Pepfeature: A Python package for feature calculation of protein epitope data

Student Name Essa Umar Khan

University Aston University

Module Code CS3010

Module Name Individual Project

Student Number 170077653

Contact Email khane2@aston.ac.uk

Supervisor Prof. Dr. Felipe Campelo

Contact Email f.campelo@aston.ac.uk

Word Count

Submission Date

Contents

[1. Introduction 5](#_Toc61849155)

[1.1 Project Explanation 5](#_Toc61849156)

[1.2 Motivations 5](#_Toc61849157)

[1.3 Contributions 6](#_Toc61849158)

[1.4 Report Structure 7](#_Toc61849159)

[2. Background research 7](#_Toc61849160)

[2.1 What is epitope prediction? 7](#_Toc61849161)

[2.2 The epitope dataset 7](#_Toc61849162)

[2.3 Feature Extraction 8](#_Toc61849163)

[2.3.1 Conjoint Triad Frequencies 9](#_Toc61849164)

[2.4 Technologies used 11](#_Toc61849165)

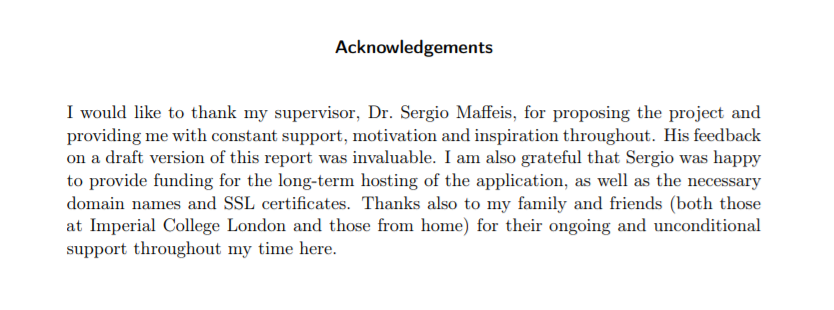
[2.5 Existing Solutions 11](#_Toc61849166)

[3. Preparation 12](#_Toc61849167)

[3.1.1 What is a MoSCoW hierarchy? 12](#_Toc61849168)

[3.1.2 What are functional & non-functional requirements? 12](#_Toc61849169)

[3.2 Requirements 13](#_Toc61849170)





# Introduction

## What is computational epitope prediction and why is it important?

Linear B-cell epitopes are short strings of amino acids within a continuous stretch of a protein sequence (Wang et al., 2011); these are located on the surface of an antigen, they trigger an immune response and are recognized by specific antibodies, which bind to and remove the antigen from the body (Wang and Pai, 2014).

There exists motivation for the identification of linear B-cell epitopes as they provide the underpinning knowledge for the development of epitope-based vaccines, diagnostic tests, and various disease prevention techniques (Potocnakova, Bhide and Pulzova, 2016). Fortunately, the increasing availability of verified epitope databases has paved the way for the utilization of computational techniques for linear B-cell epitope prediction (ibid.). Compared to experimental identification, computerized techniques for prediction are more efficient as it demands lesser resources, time and effort (EL-Manzalawy and Honavar, 2010). Computational prediction serves as a prioritization strategy for laboratory validation - it is not a substitute for laboratory identification but rather a pre-filtering tool to facilitate more efficient investigation of the most promising targets.

Computational methods for prediction consist (but not solely) of algorithms (/pipelines?) that predict linear B-cell epitopes based on data derived from the protein sequence of the anti-gen (Sanchez-Trincado, Gomez-Perosanz and Reche, 2017). Feature calculation is an important part of the development of good prediction pipelines for this specific data, i.e., the generation of the most informative features that can help an algorithm such as a classifier, correctly identify the most promising regions from what would be a linear B-cell epitope (also called potentially immunogenic peptides (PIPs) in a given protein sequence.

The focus of this project is the development of a python package that consists of routines that calculate features in epitope data frames (i.e. feature extraction) for training epitope classification models.

Proteins are composed of 20 amino acids, represented by different letters, the valid letters are: A, C, D, E, F, G, H, I, K, L, M, N, P, Q, R, S, T, V, W, Y.

## Contributions

The main contributions of this project are summarised below:

• A publicly available Python package named Pepfeature consisting of feature calculation routines for protein Epitope data, namely: Amino Acid (AA) frequencies (20 features), Conjoint Triads (343 features), K-mer frequencies (400+ features), AA descriptors (XYZ features), Sequence Entropy (XYZ features), Molecular Weight (XYZ features) and Number of Atoms. This package was designed and implemented in a way so that it is capable of integrating with the client’s classification pipeline (see section 1.3 for more information) but be ﬂexible enough for general use too for wider use by the scientific community, and designed to ease future additions to the package.

• Systematic evaluation of performance of the implemented routines in Pepfeature with that of the existing R package solution (named ‘Epitopes’), covering the aspect of runtime and accuracy.

## Context & Clients

To understand the justification for the development of these feature extraction capabilities in Python, an understanding of the context in which it is being developed is needed and this is elaborated as follows:

The main client of this project is Jodie Ashford - a researcher in the field of machine learning approaches for epitope prediction. She uses a classification pipeline (see Figure 1) for her research in which currently the model fitting & prediction block use Python tools and the preparation for these i.e. the data retrieval, consolidation & feature calculation (i.e. feature extraction) block of the pipeline utilize tools coded in the R programming language. In particular, the feature calculation block is facilitated by the usage of a package coded in R named ‘Epitope’ (see section for more details).

The Python package deliverable of this project is part of a collective effort to transform the current mixed-languages prediction pipeline into a full Python pipeline. This project does this by the Refactoring of code from the R ‘Epitope’ package to a novel Python package (based on first principles of what those features are intended to calculate rather than simple reproduction of the existing R implementation). Therefore, the output of this project is the substitute Python solution to execute the ‘feature calculation’ block of the epitope prediction pipeline (see Figure 1). Likewise, there are other students working on developing (for their own final year projects) Python solutions for other different blocks of this pipeline, down & up-stream to the feature calculation block.

Besides fulfilling the requirement of the main client (Jodie) and having the package only locally available for use to her, the package is also available through *pypi* to the wider Python community. Moreover, Dr. Campelo acts as both project supervisor and as part-client, representing the wider research community in Epitope Prediction – to whom the public Package will also be useful.

## Justification

Although the R package suffices the main client needs for her feature calculation block (see Figure 1), but a Python package alternative holds the potential to bring multiple improvements in many aspects to this process. These client focused benefits are elaborated as follows.

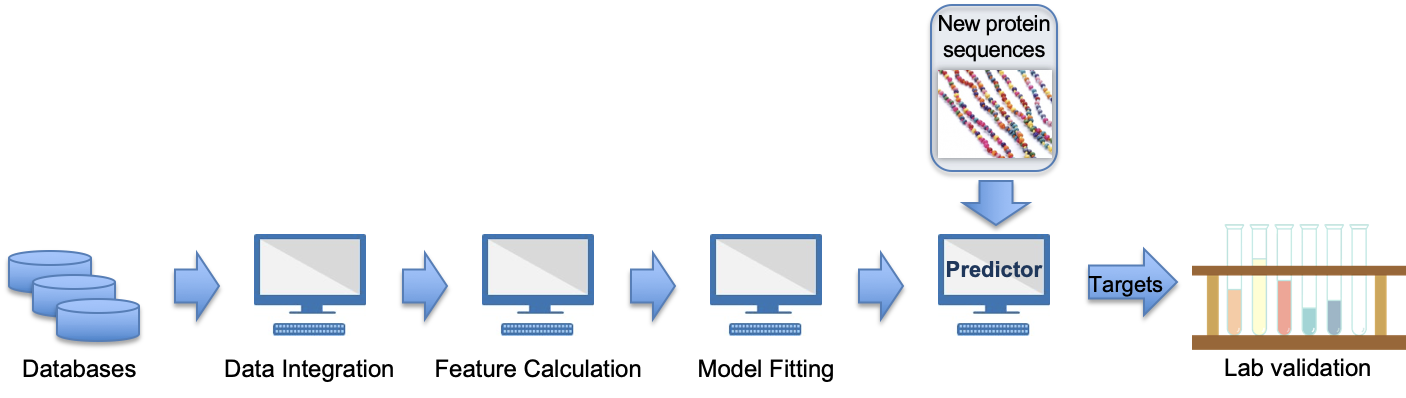


Figure : Client’s Machine Learning Pipeline [Source: Dr. Felipe Campelo, private communication]

For the client, the utilization of the envisioned Python solution is in practice more convenient and less complex to work with than the current pipeline, as it eliminates the need to switch from an R package for the pipeline’s ‘Feature Extraction’ block to a Python package for the ‘Model Fitting’ block (see section). This benefit of this is that the updated pipeline will be executable completely from within a single IDE or another deployment environment.

Moreover, as there is another student working on developing a Python solution for the ‘Data Integration’ block - which directly precedes the ‘Feature Calculation’ block - as part of a collective effort to transform Jodie’s pipeline into a pure Python pipeline (see section for more information 1.3); in such an outcome the Pepfeature package integrated in this envisioned Pythonic pipeline will result in seamless compatibility between the inputs & outputs of the blocks directly upstream and downstream respectively since, Pepfeature’s outputs also connects seamlessly with the ‘Model Fitting’ block which is directly down-stream of the ‘feature calculation’ block if integrated in the current existing pipeline.

All of the aforementioned justification is based on the intuitively derived general principle that the least variance there is in the programming languages utilized to code the modules in a prediction pipeline, the less complex development, collaboration, version control, package control, distribution and testing become. There is great motivation for diminishing complexity, as it saves resources and time.

Additional motivation for the development of Pepfeature is that it is a consolidated, publicly available Python package to calculate protein-based features; the utility of this is already excellent and a contribution to the diverse Python ecosystem of scientific libraries that makes it a preferred tool for data mining. Results from a series of R vs Python speed benchmark tests conducted by Korstanje (2020) also suggest that Python may carry out the feature calculation tasks in less time; for the end user this provides the benefit of training their models faster, as Machine Learning pipelines can be used iteratively to continuously improve the model (Kunft et al., 2019). This will be verified through benchmarks against the existing R package during the benchmarking stages of the deliverable.

## Report Structure

The remainder of the report is organised as follows:

• Section 2 consists of an overview of the key research done in preparation & during the development of Pepfeature and further necessary theory that this deliverable relied on.

• Section 6 produces the conclusion of our work, summarizing what we covered in this report. Additionally, It provides a guide to further research that can be conducted in this ﬁeld.

# Background research

Here things that are needed to know to develop the project are expounded.

## Existing Solutions

In this section the research and analysis of existing solutions are summarised. This was conducted for the purpose of

### The Model ‘R’ package: Epitopes

‘Epitopes’ is a prototype R package for feature extraction and processing of epitope data from the Immune Epitope Database (Campelo, Ashford and Lobo, 2021). The developers of Epitopes include this project’s supervisor/secondary client Dr. Felipe Campelo and the main client, Jodie Ashford. The main client currently uses this R package to facilitate both her ‘Feature Calculation’ & ‘Data Integration’ blocks of her *Epitope* prediction pipeline (see Figure 1). The routines this Rpackage calculates are the same features that Pepfeatures is aimed to calculate, thus the latter can be thought of as a partial refactoring of the R ‘Epitope’ package to a novel Python package. It will only be a partial port because the Epitopespackage also provides additional processing routines that facilitate the ‘Data Integration’ block of the epitope prediction pipeline (see Figure 1), which are outside the scope of requirement forthis project*.* The features calculated by this package (and Pepfeature) are detailed in section 2.4 (feature extraction capabilities to develop).

The source code of this package was not public during most of the development of this project, and was only released in early 2021. This package was not specifically made for the client’s pipeline - rather she adopted around it. Pepfeature on the other hand, is specifically made to integrate with the client’s pipeline.

I did not have direct access to this R package throughout the development of Pepfeature. Pepfeature’s routines’ development was instead based on first principles of what those features are intended to calculate. This was an intentional development decision, so that Pepfeature’s code development would not be influenced by specific design decisions of the Epitopes package, thereby increasing the likelihood of developing a novel way of calculating those features which may potentially be more efficient; and to ensure that its distinctions will make it a worthwhile alternative of a package to the scientific community to whom it will be made available.

### Pfeature Python package

Pfeature is a collection of routines for computing a wide range of protein & peptide features. This tool is capable of calculating more features than this project’s deliverable (Pepfeature). Its developers consist of multiple scientists.

Pfeature can be used in two main ways:

1) A webserver that provides the Pfeature’s functionality via a web interface[[1]](#footnote-1)

2) A python package

A detailed study of this package’s code was not conducted for the same reasons stated for the case of the reference R package (see section above): to ensure that Pepfeature’s code development would not be influenced by Pfeature’s design decisions, to increase the likelihood of coming up with novel ideas and to ensure that it is different and a real worthwhile distinctive alternative of a package to the scientific community to whom it will be made available.

Pfeature is a direct competitor but what distinguishes Pepfeature from this package is that firstly it is not on *pypi* for the public to install in a manner that is conventionally considered as a professional way to install python packages with ease – the install is done through github instead. Secondly, Pepfeatue is an in-house package that can and will be customized to the client’s needs, but at the same time have a level of flexibility so that it could be utilized by the wider scientific community. The features that Pepfeature calculates are focused/niched for epitope prediction whereas Pfeature is capable of calculating an enormous amount of features that are relevant for protein and peptides but not necessarily relevant for epitope prediction. Moreover, Pepfeature will be expandable enough to allow others to add more features if they desire.

## Feature Extraction capabilities to develop

In this section the underpinning theoretical and mathematical concepts of the feature extraction routines that are implemented in Pepfeature are expounded. This lays the groundwork for understanding the logical steps taken during the programmatic implementation. It also provides a simple example of feature extraction to give the reader a taste of the various quantities involved. A description of each feature that will be calculated are given along with a mathematical expression of the feature calculation process – these mathematical expressions are an aid in understanding, formulation and in the translation of the logic to algorithmic implementations of the routines to Python code.

Proteins are composed of 20 natural amino acids, represented by different letters. In the subsections that follow these 20 natural amino acids are referred to which are: A, C, D, E, F, G, H, I, K, L, M, N, P, Q, R, S, T, V, W, Y.

### Proportion of Individual Amino Acids in sequence

These are the simplest features to calculate. It consists of the proportion (out of 1) of all 20 natural amino acids in a given peptide. Thus, this feature calculation gives 20 newly derived features for each peptide.

A peptide composed of number of amino acids can be expressed as:

The amino acid frequencies features for a particular peptide e.g. , can be expressed as:

where ; is the number of amino acids of type in the peptide, where corresponds to the 20 natural amino acids (.

### -mer Composition

-mers in this project’s context are -length contiguous combinations of subsequences of amino acid letter(s) in a peptide.

As a visual example, all the possible *k*-mers of a sequence are shown in Table 1.

Table : k-mers for the sequence: DVHIE

|  |  |
| --- | --- |
| **k** | **k-mers** |
| 1 | D, V, H, I, E |
| 2 | DV, VH, HI, IE |
| 3 | DVH, VHI, HIE |
| 4 | DVHI, VHIE |
| 5 | DVHIE |

The process of -mers where =3 can be expressed as follows:

A peptide composed of number of amino acids can be expressed as:

Next, successively sliding windows with continuous three amino acid sequences gives: .

Generally, will have -mers and is the count of total possible -mers, where is the number of possible monomers. For example, since there are 20 valid amino acid letters, there are 400 () possible 2-letter combinations, 8000 () 3-letter combinations, etc.

The -mer composition features that represent a particular peptide with , can be expressed as:

where ; denotes the set of all different -mers in and is the count of elements in this set (i.e. the count of -mers in ); for the sequence of the -th type, where . Let be the count of positions in where occurs; corresponds to the -th k-mers in .

All other possible k-mer sequences (amounting to ) that are by default assumed to have zero as their frequency value within as their feature value.

### Conjoint Triad Frequencies

Conjoint Triad Frequency Features (CTFF) are based on structural neighbours, where amino acids are assigned to one of 7 classes based on their physiochemical properties. The groups for the 20 amino acids are shown in Table xx.

Table : Classification Of Amino Acids

|  |  |
| --- | --- |
| Group | Amino Acid belonging to the Group |
| 0 | A, G, V |
| 1 | C |
| 2 | F, I, L, P |
| 3 | M, S, T, Y |
| 4 | H, N, Q, W |
| 5 | K, R |
| 6 | D, E |

To calculate the CTFF of a peptide, first represent each amino acid in the peptide by its group value. For example, for the following sequence below:

MVRKGEKKKAKP ---> 305506555052

Secondly, calculate the frequencies of each 3-number subsequence. By this way, a peptide is represented by 343 (7 × 7 × 7) calculated new features (Wang and Hu, 2015). This can be expressed as follows:

A peptide composed of number of amino acids can be expressed as:

For the next step, successively sliding windows with continuous three amino acid sequences gives: .

The CTFF that represent a particular peptide , can be expressed as:

Where , and is the count of positions in where the -th triad type of all contiguous three amino acids occur, where values for correspond to the -th triads (= 1, 2, 3, ⋯, 343) (Yang, Xia and Gui, 2010).

### Sequence Entropy

To have a measure of the complexity of a peptide at its amino acid composition level its entropy is calculated. Calculating the entropy for a given peptide results in a single feature can be expressed as follows:

A peptide composed of number of amino acids can be expressed as:

And Sequence Entropy feature for is calculated using the following function :

where is the proportion of occurrence of the -th amino acid in , where values for correspond to the -th amino acids (= 1, 2, 3, ⋯, 20).

### Frequency of AA types

For a peptide, the resulting features here are calculations of the amount of amino acids of a particular class. The amino acid’s are classified based on their physiochemical characteristics shown in Table

Table : Classification Of Amino Acids

|  |  |
| --- | --- |
| **Class** | **Amino Acids belonging to class** |
| Tiny | "A", "C", "G", "S", "T" |
| Small | "A", "B", "C", "D", "G", "N", "P", "S", "T", "V" |
| Aliphatic | "A", "I", "L", "V" |
| Aromatic | "F", "H", "W", "Y" |
| NonPolar | "A", "C", "F", "G", "I", "L", "M", "P", "V", "W", "Y" |
| Polar | "D", "E", "H", "K", "N", "Q", "R", "S", "T", "Z" |
| Charged | "B", "D", "E", "H", "K", "R", "Z" |
| Basic | "H", "K", "R" |
| Acidic | "B", "D", "E", "Z" |

The process of calculating this can be expressed as follows:

for LLLLLLLLDVHIESG

The frequencies are as followes:

Tiny: {G, S} -> 2/15

Small: {D, V, G, S} -> 4/15

Aliphatic: {L,L,L,L,L,L,L,L,V,I} -> 10/15

...

Charged: {D, E, H} -> 3/15

### Number of atoms

### Molecular Weight

### AA descriptors

# Project management

## Interactions with clients

Communication with the main-client throughout the development of Pepfeature was solely through email correspondence and with the secondary client through both email and Microsoft Teams meetings. The meetings with Dr. Campelo were audio recorded as well as extensive notes were taken, which helped in recalling information in instances where substantial breaks were taken from working on the final year project. They mainly took the format of a session where I would prepare questions before hand and ask them or engage in consultations and he would offer guidance.

It has to be noted that the secondary client, Dr. Campelo, is also the supervisor of both Jodie (the main-client; as her PhD Supervisor) and myself. This meant that Dr. Campelo had a comprehensive awareness of the details and progress of both of our projects and therefore was to a sufficient extend able to guide me in matters pertaining to Jodie’s requirements. For instance, the information and clarifications pertaining to the necessary details regarding Jodie’s pipeline were given to me by Dr. Campelo – it was more convenient and efficient for me to obtain such information through him as due to his dual role as a supervisor he was able to contextualize and simplify the complex information to my level of exact needs.

However, the main client was of course consulted on distinct occasions to obtain feedback regarding the deliverable and whether it met her expectations. Besides the initial contact at the start of the project, the main client was consulted after the very first feature calculation algorithm that was coded. In this instance, the main client provided feedback that solidified my understanding of her expectations. As this early implementation was approved with only some pointers to improvement, it gave assurance that the requirements had been interpreted correctly and paved the way for how to approach the rest of the implementations of features. Once the deliverable was considered to meet all the set requirements, it was requested that the main client test it out and a answer a set of objective questions, so as to provide feedback systematically (see section …. for main-client feedback).

Furthermore, in section 2.1.1 it was expounded that both the clients were involved in the development of the R package – the existing solution in use by the main client - of which Pepfeature is a partial port of. Due to this the clients were naturally already aware of their desired requirements for a Python package replacement, viz. Pepfeature, that delivers practically the same feature extraction functionalities as their own developed *epitopes* R package. Thus, a systematic and intricate requirement elicitation prior to the development of the package was not needed – the initial requirements were handed over and settled over a span of a few email communications between the clients and myself. Any further changes and expansion to these initial requirement were carried out with the consultation of the secondary client as his guidance was certain to be compatible with the main client needs, and it was more straightforward to consult him in regards to these matters during our bi-weekly one-to-one supervisory meetings pertaining to the progress of the project.

## Set Requirements and Formulation

The initial requirements for the Python package Pepfeature were mutually agreed upon amongst the clients and myself in the form of a MoSCoW[[2]](#footnote-2) hierarchy (Appendix 1). However, these initial requirements just gave a macro overview of what was required. The clients gave me the freedom to choose the Pythonic tools and approach to take to meet these requirements as long as it conforms to be a self-contained Python package in the end. Therefore, during the course of development, through consultations with the partial client certain design decision – micro requirements – were formulated for the ultimate purpose of achieving these major requirements stated in Table 4 – these design decisions and the rationale behind these are expounded upon in Design and Implementation section.

Table 4 presents the requirements for the Python Package. This table also captures the priority with what things should be approached with through the ‘MoSCoW Rank’ column. This priority was set and informed by the clients based on their personal experience of what feature calculation capabilities would likely be the easiest to implement.

From this Table (4) the requirements with IDs 1, 3, 4 are necessary to facilitate the users access to the implementations of the requirements pertaining to the feature calculation capabilities, viz. IDs 2, 5 and are ranked as ‘Must Have’ – without these foundational requirements the user will be unable to make a convenient use of the feature calculation algorithms that this Package envisions to provide.

Furthermore, in Table 4, the reason why the requirement with ID 2 is a ‘Must Have’ and the one with ID 5 is a ‘Should have’ is for the purpose of setting a threshold of what inclusion of feature-calculation capabilities would be considered a minimum viable deliverable that has at least some useful and worthwhile feature calculation capabilities for the clients to benefit from. Meaning if the deliverable only ended up covering the requirement with ID 2 (and all the other ‘Must Have’ requirements) then it would still be deemed a useful product in line with what was envisioned and such a deliverable can be wrapped up to be a Python package that calculates some useful epitope prediction related features and can be voluntarily expanded upon by others to calculate more features. ‘Should have’ requirements do weigh as highly as ​must haves,​ although they do not necessarily require completion within the current timeframe – but their inclusion in the current timeframe would be excellent.

These requirements have been expanded upon to categorise them into functional and non-functional requirements.

Table : Pepfeature Requirements

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ID** | **Title** | **Functional (F) or Non-functional (NF)** | **Description** | **MoSCoW Rank** |
|  | Read the standardised windowed data (input) coming from the Data Integration block. | F | Python package should facilitate functionality that takes in the input of a (pandas) DataFrame. | Must Have |
|  | Calculate the frequency-related features | F | Python package should facilitate functionality that calculates the following features on input data:   * AA proportion * K-mer frequencies * Conjoint Triads | Must Have |
|  | Return a data frame (output) containing the input observations + calculated features with columns in a specific format. | F | Python package should facilitate functionality that outputs the initially input DataFrame with the features calculated appended as columns. | Must Have |
|  | Column names must follow the convention "feat\_XYZ" for calculated features | NF | Python package should facilitate functionality that calculates the following features on input data:  Column names of the features calculated and appended to inputted Data Frame must be named in the format feat\_XYZ. | Must Have |
|  | Calculate additional features (See ‘Description’ column for this row.) | F | Python package should facilitate functionality that calculates the following features on input data:   * AA descriptors * Sequence Entropy * Molecular Weight * Number of Atoms (C, H, O, N, S) * Frequency of AA types | Should Have |
|  | Be able to process up to 10 million observations. | F | Python package should facilitate functionality that allows such memory efficient processing that 10 million observations could be processed in one go. | Could Have |
|  | Parallelise calculations on multiple cores | F | Python package should facilitate functionality that allows multi-processing of the feature calculation functionalities. | Could Have |
|  | data-pre-processing of invalid AA sequences | F | Python package should facilitate functionality that does data pre-processing prior to feature calculation:  If an amino acid sequence contains an invalid amino acid code (B, J, X or Z), the invalid AAs are removed prior to the calculation of any feature. | Could Have |

In Table 4, the requirements ranked with a priority of ‘Could Have’ in the ‘MoSCoW rank’ column were decided to be only implemented if there was enough resources (time) to do so. These requirements are not essential for the project to be deemed a success.

Besides the functional and non-functional deliverable related requirements in Table 4 another optional requirement set by the secondary client was to benchmark the deliverable’s calculation speeds against R package *epitopes* – this was to elucidate if Pepfeature performs better than this current R package solution used by the main client. Albeit the deliverable performing better than *epitopes* is not a factor of success, however, it will be an indicator that to what extent the package is useful to the main client.

## Contingency plan and Risk mitigation

A contingency plan to take account of possible future event or circumstance that could have unfavourable affects and hinderance on the optimum completion of the project was formulated at the commencement of the project. None, of the risks materialised throughout the course of the project. The plan is shown in Table 5.

Table : Contingency plan

|  |  |
| --- | --- |
| **Risk** | **Mitigation** |
| Supervisor has to take time off (e.g. due to becoming ill) | Seek support from the university and I shall keep working on my own discretion aiming to get as much of the goals done as possible. |
| I have to take time off (e.g. due to becoming ill) | The nature of the project allows for many ‘finishing’ points; I shall readjust my goals to aim for a ‘finishing’ point that is within my capacity with the guidance of the advisor. |
| Data sets get lost completely | Use a dataset easily available on the internet instead. |
| Main client (PhD student) becomes unavailable for contacting (e.g. due to becoming ill) | I will contact the advisor who is probably aware of what the client would require at each stage of the project anyways for further guidance. |
| I lose my progress on the project | Stick to a plan that requires frequent backups on to the cloud of the progress made – I will be using git for all my files, so there will be incremental backups. |
| Due to hardware & software issues I become unable to progress further on the project | I am expected to mainly only make use of a laptop to complete the project. In the unlikely scenario that my laptop of use for example breaks then I have a backup laptop ready from which I can continue the project promptly |

## Work schedule

The schedule/time table shown in figure 2 was formulated at the commencement of the final year project. Practically the project itself loosely followed these stages within the stated time frames.

Since, I had minimal Python exposure, around a month was dedicated and set aside at the beginning of the project to learn Python. What was learnt during this time were the basics of Python use & syntax and also the Pandas and Numpy libraries as these were certainly going to be used for the development of the deliverable. Only sufficient enough skills were learned in this field before proceeding to actually dive deep into the development of deliverable.

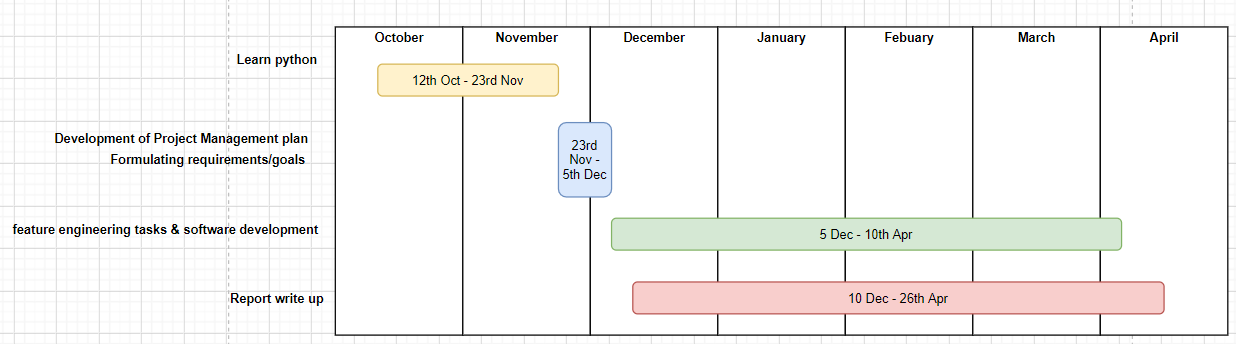


Figure : Work Schedule for the entire final year project

## Software Methodology

Analysis, design and implementation, testing and maintenance are the industry recognised stages that the course of development traversed through leading to the final deliverable.

It was decided that an appropriate project development approach would be a sprint based one, Agile, based on bi-weekly sprints.

The project was research-guided such that sufficient information regarding each epitope feature calculation functionality minus the biological theoretical information associated with them was provided by the supervisor; such that the development of the algorithms to calculate those features could by purely using first principles.

Moreover, the project being research-guided also meant that details of the features to calculate were disclosed to me in increments in the order of priority of development – only gaining the next set of ‘task details’ or feature calculation functionality details if I confirmed the functioning of the current feature algorithm being worked on as working.

This approach had several benefits. Firstly, since the functionalities to implement are complex and I lacked the domain knowledge and experience in the field of feature calculation it ensured that I did not overwhelm myself and took an approach to developing the package in a manner such that in the case if time ran out for the completion of all features I aimed for then I would still have a viable wrapping up point as a Python package deliverable – the only difference being that this would be capable of calculating fewer features.

Moreover, the supervisor was the main developer of the R package *epitopes*, therefore through his personal domain knowledge he laid out a road map for me to complete the course of feature functionality development in the most coherent and efficient manner; for example, the features ‘k-mer frequencies’ and ‘conjoint triad frequencies’ essentially have the same programmatic principles in their algorithms, therefore it would only be efficient to develop these one after the other rather than have a delay between their implementation and focus on the other complex features and consequently having to waste time re-discovering or re-learning the methods of development.

## Documenting Tasks

Trello[[3]](#footnote-3) boards were used in this project to document progress and keep track of tasks mainly related to the deliverable. Cards (representing tasks or progress notes) were grouped together under a timeframe of 2 week; more specifically – in line with the Agile approach – for these 2-week sprints there would be 3 boards titled: ‘To Do’, ‘Doing’ and ‘Done’ representing each state of the task or progress note is currently in. The supervisor would add cards that contain tasks and general notes (e.g. the descriptions of the feature extraction algorithms to implement) in a separate board titled ‘To do (at some point)’ and from here these cards would be dragged and dropped into the latest ‘To Do’ board of the current 2-week sprint. See Figure 2 for a visual of the Trello Board set-up.

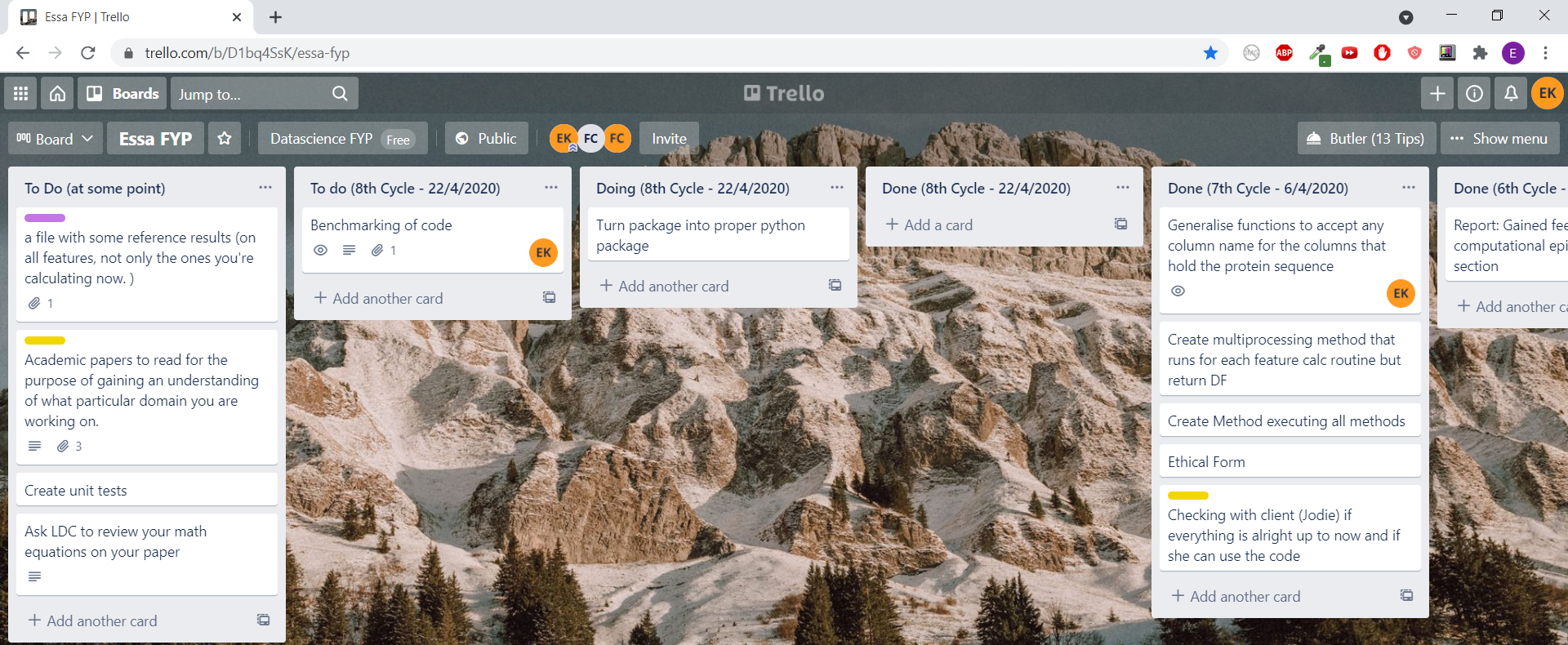


Figure ) A screenshot of the Trello board for this project showing the set-up of the boards and cards use.

The benefits of the Trello board are organisation related and it can help show the progress of the project to stakeholders as it is essentially a project diary. The Trello board was generally updated at least bi-weekly, primarily following the arranged bi-weekly supervisor meetings.

## Code Repository

GitHub[[4]](#footnote-4) was used for version control and as the codebase. It is also from where people can read the documentation of the Package such as installation instructions and example use cases.

# Design and implementation

A Python package named Pepfeature has been developed. In Section 3.2 of this report the set Requirements are given in Table 4 and it was said that the clients gave the freedom to choose the Pythonic tools and approach to take to meet these requirements as long as it conforms to be a self-contained Python package in the end. Therefore, during the course of development, through consultations with the secondary client certain design decision were formulated for the purpose of achieving these major requirements stated in Table 4 – these design decisions were necessary as the requirements given by the clients were not detailed enough and scope was given to allow my own reasonable judgements and interpretations guide me in covering the requirements which manifest through the design decisions expounded in the following section. The rationale behind these decisions are also discussed where appropriate.

The approach taken to show these design decision is to first state the requirement that the particular design decision is trying to cover. A benefit of this is also that a good overview what has been implemented in accordance with the requirements will be documented.

## The epitope sample dataset

The Supervisor provided a sample dataset, namely Ov\_data.csv. The important column in this data set is named "Info\_window\_seq". It contains 15-character amino acid strings representing fixed-length protein fragments (called peptides). This sample data set was used to as a tool for inputting into the feature calculation routines developed for the purpose of testing.

## What has been implemented

### Availability on PyPi

The Python package - Pepfeature - is available for download and use from GitHub as a project and it is also hosted at the Python Packaging Index[[5]](#footnote-5) (PyPI) to download as a consolidated package. This means that the Package is pip installable from the console as:

*pip install Pepfeature*

This was not a requirement from either of the clients – the Package being installable from GitHub was sufficient. This is an example where the original expectation of the work is exceeded.

It being installable through PyPi has the main benefit of easy availability and accessibility when users are wanting to use Pepfeature in their own project. It is very easy to install, delete, check for and install updates using a package installer such as Pip, which also manages the package’s dependencies – ensuring their availability upon installing.

Moreover, it also abstracts away the complex code from the user and leaves them with an API to interface with – making it Pythonic - in line with what Python users are generally used to with scientific packages such as pandas and numpy.

Before proceeding with hosting it on PyPi, the secondary client was consulted and gave the advice that as this is a scientific package, therefore the veracity of the outputs produced by the package have to be flawless. Only when it was ensured that the features produced by Pepfeature were accurate (see Section 4.3 for details regarding the testing of Pepfeature) was the package hosted on PyPi.

Pepfeature’s availability on PyPi contributes to its ‘consolidated self-contained Python package’ status - which was a set requirement by the clients as discussed in Section 3.2.

### Package structure

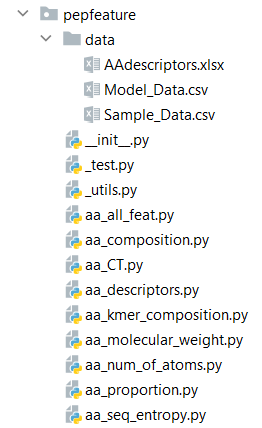


Figure : File Structure of Package Pepfeature

The file structure of the Pepfeature package is shown in Figure 4. Here the .py files with names that start with an underscore character are private modules, viz. ‘\_\_init\_\_.py’ , ‘\_test.py’ and ‘\_utils.py’ in accordance with the *Python Pep 8 style guide*[[6]](#footnote-6) – these modules are intended to be accessed only by the developer of the package and are not suitable for normal users and nor form part of the public API.

The rest of the modules in the package are public and form part of the API. They are feature calculation modules and have a name starting with ‘aa’ – which stands for amino acid – followed by an indication of what epitope feature(s) (out of the eight possible features) its constituent ‘\_algorithm’ function calculates, this is expounded upon in the next **Section 4.1.3**.

See docstrings associated with each such module for details of what feature they aim to calculate or see Table 9 in section 4.1.6 of the report.

Moreover, in Figure 4 the package also contains a sub directory named ‘data’. This consists of the excel file ‘AAdescriptors.xlsx’ that holds data pertaining to the individual weights for each amino acid on each amino acid descriptor – this file is used in the feature calculation algorithm (i.e. the function named \_algorithm) held in the ‘aa\_descriptors.py’ module. The other two .csv files in the ‘data’ folder are used in the module ‘\_test.py’ and their purpose has been explained in section Testing.

### The Functions in Modules

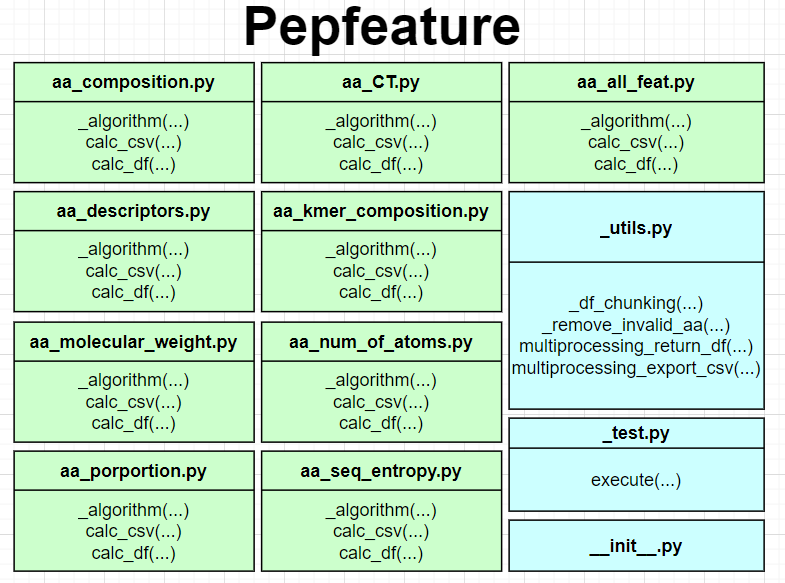


Figure : High level overview of Pepfeature's modules and the functions within them. Green rectangles represent public modules and blue rectangles respresent private modules. Only functions that start with the underscore character (‘\_’) in their names are private.

*See docstrings associated with each function (in Figure 5) to see its specification (such as argument details) and explanation in detail. In the report only high-level details will be discussed.*

**The Public Modules’ Functions**

All public modules (green rectangles in Figure 5) contain a private[[7]](#footnote-7) function named \_algorithm. The \_algorithm function is what contains the code to calculate the feature(s) corresponding with what is indicated as the module name (it is contained within); for example, the module aa\_CT.py**’s** ‘\_algorithm’ function contains code to calculate Conjoint triad frequencies features only. However, the module aa\_all\_feat.py is the odd one out and its ‘\_algorithm’ function calculates all of the eight features on the input in one go.

The various ‘\_algorithm’ functions return a data frame with the features calculated appended as columns. They are wrapped by other functions; these wrapper functions are either for the purpose of formulating the public API or to add multiprocessing functionality and/or to add exportation of the data frame produced as a .csv functionality, this is expounded further in this subsection.

In these public modules (green rectangles in Figure 5) there are two additional functions, viz. ‘calc\_csv’ & ‘calc\_df’ which are wrapper functions intended to be part of the public API for the users, but the \_algorithm function is not and is private – only to be used internally within the module.

**\_util.py**

In \_util.py the function \_remove\_invalid\_aa is a helper functions for the wrapper functions multiprocessing\_return\_df & multiprocessing\_export\_csv. Moreover, \_df\_chunking is a helper function for multiprocessing\_export\_csv only.

Here, the purpose of the wrapper function multiprocessing\_return\_df is to take in as an argument a function (along with other associated arguments) that calculates feature(s) (i.e. the various ‘\_algorithm’ functions across each of the public modules) and execute that function with the option of multiprocessing functionality and returning the Data Frame produced.

The ‘multiprocessing\_export\_csv’ wrapper also takes in as an argument a function (along with other associated arguments) that calculates feature(s) (i.e. the various ‘\_algorithm’ functions in each of the public modules) and execute that function with multiprocessing functionality and instead exports the data frame produced as a CSV and saves it into a stated location (also passed as an argument).

The wrapper functions’ ‘calc\_df’ & ‘calc\_csv’, across the public modules (see Figure 5) wrap these functions, viz. ‘multiprocessing\_return\_df’ & ‘multiprocessing\_export\_csv’ respectively. Figure illustrates this visually ( A sequence diagram ??)

### Maintainability and expansion

The aforementioned helper and wrapper functions contribute towards the maintainability of the code. Wrapper functions serve as an interface to adapt to the existing code, viz. the ‘\_algoirthm’ functions across the Public modules with multiprocessing and .csv exportation functionalities through the aforementioned functions in the \_utils.py module – it saves from the hassle and the mistakes that can slip through by modifying the codes back and forth had these functionalities been implemented without wrapper functions. Moreover, due to the use of these wrapper and helper functions the package – Pepfeature - is given a relatively easily comprehensible structure which contributes to maintainability.

Furthermore, all the modules and their constituent functions are annotated formally using docstrings which increases maintainability.

The ultimate goal of this effort towards maintainability is to make future additions easy. Albeit, this is not a requirement of the clients, nor has extensive effort been put in to facilitate protocols for open-source development of the package and the specific kind of documentation that goes along with it. But the current maintainable and structured package lays down the foundations for this, and users can informally expand the package (through Github) with more feature calculation capabilities beyond the current eight features calculated if they wish to do so. This is an example where the original expectation of the deliverable is exceeded.

### API overview

The aim was to make the API intuitive and Pythonic (similar to the API of the pandas and numpy package) – which successfully has been done so; this can be gauged from Figure 6 which displays the structure of generic line of code when the package in in use; the possible options for the user after importing the Pepfeature package into their code and then making use of it are shown. The API that the user interfaces is intuitive and simple. Further use case examples of the API are given in the documentation.

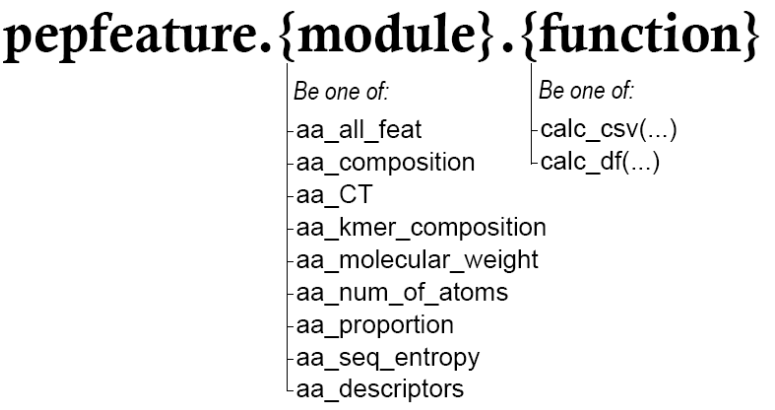


Figure : Pepfeature's Public API overview

Table : Subset of ‘*Table 7: Pepfeature Requirements’* showing requirement with ID 1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ID** | **Title** | **Functional (F) or Non-functional (NF)** | **Description** | **MoSCoW Rank** |
| 1 | Read the standardised windowed data (input) coming from the Data Integration block. | F | Python package should facilitate functionality that takes in the input of a (pandas) Data Frame. | Must Have |

The requirement with *ID 1* has been reproduced in this section (see Table 6). The API that the user interfaces to use Pepfeature covers this requirement. Essentially, both the functions in Figure 6, viz. calc\_csv & calc\_df accept a Data Frame argument, this can be seen in Figure 7 that shows a function definition of calc\_df along with its docstring.

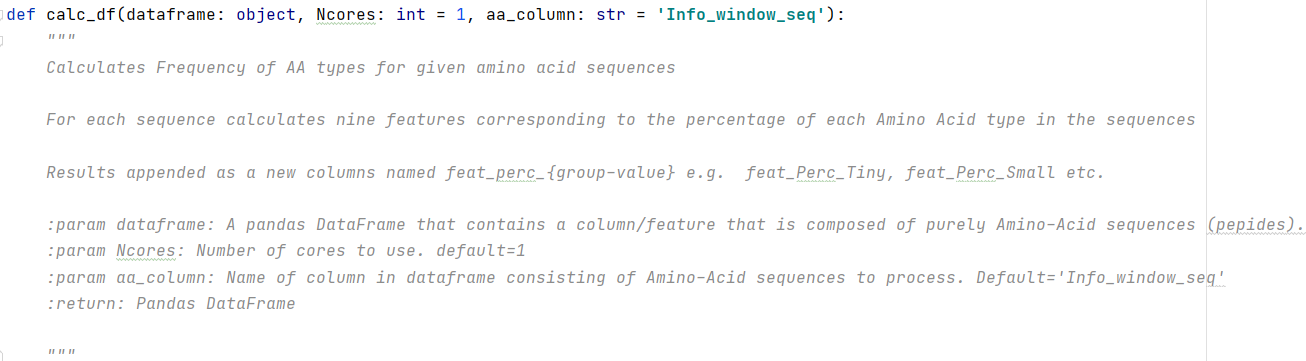


Figure : Part of a the function definition of calc\_df within aa\_composition,py

All calc\_df & calc\_csv functions across all Public modules accept an argument ‘aa\_column’ (see Figure 7 for visual). This argument refers to the name of the column in the input data frame that holds the amino acid sequences to process. Since, this is a Python package built around the main client’s prediction pipeline and their ‘aa\_column’ was expected to be “Info\_window\_seq” therefore this previously had been hardcoded in the routines of the Package. However, after consultations with the secondary client it was decided that for the package to be more generalised for the rest of the scientific community, the ‘aa\_column’ argument was added. Moreover, for the convenience of the main-client this argument was defaulted to “Info\_window\_seq”. This is an example where the original expectation of the deliverable is exceeded, albeit a minor design decision.

### Feature Calculation capabilities

Table : Subset of *‘Table 4: Pepfeature Requirements’* showing requirements with ID 2, 3 and 5

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ID** | **Title** | **Functional (F) or Non-functional (NF)** | **Description** | **MoSCoW Rank** |
| 2 | Calculate the frequency-related features | F | Python package should facilitate functionality that calculates the following features on input data:   * AA proportion * K-mer frequencies * Conjoint Triads | Must Have |
| 5 | Calculate additional features (See ‘Description’ column for this row.) | F | Python package should facilitate functionality that calculates the following features on input data:   * AA descriptors * Sequence Entropy * Molecular Weight * Number of Atoms (C, H, O, N, S) * Frequency of AA types | Should Have |

The requirement with *ID 2, 3 and 5* has been reproduced in this section (see Table 8). Each of the required eight features stated in requirement ID 2 & 5, have been implemented.

As shown in preceding sub-sections, in Pepfeature each epitope feature has its own module (.py file) named after it, and contained within every such module is a function named ‘\_algorithm’ which contains the code to calculate the particular feature(s). Table 9 shows which Module in the package corresponds with what feature calculation capability and which feature description it matches from section 2.3 (part of the background research section) of the report.

Table : Showing which Module in the package corresponds with what feature calculation capability.

|  |  |
| --- | --- |
| **Package’s module name** | **Section of report with details about the feature and its calculation** |
| aa\_proportion.py | 2.3.1 Proportion of Individual Amino Acids in sequence |
| aa\_kmer\_composition.py | 2.3.2 k-mer Composition |
| aa\_CT.py | 2.3.3 Conjoint Triad Frequencies |
| aa\_seq\_entropy.py | 2.3.4 Sequence Entropy |
| aa\_composition.py | 2.3.5 Frequency of AA types |
| aa\_num\_of\_atoms.py | 2.3.6 Number of atoms |
| aa\_molecular\_weight.py | 2.3.7 Molecular Weight |
| aa\_descriptors.py | 2.3.8 AA descriptors |

Furthermore, an additional Based on discussions with the clients it was decided to implement separate functions to calculate different families of features, and then provide an umbrella - an envelope function - that calculates everything together. This module is named aa\_feat\_all.py and its \_algorithm function acts as the envelope function that executes all the functions to calculate the eight features in one go and, in the end, concatenates each of their resultant data frames as an output.

### The API Function ‘calc\_csv’ and ‘calc\_df’

Table : A Subset of ‘Table 4: Pepfeature Requirements’ showing requirements with ID 3, 4, 6, 7 and 8

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ID** | **Title** | **Functional (F) or Non-functional (NF)** | **Description** | **MoSCoW Rank** |
| 3 | Return a data frame (output) containing the input observations + calculated features with columns in a specific format. | F | Python package should facilitate functionality that outputs the initially input DataFrame with the features calculated appended as columns. | Must Have |
| 4 | Column names must follow the convention "feat\_XYZ" for calculated features | NF | Python package should facilitate functionality that calculates the following features on input data:  Column names of the features calculated and appended to inputted Data Frame must be named in the format feat\_XYZ. | Must Have |
| 6 | Be able to process up to 10 million observations. | F | Python package should facilitate functionality that allows such memory efficient processing that 10 million observations could be processed in one go. | Could Have |
| 7 | Parallelise calculations on multiple cores | F | Python package should facilitate functionality that allows multi-processing of the feature calculation functionalities. | Could Have |
| 8 | data-pre-processing of invalid AA sequences | F | Python package should facilitate functionality that does data pre-processing prior to feature calculation:  If an amino acid sequence contains an invalid amino acid code (B, J, X or Z), the invalid AAs are removed prior to the calculation of any feature. | Could Have |

As expounded in preceding sections – in particular section 4.1.5 – the functions that the user calls to interface with the package are either calc\_csv or calc\_df. Calc\_df simply returns a Pandas data frame containing the calculated features appended as new columns in the specified format of requirement ID 4 (Table 10). Thus, the requirement with ID 3 shown in table 10 has also been covered.

Under the hood both calc\_csv and calc\_df, in all public modules, come with multiprocessing functionality and they accept a parameter ‘Ncores’ to set the cpu cores to use for this. Thus, the requirement with ID 7 shown in table 10 has also been covered.

In order to cover the requirement with ID 6 (see Table 10) in the deliverable, it was decided after consultations with the secondary client to develop functionality where the input data frame is processed in chunks and then exported as a CSV.

Consequently, the functions ‘calc\_csv’ were created to do this. Calc\_csv is is intended to be a memory (RAM) efficient way of calculating the Features; they are calculated on a single chunk of the input data frame (of ‘chunksize’ number of rows – passed as a parameter) at a time and a chunk is deleted from memory – freeing up space - once it has been processed and exported as a CSV. Moreover, if this functionality is paired with the Pandas’ ‘read\_csv’ function’s ‘chunksize’ attribute – which allows to specify how many rows of data to read into local memoy at a time – and these chunks of the larger data frame are input into calc\_csv (which also in turn has an appropriate ‘chunksize’ as its parameter set) then further significant memory efficient performance is expected, however this remains to be tested and benchmarked to conclusively say so.

It has to be conceded that due to a lack of time there is no attempt made to process a data set of 10 million observations and this should be done at some point. Albeit theoretically this is expected to be possible and therefore the requirement with ID 6 is deemed to be covered.

## Testing

The Pepfeature package contains a sub directory named ‘data’ (see Figure 4, section 4.2.2). This contains two .csv files, viz. Model\_Data.csv & Sample\_Data.csv. Model\_Data.csv is a data set that consists of a column containing amino acid sequences, plus all the features calculated on them by the R package *epitopes*, appended as columns; this Model data set – containing accurate feature calculations – was provided by the developers of the *epitopes* R package for the purpose of myself comparing and ascertaining the veracity of the results produced by the Pepfeature package.

Sample\_Data.csv consists of a single column containing the very same amino acid sequences as Model\_Data.csv. This data set is inputted into the feature calculation routines of Pepfeature as a data frame to be processed, and in return the same data frame with the features calculated appended as columns would be outputted.

For the bulk of the duration of development of Pepfeature, whenever a newly developed feature calculation capability would need testing (a variation of) Sample\_Data.csv would be input into the routine and following this, the results – i.e. the features produced – would be manually checked against the Model\_Data.csv for correctness. Since, this was a manual check only a few of the features calculated would be matched. If no flaws were noticed, then that specific feature calculation capability of Pepfeature would be assumed to produce the correct features and thus be working as intended.

However, as the complexity of the package grew with the creation of the wrapper functions that add additional functionality such as multiprocessing to the raw \_algorithm functions, and the creation of the aa\_feat\_all.py module that is intended to calculate all features in one go, and the emphasised expectations of the client that the results produced must be accurate, it was decided that in an automated manner all of the 8 features that Pepfeature calculates on a given amino acid string to be compared with the Model\_Data.csv for accurateness. The module \_test.py was created for this. This module is intended to be user by developers whenever they want to ascertain that the main functionality of the Package works and nothing has broken.

The module \_test.py consists of one function, viz. ‘execute(Ncores, save\_folder)’ to execute the test; where the parameter Ncores is the cores to use for the computation (multiprocessing) and ‘save\_folder’ the path to export the resultant CSV to. This tests two things:

Firstly, for the purpose of generating the features on Sample\_Data.csv under the hood it executes each of the calc\_df functions and a level deeper, all the \_algorithm functions in Pepfeature (through the usage of *aa\_all\_feat*.calc\_df(…) – thus, if any of these functions were programmatically broken an error would be raised. Moreover, multiprocessing is used for this, so if this execution is successful that means that the multiprocessing functionality is sound.

Next, the first intended test (Test 1) occurs where the resultant pandas Data Frame with the features calculated are then compared with the values of the Model Data frame (Model\_Data.csv) for accuracy – the results of the test are printed on console. Figure 8 shows s result of this test in a case where certain features did not match – in a non-match cases the test prints out the column names where there is a mismatch of values; I can use this to trace where the errors are coming from (i.e., which of the \_algorithm function across the public modules in the package).

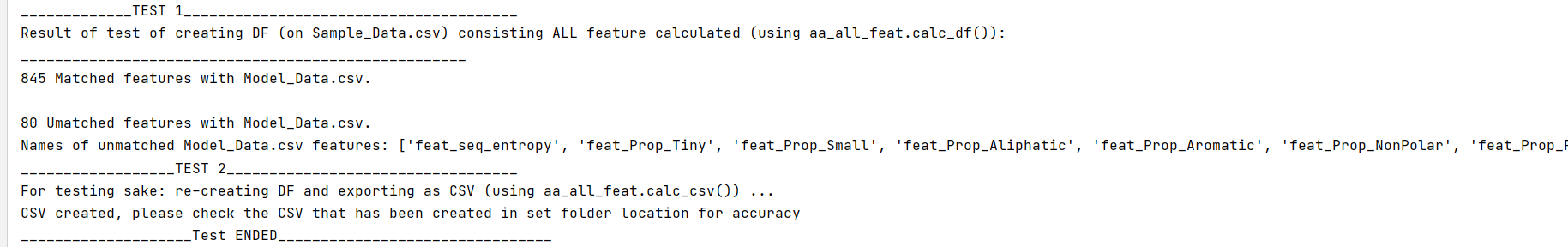
Finally Test 2 occurs, the aa\_all\_feat.calc\_csv() function with the Sample\_Data.csv derived Data Frame as a parameter is executed. Under the hood, this requires traversing through each of the calc\_df() in the other modules – executing each of them, but if test ‘1’ mentioned in this list was successful then these would be confirmed as working already. Therefore, this test is to solely check that the calc\_csv() in aa\_all\_feat.py works as intended – a CSV is created in the correct ‘save\_folder’ location and the results in the CSV are sound matching with Model\_data.csv (this would be matched manually, as instructed in the output of this test which has been reproduced in Figure 8)

Figure : Output of \_test.py’s execute(…) function in the case where there is a mismatch of features between those generated by Pepfeature and those in Model\_Data.csv

From the aforementioned list, it is clear that these tests do not test the working of the API function calc\_csv() belonging to the other modules, besides the one in aa\_all\_feat.py. Moreover, this automated test is not exhaustively testing every parameter. For example, the ‘chunksize’ parameter of aa\_all\_feat.calc\_csv() is not tested for any other realistic test case. Anything that \_test.py’s execute(…) function does not test has been loosely tested manually for common use cases – in a non automated manner – throughout development stages and it can be said with reasonable confidence that the package works for at least common use cases.

However, in the future a testing strategy should be formulated with the aim of exhaustively testing every interface API function for correctness along with every parameter (such as by using black-box testing techniques). Additionally, \_test.py should be converted into a formal unit test – it is essentially an informal unit test at this point and more unit tests should be created testing other aspects of the system. This should be done in some point in the future. Moreover, the package is on PyPi and GitHub, people have been requested in the documentation to report any bugs or glitches they find.

firstly, it generates peptide features on Sample\_Data.csv and compares in an automated manner each value of the output Data Frame with a Model Dataframe (Model\_Data.csv) for accuracy. For the developer if o

The To check the veracity of the results produced from the feature calculation algorithms of this package.

The test consists of code that first generates peptide features on Sample\_Data.csv and compares each value of the output DataFrame with a Model Dataframe (Model\_Data.csv) for accuracy.

This data set During the development of the package was manually tested as development progressed.

‘\_test.py’ and their purpose has been explained in section Testing.

with all the features that consists of

For example, as a feature calculation capability, viz. a ‘\_algorithm’ function was implemented its resultant Data Frame would be manually checked against a Model – gold standard – data set that has all the ; this data set was provided by

|  |  |  |  |
| --- | --- | --- | --- |
| **ID** | **Test** | **Expected outcome** | **Outcome** |
|  | Install Pepfeature from PyPi without any hard dependencies installed. | PyPi installs Pepfeature + its dependencies | All required dependencies numpy, setuptools, pandas, openpyxl and the package itself is installed sucessfully |
|  |  |  |  |

## Benchmark: Pepfeature vs. Epitopes (R package)

The strategy for benchmarking was to use five data sets consisting of a column composed of amino acid sequences containing 10, 50, 100, 500 and 1000 observations respectively. These were fed into the respective umbrella functions from both Pepfeature and the R package to calculate all the eight features.

For both the R package and Pepfeature, each dataset was passed into their respective function tree times and timed for the execution to complete. Both of the benchmarking scripts[[8]](#footnote-8) for each of the packages along with the datasets can be found in the ‘benchmark’ directory on Pepfeature’s GitHub repository[[9]](#footnote-9).

Both benchmarking scripts were executed under the same conditions such as there being no other CPU intensive application running concurrently whilst benchmarking. The system used was ‘iMac 3.6 GHz Quad-Core Intel Core i7, 16Gb RAM running Mac OS 11.2.3’ and the Python script was running under Python 3.7.4 & the R script under version 4.0.5. The secondary client was requested to run both scripts on their system; moreover, since they are using a Mac their execution of the Python benchmarking script confirms Pepfeature being able to run on a Mac as well (whereas prior to this instance Pepfeature only had been tested on Windows).

The results are reproduced in Table 10 and 11.

Table : Benchmark results of Python package Pepfeature

|  |  |  |  |
| --- | --- | --- | --- |
| **Amino Acid observations in Data Frame (size)** | **Minimum execution time out of 3 runs (seconds)** | **Mean execution time of 3 runs (seconds)** | **Maximum execution time out of 3 runs (seconds)** |
| 10 | 1.751 | 2.243 | 3.200 |
| 50 | 3.561 | 3.578 | 3.595 |
| 100 | 5.428 | 5.542 | 5.612 |
| 500 | 20.897 | 21.341 | 21.888 |
| 1000 | 39.967 | 40.582 | 41.515 |

Table : Benchmark results of R package Epitopes

|  |  |  |  |
| --- | --- | --- | --- |
| **Amino Acid observations in Data Frame (size)** | **Minimum execution time out of 3 runs (seconds)** | **Mean execution time of 3 runs (seconds)** | **Maximum execution time out of 3 runs (seconds)** |
| 10 | 0.793 | 0.797 | 0.805 |
| 50 | 3.270 | 3.320 | 3.340 |
| 100 | 6.380 | 6.600 | 6.760 |
| 500 | 31.700 | 34.200 | 36.100 |
| 1000 | 69.100 | 76.600 | 88.600 |

A graph plotting size vs. minimum time is plotted for comparison in Figure. Minimum time has been deemed more appropriate to plot on the assumption that the Minimum execution time gives a lower bound for how fast the machine can run the functions and that higher values in the results are most probably not caused by variability in the programming language’s speed, but by other active processes on the machine interfering with the timing (docs.python.org, n.d.).

Figure : Graph of Size of Data Set vs. Minimum Time

From the Table 10, 11 and the trend illustrated in Figure 9, it can be concluded that Pepfeature is roughly 50% more time efficient when calculating all of the eight features on the largest Data Set (for 1000 observations) and it is reasonable to expect this trend to continue or for the gap of efficiency to grow.

In Figure 9 there is a smaller time efficiency difference for the smaller datasets (10, 50, 100 observations), this is very likely due to the inefficient runtime overhead caused by the usage of the dependency – a package – ‘pkg\_resources’ in Pepfeature’s module aa\_descriptors.py; this dependency unnecessary builds a working set of *all* installed packages (taking up extra time during runtime), even though only the packages that are in use are needed to be loaded up (setuptools.readthedocs.io, n.d.). The overhead this creates is negligible in practical terms, but it has been noted down to be resolved in a future update to Pepfeature.

The intended users of this package are likely only to use this package for feature calculation of very large datasets consisting of thousands of observations and the time efficient improvements over *epitopes* that this package has displayed has been greatly appreciated by the clients. This is an instance where the original expectation of the deliverable is exceeded.

Other useful benchmarks to conduct would be to determine which of the two packages are more memory efficient. Moreover, to conclusively proof the memory efficient benefits that Pepfeature’s calc\_csv() functions is intended to have (as explained in section 4.2.7 of report) a benchmark would need to be conducted. Due to time constraints these could not be performed.

## Main technologies used

The deliverable is implemented in the Python programming language – the main reason as to why Python was used is because it was a need of the clients, this has been expounded in section 1.4 of the report. Besides this, Python has other inherent benefits which makes it suitable for this project. It has many libraries available for data science and data mining purposes that have been used for this project, viz. Numpy and Pandas. Moreover, vanilla Python also has embedded within the standard library that provides a wide range of modules that are standardized solutions for many problems that occur in everyday programming pre-installed; therefore the only external libraries the user has to install is Numpy and Pandas, the rest of the libraries are already pre-installed, making the install of Pepfeature slightly faster.

In Pepfeature Pandas & Numpy are the core packages powering the actual data manipulation & processing of the input date sets for the purpose of feature extraction. Specifically, Pandas facilitated in-memory manipulation of the large datasets through a simple interface. It utilizes NumPy arrays under the hood for eﬃcient, and fast data manipulation as NumPy has bindings to C. Moreover, the fast column and row selection and the visualization of results with output support that Pandas provides made it very intuitive to code with and provided a visual way to see the results of the feature extraction algorithms, speeding up debugging and development overall.

There are parts in the implementation of Pepfeature where the NumpPy package for its efficient arrays have been used standalone due to their great efficiency in terms of performance and mainly to make use of the optimized built in functions that they provide.

The Python standard libraries that have been used in the project are: itertools, os, re, multiprocessing, functools, datetime and timeit.

However, the drawbacks of Python such as its lenient type system, its slow nature of dynamic dispatch, and the global interpreter lock (GIL) which prevents the use of multithreading, all make python itself unsuitable for CPU intensive computations but non of the algorithms developed were CPU intensive. However, it is an ideal staging language, allowing for easy set-up of objects, which can then be sent to more performant compute routines. Figure 2.5 illustrates a small selection of the selection of Python libraries and tools available to the scientiﬁc community. [Extracted from top paper?]

Lastly, the IDE for development of the deliverable was Pycharm community edition.

# Maintenance

There has been many instances of modification of the Package and its documentation after deliver to correct faults. One significant example of this is that soon after Pepfeature was deemed to be complete and was hosted on PyPi the main client was asked to test out the Package. However, immediately after delivery of the deliverable the client corresponded about the Package not working on her end and certain package dependency issues occurring. The packages that Pepfeature depends on and taking into consideration their compatibility with other development environments such as Anaconda was something that I was oblivious of uptill then.

To fix this issue

modification of a software product after delivery to correct faults,

modification of a software product after delivery to correct faults,

-What \_test.py discorvered

-Manuall testing table:

-Maintainance after the package was launched to pypi,

-For example the data file is still showing

- For example dependancie issues were discovered of and fixed

-Made package conform more to the reality of private functions more i.e. wherever private functions were not only used internally within a module, then this would be fixed. For example, changed aa\_feat\_all.\_algorithm from calling the \_algorithm of the eight other aa modules and now it calls the public calc\_csv

# Conclusion

-Add list of things that you did above and beyond of what was required

--Package is on PyPi therefore pip installable.

# OLD

Thus, what the R package outputs it the standard of correctness of my code.

This makes the nature of the project deliverable – the python package – such that it essentially aims to be an enhancement of the R package, which it already naturally will be in many aspects as discussed in the Justification section. However, the

of In an Email interaction with the client

learning pipeline, version control, package control, distribution and testing are simpler

[Project Management]

Requirement analysis

The following table shows the definition of requirements, using a MoSCoW hierarchy & each requirement is supplemented with an explanation:

and then upload it to [Python Package Index (PyPi)](https://pypi.org/) so that anyone can download and use your code as a software library using pip install .

[Risk assessment]

Implementation & testing

Thanks to the technological advances in genomics, proteomics, and epitope mapping techniques, huge amounts of data are being generated and are necessary to organize in a searchable form. B-cell epitope databases provide a training set for evaluation of existing epitope prediction methods and constitute platform for development of novel and better algorithms for prediction.

Linear B-cell epitopes consist of contiguous amino acid residues within a continuous stretch of a primary protein sequence – also known as peptides

There is great interest in identifying epitopes in antigens for several practical reasons such as to produce epitope-based vaccines, therapeutic antibodies and diagnostic instruments [1]. Given that the experimental ~~investigation~~ identification of epitopes demands a large amount of resources, time and effort, therefore the development of computational methods, such as **machine-learning based algorithms are used as an alternative. Due to the availability of validated epitope databases, machine-learning based algorithms could be adopted on all curated data implemented.**

Algorithms for all curated data to design an improved predictive tool for biomedical researchers

[2: Epitopes or antigenic determinants are defined as clusters of amino acid segments located on the surface of an antigen that bind to antigen-specific membrane receptors on lymphocytes or to secreted antibodies, and which elicit either cellular or humoral immune response and are recognized by specific antibodies.]

In order to establish epitope-based vaccines, therapeutic antibodies and diagnostic

instruments, the detection of B-cell epitopes is a fundamental step.

Identification of B-cell epitopes is a crucial phase in

the production of epitope-based vaccines, therapeutic antibodies and diagnostic methods.

The identification of

Antigen: a toxin or other foreign substance which induces an immune response in the body, especially the production of antibodies.

Epitope: The part of an antigen molecule to which an antibody attaches itself [Definitions from Oxford Languages]

Specification

Design

This project will be coded in Python. One reason for this is that I am new to the domain of feature engineering & data science, and Python has proven itself to be an excellent beginner tool for data scientists. This is because it is an easy to learn language from a syntax point of view and it has a vast selection of libraries and resources geared towards data science; two of the main libraries that are available and will have the most utility in my case will be Numpy and Pandas due to the features that they bring for data-manipulation.

To understand the main reason as to why Python will be used then the ‘place’ in which this project fits in the Linear B-Cell Epitope prediction pipeline has to be elaborated: The output of this project is meant to supplement the ‘feature calculation’ block of the Linear B-Cell Epitope prediction pipeline (see the diagram below). Likewise, there are people working down & up-stream on different blocks of the pipeline in the form of projects; the goal is to make the outputs of each of the projects fit together relatively seamlessly. For example, another student, working directly up-stream of the ‘feature calculation’ block, is doing a project that consists of scraping & consolidating data from multiple different verified databases, and this will be delivered as an input to my produced code in the form of a pre-processed dataset in a particular format. Likewise, my project’s output will facilitate feature calculation on this dataset & (some level of) filtering and the output of my project is what my client (Jodie) can utilize for her modelling strategies – she is working directly downstream of me. Currently, this ‘feature-calculation’ block has already been done in the language ‘R’ by the project supervisor (Felipe), whereas the rest of the pipeline is done in python, thus the need was felt to also do the ‘feature-calculation’ block in python so that the outputs of each block fit together relatively seamlessly and a single program can be used.

This will be the first existing Implementation of these types of protein based features calculation in the python language that we know of - so such a package will be something that does not exist. It will be the first consolidated python package to calculate features. And the utility of this in itself is already good enough. If the end product (i.e. the package) People are going to be using this not only us but others too. if the code is available and the code is robust enough to take in an input in a specific data format that you are going to document and it is going to output specific features in a specific format that you will also document as part of your final year report - this Module/package in it self is already a very great product.

In terms of background research have discussion on the tools that you are going to be using: Python and the specific python packages that im using i.e. Pandas, numpy etc. - this is the Language and packages that I am using and WHY I am using them for calculating those features and the discussion has to be in the context of the stage of the pipeline I am working at [Teacher: "it is a python Pipeline, so it makes sense to do it in Python"

Another possible choice would have been Python, which is overall a better programming language than both R and Matlab. Its object oriented and functional nature, together with libraries such as Numpy, Scipy, statsmodels, and matlibplot make it a powerful statistical tool. However, it lacks a strong community of mathematicians, so many of the functionalities already existing in Matlab and R are not yet available.

The production of epitope-based vaccines, therapeutic antibodies, and diagnostic tools are some of the practical benefits of identifying linear B-cell epitopes for laboratory investigation, and the motivation to utilize computational techniques to do so (Potocnakova, Bhide and Pulzova, 2016), complemented with the reason that it provides improvements in the efficiency of this particular process compared to experimental identification, which demands a large amount of resources, time and effort (EL-Manzalawy and Honavar, 2010).

Utilities for generating features for machine learning pipelines

Amino acids that have been formed into **peptides** are termed **residues**.

As of 13/01/21 there are no known publicly available python packages that calculate the protein-based features that this projects’ deliverable will facilitate. This has been verified by searching the Python Package Index website – a public package repository - (www.PyPI.org) using its search facility; relevant combinations of key words to formulate phrases such as, ‘feature calculation(/extraction) for epitope data(/prediction)’ along with various relevant filters were applied such as ‘Bio-informatics’ to make a thorough search of the website for the existence of a similar package but none were found.

A feature is the speciﬁcation of an attribute and its value. Feature extraction/calculation creates newly derived features from existing attributes. Features are synonymous to columns in tabular formatted data.

Introduction

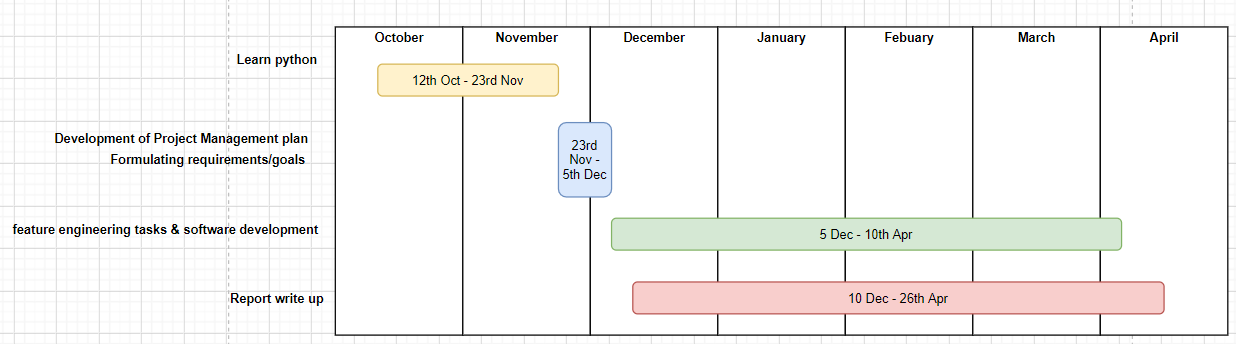
Background research

* Requirement elucidation
* Existing solution
* Technologies to be used
* Features to calculate

Project management

Design and implementation

Skeleton Outline of what will be done



What is a MoSCoW hierarchy?

MoSCoW Analysis is a prioritization technique that is common for projects such as this, that follow an agile approach with a fixed deadline. The aim is to have emphasis on prioritising the implementation of critical requirements (IIBA, 2009). The requirements are categorised in order of priority as follows:

1) Must Have (M)

For the project to be deemed a success these critical requirements must be met in the available timeframe.

2) Should Have (S)

These ​requirements can weigh as highly as ​must haves,​ although they do not necessarily require completion within the current timeframe.

3) Could Have (C)

Providing there are enough resources to do so, these requirements are usually capable of being included in the current timeframe. Could haves are not essential for the project to be deemed a success.

4) Won’t Have (W)

These requirements are deemed the least critical and are agreed by stakeholders as inappropriate for the current timeframe, however, if all other requirements have been met and there is still time & resources left, then these requirements may be persued.

What are functional & non-functional requirements?

Functional Requirements detail actions that the system can perform, along with actions the user can do on the system, for example… “A system must send an email whenever a certain condition is met.”

The non-functional requirement elaborates a performance characteristic of the system, for example… “Emails should be sent with a latency of no greater than 12 hours from such an activity.”

# Appendices

**Appendix 1: Initial agreed upon requirements.**

1. (M) Read the standardised windowed data (input) coming from the Data Integration block.
2. (M) Calculate the frequency-related features (AA frequencies, K-mer frequencies, Conjoint Triads)
3. (M) Return a data frame (output) containing the input observations + calculated features with columns.  
   Column names must follow the convention "Info\_XYZ" for general information/metadata, "feat\_XYZ" for calculated features, "Class" for the class attribute.
4. (S) Calculate additional features: AA descriptors, Sequence Entropy
5. (S) Calculate additional features: Molecular Weight, Number of Atoms (C, H, O, N, S)
6. (S) Be able to process up to 10 million observations.
7. (C) Parallelise calculations on multiple cores
8. (C) Incorporate feature selection based on Information Gain / Mutual Information.
9. (W) Benchmark calculation speeds against R package *epitopes*

1. Available from URL: https://webs.iiitd.edu.in/raghava/pfeature/ [↑](#footnote-ref-1)
2. MoSCoW Analysis is a prioritization technique that is common for projects such as this, that follow an agile approach with a fixed deadline. The aim is to have emphasis on prioritising the implementation of critical requirements (IIBA, 2009). [↑](#footnote-ref-2)
3. The Trello board of this project can be accessed at: https://trello.com/b/D1bq4SsK/essa-fyp [↑](#footnote-ref-3)
4. The GitHub repository can be accessed at: https://github.com/essakh/pepfeature [↑](#footnote-ref-4)
5. Pepfeature’s PyPi page: https://pypi.org/project/Pepfeature/ [↑](#footnote-ref-5)
6. There is no official Pythonic way to denote private modules and enforce such restrictions. However, there are conventions to denote private Modules such as naming them with an ‘\_’ at the start. See: https://www.python.org/dev/peps/pep-0008/ [↑](#footnote-ref-6)
7. There is not Pythonic way to denote private functions and enforce such restrictions. However, there are conventions to denote private functions such as naming them with an ‘\_’ at the start. See: https://www.python.org/dev/peps/pep-0008/ [↑](#footnote-ref-7)
8. The R package’s benchmarking script was provided courtesy of Dr. Campelo. [↑](#footnote-ref-8)
9. https://github.com/essakh/pepfeature/tree/master/benchmark [↑](#footnote-ref-9)