Pre-processing and analysis of genomic data using data mining tools

Théo FIGINI
M2 Computer Science
Academic year 2024-2025

Hosting organisation : *Université des Antilles*

January, 13th, 2025 - June, 16th, 2025

Refering teacher:
Emmanuel BIABIANY

Tutor :
Wilfried SEGRETIER

June, 20th 2025

Contents

Introduction

1 Related Work

Genomics is a field that has been growing rapidly in the past few years. The advent of high-throughput sequencing technologies has made it possible to sequence the entire genome of an organism in a matter of days. This has led to an explosion of data, with the number of sequenced genomes increasing exponentially. This has created a need for new tools and algorithms to analyze this data. In this chapter, we review some of the existing tools and algorithms for analyzing genomic data.

1.1 Genomic data analysis

Random forests

Another method that has been used for genomic data analysis is random forests (RF) [2]. This method is based on the idea of ensemble learning, where multiple decision trees are trained on different subsets of the data and then combined to make a final prediction.

AI applications in genomic analysis

Many researchers have used AI techniques to analyze genomic data. For example, [1] reviews different AI techniques that have been used for genomic data analysis, including CNNs, autoencoders, etc.

More recently, the autors of [9] designed a tool based on multiple LLM backends for multi-omics analysis with minimal human intervention.

2 Problem Statement

2.1 Context

2.2 Work Environment

This project was done within the framework of the Master 2 Computer Science program at the **Laboratoire de Mathématiques Informatique et Applications (LAMIA)** of the **Université des Antilles**. The following tools and technologies were used to develop the algorithms and document the project:

- **Python**: a programming language widely used in scientific computing, data analysis, and machine learning.
- **GitHub**: a version control platform for managing the source code.
- **Jupyter Notebook**: an interactive environment for writing and executing Python code, ideal for data analysis and visualization.
- **sklearn**: a Python library for machine learning that provides various algorithms and tools for data preprocessing, model selection, and evaluation.
- LaTeX: a typesetting system used for writing technical and scientific documents.
- A personal computer running Windows 11 with a **Ryzen 7 6800H** processor and **32 GB** of RAM.

2.3 Objectives

3 Genomic data

Genomic data refers to the information contained in the DNA of an organism, which includes the sequences of nucleotides (A, T, C, G). These are gathered through various methods, such as DNA sequencing, and can be used to study the genetic makeup of organisms, identify genetic variations, and understand the relationships between different species. The Institut Pasteur has a large collection of genomic data, including the genomes of various strains of Tuberculosis (TB), which is a major public health concern worldwide.

3.1 The FASTA format

The FASTA format is a text-based format for representing nucleotide or protein sequences. It is widely used in bioinformatics to store and exchange sequence data. A FASTA file consists of one or more sequences, each represented by a header line starting with a greater-than symbol (`>`), followed by the sequence itself on the next line. The header line can contain additional information about the sequence, such as its identifier, description, or source. An example of a FASTA file is shown in Figure 1.

```
>sequence_id
ATCGATCGATCGATCGATCGATCGATCG
>another_sequence_id
GCTAGCTAGCTAGCTAGCTAGCTAGCTA
```

Figure 1: Example of a FASTA file containing two sequences.

The FASTA format is simple and easy to read, making it suitable for storing large amounts of sequence data. It is also compatible with many bioinformatics tools and software, allowing for easy analysis and manipulation of sequence data.

4 Clustering genomic data

4.1 Introduction to K-means

K-means clustering is a widely used unsupervised learning algorithm for partitioning a dataset into k distinct clusters. It works by iinitializing k cluster centers, then iteratively assigning data points to the nearest cluster center and updating the centers based on the distances to the assigned points (usually using Euclidean distance). The algorithm continues until the cluster centers stabilize or a maximum number of iterations is reached.

Results can vary based on the initial choice of cluster centers, which can lead to different clustering outcomes. The quality of clusters is typically evaluated using at least two criteria:

- **Intra-cluster variance** : the average distance between points within the same cluster.
- **Inter-cluster variance** : the average distance between centers of different clusters.

4.1.1 K-means Algorithm

The K-means algorithm consists of the following steps:

1. Initialize k cluster centers randomly.
2. Assign each data point to the nearest cluster center.
3. Update the cluster centers by calculating the mean of the points assigned to each cluster.
4. Repeat steps 2 and 3 until convergence (i.e., when cluster centers do not change significantly).
5. Optionally, evaluate the clustering quality using metrics like inertia or silhouette score.

4.2 Silhouette Score

The silhouette score is a metric used to evaluate the quality of clustering. It measures how similar an object is to its own cluster compared to other clusters. The silhouette score is calculated for each data point and ranges from -1 to 1. The formula for the silhouette score of a point i is:

$$s(i) = \frac{b(i) - a(i)}{\max(a(i), b(i))} \tag{1}$$

Where $a(i)$ is the average distance from point i to all other points in the same cluster, and $b(i)$ is the average distance from point i to all points in the nearest cluster. A higher silhouette score indicates better clustering quality.

4.3 Principal Component Analysis (PCA)

Principal Component Analysis (PCA) is a dimensionality reduction technique that transforms a dataset into a new coordinate system, where the axes (principal components) are ordered by the amount of variance they capture from the data. It was first introduced by **Karl Pearson** in 1901[6] and later popularized by **Harold Hotelling** in 1933[4]. PCA is widely used in data analysis, PCA is particularly useful for reducing the dimensionality of large datasets while preserving as much variance as possible. The steps involved in PCA are:

1. **Standardize the data**: Center the data by subtracting the mean and scaling it to unit variance.
2. **Compute the covariance matrix**: Calculate the covariance matrix of the standardized data.
3. **Compute the eigenvalues and eigenvectors**: Find the eigenvalues and eigenvectors of the covariance matrix.
4. **Sort the eigenvalues and eigenvectors**: Sort the eigenvalues in descending order and arrange the corresponding eigenvectors accordingly.
5. **Select the top k eigenvectors**: Choose the top k eigenvectors corresponding to the largest eigenvalues, where k is the desired number of dimensions to retain.
6. **Transform the data**: Project the original data onto the new coordinate system defined by the selected eigenvectors.

Note: an eigenvector or characteristic vector is a vector that does not change direction during a linear transformation, but may change in magnitude. The eigenvalue is the factor by which the eigenvector is scaled during the transformation.

In this project, we use the integrated PCA implementation from the *sklearn* library, which provides a convenient way to perform PCA on datasets.

4.4 Preprocessing Genomic Data

Our data is a collection of Tuberculosis (TB) genomes, where each file contains the proteins and their respective nucleotide sequences in FASTA format, for a total of 7057 files. TODO: parler de l'institut Pasteur? The goal is to cluster these genomes based on their protein sequences and find ways to distinguish between different TB strains.

We started by creating a dataset containing the number of times each protein appears in each genome. This was done by parsing the FASTA files and counting the occurrences of each protein sequence, we then removed the outliers, namely "hypothetical proteins" and "putative proteins", which are not well characterized and do not provide useful information for clustering. The resulting dataset is a CSV file of 7057 columns plus 1 column for the protein names and 2732 rows plus 1 row for the header.

Considering the size of the dataset, we used Principal Component Analysis (PCA) set to 95% variance to reduce the dimensionality of the data while preserving as much variance as possible. PCA is a technique that transforms the data into a new coordinate system, where the first principal component captures the maximum variance, the second captures the second maximum variance, and so on.

We then calculated the silhouette score for different values of k, ranging from 2 to 10, to determine the optimal number of clusters. The silhouette score was calculated using the ‘silhouette_score‘ function from the ‘sklearn.metrics‘ module, which takes the data and the cluster labels as input. The optimal number of clusters is the one that maximizes the silhouette score.

4.4.1 First Results

The initial results showed that the silhouette score was highest for $k = 9$, indicating that this was the optimal number of clusters. However, the clusters were not well separated or some points are alone in their cluster, as seen in Figure TODO:figure:clusters_k9. Which suggests that the data may not be well suited for clustering with K-means, or that the features used for clustering are not informative enough.

TODO: figure clusters_k9

We also tried without applying PCA but achieved similar results, we also observed a fewer points than expected as seen in Figure TODO:figure:clusters_k9_no_pca, after reviewing the data, we found that there is significant overlap between the points.

TODO: figure clusters_k9_no_pca

So we switched to 3D visualization to better understand the clusters,

4.5 New Approach using DBSCAN and HDBSCAN

To address the limitations of K-means clustering, we explored alternative clustering algorithms, specifically **DBSCAN** and **HDBSCAN**.

DBSCAN (Density-Based Spatial Clustering of Applications with Noise) is a density-based clustering algorithm that groups points that are close together based on a distance measurement and a minimum number of points. It is particularly effective for datasets with varying densities and can identify noise points that do not belong to any cluster.

HDBSCAN (Hierarchical Density-Based Spatial Clustering of Applications with Noise) is an extension of DBSCAN that builds a hierarchy of clusters and allows for more flexible clustering by varying the density threshold. It can also handle varying cluster shapes and sizes.

4.5.1 Principle

DBSCAN

DBSCAN works by defining a neighborhood around each point based on a distance metric (usually Euclidean distance) and a minimum number of points required to form a dense region. The algorithm proceeds as follows:

1. For each point in the dataset, find its neighbors within a specified radius ε.
2. If the number of neighbors is greater than or equal to the minimum number of points, a new cluster is formed.
3. The algorithm recursively expands the cluster by including all points that are reachable from the initial point.
4. Points that do not belong to any cluster are classified as noise.

HDBSCAN

HDBSCAN builds on the principles of DBSCAN by introducing a hierarchical clustering approach. It is more robust to varying densities and can identify clusters of different shapes and sizes. The algorithm works as follows:

1. Compute the mutual reachability distance between points, which takes into account the distance to the nearest neighbor.
2. Build a minimum spanning tree (MST) based on the mutual reachability distances.
3. Extract clusters from the MST by varying the density threshold, resulting in a hierarchy of clusters.
4. Assign points to clusters based on their membership in the hierarchy, allowing for flexible clustering.

4.5.2 Results

4.6 Trying an autoencoder

5 Using CNNs to identify TB Strains

5.1 Introduction to CNNs

The fondamental idea behind **Convolutional Neural Networks (CNNs)** was introduced by Kunihiko Fukushima[3] in 1980 and later popularized by Yann LeCun in the 1990s with the LeNet architecture[5]. CNNs are a class of deep learning models specifically designed for processing structured grid data, such as images. They are particularly effective for tasks like image classification, object detection, and segmentation.

CNNs work by applying convolutional filters to the input data, which allows them to learn spatial hierarchies of features. It shares some similarities with the more conventional neural networks, but there is at least two different types of layers that are specific to CNNs:

- **Convolutional layers**: These layers apply convolutional filters to the input data, extracting local features that are invariant to translation. The filters slide over the input data, computing dot products and producing feature maps.
- **Pooling layers**: These layers downsample the feature maps, reducing their spatial dimensions while retaining important information. Common pooling operations include max pooling and average pooling.

The architecture of a CNN typically consists of multiple convolutional and pooling layers followed by fully connected layers. The convolutional layers learn to extract features from the input data, while the fully connected layers perform the final classification or regression task. The training process involves optimizing the weights of the filters and fully connected layers using backpropagation and a loss function, such as cross-entropy for classification tasks.

5.2 Building the dataset

For this experiment, each genome was converted into an image where each pixel is determined by a combination of three metrics of following metrics:

- **Chargaff**
- **Composition**
- **Diversity**
- **Skew**
- **Nuclescore**

We get ten combinations of these metrics, each represented by a different color channel in the image. We applied this conversion at different resolutions, from 4px (2x2) to 100px (10x10), the resulting images put into their corresponding class based on the strain they belong to. We have the following classes:

- **M** with 185 samples.
- **East Asian** with 1651 samples.
- **East-African Indian** with 267 samples.
- **Euro-American** with 4409 samples.
- **Indo-Oceanic** with 259 samples.

5.2.1 Initial testing

The resulting dataset is highly imbalanced, with some classes having significantly more samples than others. This can lead to biased models that perform poorly on minority classes. We decided to train our CNN on the dataset as-is, without any preprocessing or balancing techniques, to see how it performs on the imbalanced dataset.

5.3 Mitigating class imbalance

We have multiple ways of addressing the class imbalance issue in our dataset, including:

- Cross validation
- Under-sampling
- Data augmentation
- Combining under-sampling and data augmentation

We will explore each of these techniques in the following sections and compare the results to see which one yields the best performance.

5.3.1 Cross validation

Cross validation is a technique used to evaluate the performance of a model by splitting the dataset into multiple subsets, or folds. The model is trained on a subset of the data and tested on the remaining data, and this process is repeated for each fold. This allows for a more robust evaluation of the model's performance, as it reduces the risk of overfitting to a specific subset of the data.

K-fold cross validation

K-fold cross validation is a specific type of cross validation where the dataset is divided into k equally sized folds. The model is trained on $k-1$ folds and tested on the remaining fold, and this process is repeated for each fold. The final performance is averaged over all folds to obtain a more reliable estimate of the model's performance. The graphical representation of K-fold cross validation is shown in Figure 2.

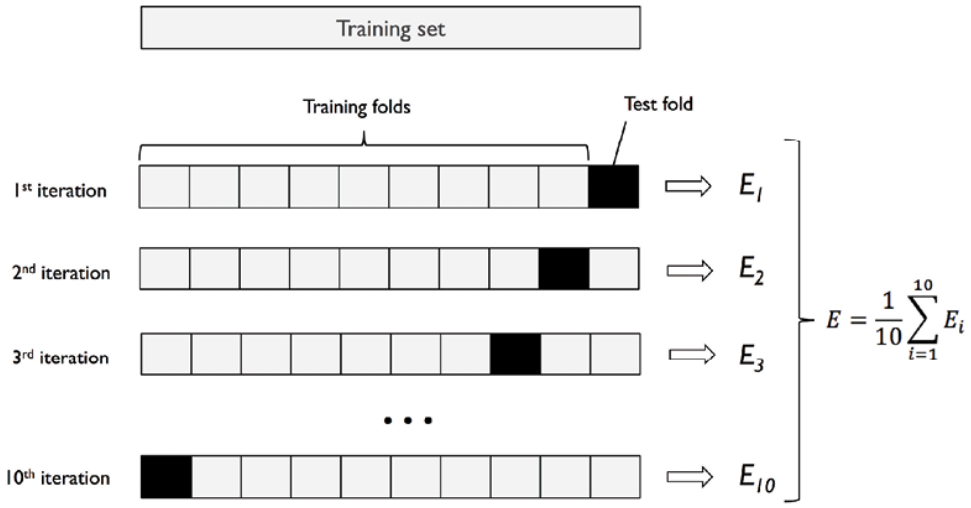

Figure 2: How K-fold cross validation works.[7]

Stratified K-fold cross validation

Stratified K-fold cross validation is a variation of K-fold cross validation that ensures that each fold has the same proportion of samples from each class. This is particularly useful for imbalanced datasets, as it ensures that each fold has a representative sample of each class. Even if this method is recommended for imbalanced datasets, we did not have enough time to implement it, so we used the standard K-fold cross validation.

Remaking the dataset

During our testing, we realized that we were testing our model with images the model was trained on, which is not a good practice. We decided to split our dataset into a training set and a test set, with 90% of the samples in the training set and 10% in the test set. The files are separated into two folders to avoid any overlap between the training and test sets.

5.3.2 Under-sampling

5.3.3 Data augmentation

SMOTE

Data augmentation is a technique used to artificially increase the size of a dataset by applying various transformations to the existing data. This can help improve the model's performance by providing more diverse training samples and reducing overfitting. in our project, we used SMOTE (Synthetic Minority Over-sampling Technique) to generate synthetic samples for the minority classes in our dataset.

SMOTE works by creating new samples by interpolating between existing samples in the feature space. It generates synthetic samples by selecting a minority class sample, finding its nearest neighbors, and creating new samples by interpolating between the selected sample and its neighbors. This helps to balance the dataset by increasing the number of samples in the minority classes, which can improve the model's performance on those classes.

New samples are generated by taking a random sample from the minority class and finding its nearest neighbors in the feature space. The new sample is then created by interpolating between the selected sample and its neighbors, as shown in Figure 3. It uses the following formula:

$$x_{new} = x_i + \lambda(x_j - x_i) \tag{2}$$

where x_{new} is the new sample, x_i is the selected sample, x_j is one of its nearest neighbors, and λ is a random number between 0 and 1. This process is repeated until the desired number of samples is generated for the minority class.

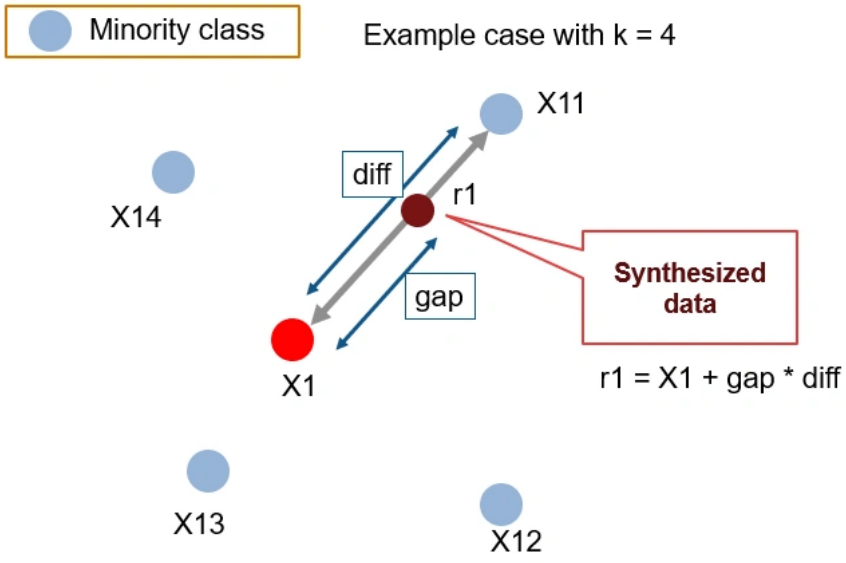

Figure 3: Illustration of the SMOTE algorithm.[8]

Results

5.3.4 Combining techniques

Results

Conclusion

Conclusion

Bibliography

[1] Claudia Caudai et al. "AI applications in functional genomics". In: *Computational and Structural Biotechnology Journal* 19 (2021), pp. 5762–5790. ISSN: 2001-0370. DOI: https://doi.org/10.1016/j.csbj.2021.10.009. URL: https://www.sciencedirect.com/science/article/pii/S2001037021004311.

[2] Xi Chen and Hemant Ishwaran. "Random forests for genomic data analysis". In: *Genomics* 99.6 (2012), pp. 323–329. ISSN: 0888-7543. DOI: https://doi.org/10.1016/j.ygeno.2012.04.003. URL: https://www.sciencedirect.com/science/article/pii/S0888754312000626.

[3] Kunihiko Fukushima. "Self-organizing Neural Network Models for Visual Pattern Recognition". In: *Plasticity of the Central Nervous System*. Ed. by Keiji Sano and Shozo Ishii. Vienna: Springer Vienna, 1987, pp. 51–67. ISBN: 978-3-7091-8945-0.

[4] Harold Hotelling. "Analysis of a complex of statistical variables into principal components." In: *Journal of educational psychology* 24.6 (1933), p. 417.

[5] Y. Lecun et al. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324. DOI: 10.1109/5.726791.

[6] Karl Pearson. "LIII. On lines and planes of closest fit to systems of points in space". In: *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science* 2.11 (1901), pp. 559–572. DOI: 10.1080/14786440109462720. eprint: https://doi.org/10.1080/14786440109462720. URL: https://doi.org/10.1080/14786440109462720.

[7] Sebastian Raschka and Vahid Mirjalili. "Python machine learning second edition". In: *Birmingham, England: Packt Publishing* (2017), p. 192.

[8] SWASTIK. *Smote for Imbalanced Classification with Python, Technique*. Oct. 2020. URL: https://www.analyticsvidhya.com/blog/2020/10/overcoming-class-imbalance-using-smote-techniques/.

[9] Juexiao Zhou et al. "An AI Agent for Fully Automated Multi-omic Analyses". In: *bioRxiv* (2024). DOI: 10.1101/2023.09.08.556814. eprint: https://www.biorxiv.org/content/early/2024/01/05/2023.09.08.556814.full.pdf. URL: https://www.biorxiv.org/content/early/2024/01/05/2023.09.08.556814.