Laboratory of biological data mining

Giacomo Fantoni
Telegram: @GiacomoFantoni
Github: https://github.com/giacThePhantom/LaboratoryOfBiologicalDataMining

November 24, 2021

Contents

Chapter 1

Introduction

Data mining and data analysis are not the same thing: the former deals with data that pre-exists with respect to the research question, whereas the latter deals with data that is generated and collected in order to answer some research question.

1.1 Type of data

Data is canonically represented via tables. It implies the existence of a mapping between facts of the world and symbols. The measurement is a mapping between facts in the world and elements of sets equipped with some mathematical structure.

1.1.1 Categorical data

In categorical data the set is finite and it has no structure. The only operation that can be done on it is to distinguish the elements of the set.

1.1.2 Ordered categorical data

In ordered categorical data the set has an order: its element are in a mathematical relationship with the properties of an order:

- Reflexivity.
- Antisymmetry.
- Transitivity.

1.1.3 Discrete data

In discrete data the set is a subset of the natural numbers. Operations permitted are sum and difference.

1.1.4 Continuous data

In continuous data the set is an interval of the real numbers. Some scales have an absolute zero. The operations permitted are sum and difference. Division makes sense only if the scale is absolute.

1.2 Metadata

Metadata is data about the data. For example it contains informations about its origin, the method, time, format, source and owner.

1.3 Interventional data and observational data

Interventional data and observational data are distinguished on the basis of the intervention on the system. The former is generated by experimental procedures and the latter by pure observation/

Chapter 2

PC algorithm

2.1 Estimating high-dimensional directed acyclic graphs with the PC-algorithm

2.1.1 Introduction

Graphical models are a popular probabilistic tool to analyse and visualize conditional independence relationships between random variables. The major building blocks of these models are nodes (the random variables) and edges (conditional dependence). The structure of conditional independence among the random variables can be explored using the Markov properties. The estimation of a DAG from data is difficult due to the enormous size of the space of DAGs. The PC-algorithm is an alternative to greedy or structurally restricted approaches. It starts from a complete, undirected graph and deletes recursively edges based on conditional independence decisions. This yields an undirected graph that can be partially directed and further extended to represent the underlying DAG. This algorithm runs in the worst case in exponential time, but if the true underlying DAG is sparse, it is reduced to a polynomial runtime. Here there is a focus on estimating the equivalence class and the skeleton of DAGs corresopnding to multivariate Gaussian distributions in high-dimensional context, or where the number of nodes p may be much larger than the sample size n.

2.1.2 Finding the equivalence class of a DAG

Let $G = (V, E)$ a graph. The set of vertices V corresponds to the components of a random vector $\vec{X} \in \mathbb{R}^p$. A probability distribution P on $\mathbb{R}^p$ is said to be faithful with respect to G if conditional independences of the distribution can be inferred from d-separation in G. Consider a random vector $\vec{X} \sim P$. Faithfulness of P with respect to G means that for any $i, j \in V$ with $i \neq j$ and any set $s \subseteq V$:

$$\vec{X}^{(i)} \wedge \vec{X}^{(j)} \text{ are conditionally independent given } \{\vec{X}^{(r)} | r \in s\} \Leftrightarrow i \wedge y \text{ are d-separated by } s$$

Faithfulness is ruling out some classes of probability distributions. The skeleton of a DAG is the undirected graph obtained from G by substituting undirected edges for directed ones. A v-structure in a DAG is an ordered triple of node such that $(i, j) \in G \wedge (k, j) \in G \wedge (i, k) \notin G$. Two DAGs are equivalent is and only is they have the same skeleton and v-structures. If P is faithful with

respect to a DAG G there is an edge between node i and j in the skeleton of DAG G is and only if $\forall s \subseteq V \backslash \{i,j\}, \vec{X}^{(i)} \wedge \vec{X}^{(j)}$ are conditionally dependent given $\{\vec{X}^{(r)}, r \in s\}$. If P is faithful with respect to a DAG G the skeleton of the DAG is a subset to the conditional independence graph corresponding to P. Every edge in the skeleton indicates some strong dependence which cannot be explained by accounting for other variables.

2.1.2.1 PC-algorithm for finding the skeleton

The PC-algorithm betters from a naive strategies that checks for conditional independences given all subsets. For the variance PC-stable of this algorithm the deletion of the arc is postponed to the change of l so the result does not depend on the order of the variables and it is parallelizable.

2.1.2.1.1 Population version In the population version of the PC-algorithm perfect knowledge about all necessary conditional independence relations is assumed available.

: $PC_{pop}(V)$

l = -1
C = $\tilde{C}$
repeat
  l += 1
  repeat
    Select a new ordered pair of nodes i, j that are adjacent in C such that $|adj(C,i)\backslash\{j\}|gel$
    repeat
      Choose new $k \subseteq adj(C,i)\backslash\{j\}$ with $|k| = l$
      if $i \wedge j$ *are conditionally independent given* k **then**
        Delete edge i, j
        Denote this new graph by C
        Save k in $S(i,j)$ and $S(j,i)$
    until *edge* i, j *is deleted or all* $k \subseteq adj(C,i)\backslash\{j\}$ *with* $|k| = l$ *have been chosen*
  until *all ordered pairs of adjacent variables* i *and* j *such that* $|adj(C,i)\backslash\{j\} \geq l$ *and* $k \subseteq adj(C,i)\backslash\{j\}$ *with* $|k| = l$ *have been tested for conditional independence*
until *for each ordered pair of adjacend nodes* $i, j : |adj(C,i)\backslash\{j\}| < l$
return Estimated skeleton C, separation sets S

The maximal value of l is denoted m_{reach} and depends on the underlying distribution. Considering a DAG G and assume that the distribution P is faithful to G. Denote the maximal number of neighbours by $q = \max_{1 \leq j \leq p} |adj(G,j)|$. Then the PC_{pop} algorithm construct the true skeleton of the DAG. Moreover $m_{reach} \in \{q-1, q\}$.

2.1.2.1.2 Sample version For finite sample there is a need to estimate conditional independencies. Assuming faithful models (conditional independence relations correspond to d-separations) in the Gaussian case conditional independences can be inferred from partial correlations. Assume that distribution P of the random vector $\vec{X}$ is a multivariate normal. For $i \neq \in \{1, \ldots, p\}, k \subseteq \{1, \ldots, p\} \backslash \{i,j\}$, denote $\rho_{i,j|k}$ the partial correlation between $\vec{X}^{(i)}$ and $\vec{X}^{(j)}$ given $\{\vec{X}^{(r)}, r \in k\}$ then $\rho_{i,j|k} = 0$ if and only if $\vec{X}^{(i)}$ and $\vec{X}^{(j)}$ are conditionally independent given $\{\vec{X}^{(r)}, r \in k\}$. The partial correlations can be estimated and the sample partial correlation $\hat{\rho}_{i,j|k}$ can be calculated, for some $h \in k$:

$$\rho_{i,j|k} = \frac{\rho_{i,j|k\setminus h} - \rho_{i,h|k\setminus h}\rho_{j,h|k\setminus h}}{\sqrt{(1-\rho^2_{i,h|k\setminus h})(1-\rho^2_{j,h|k\setminus h})}}$$

For testing whether a partial correlation is zero or not Fisher's z-transform is applied:

$$Z(i,j|k) = \frac{1}{2}\log(\frac{1=\hat{\rho}_{i,j|k}}{1-\hat{\rho}_{i,j|k}})$$

The null hypothesis is rejected $H_o(i,j|k) : \rho_{i,j|k} = 0$ against the two sided alternative $H_A(i,j|K) : \rho_{i,j|k} \neq 0$ if $\sqrt{n-|k|-3}Z(i,j|k) > \Phi^{-1}(1-\frac{\alpha}{2})$, where Φ denotes the cdf of $\mathcal{N}(0,1)$. The sample version of the PC-algorithm is the same of the population version, with the if statement replaced by $\sqrt{n-|k|-3}Z(i,j|k) > \Phi^{-1}(1-\frac{\alpha}{2})$. This algorithm yields a data-dependent $\hat{m}_{reach,n}$. The only tuning parameter is α, the significance level for testing partial correlations. This algorithm is asymptotically consistent even if p is much larger than n but the DAG is sparse. For the genehome project, linear correlations between genes are computed using the Pearson coefficient:

$$r = \frac{\sum_{i=1}^{n}(X_i - \bar{X})(Y_i - \bar{Y})}{\sqrt{\sum_{i=1}^{n}(X_i - \bar{X})^2}\sqrt{\sum_{i=1}^{n}(Y_i - \bar{Y})^2}}$$

2.1.2.2 Extending the skeleton to the equivalence class

While finding the skeleton the separations sets that made edges drop out were recorded by S. This is essential for extending the skeleton to the equivalence class.

: `Extending_to_CPDAG(`G_{skel}, S`)`

for all *pairs of non adjacent* i, j *with common neighbour* k **do**
  if $k \notin S(i,j)$ **then**
    Replace $i - k - j \in G_{skel}$ with $i \to k \leftarrow j$

In the resulting PDAG try to orient as many undirected edges as possible by repeated application of:

R1 orient $j - k$ into $j \to k$ whenever there is $i \to j$ such that i and k are non-adjacent.

R2 orient $i - j$ into $i \to j$ whenever there is a chain $i \to k \to j$.

R3 orient $i - j$ into $i \to j$ whenever there are two chains $i - k \to j$ and $i - l \to k$ such that k and l are non-adjacent.

R4 orient $i - j$ into $i \to j$ whenever there are two chains $i - k \to l$ and $k \to l \to j$ such that k and l are non-adjacent

return CPDAG G

2.2 A computing system for discovering causal relationships among human genes to improve drug repositioning

The genehome project aims to expand gene networks using transcriptomic datasets. For human data the objective is to provide a public resource to navigate and combine results by expanding each single human transcript. The platform hosted the NES^2RA algorithm. Starting from a local gene

network LGN based on previous biological knowledge, its expansion consists in a set of genes and a list of interactions which describe putative causal relationships with the genes in the LGN. The expansion is calculated on observational gene expression data organized in a coherent normalized data matrix. To overcome the problem of unique elaborations OneGenE has been developed. The list of gene expansions is calculated for each gene in an organism by systematically running single-gene NES^2RA expansions with fixed parameters and then combine them afterwards to simulate LGN expansions. The expansions of the gene networks is based on the transcriptomic dataset provided by the FANTOM project. Their data comes from sequencing of RNA extracted from different samples of human tissues and cell lines and contains expression profiles of 201802 gene isoforms. Drug repositioning is an alternative approach for the discovery of new therapeutic opportunities for already approved medicines. This method, which relies on previous knowledge, speeds up the approval procedure of the drug regulators and can represent a valuable approach.

2.2.1 Method

OneGenE is a method to compute ranked candidate gene lists that expands known local gene networks given gene expression data. It is based on the systematic and iterative application of the skeleton function of the PC algorithm on subsets of the input data. Candidate expansion lists are pre-computed for each gene of the target organism. Secondly a set of transcript of interest LGN is provided as input and the intermediate results are aggregated.

```
: Pre-computation-step(I, D, A, S, E)
L = ∅
foreach g ∈ S do
    foreach θ = (α, d, i) ∈ A × D × I do
        %NES²RA ranking procedure call
        %l_{g,θ} = RP(S, g, E, 1, i, d, α, or:
        foreach j ≤ i do
            Randomly generate a minimal collection of subsets of dimension d of S such that
             g is in every subset and each transcript is in at least one subset
        foreach subset do
            Run the PC-skeleton on the expression data E restricted to the transcripts of the
             subset and generate a network
        foreach γ adjacent to g in the network do
            %Compute absolute frequency
            f_γ = numbers of networks where γ and g are adjacent
            %Compute relative frequency
            f'_γ = f_γ / (numbers of subsets that contains γ)
        l_{g,θ} = genes ordered with respect to f'_γ
        L = L ∪ l_{g,θ}
return L
```

2.2.1.1 Data and input

The algorithm starts with an $n \times m$ gene expression data matrix E, where $n = |S|$ is the number of transcripts S and m is the number of samples and a set of parametertuples $\Theta = \{\theta\} = \{(\alpha, d, i) | \alpha \in A, d \in D, i \in I\}$ where A, D and I are the sets of alpha values, tile sizes and the number of iterations.

2.2.1.2 Nested loop

For each transcript g in S, p instances of PC-IM are executed, where $p = |\Theta| = |A||D||I|$. The internal loop receive as input a single gene g with probability vbector $\Pi = 1$. The internal loop comprises the subsetting of the transcripts, the run of PC-skeleton on each subset and the computation of absolute frequencies, relative frequencies and the corresponding order of the candidates list.6

2.2.1.3 Pre-computation result

In OneGenE the ranking aggregation is postponed. Each PC-IM returns a candidate expansion list $l_{g,\theta}$ for each tuple of parameters corresponding to the set of lists resulting from the algorithm. The candidate lists are stored and the data is ready to be queried.

2.2.1.4 Ranking or list aggregation

Let S_{LGN} be the set of transcripts in an input LGN and $l_{g,\theta}$ the candidate expansion list of g with parameters θ. The final candidate expansion list is obtained combining $\mathcal{L} = \{l_{g,\theta} | g \in S_{LGN}, \theta \in \Theta\}$ by means of a ranking aggregator. High relative frequency provides evidence of a putative direct causal relationship. Ranks are instead useful for comparing list and prioritization.

```
: List_Aggregation_Complete(f'_min, K, L)
while true do
    L_temp = ∅
    User selects T set of transcript
    foreach l_{t,θ} ∈ L such that t ∈ T do
        L_temp = L_temp ∪ {l_{t,θ}}
    Case frequency: L_temp =fre(L_temp, f'_min)
    Case top k: L_temp = top(L_temp, K)
    Case top variable k_t: L_temp = top(L_temp, K)
    return List_Aggregation(L_temp)
```

2.2.1.5 Data and running paramters

The human transcriptome data have been obtained from the FANTOM5 project. It was generated using single molecule CAGE or cap analysis gene expressione. Normalized expression values are extimated as transcripts per milion TPM. The data has been filtered removing unknown transcripts and with absent or low expression values among almost all samples. The single gene NES^2RA expansion where submitted with a tile size of 1000, 1000 iterations and $\alpha = 0.05$.

2.2.2 Validation

To evaluate the biological pertinence of the single-gene NES^2RA expansion obtained by OneGenE where used gene expansions involving genes of medical relevance for neural motor dideases and hematopoietic tumors. For each expansion, the top scoring 250 transcripts where considered. OneGenE expansions where benchmarked against a simple Pearson correlation analysis. Starting from the same seed transcript, the top 250 correlated transcripts were considered and compared to the 250 transcripts identified by OneGenE. To quantify the overlap among the expansions obtained from the same transcript the Jaccard index was calculated: the number of items shared between the two sets divided by the total number of items in both sets. OneGenE expansions are largely pppulated

by distinct genes with respect to the correlation approach. To evaluate the biological pertinence, a functional enrichment was performed and the single-gene expansions consistently achieves higher biological pertinence than the correlation approach using the fraction of pertinent enrichments. Lastly using known protein-protein interaction in STRING, for each of the expansion the list of gene was compared with the list of interactions in STRING. Based on the overlap between the genes and annotated interactions odds ratio values were calculated with outstanding results.

2.2.3 Prostate cancer

22 genes expanded as single-gene by NES^2RA on two transcriptomic datasets. First analyse direct interactions within input genes and represent as graphs. Focusing on two networks and aggregating the expansion list og the genes belonging to it. A comparison with STRING and functional enrichment analyses allowed to understand nature and composition of those networks. After filtering genes alredy known to be related to prostate cancer, query against Gene Drug interaction database identified novel targets for this disease that can support drug repositioning. The functional relationship between the input genes obtained is studied by their mutual interaction: the pair (x, y) of input genes is defined as the presence of gene x into the expansion list of y and viceversa. After a comparison with STRING functional involvement of the networks and the gene composition enrichment analyses with KEGG pathways and gene anthology biological process categories. Finally after filtering the networks for genes already known DGIbd database of gene-drug interaction was queried and retrieved some drugs involved. After that target that may be cytotoxic where removed.

2.2.4 Coronary artery disease

The OneGenE approach was used to identify novel putative drug target for coronary artery disease CAD, considering genes genetically associated with it. The list of those genes has been obtained from open targets. In order to retain the genes most related the expansions list where trimmed containing only the first $N_l = \max\{5, 1 - R_l + 1\}$, where R_l is the rank based on genetic association derived from open targets. The lists of isoforms were aggregated summing the relative frequencies computed by NES^2RA. Then the list of isoforms was ranked and converted into a ranked list of genes. ToppGene was used to perform enrichment analysis against the biological processes of the gene ontology. To quantify the overlap between the ranked list og genes genetically associated with CAD and the ranked expansion lists obtained from NES^2RA, they used the weighted jaccard similarity: $WJS(\rho, \sigma) = \frac{\sum_{i=1}^{n} \min(\rho_i, \sigma_i)}{\sum_{i=1}^{n} \max(\rho_i, \sigma_i)}$, where ρ_i and σ_i are the weights corresponding to the same item i, the weight of a feature i in a ranked list ρ is computed as $len(\rho) - rank(i) + 1$. These scores depend on the length of the list, so they are not comparable. To make them comparable a permutation approach was used to estimate a set of score distributions. For each length 2000 random list of genes were generated and the WJS was computed and the score distribution associated to that length generated. Then these distributions where used to compute empirical p-values for multiple hypothesis testing.

2.3 NESRA

Before genehome the scientific pipeline consisted of:

- Organism choice.
- Finding a suitable gene expression dataset.
- Dataset filtering and imputing.
- Choose target LGN.
- PC-IM parameters test.

NESRA is a network expansion by variable subsetting and ranking aggregation. It systematically and iteratively apply subsetting varying parameters and the list is then aggregated.

```
: NESRA(I, D, A, k)
%Where I os the set of values of number of iterations, D is the set of
  values of the subset dimension, A the set of values of the significance
  level and k the maximum lenght of the list
L = ∅
foreach α ∈ A do
    foreach d ∈ D do
        foreach i ∈ I do
            L = L∪Ranking_Procedure(S, S_LGN, E, i, d, α)
L = Top(L, k)
return Ranking_Aggregation(L)
```

Its successor NES^2RA models, using a probability vector, the confidence of the presence of the genes belonging to the local gene network. NES^{2}RA or network expansion by stratified subsetting and ranking aggregation tries to solve the problem of finding candidates for gene expansion. It alllows to model with a probability the presence of the genes of the network to be expanded in the subset and the sampling is stratified. It is based on the PC-algorithm.

2.3.1 Gene network expansion

Given a set S of gene transcripts whose level of expression has been measured p times in different conditions, such that for reach $s_i \in S$ there is a vector $x_i \in \mathbb{R}^p$ of exression levels. Let us assume that there exists a generally unknown ground truth direct graph $G = (S, B)$ wich represents the real causal relationships between the gene transcripts. The discovery of candidate genes for GNE is defined as follows. Given a graph $G = (N, B)$ where $N \subseteq S$ and $B \subseteq N \times N$, find a ranked list of elements of $S \backslash N$ such that the elements of the list are connected or very near to the elements of N in G.

2.3.2 NES^2RA

This algorithm as input data the LGN and the probability of each gene of the LGN to be included in the subsets, the set of parameters to be used and the gene expression levels for the considered organisms. The inclusion of the LGN in the subsetting step improves the quality of the result because the composition of each subset is influenced by the LGN nodes added. The vector of probability is the representation of the knowledge of the user about the presence of specific genes in the network. Depending on the probabilities genes will be included in the data for the run of the PC-algorithm.

2.3.2.1 Ranking procedure

The ranking procedure contains three steps which create the subsets, execute several calls of the skeleton of the PC-algorithm and compute the transcripts frequency that defines the order of each

ranking. The RP takes as parameters the number of iterations, de dimension of the subset, the significance level for the skeleton and the probability vector. The output depends on the order of the inputs, mitigated by the different iterations. This procedure returns a ranked list of k elements.

: `NESRA-RP`(S, S_{LGN}, Π, i, d, α)

foreach $g \in S$ **do**
  $p_g = 0$
  $f_g = 0$
%`Subsets creation`
foreach $j, 1 \leq j \leq i$ **do**
  h = l
  $S_{temp} = S \backslash S_{LGN}$
  while $S_{temp} \neq \emptyset$ **do**
    foreach $g_i \in S_{LGN}$ **do**
      With probability π_i: $T_{h,j} = T_{h,j} \cup \{g_i\}$
    $S_{temp} = S_{temp} \cup (S_{LGN} \backslash T_{h,j})$
    while $|T_{h,j}| < d$ **do**
      Uniformely random select $g \in S_{temp}$
      $T_{h,j} = T_{h,j} \cup \{g\}$
      $S_{temp} = S_{temp} \backslash \{g\}$
      $p_g = p_g + 1$
    if $S_{temp} = \emptyset \wedge |T_{h,j}| < d$ **then**
      while $T_{h,j} < t$ **do**
        Uniformely random select $g \in S \backslash T_{h,j}$
        $T_{h,j} = T_{h,j} \cup \{g\}$
        $p_g = p_g + 1$
    h += 1
  $N_j = h$
%`Skeleton`
foreach $j, 1 \leq j \leq i$ **do**
  foreach $h, 1 \leq h \leq N_j$ **do**
    $R_{h,j} = skeleton(T_{h,j}, E, \alpha)$
%`Frequency computation`
foreach $g \in S$ **do**
  foreach $q \in S_{LGN}$ **do**
    foreach $j, 1 \leq j \leq i$ **do**
      foreach $h, 1 \leq h \leq N_j$ **do**
        if $g \in Adj_{R_{h,j}}(q)$ **then**
          $i = I \cup \{g\}$
          $f_g = f_g + 1$
  $f'_g = \frac{f_g}{p_g \cdot |S_{LGN}|}$
return I ordered with respect to f'_g

2.3.2.2 Subsetting

A subsetting operation is done systematically and iteratively on the whole data set in order to randomly select genes that will be processed by the skeleton PC. It is controlled by iteration and subset size parameters. The subsetting is stratified and genes of the LGN can have increased probability of beein in the subsets. For a given pair of subset a first selection controlled by the probability vector, specifies which genes of the LGN are present in it. The one not selected in the first are considered in the second with uniform probability and then there is a third selection restricted to genes not already present. The probability vector is such that the ith component π_i modulates the probability of the precence of the ith gene g_i in the subsets.

2.3.2.3 Aggregation

For each pair subset size and iteration NES^2RA executes a number of skeleton procedures. This yields list of genes, which are combined ranked by their number of appearances. The list of candidate genes is produced applying different ranking aggregation methods on the ranked lists using the number of appearances, Borda Count and MC4 heuristic. The number of appearances counts how many rankings a certain genes has. The Borda Count consists of constructing a matrix $A(m, n)$ with m rows and n columns, corresponding to the genes and the rankings. The element a_{ij} is the rank of gene i on ranking j and a statistic for each gene is computed on the rows of the matrix. The two statistics are the BC-mean and BC-min. The MC4 heuristic is an aggregator based on Markov chains. It computes the transition matrix of the pairwise comparison of all the rankings for each gene. A step in the Markov chain assigns a higher probability to a gene q is $rank(q) < rank(p)$ for a majority of the lists that ranked both. The steady state assigns higher probability to the genes with higher ranks.

2.3.2.3.1 OneGenE In OneGenE this step is postponed when there is a query. Let S_{LGN} be the set o transcript in an input LGN and $l_{g,\theta}$ the candidate expansion list of the gene g with parameter tuple θ. The final candidate gene expansion list is obtained combining the set of partial result: $\mathcal{L} = \{l_{g,\theta} | g \in S_{LGN}, \theta \in \Theta\}$ by means of a ranking aggregator. Given the set of ranked list the ranking aggregation problem is combining the ranking in it in order to obtain a better final ordering l^*.

2.3.2.3.1.1 Borda Count It associates to each element u in a list l a Borda score $B_l(u)$. The final rank of the element u will be computed using an aggregation function: $f(B_1(u), \ldots, B_n(u))$. Common aggregation functions are the arithmetic mean or the median.

2.3.2.3.1.2 RnakAggreg Package This algorithm uses local search trying to find $l^* = \arg\min_l \left(\sum_{i=1}^{|\mathcal{L}|} w_i d(l, l_i) \right)$ Where w_i is the weight associated with list l_i and d is the Kendall Tau or Spearmann distance.

2.3.2.3.1.3 RobustRankAggreg package It is an aggregator based on order statistics. The assumption is that the rank of each element u in lists in S is sampled from a distribution and, once it is known, it can be compared with a null distribution obtained by sampling the ranking uniformly. The aggregator associates at each element a corrected p-value and the final ranking is obtained sorting the elements by p-value.

2.3.2.3.1.4 Markov chain method Markov chain 4 method builds an ergodic Markov chain where each state represents a ranked element using the ranking in $\mathcal{L}$. It is build starting from u and an element v is sampled uniformly. If v is ranked lower for a majority of the lists where both elements are present, it associates a probability of staying in u. The final rank is obtained computing the stationary distribution of the chain and sorting the element by their stationary probability.

Chapter 3

Autoencoders

3.1 Introduction

An autoencoder is a specific type of a neural network, which is mainly designed to encode the input into a compressed an meaningful representation, and then decode it back such that the reconstructed input is as similar as possible to the original one. Their main purpose is learning in an unsupervised manner an informative representation of the data that can be used for various implications. The problem is to learn the functions encoder $A : \mathbb{R}^n \to \mathbb{R}^p$ and decoder $B : \mathbb{R}^p \to \mathbb{R}^n$ such that:

$$\arg\min_{A,B} \mathbb{E}[\Delta(\vec{x}, B \circ A(\vec{X}))]$$

Where $\mathbb{E}$ is the expectation over the distribution of $\vec{x}$ and Δ is the reconstruction loss function, which measures the distance between the output of the decoder and the input, typically the l_2 norm. Typically A and B are neural networks or linear operations (linear autoencoder). An autoencoder is a generalization of PCA, where instead of finding a low dimensional hyperplane where the data lies, it is able to learn a non-linear manifold. Autoencoders can be trained end-to-end or layer-by-layer. In the latter case they are stacked together to a deep encoder.

3.1.1 Regularized autoencoders

Regularization is required since one may get the identity operator for A and B if it is not used. The most common option is to use a bottleneck, making the dimension of the representation smaller than the input. This directly creates a low dimensional representation of the data. It must still be possible to have overfitting if the capacity of the encoder and the decoder is large enough to encode each sample to an index. There are different possibility other than introducing a bottleneck. An important trade-off in autoencoders is the bias-variance trade-off: there is a need for an architecture able to reconstruct the input well. Such low-dimension representation should generalize to a meaningful one.

3.1.1.1 Sparse autencoders

To deal with this trade-off one can enforce sparsity on the hidden activations. This can be added on top of the bottleneck. To enforce sparsity regularization one can use ordinary regularization applied to the activations instead of the weights.

3.1.1.1.1 L_1 regularization L_1 regularization introduce sparsity, so the autoencoder optimization objective becomes:

$$\arg\min_{A,B} \mathbb{E}[\Delta(\vec{x}, B \circ A(\vec{x})] + \lambda \sum_i |a_i|$$

Where a_i is the activation at the ith hidden layer and i iterates over all of them.

3.1.1.1.2 KL-divergence Another way to enforce sparsity is to use the KL-divergence, a measure of the distance between two probability distributions. Instead of tweaking λ, one can assume that the activation of each neuron acts as a Bernoulli variable with probability $\hat{p}_j = \frac{1}{m} \sum_i a_i(x)$, where i iterates over the samples in the batch. The overall loss function would be:

$$\arg\min_{A,B} \mathbb{E}[\Delta(\vec{x}, B \circ A(\vec{x})] + \sum_j KL(p||\hat{p}_j)$$

Where the regularization term aims at matching p to $\hat{p}$.

3.1.1.2 Denoising autoencoders

Denoising autoencoders can be considered as a regularization option or as robust autoencoders which can be used for error correction. The input is distributed by some noise and the autoencoder is expected to recunstruct the clean version of the input. Common options for C, the function that introduces with $\bar{x}$ a random variable are:

$$C_\sigma(\bar{x}|\vec{x}) = N(\vec{x}, \sigma^2 I)$$

Where σ sets the impact of the noise.

$$C_p(\bar{x}|\vec{x}) = \beta \odot \vec{x}, \beta \sim Ber(p)$$

Where $\odot$ is an element-wise (Hadamard) product. Also p sets the probability of a value in $\vec{x}$ not being nullified.

3.1.1.3 Contractive autoencoders

In contractive autoencoders the objective is to make the feature extraction less sensitive to small perturbations forcing the encoder to disregard changes in the input that are not important for the reconstruction by the decoder. To do so a penalty is imposed on the Jacobian of the network, trying to minimize its L_2 norm:

$$\arg\min_{A,B} \mathbb{E}[\Delta(\vec{x}, B \circ A(\vec{x})] + \lambda ||J_A(\vec{x}||_2^2$$

The reconstruction loss function and the regularization loss pull the result towards opposite directions. By minimizing the squared Jacobian norm, all the latent representation of the input tend to be more similar to each other making the reconstruction more difficult. The idea is that those variations that are not important for the reconstruction would be diminished by the regularization factor, while important variations would remain because of their impact on the reconstruction error.

3.1.2 Variational autoencoders

Variational autoencoders VAE follows from variational bayes inference. They aer generative models that attempt to descrive data generation through a probabilistic distribution. Given an observed dataset $X = \{x_i\}_{i=1}^N$ of V samples, a generative model for each datum x_i conditioned on an unobserved random latent variable z_i is assumed, where θ are the parameters governing the distribution. This is equivalent to a probabilistic decoder. Symmetrically an approximate posterior distribution is assumed over z_i given x_i, a probabilistic decoder governed by ϕ. A prior distribution is assumed for z_i: $p_\theta(z_i)$. θ and ϕ need to be learned from data. z_i can be interpreted as a code given by the recognition model $q_\phi(z|x)/$ The marginal log-likelihood is expressed as sum over the individual data points $\log p_\theta(x_1, \ldots, x_N) = \sum_{i=1}^{N} \log p_\theta(x_i)$ and each point can be rewritten:

$$\log p_\theta(x_i) = D_{KL}(q_\phi(z|x_i)||p_\theta(z|x_i)) + \mathcal{L}(\theta, \phi, x_i)$$

Where the first term is the Kullback-leibler divergence of the approximate recognition model from the true posterior and the second is the variational lower bound on the marginal likelihood:

$$\mathcal{L}(\theta, \phi, x_i) = \mathbb{E}_{q_\phi(z|x_i)}[-\log q_\phi(z|x) + \log p_\theta(x, z)]$$

Maximizing this lower bound improves approximation of the posterior and can be expanded as:

$$\mathcal{L}(\theta, \phi, x_i) = -D_{KL}(q_\phi(z|x_i)||p_\theta(z)) + \mathbb{E}_{q_\phi(z|x_i)}[\log p_\theta(x_i|z)]$$

Variational inference follows by maximizing this lower bound for all data points with respect to θ and ϕ. Given a dataset $X = \{x_i\}_{i=1}^N$ with N data points, the marginal likelihood lower bound of the full dataset can be estimated using a mini-batch of size M:

$$\mathcal{L}(\theta, \phi, x_i) \approx \tilde{\mathcal{L}}^M(\theta, \phi, X^M) = \frac{N}{M} \sum_{i=1}^{M} \mathcal{L}(\theta, \phi, x_i)$$

With its gradients approximated using a reparameterization trick and stochastic gradient optimization.

3.1.2.1 The reparameterization trick

The random variable $\tilde{z} \sim q_\phi(z|x)$ can be reparameterized using a differentiable transformation $g_\phi(\epsilon, x)$ using auxiliary noise ϵ drawn from some distribution:

$$\mathcal{L}(\theta, \phi, x_i) \approx \tilde{\mathcal{L}}(\theta, \phi, x_i) = \frac{1}{L} \sum_{l=1}^{L} \log p_\theta(x_i, z_{(i,l)}) - \log q_\phi(z_{(i,l)}|x_i)$$

Where $z_{(i,l)} = g_\phi(\epsilon_{(i,l)}, x_i)$ with $\epsilon_{(i,l)}$ is random noise. To optimized mini-batch estimates we use the differentiable with respect to θ and ϕ equation:

$$\hat{\mathcal{L}}^M(\theta, \phi, X) = \frac{N}{M} \sum_{i=1}^{M} \tilde{\mathcal{L}}(\theta, \phi, x_i)$$

L can be set to 1 if M is large enough. Instead of taking the gradient of a single randomized latent representation, the gradients of the generative network are learned by a weighted average of the sample over different samples from its posterior distribution. The weights follows the likelihood functions $q_\phi(z_{(j,l)}|x_i)$.

3.1.2.2 Disentangled autoencoders

The variational lower bound can be viewed as the summation of the reconstruction capability of the samples and the regularization that biases $q_\phi(z|x^{(i)})$ towards the assumed prior $p_\theta(z)$. Disentangled autoencoders introduce another parameter β that is a multiplicative factor for the KL divergence:

$$\mathcal{L}(\theta, \phi, x_i) = -\beta D_{KL}(q_\phi(z|x_i)||p_\theta(z)) + \mathbb{E}_{q_\phi(z|x_i)}[\log p_\theta(x_i|z)]$$

The prior $p_\theta(z)$ is set as the standard multivariate normal distribution. All the features are uncorrelated and the KL divergence regularize the latent features distribution $q_\phi(z|x^{(i)}$ to a less correlated one. The larger the β the less correlated the features will be.

3.1.3 Applications of autoencoders

3.1.3.1 Generative model

Once the autoencoder is trained, to obtain new data from it one can simply sample random variables from the same prior $p_\theta(z)$ and feed it to the decoder, which would generate a new meaningful sample.

3.1.3.2 Classification

Once the autoencoder is trained in an unsupervised manner one can use its decoder to do feature extraction for classification. The key assumption is that samples with the same label should correspond to some latent presentation. Another approach is to use them as a regularization technique.

3.1.3.3 Clustering

Once the autoencoder is trained in an unsupervised manner one can use its decoder to obtain a low-dimensional representation used to solve the dimensionality problem of clustering. Also during clustering the embeddings aer retrained at each iteration, adding an argument to the autoencoder loss function, penalizing the distance between the embedding and the cluster centre.

3.1.3.4 Anomaly detection

A trained autoencoder would learn the latent subspace of normal samples. Once trained it would result with a low reconstruction error for normal samples and high for anomalies.

3.1.3.5 Dimensionality reduction

The dimensionality reduction is performed by every autoencoder in the bottleneck layer. The nonliearity activation functions often allows for a superior reconstruction when compared to simple PCA.

3.1.4 Advanced autoencoder techniques

Autoencoders' strength is their ability to use their latent representation for different usages, but their reconstructions are usually blurry. The reason for that is the used loss function.

3.1.4.1 Autoencoders and GANs

GANs create results visually compelling but in the cost of the control on the resulting images. So a work has been done to combine autoencoders and GANs. In adversarial autoencoders the KL-divergence in VAE is replaced by a discriminator network that distinguishes between the prior and the approximated posterior. It can be done replacing the reconstruction loss or making the encoder and the discriminator share weights.

3.1.4.1.1 Adversarially learned inference In adversarially learned inference ALI there is an attempt to merge VAEs and GANs. A discriminator is used to distinguish between $(x, \hat{z})$ pairs, where x is an input and $z \sim q(z|x)$ is sampled from the encoders output and $(\tilde{x}, z)$ pairs, where $z \sim p(z)$ sampled from the prior in VAE and $\tilde{x} \sim p(x|z)$ is the decoder's output.

3.1.4.1.2 Wasserstein autoencoders Wasserstein-GAN WGAN solve a lot of problems to the learning stability of GANs using the Wasserstein distance for the optimizations loss function. It is the distance between two probabilities:

$$W_c(P_X, P_G) = \inf_{\Gamma \in P(X \sim P_x, Y \sim P_G)} \mathbb{E}_{(X,Y)\sim\Gamma}[c(X, Y)]$$

Where $c(x, y)$ is some cost function. When $c(x, y) = d^p(x, y)$ is a metric measurement, the p-th root of W_c is called the p-Wasserstein distance. When $c(x, y) = d(x, y)$ acn be defined as:

$$W_1(P_X, P_G) = \sup_{f \in \mathcal{F}} \mathbb{E}_{X \sim P_X}[f(X)] - \mathbb{E}_{Y \sim P_G}[f(Y)]$$

IN Wasserstein autoencoders the loss function is modified and the objective becomes:

$$D_{WAE}(P_X, P_G) = \inf_{Q(Z|X) \in \mathcal{Q}} \mathbb{E}_{P_X} \mathbb{E}_{Q(Z|X)}[c(X, G(Z))] + \lambda \cdot D_Z(Q_Z, P_Z)$$

Where Q is is the encoder and G is the decoder. The left part is the new reconstruction loss, which penalizes on the output distribution and the sample distribution.

3.1.4.2 Deep feature conssitenf variational autoencoder

Given an original image and a reconstructed one, a different measure is used that takes into account the correlation between pixels. A pretrained network can be used for creating a loss function for autoencoders. After encoding and decoding an image, both are inserted as input to a pretrained network. Assuming results with high accuracy, and a not so different domain, eac layer can be seen as a successful feature extractor of the input image. Therefore instead of measuring the difference between two images, it can be measured between their representation, imposing a more realistic different measure for the autoencoder.

3.2 Performance comparison of deep learning autoencoders for cancer subtype detection using multi-omics data

3.2.1 Introduction

Different omics technologies provide the amility to interpret and explain the genome through DNA sequencing, genome expression, protein identification and others. Such individual data is limited

in the inofrmation on the molecular complexity occurring inside the organism. Furthermore, the dimension and diversity of such data make it extremely challenging to perform proper data analysis that can efficiently fuse these data. Multi-omics data integration is the method in which diverse types of omics data are combined as predictor variables. This methods can allow for the identification of crucial genomic factors and biomarker, generate models and understand the genetics and genomics architecture of complex phenotypes. Several data fusion model have been proposed: early, intermediate and late fusion. One example is similarity network fusion where diverse types of data are normalized into a network through a non linear kernel function, next these networks are fused through an iterative fusion algorithm. Also the deep-learning framework of autoencoders exhibited significant potential as data fusion algorithm. The autoencoder generates new non linear features from its original input feature set. They are algorithms for dimensionality reduction and heterogeneous data integration based on feed-forward neural networks. They have a butterfly structure with number of neuron in input equals to the one in output and smaller hidden layers. This design creates a compressed representation of the data while preserving the input data's most important feature. It also allows it to concatenate the features and information of different omics sources. A critical application of such data fusion algorithms is cancer subtype detection using omics data. THe performance of four different autoencoders to integrate and reduce multi-omics data is used in this work. By data fusion the autencoders created new features to represent the input datasets., used to implement a survival-based clustering algorithm to define groups of patients with a similar distribution of features and survival prognosis.

3.2.2 Materials and methods

3.2.2.1 Dataset and prepossessing

From the TGCA database data from glioblastoma multiforme, colon adenocarcinoma, kidney renal clear cell carcinoma and breast invasive carcinoma with data of gene expression, DNA methylation and miRNA expression. From the datasets the most common patients where chosen and each data was scaled according to $x_n = \frac{x_i - x_{min}}{x_{max} - x_{min}}$. The most important features where selected based on maximum variance. These selected features where fed into the autoencoders as input.

3.2.2.2 Autoencoder construction

3.2.2.2.1 Vanilla autoencoder A vanilla autoencoder minimizes:

$$L(x, g(f(x))$$

Due to the non linearity of the encoder and decoder's activation function, the vanilla encoder learns non linear features from the data. A vanilla autoencoders with multiple hidden layers is called a deep vanilla autoencoder. This autoencoder has a high possibility of over-fitting.

3.2.2.2.2 Denoising autoencoder A denoising autencoder reconstructs the original input from a corrupt copy of an input minimizing:

$$L(x, g(f(\tilde{x}))$$

THe corrupt copy is formed introducing noise to the original input and denoising is achieved through stochastic mapping some input values to zero. This helps the autoencoder learn features other than the original features directly from the data.

3.2.2.2.3 Sparse autoencoder A parse autoencoder is a regularized version of vanilla autoencoder with a sparsity penalty $\Omega(h)$ added to the bottleneck layer. The learning minimizes:

$$L(x, g(f(x)) + \Omega(h)$$

This helps to learn the important features of data even when there are meany hidden units in the autoencoder.

3.2.2.2.4 Variational autoencoder A variational autoencoder uses a strung assumption about latent variables, generally using a latent gaussian distribution. It imposes a constraint in the encoder network, forcing the bottleneck layer to follow a Gaussian distibution. It minimizes:

$$L(x, g(f(x)) + L(l)$$

Where $L(l)$ is the latend loss, measured in terms of the Kullback-Leibler divergence of the bottleneck layer to a unit gaussian distribution, quantifying the difference between them. THis generates the latent variable with a generalization of the network.

3.2.2.3 Implementation

For vanilla, denoising and sparse autoencoders, the hidden layers where 500, 100 and 500 nodes and input and output of 1000. They were selected based on the maximum variance of three data types. For the denoising a noise factor of 0.5 was applied. For sparse a L_1 regularization penalty of 0.01 and an L_2 regularization penalty of 0.02 on the nodes to induce sparsity was implemented. Log variance and lambda layer were used to convert the standard deviation for numerical stability when necessary. To optimize the autoencoders the ADAM algorithm was used and for vanilla, sparse and denoising the activation function for the input and hidden layer was the hyperbolic tangent and for the output the sigmoid. For the variational a ReLU on the input and hidden and sigmoid on the output was used. Tho measure the loss vbetween the input and output the means square error function was used for vanilla and denoising, the binary cross-entropy for sparse and the negative log-likelihood for variational.

3.2.2.4 Clustering and subtyping

The reduced number of features obtained from the bottleneck layer was used to apply standard subtyping methods. First the similarity of each patient pair was calculated by euclidian distance and spearman correlation. Then a clustering algorithm was used to cluster similar groups. An unsupervised subtypes discovery method combined with k-means and partitioning around medoids was used for clustering. k-means and PAM were used in a window between 3 and 6 clusters.

3.2.2.5 Evaluation metrics for subtyping

To evaluate the performance of the autoencoders survival analysis was used to evaluate the survival patterns from different subtypes. Next the p-value of the log-rank test was used to identify the difference in Kaplan-Meier survival curves between different subtypes. Low p-value ensure high confidence of different survival times for the different identified subtypes. Also the silhouette width of the clusters was used to benchamrk the performance of clustering. Silhouette scores measure how well a patient is matched to its identified cluster compared to other clusters. High silhouette scores indicates a proper group distribution.

3.2.2.6 COX model for feature selection

To validate data fusion, the two datasets that obtained the lowest results with the feature selection by variance were selected and COX was used to make a new selection. These new selected feature were used as input for the four different autoencoder implementations.

3.2.2.7 Comparison with other data integration methods

The results obtained were compared with SNF, PCA, kernel PCA and sparse PCA. SNF fuses the similarty network to aggregate multi-omics data. COX was used for the feature selection. After that cluster from 3 to 6 where used for the clustering. PCA, kernel PCA (non-linear), and sparse PCA (regularized) where used to transform datasets in which features where selected based on variance as input to k-means/PAM clutering.

3.2.2.8 Differential expression and enrichment analysis on detected subtypes

A differential expression and functional enrichment analysis of the clusters was perfumed. DE genes and enriched processes where detected using LIMMA and ClusterProfiler respectively.

3.2.3 Results

3.2.3.1 Performance of different autoencoders

The silhouette score differs depending upon the regularization method: the optimal cluster number where selected counting the number of autoencoders that achieved a high silhouette score. A log-rank test were used to check if the identified clusters have different survival profiles. The lowest p-values with a high silhouette score for the optimal cluster number were considered as the final cluster prediction. There is no single winner architecture and the performance varies depending on the dataset.

GBM Cluster number 3, variational autoencoder with PAM/Spearman.

Coad Cluster number 3, vanilla autoencoder with PAM-Spearman.

3.2.3.2 Different similarity measures

PAM clustering with Spearman performed favorably than k-means clustering with Euclidian distance on the silhouette score, but worse on p-value for the survival difference between the identified clusters.

3.2.3.3 Supervised feature selection

Using a COX model there was a significant improvement of the p value for survival difference between the clusters, but a decrese in the silhouette score. Based on silhouette score VAE with Spearman was best.

3.2.3.4 Other subtype detection methods

PCA performed poorly, SNF comparable performance to autoencoders, but it has a few additional hyperparameters and the result is sensitive to their selection.

3.3 Variational autoencoders for cancer data integration: design principles and computational practice

3.3.1 Introduction

A complex biologica process is a combination of many simpler processes and its function is greater than the sum of its parts. Integrating and analysing simultaneously different data types offers better understanding of the mechanism. VAEs generate meaningful latent representations of integrated data that can be used for analysis bt a machine learning technique or be deployed on other heterogeneous datasets. The different machine learning approaches that integrate diverse data can be classified in:

- Output or late integration: different data are modelled separately and the output is the combined.
- Partial or intermediate integration: they produce a joint model learned from multiple data simultaneously.
- Full or early integration: different data is combined before applying a learning algorithm by aggregating them or by learning a common latent representation.

Unsupervised learning approaches learn representations by identifying patterns in the data and extracting meaningful knowledge while overcoming data complexities. Autoencoders are variants of deep learning networks that have good performance for unsupervised representation learning. The compressed representation they learn should capture the structure of the data and allow for more accurate downstream analysis. Now variational bayesian inference autoencoders learn the parameters of the underlying distribution of the input data and be utilized as methods for full integration of data. In this way they allow learning representations from heterogeneous data on different scales from different sources.

3.3.2 Materials and methods

VAEs are combined into a number of different architectures for a deep analysis and comparison with respect to specific data features and tasks. They are suitable since they are generative, non-linear, unsupervised and amendable to integrating diverse data.

3.3.2.1 Variational autoencoders

An autoencoder consist of an encoder that maps the high dimensional input data into a latent variable embedding with lower dimension and a decoder that attempts to reconstruct the input data from the embedding. There is a decoder function f and an encoder one g. The lower dimensional embedding is $h = g_\phi(x)$ and the reconstructed input $x' = f_\theta(g_\phi(x))$. θ and ϕ are learned together to output a reconstructed data sample ideally the same as the original input. To quantify the error one can use the cross entropy or the mean squared error. The main variants to autoencoders aim to address improving generalization, disentanglement and modification to sequence input models. The VAE uses stochastic inference to approximate the latent variables z as probability distributions. This represents and capture relevant features from the input. The are scalable and can deal with intractable posterior distributions by fitting an approximate inference or recognition model using a reparameterized variational lower bound estimator. The high dimensional data x is drawn from a random variable with distribution $p_{data}(x)$. It asssumes that x lies in a lower dimensional space that

can be characterized by an unobserved continuous random variable z. The prior $p_\theta(z)$ and likelihood $p_\theta(x|z)$ come from a family of parametric distributions, with its probability density functions differentiable almost everywhere. θ and the values of z are unknown and VAE approximates $p_\theta(x|z)$ using a recognition model $q_\theta(z|x)$ and the learned parameter ϕ. A VAE build an inference where given a data-point x it produces a distribution over the latent values z. A probabilistic decoder will, given a certain value of z, produce a distribution over the possible corresponding values of x, constructing the likelihood $p_\theta(x|z)$. Latent variables are tipically assumed centred isotropic multivariate gaussian $p_\phi(z) - N(z; 0, I)$ and $p_\theta(x|z)$ a multivariate Gaussian or bernoulli with parameters approximated using a fully connected NN. The posterior is intractable and it is assumed it takes the form of a gaussian with an approximately diagonal covariance. This allows to approximate the true posterior, converting the problem in an optimization one. The variational approximate posterior will be:

$$q_\phi(z|x^{(i)}) = N(z, \mu^{(i)}, \sigma^{(i)} I)$$

Where the mean and the variance are outputs of the encoder. The discrepancy between $p_\theta(z)$ and $q_\phi(z|x^{(i)})$ can be directly computed and differentiated. The likelihood will be:

$$l_i(\theta, \phi) = -\mathbb{E}_{q_\phi(z|x^{(i)})}[\log p_\theta(x|z)] + KL(q)\phi(z|x^{(i)}||p_\theta(z))$$

There are other choice for the prior other than the Gaussian like a mixture of Gaussians, or like a mixture of approximate posteriors, a Dirichlet process as a non-parametric prior through stick-breaking process, generalizing over the generative process, or a graphical model. Other posteriors are normalizing flows, auto-regressive flows and inverse auto-regressive flows. The influence of the disentanglement factore is controlled using a parameter β. Other retularizatio nterms can be maximum mean discrepancy MMD instead of KL diveregns.

$$MMD(q(z)||p(z)) = \mathbb{E}_{p(z),p(z')}[k(z, z')] + \mathbb{E}_{q(z),q(z')}[k(z, z')] - 2\mathbb{E}_{q(z),p(z')}[k(z, z')]$$

Where $k(z, z')$ is any universal kernel.

3.3.2.2 Variational autoencoders for data integration

3.3.2.2.1 Variational autoencoder with concatenated inputs In CNC-VAE the encoder is directly trained from different data sets aligned and concatenated at input. Beside this the rest of the architecture is a standard VAE: the input data is first scaled, aligned and concatenated before being fed to the network. It has one objective function that reconstructs the combined data. It aims at reducing redundancies and extracting meaningful structure across all input sources.

3.3.2.2.2 X-shaped variational autoencoder X-VAE merges high level representations of several heterogeneous data sources into a single latent representation by learning to reconstruct the input data from the common homogeneous representation. The architecture consists of individual branches, one for each data source that are combined into one before the bottleneck layer. In the decoding phase the merged ranch splits again into several branches that produce individual reconstructions of the inputs. It combines different loss functions for each data source, allowing for better learning an more meaningful repreentations.

3.3.2.2.3 Mixed-modal variational autoencoder MM-VAE attempts to address some of the limitations of X-VAE employing a more gradual integration in the hidden layers of the encoder. It builds upon transfer learning. It consists of branches that individually reconstruct the input data sources, however those share information with each other in the encoding phase. Higher-level

concepts are shared between all and used deeper in the network. So the information from different sources can be combined more gradually. The objective function combines different reconstruction loss functions.

3.3.2.2.4 Hiearachical variational autoencoder H-VAE builds upon meta-learning approaches for combining multiple individual models. It is comprised of several low level VAE that relate to each data source and the result is assembled together in a high-level VAE. Each of the low-level VAEs in emplyed to elarn a representation of an individual data source that are then merged together and fed to another VAE that produces the integrated data representation. The input to the high-level autoencoder implement distribution regularization terms in each of them separately and consist of approximated multivariate standard normal distributions.

3.3.2.3 Experiment setup

The aim of the evaluation is to seek the optimal configuration of choosing the appropriate objective function and parameters. Then evaluate and chose the most appropriate architectures for the data integration task performing a comparative quantitative analysis of the representations obtained from each of the architectures. Finally the results are chosen in terms of their application to cancer data integration. The genes were selected with the most significant Bonferrroni adjusted p-value. After missing data removal the input data consisted of 1000 fatures. All the datasets where sampled into five-fold cross-validation splits for each classification task. The depth of the architectures remained moderate and constant. All designs expect for MM-VAE the encoder and decoder were symmetric and consisted of compression and decompression dense layers placed before and after data merging. MM-VAI had an additional data-merging layer in the encoder network. They had depth between 2 and 4 hidden layers. They all used batch normalization and exponential linear unit activation. They had a hidden dropout with rate 0.2 The final layer employed sigmoid activation function. They used the Adam optimizer with $lr = 0.001$ and batch size of 64. To chose the optimal objective function that consider the reconstruction loss and a regularization term. For the former they used binary cross entropy loss and mean squared error loss. For the latter wighted KL and weighted MMD with different values of β. To evaluate the quality of the representations thwy used three integrative tasks, training a predictive model on the produced representation and measuring its predictive performance on a binary classification task of IHC cancer subtypes. For this they trained and evaluated a gaussian naive Bayes classifier. Then, they evaluated the performance of three different methods trained on different representation: Gaussian naive Bayes classifiaer, SVMs and Random Forest. These result where compared with the raw data and data transformed with PCA.