Interim Report

An extendible clustering package for bioinformatics

By Karan Alreja

Supervised by Susnas Sourjah

1.Introduction

1.1 Aim and objective of the project

Bioinformatics is an interdisciplinary field at the intersection of biology and computer science. It originated from the understanding that both disciplines deal with large amounts of complex data that need systematic/algorithmic organization and analysis. In biology, data such as DNA sequences, protein structures, and gene expression profiles contain important information about living organisms, while computer science offers tools and methods to process this data efficiently. (Jeff Gauthier, 2019)

The role of computers in bioinformatics is important because of their ability to handle vast datasets, perform rapid computations, and find patterns that are otherwise difficult to find manually. By using algorithms, visualization techniques, and modeling, bioinformatics enables researchers to gain insights into biological phenomena, such as understanding genetic variations, predicting protein functions, and evaluating evolutionary relationships (Lesk, 2019). This combination of computational power and biological research forms the foundation of modern bioinformatics.

This project aims to use computational resources to develop a useful and extensible software application for clustering biological datasets. By integrating clustering algorithms, datasets, and visualization techniques within a plug-in framework, the software ensures adaptability and scalability.

This allows researchers to extend the system in vital ways, such as incorporating new datasets, visualization methods, and integrating additional clustering algorithms. Which creates use of new functionalities, providing a versatile solution to meet the ever-increasing needs of bioinformatics research.

1.2 Motivation for the Project

Bioinformatics has emerged as an important topic in modern biological research, where computational tools are crucial for analyzing complex biological datasets, such as gene expression profiles and protein sequences (Lesk, 2019). However, existing bioinformatics systems often present significant barriers to entry due to their complexity and the specialized knowledge required to use them effectively (Steiper, 2005). Researchers frequently encounter challenges with these tools, as they require advanced programming skills or specific domain expertise to customize or extend their functionalities, as highlighted in tools like **Bioconductor** (Gentleman, 2008). These barriers can limit the accessibility and usability of bioinformatics platforms for those without specialized backgrounds, slowing down the research process. (Anna Niarakis, 2022).

Prominent systems such as **GenePattern** (Reich, 2006) , **Cytoscape** (Shannon, 2003), and **Galaxy** (Goecks, 2010), provide significant features for data analysis and visualization, but they frequently necessitate a high learning curve for customization and integration of new functionality. Many of these systems use complicated scripting languages or highly specialized interfaces that require users to have extensive technical knowledge. As a result, users are often restricted by the scope of the pre-existing toolsets and must invest significant time in learning the underlying frameworks.

In contrast, the proposed software aims to reduce this knowledge barrier by providing an intuitive **graphical user interface (GUI)** and an **extensible plug-in framework**. The use of a GUI will simplify user interactions, making it easier for researchers to use clustering algorithms, upload new datasets, and integrate new visualization techniques without needing to understand underlying code. Furthermore, the plug-in architecture will ensure that the software can be easily expanded in vital ways. This approach will dramatically reduce the technical hurdles and offer a more flexible, user-friendly experience, allowing researchers to customize and extend the software according to their individual needs.

This software will allow a broader audience of academics to do advanced analysis on biological data without requiring substantial programming expertise. With a focus on adaptability, scalability, and usability, the project seeks to provide a valuable tool for bioinformatics researchers, making data analysis more accessible and effective. (Spjuth, 2007)

1.3 Technology Choices

*Programming Language: Java*

**Why Java?**

* **Platform Independence:** Java’s "write once, run anywhere" philosophy ensures the application can work on a variety of systems without requiring significant modifications. This is essential for ensuring accessibility across diverse user environments​.
* **Rich Ecosystem for Plug-in Architectures:** Java's object-oriented structure and widespread libraries (like the Java Plug-in Framework) makes it a lucrative choice for building extensible software systems, a core aspect of the project. (Shannon, 2003)
* **Community Support and Reliability**: Java has a mature ecosystem, extensive documentation, and robust community support. Its proven stability in enterprise and academic projects makes it a reliable choice for bioinformatics applications​ (TIOBE, n.d.)

**Comparison to Alternatives:**

* **Python:** While Python is widely used in bioinformatics due to libraries like Biopython, it is interpreted, which can lead to slower execution times for large-scale or real-time applications. Java, being compiled, offers better performance for computation-heavy tasks.
* **C++:** Though faster in execution, C++ has a steeper learning curve and lacks the built-in support for GUI development and cross-platform functionality provided by Java.

*User Interface Design: JavaFX / Figma*

**Why JavaFX?**

* **Integration with Java:** JavaFX is native to Java, making it easier to develop cohesive applications where the GUI and backend work seamlessly.
* **Rich GUI Capabilities:** JavaFX supports advanced visualizations, animations, and a wide range of controls, which are ideal for presenting complex data such as biological clustering outputs.
* **Ease of Customization:** It allows for the creation of modern, dynamic interfaces, essential for usability in bioinformatics tools.

**Why Figma?**

* Figma can be used for prototyping and designing the user interface before implementation, ensuring a user-centered approach. This reduces the risk of usability issues during the development phase.

**Comparison to Alternatives:**

* **Tkinter (Python):** Although simpler, Tkinter lacks the sophistication needed for modern GUIs and isn’t as suitable for high-quality, data-intensive visualizations.
* **Swing (Java):** While Swing is another Java-based option, it is considered outdated compared to JavaFX, which offers better functionality and design flexibility.

1.4 List of Key Milestones (Timeline)

1.4.1 Term 1

Week 5-6: Software Design and Initial Implementation

* Worked on designing the software architecture, focusing on modularity and extensibility.
* Work on converting clustering algorithms, such as K-Means, into executable code.
* Use small-scale test data to validate early implementations of clustering algorithms.

Week 7-8: Biological Data Integration and Interface Prototyping

* Set up and parse biological datasets, such as SOFT files (GDS3310\_full.soft, GDS4794\_full.soft). Validate the parsed data structure and its compatibility with clustering algorithms.
* Develop an initial prototype for the user interface using Figma, focusing on functionality like algorithm selection, data parsing, and visualization placeholders.
* Start integrating clustering algorithms with parsed datasets to ensure proper interaction.

1.4.2 Term 2

Week 3-4: Expanded Implementation and Data Integration

* Improve the implementation of clustering algorithms, ensuring efficiency and accuracy when handling larger datasets.
* Finalize parsing techniques and ensure seamless interaction between datasets and clustering modules.
* Develop functionality to switch between different datasets and algorithms within the interface.

Week 5-6: User Interface and Visualization

* Transition from the Figma prototype to JavaFX for a fully functional and interactive user interface.
* Implement visualization techniques to display clustering results meaningfully, such as dendrograms for hierarchical clustering or scatter plots for K-Means clusters.
* Integrate user options to add new algorithms or datasets dynamically.

Week 7-8: Bug Fixing, Testing, and Extensions

* Conduct comprehensive testing to identify and resolve bugs, focusing on edge cases like incomplete datasets or outliers.
* Implement extensions such as additional clustering algorithms or dataset formats if time permits.
* Optimize software performance, ensuring compatibility and usability across different environments.

2.Background and Theory

In the development of bioinformatics software, several key theories and methodologies have shaped the direction of this project. Drawing from principles of object-oriented software engineering, bioinformatics, and computational biology, the following literature provides a strong foundation for the decisions made in the design and implementation of the software. I will explain their importance and how they have helped me shape this project.

2.1 Books:

**Design Patterns: Elements of Reusable Object-Oriented Software** (Erich Gamma, 1994)and **Eclipse Plug-ins, Third Edition** (Eric Clayberg, 2008) have been important for shaping my understanding of software architecture, especially for plug-in frameworks and modularity. These books are the industry standard in their respective fields for object-oriented design patterns and how they can be applied to create reusable, flexible, and scalable software. I haven’t read the book exhaustively but I keep revisiting them to ensure that I am following proper Software Engineering (SE) principles. The concepts covered, such as the use of abstract classes, interfaces, and the principles of loose coupling and high cohesion, helped me learn and improve the way I make the software, ensuring maintainability and extensibility. **Eclipse Plug-ins** has been very important for reference on how to design a modular system that allows for easy integration of new functionalities, which is fundamental to my bioinformatics project. By frequently referring to these texts, I try to apply the best SE principles possible.

**Effective Java** (Bloch, 2017)has good insights into best practices for Java programming, espically in terms of object-oriented design and optimization techniques. It provides concrete guidelines for writing clean, efficient, and maintainable code, which are essential for developing robust software. I have focused on the sections concerning concerning memory management, performance optimization, and the use of common Java libraries. Bloch’s emphasis on effective use of Java’s features, such as collections, concurrency, and exceptions, is beneficial for my project so that it operates smoothly and efficiently. This ensures that the software is functional and maintainable.

**Building Bioinformatics Solutions** (Bessant, 2009)has been a big help for my understanding of bioinformatics tools and methods, which are important for my project. The book covers a wide range of topics related to bioinformatics, such as, handling biological datasets, apply clustering algorithms, and integrate databases and visualization tools. This book taught me crucial topics including the several types of datasets commonly used in bioinformatics, such as gene expression data, protein sequences, and microarray data. It also helped me understand the different algorithms commonly employed, such as K-means clustering and hierarchical clustering, which I am using in my project. Additionally, it provided a foundation for understanding the role of databases and visualization tools in bioinformatics, offering guidelines on how to integrate these components effectively. Even though I have not read every chapter, Building Bioinformatics Solutions has been essential in shaping my understanding of what bioinformatics software should do and how it can be developed to handle complex biological data efficiently.

2.2 Articles/Journals:

**The Central Dogma** (CRICK, 1970) established fundamental concepts in molecular biology, particularly the processes of transcription and translation, which convert DNA into RNA and then into proteins. This fundamental concept is crucial for understanding bioinformatics datasets, particularly those containing genetic and protein sequences. This study provided a conceptual framework for analyzing and interpreting biological data, such as gene expression patterns and protein sequences, which are critical in bioinformatics research, by recognizing how genetic information moves throughout a cell. This has helped me understand the various types of data utilized in bioinformatics and how they interact in biological systems

**A Brief History of Bioinformatics** (Jeff Gauthier, 2019)helped shape my understanding of bioinformatics, its origins, and its goals. The article discusses the development of the field, starting with the early days of computational biology and its progression into the sophisticated tools and systems used today. This historical context helped me understand bioinformatics as an interdisciplinary field, bringing together biology, computer science, and mathematics to address complex biological questions. It emphasised the importance of computational tools in processing and analyzing biological data, which laid the foundation for my approach in developing a software system that integrates these elements to facilitate bioinformatics research.

**Cytoscape: A Software Environment for Integrated Models of Biomolecular Interaction Networks** (Shannon, 2003)and **Galaxy** (Goecks, 2010)offer important understandings of bioinformatics software systems. These papers describe widely used bioinformatics platforms that provide powerful features for data analysis and visualization. Cytoscape is known for visualizing molecular interaction networks, while Galaxy provides a web-based platform for data analysis and workflow management. These have guided my understanding of the types of functionalities a bioinformatics software system should have, especially when it comes to integrating data analysis with visualization techniques. Both papers highlight the importance of creating user-friendly platforms that allow researchers to work with complex biological data, which directly influences the design of my software system’s interface and extensibility.

**Addressing Barriers in Bioinformatics** (Anna Niarakis, 2022)discusses obstacles in bioinformatics software, such as comprehensiveness, accessibility, reusability, interoperability, and reproducibility. This paper identified common barriers that users face when working with bioinformatics tools, particularly for those without much programming knowledge. This paper's findings have helped shape the design of my software, particularly in terms of making the platform more accessible, expandable, and user-friendly. Understanding the barriers stated in the study, I prioritized simplifying the user interface and providing a plug-in structure to make the software flexible to varied user needs, which matches with the goal of enhancing bioinformatics software usability.

2.3 Websites:

**Baeldung’s article on the K-means Clustering Algorithm** (Ali Dehghani, 2024)was very helpful in my implementation of K-means clustering. The article provided clear insights into how K-means works, and practical guidance was essential for implementing the algorithm correctly in the context of bioinformatics data. The article helped clarify key details about choosing the right distance metric, managing datasets, and optimizing the clustering process, all of which are critical for achieving meaningful results in biological data analysis.

3.Implementation so far

3.1 Parsing Biological Datasets

Parsing biological datasets is essential in bioinformatics software as it allows raw data, often from diverse sources, to be transformed into a structured form that can be analyzed. Different formats like FASTA, CSV, or SOFT are common in bioinformatics, with each requiring specific parsing strategies to extract the relevant biological information.

For this project, I have chosen to parse SOFT files from the NCBI repository, specifically the datasets *GDS3310\_full. soft* and *GDS4794\_full.soft*. These files are often used in gene expression studies and contain structured data about gene expression levels, making them highly relevant for clustering analysis.

To achieve this, I implemented a parsing mechanism that converts the data into a List<String, double> format, where each entry corresponds to a GSM sample ad gene along with its expression values. The data is then converted into a class, Entry, which allows for a structured representation of each dataset in the software. This not only ensures that the data is correctly handled and mapped but also provides an easy-to-manage format for further analysis, especially for the clustering algorithms used later. Parsing is needed to ensure that the biological data is properly integrated into the software workflow, and it directly influences the accuracy and efficiency of the downstream processing steps, such as clustering.

3.2 K-Means Clustering

K-means clustering is a popular unsupervised machine learning approach used mostly in bioinformatics to categorize big datasets based on feature set similarities(Quan Zou, 2020).The KMeansClustering class, which implements the K-means algorithm, is the primary class in this project that does clustering. This class partitions data into clusters based on the distances between data points and centroids, discussed below.

The algorithm starts by generating a random centroid from the dataset and then assigns data points to the nearest centroid. After each assignment, the centroids are recalculated based on the mean value of the assigned points, and the process repeats until the centroids stabilize.

The K-means algorithm is especially important to this project because it can handle big, multidimensional datasets such as gene expression data, where discovering natural groupings can yield useful insights. The software uses K-means clustering to help researchers uncover relevant patterns in biological data, allowing them to analyze gene expression profiles and other information. The usage of centroids in this context ensures that the produced clusters reflect the underlying data structure, allowing for more effective biological insights and data analysis.

3.2.1 Calculating distance

A key component in clustering algorithms is the calculation of the distance between data points, particularly in centroid-based methods like K-means. For this, I implemented a Distance class, which is extended by the EuclideanDistance class to compute the Euclidean distance between data points. The Euclidean distance metric is commonly used in clustering due to its simplicity and effectiveness in multidimensional spaces, such as gene expression profiles (Quan Zou, 2020) This class calculates the distance between a centroids -> data points and data point -> data point, which is crucial for determining the placement of points within clusters.

The choice of Euclidean distance conforms with standard practices in bioinformatics clustering because it offers a straightforward and computationally efficient way to evaluate how similar or dissimilar samples are. The ability to compute distances efficiently is critical to the K-means algorithm, as it directly impacts the accuracy of cluster formation and the overall performance of the software.

3.3 Interface Design and Prototype

A user-friendly interface is essential for making bioinformatics software accessible, particularly for researchers who may not have extensive technical knowledge (Shannon, 2003). Bioinformatics tools can be complex, so simplifying user interactions is crucial for broadening accessibility and improving user experience.

For the prototyping phase, I used **Figma** due to its ease of use and rapid design capabilities. Figma allowed me to create an interactive prototype that visualizes how the software will allow users to add algorithms, select visualization tools, and choose parsing methods for their datasets. While this is just an early prototype, it serves as a foundation for refining the final design.

The prototype offers:

* **Adding new algorithms** to extend the software's analysis capabilities.
* **Choosing visualization methods** to customize data presentation.
* **Selecting parsing methods** to work with different dataset formats.

Next term, I plan to implement the interface in **JavaFX**, which will provide a robust framework for building dynamic, interactive desktop applications. JavaFX's flexibility and support for complex UI elements make it ideal for creating a fully functional interface that integrates well with the software’s underlying modular architecture (Erich Gamma, 1994).

4.Software Engineering

4.1 Software Design

This is our use case diagram so far:   
A diagram of a company

Description automatically generated

This is our sequence diagram for a typical use case of the user, who feeds in a file:

A diagram of a function

Description automatically generated with medium confidence

This is our class diagram:

A screenshot of a computer

Description automatically generated

Our Distance class follows a Strategy pattern since we can choose the type of distance we want.

I have discussed the plans to implement different patterns in the code quality section.

4.2 Testing

As the project advances, testing will play a vital role in ensuring that all components integrate seamlessly and function as intended on a larger scale. Currently, I have employed basic testing methods to validate individual components and their interactions:

* **Main Class Testing**: The main class serves as a temporary test environment, allowing me to simulate scenarios and verify that features like data parsing, distance calculations, and clustering behave as expected.
* **Debugger Usage**: Extensive use of debugging tools has been essential for identifying and resolving issues during development. This iterative process has helped ensure the accuracy of calculations and the proper functioning of the algorithmic logic.

Given the modular and extensible nature of the project, thorough testing strategies will be crucial in the next phase. This will involve systematic unit tests for individual components, integration tests to verify interoperability, and performance evaluations to ensure scalability. These efforts will align with the overall goal of building a reliable and robust bioinformatics tool capable of handling diverse datasets and plug-ins efficiently.

4.3 Code Quality

Ensuring high-quality code is essential for building a maintainable and extensible software project. Here’s how I’ve approached it so far:

**Documentation**

* **Javadoc Implementation**: I’ve implemented Javadoc for all classes and methods, which ensures that the code is well-documented and easily understandable. This not only helps future collaborators but also allows the codebase to remain accessible and organized.
* **Comments**: Ample inline comments accompany the code to explain logic, clarify complex sections, and enhance overall readability.

**Design Patterns**

* **Inheritance for Reusability**: The EuclideanDistance class extends a generic Distance class. This follows the inheritance principle, promoting reusability and modularity in the code.
* **Factory Pattern (Next term)**: To manage plug-ins for different algorithms, datasets, or visualizations in the future, I plan on implementing a Factory design pattern. This would encapsulate the creation logic and help manage plug-ins dynamically.
* **Strategy Pattern (Next term)**: For clustering algorithms, the Strategy pattern could be helpful. It will be used to define a family of algorithms (e.g., K-means, hierarchical clustering) and make them interchangeable without altering the code that uses them.
* **Observer Pattern (Next term)**: As the software grows, adding event-driven features such as updates when new plug-ins or datasets are added will benefit from the Observer pattern.

**Code Practices**

* **Modularity**: Classes and methods are designed with single responsibilities to enhance maintainability.
* **Debugging and Testing**: Extensive use of debugging tools has helped verify the correctness of the implementation, with plans for more structured testing in later stages.

4.4 Version Control

For version control, I chose Git and OneDrive for managing different aspects of my project.

**Git**

* **Purpose**: Git is the primary version control system I use for managing my project code and diary entries. It allows me to track changes over time, revert to earlier versions if needed, and maintain a clear history of modifications.

**Why Git?**

* + **Preventing Progress Loss**: By regularly committing my changes, Git acts as a safeguard against losing progress due to errors or system failures.
  + **Branching and Merging**: Git's branching feature allows for changes without risking the integrity of the main codebase.
  + **Documentation and Traceability**: By committing with descriptive messages, I maintain a detailed and chronological record of the project's evolution.

**OneDrive**

* **Purpose**: OneDrive serves as a repository for storing technical documents, such as reports, design notes, and other resources. It provides an additional layer of redundancy to ensure nothing is lost.
* **Why OneDrive?**
  + **Ease of Access**: Cloud storage ensures that my documents are accessible from anywhere, making it easier to work across different devices.
  + **Automatic Backup**: OneDrive’s syncing feature ensures that my files are always backed up, reducing the risk of accidental data loss.

By combining Git for version control with OneDrive for document storage, I ensure that both my code and technical resources are organized, accessible, and protected against loss.

4.Reflections

4.1 Reflections on project plan

This term focused on developing proof-of-concept programs, establishing a foundation for the final software. Many design patterns will be included next term during the integration of all components.

Although the initial plan included using an SQL database, it became clear that this wasn't necessary at this stage, so its implementation has been deferred. Spring Boot, too, will not be required, as the focus will shift toward using more lightweight plugin frameworks.

These frameworks, such as OSGi or alternatives like Apache Felix or Eclipse Equinox, offer a more granular and flexible approach to software modularization, which aligns better with the project's goals of extensibility and scalability. These frameworks are designed to allow for easier management of dynamic components and plugins, which is essential for the project's future development and integration of new features.

4.2 Milestones achieved

* **Parsing SOFT Files**: Successfully parsing SOFT files (gene expression), such as GDS3310\_full.soft and GDS4794\_full.soft, was a significant milestone. These files are complex to handle, and parsing them into effective data structures (e.g., List<String, Double> with sample names) simplifies data processing and integration for any future biological data.
* **Implementation of K-Means Clustering**: Completing the implementation of K-Means clustering results in a fundamental step for the success of this project. Clustering is a fundamental aspect of bioinformatics, and this functionality forms the core of the software's analytical capabilities.
* **Prototype Interface Development**: Creating a prototype interface using Figma was another milestone. Although introductory, it translated abstract ideas like user stories and use-case diagrams into a visual, tangible design. This prototype sets the groundwork for next term’s implementation in JavaFX.

4.3 Next steps

Next term will focus on completing the full implementation of the project:

* Integrating Software Engineering (SE) principles, including the application of design patterns and modularity for better scalability.
* Developing the final user interface in JavaFX, ensuring it aligns with the project’s functional and aesthetic requirements.
* Implementing the SQL database to store and manage datasets and results effectively.
* Comprehensive testing and evaluation to ensure the software meets specifications and performs reliably under different scenarios.

5. Challenges faced

5.1Biological Overhead

Coming from a computer science background, bioinformatics presented a steep learning curve. Understanding different biological datasets such as gene expressions (used in our project)*,* protein sequences and choosing one for our POC required gaining foundational knowledge of the field. For instance, the Central Dogma (CRICK, 1970)and the basics of gene expression and sequencing were not immediately intuitive concepts. I had to understand where these datasets come from, their formats, and the biological processes they represent. This was a necessary step to ensure the software would not only process the data but also provide meaningful insights for bioinformatics researchers.

5.2 Technical challenges

5.2.1 Technical Challenges in Parsing

Parsing SOFT files was challenging due to their large size and complex structure. The SOFT files had a complex and hierarchical structure which warranted more time for handling them. Converting the data into a structured format like List<String, Double> while preserving the GSM sample names involved careful design and optimization. Additionally, handling large files efficiently to manage memory consumption was challenging, encouraging the search for effective data-handling strategies.

5.2.2 Adopting New Tools and Techniques

Using new tools like Figma for prototyping and implementing machine learning algorithms like K-means clustering required research and hand on practice. Figma was a great choice for prototyping due to its user-friendly interface, but understanding how to transition these designs into functional JavaFX interfaces posed a forward-looking challenge. Similarly, implementing K-means clustering from scratch meant understanding the algorithm’s theory and adapting it for biological datasets. Balancing computational efficiency with biological relevance added extra layer of technical complexity.

6.Diary

2-3th October

Set up git repo, started structing my project plan, did intial reading in bioinfomratics.

4th October

Read about protein sequencing, DNA synthesis which is how later on CS and biologists came together to create this field of bioinformatics, since this created a lot of data which had to be analysed. This helped me start my abstract of the project paper.

5th October

Reading a Research paper that has helped me understand the evolution of protein to DNA analysis, which has helped me refine my abstract until now. Plan is to finish it and majority of my abstract by tomorrow.

7th October

On my way to finish the research paper i was reading on the 5th, went on a tangent of reading 2 more research papers which were informative but not too useful for abstract, hoping to finish abstract tomorrow and start working on timeline and risk mitigations.

9th October

Finally finished my abstract and got my citations in, going to start working on the timeline and risk mitigations.

11th Ocotber

Finished my plan and submitted

28th October

Reading through SE books to plan out my code - Hope to have a rough outline by Wednesday to discuss with supervisor on Thursday

11th November

Looked at different biological sets to use and their respective clustering that can be used - going to use gse10072 for the first biological

dataset - and implement k means clustering on it.

12th

Tried to find datasets that i could use - but ended up starting from the ground up and reading a bit of lit before i start my project, there are a lot of things that i dont understand that i hope i can once i am through with the lit - plug in system, datasets for bioinformatics, software engineering principles.

I started reading building bioinformatics solution(by conrad bessant et al) to understand how the data is stored and their relevant metadata, this book teaches us how a bioinformatics software is created from scratch - hopefully when i am through with the relevant chappters i can have more confidence.

15th-18th

Reading on lit, the design pattern (gamma et al)help understand what i could implement and working on the parser for a soft file since its one of the harder files to parse, this will give me a good grasp for future file parsing.

19th november

Worked on the SoftParser class so it can parse SOFT data files - in progress

2nd decemeber

Finished SoftParser class, now it can successfully parse a SOFT file, was working on a bug 21-24 which caused the parser to take in unnecessary lines.

3rd december

Worked on making suplementary classes for my K means clustering, we can turn our parser output into it's respective classes so it can be encapsulated well, this will help for our code to be cleaner and more useable.

Worked on eucledian distance - i am using the eucledian because the implementation is pretty straightforward.

Worked on centroid representation for our k means - so our clusters can be in the same dimension and converge faster

4th december

Our softparser was giving out a very complicated data structure which didn't align well with good perormance and optamization, started working on refactoring that.

Worked on centroid generation, and added SE elements into code for better SE practise(Strategy)

5th december

Worked on K means more - for random centroid generation

6th-8th decemebr

Finished working on K means - assign, average and the other methods

9th-11th

refactoring SoftPArser for a more apropriate data structure.

6.1 Video link

Please find the google drive link for my code below:

https://drive.google.com/drive/folders/1me90PuRkjOP7paw-0iZjc-noH6VxVnXv?usp=sharing

Works Cited

(n.d.). Retrieved from TIOBE: https://www.tiobe.com/tiobe-index/java/

Ali Dehghani, E. M. (2024, January 8). *The K-Means Clustering Algorithm in Java*. Retrieved from baeldung: https://www.baeldung.com/java-k-means-clustering-algorithm

Anna Niarakis, D. W. (2022). Addressing barriers in comprehensiveness, accessibility, reusability, interoperability and reproducibility of computational models in systems biology. *Briefings in Bioinformatics*, Volume 23, Issue 4.

Bessant, C. S. (2009). *Building bioinformatics solutions : with Perl, R and MySQL.* Oxford University Press.

Bloch, J. (2017). *Efficitive Java.* Addison-Wesley Professional.

CRICK, F. (1970). Central Dogma of Molecular Biology. *Nature , 227*, 561–563 .

D., P. (n.d.). Retrieved from https://peerdh.com/blogs/programming-insights/developing-a-plugin-architecture-for-modular-bioinformatics-tools

Eric Clayberg, D. R. (2008). *Eclipse Plug-ins 3rd Ed.* Addison-Wesley Professional.

Erich Gamma, R. H. (1994). *Design Patterns: Elements of Reusable Object-Oriented Software.* Addison-Wesley Professional.

Gentleman, R. C. (2008). Bioconductor: open software development for computational biology and bioinformatics. *Genome Biol 5*.

Goecks, J. N. (2010). Galaxy: a comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences. *Genome biology*, 1-13.

Jeff Gauthier, A. T. (2019). A brief history of bioinformatics. *Briefings in Bioinformatics, 20*(6), 1981–1996.

Lesk, A. M. (2019). *Introduction to bioinformatics.* Oxford university press.

Quan Zou, G. L. (2020). Sequence clustering in bioinformatics: an empirical study. *riefings in Bioinformatics*, 1-10.

Reich, M. L. (2006). GenePattern 2.0. *Nat Genet*, 500-501.

Shannon, P. M. (2003). Cytoscape: a software environment for integrated models of biomolecular interaction networks. *Genome research*, 2498-2504.

Spjuth, O. H. (2007). Bioclipse: an open source workbench for chemo- and bioinformatics. *BMC Bioinformatics*, 59.

Steiper, M. E. (2005). *Bioinformatics: A Practical Guide to the Analysis of Genes and Proteins.* New York: John Wiley and Sons.