## SUMMARY

This project explores different approaches in terms of pre- and post-processing for improving the results of autoencoder-extracted mutational signatures.

The signatures we are trying to extract are created when mutations take place in cancer cells and can help determine the cause and origin of the cancer mutations. Cancer is a genetic disease that can be caused by many different factors which is why these signatures are interesting since each cause leaves a uniquely discoverable signature in the DNA, which is what we are aiming to extract.

The standard method for the current extraction of mutational signatures is Non-Negative Matrix Factorization (NMF) [19]. Our aim is to explore autoencoders for this purpose. The idea behind using autoencoders is their ability to encode data into a latent layer, we can then cluster to try and find the patterns in the data which equals the mutational signatures. We implemented all the different methods to be toggleable, for the purpose of testing different combinations and gathering a large amount of data to analyze the effects of these different methods.

For autoencoder we have taken inspiration from Pei et al. [13], who introduced their model called DeepMS, which is based on the Denoising Sparse Autoencoder [10], their approach was able to identify some of the same signatures that Alexandrov et al. [4] did, with the same dataset, But it's important that other are able to collaboratively validated this results. Pei used Keras with TensorFlow to implement the underlying autoencoder, while we're using PyTorch as our autoencoder.

Through experimenting with various combinations of values for the modules in pre- and post-processing and subsequently running it on the same dataset, we aim to analyze it, by validating against signatures extracted by Alexandrov et al. [4]. These methods are injection, feature filtering, and bootstrapping for pre-processing and clustering method, multi run of extraction method clustering, and silhouette metric for post-processing. This validation is performed by using cosine similarity, where the higher values indicate a greater resemblance to the extracted signature. The goal is to have a quantifiable metric for determining the performance of each method, thereby gaining valuable insight into their individual contributions to the extraction process.

The results of our study on different methods for enhancing the capabilities of autoencoders for mutational signature extracting were overall a success. A potential change to see better results would be changing the amount of method extraction runs. The reason for not changing this was the overall time cost of increasing the amount of runs. The benefits we could have gained from this would mainly be seen in the effect of bootstrapping the data. Our main issue holding us back from implementing this change was the time clustering took when increasing the method extraction runs. The reason behind the clustering being the bottleneck is that it currently is the only process not running on the GPU and instead running on a CPU, which is slower. This is something we tried to improve and make an implementation running on the GPU but the cost was worse clustering results were deemed too high.

# Autoencoders for Signature Extraction: Systematically evaluating Pre- and Post-Processing

Magni Jógvansson Hansen
Aalborg University
mjha19@student.aau.dk

Nikolai Eriksen Kure
Aalborg University
nkure19@student.aau.dk

## ABSTRACT

Cancer is fundamentally a genetic disorder caused by various different factors. Each mutation within the cancer genome leaves a unique and identifiable signature in the DNA sequence. These signatures are the focus of this paper, and how they can be used to identify the cause. In this paper, we aim to enhance the performance and accuracy of extracted signatures using autoencoders. To enhance the performance we explore and analyze different pre- and post-processing methods and compare the results to a baseline. To determine the performance of the different methods, by comparing the extracted signatures to known signatures from COSMIC and Signal. The comparison uses cosine similarity as the metric, and then later plotted for visualization of the results. The findings from the study showed that the added steps in the pipeline had a good effect and increased the performance and accuracy of the extracted signatures.

## 1 INTRODUCTION

Cancer is a genetic disease that can be caused by many different factors. This then leads to processes that mutate the DNA and cause uncontrollable cell growth and potentially abnormal cells [7]. These processes in the DNA lead to somatic mutations which are the focus area of this paper in the form of mutational catalogs where these mutations become visible. These catalogs are filled with a wide variety of somatic mutations, where the damage done has been logged throughout the lifespan of the cell in its DNA. Each of these somatic mutations stems from a different mutational processes, which leaves its own uniquely identifiable mutational signature in the DNA. Each of these signatures can be linked to a different mutational process that has taken place [4].

Finding these unique mutational signatures and linking them to different mutational processes allows for more personalized treatment and a deeper understanding of cancer. These signatures can be studied by extracting the patterns imprinted by the mutational processes in the DNA, which is the aim of this paper.

Trying to isolate these mutational signatures, resembles the blind source separation (BSS) problem since distinguishing what mutational processes are responsible for the exposures. A common thought experiment to explain this is the "cocktail party" problem; the basic idea is trying to distinguish one voice in a room full of voices using microphones. This translates well into how mutational signatures work, the loudness of the voice resembles the exposures of the mutational process, the microphones are the DNA itself recording the changes, and these recordings are then the mutational catalog we aim to understand [4].

In our previous project from our 9$^{th}$ semester [6], we aimed to develop a new method for extracting signatures with autoencoders instead of using non-negative matrix factorization. Overall, it was a success and the method were proven to work but had its limitations, which we explored and improved upon in this paper.

In this project, our primary goal is to build upon the foundation laid during our 9$^{th}$ semester [6], aiming for significant improvements in the framework we developed. Our focus is implementing a series of pre- and post-processing methods around the existing models. By implementing these methods, we aim to provide a comprehensive evaluation of the impact of different approaches has on the extraction process and improve the quality of results obtained.

Drawing inspiration from the methods outlined in the Alexandrov et at. paper [4], bootstrap, and feature filtering[1] we intend to implement and test these techniques, to find the optimal suite for our specific framework, and autoencoders instead of the NMF approach they took. They were able to identify 67 distinct mutational signatures from a dataset that is from Pan-Cancer Analysis of Whole Genomes (PCAWG) [12] [9], that have been added to the database COSMIC [1].

The knowledge gained from Alexandrov et al. work, is used as a guideline as we implement different versions of pre- and post-processing. The parameters are split into the previously mentioned pre- and post-processing, for pre-processing we concentrated on the effects of injection, bootstrapping, and feature filtering. The aim of these methods where to test their effect when improving the dataset before it was passed to the autoencoder. Post-processing focused on different methods to extract signatures from the latent space. This included clustering methods, optimal cluster amount, and how to optimally cluster multi-run extractions.

When exploring different combinations of the above-mentioned parameters, multiple runs were executed to generate data on all the different effects. To be certain that we're able to extract signatures, we're going to use another dataset that Degasperi et al. [5] used to extract signatures. They were able to extract and validate against Alexandrov et al. findings and identify 40 new signatures that were added to Signal database [3]. These samples of genome datasets are added together from 3 different sources, called Genomics England International (GEL) [17], International Cancer Genome Consortium (ICGC) [11], and Hartwig Medical Foundation (HMF) [15] [3].

The methods tested had the desired effect on the extracted signature and improved the extracted signature accuracy and the amount found. Some of the methods tested had better results compared to others. One of the main reasons behind the difference in success between our bootstrapping and other versions comes down to the lower amount of method extraction runs. This was a deliberate choice on our end as a trade-off for more data on other parameters.

In essence, our project aims to use already explored methods in new ways to try and push the boundaries of current methods thereby

[1] In the Alexandrov et al. paper this is called Dimension reduction

improving the field of mutational signature research and helping doctors better understand cancer and what causes it.

## 2 BACKGROUND

The goal of this section is to introduce key theories and methods used in this paper. This includes the computational methods used for the extraction, and the different pre- and post-processing methods used in the development of the framework.

### 2.1 Single-Base Substitutions

Single-Base Substitutions (SBS), are a class of somatic mutations in DNA. An important attribute of DNA is strand symmetry, which is responsible for ensuring the structure of the double helix is preserved [4], meaning they are stored in these six sub-types; C:G > A:T, C:G > G:C, C:G > T:A, T:A > A:T, T:A > C:G, and T:A > G:C. The context

| Type | G1 | G2 | G3 | G4 | G5 |
|---|---|---|---|---|---|
| A[C>A]A | 53 | 33 | 131 | 279 | 54 |
| A[C>A]C | 43 | 19 | 81 | 199 | 54 |
| A[C>A]G | 16 | 5 | 11 | 32 | 11 |
| A[C>A]T | 28 | 19 | 70 | 202 | 38 |

**Table 1: A sample of the data the framework uses**

of the surrounding bases is an important part when working with signatures and mutations in general. When handling the surrounding bases we concentrated on one base to each side for increased context, as we can have 4 different bases, therefore having 4x4 different possibilities of neighbor to the mutation. With one base on each side and types of mutations, it added up to 96 possible mutations (e.g., $4 \times 4 \times 6 = 96$ possible pairs). The format of the data when working with neighbors becomes as shown in table 1, where [C>A] is the mutation type C:G > A:T, and the bases beside the mutation are its neighbors.

### 2.2 Autoencoders

An autoencoder stands as a distinct variant within the field of neural networks, developed to transform input data into a more compressed latent representation of said data, a process often referred to as dimension reduction. This representation is then utilized to reconstruct the original input as accurately as possible.

This is achieved by utilizing two different functions, an encoding function that is responsible for transforming the input data into the compressed latent layer, and a decoding function that takes the compressed representation layer and tries to reconstruct the original data. By using these two methods in correlation with each other the model aims to find an efficient compressed representation of the input data provided [10].

The model achieves this representation over iterations of the data and measures the progress with the use of a loss function. A loss function in the case of an autoencoder is often referred to as reconstruction loss which measures the difference between the input data and the reconstructed output. The lower the reconstruction, loss the better the model is interpreting the input data in its latent space. Within autoencoders, various loss functions such as Kullback-Leibler divergence (KL) and Mean Squared Error (MSE) are utilized to estimate the reconstruction loss, indicating how well the dot product of $W$ and $H$ describes the original dataset.

### 2.3 Non-negative Matrix Factorization

Non-negative Matrix Factorization (NMF) [19] is a technique used for breaking down matrices into non-negative components. This method is used for dimensionality reduction in multivariate data analysis to find the patterns in the DNA. Each component, representing a linear combination of the initial attributes, comprises solely non-negative coefficients.

Given a matrix $A \in \mathbb{R}_+^{m \times n}$, NMF decompose $A$ into two matrices $W \in \mathbb{R}_+^{m \times k}$ and $H \in \mathbb{R}_+^{k \times n}$, such that $A \approx W \cdot H$, where $0 < k < m \mid 0 < k < n$.

To achieve this approximation, NMF employs an iterative approach. Starting with initial values for $W$ and $H$, the algorithm adjusts these matrices iteratively to minimize the difference between their product and $A$. This process continues until convergence, where either no significant changes occur over several iterations or the maximum specified iterations are reached. For NMF the same loss functions can be used as mentioned in the section 2.2 regarding autoencoders apply.

In practical applications, NMF effectively transforms the original data into a new set of attributes, represented by the components derived through the factorization process. Alexandrov et al. [4] used NMF to extract signatures, by letting matrix A be the dataset of genomes, and W and H matrices be the exposure of a given set of signatures and the extracted signatures.

### 2.4 Bootstrap

By using Bootstrap, we can create multiple samples with some variation on its mutations, based on properties of the original dataset, while keeping the representative of the original dataset in the new dataset. This can help to extract more signatures which wouldn't be found in the original dataset [16].

### 2.5 Feature Filtering

Feature filtering is a method to reduce the overall dimensions of data by cutting out statistically less important data while keeping the important properties of the data. This is done to decrease computation time when training our model, but more importantly, it can improve the accuracy of the model [14]. The implementation of feature filtering in our framework is based on a cut-off threshold that decides if the mutation contributes enough of the overall mutations to stay in the dataset. If the mutation doesn't meet the cut-off they are removed and reintroduced as zeros when extracted signatures are done, the feature filtering threshold has been tested with different values to find the best result.

## 3 RELATED WORK

In this section, we review and discuss other papers within the same field of research.

This paper is a further development of our $9^{\text{th}}$ semester paper [6]. We utilized the knowledge gained from a semester exploring the domain and improved on the developed framework from our $9^{\text{th}}$ semester [6]. To improve the framework from our last semester we looked at Alexandrov et al. [4] and Degasperi et al. [5] papers for

ways to extend the current framework, which we go more in-depth below here and in appendix A.

The paper our project is based on, is the Alexandrov et al. paper on mutational signature extraction with NMF [4]. At the time this field was still new and therefore lacked a good understanding of both mutational signatures and how to extract them. This makes this paper one of the groundbreaking methods used to extract these signatures and develop the research field.

Their solution was based on NMF for extracting the signatures from the input data. Their NMF was created with a focus on the convergence criteria to help stability and make it reliable over multiple runs.

Their work led to the development of the SigProfleExtractor Python library[2]. This software framework offers a streamlined method for signature extraction from the input data and a search space. With this method, researchers can effortlessly extract and try to understand the different signatures found across the different types of cancer tested. The framework gives both a data and visual representation of the signatures extracted in the process, which allows the researchers to compare different signatures found across the cancer types and patients and find common signatures.

The approach of using machine intelligence, more specifically autoencoders, to try and work around the limitations of NMF approaches was developed in a study by Pei et al. [13]. Their approach utilized a denoising space autoencoder intending to improve the precision of extracted signatures from various datasets used in training. The model used tried to break down the input into a latent layer which then could be reconstructed in the decoder part of the autoencoder, to get as low a reconstruction error as possible. This then allowed them to look at the latent layer to see which patterns it had generated and these would then be the signatures. Before the signatures could be used they were clustered together with the use of cosine similarity since each latent node can be processed as a vector.

Another relevant paper is by Genomics England and Andrea Degasperi et al. [5]. Their approach is similar to Alexandrov et al. by also using NMF, but in addition using GINI, to see which signatures had the most exposures for each genome and removing those genomes from the dataset, then run again, to see if they could find rare signatures, that weren't able to be extracted in the previous run. The main part of their paper we were interested in was the dataset they used. Since that dataset has a higher genome count, resulting in a greatly increased amount of training data. The paper was also a good insight into new processes to explore for a potential increase in the accuracy of extracted signatures.

In our study of the papers above we analyze the different pre- and post-processing methods. The idea behind this is to utilize these methods with the autoencoder developed in our 9$^{th}$ semester [6] project. To improve the framework and explore the effects on extracted signatures.

# 4 METHODOLOGY

In this section we are describing the process of how our extended framework is different and how we aim to provide quantifiable data on the effect of different pre- and post-processing.

[2]https://github.com/AlexandrovLab/SigProfilerExtractor

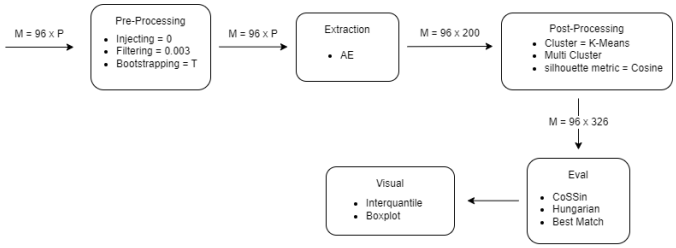


**Figure 1: Flowchart of the updated framework, based on PCAWG breast dataset**

In the flowchart 1 the flow of the extended SEEF framework is shown. All the steps and matrixes passed between each step are in an abstract format. Post-processing steps include clustering the latents with K-Means and the silhouette metric being used is cosine. The multi clustering step is described in subsection 4.7. Ending with evaluating the extracted signatures against known ones and visualizing the results for analysis.

## 4.1 Method Comparison

To give quantifiable statistics on how each of the different pre- and post-processing methods impacted the result we are using cosine similarity to determine this, by validating extraction signatures against reference signatures.

## 4.2 Method Evaluation

To evaluate the performance of the parameters, we need a way to determine the accuracy of the extraction method. Taking inspiration from Alexandrov et al. [4] and Degasperi et al. [5], who used cosine similarity to validate their signatures. Cosine similarity utilizes the property of interpreting the signatures as vectors to compare how similar they are. The equation to compare two vectors in cosine similarity is defined as follows:

$$cosine\ similarity(A, B) = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} \mathrm{A}_i^2} * \sqrt{\sum_{i=1}^{n} \mathrm{B}_i^2}} \quad (1)$$

[18] where n is the given size of the dimension, the value ranges from 0 to 1, 0 indicates they're orthogonal, and 1 indicates they are the same vector. Using cosine similarity also allows us to validate against reference signature in COSMIC [1] and from Signal [3]. One potential issue, with only using Cosine similarity, is that we can't say if multiple extracted signatures are matching to the same reference signatures. To solve this issue, we employ the Hungarian algorithm [2], where the reference signature is the column, and the extracted signatures are the row, and the cosine similarity is the weight, normally the Hungarian algorithm focuses on getting the lowest overall weight, but in our case we need the highest weight.

## 4.3 Datasets

The real datasets used to perform evaluation of the framework are PCAWG and GEL, as described in section 1. These are the same datasets that Alexandrov et al. and Degasperi et al. used to identify their signatures. As described in section 1 we inject synthetic data into the real dataset.

*4.3.1 Real Data.* Both real datasets consist of samples taken from 22 different organs, and we have specifically focuses on breast. The dimensions of these two datasets with breast are different. The PCAWG dataset comprises of 96 rows (SBS mutation types) by 214 columns (samples) [9], while the dimension for GEL dataset has 96 rows by 2572 columns [5].

*4.3.2 Signatures Data.* The reference signatures are in two different datasets, COSMIC and Signal, as described in section 1. We can evaluate the framework, by validating the extracted signatures against the reference signatures, by using cosine similarity to determine the accuracy of the signatures. Additionally, all extracted signatures get counted and compared to reference signatures to see how many of the found signatures have a good match with already referenced signatures.

*4.3.3 Synthetic Data.* Comparing synthetic data is a slightly different approach than with the real data process mentioned in section 4.3.2. When it comes to synthetic data the framework is already aware of which signatures were used to generate the dataset and that is therefore the signatures it is looking for in the data. This also means it can tell if the extraction method found the correct number of signatures or if it found too many or too few since the number of signatures and which ones were known in advance when the dataset was generated. Otherwise, the process is similar to the real data.

**Algorithm 1** SynthData algorithm from our 9th semester paper [6]

**Input:** Array of random given amount reference signatures S, Sample amount N
**Output:** Mutation catalog

**for** i = 0 **to** N **do**
$x = \{y \mid \forall x \in shuffle[1, 1, 0, 0, 1]\ y = x \cdot Pois(1)\}$
$xs = \sum_{i=0}^{4} x_i$
$P = \frac{x}{xs} \cdot \text{unif}(500, 1000)$
$V = S \odot P$
$Noise = \{y \mid \forall x \in [0..95],\ y = Pois(1)\}$
$NoiseSum = \sum_{i=0}^{95} Noise$
$Noise = \frac{Noise}{NoiseSum} * P_i * 0.01$
$Final = V \odot Noise$
**end for**

The algorithm 1 is from our 9th semester paper [6] since the method for generating synthetic data is the same. The algorithm works by selecting a set amount of reference signatures that have a shared cosine similarity lower than 0.7 to ensure different signatures. Secondly, a matrix for exposure is generated with random values between 500-1000 in order to represent the mutations in each sample. This mutation count is then distributed among the mutations in the signature and aa set amount of the signatures are selected to have zero mutations. To finalize the synthetic data the dot product of the two matrices is calculated and Poisson noise is added.

*4.3.4 Injected Data.* Injected data is a new dataset in the upgraded framework, so it also needs a new and updated comparison method to go along with it. With this kind of data, the framework knows which signatures were injected, but it doesn't know about which ones are in the real data. So, when this option is used the framework combines the two methods mentioned above.

## 4.4 Ground Truth

By using ground truth, we are able to assess whether the method extraction is extracting signatures from the dataset. To do this we need to use a custom-made signature, that we can reference back to, to be certain that it is able to extract it.

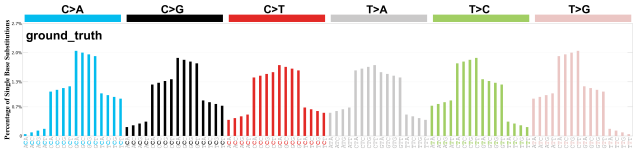


**Figure 2: Signature of our custom-made one**

Using the custom-made signature shown in figure 2, we can create synthetic data from it to inject into the real dataset, and we can be certain that we're extracting our own signature, when running on both datasets, as it has a maximum cosine similarity 0.75296 to Signal and 0.68001 to COSMIC.

## 4.5 Clustering Method

To gain useful information about the latent space, we need to run clustering on it, as there is a chance that some of the latent are not distinct enough. Therefore it should be clustered together with its neighbor, and to find the correct number of clusters we run from 2 up to the latent dimension.

*4.5.1 K-Means.* The K-Means algorithm groups data points into a specific amount of clusters set by K, and adds them to a group of data points. The most common way of doing this is by calculating the squared Euclidean distance to each centroid, and then adding it to the cluster where the distance is the lowest. The points of the centroid for each cluster will be defined as the sum of all Euclidean distances from centroids to each data point that belongs to that cluster and divided by the amount of data points in that cluster. [8]

*4.5.2 Cosine Clustering.* Cosine Clustering is based on how the K-Means algorithm works, but instead of using squared Euclidean distance to calculate the distance we use cosine similarity, where the highest value for each data point will be assigned to that cluster, you still have to specify the number of how many clusters it should use.

## 4.6 Optimal Amount Clusters

The problem with the optimal amount of clusters in a given latent space is not knowing what the optimal amount of clusters is, nor how one should define it. Therefore, we first need to run through all possible amounts of clusters in a given latent space, defined as: $k = \{x \in \mathbb{Z}_+ \mid 2 < x < |latentspace|\}$, To define how we choose the optimal amount, we define it differently for each clustering method. For K-Means we define it as:

$$auxiliary = \widehat{inertia} - \alpha * \widehat{silhouette} \quad (2)$$

The values are normalized, indicated by $\widehat{\ }$. Inertia is the sum of the squared distance of each sample in a given cluster. Silhouette

calculates the difference between the sample distance to its given cluster, and its mean nearest cluster distance. The score ranges from -1 to 0, -1 indicates that it's assigned to the wrong cluster, 0 indicates that the clusters are overlapping, and 1 indicates that the sample is assigned to the correct cluster. The $\alpha$ parameter is used as a weight to determine how much contribution the silhouette score to the equation 2, 3, and 4. For the optimal amount of clusters in Cosine Clustering, we define the equation as:

$$auxiliary = cosine\ similarity\ mean + \alpha * silhouette \quad (3)$$

Cosine similarity mean is the sum of each sample cosine similarity score to its given cluster centroid, divided by the amount of samples in a given cluster.

## 4.7 Optimal Cluster Selection in Multi Run Extraction

When picking the optimal solutions for the cluster result, there are two ways of choosing. The first way is to pick the most optimal cluster out of all the method extraction runs. The problem with this approach is that all the other runs become discarded, and the variation in those runs are lost.

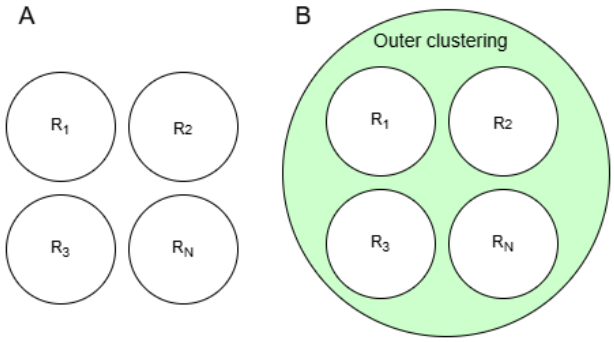


**Figure 3: Illustration of Optimal Cluster Selection in Multi Run Extraction**

Therefore, a new implementation was made, that would concatenate the optimal cluster from each method extraction run, and cluster again. By concatenating all clusters from each method extraction run together, the variation from both bootstrapping and each method extraction is preserved, since bootstrap is run for each method extraction. We can see how this works in figure 3, where A shows the approach of picking the optimal cluster from a given method extraction run and discards the other, and B shows the new implementation where the optimal cluster is chosen from each run and cluster again.

If chosen using the first implementation the equation for K-Means is defined as follows:

$$auxiliary = \widehat{loss} - \alpha * \widehat{silhouette} \quad (4)$$

The loss in equation 4 is the loss value from each method extraction run, while using the second implementation K-Means uses equation 2 for finding the optimal amount of clusters in the given space, and cosine uses 3.

## 4.8 Output

The output of the framework is a series of files containing data about how many signatures were found, how they matched up with reference signatures, and general information about the run. This is the same as the previous framework developed in the 9$^{th}$ semester [6], in addition, the functionality added in this semester will also be outputted. This extra output will be the information about how different methods impacted the results and raw data to avoid needing to run the entire process when exploring output data.

*4.8.1 Pre-process Parameter.* The pre-process parameter is saved into a tsv file, which contains the extraction method used, the number of times the method extraction is set to run, what the feature filtering threshold is set to, if feature filtering was applied, to what mutation type it was applied to, if bootstrap was used on the data, what synthetic injection percentage was used, what noise percentage was used, how many rows and columns are in the dataset before going into method extraction.

*4.8.2 Latent Space.* For each method extraction run, the latent space is saved.

*4.8.3 Signatures.* After the optimal clusters has been chosen, the centroids is saved into a signatures.tsv file.

*4.8.4 Cosine Matrix.* The cosine matrix is the file that contains cosine similarity between the clustered latent and known mutational signatures from either COSMIC or Signal.

*4.8.5 Known Signatures.* The known signatures file is a file that focuses on assigning each latent to a unique reference signature with the highest cosine similarity.

*4.8.6 Best Match.* The best Match file, contains the best overall extracted signatures to reference, by using the Hungarian algorithm

*4.8.7 Results.* The results are saved into a tsv file. It contains the pre- and post-process parameters that were used, how many signatures were found, how high of a cosine similarity they have against reference signatures from either COSMIC or Signal.

*4.8.8 Visual Results.* The Visual Results files, are the saved images of the box-plot for the pre- and post-process parameters that were used.

# 5 EXPERIMENTAL SETUP

In this section, we present and analyze the experimental findings from the improved SEEF framework. This includes how the different pre- and post-processing methods affect the accuracy of the extracted signatures

## 5.1 Dataset

The evaluation of the different pre- and post-processing methods is done on a few different datasets, real dataset, real dataset with synthetic dataset by injection. The synthetic data is generated with the same process as in our previous project, with the addition of the capability of creating injected data sets too. These injected datasets allow us to inject a percentage based on sample size into real data to make sure we can extract signatures we know to be in there.

*5.1.1 Real Datasets.* As mentioned in section 4.3.1 we are using two different datasets to extract signatures from PCAWG and GEL, and they have different amount of samples.

| | Total sum | Mean | Median | STD | Max | Min |
|---|---|---|---|---|---|---|
| GEL | 16.841.637 | 6.548 | 3.497 | 10.777 | 243.907 | 808 |
| PCAWG | 1.515.569 | 7.082 | 4.639 | 7.874 | 65.065 | 1.203 |

**Table 2: quantitative description of the two breasts datasets**

As seen in table 2 the datasets are different, interestingly the std, for GEL is higher than PCAWG, as GEL is 10 times bigger than PCAWG, GEL has the largest genome that has over 200.000 mutation, while PCAWG largest has just over 65.000, but interestingly the mean and median for PCAWG is higher than GEL.

*5.1.2 COSMIC Database.* COSMIC is a database comprised of mutational signatures that have extracted 87 signatures with the current state-of-the-art methods for signature extraction. The dataset used to compile the data is the PCAWG dataset. Some of the extracted signatures have also been validated with real-life research and some have just been found in many patients [1].

*5.1.3 Signal Database.* The Signal database [3] is a mutational signature database from the University of Cambridge using COSMIC as a reference, they have been able to extract 121 signatures. We chose to use the organ-specific breast reference signatures from Signal based on GEL, that contains 28 signatures they extracted, rather than the whole dataset. As we're using the GEL dataset to extract signatures from.

*5.1.4 Synthetic Dataset Generation.* In short, synthetic data can be generated from COSMIC, Signal, or custom-made signatures, combined with random amounts of exposure in each patient. The amount of signatures and patients used is a parameter when running the data generation. This process is explained in section 4.3.3.

*5.1.5 Injected Dataset Generation.* Injected data works by utilizing the synthetic data generation tool described in section 4.3.3 to generate the part injected into the real data. The amount of injected data into the real data is controlled by a parameter for what percentage of the data should be injected. When the desired amount of data is generated the final step is concatenating the generated data onto the real data and shuffling the columns.

## 5.2 Parameter

The parameters are split into three different groups, pre-processing, post-processing, and method extraction.

*5.2.1 Pre-processing.* In pre-processing, we have 6 different parameters we can change, before it goes into the method extraction part. feature filtering, bootstrap, injection, and noise.

*5.2.2 Method Extraction.* For our method extraction, we used Optuna[3] to help find the suitable hyper-parameters for the autoencoder. The hyper-parameters that were tuned are epochs, batch size, latent dimension, and the learning rate. Through Optuna recommendation and observation of the results, the conclusion was that epochs should be set at 500, batch sizes at 8, learning rate at 1-e03, and latent was most optimal at 200.

*5.2.3 Post-processing.* When method extraction is done, we have options to choose from on how the post-processing should go, as mentioned in section 4, we can choose what kind of clustering method we want to use, how to pick the optimal solution, how the silhouette score should be, and how much weight should the silhouette score have.

*5.2.4 Combination of Parameters.* With this we have multiple parameters to tune, but the focus lies on 10 parameters. Those 10 parameters are; the number parameters are alpha, latent, feature filtering, injection, and noise, the Categorical parameters are clustering method, optimal clusters selection, and silhouette metric, and for the Boolean parameter, we have bootstrap.

*Latent.* With latent dimension tuning, when training on the two datasets, the latent starts from different sizes, because of the difference between the two dataset sizes as described in section 4.3.1 affecting how big the latent dimension can be; therefore when training on PCAWG starts from 200, while GEL starts from 250. Alexandrov et al. and Degasperi et al. ran with component sizes 20 and 10 individually, which is lower than our optimal results. Therefor the tuning steps are down to 20, with an increment of 50.

*Feature Filtering.* As mentioned is inspired by Alexandrov et al. paper [4], they use 1% as their threshold, to find if it is optimal for autoencoder, we start from 0 to get a baseline, and then step up to 1%.

*Injection.* With Injection we only need to prove that it is learning, which allows us not to run it with such fine increments. Therefore we start with 0% injections to get a baseline of cosine similarity against the custom-made signature and see if it increases the more injection data is put into the real data.

*Noise.* To see if by adding noise the autoencoder gets better at learning the underlying structure of the data, therefore we start from 0% to get a baseline, and go up to 5%.

*Clustering Method.* As explained in section 4.5, we implemented two different methods: K-Means and cosine clustering.

*Cluster Selection in Multi Run.* As explained in section 4.7 cluster selection only has two options, the first option is picking the single best run, which was the implementation from the previous paper, and the second option is the new implementation.

*Silhouette Metric.* Silhouette has multiple metrics it can use to calculate it's score for the clusters in a given space, the standard metric is Euclidean distance. We want to see whether using cosine similarity to calculate the score, we can improve the evaluation against reference signatures, as we are using cosine similarity to validate extracted signatures against reference signatures.

$\alpha$. To see how much effect silhouette score has on picking the optimal cluster, we are starting with the silhouette having equal weight to the equation, as from the previous paper, and then incrementally step down to 0.6.

With this the list of each parameters that both dataset ran on are as follow:

[3] https://optuna.org/

**Feature filtering** = [0, 0.001, 0.003, 0.005, 0.007, 0.009, 0.01]
**Injection** = [0, 1, 5, 10]
**Noise** = [0, 0.01, 0.03, 0.05]
**Bootstrap** = [True, False]
**Clustering method** = [K-means, Cosine]
**Cluster selection in multi run** = [single, multi]
**silhouette metric** = [cosine, euclidean]
$\alpha$ = [1, 0.9, 0.8, 0.7, 0.6]

The only difference is the latent list. For PCAWG it is as follows:

**latent** = [200, 150, 100, 50, 20]

While for GEL the latent list is:

**latent** = [250, 200, 150, 100, 50, 20]

As the combinations are different for each data, the post-process combination is the same, which comes to 40 combinations, while pre-processing and method extraction are different. For PCAWG the combination comes to 1120, and for GEL it's 1344. The combination total for the PCAWG becomes 44800, and for GEL it's 53760. The combination is run 3 times.

## 5.3 Parameter Runs

When a pre-parameter combination is run, the 40 different combinations post-parameter is then run on the latent space, the method extraction run runs 10 times. This full process runs 3 times.

## 5.4 Parameter Results Setup

The parameter results are split into 3 groups, pre-process parameters, post-process parameters, and injection. For pre- and pro-process, the injection parameter gets filtered out if it's over 0, to not influence the results. The value is then aggregated into the mean value. For injection it does not get aggregated, but the max, mean, and min get calculated. Then each is box-plotted.

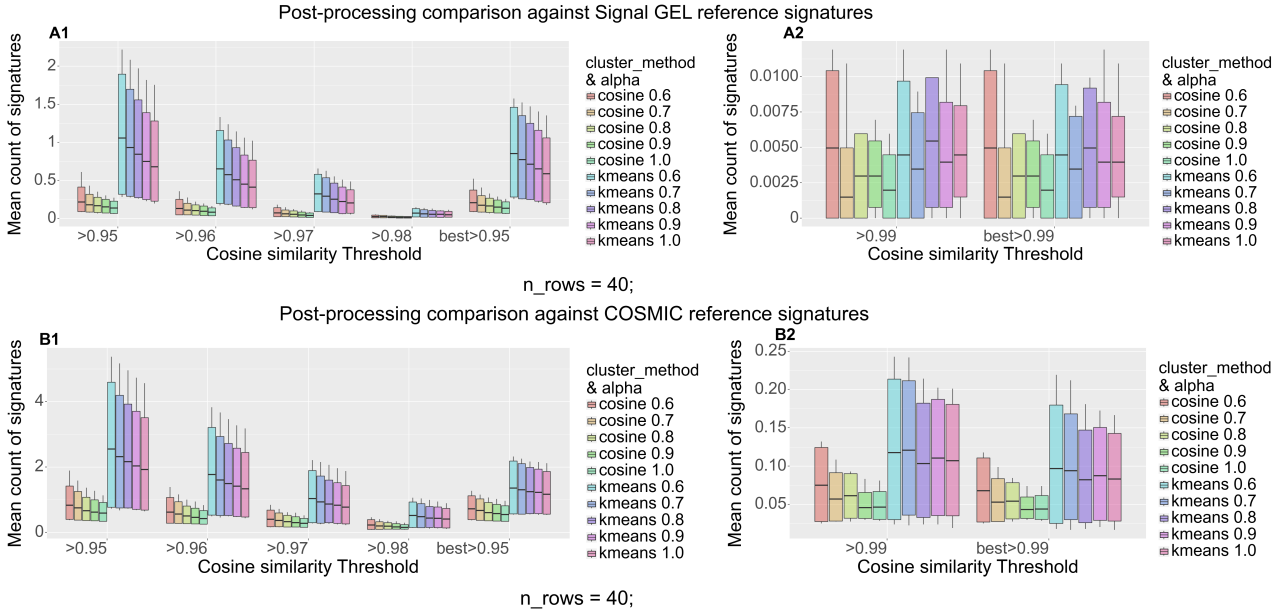


**Figure 4: Boxplots of clustering methods and alpha variable post-processing**

## 6 EXPERIMENTAL RESULTS

In this section, we explore the findings from the different tests of multiple combinations of pre- and post-processing. This includes different plots to help distinguish the best combination of parameters for extracting the most accurate signatures.

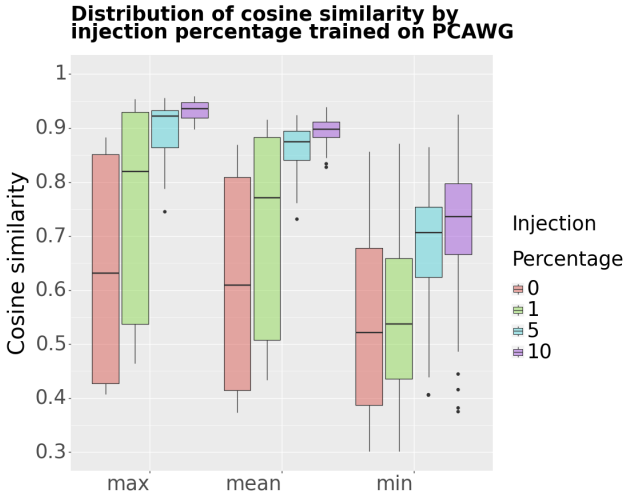


**Figure 5: Box-plot of injection percentage**

### 6.1 Ground Truth

Before we look at the findings, we need to know if the autoencoder is able to learn from the dataset, and to do that, we have to look at the injection part, to see if the cosine similarity to the custom-made signature is higher when injection percentage increases.

The autoencoder is able to extract the custom-made signature from the PCAWG dataset, and we can see that in figure 5 that the higher the injection percentage is, the higher and more consistent the extract becomes, and we know this is not a false positive as described in section 4.4 the maximum cosine similarity for custom-made signature to reference signature is 0.68001.

### 6.2 Parameter Results For Pre- and Post-processing

In this section, we present the results from the test of different combinations of pre- and post-processing methods and the optimal options.

*6.2.1 Post-processing results.* With the problem as described in section 4.7, picking the single best extraction, seems correct as can be seen in figure 6. When concatenating all optimal clusters and clustering again, it results in a more accurate extraction of signatures from both datasets. For both clustering methods with silhouette metrics, they both performed better when using the opposite metric, through all cosine similarity threshold steps. When the $\alpha$ weight is lower, it is able to extract more signatures through all steps, except at 0.99 for K-Means against Signal reference signatures, where 0.8 weight has worse $Q_3$, but better $Q_1$ and median. When looking through figures 7, 4, and 6, the K-Means is able to extract around double the signatures than Cosine cluster.

*6.2.2 Optimal post-parameter.* When looking through the figures for both datasets, it becomes clear that K-Means is the clustering method to use, the silhouette should be using Cosine as its metric, the $\alpha$ weight should be at 0.6, and the lastly the type clustering should be using the multi-implementation

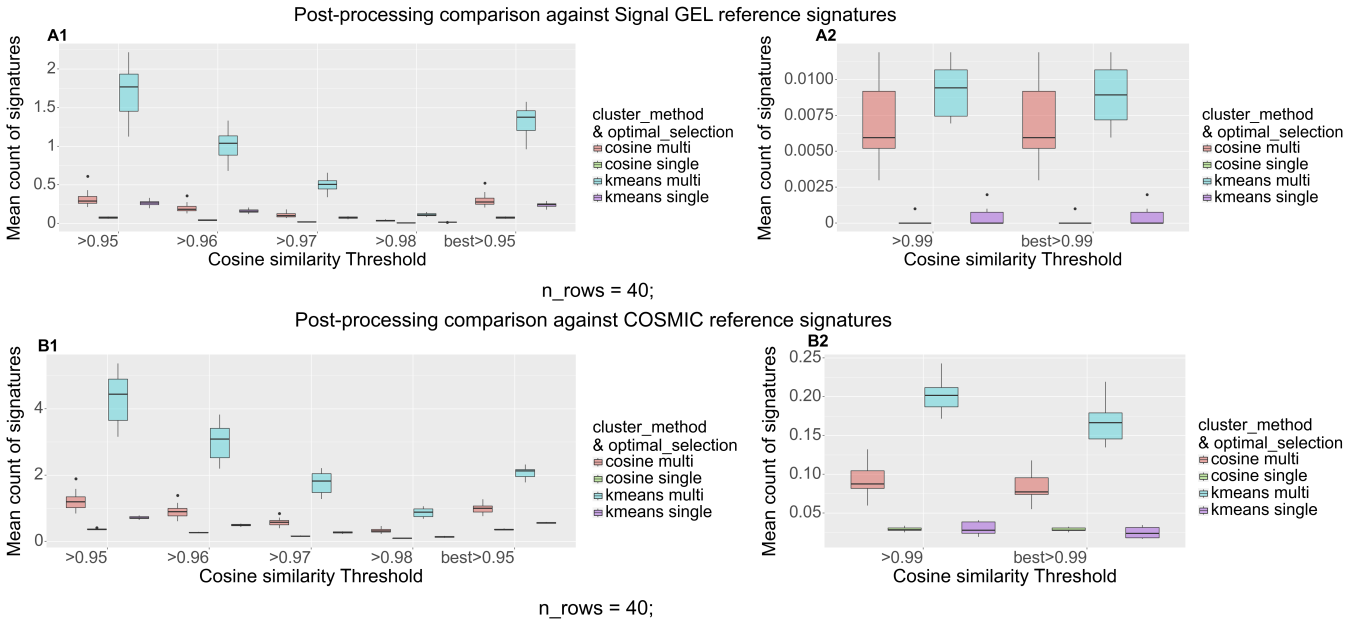


**Figure 6: Box-plots of optimal selection in multi run post-processing**

**A1 & A2**

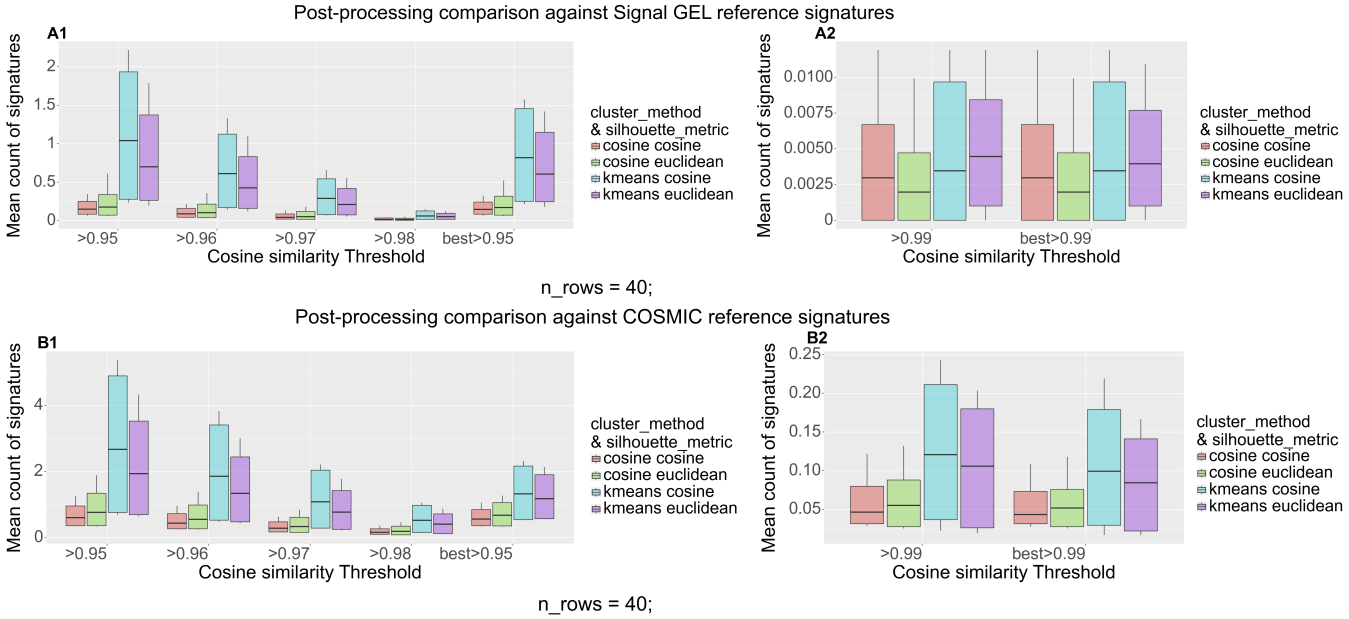


**Figure 7: Box-plots of the clustering methods post-processing**

**A1 & A2**

*6.2.3 Pre-parameters results.* Looking at latent results in figures 8, the autoencoder extracted more signatures for both datasets, bigger the latent dimension is. Interestingly the feature filtering threshold has a limit on how high it can be before it starts extracting less signatures, as can be seen in figure 9, and the limit is different between the two datasets. For COSMIC it started performing worse higher than 0.003, while the value 0.01, which Alexandrov et al. [4] used, performed worse than the baseline. The noise value, performs similarly between the two datasets, where 0.01 noise is the limit, before the extractions gets worse. When it comes to bootstrapping

the current amount of method extraction runs makes the results negligible.

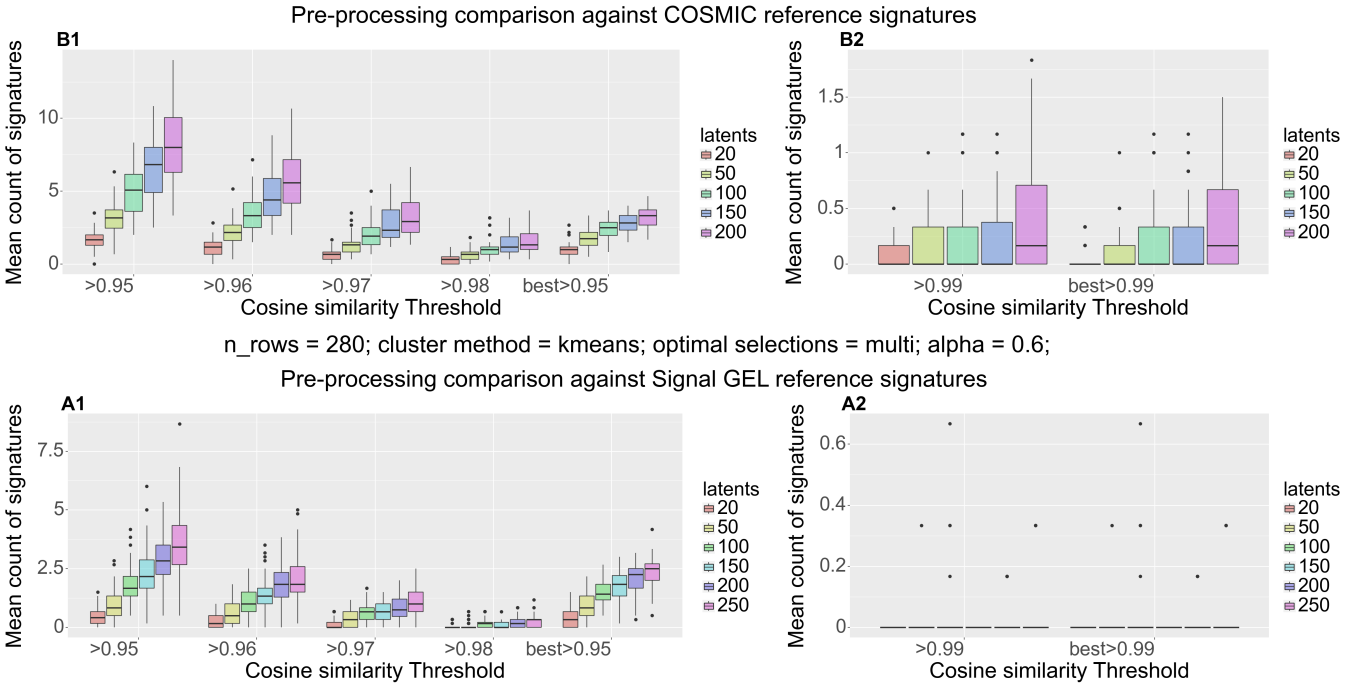


**Figure 8: Boxplots of latent space pre-processing**

**A1 & A2**

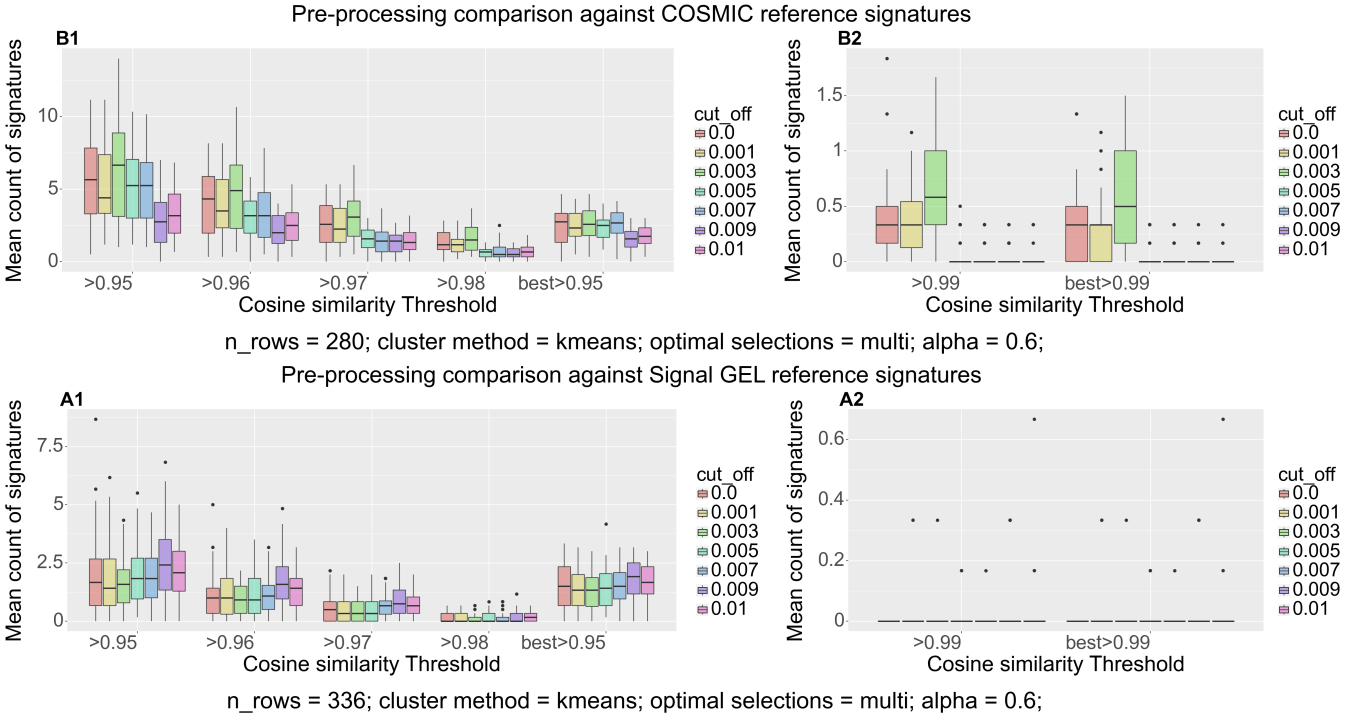


**Figure 9: Boxplots of the feature filtering pre-processing**

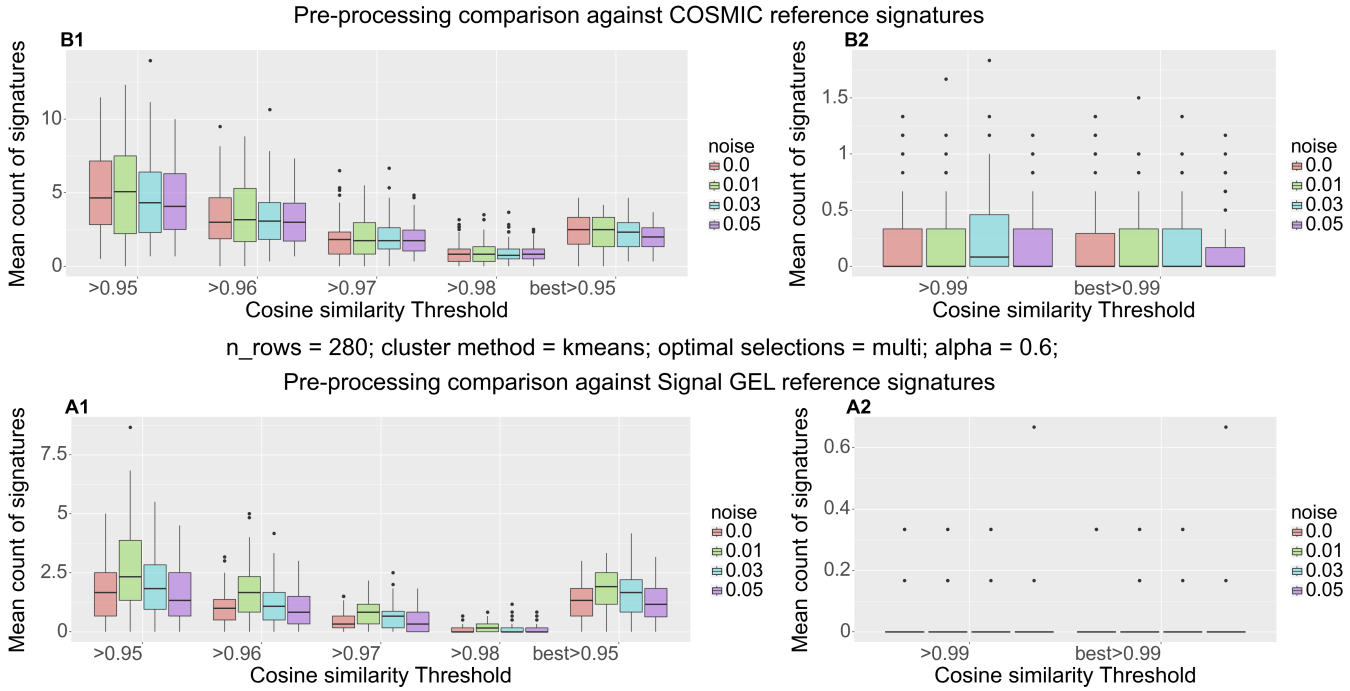


**Figure 10: Boxplots of the noise pre-processing**

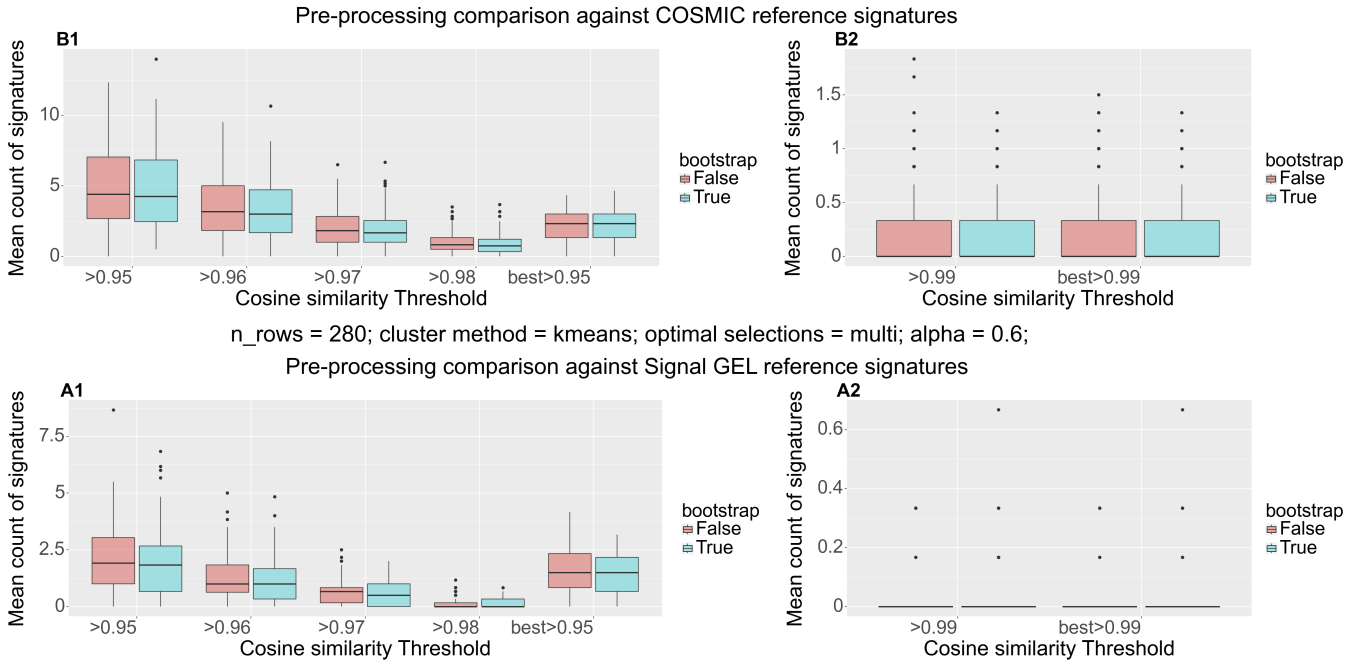


**Figure 11: Boxplots of the bootstrapping pre-processing**

# 7 DISCUSSION

In this section, we discuss and provide an in-depth analysis of the topics mentioned in the sections: Methodology and Experimental Findings.

## 7.1 Cosine Similarity Thresholds

To enhance the understanding of the results, the frequency of the threshold interval has been significantly increased, since the previous thresholds failed to provide an accurate insight into the trends due to the gaps between thresholds. These changes also improve the signatures due to the large difference between a signature with a cosine of 0.85 and 0.9.

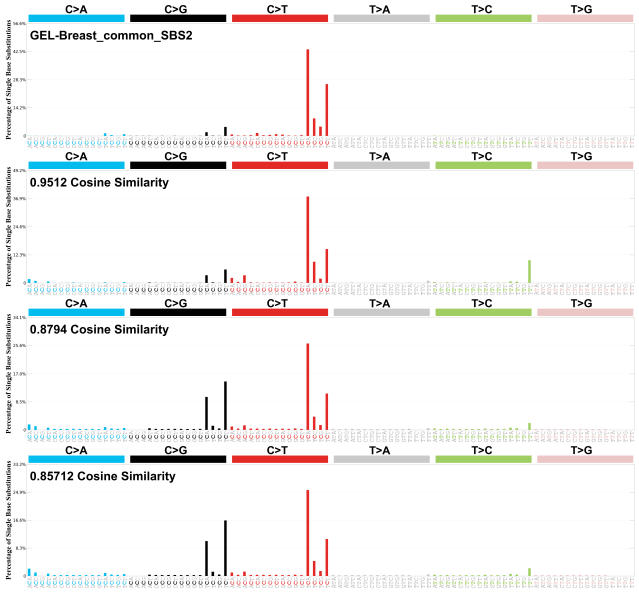


**Figure 12: 3 extracted signatures and their cosine against one of the reference signatures from Signal**

## 7.2 Pre-processing

*Injection.* After the exploration phase, we analyzed the results and concluded that our model can extract the signatures we know are there. This gives us confidence that the model is working correctly and can extract correct signatures in big datasets. This feature was designed and intended to be a method to test our model and ensure it can extract signatures correctly. The injection feature was a great addition to the framework for adding confidence and fulfilled that role well both when testing with real signatures and synthetic test signatures meant to look nothing like real signatures. These test signatures were used early on in testing to ensure the feature was working correctly and that the model could find them at the easiest level for a start before moving on to real signatures.

*Bootstrapping.* The results for bootstrapping were better than expected as seen in A1, A2, B1, and B2 11. The goal of bootstrapping the dataset is to increase the variance of the data and help the autoencoder to better comprehend and find patterns in the dataset. This makes the model more versatile by having more examples of different signatures.

*Feature filtering.* The model performed better when feature filtering was used due to helping to reduce the dimensions. Reducing the dimensions makes the dataset easier to learn for the autoencoder, and makes the signatures more accurate since it helps with the curse of dimensionality. It can be seen in plots A1, A2, B1, and B2 9, where the higher cutoffs can be seen to have an improvement over no cutoff.

*Latent.* Throughout the exploration of different parameters, it was discovered that the higher latent values gave significantly better results. This can be seen clearly in A1, A2, B1, and B2 8. The higher amount of latent helps by having more nodes to represent the data and more data when clustering afterward to find the signatures. This does result in duplicate latents, but is worth it for the overall extraction result as seen in the previously mentioned plots.

*Noise.* Adding noise to the dataset before passing it to our autoencoder improved the results as can be seen in A1, A2, B1, and B2 10. This is most likely due to learning the underlying structure and not the individual signatures. On the other hand, adding too much noise hinders the extraction process. Leading us to determine adding some noise will be the optimal solution.

## 7.3 Method Extraction

The main focus of this paper was on the pre- and post-process, but we can see that the latent part of method extraction plays a big role in the extraction of signatures. When adding noise it maybe should have either increased the epochs or lowered the learning rate, or both, as it could be not getting enough time to learn to extract signatures, as mentioned in section 5.2.2. The hyper-parameters for the method extraction were done on a dataset, without taking into consideration what would happen if we added noise to the data, or changed the latent space.

## 7.4 Post-processing

*Alpha.* As can be seen in A1, A2, B1, and B2 4 tuning this value to change the weight of these silhouette metric when determining the best clusters. It can be seen that the lower end is better for both methods, but one method clearly finds more signatures than the other.

*Cluster selection in multi run.* After exploring the different clustering selection with relation to different methods, it stands to show the new clustering of the best clusters is performing way better in A1, A2, B1, and B2 6. This is a good result since this is one of the custom-made methods used in this paper that is different from the other approaches taken in previous papers.

*Silhouette metric.* This is part of what is used for calculating the correct clusters by trying to ensure clusters are distinct. As can be seen in A1, A2, B1, and B27, the clusters found with this metric vary quite a lot depending on what clustering method is used and less based on the silhouette metric.

## 7.5 Datasets

With the addition of the Degasperi et al. paper [5] data, the model gained a great advantage over our 9th semester [6] version of the same autoencoder due to the increased performance gain with more training data.

## 7.6 Finding the Optimal Parameters

Taking in all the information gained from test runs over 50.000 combinations and careful analysis gave the results shown in 6. With all the data collected and processed, it allows us to analyze the processed data and deduct what the optimal parameters are for extracting signatures. The parameters are listed in section 8.

# 8 CONCLUSIONS

In conclusion, we have extended the SEEF framework from our 9th semester paper [6], with multiple different pre- and post-processing methods to gain insight into which processes give the best outcome.

The results we discovered during the exploration of the different combinations, were that some combinations and processes greatly affected the outcome. Striking the right balance turned out to be the most important factor when choosing what parameters to use, and the parameters for one dataset, might not be optimal for another dataset, but can give a good baseline to start from. After a thorough exploration of possible combinations, this set turned out to be the most effective parameter for post-processing: K-Means as the cluster method, Euclidean as the silhouette metric, $\alpha$ weight should be set at 0.8, and when running multiple method extraction, the optimal selection multi should be used. The most effective pre-parameters that worked both on breast datasets turned out to be: noise at 0.01, the highest latent used is 200 for PCAWG and 250 for GEL, and for bootstrapping either true or false will have very similar results. The pre-parameter most effective, that was different, was feature filtering, where for PCAWG feature filtering is best at 0.03, while for GEL it is at 0.09.

# 9 FUTURE WORK

In this section, we discuss potential future directions further development on this paper can take.

## 9.1 Two Neighboring Bases

One proposal is to extend the analysis by taking the two neighboring bases on each side of a given mutation into account, increasing it to a total of four bases in each direction. This approach considers 1,536 possible mutation contexts, calculated as $4\times4\times6\times4\times4 = 1536$. Such an expansion would allow for a more comprehensive examination of mutational patterns compared to the 96 mutations we currently explore.

One issue with the approach of 4 neighboring bases on each side is a lack of big datasets covering this mutational setup. Hence we focus on the more common 96 mutational catalog that both COSMIC and most other research papers also are based on.

| Feature Filtering | C>A | C>G | C>T | T>A | T>G | T>C |
|---|---|---|---|---|---|---|
| 0.000 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0.001 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0.003 | 4 | 3 | 0 | 1 | 0 | 6 |
| 0.005 | 4 | 7 | 0 | 6 | 3 | 14 |
| 0.007 | 5 | 10 | 0 | 12 | 8 | 14 |
| 0.009 | 6 | 13 | 1 | 14 | 13 | 15 |
| 0.010 | 8 | 13 | 4 | 16 | 14 | 16 |

**Table 4: Feature filter threshold and what mutation type are removed from the PCAWG breast dataset**

## 9.2 Mutation Type feature Filtering

One of the things we noticed while analyzing the results, was which mutations were filtered out depending on the parameter threshold is set to.

| Feature Filtering | C>A | C>G | C>T | T>A | T>G | T>C |
|---|---|---|---|---|---|---|
| 0.000 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0.001 | 0 | 1 | 0 | 0 | 0 | 0 |
| 0.003 | 4 | 4 | 0 | 4 | 0 | 7 |
| 0.005 | 4 | 10 | 0 | 9 | 1 | 14 |
| 0.007 | 6 | 12 | 0 | 13 | 7 | 14 |
| 0.009 | 9 | 13 | 1 | 13 | 12 | 15 |
| 0.010 | 11 | 13 | 1 | 16 | 13 | 16 |

**Table 3: Feature filter threshold and what mutation type are removed from the GEL breast dataset**

The mutation C>T with its neighbors has significantly more mutation than the other mutations with its neighbors, therefore getting filtered out less, while T>A and T>C are completely removed on the threshold set to 0.01 for both datasets. Therefore it would be interesting to see how much its effect on the extraction of signatures would be, if the filter threshold was set at each mutation type instead, and not the full datasets

## 9.3 Bootstrap and Method Extraction Run

Bootstrap depends heavily on the method extraction run, as Degasperi et al. ran their extraction at least 300 times [5], Alexandrov et al. ran theirs 1000 times [9], while we ran it 10 times.


# ACKNOWLEDGMENTS

We would like to extend a thank you to our dedicated supervisors, Daniele Dell'Aglio and Rasmus Froberg Brøndum, for great discussion and feedback on our project. They provided valuable knowledge and insight that contributed to a significant improvement in the quality of this paper.. We would also like to thank CLAAUDIA[4] for allowing us to use their services for training and running our models. This project would not have been possible without their services.


[4] https://www.researcher.aau.dk/contact/claaudia

## CODE AVAILABILITY

The source code is available on

https://github.com/Magnijh/Master_thesis

## TECHNICAL ASPECT

The code ran on CLAAUDIA, which is an AI cloud. The nodes most commonly used were a combination of NVIDIA t4 with 16 CPU cores and NVIDIA A10 with 16 CPU cores. The code is set up to run as multi-process, our method extraction only has one layer, therefore takeing less than a minute to run all method extraction runs, but the clustering part can take up to 2 hours to run. The highest factor to computation time for clustering is the latent dimension and if using the optimal selection multi cluster. If only using single a node to run all combination, expect more than 3 weeks to run through all of the combinations.

## GLOSSARY

**exposures** the frequency and duration of a cell's exposure to a specific mutational process. 2

**mutational catalogs** a detailed log of genetic mutations in a sample. 2

**mutational processes** a change in DNA caused by DNA damage, DNA repair, DNA replication etc.. 2

**mutational signature** a characteristic patterns of somatic mutations in cancer genomes. 2

**somatic mutation** a mutation occurring in a somatic cell and inducing a chimera. 2

# REFERENCES


[1] [n. d.]. Catalogue Of Somatix Mutations In Cancer. https://cancer.sanger.ac.uk/signatures/sbs/. Accessed: 2024-01-18.

[2] 2022. *Scipy.optimize.linear_sum_assignment*. https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.linear_sum_assignment.html

[3] Degasperi A, Amarante TD, Czarnecki J, Shooter S, Zou X, Glodzik D, Morganella S, Nanda AS, Badja C, Koh G, Momen SE, Georgakopoulos-Soares I, Dias JML, Young J, Memari Y, Davies H, and Nik-Zainal S. 2020. A practical framework and online tool for mutational signature analyses show inter-tissue variation and driver dependencies. *Nat Cancer* (2020). https://doi.org/10.1038/s43018-020-0027-5

[4] L.B. Alexandrov, S. Nik-Zainal, D.C. Wedge, P.J. Campbell, and M.R. Stratton. 2013. Deciphering signatures of mutational processes operative in human cancer. *Cell Rep* (2013).

[5] Andrea Degasperi, Xueqing Zou, Tauanne Dias Amarante, Andrea Martinez-Martinez, Gene Ching Chiek Koh, João M. L. Dias, Laura Heskin, Lucia Chmelova, Giuseppe Rinaldi, Valerie Ya Wen Wang, Arjun S. Nanda, Aaron Bernstein, Sophie E. Momen, Jamie Young, Daniel Perez-Gil, Yasin Memari, Cherif Badja, Scott Shooter, Jan Czarnecki, Matthew A. Brown, Helen R. Davies, Genomics England Research Consortium3†, Serena Nik-Zainal, J. C. Ambrose, P. Arumugam, R. Bevers, M. Bleda, F. Boardman-Pretty, C. R. Boustred, H. Brittain, M. J. Caulfield, G. C. Chan, T. Fowler, A. Giess, A. Hamblin, S. Henderson, T. J. P. Hubbard, R. Jackson, L. J. Jones, D. Kasperaviciute, M. Kayikci, A. Kousathanas, L. Lahnstein, S. E. A. Leigh, I. U. S. Leong, F. J. Lopez, F. Maleady-Crowe, M. McEntagart, F. Minneci, L. Moutsianas, M. Mueller, N. Murugaesu, A. C. Need, P. O'Donovan, C. A. Odhams, C. Patch, D. Perez-Gil, M. B. Pereira, J. Pullinger, T. Rahim, A. Rendon, T. Rogers, K. Savage, K. Sawant, R. H. Scott, A. Siddiq, A. Sieghart, S. C. Smith, A. Sosinsky, A. Stuckey, M. Tanguy, A. L. Taylor Tavares, E. R. A. Thomas, S. R. Thompson, A. Tucci, M. J. Welland, E. Williams, K. Witkowska, and S. M. Wood. 2022. Substitution mutational signatures in whole-genome–sequenced cancers in the UK population. *Science* 376, 6591 (2022), abl9283. https://doi.org/10.1126/science.abl9283 arXiv:https://www.science.org/doi/pdf/10.1126/science.abl9283

[6] Rasmussen F, Gislum C, Sinding K.R, Hansen M.J, Jensen M.V, and Kure N.E. 2023. SEEF: A Signature Extraction and Evaluation Framework. (2023).

[7] Cooper GM. 2000. The Cell: A Molecular Approach. 2 (2000).

[8] Gareth James, Daniela Witten, Trevor Hastie, Robert Tibshirani, and Jonathan Taylor. 2023. An Introduction to Statistical Learning. 1 (2023). Accessed: 2024-06-5.

[9] Alexandrov LB, Kim J, Haradhvala NJ, Huang MN, Tian Ng AW, Wu Y, Boot A, Covington KR, Gordenin DA, Bergstrom EN, Islam SMA, Lopez-Bigas N, Klimczak LJ, McPherson JR, Morganella S, Sabarinathan R, Wheeler DA, Mustonen V, PCAWG Mutational Signatures Working Group, Getz G, Rozen SG, Stratton MR, and PCAWG Consortium. 2020. The repertoire of mutational signatures in human cancer. *Nature* 578 (2020), 94–101. https://doi.org/10.1038/s41586-020-1943-3

[10] Andrew NgAndrew Ng. [n. d.]. Sparse autoencoder. 1 ([n. d.]), 19. Accessed: 2024-06-6.

[11] Serena Nik-Zainal, Helen Davies, Johan Staaf, Manasa Ramakrishna, Dominik Glodzik, Xueqing Zou, Inigo Martincorena, Ludmil B. Alexandrov, Sancha Martin, David C. Wedge, Peter Van Loo, Young Seok Ju, Marcel Smid, Arie B. Brinkman, Sandro Morganella, Miriam R. Aure, Ole Christian Lingjærde, Anita Langerød, Markus Ringnér, Sung-Min Ahn, Sandrine Boyault, Jane E. Brock, Annegien Broeks, Adam Butler, and . . . Michael R. Stratton. 2016. ntroducing whole-genome sequencing into routine cancer care: the Genomics England 100 000 Genomes Project. *Nature* 534 (2016), 47–57. https://doi.org/10.1038/nature17676

[12] PCAWG Consortium. 2020. Pan-cancer analysis of whole genomes. *Nature* 578, 7793 (2020), 82–93. https://doi.org/10.1038/s41586-020-1969-6

[13] Guangsheng Pei, Ruifeng Hu, Yulin Dai, Zhongming Zhao, and Peilin Jia. 2020. Decoding whole-genome mutational signatures in 37 human pan-cancers by denoising sparse autoencoder neural network. *Oncogene* (2020). https://doi.org/10.1038/s41388-020-1343-z

[14] Jacob Murel Ph.D. and Eda Kavlakoglu. 2024. What is dimensionality reduction? 1 (2024). Accessed: 2024-06-3.

[15] Peter Priestley, Jonathan Baber, Martijn P. Lolkema, Neeltje Steeghs, Ewart de Bruijn, Charles Shale, Korneel Duyvesteyn, Susan Haidari, Arne van Hoeck, Wendy Onstenk, Paul Roepman, Mircea Voda, Haiko J. Bloemendal, Vivianne C. G. Tjan-Heijnen, Carla M. L. van Herpen, Mariette Labots, Petronella O. Witteveen, Egbert F. Smit, Stefan Sleijfer, Emile E. Voest, and Edwin Cuppen. 2019. Pan-cancer whole-genome analyses of metastatic solid tumours. *Nature* 575 (2019), 210–216. https://doi.org/10.1038/s41586-019-1689-y

[16] Jack Trainer. 2021. Bootstrapping in Statistics. 1 (2021). Accessed: 2024-06-5.

[17] C. Turnbull. 2018. ntroducing whole-genome sequencing into routine cancer care: the Genomics England 100 000 Genomes Project. 29 (2018), 783–1078. https://doi.org/10.1093/annonc/mdy054

[18] Varun. 2020. Cosine similarity: How does it measure the similarity, Maths behind and usage in Python. 1 (2020). Accessed: 2024-06-5.

[19] Yu-Jin Zhang Yu-Xiong Wang. 2012. Nonnegative Matrix Factorization: A Comprehensive Review. *IEEE Transactions on Knowledge and Data Engineering* 25, 6591 (2012), 1353. https://doi.org/10.1109/TKDE.2012.51 Accessed: 2024-06-5.

# A CHANGES FROM LAST SEMESTER

This paper is based on continuing the framework from the last paper. Therefore in this section, we are going through the differences and the new implementation between the two papers, and how the new framework fares against the older framework.

## A.1 Methodology

This section describe and explain the processes that have been improved to extend the framework developed during the 9$^{th}$ semester [6].

*A.1.1 Pre-processing.* In the old framework, the only pre-processing that would happen was that column-wise would be normalized to sum to 1, which still happens in the new framework. Now we have multiple modules that the pre-processing can use. As explained in section 4, we have bootstrapping, feature filtering, injections, and noise.

*A.1.2 Post-processing.* The difference in the post-processing part is that in the old paper, it would cluster using K-Means, each latent space from the method extraction individual, then pick the most optimal cluster with n_cluster from one of the runs. In our new framework, before it goes to clustering, we check that the latent space does not have any empty latent, and that there are none latent duplicates. Then it goes to the clustering part, where we now can choose what kind of clustering method it should use. Currently implemented is K-Means and our own clustering method called Cosine. Afterwards you can choose if you should only pick the most optimal run from a single method extraction run or concatenate each optimal run from each method extraction run, and again run clustering on it, to get the most optimal cluster

*A.1.3 Computing Time Optimization.* In the last semester, the whole framework was implemented using only a single process, while this semester we have implemented the method extraction and clustering method run to as multi-process, by splitting each run into its own process.

| | Single | Multi | Difference |
|---|---|---|---|
| Each extract run | 8 seconds | 30 seconds | 350 % |
| Total extract run | 83 seconds | 32 seconds | 159 % |
| Each cluster run | 5 seconds | 25 seconds | 400 % |
| Total cluster run | 51 seconds | 26 seconds | 96 % |
| Total time | 134 seconds | 58 seconds | 131 % |

**Table 5: Table is based on the mean value of 5 runs, with each run running 10 method extraction runs thereafter clustering each method extraction run.**

As can be seen in table 5, both extracting and clustering individual runs take less than 10 seconds to run when running as a single process, but the problem lies in that it has 10 runs to go through. Therefore the total time becomes longer than the multi-process, and it would only get worse for the single process if the amount of method extraction increases, as it would run sequentially, while with multi-process it would run all method extraction in parallel.

## A.2 Experimental Setup

Each framework ran 100 times on both datasets breast part, with 10 method extraction runs. Results of the 100 runs from each framework, are then box-plotted, to be able to see which framework performed better.

*A.2.1 Parameters.* The old framework parameters that could be changed were only those that belonged to the method extraction. Here were used Optuna to help find the most optimal parameters, through Optuna, and observation of cosine similarity to reference signatures. The choice was made that learning rate is set at 1-e03, epochs at 500, batch size of 8, and latent dimension of 200. These method extraction parameters are used in both frameworks, to show the difference that the new pre- and post-processing has on the results

*New framework pre- and post-processing parameters.* we are using the optimal parameter that was explained in section 8

*A.2.2 Fair Comparison.* In the older framework, the evaluation is missing a check for the order of the mutation type. Therefor when doing cosine similarity it will give worse results than it should. To make it a fair comparison between the old framework and the new one, we have implemented this part in the old framework.

*A.2.3 Cosine Similarity Threshold Results.* The cosine similarity threshold steps up from 0.85 as that is what was used in our 9$^{th}$ semester paper [6], and steps up to 0.99, where two extra steps are included called best>0.95 and best>0.99, which used the Hungarian algorithm. This can remove duplicates

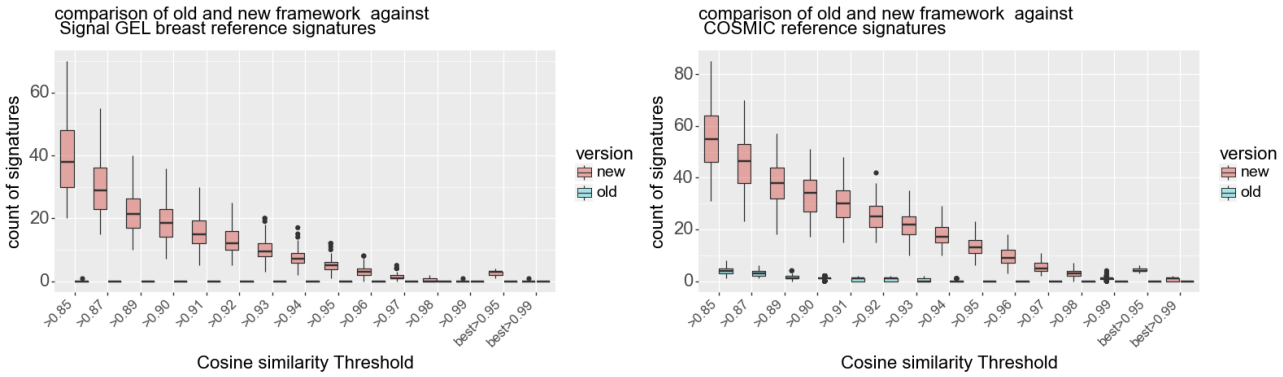


**Figure 13: comparison between the frameworks from this paper and last semester's paper on both datasets for breast**

## A.3 Experimental Results

This section, contains the results found from the experimental setup, providing a comparison between the two frameworks.

As seen in figure 13 the new framework extracts more signatures in both datasets, through all cosine similarity, for the old framework its only able to extract signatures with over 0.90 cosine similarity in PCAWG, but not higher than 0.95 cosine similarity. While the new framework is able to extract all the way up to 0.99 in PCAWG almost consistence, but in GEL its only able to extract up to 0.97 consistent. Therefore we can see that the new methods and small improvements have improved the performance of the new framework