank_exp**)
c. **Fit** a Random Forest Regressor to predict y from W
d. Store the feature importances of the k components

5. Reverse Projection (Optional):
a. **true_rank_feature_importance** = feature_importances × H
b. **Normalize** the output so the values exist in a comparable range

6. Output:
- matrices W and H
- true_rank_feature_importance, if calculated
- GRN edges, if converted from scores

End.

3.1.9 Kullback-Leibler Divergence NMF (KLD-NMF) :

Kullback-Leibler Divergence NMF (KLD-NMF) is a version of Non-negative Matrix Factorization that takes the Kullback-Leibler divergence (often D KL) as its cost function to measure the difference between the data matrix X and the factorization WH. The KL divergence (or relative entropy) is useful for measuring differences between probability distributions or non-negative count data, and measures the information lost when approximating one distribution for another, making it quite useful for NMF. The goal of KLD-NMF is to minimize $D_{KL}(X||WH)$, where $D_{KL}(P||Q) = \Sigma_{i,j} P_{ij} log\frac{P_{ij}}{Q_{ij}} - P_{ij} + Q_{ij}$ (or other variations, usually the generalized KL divergence or I-divergence for NMF).[34]

Algorithm 9: Kullback-Leibler Divergence Non-negative Matrix Factorisation (KLD-NMF)

Begin:
1. Initialization:
a. Derive from (i.e. subclass) **NonNegativeMatrixFactorization_NIMFA** class
b. Store the expression matrix E as **self.exp**
c. Set the number of components: **self.n_components** = k

2. Pre-processing:

a. Assign the scaled expression matrix:
scaled_exp ← E (no normalization or log-scaling)
b. Write to **self.data**['_scaled_data']

3. Matrix Factorization using NMF:
a. Initialize NMF model from nimfa:
- Method: **nimfa.Nmf**
- Input matrix*: V = scaled_exp
- Rank = k
- Seed = "**random**"
- Update rule = "**divergence**" (i.e. Kullback-Leibler divergence minimization)

b. Run the **factorizer**:
- Find W and H, so that $E \approx W \times H$
- Objective of this optimization, which minimize
$D(E \| W \times H) = \Sigma\ E_{ij} * \log(E_{ij} / (W \times H)_{ij}) - E_{ij} + (W \times H)_{ij}$

4. (optional) Estimation of Feature Importance:
For each regulator gene i:
a. Extract its expression y = E[:, i]
b. Use W as low-dimensional features
c. Fit RandomForestRegressor to predict y from W
d. Store the feature importances

5. (optional) "Reverse Projection":
a. Multiply feature importances with H to obtain gene-level scores:
true_rank_feature_importance = feature_importance × H

6. Outputs:
- W (samples * k) and H (k * genes)
- Feature importance scores (optional)

End.

3.1.10 Euclidean Non-negative Matrix Factorisation (ENMF) :

Euclidean NMF (ENMF) can be thought of as corresponding directly to Least Squares NMF (LSNMF). ENMF minimizes the squared Euclidean distance (Frobenius norm), between the data matrix X, and nonnegative factorization WH. The error function is $||X - WH||_F^2 = \Sigma_{i,j}(X_{ij} - (WH)_{ij})^2$. As such, which assumes that the noise in the data is Gaussian, and that processes in biology combine in a linear manner.[10]

Algorithm 10: Euclidean Non-negative Matrix Factorisation (ENMF)

Begin:
1. Initialization:
   a. Subclass the **NonNegativeMatrixFactorization_NIMFA**
   b. Set expression matrix: self.exp ← **E**
   c. Set number of components: self.n_components ← **k**

2. Preprocessing:
   a. **No** transformations and/or normalizations performed.
   b. Set the following: **self.scaled_exp** ← self.exp
   c. Save this as: self.**data**['_scaled_data']

3. Matrix Factorization (NMF with Euclidean Distance):
   a. Initialize the NMF with nimfa:
      nmf_model ← **nimfa.Nmf**(
                  V = self.scaled_exp, rank = self.n_components, seed = 'random',
                  update = 'euclidean'
)
   b. **Fit** the model: result ← **nmf_model**()

   c. The objective minimized is
      || **E - WH** ||² (Euclidean / Frobenius normal of the reconstruction error).

4. (Optional) Feature Importance Estimation:
   For each regulator gene i in E:
      a. Extract expression **y** ← E[:, i]
      b. Use **W** (low-dimensional features)
      c. Fit **RandomForestRegressor**: y ~ W

---

d. Save the feature importances

5. (Optional) Reverse Projection:
a. Compute: **true_rank_feature_importance** ← feature_importances × H

6. Output:
- W (samples × k) and H (k × genes)
- (Optional) feature importance scores

End.

3.1.11 Sparse Non-negative Matrix Factorisation (SparseNMF or SNMF) :

Sparse NMF (SNMF) uses an additional term to the standard NMF objective function (typically an L1-norm penalty) which promotes sparsity in the factor matrices W and/or H. An example of the objective function structure might be $||X - WH||_F^2 + \alpha||W||_1 + \beta||H||_1$, where α and β would be the regularization parameters. This type of L1-norm penalty promotes a larger quantity of zeros in the factor matrices meaning that each gene module (W columns) will be defined by a smaller, narrower set of genes, and each cell's activity profile will be generated by fewer, more defined modules.[36,37]

Algorithm 11: Sparse Non-negative Matrix Factorisation (SparseNMF or SNMF)

Begin:
1. Initialization:
a. Inherit from **NonNegativeMatrixFactorization_NIMFA**
b. Set expression matrix: self.exp ← **E**
c. Set number of components: self.n_components ← **k**

2. Preprocessing:
a. No transformations or normalization
b. Set: **self.scaled_exp** ← self.exp
c. Save: **self.data**['_scaled_data']

3. Matrix Factorization (Sparse NMF):
a. Initialize Sparse NMF using nimfa:
snmf_model ← **nimfa.Snmf**(

V = self.scaled_exp, rank = self.n_components, seed = 'random'
)
b. **Fit** the model: result ← snmf_model()

4. (Optional) Estimation of Feature Importance:
For each regulator gene i in E:
a. Get expression **y** ← E[:, i]
b. Use **W** (low dimensional features)
c. Fit **RandomForestRegressor**: y ~ W
d. Save feature importances

5. (Optional) Reverse Projection:
a. Compute: **true_rank_feature_importance** ← feature_importances × H

6. Output:
- **W** (samples × k) and **H** (k × genes)
- (Optional) feature importance scores
End.

3.1.12 Penalised Matrix Factorisation for Constrained Clustering (PMFCC) :

Penalized Matrix Factorization for Constrained Clustering (PMFCC) incorporates clustering information directly into the matrix factorization objective function as penalties. This means that the factorization is guided by reconstructing the original data, and also assisted by incorporating prior knowledge of the types of cluster structures we want or know the group labels of the samples (cells). For example, if we know via expert judgment (or previous clustering) that certain cells belong to a specific cell type, we could include a penalty that ensures these cells cluster together in the latent space defined by the factorization.[38]

Algorithm 12: Penalised Matrix Factorisation for Constrained Clustering (PMFCC)

Start:
1. Initialization:
a. Subclass from **NonNegativeMatrixFactorization_NIMFA**
b. Set expression matrix: self.exp ← **E**
c. Set number of components: self.n_components ← **k**

2. Preprocessing:
  a. No transformations or normalizations.
  b. Set: **self.scaled_exp** ← self.exp
  c. Save as: **self.data**['_scaled_data']

3. Matrix Factorization (PMFCC - Penalized Matrix Factorization for Constrained Clustering):
  a. Initialize the PMFCC model with nimfa:
    pmfcc_model ← **nimfa.Pmfcc**(
        V = self.scaled_exp, rank = self.n_components
)
  b. **Fit** the model: result ← pmfcc_model()

4. (Optional) Feature Importance Estimation:
  For each regulator gene i in E:
    a. Extract expression **y** ← E[:, i]
    b. Use **W** (low-dimensional features)
    c. Fit **RandomForestRegressor**: y ~ W
    d. Save feature importances

5. (Optional) Reverse Projection:
   a. Compute: **true_rank_feature_importance** ← feature_importances x H

6. Output:
  - **W** (samples x k) and **H** (k x genes)
  - (Optional) feature importance scores
End.

3.1.13 Separable Nonnegative Matrix Factorisation (SepNMF) :

Penalized Matrix Factorization for Constrained Clustering (PMFCC) incorporates clustering information directly into the matrix factorization objective function as penalties. This means that the factorization is guided by reconstructing the original data, and also assisted by incorporating prior knowledge of the types of cluster structures we want or know the group labels of the samples (cells). For example, if we know via expert judgment (or previous clustering) that certain cells belong to a specific cell type, we could include a penalty that ensures these cells cluster together in the latent space defined by the factorization.[44]

Algorithm 13: Separable Nonnegative Matrix Factorisation (SepNMF)

Begin:
1. Initialization:
   a. Subclass the **NonNegativeMatrixFactorization_NIMFA**
   b. Set expression matrix: self.exp ← **E**
   c. Set number of components: self.n_components ← **k**

2. Preprocessing:
   a. No transformations or normalizations done.
   b. Set: self.scaled_exp ← **self.exp**
   c. Save as: **self.data**['_scaled_data']

3. Matrix Factorization (SepNMF - Separable Non-negative Matrix Factorization):
   a. Initialize SepNMF model using nimfa:
      sepnmf_model ← **nimfa.SepNmf**(
                  V = self.scaled_exp, rank = self.n_components, seed = 'random_vcol'
)
   b. **Fit** the model: result ← sepnmf_model()

4. (Optional) Feature Importance Estimation:
   For each regulator gene i in E:
      a. Extract expression **y** ← E[:, i]
      b. Use **W** (low-dimensional features)
      c. Fit **RandomForestRegressor**: y ~ W
      d. Save feature importances

5. (Optional) Reverse Projection:

---

a. Compute: **true_rank_feature_importance** ← feature_importances × H

6. Output:
   - **W** (samples × k) and **H** (k × genes)
   - (Optional) feature importance scores

End.

3.2 Principal Component Analysis (PCA) :-

PCA is a classical linear dimensionality reduction approach and it is one of the most commonly used methods. In the context of a dataset, for example a gene expression matrix from scRNA-seq, PCA would reformat the dataset into a new coordinate system, where each coordinate is an orthogonal principal component (PC) that represents a certain amount of variance within the data. The first principal component would capture the most variance possible within the data, the second principal component would capture the most variance possible given the first is orthogonal to the first and this continues until all PCs have been calculated. To mathematically capture this, PCA computes the eigenvalue decomposition of the covariance matrix of the data, or alternatively, eigenvalue decomposes the data matrix using a Singular Value Decomposition (SVD). The resultant principal components will be linear combinations of the original features (genes).

For the context of scRNA-seq data and gene regulatory network inference, the PCA decomposition is a critical pre-processing and exploratory tool. ScRNA-seq data is inherently high-dimensional, often random, contains noise, has performance variation from the technological realisation itself, and actually captures biological variances (such as cell cycle or large cellular type differences). PCA can identify the main axes of gene expression variation to minimise the huge dimensionality it has captured. For GRN inference, this suggests that typically the first few principal components tend to identify broader biological processes or biological attributes related to cell-type distinction to filter out noise and also variation that is less informative. The first few principal components can further reduce the high-dimensional gene expression data to a lower-dimensional space (as defined by the top PCs) so that a downstream GRN inference algorithm may more easily deal with the reduced dimension, and run faster and more robustly. All researchers need to do is to be aware that this initial dimension reduction is targeting the largest shifts in gene expression and further help reduce noise. PCA is consistently used in single-cell analysis for visualization (i.e. generating PCA plots to visualize separate populations of cells) and preliminary data investigation [41, 42]. (Check Appendix also)

Algorithm 14: Principal Component Analysis (PCA)

Start:
1. Initialize parameters:
   - **n_components** = k
   - get gene names, **TF** names
   - **regulators** = Intersection(gene names, TF names), assuming all genes if TFs not provided
   - define Network as a 0 matrix of shape (len(regulators) × len(regulators))

2. Get the expression matrix:
   - **get** the expression data **E** for regulatory genes only (shape = samples × regulators)

3. (preprocess) expression data:
   - **scaled_E** = E (identity operation here, no normalization/scaling was done)

4. Factorization (i.e. PCA):
   - **fit** PCA w/ full SVD solver on scaled_E
   - you receive:
     - **low_rank_E** = expression PCA transformed. (shape = samples × k)
     - **mixture_matrix** = |PCA components| (shape = k × regulators)

5. Estimating Feature Importance
   For each gene (i) in regulators:
     a. target: **y** = scaled_E[:, i]
     b. **normalize** target: y = y / std(y)
     c. inputs: **X** = low_rank_E
     d. Fit **RandomForestRegressor** on y from X
     e. Store output feature importances **fi** (shape = k)
     f. Update **Feature_Importance_Matrix**[i, :] = fi

6. Reverse factorization:
   - **Network** = Feature_Importance_Matrix × |mixture_matrix|
   - **Normalize** Network with sum of |mixture_matrix| + ε
   - self.network = Network

7. Return:
   - the fitted Network matrix
   - the list of regulators

End.

3.3 Kernel Principal Component Analysis (KPCA) :-

Kernel Principal Component Analysis (kPCA) is a non-linear form of PCA. PCA is a linear method that only finds linear relationships and allows mathematical transformations based on this linearity. In order to circumvent this linearity limitation, kPCA implicitly maps the data into a higher-dimensional feature space (where linear separation might be possible) using a kernel function and then performs PCA on the transformed data. The kernel trick is the implicit non-linear mapping of the data which does not explicitly require the data points coordinates in the high-dimensional space based on the kernel used and only needs the inner products between the data points in that space. The most common kernels are the radial basis function (RBF) kernel, polynomial kernel, or sigmoid kernel.

For gene regulatory network inference from single-cell RNA-seq data, kPCA can be much stronger than linear PCA because of its ability to extract and describe non-linear relationships in gene expression. For example, biological processes, and especially gene regulation, are often highly non-linear and have very complicated feedback loops, complexities, and thresholds. Gene regulation also often shows a high degree of combinatorial logic. Consider differentiation trajectories; self-organizing processes of self organization may not occupy simple linear trajectories in gene expression state, but rather do so in a curved, branched or moved trajectory. Linear PCA may not be able to represent this data correctly or might lose information that otherwise is relevant to regulatory shifts. kPCA seeks these non-linear structures effectively, when possible, which can then give a more accurate and more sensitive low-dimensional description of cellular states and transitions. This simple kPCA can give researchers a leg up in finding small or context-dependent regulatory interactions which linear methods may miss [43]. There is some literature now that uses kPCA to directly analyze single cell RNA-seq data, and to identify non-linear structures that separate cell types and states, facilitating the reconstruction of trajectories and developmental processes.(Check Appendix also)

Algorithm 15: Kernel Principal Component Analysis (KPCA)

Begin:
1. Initialize parameters:
   - set **n_components** = k
   - get gene names, **TF** names
   - set **regulators** = **Intersection**(gene names, TF names), assuming all genes if TFs not provided
   - create Network as a 0 matrix of shape (len(regulators) × len(regulators))

---

2. Get the expression matrix:
   - **Get** the expression data **E** for regulatory genes only (shape = samples × regulators)

3. Preprocess expression data:
   - **scaled_E** = E (identity operation here, no normalization/scaling applied)

4. Factorization (Kernel PCA):
   - Initialize KernelPCA with 'linear' kernel and k components
   - **Fit** KernelPCA on scaled_E
   - Obtain:
      - **low_rank_E** = transformed expression via KernelPCA (shape = samples × k)
      - **mixture_matrix** = feature contribution matrix (shape = k × regulators), computed via correlation between scaled_E and low_rank_E

5. Estimating Feature Importance:
   For each gene (i) in regulators:
      a. Set target: **y** = scaled_E[:, i]
      b. Set inputs: **X** = low_rank_E
      c. Fit **RandomForestRegressor** on y from X
      d. Store output feature importances **fi** (shape = k)
      e. Update **Feature_Importance_Matrix**[i, :] = fi

6. Reverse Factorization:
   - Initialize **true_rank_feature_importances** as a zero matrix (regulators × regulators)
   - **For** each regulator pair (i, j):
      a. Set **t1** = Feature_Importance_Matrix[i, :]
      b. Set **t2** = mixture_matrix[:, j]
      c. Compute **Network**[i, j] = dot(t1, abs(t2)) / (sum(abs(t2)) + ε)
   - Set self.network = Network

7. Return:
   - the fitted Network matrix
   - the list of regulators

End.

3.4 Evaluation by Edge Overlapping :-

Edge Overlapping is a single quantitative evaluation metric used to assess two networks - usually a predicted/inferred gene regulatory network (GRN) and what is sometimes referred to as a reference (and often represents high-quality or gold) network. Edge Overlapping is the degree to which the two networks' respective sets of edges (gene-gene interactions) overlap, or simply how many of the edges are common across the two networks.[45]

In GRN inference, we use a variety of methods (PCA, NMF, SVD, etc.) that create different edge weight matrices (or edges) for potential interactions between genes. It is important to compare the inferred network against an accepted ground truth since biological data is often characterized by noise, and there are limited ground truths.

Edge Overlapping provides the following:

- ❖ the ability to quantify the two networks agreement by the number of shared interactions
- ❖ be used as a benchmark for comparisons of network inference algorithms
- ❖ produces and can compute cumulative match curves from the results and calculates the AUC (Area Under Curve) as a performance metric.

Algorithm 16: Edge Overlapping

Begin:

1. **Set** parameters:
   - self.top = top (i.e., 10000)
   - self.dx = dx (i.e., 500)
   - self.method = method name
   - self.df_global = empty DataFrame ['xt', 'matches', 'method']

2. Turn networks into edge lists:
   - **g1** = Turn self.network1 (correlation matrix) into edge list with **corr2adjlist** and top=self.top
   - **g2** = Turn self.network2 (correlation matrix) into edge list with corr2adjlist (all edges).

3. Create storage for the results:
   - **df_temp** = empty DataFrame ['xt', 'matches']

4. Sliding window matching:
   - **For** xt from 0 to g2 total edges by dx:

---

a. **e1** = first two columns of g1 as list of tuples (edges in network1)
b. **e2** = rows [xt : xt+dx] of first two columns of g2 as list of tuples (edges in network2)
c. **match_count** = number of edges in e1 and e2 that are the same
d. append (xt, match_count) to df_temp

5. Update global matches:
- Add the method column to df_temp.
- Append df_temp to **self.df_global**

6. Plot cumulative matches (optional):
- **If** plot=True:
a. Compute **cumulative_matches** = cumulative sum of 'matches' in df_temp.
b. Compute **percentage_matches** = (cumulative_matches / self.top) * 100.
c. **Plot** xt vs percentage_matches as a line plot.
d. Compute **AUC** using **Simpson's** rule on xt vs cumulative_matches.
e. **Normalize** AUC by dividing by max possible AUC.
f. **Print** normalized AUC.

7. Return:
- The DataFrame df_temp contains xt and match counts.

End.

Chapter 4

Methodology

4.1 Data Loading and Processing :-

The data processing step of NIRD (Network Inference by Reduced Dimensions) is built with flexibility in order to meet users' data input options for single-cell transcriptomic datasets. It accepts a raw gene expression matrix with samples as columns (typically individual cells) and gene identifiers as rows. The tool identifies and subsets batches of the data based on systematic patterns in the column names, such as experimental conditions, cell types, or sequencing protocols. As a modular design, the user can input a variety of data inputs by establishing their file paths and naming conventions accordingly.

Once the information is divided into appropriate subsets, each subset is transposed to put it into shape for subsequent purpose—usually samples as rows and genes as variables. Next we apply a feature selection step to each subset, which will eliminate low-quality/uninformative genes, using criteria such as initial expression levels or variance. For consistency across our subsets, retained will only be genes that were found across all subsets. The resultant used, cleaned datasets can then be saved for future use, and wrapped into dict objects that are structured to include metadata along with the expression matrix. All this will support the standardization of input for the subsequent stages of the NIRD pipeline, including dimensionality reduction and network inference.

4.2 Dimension Reduction using Matrix Factorization Algorithms :-

When explorations and preprocessing of gene expression data are done, the next step is dimension reductions using matrices factorization techniques. Gene expression data usually contains thousands of genes, rendering the data high-dimensional and often sparse so the biological signals of interest become hidden and hard to detect. Therefore we will use matrix factorization considered exploratory methods to reduce the initial large gene expression matrix into latent known or unknown matrices that capture the most relevant signals pertaining to the study. The latent features are simply linear combinations of genes that co-vary across the samples, and can expose underlying biological processes or regulatory modules.

NIRD has included thirteen matrix factorization algorithms in the NIRD toolbox; but each has its own distinct mathematical assumptions and properties. Some of these algorithms include non-negative matrix factorization (NMF)[10], singular value decomposition (SVD)[9], or independent component analysis (ICA)[46]. All of these methods will decompose, or factorize the gene expression data into two parts: (1) the component matrices (also called loading or coefficient matrices), which relate the original genes to the latent known, or unknown features; and (2) a mixture matrix, which describes how strongly each sample expresses these features. Therefore, this step again reduces data complexity into a more interpretable framework and it also reduces data complexity in a form that is more amenable to downstream network modeling.

The benefit of a variety of methods is to allow flexibility in the use of different methods that may capture different forms of gene expression patterns. For example, some methods assume orthogonality of factors, some prioritize non-negativity, and some rely on statistical independence. The user is thus given the option to select the method best suited to the characteristics of their sampling dataset, or they can examine outputs generated by multiple methods using different decomposition approaches to potentially better understand some underlying regulatory properties.

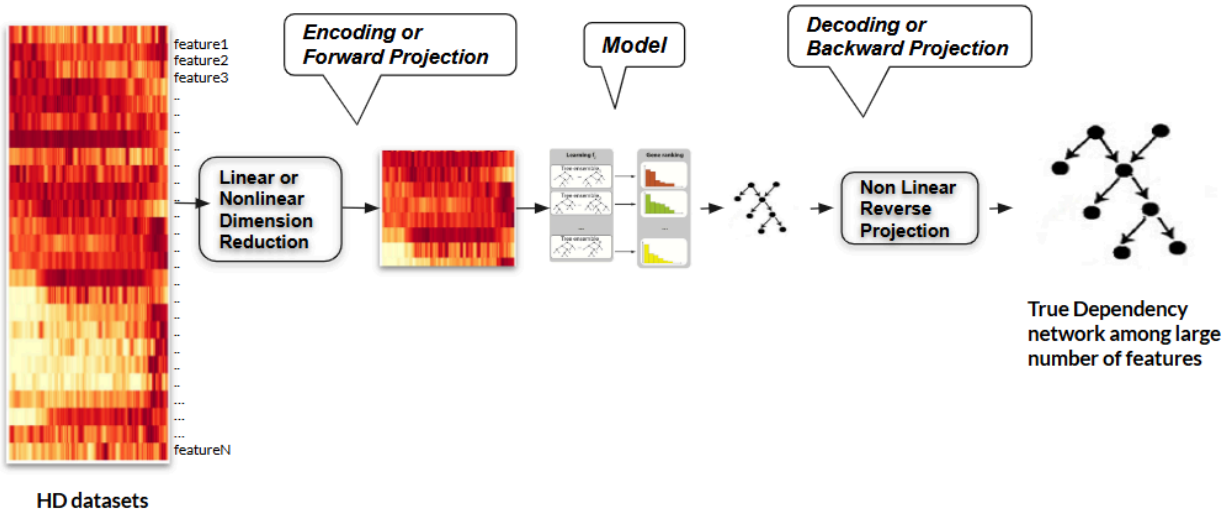


Figure 4.1: Workflow of Network Inference by Reduced Dimensions (NIRD)

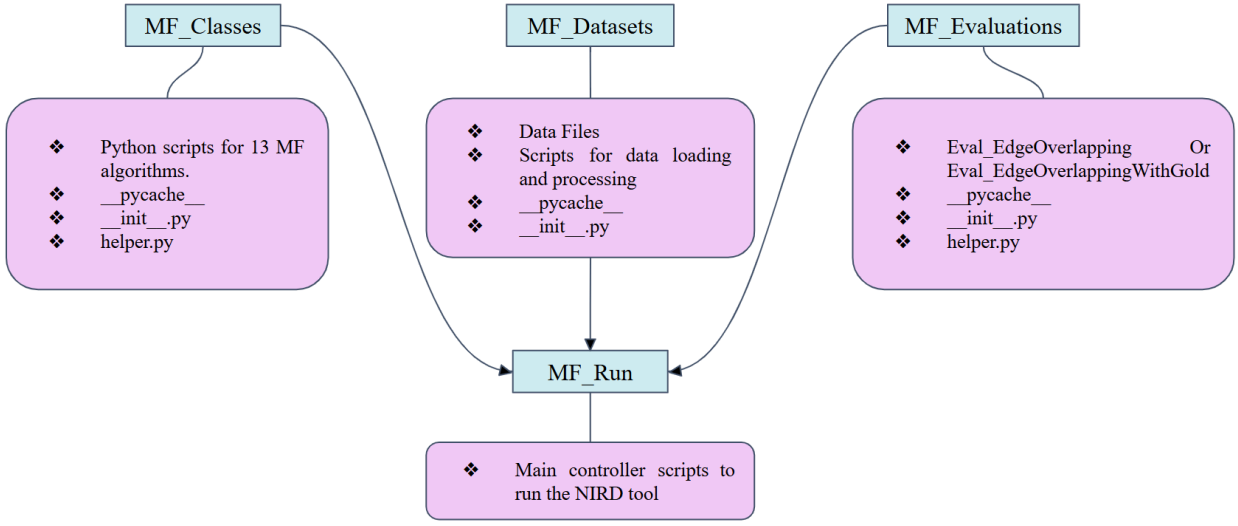


Figure 4.2: Architecture of Network Inference by Reduced Dimensions (NIRD)

4.3 Construction of GRNs :-

After reducing dimensionality, NIRD then forms GRNs by utilizing the low-dimensional representation obtained from the first step. In this phase, we proceed with our objective to model the influence of the expression of certain key genes, either transcription factors (TFs) or regulatory genes, on other genes across the set of latent features. We again treat the low-rank representations as predictors, and we model the expression of each gene using a machine learning approach, usually in the form of regression trees such as random forests. These regressors account for any non-linear dependencies across the genes and encompass any latent biological signals identified previously in the matrix factorization step outlined prior.

For each regulatory gene, the regressor predicts the expression based on the latent components; thus, each latent feature will contain some level of importance based solely on the regression fit and will indicate the strength of the contribution of the latent feature to gene regulation. Importantly, when we consider the importance scores for the regulatory gene, we combine that score with the weights from the factorized components to derive pairwise regulatory relationships among the genes. We output a weighted adjacency matrix or the network, where the edges will represent the strength/likelihood of regulatory relationships among genes.

NIRD uses a modular approach in this step by decoupling dimensionality reduction from the network inference step, which provides flexibility and robustness. Working in a reduced feature space helps increase the signal-to-noise ratio and improve our understanding of gene-gene relationships. We provide a high-confidence gene regulatory network output that can be subsequently used for other downstream analyses, such as module detection, pathway enrichment, or comparing networks arising from different conditions or datasets.

4.4 Evaluation of Inferred Networks :-

The final step of NIRD is to evaluate the inferred gene regulatory networks (GRNs) for their structure, consistency, stability, and biological relevance. Since the inferred structure of a gene regulatory network can vary based on the original datasets, the matrix factorization methods employed (if any), and many other factors, it is critical to systematically and quantitatively compare the inferred networks in order to give confidence that the gene interactions predicted are not simply due to a given data type or a specific method or approach -- but are consistent and reproducible across a range of biological systems.

As with NIRD, the evaluation is accomplished with an edge-overlapping algorithm [45] that has primarily been developed to compare pairs of inferred networks derived from different datasets or experimental conditions. This module employs the same fundamental approach - that is to convert correlation or adjacency matrices into ranked lists of gene-gene interactions (edges), and then measure how many of the best edges in one network appear in another network. The comparison of the two networks is done in a stepwise approach where the total number of edges shared is counted in increments over the ranked list to develop an edge-overlap curve.

To measure this overlap, a measure known as the area under the curve (AUC) is computed. The AUC provides a single numerical score that quantifies the similarity of two networks, such that a higher AUC indicates more overlap, and thus, a correspondingly higher degree of similarity of the implied underlying network structures that were inferred from disparate sources. These evaluation modules are adaptable and can be used for a range of input datasets and factorization outputs; thus they can be useful for comparing algorithms and validating inferred GRNs along the NIRD pipeline.

Chapter 5

Results and Discussion

To assess how well our proposed tool, NIRD, operates, we performed extensive benchmarking on a wide range of biological datasets. This benchmarking included both real and simulated datasets from varying biological contexts. This gave us an opportunity to thoroughly evaluate the generalizability, robustness, accuracy, and scalability of the tool.

1. Performance on Heterogeneous Biological Matrices:
NIRD was tested on the following biological matrices:

- ❖ Simulated data
- ❖ Transcriptomics data (bulk and single-cell)
- ❖ Time Course Data
- ❖ Transcription Velocity Data

The diversity of matrices listed above illustrates the versatility of NIRD as an analysis tool capable of working in a heterogeneous omics space.

2. Matrix Factorization Provides More Accurate Inference:
NIRD included 13 matrix factorization techniques to reduce the size of the input matrices before generating interaction networks. Dimensionality reduction mitigates the noise and sparsity common in biological datasets, thereby enabling signal extraction. Random Forest is then applied to the original, reduced data to create a gene-gene or gene-TF interaction network, which generates a scored interaction network according to feature importance.

3. Comparison to Best Practices:
We compared the performance of the matrix factorization methods of NIRD to six tried-and-true network inference algorithms:

1) ARACNE
2) RELNET
3) MRNET
4) C3NET
5) GENIE3
6) GrnBoost2

We evaluated our comparative performance using edge overlapping analysis, which includes evaluating the area under the curve (AUC) of matched edges between the predicted and gold standard networks.

4. Performance Analysis Across Datasets:

Characterization was completed for the following datasets:

1) DREAM5 Challenge Datasets
    a) Net1 (in silico)
    b) Net2 (S. aureus)
    c) Net3 (E. coli)
    d) Net4 (S. cerevisiae)
2) Real single-cell and bulk datasets
    a) mESC: Batches, Gold Standard
    b) Pancreas (GSE81547)
    c) Human knee cartilage (GSE255460)
    d) hESC (GSE75748): Time course (0hr, 12hr), transcriptional velocity

5. Summary of Findings:

- NIRD's matrix factorization methods consistently performed better than competitor algorithms across biological datasets with varying properties. In general, the AUC scores were significantly higher than the competitors in the majority of experiments.
- With exception of a dataset in silico (DREAM5-Net1) which GrnBoost2 outperformed other methods.
- The runtime of NIRD's matrix factorization methods was 5-10x faster than competitors suggesting the algorithms are computationally efficient.
- The AUC-based characterizing method permitted the identification of the optimum matrix factorization method, across data sets and provided a criteria-based approach for selection of one of the multiple downstream candidate methods into network analysis.

6. Robustness and Generalizability:

- NIRD's ability to perform across networks with increasing levels of noise and sparsity, as well as its ability to add innovation into the biological problem space, speak to the realism of the NIRD-modules.
- Time-course and transcriptional velocity data additionally provided further validation of NIRD for inferring regulatory interactions with temporal relevance.
- Single-cell datasets with batch effects were effectively down-sampled by using the denoising capabilities of dimensionality reduction.

1) DREAM5 Datasets Results :-

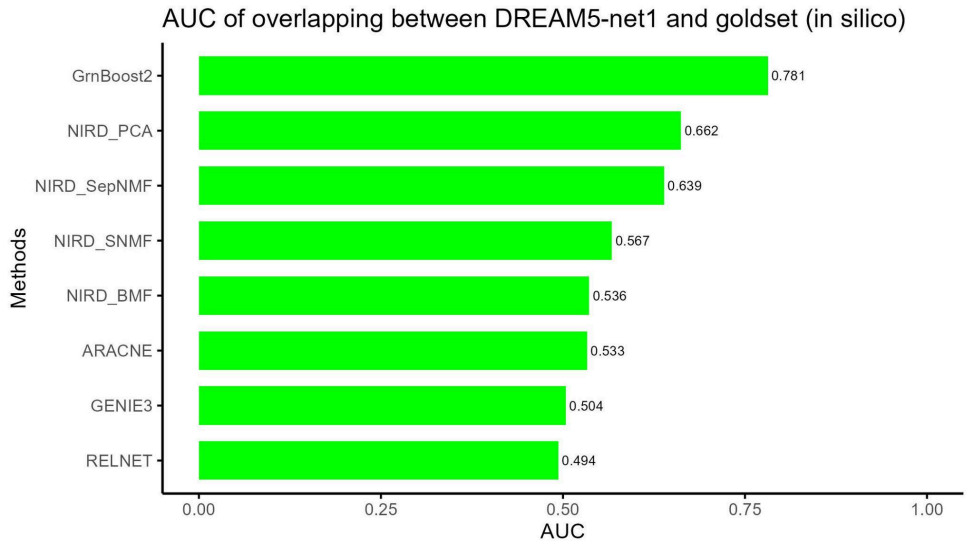


Figure 5.1: Method-wise AUC for DREAM5-net1 & goldset Network Overlap

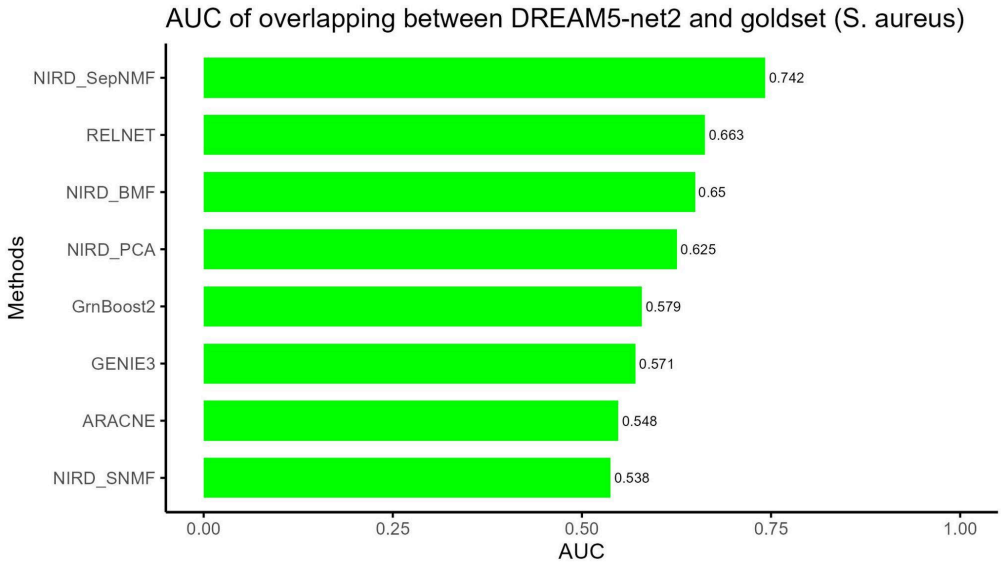


Figure 5.2: Method-wise AUC for DREAM5-net2 & goldset Network Overlap

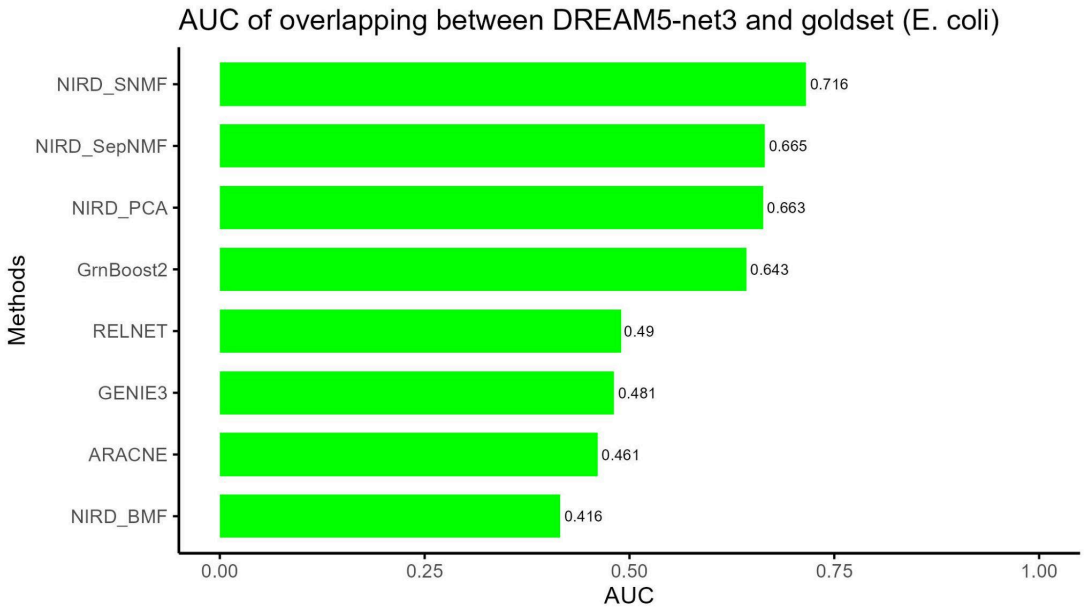


Figure 5.3: Method-wise AUC for DREAM5-net3 & goldset Network Overlap

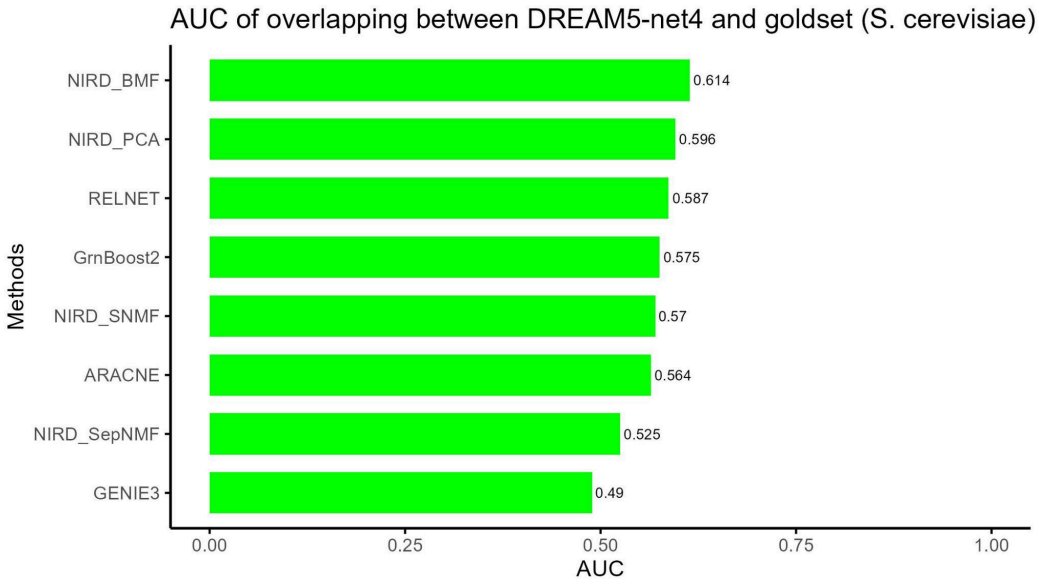


Figure 5.4: Method-wise AUC for DREAM5-net4 & goldset Network Overlap

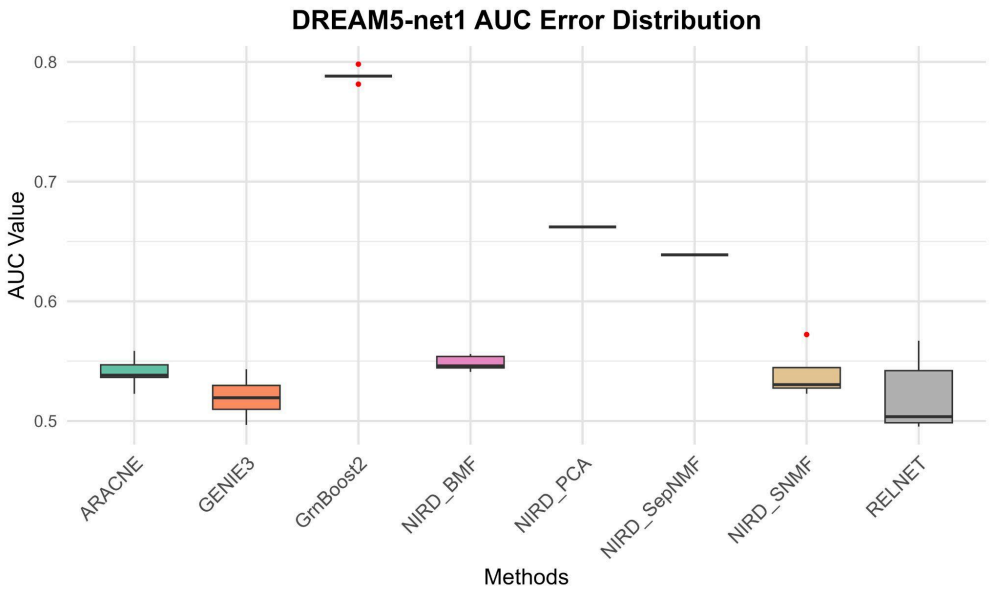


Figure 5.5: AUC Distribution Across Methods (DREAM5-net1)

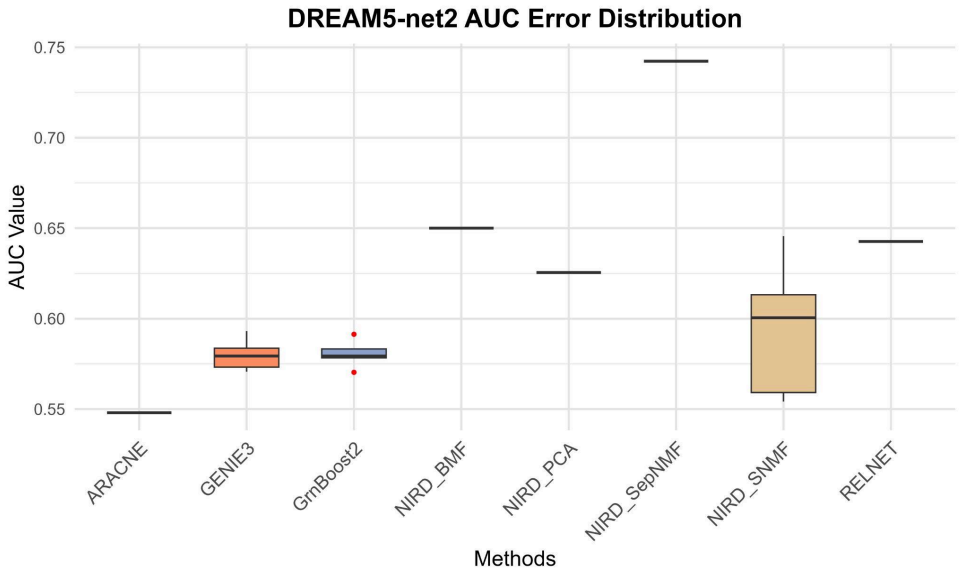


Figure 5.6: AUC Distribution Across Methods (DREAM5-net2)

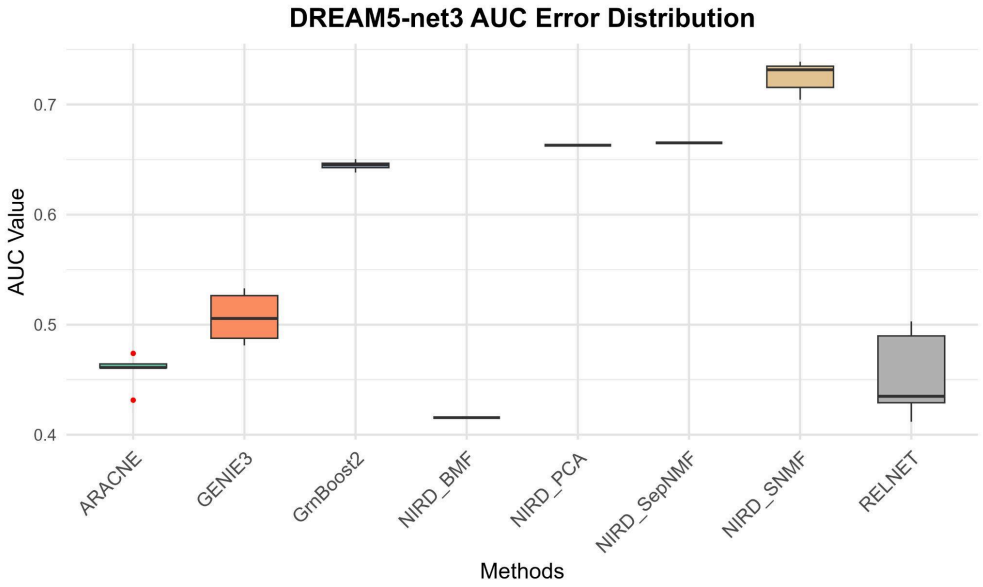


Figure 5.7: AUC Distribution Across Methods (DREAM5-net3)

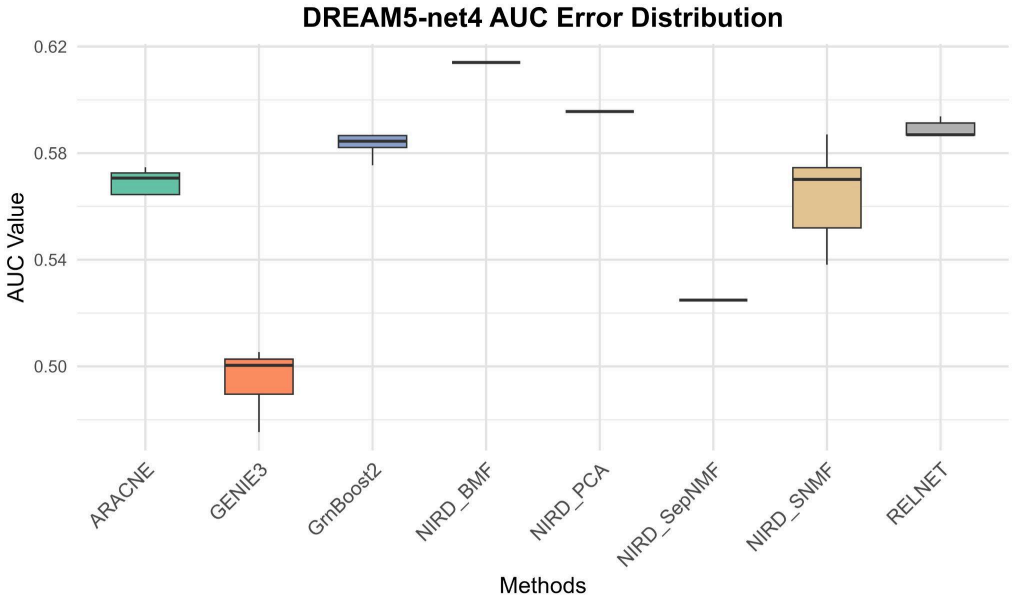


Figure 5.8: AUC Distribution Across Methods (DREAM5-net4)

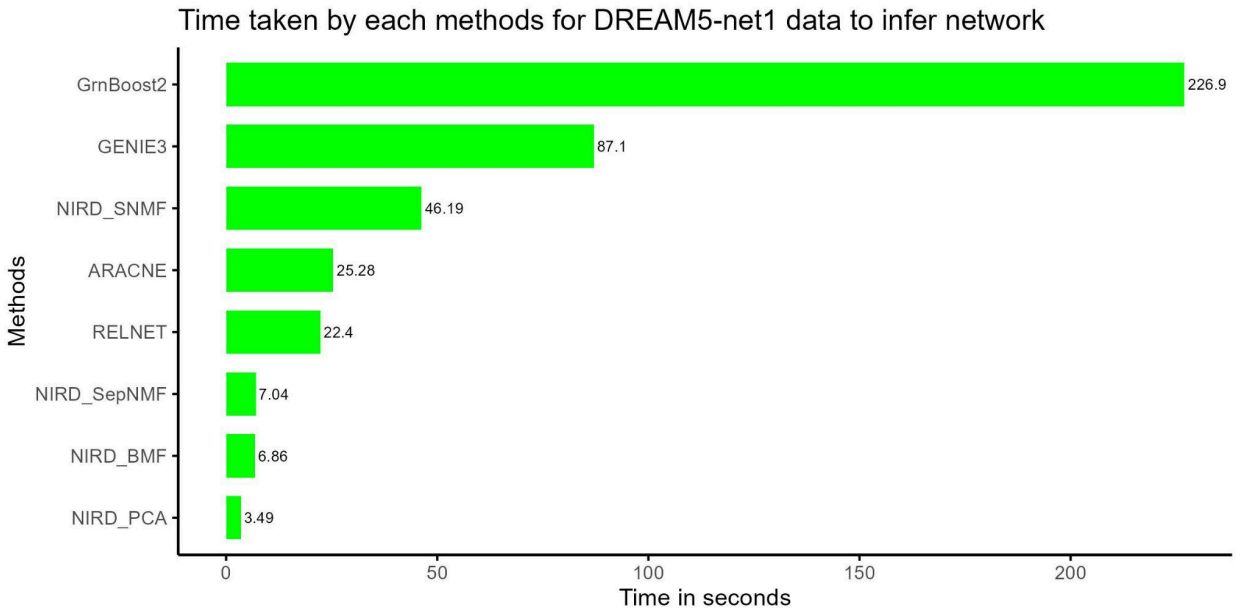


Figure 5.9: Execution Time per Method on DREAM5-net1 Dataset

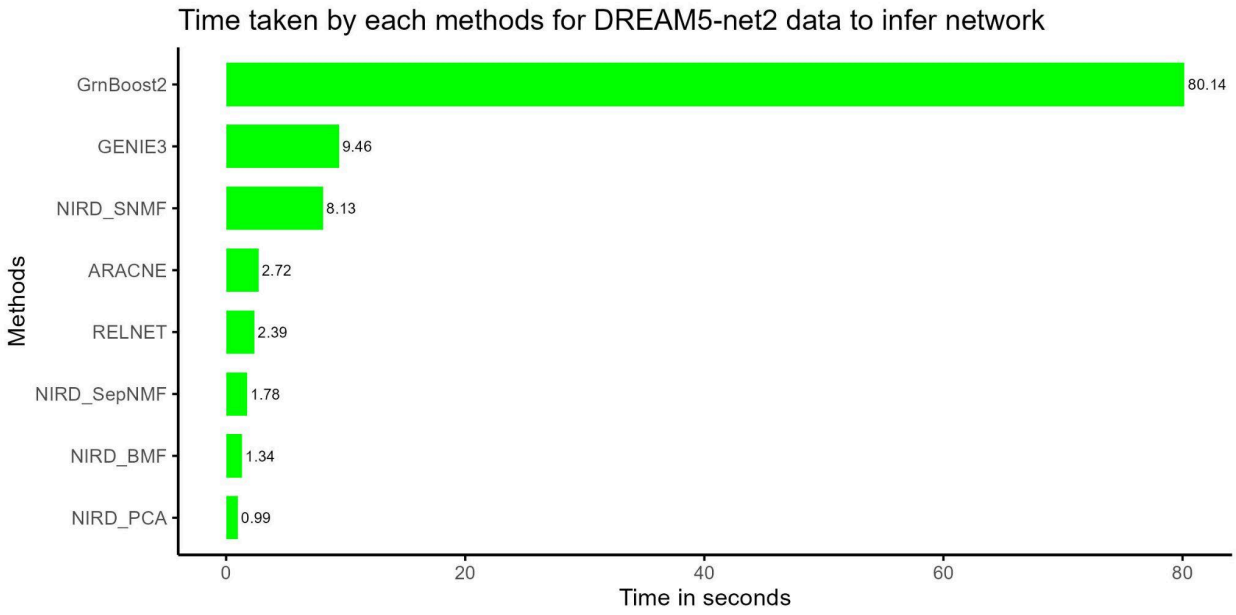


Figure 5.10: Execution Time per Method on dream5-net2 Dataset

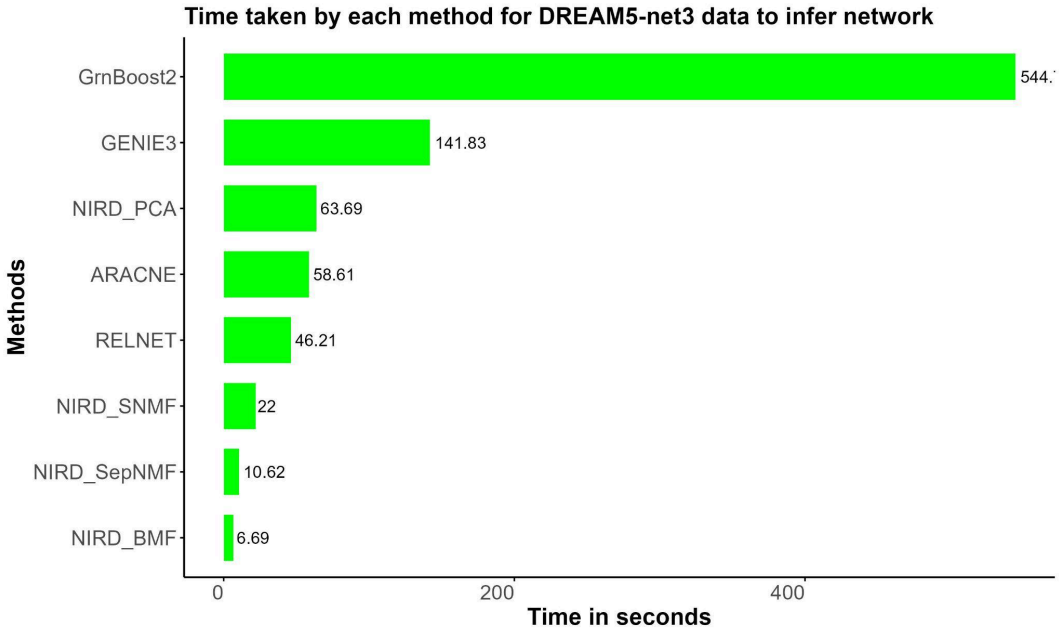


Figure 5.11: Execution Time per Method on DREAM5-net3 Dataset

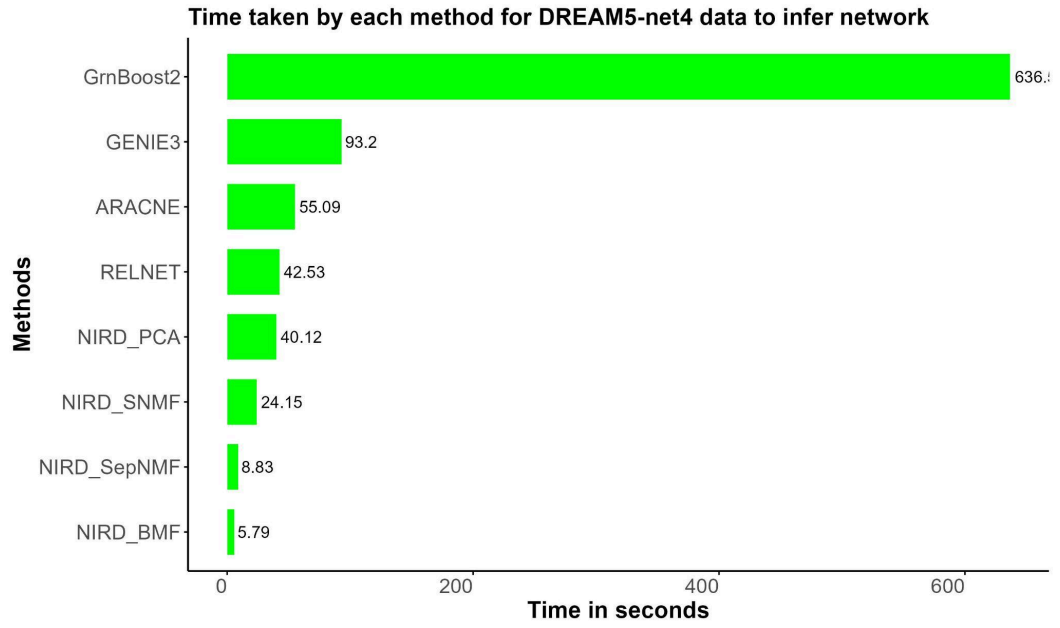


Figure 5.12: Execution Time per Method on DREAM5-net4 Dataset

2) mESC Results :-

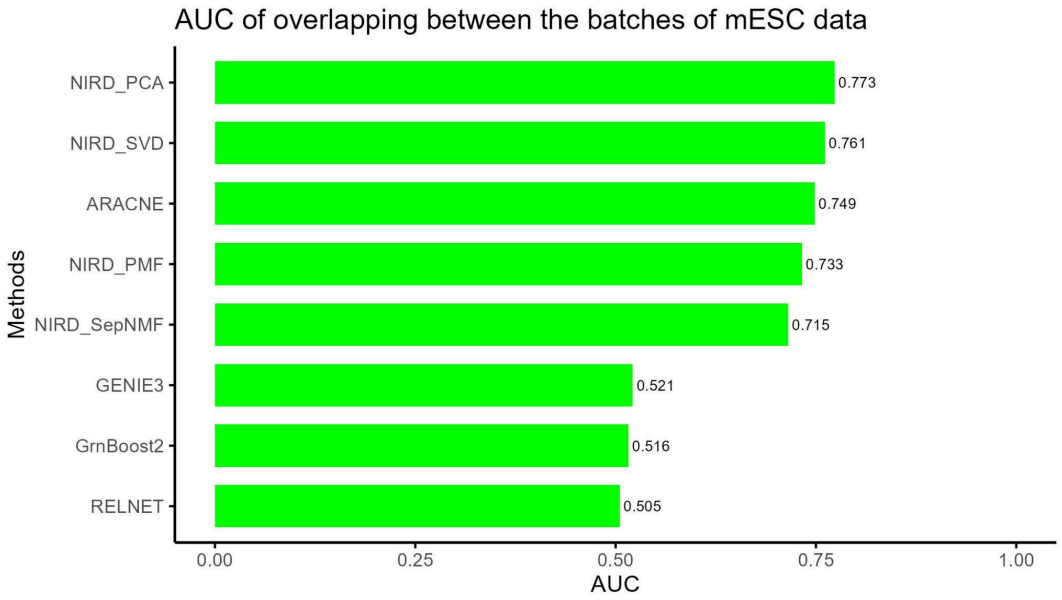


Figure 5.13: Method-wise AUC for mESC smartSeq-dropSeq Network Overlap

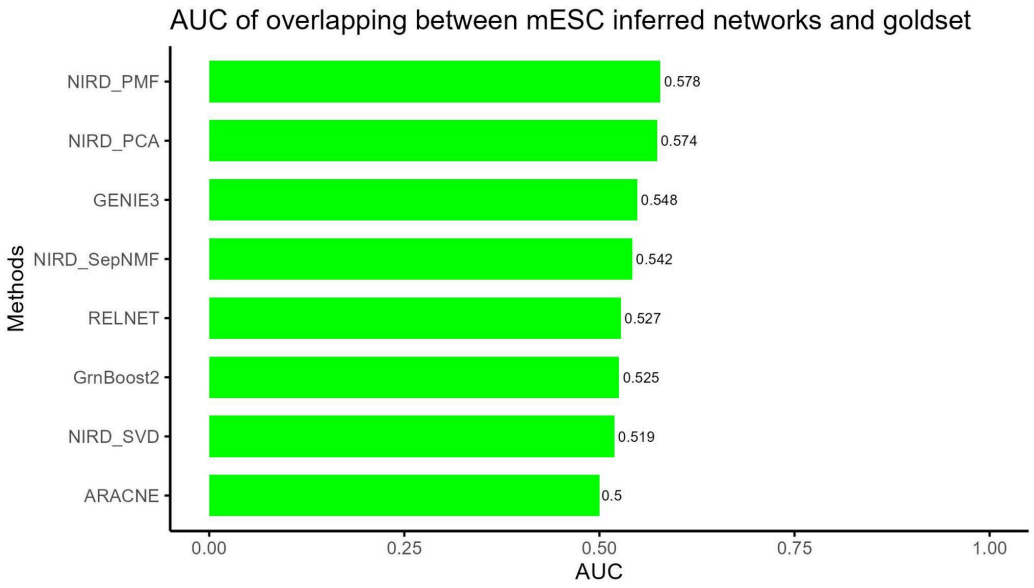


Figure 5.14: Method-wise AUC for mESC inferred networks & Goldset Overlap

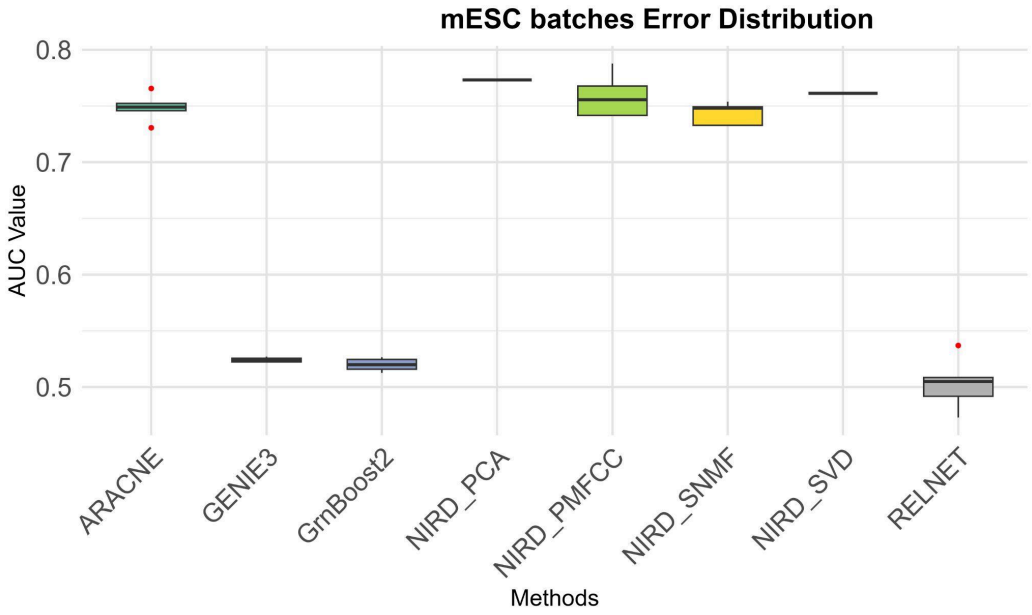


Figure 5.15: AUC Distribution Across Methods (mESC batches)

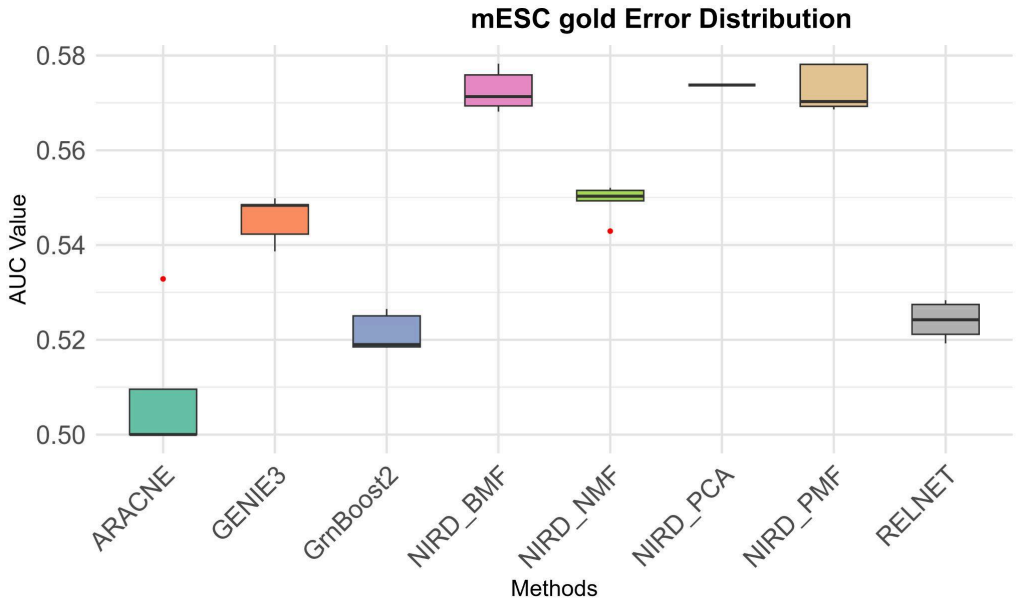


Figure 5.16: AUC Distribution Across Methods (mESC Gold)

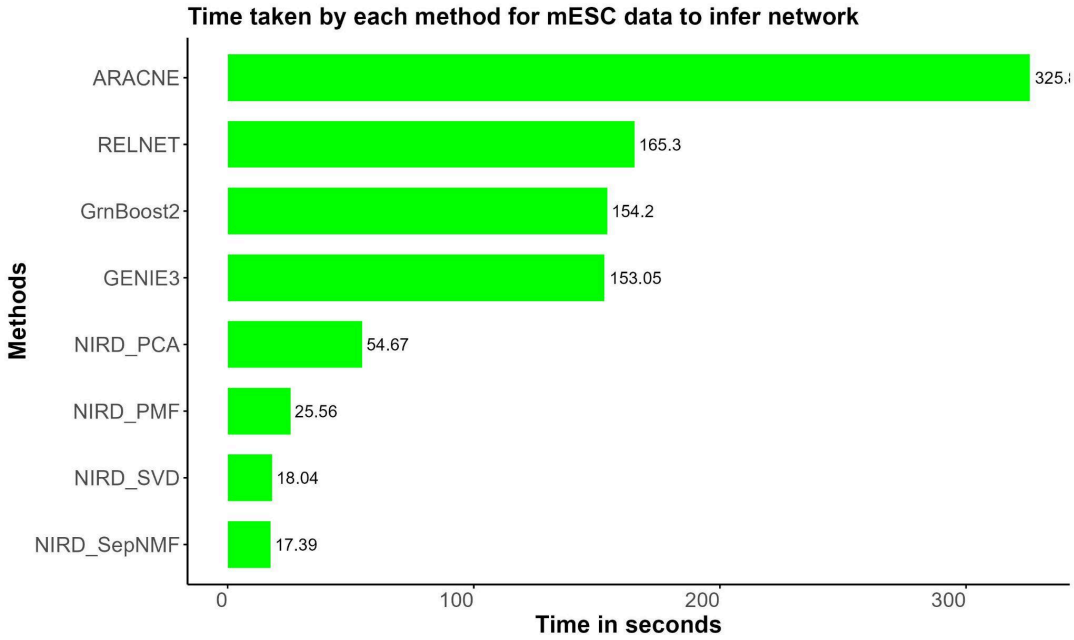


Figure 5.17: Execution Time per Method on mESC Dataset

3) Pancreas Results :-

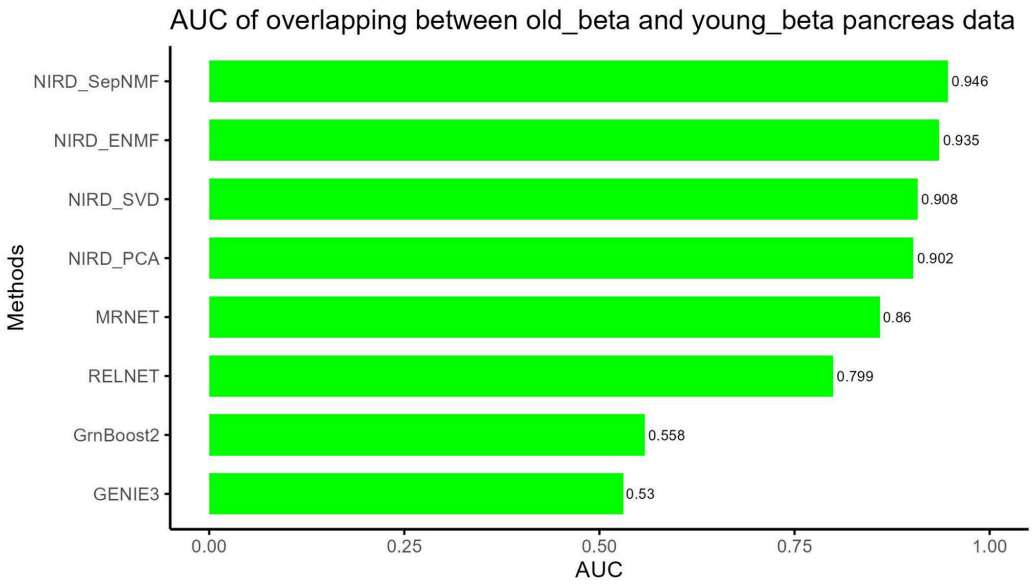


Figure 5.18: Method-wise AUC for pancreas old_beta & young_beta Network Overlap

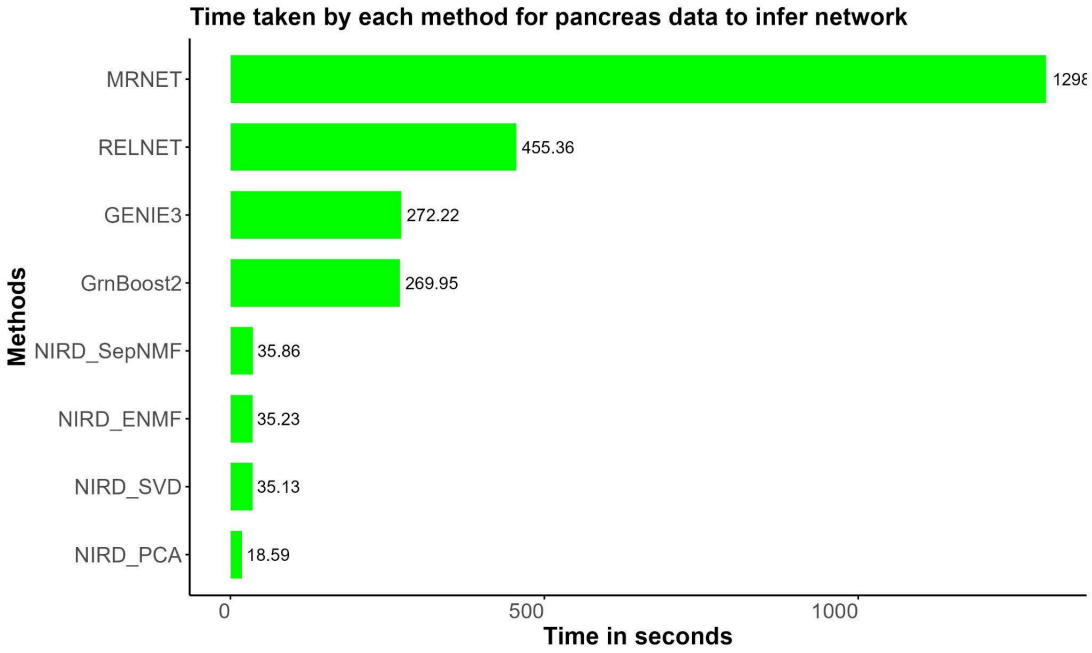


Figure 5.19: Execution Time per Method on Pancreas Dataset

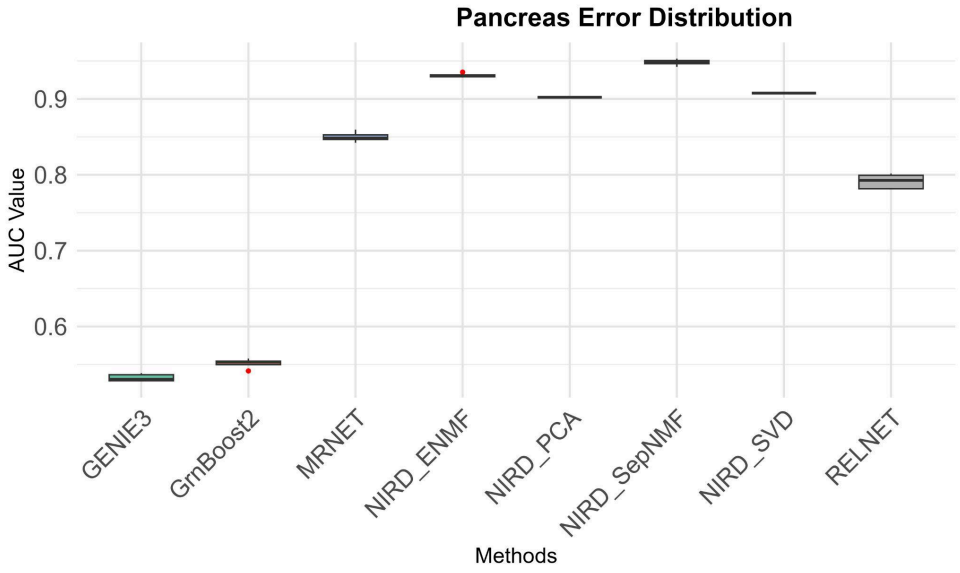


Figure 5.20: AUC Distribution Across Methods (Pancreas Data)

4) Knee Cartilage (Osteoarthritis) Results :-

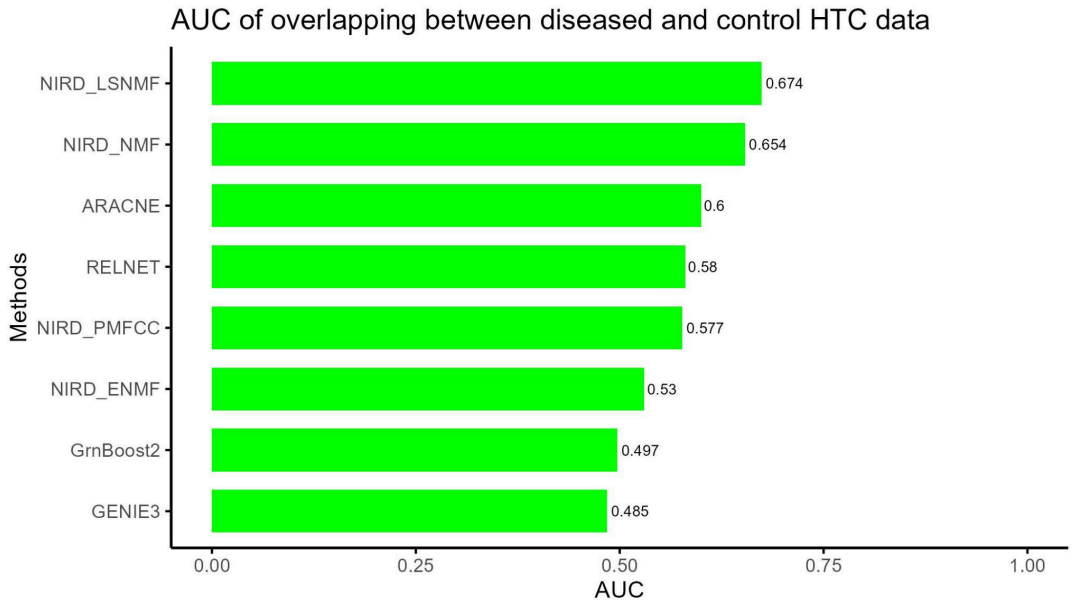


Figure 5.21: Method-wise AUC for HTC Disease-Control Network Overlap

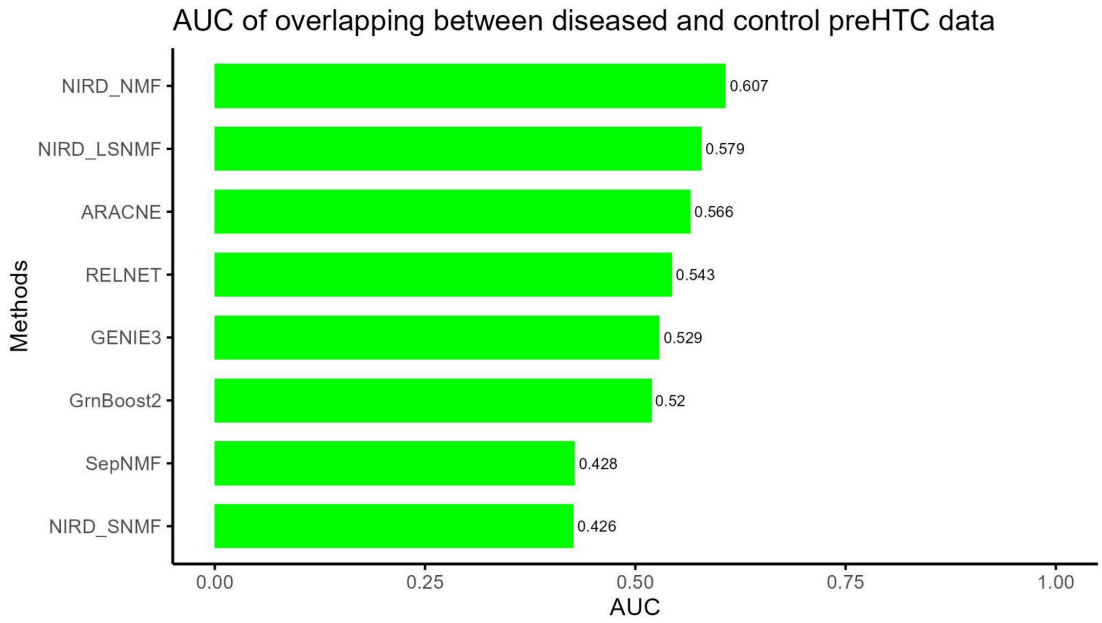


Figure 5.22: Method-wise AUC for preHTC Disease-Control Network Overlap

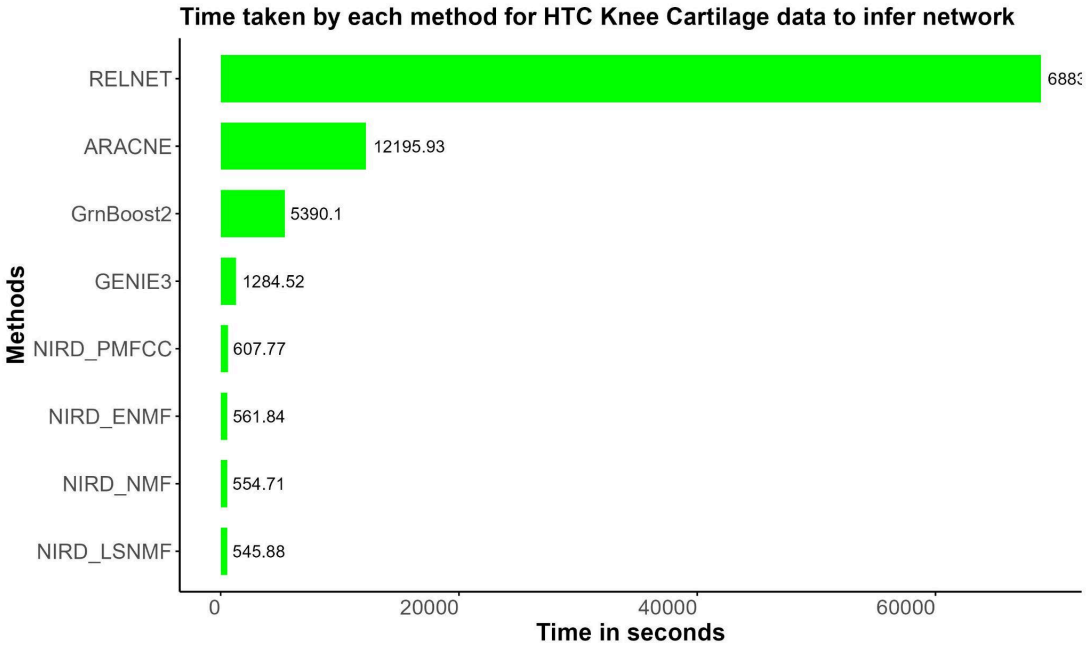


Figure 5.23: Execution Time per Method on HTC Dataset

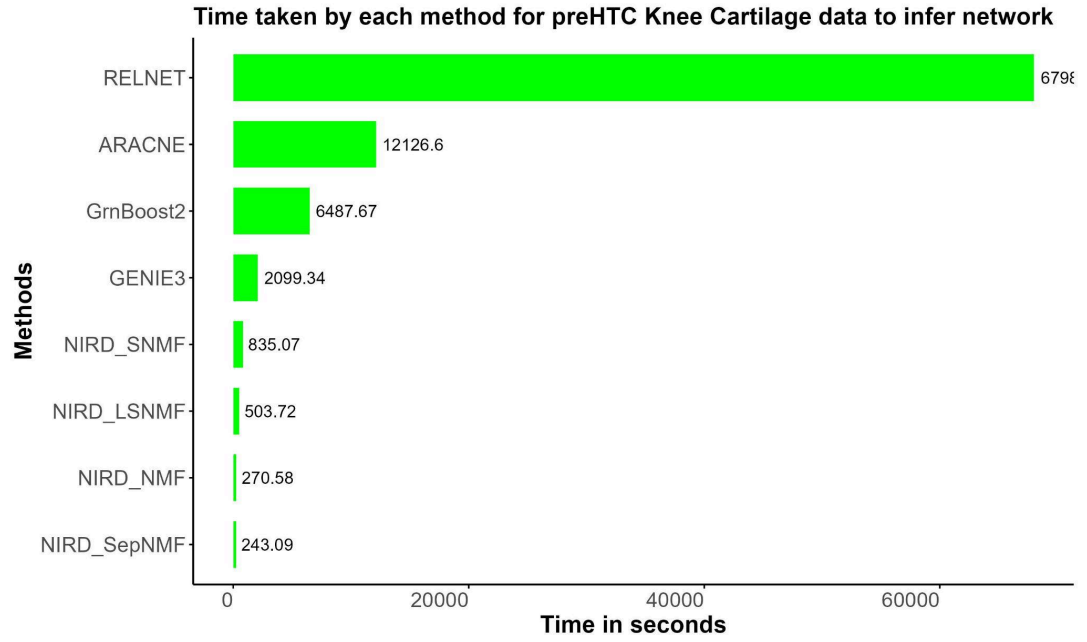


Figure 5.24: Execution Time per Method on preHTC Dataset

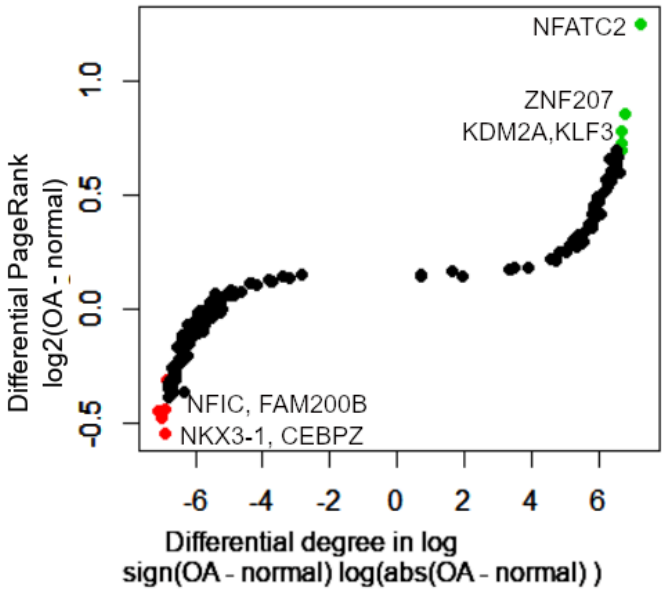


Figure 5.25: Differential Centrality Analysis (HTC OA vs Normal)

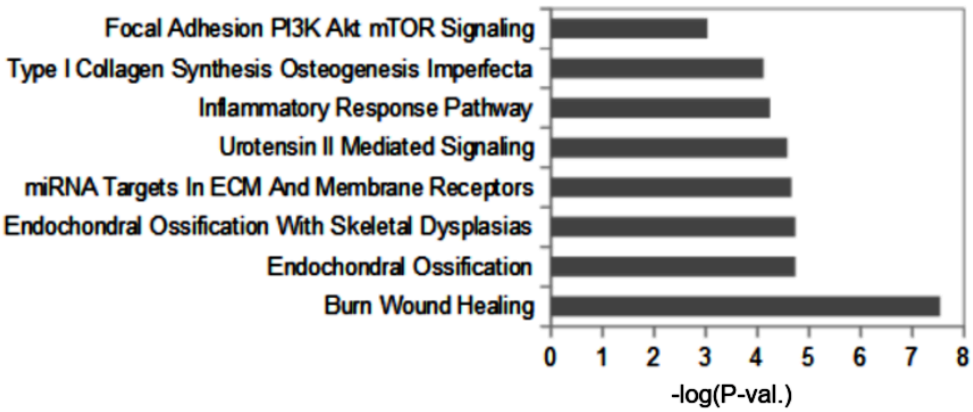


Figure 5.26: Pathway Enrichment in HTC Data

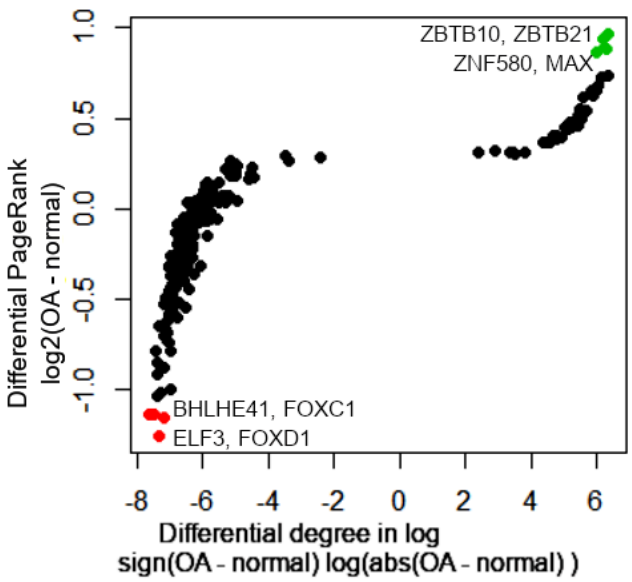


Figure 5.27: Differential Centrality Analysis (preHTC OA vs Normal)

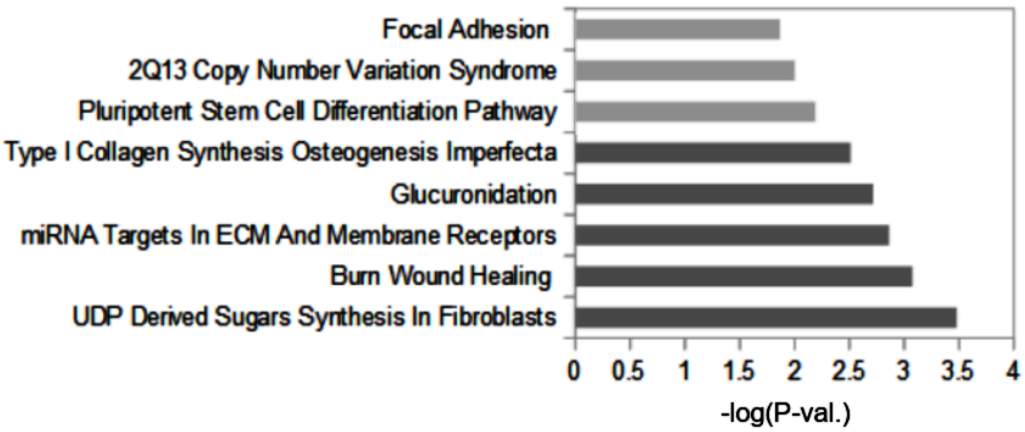


Figure 5.28: Pathway Enrichment in preHTC Data

5) hESC (Time Course and RNA Velocity) Results :-

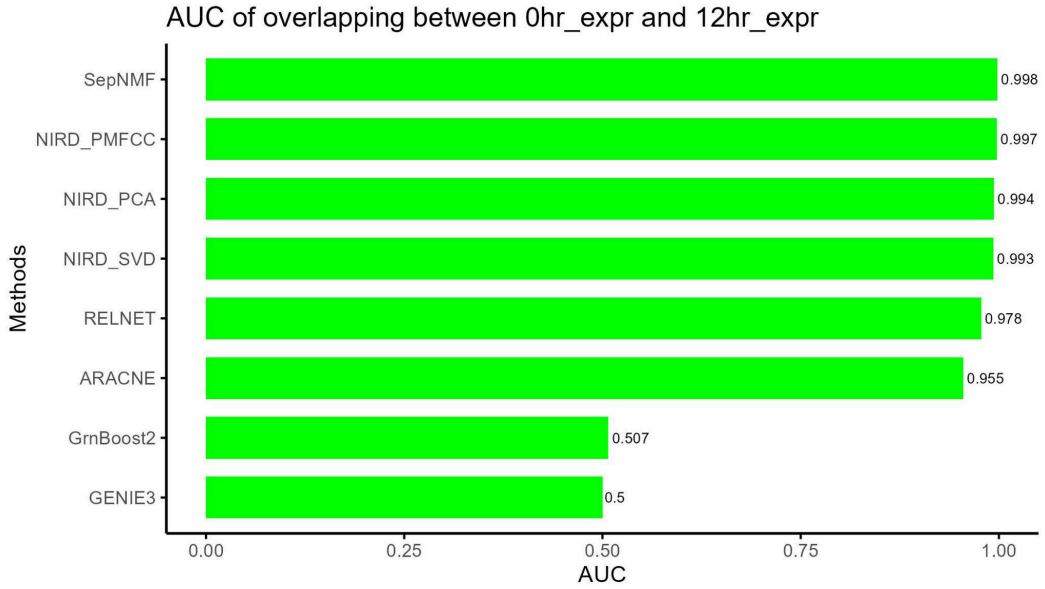


Figure 5.29: Method-wise AUC for 0hr_expr and 12hr_expr Network Overlap

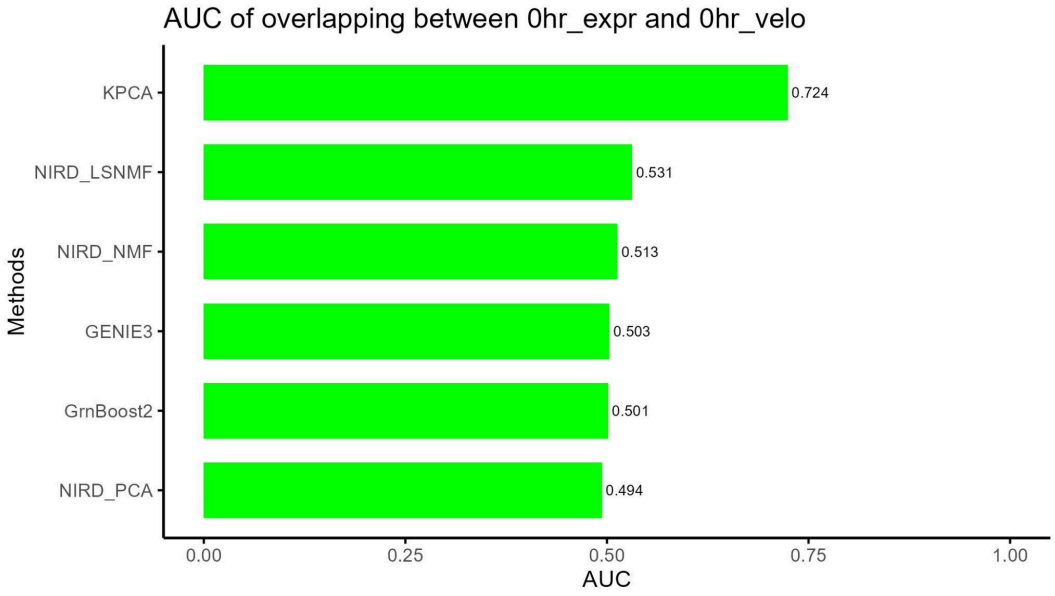


Figure 5.30: Method-wise AUC for 0hr_expr and 0hr_velo Network Overlap

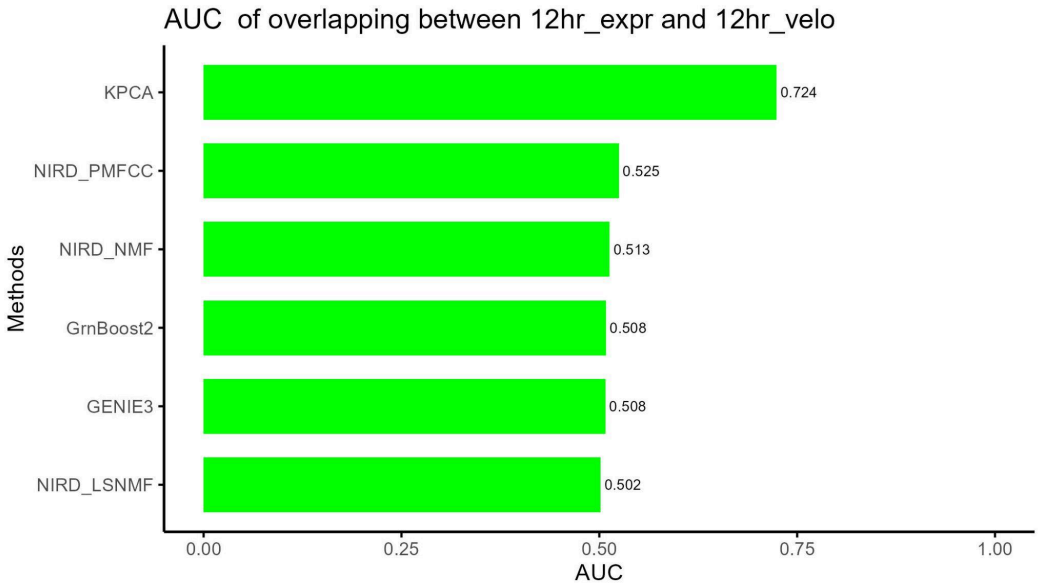


Figure 5.31: Method-wise AUC for 12hr_expr and 12hr_velo Network Overlap

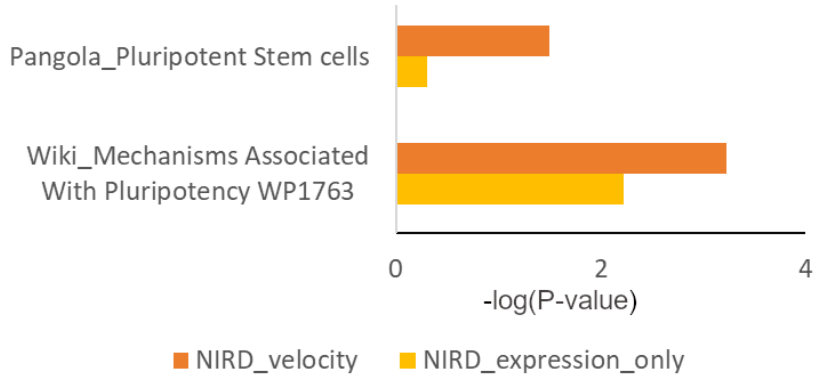


Figure 5.32: Pluripotency Pathway Enrichment: Velocity vs Expression

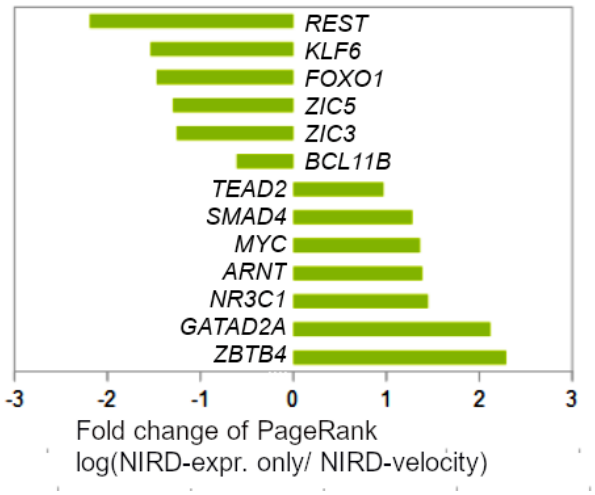


Figure 5.33: Differential PageRank of TFs: Velocity vs Expression

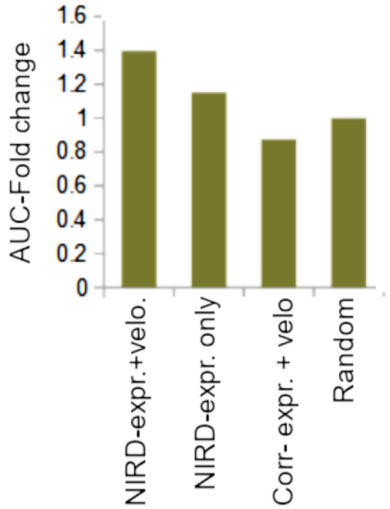


Figure 5.34: AUC-Fold Change Comparison Across Methods

Discussion and Future Scope :-

The performance of the NIRD (Network Inference by Reduced Dimensions) tool has proven reliable across a variety of low dimensional biological datasets organized in matrix format - genes × samples or genes × cells. During both the benchmarking and validation experiments, we had success with NIRD inferring meaningful gene regulatory networks from all of the data used, such as:

- Simulated gene expression datasets to validate ground truth recovery,
- Bulk RNA-seq gene expression data from tissue- or population-level sampling,
- Single cell RNA-seq gene expression data, which had expression heterogeneity across individual cells,
- Time-course expression data, where we modeled dynamic patterns of regulation over sequential time points,
- RNA velocity data measuring directional transcriptomic changes based on unspliced and spliced mRNA abundance.

Because NIRD employs a matrix-based structure, it is also conceivable that NIRD could be extended to other non-transcriptomic datasets. Many omics and functional genomics datasets are also organized in a matrix-style fashion and therefore allow for new data sources to develop new inference networks using NIRD. Below are a few of these type of datasets that NIRD could conceivably be adapted to:

1) **Mutation Matrices (Gene/Position × Sample):** These matrices contain information on the occurrence (or frequency) of certain mutations - like SNPs, insertions, deletions - in different samples. NIRD could be applied in order to examine co-mutated gene networks or infer mutational-co-occurrence modules relevant to diseases (such as cancer).

2) **ChIP-seq Matrix (Peak × Sample):** In this case, each row of the matrix indicates a genome binding peak (e.g., transcription factor binding site), and the associated value could be a read count or signal intensity. NIRD might possibly identify co-regulatory modules, or interactions between binding sites across conditions or samples.

3) **Drug-response Matrix (Drug × Sample):** Drug-response datasets measure cellular responses to drug perturbations like IC50 or AUCs. Drug-response features might be treated as features in terms of how NIRD may develop new drug-gene networks, or possibly infer mechanisms of drug-resistance.

4) **Methylation Matrices (CpG Site × Sample):** These matrices provide the relative DNA methylation of targeted CpG sites. NIRD could potentially reveal epigenetically regulated gene modules or modules consisting of regions that behave together (i.e. collectively) across phenotypes or cell states.

5) **Copy Number Variation (CNV) Matrices (Genomic Region × Sample):** CNV data reflect gains, losses or neutral copy number states at genomic segments. Incorporation of NIRD might be helpful to identify patterns related to changes in regulatory gene function associated with CNV or genomic alterations specific to tumors

6) **Proteomics Matrices (Protein × Sample):** These data reflect the relative abundances of proteins. Because proteins are direct effectors of cellular functions, NIRD could offer adjacent regulatory insights beyond the transcriptomics cue if we applied NIRD in proteomics datasets.

7) **Metabolomics Matrices (Metabolite × Sample):** These matrices reflect metabolite concentrations, and therefore, also have potential to identify specific functional metabolic pathways. Further, network inference methods applied to metabolomics data could potentially identify patterns of co-regulation between metabolites or reveal metabolic reprogramming associated with disease.

8) **Abundance Matrices from metagenomics analyses (Species/Gene × Sample):** These correspond to the makeup or the abundances of genes, in environmental or host samples. NIRD could be used to characterize microbial interaction networks or to identify functional gene modules from microbiomes.

In conclusion, the versatility of NIRD applied to many types of matrix-based omics data gives a new possibility for evaluation in more systems biology applications in the future. Future work should deliver optimizations to the preprocessing pipeline and distance selection related to these various types of omics data. It is also important to consider context-specific validation strategies, as inferences from NIRD should be considered within the context of each study, where the inferred networks are validated in meaningful ways.

Bibliography

[1] Sandberg, R. (2014). Entering the era of single-cell transcriptomics in biology and medicine. *Nature Methods*, *11*(1), 22-29.

[2] Kharchenko, P. V., Silberstein, M., & Scadden, D. T. (2014). Bayesian approach to single-cell differential expression analysis. *Nature Methods*, *11*(7), 740-742.

[3] Chen, G., Ning, K., & Shi, T. (2018). scRNA-seq data analysis: challenges and opportunities. *Bioinformatics*, *34*(17), i3-i9.

[4] Davidson, E. H. (2001). *Genomic regulatory systems: development and evolution*. Academic press.

[5] Bellman, R. E. (2015). *Dynamic programming*. Courier Dover Publications. (While a classic, you'd cite a more relevant paper discussing the curse of dimensionality in the context of scRNA-seq if possible).

[6] Hicks, S. C., Townes, F. W., & Irizarry, R. A. (2018). Missing data and technical variation in single-cell RNA-seq. *Genome Biology*, *19*(1), 206.

[7] Kauffman, S. A. (1969). Metabolic stability and epigenesis in randomly constructed genetic nets. *Journal of Theoretical Biology*, *22*(3), 437-467. [1] (Consider a more modern reference on non-linear gene regulation if available).

[8] Bansal, M., Della Gatta, G., & di Bernardo, D. (2007). Inference of gene regulatory networks and their modules by multi-objective optimization. *Bioinformatics*, *23*(13), i171-i179.

[9] Leveraging prior knowledge to infer gene regulatory networks from single-cell RNA-sequencing data | Molecular Systems Biology - EMBO Press. (n.d.)

[10] Lee, D. D., & Seung, H. S. (1999). Learning the parts of objects by non-negative matrix factorization. *Nature*, *401*(6755), 788-791.

[11] Davidson, E. H. (2001). *Genomic regulatory systems: development and evolution*. Academic press. (A foundational text)

[12] Butte, A. J., Tamayo, P., Slonim, D., Golub, T. R., & Kohane, I. S. (2000). Discovering functional relationships between genes through the analysis of microarray data. *Proceedings*

of the National Academy of Sciences, *97*(22), 12132-12137.

[13] Margolin, A. A., Nemenman, I., Basso, K., Califano, A., & Stolovitzky, G. (2006). ARACNE: an algorithm for the reconstruction of gene regulatory networks in a mammalian cellular context. *BMC bioinformatics*, *7*(Suppl [1] 1), S7.

[14] Basso, K., Margolin, A. A., Stolovitzky, G., Klein, U., Dalla Gatta, G., Califano, A., & Stolovitzky, G. (2005). Reverse engineering of regulatory networks in human B cells. *Nature genetics*, *37*(4), 382-390.

[15] Faith, J. J., Driscoll, L. S., Pepin, D., Schwikowski, B., &жным, T. G. (2007). Inferring transcriptional regulation using modularity clustering by edge weights (MOCEW) in Escherichia coli. *Molecular systems biology*, *3*(1), 103.

[16] Li, J., Zhou, L., & Zhou, Y. (2018). scImpute: accurate and robust imputation for single-cell RNA-seq data. *Nature communications*, *9*(1), 1773. (Example of imputation in scRNA-seq)

[17] Trapnell, C., Cacchiarelli, D., Grimsby, J., Karthaus, W. M., Koch, C., Hennig, M., ... & van Oudenaarden, A. (2014). The dynamics and regulators of cell fate decisions are revealed by pseudotemporal ordering of single cells. *Nature biotechnology*, *32*(4), 381-386. [2]

[18] Moerman, T., Aibar, S., Bravo González-Blas, C., & Geurts, P. (2019). GRNBoost2: scalable and robust network inference using gradient boosting. *Bioinformatics*, *35*(21), 4344-4346. (Example of a more recent GRN inference tool)

[19] Raychaudhuri, S., Stuart, J. M., & Troyanskaya, O. G. (2010). Inferring gene–gene relationships from expression data. *PLoS computational biology*, *6*(6), e1000802. (Discusses dimensionality reduction in GRN inference)

[20] Huynh-Thu, V. A., & Sanguinetti, G. (2015). Inferring gene regulatory networks using tree-based methods. *BMC bioinformatics*, *16*(1), 11.

[21] Liang, Y., Zhao, X., Li, J., & Zhang, S. (2017). Inferring gene regulatory networks from single-cell RNA-seq data using deep learning. *Bioinformatics*, *33*(19), 2942-2949.

[22] Siahpirani, A., & Roy, S. (2017). Bayesian inference of gene regulatory networks from single-cell transcriptomic data using Gaussian processes. *Bioinformatics*, *33*(14), 2156-2164.

[23] Meyer, P. E., Lafitte, F., & Bontempi, G. (2018). The DREAM5 network inference challenge: statistical assessment of results. *BMC Systems Biology*, *12*(Suppl 6), 114.

[24] Fan, Y., Bian, X., Meng, X., Li, L. et al. Unveiling inflammatory and prehypertrophic cell

populations as key contributors to knee cartilage degeneration in osteoarthritis using multi-omics data integration. *Annals of Rheumatic Diseases*, *83*(7), 926-944.

[25] Enge, M., Arda, H. E., Mignard, M., Beausang, J., et al. Single-Cell Transcriptome Analysis of Human Pancreas Reveals Transcriptional Signatures of Aging and Somatic Mutation Patterns. *Cell Metabolism*, *26*(5), 1287-1302.e14.

[26] Chu, J., Zhou, Z., Zhu, S., Jiang, F., Trapnell, C., Yu, J., ... & Huang, H. (2016). Snapshot and temporal single-cell RNA-seq reveals the dynamics of human ESC differentiation to definitive endoderm. *Cell Stem Cell*, *19*(6), 771-784.

[27] Koren, Y., Bell, R., & Volinsky, C. (2009). Matrix factorization techniques for recommender systems. *Computer*, *42*(8), 30-37.

[28] Ricci, F., Rokach, L., & Shapira, B. (2011). Introduction to recommender systems handbook. In *Recommender systems handbook* (pp. 1-35). Springer, Boston, MA.

[29] Strang, G. (1993). *Linear algebra and its applications* (3rd ed.). Harcourt Brace Jovanovich.

[30] Schmidt, M. N., Winther, O., & Hansen, L. K. (2009). Bayesian Non-negative Matrix Factorization. In T. Adali, C. Jutten, J. M. T. Romano, & A. K. Barros (Eds.), *Independent Component Analysis and Signal Separation* (pp. 540–547). Springer.

[31] Zhang, Z., Li, T., Ding, C., & Zhang, X. (2007). Binary Matrix Factorization with Applications. *Seventh IEEE International Conference on Data Mining (ICDM 2007)*, 391–400.

[32] Jia, Y. W. Y., & Turk, C. H. M. (2004, January). Fisher non-negative matrix factorization for learning local features. In Proc. Asian conf. on comp. vision (pp. 27-30). Citeseer.

[33] Kim, H., & Park, H. (2008). *Non-negative matrix factorization based on alternating non-negativity constrained least squares and active set method.* SIAM Journal on Matrix Analysis and Applications, 30(2), 713–730.

[34] Yang, Z., Zhang, H., Yuan, Z., & Oja, E. (2011). Kullback-Leibler Divergence for Nonnegative Matrix Factorization. In T. Honkela, W. Duch, M. Girolami, & S. Kaski (Eds.), *Artificial Neural Networks and Machine Learning – ICANN 2011* (pp. 250–257). Springer..

[35] Salakhutdinov, R., & Mnih, A. (2008). *Probabilistic Matrix Factorization*. In Advances in Neural Information Processing Systems (NeurIPS).

[36] Park, H., & Kim, H. (2007). *Sparse non-negative matrix factorizations via alternating non-negativity-constrained least squares for microarray data analysis*. *Bioinformatics*, 23(12),

1495–1502.

[37] van Benthem, M. H., & Keenan, M. R. (2004). *Fast algorithm for the solution of large-scale non-negativity-constrained least squares problems*. *Journal of Chemometrics*, 18(10), 441–450.

[38] (PDF) Semi-Supervised Clustering via Matrix Factorization. (n.d.). *ResearchGate*. Retrieved June 27, 2025.

[39] Salem, N., & Hussein, S. (2019). Data dimensional reduction and principal components analysis. *Procedia Computer Science*, *163*, 292–299.

[40] Schölkopf, B., Smola, A., & Müller, K.-R. (1998). *Nonlinear component analysis as a kernel eigenvalue problem*. *Neural Computation*, 10(5), 1299–1319.

[41] Lähnemann, D., et al. (2020). Eleven grand challenges in single-cell data science. *Genome Biology*, 21(1), 31.

[42] Zheng, Y., et al. (2022). Comprehensive comparison of dimensionality reduction methods for single-cell RNA sequencing data visualization. *Briefings in Bioinformatics*, 23(1), bbab440.

[43] Coifman, R. R., et al. (2005). Geometric diffusions as a tool for multiscale data analysis. *Proceedings of the National Academy of Sciences*, 102(21), 7426-7431. (While this paper introduces Diffusion Maps, the principle of using kernels to capture non-linear relationships for dimensionality reduction is a common theme in manifold learning methods, including kPCA.)

[44] Nadisic, N., Vandaele, A., Cohen, J. E., & Gillis, N. (2020). *Sparse Separable Nonnegative Matrix Factorization* (arXiv:2006.07553). arXiv.

[45] (PDF) Using Network-Based Machine Learning to Predict Transcription Factors Involved in Drought Resistance. (n.d.). *ResearchGate*.

[46] *Independent Component Analysis—An overview | ScienceDirect Topics*. (n.d.). Retrieved June 30, 2025.

[47] Lähnemann, D., et al. (2020). Eleven grand challenges in single-cell data science. *Genome Biology*, 21(1), 31. (This provides a broad overview of challenges and methods in scRNA-seq analysis, including common DR techniques).

[48] Jolliffe, I. T., & Cadima, J. (2016). Principal component analysis: A review and recent developments. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and*

Engineering Sciences, 374(2065), 20150202. (A classic review of PCA properties).

[49] Coifman, R. R., & Lafon, S. (2006). Diffusion maps. *Applied and Computational Harmonic Analysis*, 21(1), 5-34. (Introduces a non-linear DR method, indirectly highlighting limitations of linear methods for complex data).

[50] Moon, K. R., et al. (2018). Manifold learning for single-cell data. *Nature Biotechnology*, 37(12), 1461-1473. (Discusses how manifold learning, including non-linear DR, is necessary for complex biological trajectories).

[51] Van der Maaten, L., & Hinton, G. (2008). Visualizing Data using t-SNE. *Journal of Machine Learning Research*, 9(Nov), 2579-2605. (The seminal paper on t-SNE).

[52] Wattenberg, M., et al. (2016). How to use t-SNE effectively. *Distill*, 1(10), e5. (Discusses the practical aspects and limitations of t-SNE, including hyperparameter effects and global structure distortion).

[53] Zheng, Y., et al. (2022). Comprehensive comparison of dimensionality reduction methods for single-cell RNA sequencing data visualization. *Briefings in Bioinformatics*, 23(1), bbab440. (Compares various DR methods for scRNA-seq, including reproducibility aspects of t-SNE).

[54] Lin, Y., et al. (2023). Dissecting and improving gene regulatory network inference using single-cell transcriptome data. *Genome Research*, 33(9), 1546-1559. (This paper specifically discusses challenges and opportunities for GRN inference from scRNA-seq).

[55] Zhu, X., et al. (2017). Detecting heterogeneity in single-cell RNA-Seq data by non-negative matrix factorization. *PeerJ*, 5, e2850. (Highlights NMF's utility for scRNA-seq, addressing high-dimensionality and sparsity).

[56] Elyanow, L., et al. (2021). Fast and interpretable non-negative matrix factorization for atlas-scale single cell data. *bioRxiv*, 2021.09.01.458620. (Emphasizes interpretability and suitability for large, sparse datasets).

[57] Alexandrov, L. B., et al. (2013). Deciphering signatures of mutational processes operative in human cancer using nonnegative matrix factorization. *Cell Reports*, 3(1), 246-259. (Though on mutational signatures, it beautifully illustrates how NMF can extract interpretable "latent features" representing biological processes).

[58] Aerts, S., et al. (2017). SCENIC: single-cell regulatory network inference and clustering. *Nature Methods*, 14(11), 1083-1086. (A prominent example of a tool leveraging NMF-like principles for GRN inference and identifying gene modules/cell states).