



## **Graduation Project Report**

# **Predicting Antimicrobial Peptides Activity Using Machine Learning for Drug Design Development**

### **Students Names and IDs**

**Batool Bahaa Abu-Najem 2019978017**  
**Raghda Emad Obeidat 2020978194**

**Supervisor Name**  
**Dr. Ahmad Al-Omari**

**Semester: Second 2023/2024**

**Date: 10<sup>th</sup> August 2024**

## **Students' Property Right Declaration and Anti-Plagiarism Statement**

We hereby declare that the work in this graduation project at Yarmouk University is our own except for quotations and summaries which have been duly acknowledged. This work has not been accepted for any degree and is not concurrently submitted for award of other degrees. It is the sole property of Yarmouk University, and it is protected under the intellectual property right laws and conventions.

We hereby declare that this report is our own work except from properly referenced quotations and contains no plagiarism.

We have read and understood the school's rules on assessment offences, which are available at Yarmouk University Handbook.

### **Students:**

Name: Batool Bahaa Abu-Najem

Signature: *Batool*

Date: August 10<sup>th</sup>, 2024

Name: Raghda Emad Obeidat

Signature: *Raghda*

Date: August 10<sup>th</sup>, 2024

## Table of Contents

---

Students' Property Right Declaration and Anti-Plagiarism Statement .....	i
List of Tables .....	iii
List of Figures .....	iv
Abstract .....	v
<b>Chapter 1: Introduction</b> .....	1
1.1 Problem and Purpose .....	1
1.2 Background .....	1
1.3 Aims and Objectives .....	3
1.4 Current solutions and approaches .....	4
1.5 Main Solution.....	4
1.6 Technical details .....	5
1.7 Contributions.....	5
<b>Chapter 2: Background</b> .....	6
2.1 Python .....	6
2.2 Google colab .....	6
2.3 Machine Learning .....	7
2.3.1 Random forest .....	9
2.3.2 Deep learning .....	10
2.4 Peptide Features .....	10
2.5 Potential Ethical and Invironmental Issues .....	11
<b>Chapter 3: Design</b> .....	12
3.1 Design overview .....	12
3.1.1 Design Description.....	12
3.1.2 Addressing problem statement.....	12
3.1.3 Detailed Figure For Our Solution .....	12
3.2 Design Details .....	12
3.2.1 Design specifications .....	12
3.2.2 Design Process .....	13
3.2.3 Legal Aspects.....	13
3.2.4 Design Constraints .....	13
3.2.5 Design Standards.....	13
3.2.6 Design Alternatives.....	14
3.2.7 Safety Consideration .....	14
3.2.8 Design Considerations Table .....	14

<b>Chapter 4: Implementation</b>	15
4.1 Dataset	15
4.2 Proprocessing	16
4.2.1 Normalization and PCA	17
4.3 Libraries	18
4.4 Machine Learning	23
4.5 Deep Learning	23
4.6 Performance improvement with CUDF for our ML Model	26
<b>Chapter 5:Results and Discusion</b>	27
5.1 Results	27
5.2 Discusion	29
<b>Chapter 6: Economical, Ethic, and Contemporary Issues</b>	31
6.1 Preliminary Cost Estimation and Justification	31
6.2 Relevant Codes of Ethics and Moral Frameworks	31
6.3 Ethical Dilemmas and Justification of Proposed Solution	31
6.4 Relevant Environmental Considerations	31
6.5 Relevance to Jordan and Region (Social, Cultural, and Political)	32
6.6 Other Issues and Constraints	32
<b>Chapter 7: Project Management</b>	33
7.1 Schedule and Time Management	33
7.2 Resource and Cost Management	33
7.3 Quality Management	33
7.4 Risk Management	33
7.5 Project Procurement	33
<b>Chapter 8: Conclusion and Future Work</b>	35
8.1 Contibution of the work	35
8.2 Further future work	35
8.3 Lessons learned	36
References	37
APPENDIX	39

## List of Tables

---

Table 1. Peptide functions and their description..	11
Table 2. Design considerations table...	14
Table 3. The model performance for training set	21
Table 4. Modle Runtime comparison with and without the code.....	26
Table 5. The accuracy score for all feature combined then excluding one feature at a time.....	27
Table 6. PCA accuracy results	29

## List of Figures

---

Figure 1. Python logo.....	6
Figure 2. Google Colabratory logo.....	6
Figure 3. Google colab interface for a new notebook.....	7
Figure 4. Venn diagram of Artificial intelligence, Machine learning, and Deep learning.....	8
Figure 5. Process of Machine learning .....	9
Figure 6. Process of random forest.....	9
Figure 7. Computing different types of composition-based features of peptide composition .....	10
Figure 8. Graphical representation of tool .....	12
Figure 9. Sample of the active AMP dataset.....	15
Figure 10. Sample of the second dataset .....	15
Figure 11 .Install miniconda in a Linux environment.....	16
Figure 12. Running the CD-HIT code with the results .....	17
Figure 13. Download and install Pfeature library .....	19
Figure 14. The code for Extracting and Processing Features of AMP Sequences.....	19
Figure 15. Calculating and Labeling Features for Positive and Negative AMP Sequence.....	20
Figure 16. A sample of the results from feature_calc function.....	21
Figure 17. LazyPredict for AMP Classification.....	22
Figure 18. Build random forest model.....	23
Figure 19. Prediction on train and test data.....	23
Figure 20. Accuracy result for all features combined .....	27
Figure 21. Feature important code .....	28
Figure 22. Neural network results.....	29
Figure 23. Confusion matrix .....	30
Figure 24. Gantt chart of the project.....	33
Figure 25. Install libraries.....	39
Figure 26. Download Data.....	39
Figure 27. Quantify peptide features .....	40
Figure 28. Add Class Column.....	41
Figure 29. Data Preprocessing.....	42
Figure 30. Lazypredict Library .....	42
Figure 31. Neural Network code .....	43
Figure 32. Random Forest model and checking model performance .....	44
Figure 33. Checking model performance.....	45

## Abstract

---

In this project, we developed an efficient and simple method to predict antimicrobial peptides (AMPs) activity from their amino acid sequences. This prediction aids in drug design by streamlining the lab work of pharmacists. Utilizing an extensive dataset from different resources, one of which was a collaborative work with the results of a research study by Dr. Ahmad Al Omari on big data bot, which included 1,295 sequences of both AMPs and non-AMPs. we investigated different peptide features such as amino acid composition, physicochemical properties and amino acid index and many more then tried to train the model to find correlations between the features frequencies and peptide activity to help in the prediction process, using the Random Forest model, we achieved an accuracy of 92% and after many trials we decided to apply Principal Component Analysis (PCA) and normalization with the data preprocessing step then we were able to enhance the accuracy to 95%. Additionally, we made use of NVIDIA's new technology CUDF to accelerate pandas performance. We achieved the end goal of a robust machine learning model that significantly reduces the complexity and time required for AMP classification, providing a valuable tool for pharmaceutical research and development in drug design that can accelerate the discovery and development of new, effective antibiotics.

**Keywords**— Antimicrobial Peptides, Machine Learning, Drug design development, Prediction, Antibiotics.

## Chapter 1: Introduction

---

### 1.1 Problem and Purpose

The rise of antibiotic-resistant bacteria poses a significant challenge to modern medicine, necessitating the urgent development of new antimicrobial agents. Antimicrobial peptides (AMPs) have emerged as promising candidates due to their broad-spectrum activity and unique mechanisms of action. However, identifying and classifying effective AMPs from vast peptide libraries remains a complex and labor-intensive process [1].

In this project, we aim to harness the power of machine learning to streamline and enhance the prediction of antimicrobial peptides into active and inactive categories. We plan to utilize a Random Forest model, known for its robustness and ability to handle complex datasets. By leveraging computational techniques, we can extract relevant peptide features, such as amino acid composition and charge, and train the Random Forest model to accurately predict peptide activity.

### 1.2 Background.

In the world of medicine, antibiotics are useful to treat and even prevent some types of bacterial infections which makes it necessary to perform surgical procedures safely.

When bacteria mutate and become resistant to the antibiotics used to treat the infection they caused, Antibiotic resistance. The abuse of using antibiotics increases the chance for bacteria to develop that resistance to the medication. As the World Health Organization (WHO) has extensively announced, the alarming rise globally in resistance towards conventional antimicrobials represents a potential and serious risk to public health [2].

At least 700,000 people per year worldwide die from antibiotic resistant infections. Due to the increasing overuse of antibiotics, this could rise to 10 million per year globally by 2050, exceeding cancer [3]. Some tried to solve the bacteria resistance issue by creating new drugs but that could cost hundreds of millions of dollars and take up to a decade to develop. This obstacle has prevented the introduction of new classes of antibiotics into clinical use in the last 30 years.

And now, antibiotics are no longer routinely prescribed for infections because:

- antibiotics become ineffective with treating serious conditions when used to treat trivial conditions.
- antibiotics can cause side effects.



- antibiotics are not effective with virus infections.

Accelerating growth and global expansion of antimicrobial resistance has deepened the need for discovery of novel antimicrobial agents. This led to the discovery of AMPs in the 1980s, thanks to modern immunology [4].

AMPs have a wide range of inhibitory effects against bacteria, fungi, parasites and viruses. AMPs are a class of small peptides typically 10 – 50 amino acids in length. More than 1000 AMPs have been found naturally in many tissues of many different species [5]. Hundreds of synthetic AMPs have also been created, often by mimicking natural sequences in combination with trial-and-error experimentation. They are an important part of the innate immune system of different organisms [6], showing the ability to kill microbial pathogens but also propose other benefits such as wound healing, anti-tumor, immune modulation [7]. The main Advantages are that they have a Broad-spectrum activity and are less likely to induce resistance due to their mechanism of action.

The emergence of antibiotic-resistant microorganisms and the increasing of concerns about the use of antibiotics resulted in the development of AMPs, which have a good application prospect in medicine, food, animal husbandry, agriculture and aquaculture [8]. The development process included research on the topic of the usage of artificial intelligence after the worldwide spread usage after the launch of open AI's website we looked into the applications of machine learning in bioinformatics, we were drawn to drug design as a way to help pharmacists with lab work. The integration of computational tools and algorithms with biological and chemical data has accelerated the drug discovery process, improved success rates, and reduced costs.

Previous methods for discovering the activity of antimicrobial peptides

Mass Spectrometry: identifies peptides based on their mass-to-charge ratio also known as molecular weight determination. classify them based on structural\_properties. First step is Nano electrospray ionization, creating positively or negatively charged ions, couple the outlet of a small-scale chromatography column directly to the inlet of a mass spectrometer. The flow from the column is passed through a needle that is 10-15  $\mu\text{m}$  at its tip. Second step the ions are sorted and separated according to mass-to-charge ( $m/z$ ) ratios in the mass analyzer. often works in concert with the ion detection system. The third and final step the separated ions are then measured and sent to a data system where the  $m/z$  ratios are stored together along with their relative abundance [9].

We can sum up the disadvantages of this technique by mentioning the high cost of instrumentation and maintenance. And it being a time-consuming analysis for large datasets.

**Chromatography Techniques** High-Performance Liquid Chromatography (HPLC): Separates peptides based on their physical and chemical properties. Application: HPLC can be used to purify and identify AMPs, providing insight into their hydrophobicity and other characteristics.

Disadvantages: Time-Consuming: Long run times for each sample. Reagent Costs: Requires costly solvents and reagents. Expertise Needed: Requires skilled operators for proper use and interpretation.

**Circular Dichroism (CD) Spectroscopy:** CD spectroscopy analyzes the secondary structure of peptides. Some of its applications are observing the optical activity of AMPs, researchers can classify them based on their structural motifs (e.g.,  $\alpha$ -helices,  $\beta$ -sheets).

Disadvantages: Limited Structural Information that provides only secondary structure data. Sample Requirements: Requires relatively pure and concentrated samples. Interpretation Complexity with data interpretation can be challenging without complementary techniques.

**Nuclear Magnetic Resonance (NMR) Spectroscopy Overview:** NMR provides detailed information about the 3D structure of peptides in solution. Application in helping understand the structural conformation of AMPs, which is crucial for classification based on structural features.

Disadvantages: Expensive Equipment with the high cost of NMR machines. Time-Intensive for long data acquisition and analysis times. Complex Data requires advanced expertise for data interpretation.

Our Machine Learning Solution can automate NMR data analysis, reduce interpretation complexity, and potentially decrease analysis time through predictive models.

Extract peptides from a biological sample (e.g., human saliva) using methods like acid extraction or affinity purification.

### **1.3 Aims and Objectives.**

**Aims:** Develop a solid classification tool based on machine learning to distinguish between active and inactive AMPs only from their amino acid sequences.

Offer pharmaceutical researchers a fast, accurate way of prediction for the antimicrobial activity of peptides and help in screenings of potential candidates for further development in drug design.

## Objectives:

To develop a classification tool for AMPs that is efficient, we began with finding appropriate data. started preprocessing it, making sure the activity was labeled as active or inactive to train the model.

We ensured that the data source was reliable, and we started standardizing the peptide sequences to remove inconsistencies or any repetition. After that, we focused on peptide features from the sequences, identifying and extracting relevant features, and using appropriate encoding techniques to convert these sequences into numerical values for it to be more suitable for the machine learning models. amino acid composition, physicochemical properties, and sequence motifs are some examples of peptide features that we have extracted.

We trained some of the most well-known ML models Random Forest, and deep learning models like Convolutional Neural Networks, on the processed dataset. These models have been evaluated with performance metrics like accuracy, Matthew Correlation Coefficient, and several others to determine how well they are suited for classifying AMP activity. Finally, to ensure robustness and generalizability, we validated the classification tool using independent datasets and conducted extensive testing to identify and address any potential issues or biases in the model.

The tool can help pharmaceutical companies identify active AMPs in the process of drug development and classify them to make the identification faster of a few that are viable to become candidates for further development. Saving time and costs of drug discovery at the early-stages by increasing productivity in the development pipeline is one of the main advantages we aim for in this project.

### **1.4 Current solutions and approaches**

While traditional experimental methods have been the cornerstone of peptide activity evaluation, advancements in computational and machine learning approaches have significantly enhanced prediction accuracy and efficiency. However, challenges remain in data quality, model generalization, and integrating complex biological data. The field continues to evolve with ongoing research aiming to address these challenges and improve predictive models for antimicrobial peptide activity.

### **1.5 Main Solution**

In our project, our main solution approach involves leveraging advanced machine learning techniques to enhance the prediction of antimicrobial peptide (AMP) activity. We utilize state-of-the-art algorithms, specifically deep learning models, to analyze peptide sequences and extract complex patterns that correlate with antimicrobial efficacy. By employing Google Colab for its robust computational

capabilities, we ensure efficient and scalable model training. Our approach integrates high-quality, publicly available datasets, along with additional data provided by Dr. Ahmed al-Omari, to build and validate our predictive models. This methodology aims to improve the accuracy and speed of predicting peptide activity compared to traditional experimental methods. The ultimate goal is to provide a reliable tool for drug design, enabling the rapid identification of potential antimicrobial peptides and accelerating the development of new therapeutic agents.

## **1.6 Technical details**

Antimicrobial resistance, exacerbated by the overuse of antibiotics leading to resistant strains, is a significant global health threat. In 2019, it was the third leading cause of death, surpassing HIV and malaria. The lack of new antibiotics over the past 30 years underscores the urgent need for innovative antimicrobial drugs [10]. AMPs are a promising alternative to traditional antibiotics due to their broad-spectrum activity, unique mechanisms of action, lower likelihood of resistance development, potential for synergistic effects, modifiable structures, immunomodulatory properties, and effectiveness against resistant pathogens.

Traditional methods for treating AMPs include isolation, purification, structural characterization, activity assays, mechanism of action studies, optimization, formulation development, safety and toxicity assessment, and clinical trials. While these methods provide a thorough evaluation and optimization of AMPs, they are often time-consuming, costly, limited in throughput, complex, and resource intensive.

The limitations of traditional methods have motivated our ideas in this project to rely on computational techniques such as machine learning and highlighting the utilization of the Python programming language, which provide enhanced efficiency and scalability of amp classification. Machine learning provides speed, cost-effectiveness, scalability, predictive power, reproducibility, data integration and Automated Insights, making it invaluable in accelerating the discovery and development of effective antimicrobial therapies.

## **1.7 Contributions**

- Development of an Advanced Predictive Model.
- Integration of Diverse Data Sources.
- Utilization of Google Colab for Scalable Computation.
- Contribution to Drug Design and Development.

## Chapter 2: Background

---

Here we will provide a background of the project starting from the language used then will talk about the tools used to help us construct our solutions.

### 2.1 Python

Python is an interpreted, interactive, object-oriented, high-level programming language.

Python's simple, easy-to-learn syntax using the English language for its keywords and simple punctuations emphasizes readability and therefore reduces the cost of program maintenance. Python supports modules and packages, which encourages program modularity and code reuse. The Python interpreter and the extensive standard library are available in source or binary form without charge for all major platforms and can be freely distributed. Python offers increased productivity and can suggest some useful prompts while typing which makes it a great language for beginners.

Debugging Python programs is easy, and the edit-test-debug cycle is incredibly fast without the need to compile the program. The debugger is written in Python itself, testifying to Python's introspective power [11].



**Figure 1.** Python logo [12].

### 2.2 Google Colab or Colaboratory

Google Colaboratory is a free online cloud-based Jupyter Notebook environment that allows us to train our machine learning and deep learning models [13]. It is capable of working on any computer since it works on the web browser. It is considered an excellent choice for students, data scientists, researchers, and anyone who wants to start with machine learning or data science.



**Figure 2.** Google Colaboratory logo [14].

## Benefits of Google Colab:

- **Accessibility:** as a cloud-based platform, there is no need to install any software.
- **Power:** it provides access to powerful computing resources, including GPUs and TPUs. Giving the opportunity to run complex machine-learning models quickly and efficiently.
- **Collaboration:** it was easy for us as team members to collaborate on this project editing and running the code in real-time by sharing the notebook.
- **Flexible:** Train and run machine learning models, process data, and create visualizations. pre-installed with many popular libraries and tools to save time and eliminate the need to manually install and configure these tools.

A **notebook** is an environment that is web-based for writing and running code. Notebooks are like scripts or code files in other programming environments but offer some unique advantages like allowing one to write and execute code in a web browser and displaying the output in real-time. This facilitates iterating through your code and visualizing the outcomes as you go. Markdown is also supported in Colab notebooks, so you can add formatted text, equations, and photos in addition to your code. Additionally, you can add notes and comments on your code, which promotes comprehension and teamwork. For data scientists and machine learning professionals, notebooks are an effective tool because they offer an adaptable and interactive environment for developing and testing code.



**Figure 3.** Google colab interface for a new notebook

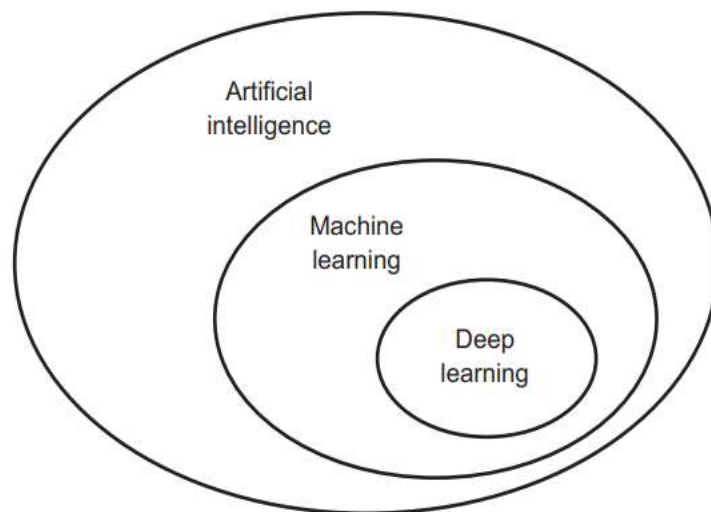
## 2.3 Machine learning

is a field of computer science that gives computers the ability to learn without being programmed explicitly for that specific function [15] . This computational method was inspired from the study of pattern recognition and computational learning theory in artificial intelligence. The main working

principle is that the model is built from simple inputs and then run through an algorithm that can learn from that data and make predictions or decisions depending on the application needed for it.

ML also refers to a method of data analysis that automates analytical model building. It is a branch of artificial intelligence based on the idea that systems learn from data, identify patterns, and make decisions with as little human intervention as possible. Beginning with the possibility of automatically applying complex mathematical calculations to big data, a number of ML applications have been developed such as self-driving cars, online shopping recommendations, email filtering, intruder prevention, fraud detection, and natural language interfaces.

A field that is closely related to machine learning is the field of computational statistics, which shares its structural view on computer prediction-making. Growing volumes and varieties of data available, cheaper compute processing, and more affordable data storage drive new demands for ML. Enterprises will continue to look out on how quickly and, where possible, automatically to produce models that analyze increasingly complex data to deliver more accurate results and thus identify new opportunities or avoid risks. However, organizations can leverage ML to build predictive models that can make data-driven decisions without the need for human involvement. Essentially, all of the ML applications require a supercomputer or high performance computing infrastructure to build.



**Figure 4.** Venn diagram of Artificial intelligence, machine learning, and deep learning [16].

The basic machine-learning process can be divided into three parts:

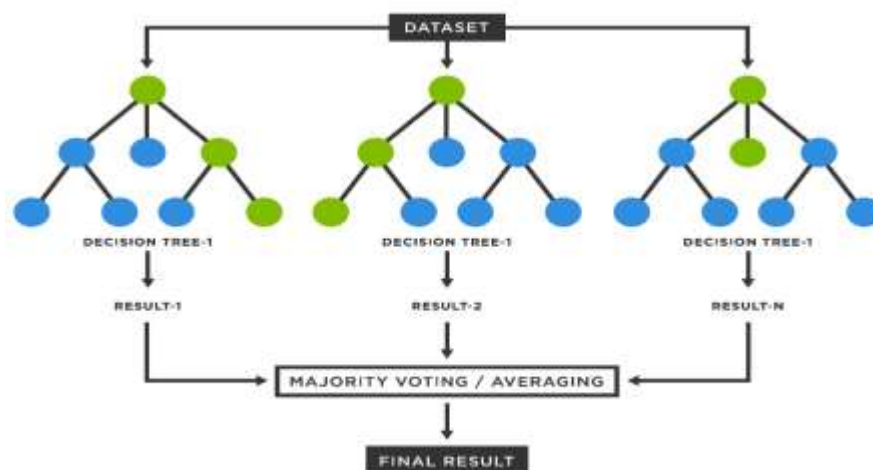
1. Data Input: Past data or information is utilized as a basis for future decision-making
2. Abstraction: The input data is represented in a broader way through the underlying algorithm
- 3.Generalization: The abstracted representation is generalized to form a framework for making decisions [17].



**Figure 5.** Process of Machine Learning [15].

### 2.3.1 Random Forest

Is a type of classifier that combines multiple decision trees to make predictions . This combining process, known as assembling, involves using different sets of features through a technique called bagging. The random forest consists of many of these trees, each trained to emphasize different aspects of the data. Once all the trees are trained, their outputs are combined using a majority vote. This approach improves prediction accuracy compared to using a single decision tree model alone. Figure 6 makes the operation of RF easier to understand [18].



**Figure 6.** Process of Random Forest [18].

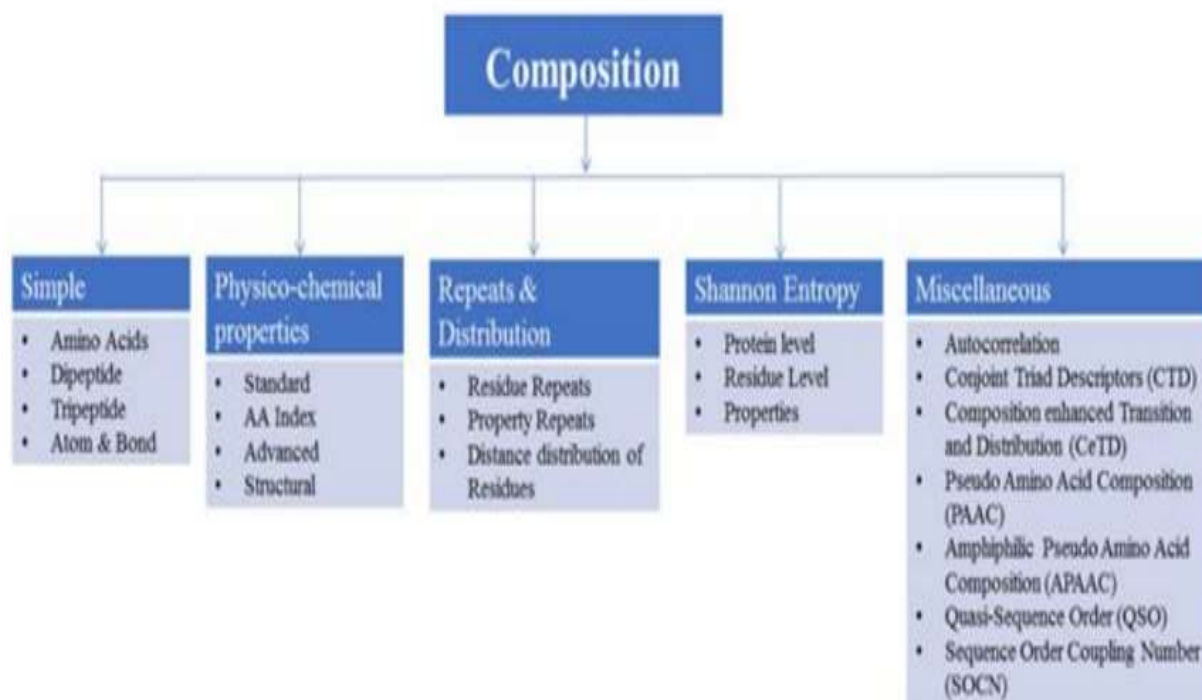


### 2.3.2 Deep learning

Is an AI function and a subset of ML, used for processing large amounts of complex data . Deep learning can automatically create data patterns based algorithms.

Deep learning that focuses on using neural networks with many layers (hence "deep") to model and understand complex patterns in data. Neural networks are inspired by the human brain. These networks consist of layers of interconnected nodes (neurons) that process and learn from data.

**2.4 Peptides feature** is the specific composition of amino acids in a Portion that determines its structure and function. Figure 7 shows the details of each peptide featured we made use of during the work time on this project



**Figure 7.** Computing different types of composition-based features of peptide composition [19].

**Table 1:** Peptide functions and their description

Function Title	Description
AAC	To calculate Amino acid composition of a peptide
DPC	To calculate Dipeptide composition of a peptide
TPC	To calculate Tripeptide composition of a peptide
ATC	To calculate atomic composition (% of Carbon, Hydrogen, Nitrogen, Oxygen, Sulphur content) of a peptide
BTC	To calculate Bond composition (% of Carbon, Hydrogen, Nitrogen, Oxygen, Sulphur content) of a peptide
PCP	To compute property composition from whole sequence of a protein using
RRI	To compute Repetitive Residue Information of amino acid of protein sequences.
DDR	Distance Distribution of residues
SEP	Shannon Entropy of a Protein
SER	To compute Shannon Entropy of all amino acids of protein/ peptide sequences.
PAAC	To compute Pseudo Amino acid composition of a peptide.
CTC	The Conjoint Triad Calculation of the Descriptors of protein/ peptide sequences.

While writing the code and making use of the pfeature library, we found multiple options to choose from and we landed on using Whole amino acid sequence (WP\_function): This option allows users to compute features of a protein from the whole sequence. This option is important for users when user wish to understand the overall properties of a protein or peptide according to the pfeature user manual.

## **2.5 Potential ethical and/or environmental issues.**

Maintaining public trust in the ethical conduct of research involving advanced technologies is essential for the acceptance and adoption of new antimicrobial therapies.

## Chapter 3: Design

### 3.1 Design Overview:

#### 3.1.1 Design Description

The system is designed to classify a sequence of amino acids that belong to a peptide into either active or inactive to help pharmacists in the drug development process.

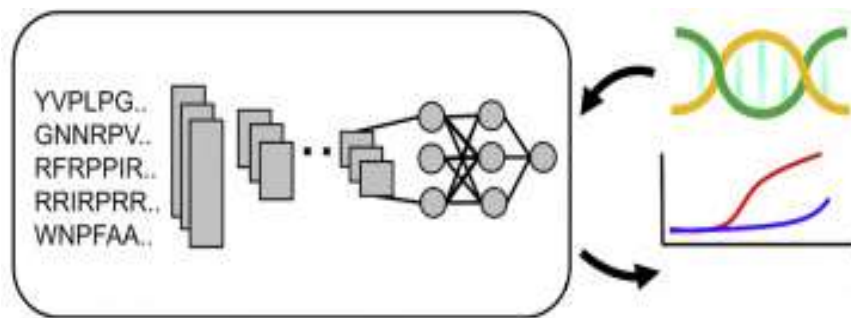
#### 3.1.2 Addressing the problem statement

The problem statement involves selecting active antimicrobial peptides efficiently to reduce time and costs. This challenge has been addressed using a machine-learning model that predicts peptides as either active or inactive based on their characteristics.

#### 3.1.3 Detailed figure of our solution

Operation scenario:

1. The pharmacist extracts the amino acids sequence that belongs to the AMP
2. The pharmacist then inserts the extracted sequence into our system



**Figure 8.** Graphical representation of tool [20].

3. The trained system will decide whether the peptide is active or not to make use of it
4. Then they can decide to conduct further research or use it in a new medication.

### 3.2 Design Details:

#### 3.2.1 Design Specifications

**Dimensions:** The system should be scalable to handle many peptide sequences with high accuracy.

**environmental factors:** The system should be designed to run in typical computational environments: including personal computers and cloud-based servers.

**ergonomic factors:** Ensure the system is accessible to users with varying levels of technical expertise.

**aesthetic factors:** Visual appeal of the user interface.

**Maintenance:** The code should be modular to facilitate updates and maintenance.

**Cost:** Costs related to running the system, such as computational resources and cloud services if applicable.

**Safety:** Implement checks to avoid misclassification that could lead to incorrect conclusions about peptide activity.

**Quality:** Continuous monitoring of the system's performance and accuracy in a production environment.

### 3.2.2 Design Process

Describe the detailed techniques in your solution. For each part of the solution, put it in the context of the overall system or solution – where does it fit, what is its functionality? Do not just give a figure or an algorithm but explain in words what the design behind the method used is. If your method expands some prior method(s), refer to that, and point out the addition that you have done.

### 3.2.3 Legal Aspects

- Respect intellectual property rights related to any databases, software libraries, or algorithms used.
- Obtain necessary licenses for proprietary tools and give proper attribution to open-source resources in accordance with their licenses.

### 3.2.4 Design Constraints

- Compliance with data protection regulations like GDPR.
- Ensure ethical use of the tool, avoiding biases in data and model predictions.
- high concentrations of AMP can be toxic to human cells. Modifying AMPs to enhance selectivity and reduce toxicity is a key area of research.

### 3.2.5 Design Standards

- Machine Learning Standards: Regularly test the model for biases and take steps to ensure fair and unbiased predictions.
- Bioinformatics Standards: Ensure that all data processing steps and results are documented in accordance with these standards.
- e.g. MIAPE (Minimum Information About a Proteomics Experiment)

### 3.2.6 Design Alternatives

- Explore different ways to classify the data with different machine learning models.
- Working on other alternatives to antibiotics such as phage therapy, probiotics, or immunotherapy.

### 3.2.7 Safety Consideration

- Provide clear disclaimers and guidelines on the intended use of the tool, emphasizing that it is a research tool and not a definitive diagnostic tool.
- Follow best practices for data handling and ensure team members are trained in data privacy and security protocols.

### 3.2.8 Design considerations table.

**Table 2:** Design considerations table.

<b>Design consideration</b>	<b>Project application</b>	<b>Relevant location in report</b>
<b>Performance</b>	predict AMPs ,cuDF (for speed up the model)	4.3 ,4.6
<b>serviceability</b>	Pharmaceutical companies and research	2.1
<b>Economic</b>	Cost-effective solution for AMPs prediction	6.1
<b>Environmental</b>	Software reduces the need for lab testing	3.2.1,5.4
<b>Environmental Sustainability</b>	Minimizes resource consumption in research	3.2.1,5.4
<b>Manufacturability</b>	Easily deployable in various research settings	4.3
<b>Ethical</b>	Access to Treatment and Healthcare Disparities	6.3
<b>Health and safety</b>	Potential for safer drug design and testing	1.1, 3.2.1
<b>Social</b>	Increased accessibility to AMP research tools	6.5
<b>Political</b>	Compliance with regulatory standards	6.5

## Chapter 4: Implementation

This section will describe the different methods we used in this project to achieve our final goal of classifying the collected data after doing the necessary processes and finally display the results.

### 4.1 Dataset

First, we looked for some datasets of AMPs with the activity linked to it and we stumbled upon the Computational Biology and Bioinformatics Lab Centre for AI-Driven Drug Discovery at Macao Polytechnic University's website [17]. Their mission is to improve computational methods to speed up the drug discovery and development process. They had a dataset for biological activity (AxPEP) that was classified as positive for active (AMP) and negative for inactive (non-AMP) as Fasta file format 1529 sample each ranging in amino acid length from 5 to 30 in both datasets.

```
>amp5_30_986
VIGSILGALASGLPTLISWIKNR
>amp5_30_987
VIPFVASVAAEMQHVCASRKC
>amp5_30_988
WATLAKGALKLIPTIANAFSSKS
>amp5_30_989
ADNKNPLEECFRETNVEEFLEIAR
>amp5_30_990
ALWKDILKNAGKAALNEINQLVNQ
>amp5_30_991
DSHAKRHHGYKRFHEKHSHRGY
>amp5_30_992
DSHEKRHHEHRRKFHEKHSHRGY
>amp5_30_993
DSMGAVKLAKLLIDKMKCEVTKAC
```

**Figure 9.** Sample of the active AMP dataset

And for the independent dataset was provided to us by our supervisor Dr. Ahmad Al-Omari generated in an innovative way. In that paper a software that can execute extraction procedures automatically and without human intervention from big data repositories. It emulates the process of extracting data from graphical user interfaces or web resources that are not reachable using command-line tools. The software has been constructed to extract data for machine learning purposes [21] which is perfect for us. The data was classified based on activity with 1 to represent an AMP being active and 0 for inactive which helped us immensely with the continuation of our project and saved us so much time and effort.

YCSYTMEA	0
SVAGRAQGM	0
TESYFVFSVGM	0
GKWWSLKHKILK	1
ILPILSLIGLL	0
SAVGRHGRRFGLRKHRRKH	1
GLLDALSGILGL	0
GLLGTGNLLNGLGL	0

**Figure 10.** Sample of the second dataset

## 4.2 Preprocessing

**Conda:** is an open-source package management and environment management system. It is widely used in scientific computing, data science, and software development.

**Miniconda:** A minimal distribution with only Conda and its dependencies, allowing users to install only the packages they need.

**Wget Command:** Downloads the package installer script for Python on Linux from the official repository.

**Chmod Command:** making the downloaded installer script executable by Changing the permissions.

**Bash Command:** Executes the installer script for the Miniconda.

Flags:

- -b runs the installer in batch mode (without user interaction).
- -f forces the installation, overwriting any existing installation.
- -p /usr/local specifies the installation path.

**Sys Path:** Adds the Miniconda installation directory to the Python path, allowing Python to locate and use the newly installed packages.

```
! wget https://repo.anaconda.com/miniconda/Miniconda3-py37_4.8.2-Linux-x86_64.sh
! chmod +x Miniconda3-py37_4.8.2-Linux-x86_64.sh
! bash ./Miniconda3-py37_4.8.2-Linux-x86_64.sh -b -f -p /usr/local
import sys
sys.path.append('/usr/local/lib/python3.7/site-packages/')
```

**Figure 11.** Install Miniconda in a Linux environment

## CD-HIT

Install CD-HIT: `! conda install -c bioconda cd-hit -y`

- `!`: indicates a shell command execution in colab's environment
- Conda install: install a package using Conda, the package and environment management system.
- -c: specifies the channel from which to install the package.
- Bioconda channel specializes in bioinformatics software.

- cd-hit: package needed to be installed, Cluster Database at High Identity with Tolerance (CD-HIT), a widely used program for clustering and comparing protein or nucleotide sequences [22].
- -y: automatically confirms the installation without user confirmation.

cluster sequences in a FASTA file using the CD-HIT program.

- -i specifies the input file.
- train\_po.fasta: the input file containing sequences in FASTA format that need to be clustered.
- -o specifies the output file. And train\_po\_cdhit.txt is the file where the output clustered sequences will be written.
- -c sets the identity threshold for clustering sequences. And set the sequences to be clustered together if they have 99% or more identity.

From the figure below we can see the difference in sequences number before and after executing the CH-HIT code. In positive dataset we had 1529 then became 1337 after Clustering similar sequences.

```
! cd-hit -i train_po.fasta -o train_po_cdhit.txt -c 0.99

! cd-hit -i train_ne.fasta -o train_ne_cdhit.txt -c 0.99

! grep ">" train_po_cdhit.txt | wc -l
1337

! grep ">" train_po.fasta | wc -l
1529

! grep ">" train_ne.fasta | wc -l
1529

! grep ">" train_ne_cdhit.txt | wc -l
1422
```

**Figure 12.** Running the CD-HIT code with the results

#### 4.2.1 Normalization & PCA

We tried to improve the accuracy of the model in this project using techniques like PCA, but first we wanted to normalize and standardize the data.

**Normalization:** typically refers to scaling individual samples to have unit norm, often used to transform the data into a range [0, 1] or [-1, 1]. The transformations can lead to better performance of many algorithms.

```
from sklearn.preprocessing import StandardScaler
```



```
# Standardizing the features
normalized_array = StandardScaler().fit_transform(normalized_array)
```

The “fit\_transform” method computes the mean and standard deviation of each feature and scales them to have a mean of 0 and a standard deviation of 1.

We initialize a ‘StandardScaler’ object and apply the ‘fit\_transform’ method to our feature array, ‘normalized\_array’.

**(PCA) Principal Component Analysis:** is a statistical technique used for dimensionality reduction. It transforms the original set of variables into a new set of uncorrelated variables called principal components. These principal components are ordered by the amount of variance they capture from the data, with the first principal component capturing the most variance and each subsequent component capturing progressively less.

```
from sklearn.decomposition import PCA

pca = PCA(n_components=148)

principalComponents = pca.fit_transform(x)

column_names = [f'principal component {i+1}' for i in range(pca.n_components_)]

principalDf = pd.DataFrame(data = principalComponents, columns = column_names)
```

We imported the PCA class from ‘sklearn.decomposition’. Then initialized a PCA object to reduce the dataset to 148 principal components. This number was not fixed but we changed it several times and check the accuracy every time. After that the code performed Fitting and Transforming the Data by applied PCA to our feature matrix ‘x’, resulting in a new matrix ‘principal Components’ containing the reduced features. Then generate column names for the principal components. The final step was creating a data frame called ‘principalDf’ to store the principal components, facilitating further analysis and use in machine learning models.

### 4.3 Libraries

**Pfeature:** We found this library on GitHub [23]. it is a standalone software package for computing wide range of protein and peptides features from their amino acid sequence. It has the following six major modules for computing protein features based on Composition, Binary profiles, Evolutionary information, Structure, Pattern, and Model building [19].

We have developed number of forms of Pfeature that include:

- i) A web server that uses Pfeature functions via web interface [24]:

- ii) Standalone version of Pfeature
- iii) Library of python for Pfeature
- iv) Python scripts for computing features.

```
! wget https://github.com/raghavagps/Pfeature/raw/master/PyLib/Pfeature.zip

! unzip Pfeature.zip

%cd /content/Pfeature

! python setup.py install
```

**Figure 13.** Download and install Pfeature library

**Pandas** (Panel Data): this library is used for working with datasets. It includes functions for analyzing, exploring, cleaning, and manipulating data and making conclusions based on statistical theories [25]. Dataframe could be defined a 2-dimensional data structure, like a 2D array, or a table with rows and columns.

After obtaining the data and performing some preprocessing, it's time to calculate the peptide features; firstly, we imported the Pfeature and pandas libraries. And from pfeature there are multiple functions for each feature (AAC\_WP). Then preform the following steps:

- Define a function called 'aac' to take file path 'input' as an argument.
- rstrip function in panda's library removes the file extension 'txt' from the input file path because that is what we put as an argument in the brackets. The R in 'rstrip' is for the right space of the path.
- append 'aac.csv' to create the file name for the output.
- The aac\_wp function processes the input file to compute features and saves the results in the output file.
- The processed data is read into a DataFrame df\_in using pandas.
- The function returns the DataFrame df\_in.

Lastly, the function AAC is called with the path to the input file containing AMP sequences, initiating the feature extraction process. All shown in Figure 14.

```
[ ] from Pfeature.pfeature import aac_wp
import pandas as pd

def aac(input):
    a = input.rstrip('txt')
    output = a + 'aac.csv'
    df_out = ctc_wp(input, output)
    df_in = pd.read_csv(output)
    return df_in

aac('/content/Pfeature/train_po_cdhit.txt')
```

**Figure 14.** The code for Extracting and Processing Features of AMP Sequences

To start with the ML algorithm training, we had to combine the positive and negative data and add 'class' column in the dataframe to label the activity of each peptide sequence. This was made possible by the following steps:

- Getting the data file paths and saving them in 'pos' or Positive and 'neg' for negative dataset
- Calculate feature: Calling feature\_name function (here it's aac) on positive and negative datasets to compute features.
- Create class label: Creates series of class labels ('positive' and 'negative') corresponding to the length of the feature sets.
- Combining positive and negative features and labels.
- Combining feature and class in one df: Concat function merges the feature DataFrame and class labels into a single DataFrame 'df'.
- Returns the combined DataFrame 'df'.

```
[ ] pos = 'train_po_cdhit.txt'
    neg = 'train_ne_cdhit.txt'

def feature_calc(po, ne, feature_name):
    # Calculate feature
    po_feature = feature_name(po) # same as: aac(po)
    ne_feature = feature_name(ne)
    # Create class labels
    po_class = pd.Series(['positive' for i in range(len(po_feature))])
    ne_class = pd.Series(['negative' for i in range(len(ne_feature))])
    # Combine po and ne
    po_ne_class = pd.concat([po_class, ne_class], axis=0)
    po_ne_class.name = 'class'
    po_ne_feature = pd.concat([po_feature, ne_feature], axis=0)
    # Combine feature and class
    df = pd.concat([po_ne_feature, po_ne_class], axis=1)
    return df

feature = feature_calc(pos, neg, aac) # AAC
```

**Figure 15.** Calculating and Labeling Features for Positive and Negative AMP Sequence

And the result of the feature\_calc function is as shown in the figure 16.

AAC_R	AAC_S	AAC_T	AAC_V	AAC_W	AAC_Y	class
9.09	0	9.09	4.55	0	0	active
8	4	0	8	0	4	active
9.09	0	9.09	4.55	0	0	active
8	4	0	8	0	4	active
0	4.76	0	4.76	0	0	active
0	0	3.57	10.71	3.57	0	active
0	0	0	0	5.56	0	active
0	0	6.25	6.25	6.25	0	active
0	0	7.69	7.69	7.69	0	active
0	0	0	0	5.56	0	active
0	11.11	0	0	5.56	0	active
0	0	0	0	5.56	0	active
0	0	0	0	5.56	0	active

**Figure 16.** A sample of the results from feature\_calc function

We can conclude from Table 3 that the best model performance in terms of accuracy at 93%.

**Table 3:** The model performance for training set

Model	Accuracy	Balanced Accuracy	F1 Score	MCC	Time Taken
<b>RandomForestClassifier</b>	0.93	0.93	0.93	0.86	0.59
<b>LabelSpreading</b>	0.93	0.93	0.93	0.86	0.68
<b>LabelPropagation</b>	0.93	0.93	0.93	0.86	0.58
<b>DecisionTreeClassifier</b>	0.93	0.93	0.93	0.86	0.07
<b>ExtraTreeClassifier</b>	0.93	0.93	0.93	0.86	0.05
<b>ExtraTreesClassifier</b>	0.93	0.93	0.93	0.86	0.41
<b>LGBMClassifier</b>	0.92	0.92	0.92	0.84	0.21
<b>BaggingClassifier</b>	0.92	0.92	0.92	0.83	0.20
<b>NuSVC</b>	0.86	0.86	0.86	0.72	0.51
<b>KNeighborsClassifier</b>	0.82	0.82	0.82	0.64	0.29
<b>XGBClassifier</b>	0.80	0.80	0.80	0.61	0.22
<b>AdaBoostClassifier</b>	0.76	0.76	0.76	0.51	0.34
<b>QuadraticDiscriminantAnalysis</b>	0.75	0.75	0.75	0.51	0.08
<b>RidgeClassifier</b>	0.74	0.74	0.74	0.48	0.08
<b>RidgeClassifierCV</b>	0.74	0.74	0.74	0.48	0.10
<b>LinearSVC</b>	0.74	0.74	0.74	0.47	0.31
<b>LinearDiscriminantAnalysis</b>	0.73	0.74	0.73	0.47	0.09
<b>LogisticRegression</b>	0.73	0.73	0.73	0.47	0.09
<b>CalibratedClassifierCV</b>	0.73	0.73	0.73	0.46	0.96
<b>NearestCentroid</b>	0.72	0.72	0.72	0.45	0.06
<b>BernoulliNB</b>	0.71	0.71	0.71	0.42	0.06
<b>SGDClassifier</b>	0.67	0.66	0.66	0.33	0.09
<b>Perceptron</b>	0.65	0.65	0.65	0.29	0.08
<b>PassiveAggressiveClassifier</b>	0.64	0.64	0.63	0.30	0.07
<b>DummyClassifier</b>	0.49	0.49	0.49	-0.01	0.05

**Numpy:** is a package for general-purpose array processing. It offers tools for manipulating these arrays as well as a high-performance multidimensional array object. This is the core Python module for scientific computing [26].

**Sklearn:** machine learning library, uses NumPy extensively for high-performance linear algebra and array operations [27].

**LazyPredict** helps build a lot of basic models without much code and helps understand which models works better without any parameter tuning.

Here we defined and built the lazyclassifier for binary classification

- Cloning the LazyPredict repository from GitHub.
- Changing Directory to LazyPredict.
- Run the setup script to install the package.
- Import lazypredict package then lazyclassifier for the Matthews correlation coefficient
- Sklearn: Import functions for train /test split and evaluation metrics

Then the main code starts with loading our dataset of features and splitting it into feature matrix defined in X, and the Target variable of the class column. After that, we split the data into training and testing sets keeping class distribution in mind. Finally, we fit the classifier model with training and testing data and evaluating it. The code is in figure 17.

```
| git clone https://github.com/shankarpandala/lazypredict.git
| cd lazypredict
| python setup.py install

# Import libraries
import lazypredict
from lazypredict.Supervised import LazyClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import matthews_corrcoef

# Load dataset
X = feature.drop('class', axis=1)
y = feature['class'].copy()

# Data split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)

# Defines and builds the lazyclassifier
clf = LazyClassifier(verbose=0, ignore_warnings=True, custom_metric=matthews_corrcoef)

models_train, predictions_train = clf.fit(X_train, X_train, y_train, y_train)
models_test, predictions_test = clf.fit(X_train, X_test, y_train, y_test)

100%|██████████| 29/29 [00:06<00:00, 4.39it/s]
```

**Figure 17.** LazyPredict for AMP Classification

#### 4.4 Machine Learning

Random Forest classifier is created with 500 trees and trained on the provided training dataset.

```
from sklearn.ensemble import RandomForestClassifier

rf = RandomForestClassifier(n_estimators=500)

rf.fit(X_train, y_train)
```

**Figure 18.** Build random forest model

Firstly, import the 'Random Forest Classifier' class from the 'ensemble' module of the 'sklearn' (scikit-learn) library. Secondly, we initialize an instance of the 'RandomForestClassifier' and store it in the variable 'rf'. 'n\_estimators=500': This parameter specifies the number of trees in the forest. In this case, we are creating a forest with 500 decision trees. Finally, we start training the Random Forest Model using the line `rf.fit(X_train, y_train)` that trains the Random Forest model using the training dataset. As you will notice this fit function will take two arguments, one of them being `X_train` which is the feature matrix for the training data. Each row represents a sample, and each column represents a feature. The other one is `y_train` that is the target vector for the training data. Each entry corresponds to the class label (active or inactive) for the respective sample in `X_train`.

Apply the model to make predictions:

- `predict()` is a method of the "RandomForestClassifier" class that generates predicted class labels for the input data.
- `y_train_pred` 'Output': The predicted class labels for the training data are stored in this variable. And another for test prediction. In the first line we use the trained RF to make predictions on the training data (`X_train`). In the second line we use the trained RF to make predictions on the test data (`X_test`).

```
[ ] y_train_pred = rf.predict(X_train)
    y_test_pred = rf.predict(X_test)
```

**Figure 19.** Prediction on train and test data

#### 4.5 Deep learning

Neural network:

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
```

```

import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

# Split the data into features (X) and labels (y)
X = df.drop('class', axis=1).values
y = df['class'].values

# Convert string labels to numerical using LabelEncoder
label_encoder = LabelEncoder() # Initialize LabelEncoder
y = label_encoder.fit_transform(y) # Transform labels to numerical

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Normalize the features
mean = X_train.mean(axis=0)
std = X_train.std(axis=0)
X_train = (X_train - mean) / std
X_test = (X_test - mean) / std

# Build the neural network model
model = Sequential()
model.add(Dense(64, input_shape=(X_train.shape[1],), activation='relu'))
model.add(Dense(64, activation='relu'))
# Use number of classes in output layer, assuming binary classification here
model.add(Dense(1, activation='sigmoid'))

# Compile the model
# Use binary_crossentropy for binary classification
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Train the model
model.fit(X_train, y_train, epochs=50, batch_size=16, validation_split=0.2)

# Evaluate the model
loss, accuracy = model.evaluate(X_test, y_test)
print(f"Test accuracy: {accuracy}")

# Make predictions
predictions = model.predict(X_test)
# Adjust prediction threshold if needed (e.g., 0.5)
predicted_classes = (predictions > 0.5).astype(int)

```

```
# Display the first 5 predictions
print("First 5 predictions:", predicted_classes[:5].flatten()) # Flatten to 1D array
print("First 5 true labels:", y_test[:5])
```

first we must Import the Libraries for this to work and find the needed functions:

- ‘pandas’ for data manipulation.
- ‘numpy’ for numerical operations.
- ‘train\_test\_split’ from ‘sklearn’ for splitting the dataset.
- ‘LabelEncoder’ from ‘sklearn’ for converting categorical labels to numerical format.
- ‘tensorflow’ and ‘keras’ for building and training the neural network.

Secondly, we started Splitting Data into Features and Labels:

The dataset df is split into features X and labels y. The feature columns are all columns except 'class', and the label column is 'class'. Then the ‘LabelEncoder’ converts the string labels in y ('active' and 'inactive') to numerical labels (0 and 1). Here's the most important part of the code which is splitting Data into Training and Testing Sets. The data is split into training and testing sets with 80% for training and 20% for testing. ‘The random\_state=42’ ensures reproducibility. Feature normalization is performed to scale the training and testing data such that they have a mean of 0 and a standard deviation of 1. This step is crucial for improving the performance of the neural network.

Building the Neural Network Model in the following steps:

- A neural network is built using the Sequential model. It consists of:
- An input layer with 64 neurons and ReLU activation.
- A hidden layer with 64 neurons and ReLU activation.
- An output layer with 1 neuron and sigmoid activation (for binary classification).

Then the model is compiled with:

- adam optimizer for efficient training.
- binary\_crossentropy loss function for binary classification tasks.
- accuracy metric to evaluate the model's performance.



Training the Model:

epochs=50: The model will iterate 50 times over the training data. This number could be adjusted

- batch\_size=16: The number of samples per gradient update.
- 'validation\_split'=0.2: 20% of the training data is used for validation.

The model is evaluated on the test data to measure its accuracy. The results are printed.

Finally, Predictions are made on the test data. The predicted probabilities are converted to binary classes using a threshold of 0.5. The first 5 predictions and their corresponding true labels are printed.

#### 4.6 Performance Improvement with cuDF our Machine Learning Models (parallel processing)

This illustrates the potential of cuDF to enhance computational efficiency in machine learning workflows, particularly in scenarios involving large datasets and complex calculations [28].

```
%load_ext cudf.pandas
```

we used cudf to bring accelerated computing to our panda's workflows without changing our code.

**Table 4:** Model Runtime comparison with and without the code

Model	Without use the code	With use the code
Lazypredict	34s	5s
Random forest	4s	4s
Neural network	Each step took 6-8ms	Each step took 2-5ms

**Accuracy:** the percentage of correct classifications that a trained machine learning model achieve [29].

Formula: 
$$accuracy = \frac{TN+TP}{TP+TN+FP+FN}$$

Where,

(**TP**) True positive: the model correctly predicted a positive class.

(**TN**) True negative: the model correctly predicted a negative class.

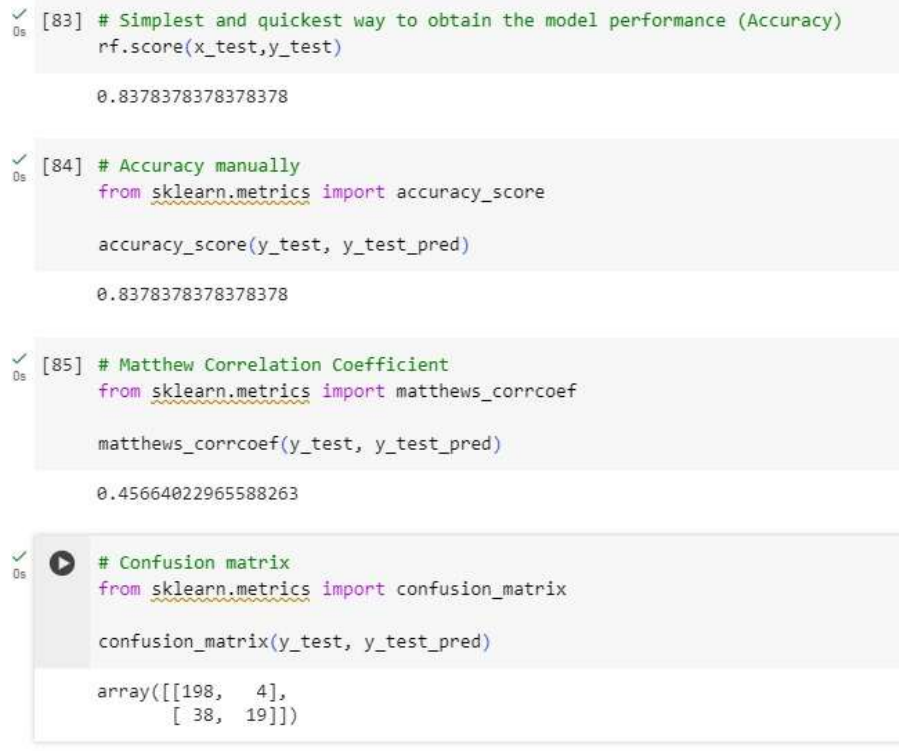
(**FP**) False positive: the model falsely predicted a positive class.

(**FN**) False negative: the model falsely predicted a negative class.

## Chapter 5: Results and Discussion

---

### 5.1 Results:



```
[83] # Simplest and quickest way to obtain the model performance (Accuracy)
      rf.score(x_test,y_test)

0.8378378378378378

[84] # Accuracy manually
      from sklearn.metrics import accuracy_score

      accuracy_score(y_test, y_test_pred)

0.8378378378378378

[85] # Matthew Correlation Coefficient
      from sklearn.metrics import matthews_corrcoef

      matthews_corrcoef(y_test, y_test_pred)

0.45664022965588263

# Confusion matrix
from sklearn.metrics import confusion_matrix

confusion_matrix(y_test, y_test_pred)

array([[198,  4],
       [ 38, 19]])
```

**Figure 20.** Accuracy results for all features combined

Now we want to increase the accuracy as much as possible, so we tried different methods to achieve this goal. In the table below we tried to combine all features in one CSV file and insert it into the code then remove a feature each time to test what that feature does to the accuracy. The results showed the highest accuracy at 92.6%. When we removed the feature TPC (Tripeptide composition) was the feature with the most negative effect on the accuracy, so we decided to exclude it.

## Feature importance

```
[ ] # Display Dataframe of the dataset after feature selection (variance threshold)
X2

[ ] # Retrieve feature importance from the RF model
importance = pd.Series(rf.feature_importances_, name = 'Gini')

# Retrieve feature names
feature_names = pd.Series(X2.columns, name = 'Feature')

# Combine feature names and Gini values into a Dataframe
df = pd.concat([feature_names, importance], axis=1, names=['Feature', 'Gini'])
df

[ ] # Plot of feature importance
import matplotlib.pyplot as plt
import seaborn as sns

df_sorted = df.sort_values('Gini', ascending=False)[:10] # Sort by Gini in descending order; Showing only the top 10 results

plt.figure(figsize=(5, 10))
sns.set_theme(style="whitegrid")
ax = sns.barplot(x = 'Gini', y = 'Feature', data = df_sorted)
plt.xlabel("Feature Importance")
```

**Figure 21.** Feature importance code.

**Table 5:** The accuracy score for all features combined then excluding one feature at a time.

the deleted feature	the accuracy score
atc	0.8687
btc	0.8571
ctc	0.861
ddr	0.87644
paac	0.861
pcp	0.8687
rri	0.8571
ser	0.8648
spc	0.8687
apaac	0.8648
dpc	0.8571
tpc	0.9266
qos	0.8725
aac	0.8648

We tried using PCA to test what it does to the accuracy. First, we tested PCA on the raw data and reduced the number of features each time we ran the code to test its effect on the accuracy of the Random Forest model. After finishing that we tried doing PCA and then normalizing the data before running it through the Random Forest model and wrote down the results. Finally, we tried normalizing the data before

putting it in the PCA code and recorded the results. In conclusion, the PCA reported the best results when the data was at 150 columns (features) and the normalization didn't affect the results significantly.

**Table 6:** PCA accuracy results.

1. PCA: Num of features	Accuracy	2.PCA + Normalization	accuracy	3.Normalization + PCA	accuracy
200	0.93				
150	0.953	150	0.953	150	0.918
100	0.94	100	0.9498	100	0.915
50	0.94	50	0.9498	50	0.861
25	0.872	25	0.861	25	0.845

Neural Network results: test accuracy = 0.899, Validation accuracy = 0.913. this result is relatively close to the RF classifier results.

```

epoch 41/50
52/52 [=====] - 0s 3ms/step - loss: 4.2123e-04 - accuracy: 1.0000 - val_loss: 0.4716 - val_accuracy: 0.9135
Epoch 42/50
52/52 [=====] - 0s 4ms/step - loss: 3.9731e-04 - accuracy: 1.0000 - val_loss: 0.4764 - val_accuracy: 0.9135
Epoch 43/50
52/52 [=====] - 0s 3ms/step - loss: 3.7536e-04 - accuracy: 1.0000 - val_loss: 0.4778 - val_accuracy: 0.9135
Epoch 44/50
52/52 [=====] - 0s 3ms/step - loss: 3.5289e-04 - accuracy: 1.0000 - val_loss: 0.4805 - val_accuracy: 0.9135
Epoch 45/50
52/52 [=====] - 0s 3ms/step - loss: 3.3628e-04 - accuracy: 1.0000 - val_loss: 0.4817 - val_accuracy: 0.9135
Epoch 46/50
52/52 [=====] - 0s 3ms/step - loss: 3.1320e-04 - accuracy: 1.0000 - val_loss: 0.4872 - val_accuracy: 0.9135
Epoch 47/50
52/52 [=====] - 0s 3ms/step - loss: 2.9443e-04 - accuracy: 1.0000 - val_loss: 0.4868 - val_accuracy: 0.9135
Epoch 48/50
52/52 [=====] - 0s 3ms/step - loss: 2.7563e-04 - accuracy: 1.0000 - val_loss: 0.4942 - val_accuracy: 0.9135
Epoch 49/50
52/52 [=====] - 0s 3ms/step - loss: 2.5891e-04 - accuracy: 1.0000 - val_loss: 0.4957 - val_accuracy: 0.9135
Epoch 50/50
52/52 [=====] - 0s 3ms/step - loss: 2.4476e-04 - accuracy: 1.0000 - val_loss: 0.5014 - val_accuracy: 0.9135
9/9 [=====] - 0s 2ms/step - loss: 0.6894 - accuracy: 0.8996
Test accuracy: 0.8996139168739319
9/9 [=====] - 0s 2ms/step
First 5 predictions: [0 0 0 1 0]
First 5 true labels: [0 0 0 1 0]

```

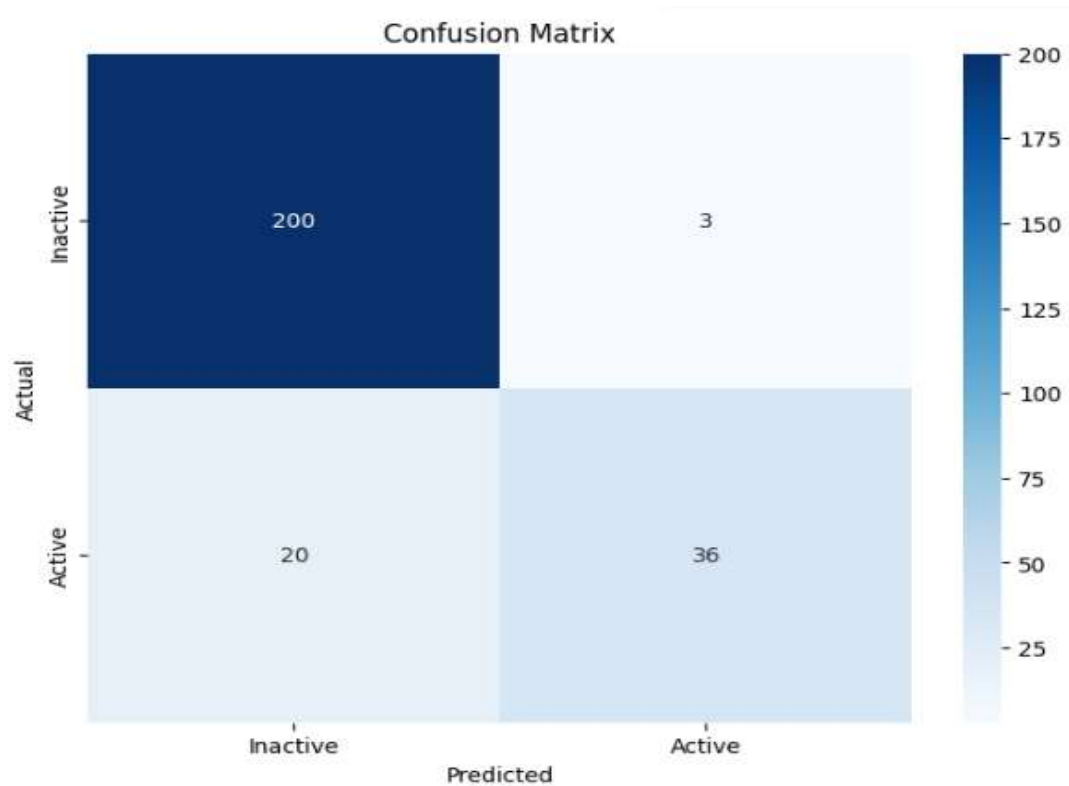
**Figure 22.** Neural network results

## 5.2 Discussion

- We initially achieved a 0.9266 accuracy with the random forest model. To further enhance this accuracy, we applied normalization and Principal Component Analysis (PCA). preprocessing steps enhanced the accuracy to 0.95, as illustrated in the PCA accuracy results Table 5. Subsequently, we employed a neural network, which yielded a test accuracy of 0.899 and a validation accuracy of 0.913. Therefore, the highest accuracy was obtained using the random forest model with PCA.
- Incorporating PCA and normalization is a good practice as it often helps in improving model performance by reducing dimensionality and scaling features.

- While the neural network performed reasonably well, its accuracy (test: 0.899, validation: 0.913) was still lower than that of the random forest. This might be due to the neural network's architecture not being complex enough for the problem at hand.

A good visualization method to show the results in an easy way to understand is using the confusion matrix.



**Figure23:** Confusion matrix

## **Chapter 6: Economical, Ethic, and Contemporary Issues**

---

### **6.1 Preliminary Cost Estimation and Justification**

There were no costs during the making of this project, the project incurred no expenses because we gathered data with assistance from Dr. Ahmed al-Omari and completed the work using our own computers and internet connection. Additionally, we utilized Jupiter (colab), a free platform, for our project needs.

### **6.2 Relevant Codes of Ethics and Moral Frameworks**

We appropriately cited all external content that was incorporated into our work.

### **6.3 Ethical Dilemmas and Justification of Proposed Solution**

In this project of "Predicting Antimicrobial Peptides Activity Using Machine Learning for Drug Design Development", several ethical dilemmas may arise that require careful consideration and justification of proposed solutions:

- **Impact on Public Health**

Dilemma: Incorrect predictions or misinterpretations of antimicrobial peptide activity could lead to ineffective or potentially harmful drug development.

Justification: We take a cautious approach by ensuring that predictions are thoroughly validated and cross-checked with experimental data before any practical application. Our findings are also subjected to peer review to prevent the dissemination of potentially misleading or harmful information.

- **Access to Treatment and Healthcare Disparities**

Dilemmas: Developing new antimicrobial therapies through advanced technologies may raise concerns about equitable access to these treatments, particularly in resource-limited regions.

Justification: Collaborate with global health organizations and regulatory bodies to ensure affordability and accessibility of developed therapies. Consider cost-effective production methods and licensing agreements to facilitate wider distribution.

### **6.4 Relevant Environmental Considerations**

There are no environmental considerations related to our project because we worked on a software system.

### **6.5 Relevance to Jordan and Region (Social, Cultural, and Political)**

Our project aims to enhance drug design efficiency and reduce costs, leading to broader positive impacts globally and specifically in Jordan.

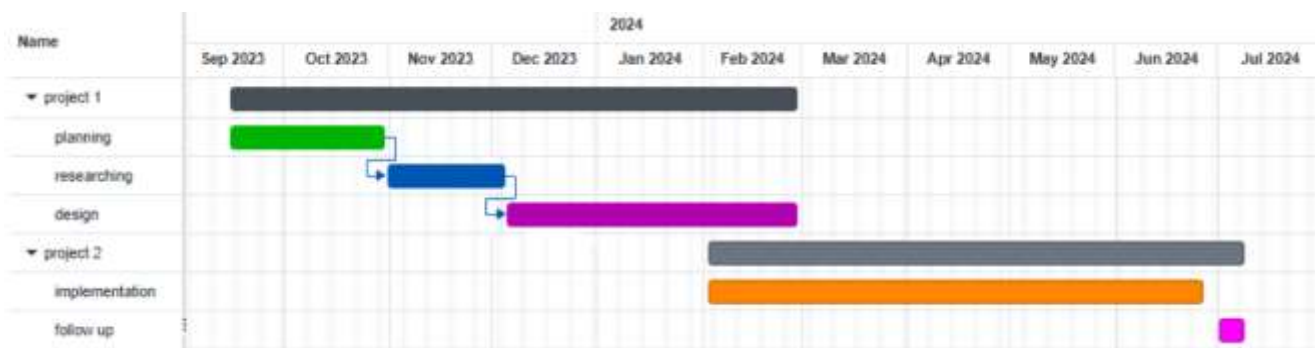
### **6.6 Other Issues and Constraints**

Validating machine learning predictions with experimental and clinical data is critical to ensure the effectiveness and safety of identified antimicrobial peptides. However, thorough validation through in lab and potentially in vivo studies is resource-intensive and time-consuming, often requiring collaboration with biomedical and pharmaceutical experts for effective validation.

## Chapter 7: Project Management

### 7.1 Schedule and Time Management

This project took close to 9 months in total from the very beginning of the planning process till the submission of the final report. We first brainstormed some ideas to work on keeping the trendy subjects in bioinformatics engineering and we met on the applications of machine learning in Bioinformatics. We started to research different topics and with the supervision of Dr. Ahmad al-Omari we were able to agree on this project's topic and title. After the research step, we started to sketch the data source and the general code needed to complete our mission with the help of the professor we started implementing our plans and collected our sketches and frequent Zoom calls to monitor the progress and help with any confusion and follow-up steps. All of that is well defined in the Gantt chart below with the time frame of each task.



**Figure 24.** Gantt chart of the project.

### 7.2 Resource and Cost Management

Regarding the resources, we acquired the initial dataset of AMPs and non-AMPs from the Computational Biology and Bioinformatics Lab Centre for AI-Driven Drug-Discovery at Macao Polytechnic University in the Antimicrobial peptide's prediction servers [18].

Google Colab is a free application provided by Google. That helped us to achieve the project's objectives with minimal costs and no risks on any device or people involved since it was software-related.

### 7.3 Quality Management

Dr. Ahmad made a lot of meetings with us to make sure everything worked perfectly and to answer our questions.

### 7.4 Risk Management

In our project there is no danger that may occur because it is a software system.

### 7.5 Project Procurement



We utilized Google Colab, a free cloud-based application provided by Google, to perform all necessary computations and model training. The datasets used for training our models were freely available online for scientific purposes. Additionally, we received a dataset directly from Dr. Ahmed al-Omari, which was instrumental to our research. This approach allowed us to effectively manage costs while accessing high-quality tools and data necessary for the successful execution of our project.

## **Chapter 8: Conclusion and Future Work**

---

### **8.1 Contributions of the work**

Our project deals with the development of a cutting-edge machine learning model specifically designed to predict AMPs activity. It makes use of modern methods in feature extraction techniques and sophisticated algorithms which enable high accuracy and reliability, thus offering a new benchmark in the subject area. A large, diverse dataset for AMPs with information on sequences and activity status has been compiled. Given that the constructed dataset could certainly support further studies and model developments for the research community in the area, it should be made available. This computational tool will, therefore, let researchers and health specialists quickly classify AMPs into active and inactive categories with much ease. Our work is therefore going to speed up the research process with fast results and accurate delivery, even within the reach of non-expert computational biologists.

In addition to its documented and guided tutorials, the project is extremely useful for a student studying AMPs, and generally in machine learning. It thus diffuses knowledge to the next generation of scientists and engineers.

### **8.2 Further future work**

#### **1. Research and Educational use:**

Designing a collaboration platform will enhance the discovery and optimization of AMP activity, taking the battle against antibiotic resistance to the community level.

#### **2. Working On Advanced Machine Learning Models**

Explore the use of advanced machine learning techniques to improve classification accuracy and capture more complex patterns and relationships within sequences.

#### **3. Development of User Interface:**

Design a user-friendly website interface to the tool for researchers and pharmacists to easily input peptide sequences and retrieve classification results. This can increase the tool's adoption and facilitate its integration into various workflows.

#### 4. Integration with Other Bioinformatics Tools:

Integrate the AMP classification tool with other bioinformatics resources and databases giving a fuller view of the functional roles and interactions of AMPs.

### 8.3 Lessons learned

- Deepened understanding of applying machine learning in bioinformatics.
- Experienced key phases of ML model construction:
  - Data collection, implementation in Python, and model training.
- Achieved classification of antimicrobial peptides (AMPs) using ML techniques.
- Applied theoretical studies to real-world problems.
- Gained insight into the model development life cycle:
  - Data acquisition, preprocessing, feature handling, training, and evaluation.
- Learned various ML classification algorithms applicable to bioinformatics.
- Enhanced teamwork and communication skills:
  - Work distribution, problem-solving, report writing, and presentation preparation
- Improved project management skills:
  - Tracking progress, meeting deadlines, and task completion.
- Acquired valuable lessons for future projects and professional growth.

## References:

- [1] P. Szymczak and E. Szczurek, “Artificial intelligence-driven antimicrobial peptide discovery,” *Current Opinion in Structural Biology*, vol. 83, pp. 102733–102733, Dec. 2023, doi: <https://doi.org/10.1016/j.sbi.2023.102733>.
- [2] A. Moretta *et al.*, “Antimicrobial Peptides: A New Hope in Biomedical and Pharmaceutical Fields,” *Frontiers in Cellular and Infection Microbiology*, vol. 11, Jun. 2021, doi: <https://doi.org/10.3389/fcimb.2021.668632>.
- [3] “How do we solve the antibiotic resistance crisis?,” *University of Plymouth*. <https://www.plymouth.ac.uk/discover/how-do-we-solve-the-antibiotic-resistance-crisis>.
- [4] Magana, M., Pushpanathan, M., Santos, A. L., Leanse, L., Fernandez, M., Ioannidis, A., et al. (2020). The Value of Antimicrobial Peptides in the Age of Resistance. *Lancet Infect. Dis.* 20, e216–e230. doi: 10.1016/S1473-3099(20)30327-.
- [5] G. Wang, X. Li, and Z. Wang, “APD3: the antimicrobial peptide database as a tool for research and education,” *Nucleic Acids Research*, vol. 44, no. D1, pp. D1087–D1093, Nov. 2015, doi: <https://doi.org/10.1093/nar/gkv1278>.
- [6] Luong, H. X., Thanh, T. T., Tran, T. H. (2020). Antimicrobial Peptides – Advances in Development of Therapeutic Applications. *Life. Sci.* 260, 118407. doi: <https://doi.org/10.1016/j.lfs.2020.118407>.
- [7] H. X. Luong, T. T. Thanh, and T. H. Tran, “Antimicrobial peptides – Advances in development of therapeutic applications,” *Life Sciences*, vol. 260, p. 118407, Nov. 2020, doi: <https://doi.org/10.1016/j.lfs.2020.118407>.
- [8] J. Yan et al., “Deep-AmPEP30: Improve Short Antimicrobial Peptides Prediction with Deep Learning,” *Molecular Therapy. Nucleic Acids*, vol. 20, pp. 882–894, Jun. 2020, doi: <https://doi.org/10.1016/j.omtn.2020.05.006>.
- [9] Broad Institute, “What is Mass Spectrometry?,” Broad Institute, Sep. 13, 2010. <https://www.broadinstitute.org/technology-areas/what-mass-spectrometry>.
- [10] P. Szymczak and E. Szczurek, “Artificial intelligence-driven antimicrobial peptide discovery,” *Current Opinion in Structural Biology*, vol. 83, pp. 102733–102733, Dec. 2023, doi: <https://doi.org/10.1016/j.sbi.2023.102733>.
- [11] Wikipedia Contributors, “Python (programming language),” Wikipedia, May 04, 2019. [https://en.wikipedia.org/wiki/Python\\_\(programming\\_language\)](https://en.wikipedia.org/wiki/Python_(programming_language)).
- [12] “Python logo,” *Wikipedia*, Aug. 29, 2021. [Online]. Available: <https://en.m.wikipedia.org/wiki/File:Python-logo-notext.svg>. [Accessed: Aug. 4, 2024].
- [13] “Google Colab,” [research.google.com](https://research.google.com/colaboratory/faq.html#:~:text=Colab%20is%20a%20hosted%20Jupyter). <https://research.google.com/colaboratory/faq.html#:~:text=Colab%20is%20a%20hosted%20Jupyter>

- [14] "Google CoLaboratory logo," **Logowik**, [Online]. Available: <https://logowik.com/vector-logo/Google%20CoLaboratory%20logo%20vector.html>. [Accessed: Aug. 4, 2024].
- [15] "Machine Learning," AIT Skadapa, [Online]. Available: [https://aitskadapa.ac.in/e-books/AI&ML/MACHINE%20LEARNING/Machine%20Learning%20\(%20etc.\)%20\(z-lib.org\).pdf](https://aitskadapa.ac.in/e-books/AI&ML/MACHINE%20LEARNING/Machine%20Learning%20(%20etc.)%20(z-lib.org).pdf). [Accessed: Aug. 4, 2024].
- [16] F. Chollet, "Artificial Intelligence, Machine Learning, and Deep Learning," in *Deep Learning*, MIT Press, 2018. [Online]. [Accessed: Aug. 4, 2024].
- [17] "Computational Biology and Bioinformatics," Computational Biology and Bioinformatics, Jan. 06, 2024. <https://cbbio.online/> (accessed Jul. 05, 2024)
- [18] D. Gunay, "Random Forest," *Medium*, Sep. 14, 2023. <https://medium.com/@denizgunay/random-forest-af5bde5d7e1e>.
- [19] "Pfeature/Pfeature\_Manual.pdf" at master.raghavagps/Pfeature, GitHub. [https://github.com/raghavagps/Pfeature/blob/master/Pfeature\\_Manual.pdf](https://github.com/raghavagps/Pfeature/blob/master/Pfeature_Manual.pdf).
- [20] J. Yan et al., "Deep-AmPEP30: Improve Short Antimicrobial Peptides Prediction with Deep Learning," *Molecular Therapy. Nucleic Acids*, vol. 20, pp. 882–894, Jun. 2020, doi: <https://doi.org/10.1016/j.omtn.2020.05.006>.
- [21] A. Al-Omari et al., "Big Data Bot with a Special Reference to Bioinformatics," *Computers, Materials & Continua*, vol. 75, no. 2, pp. 4155–4173, 2023, doi: <https://doi.org/10.32604/cmc.2023.036956>.
- [22] "Home," sites.google.com. <https://sites.google.com/view/cd-hit>.
- [23] GitHub, "GitHub," GitHub, 2024. <https://github.com/>.
- [24] S. P. Raghava Anjali Dhall, Akshara Pande, Dilraj Kaur, Neelam Sharma, Gaurav Mishra, Rajesh Kumar, Piyush Agrawal, Vinod Poriya, Anjali Lathwal, Harpreet Kaur, Shipra Jain, Prof GPS,
- [25] "Pandas Introduction," [www.w3schools.com. https://www.w3schools.com/python/pandas/pandas\\_intro.asp#:~:text=Pandas%20is%20a%20Python%20library](https://www.w3schools.com/python/pandas/pandas_intro.asp#:~:text=Pandas%20is%20a%20Python%20library)
- [26] "Python Numpy," GeeksforGeeks, Jun. 13, 2024. [https://www.geeksforgeeks.org/python-numpy/?ref=header\\_search](https://www.geeksforgeeks.org/python-numpy/?ref=header_search) (accessed Jul. 07, 2024).
- [27] Wikipedia Contributors, "scikit-learn," *Wikipedia*, Mar. 19, 2019. <https://en.wikipedia.org/wiki/Scikit-learn>.
- [28] "RAPIDS cuDF Accelerates pandas Nearly 150x with Zero Code Changes," *NVIDIA Technical Blog*, Mar. 18, 2024. <https://developer.nvidia.com/blog/rapids-cudf-accelerates-pandas-nearly-150x-with-zero-code-changes/> (accessed Jul. 28, 2024).
- [29] "What is Model Accuracy in Machine Learning," Iguazio. <https://www.iguazio.com/glossary/model-accuracy-in-ml/#:~:text=AI%20accuracy%20is%20the%20percentage>.

## APPENDIX: Code

```
[ ] #Conda
#####
# INSTALL CONDA ON GOOGLE COLAB
#####
! wget https://repo.anaconda.com/miniconda/Miniconda3-py37_4.8.2-Linux-x86_64.sh
! chmod +x Miniconda3-py37_4.8.2-Linux-x86_64.sh
! bash ./Miniconda3-py37_4.8.2-Linux-x86_64.sh -b -f -p /usr/local
import sys
sys.path.append('/usr/local/lib/python3.7/site-packages/')
```

Show hidden output

### Download and Install Pfeature

```
[ ] ! wget https://github.com/raghavagps/Pfeature/raw/master/PyLib/Pfeature.zip
```

Show hidden output

```
[ ] ! unzip Pfeature.zip
```

Show hidden output

```
%cd /content/Pfeature
```

Show hidden output

```
[ ] ! python setup.py install
```

Show hidden output

Figure 25. Install libraries

### Install CD-HIT

```
[ ] ! conda install -c bioconda cd-hit -y
```

Show hidden output

### Load peptide dataset

```
[ ] ! wget https://raw.githubusercontent.com/dataprofessor/AMP/main/test_po.fasta
```

Show hidden output

```
[ ] ! wget https://raw.githubusercontent.com/dataprofessor/AMP/main/train_ne.fasta
```

### Remove redundant sequences using CD-HIT

```
[ ] ! cd-hit -i test_po.fasta -o test_po_cdhit.txt -c 0.99
```

```
[ ] ! cd-hit -i output.txt -o F.txt -c 0.99
```

Figure 26. Download Data

## ✓ Calculate features using the Pfeature library

Feature classes provided by Pfeature is summarized in the tables below.

### Composition Based Features

Feature class	Description	Function
AAC	Amino acid composition	aac_wp
DPC	Dipeptide composition	dpc_wp
TPC	Tripeptide composition	tpc_wp
ABC	Atom and bond composition	atc_wp, btc_wp
PCP	Physico-chemical properties	pcp_wp
AAI(X)	Amino acid index composition	aai_wp
RRI	Repetitive Residue Information	rri_wp
DDR	Distance distribution of residues	ddr_wp
PRI(X)	Physico-chemical properties repeat composition	pri_wp
SEP	Shannon entropy	sep_wp
SER	Shannon entropy of residue level	ser_wp
SPC	Shannon entropy of physicochemical property	spc_wp
ACR(X)	Autocorrelation	acr_wp
CTC	Conjoint Triad Calculation	ctc_wp
CTD(X)	Composition enhanced transition distribution	ctd_wp
PAAC	Pseudo amino acid composition	paac_wp
APAAC	Amphiphilic pseudo amino acid composition	apaac_wp

### ✓ Define functions for calculating the different features

```
[ ] from Pfeature.pfeature import ctc_wp
import pandas as pd

def ctc(input):
    a = input.rstrip('txt')
    output = a + 'ctc.csv'
    df_out = ctc_wp(input, output)
    df_in = pd.read_csv(output)
    return df_in

ctc('/content/Pfeature/P.txt')
```

	CTC_111	CTC_112	CTC_113	CTC_114	CTC_115	CTC_116	CTC_117	CTC_121	CTC_122	CTC_123	CTC_124	CTC_125	CTC_126	C
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.5	0.0	0.0	0.0	
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.5	0.0	0.0	0.0	
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	
4	0.5	1.0	0.0	0.0	0.5	0.0	0.0	0.5	0.0	0.0	0.0	0.5	0.0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
1290	0.0	0.0	0.0	0.0	0.5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
1291	0.0	0.0	0.0	0.0	0.5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
1292	1.0	0.0	0.0	0.0	0.5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
1293	1.0	0.5	0.0	0.0	0.5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
1294	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	1.0	0.0	

**Figure 27.** Quantify peptide features

<pre> pos = 'T_positive.txt' neg = 'T_neg.txt'  def feature_calc(po, ne, feature_name):     # Calculate feature     po_feature = feature_name(po) # same as: aac(pos)     ne_feature = feature_name(ne)     # Create class labels     po_class = pd.Series(['active' for i in range(len(po_feature))])     ne_class = pd.Series(['inactive' for i in range(len(ne_feature))])     # Combine po and ne     po_ne_class = pd.concat([po_class, ne_class], axis=0)     po_ne_class.name = 'class'     po_ne_feature = pd.concat([po_feature, ne_feature], axis=0)     # Combine feature and class     df = pd.concat([po_ne_feature, po_ne_class], axis=1)     return df  featureDDR = feature_calc(pos, neg, ddr) featureDDR </pre>											
.AL	...	DPC1_YN	DPC1_YP	DPC1_YQ	DPC1_YR	DPC1_YS	DPC1_YT	DPC1_YV	DPC1_YW	DPC1_YY	class
.76	...	0.00	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	active
.00	...	0.00	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	active
.76	...	0.00	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	active
.00	...	0.00	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	active
.00	...	0.00	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	active
...	...	...	...	...	...	...	...	...	...	...	...
.00	...	0.00	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	inactive

**Figure 28.** Add Class Column



## ✓ Data pre-processing

```
[ ] # Assigns the features to X and class label to Y
X = df.drop('class', axis=1)
y = df['class'].copy()

[ ] # Encoding the Y class label
y = df['class'].map({"active": 1, "inactive": 0})

[ ] # Feature selection (Variance threshold)
from sklearn.feature_selection import VarianceThreshold

fs = VarianceThreshold(threshold=0.1)
fs.fit_transform(X)
#X2.shape
X2_tpc = X.loc[:, fs.get_support()]
X2_tpc
#remove low variance data

[ ] # Data split
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X2_tpc, y, test_size=0.2, random_state =42, stratify=y)
```

Figure 29. Data Preprocessing

## ✓ Quickly compare >30 ML algorithms

```
[ ] %cd lazypredict
Show hidden output

[ ] ! git clone https://github.com/shankarpandala/lazypredict.git
%cd lazypredict
! python setup.py install
Show hidden output

[ ] # Import libraries
import lazypredict
from lazypredict.Supervised import LazyClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import matthews_corrcoef

# Load dataset#####
#X = df2.drop('class', axis=1)
#y = df2['class'].map({"active": 1, "inactive": 0})#df['class'].copy()

# Data split
X_train, X_test, y_train, y_test = train_test_split(principalDf, y, test_size=0.2, random_state =42, stratify=y)

# Defines and builds the lazyclassifier
clf = LazyClassifier(verbose=0,ignore_warnings=True, custom_metric=matthews_corrcoef)

models_train,predictions_train = clf.fit(X_train, X_train, y_train, y_train)
models_test,predictions_test = clf.fit(X_train, X_test, y_train, y_test)
```

Figure 30. Lazypredict Library

```

import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

# Split the data into features (X) and labels (y)
X = df.drop('class', axis=1).values
y = df['class'].values

# Convert string labels to numerical using LabelEncoder
label_encoder = LabelEncoder() # Initialize LabelEncoder
y = label_encoder.fit_transform(y) # Transform labels to numerical

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Normalize the features
mean = X_train.mean(axis=0)
std = X_train.std(axis=0)
X_train = (X_train - mean) / std
X_test = (X_test - mean) / std

# Build the neural network model
model = Sequential()
model.add(Dense(64, input_shape=(X_train.shape[1],), activation='relu'))
model.add(Dense(64, activation='relu'))
# Use number of classes in output layer, assuming binary classification here
model.add(Dense(1, activation='sigmoid'))

# Compile the model
# Use binary_crossentropy for binary classification
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Train the model
model.fit(X_train, y_train, epochs=50, batch_size=16, validation_split=0.2)

# Evaluate the model
loss, accuracy = model.evaluate(X_test, y_test)
print(f"Test accuracy: {accuracy}")

# Make predictions
predictions = model.predict(X_test)
# Adjust prediction threshold if needed (e.g., 0.5)
predicted_classes = (predictions > 0.5).astype(int)

# Display the first 5 predictions
print("First 5 predictions:", predicted_classes[:5].flatten()) # Flatten to 1D array
print("First 5 true labels:", y_test[:5])

```

**Figure 31.** Neural Network code

## ✓ Random Forest

```
[ ] # Build random forest model  
  
from sklearn.ensemble import RandomForestClassifier  
  
rf = RandomForestClassifier(n_estimators=500)  
  
rf.fit(X_train, y_train)
```



```
RandomForestClassifier  
RandomForestClassifier(n_estimators=500)
```

## ✓ Apply the model to make predictions

```
[ ] y_train_pred = rf.predict(X_train)  
    y_test_pred = rf.predict(X_test)
```

## ✓ Model performance

```
[ ] # Simplest and quickest way to obtain the model performance (Accuracy)  
    rf.score(X_test, y_test)
```



```
0.9575289575289575
```


```
[ ] # Accuracy manually  
    from sklearn.metrics import accuracy_score  
  
    accuracy_score(y_test, y_test_pred)
```

**Figure 32.** Random Forest model and checking model performance

```
[ ] # Matthew Correlation Coefficient
    from sklearn.metrics import matthews_corrcoef

    matthews_corrcoef(y_test, y_test_pred)
```

 Show hidden output

```
 # Confusion matrix
    from sklearn.metrics import confusion_matrix

    confusion_matrix(y_test, y_test_pred)
```

 Show hidden output

```
[ ] # Classification report
    from sklearn.metrics import classification_report

    model_report = classification_report(y_train, y_train_pred, target_names=['positive', 'negative'])

    f = open('model_report_tcp.txt', 'w')
    f.writelines(model_report)
    f.close()
```

**Figure 33.** Checking model performance