UNIVERSITY
OF OSLO

Master thesis

Adding 3D structure support to immuneML

William Shum Ye

Informatics: Programming and System Architecture

60 Credits

Department of Informatics

Faculty of Mathematics and Natural Sciences

Adding 3D structure support to immuneML

William Shum Ye

Supervisors:
Lonneke Scheffer
Milena Pavlovic
Geir Kjetil Sandve

UNIVERSITY OF OSLO
May 2023

Abstract

Adaptive immune receptor repertoire (AIRR) data is used for research into the immune system. immuneML is a machine learning platform used for analysis of adaptive immune receptors. Previously immuneML only supported data in the AIRR[1] format. The AIRR format presents the immune receptors as a flat sequence. In reality the immune receptor binding takes place in the 3D space. Running analyses on 3D structure files might prove beneficial.

This project expands immuneML's capabilities by adding PDB[2] file support. PDB files can now be used to create datasets in immuneML. This enables users to utilize the information that 3D structures provide, such as the distance and the position of amino acids.

Numerous classes have been made for handling the user provided PDB files and creating a dataset object that is suited for storing the information inside.
There has also been implemented a few functionalities to show the potential of work that can be done with the data from the PDB files.

This project lays the foundation for future analysis of 3D AIRR-data with immuneML when more suitable data is available, which may assist future implementations of 3D AIRR machine learning methods.

Acknowledgements

I would like to begin by expressing my sincere gratitude to my supervisors, especially Lonneke, for providing me with invaluable guidance and support throughout this project. Especially for her comprehensive feedback on my writing, as it has significantly contributed to the quality of this thesis.

I would also like to express my deep appreciation to my family and friends for their support throughout this time. Their encouragement and love are greatly appreciated.

Lastly, I would like to end by thanking the University of Oslo for giving me the opportunity, and for providing me with the tools and knowledge to complete this project.

Contents

1 Introduction 2

2 Background 4

2.1 Adaptive immune system 4

2.1.1 Antibodies 6

2.1.2 AIRR data 6

2.2 immuneML 8

3 Goals 13

3.1 Adding 3D structure support to immuneML 13

4 Methods 14

4.1 immuneML – YAML & workflow 14

4.1.1 YAML specifications 14

4.1.2 immuneML order of execution 17

4.2 Implementation of 3D structure support 20

4.3 Generating distance heatmap 30

4.3.1 DistanceToAntigenMatrixEncoder 30

4.3.2 DistanceToAntigenHeatmapReport 33

4.4 Training a machine learning model with a PDBDataset 33

4.4.1 DistanceBetweenStructuresEncoder 34

4.5 Data selection and handling 36

4.5.1 Creating the metadata file 42

5 Results 44

5.1 Classes created 44

5.2 Starting a run with the new classes 48

5.2.1 Generating heatmaps with distances to the antigen 48

5.2.2 Starting a precomputed KNN run on a PDB dataset 53

6 Discussion 57

6.1 Results from this project 57

6.2 PDB files and numbering scheme 58

6.3 Other related 3D structure methods 62

7 Conclusion 64
7.1 Further work 64
7.1.1 Improvements to the PDBDataset 64
7.1.2 DistanceToAntigen 65
7.1.3 DistanceBetweenStructures 65
Literature references 66
Appendix A – How to run immuneML with PDB functionalities 72
Appendix B – Code 77

1 Introduction

Machine learning is a widely used tool that is being utilized in many different areas of research, including the biological sciences. Machine learning brings the advantage of rapidly analysing large amounts of data and make corresponding predictions. Machine learning algorithms are very valuable in biology where there are large amounts of complex data. One particularly interesting area in the biology community is the deciphering of our adaptive immune system[3], which is the part of the immune system that detects and fights viruses, bacteria, and cancer cells. Our immune system detects these threats through immune receptors. These immune receptors are proteins consisting of sequences of amino acids. Machine learning techniques can process the sequence or structure of the protein and find patterns which are associated with binding to a particular target antigen [4]. Categorizing these patterns could be important for diagnosing diseases, developing vaccines and other treatments [5]. Not all machine learning algorithms are equally suited for any given prediction problem [6]. How well they perform depends on the data and the purpose of the task. To find which machine learning algorithm performs best in a specific situation, you would need to try to run them all and compare the results.

immuneML[7] is a platform that brings multiple machine learning algorithms for adaptive immune receptor data together, under a united framework. immuneML is a comprehensive machine learning platform that can be used to learn whether an immune receptor binds to a specific antigen or whether a repertoire of immune receptors from an individual is associated with a sick or healthy person.

Currently, immuneML only supports immune receptors as text sequences. The text sequences are parts of the amino acid sequences of immune receptors. Immune receptor bindings are made of proteins interacting with an antigen in the 3D space. Therefore, there might be advantageous by running machine learning algorithms directly on a 3D structure, rather than a text sequence. 3D structures provide information that flat text sequences do not, such as the position and the distance. This could prove advantageous for a deeper understanding of our immune receptors.

This master thesis is centred around developing and implementing support for 3D structure data to the immuneML platform, as well as providing a few basic tools for analysing 3D structures.

2 Background

2.1 Adaptive immune system

Our immune system is a vast network of connected biological processes [8]. Its purpose is to protect our body from foreign pathogens and malicious cells, by detecting and removing them [8]. The immune system consists of two main parts, the innate immune system and the adaptive immune system [9]. The innate immune system provides fast and general countermeasures against threats, and is the body's first line of defence [9]. The adaptive immune system on the other hand, provides targeted countermeasures based on its memory [9]. The adaptive immune system is a subsystem of our immune system and focuses on detection and removal of molecules that the system recognizes as a threat. The adaptive immune system consists of T- and B-cells [9].

T- and B-cells are two types of white blood cells [8]. White blood cells are used to defend the body against infections and foreign substances [8].
There are several types of T-cells, and they all work together to maintain these main objectives:

- The cytotoxic T-cells detect and eliminate infected and cancerous cells [9].
- The helper T-cells are used to activate other nearby immune system cells to start appropriate countermeasures [9].
- After neutralizing the identified threat, some T-cells become memory T-cells [9]. These memory T-cells will remember the neutralized threat and if it encounters it again, it will handle it much faster than the initial time [9].

B-cells detect and eliminate free floating pathogens and other free floating foreign objects in the blood [9]. B-cells also produce antibodies which neutralizes the identified threat [9].
Both the T- and B-cells recognize an epitope in the antigen [9]. The antigen is recognized by the immune receptors in the T- and B-cells [8–10]. Immune receptors are displayed on the cell surface, and can recognise and bind to a specific antigen [9]. This works like a lock and key, only one key will fit the specific lock [9]. The binding causes the immune system to become activated and countermeasures are started [9]. The T-cell receptors are called TCR, and the B-cell receptors are called BCR. Antibodies are BCRs excreted from the B-cell [11].
Both the TCR and the BCR consist of two chains. The chains are made of sequences of amino acids [8].
The TCR consists of an alpha and beta chain, and the BCR consists of a light and heavy chain [8].

The TCR and BCR share a lot of similarities. Both have two chains and the same type of regions [12]. More specifically, both have a constant and a variable region. The variable region contains the complementary determining regions (CDR) [12], which are the parts of the immune receptor in direct contact with the antigen[13].
There are in total six CDRs in an immune receptor [14], three in each chain [15]. Figure 1[16] shows the structure of a BCR. The tip of the variable region contains the CDR regions.

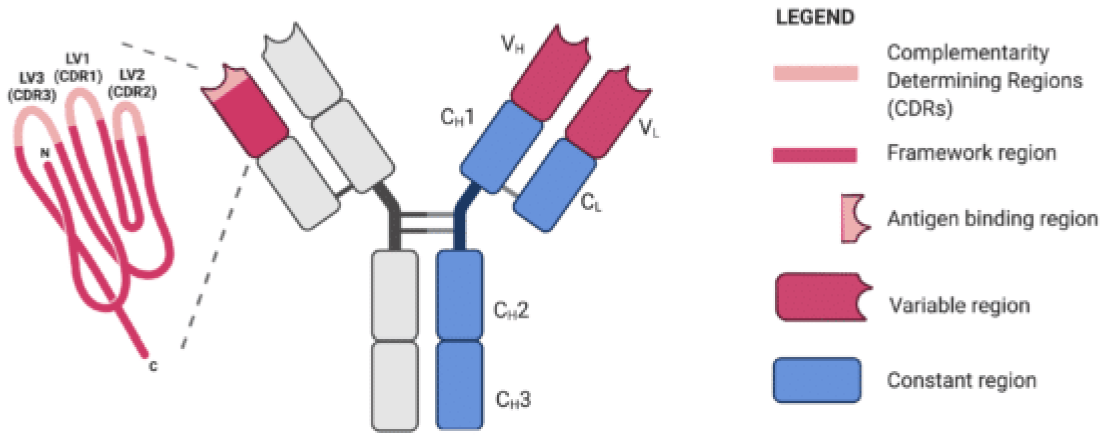


Figure 1: The two different regions are distinguished by colour and letters. C corresponds to constant region, while V corresponds to variable region. The subscript H and L corresponds to the heavy chain and the light chain, respectively. The regions are the same on both sides of the immune receptor, although mirrored. The binding between the antigen and the immune receptor takes place at the tip of variable regions. The framework region is the part between the CDR loops. (Figure by Wang[16])

2.1.1 Antibodies

When a B cell recognises an antigen, it becomes activated. An activated B cell starts to excrete antibodies. An antibody is a B-cell receptor floating[17] around in our bloodstream, which will help neutralising the antigen by binding itself to it. Each B-cell will only produce a single type of antibody[17] in its life, although there might be slight variations due to mutations. Since antibodies bind to a specific antigen it can be used in therapeutic drugs to treat specific conditions. This is the field of monoclonal antibodies(mAbs), that are creating antibodies in laboratories[18] that will be used for treatments and therapy.

Monoclonal antibody therapeutics is based on finding the specific antibody that binds to the antigen of the desired disease, and then mass producing them [19]. The victims of the specific disease might not naturally have the antibodies to fight it, therefore the monoclonal antibody will be injected into the bloodstream of the patient. The monoclonal antibody works the same as a natural antibody and will bind to the desired antigen.

The global sale of monoclonal antibodies therapeutics was $20.6 billion in 2006[20], and $153.33 billion in 2020[21]. That’s a yearly growth of 46%. In 2021 the second most sold drug was Humira made by AbbVie [22], which is a monoclonal antibody drug used to treat a number of conditions, including Crohn’s disease, psoriatic arthritis, and ulcerative colitis [23]. Humira was only surpassed by Comirnarty[22] which is the covid vaccine made by Pfizer and BioNTech, which signifies the market value of monoclonal antibody therapeutics.

2.1.2 AIRR data

A deeper understanding of whether immune receptors bind to an antigen could prove advantageous when developing monoclonal antibodies for therapeutics. Adaptive immune receptor repertoire sequencing (AIRR-seq) can be used to sequence the genes encoding TCRs and BCRs [24]. A repertoire is a collection of immune receptors from an individual.
The AIRR data format is used to store the data from the TCR and BCR, and the AIRR community has guidelines for how to use it [25]. The data stored is represented as several text sequences, which represent amino acid sequences. These text sequences can be used for machine learning analyses for prediction of their properties. There have been machine

learning experiments where they have been analysing large amount of AIRR data to create models for predictions [26]. These experiments have proven that machine learning is a powerful tool that can be utilized to get a better understanding of our adaptive immune receptors [26].

The AIRR-seq data contains sequencing data of the TCR and the BCR, which are represented as linear (2D) text sequences. There is an abundance of 2D data compared to 3D data. When in reality the binding between the immune receptor and the antigen take place in the 3D space. 3D data enables the usage of properties such as distance and position, which might provide additional information that could be important for predicting antigen binding. Running machine learning on 3D data has already been done [27]. The results were promising and indicates that this is a feasible way forward [27], although there is an issue of limited 3D data available.

Figure 2 displays the binding between an antibody and the SARS-CoV2 spike protein in 3D space. The green part is the heavy chain, the orange part is the light chain, and the purple part is the spike protein. The figure shows how the chains interact with the spike protein.

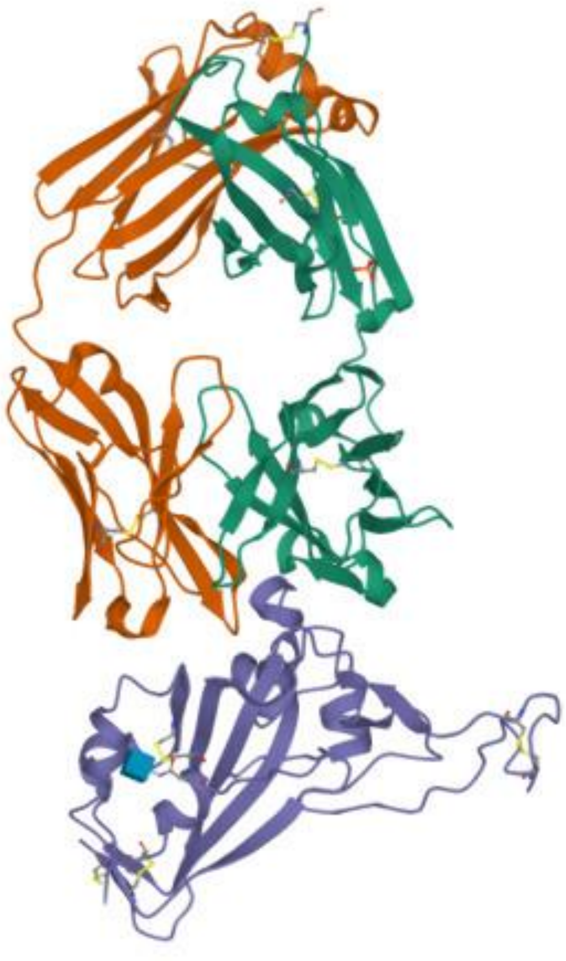


Figure 2: This is the PDB file “2dd8” that contains the crystal structure of a SARS-CoV2 spike binding with an antibody. The heavy chain(green) and light chain(orange) binds with the spike protein(purple). Figure from RCSB[28] deposited by Prabakaran[28]

2.2 immuneML

immuneML is an open-source machine learning platform that is used to analyse adaptive immune receptors and repertoires (AIRR) [7]. immuneML provides several functionalities for analysing an AIRR dataset. One of them is training machine learning classifiers. To run machine learning classifiers, the dataset should be annotated with labels, the label is the property that the machine learning model will learn to predict.
There are currently two types of machine learning classification analyses. The first one is repertoire classification, which takes a repertoire dataset and will try to categorize the different repertoires based on the labels given by the user. Usually this is used with a “sick/healthy” label. immuneML will then try to train a machine learning model that will

predict if the repertoire is associated with a person who is healthy or one that has a sickness.

The second is the receptor classification functionality, which takes in a receptor or a sequence dataset. The receptor dataset contains information about a paired immune receptor, while a sequence dataset contains information about an unpaired immune receptor.
The receptor classification will do the same as the repertoire classification, namely try to categorize the examples in the dataset based on the user given labels, except this time, each example that needs to be classified is a single immune receptor rather than a repertoire. When working with machine learning, not only is the machine learning model important, but also the way the data is represented. This data representation is called encoding.
The main difference between repertoire and receptor/sequence classification, is of course that the data is different, but this also means that they might need different encodings. In immuneML, repertoires, receptors, and sequences each have their own encodings.
Receptor and sequence classification is usually used with data of receptors/sequences that are known to bind to antigens and some that do not. immuneML will try to train a machine learning model that can be used to predict/categorize the data and split them into binders and non-binders.

In immuneML the user can choose multiple machine learning algorithms to run on the AIRR dataset and compare the results to decide which one has the results best suited for the situation. Each machine learning algorithm also has settings the user can define themselves(hyperparameters), for example KNN allows the user to select the n_neigbour value, which determines the number of neighbours required for each sample [29]. immuneML allows the user to select multiple hyperparameter values to get the optimal score between the selected values. To accomplish this, immuneML uses nested cross validation to train and compare the machine learning models, which is a technique based on splitting the data into an inner loop and an outer loop. The outer loop is used to determine the performance of the machine learning model on new data, while the inner loop is used for finding the optimal hyperparameters [30]. These two loops together will find the optimal combination of the machine learning model and hyperparameter values.

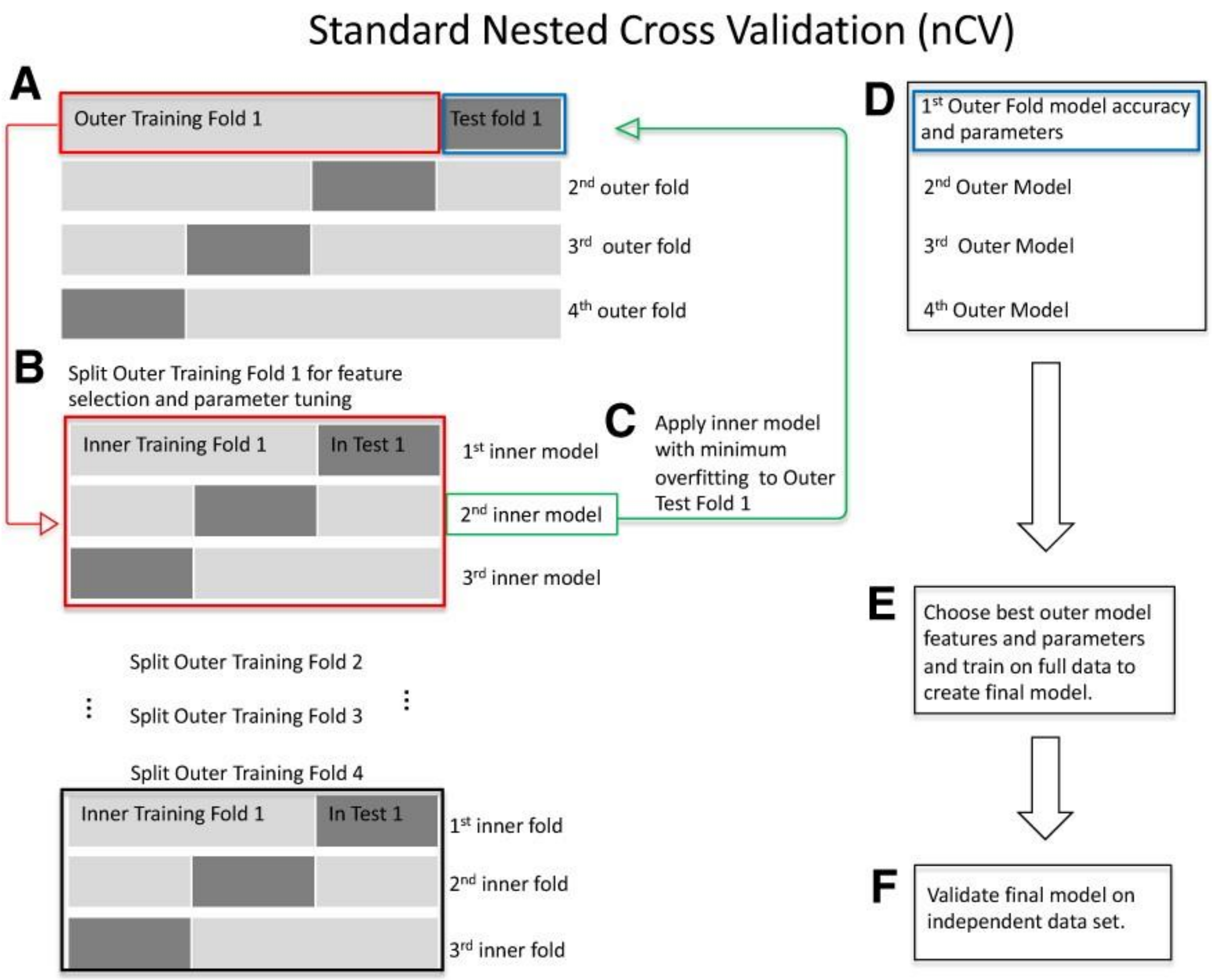
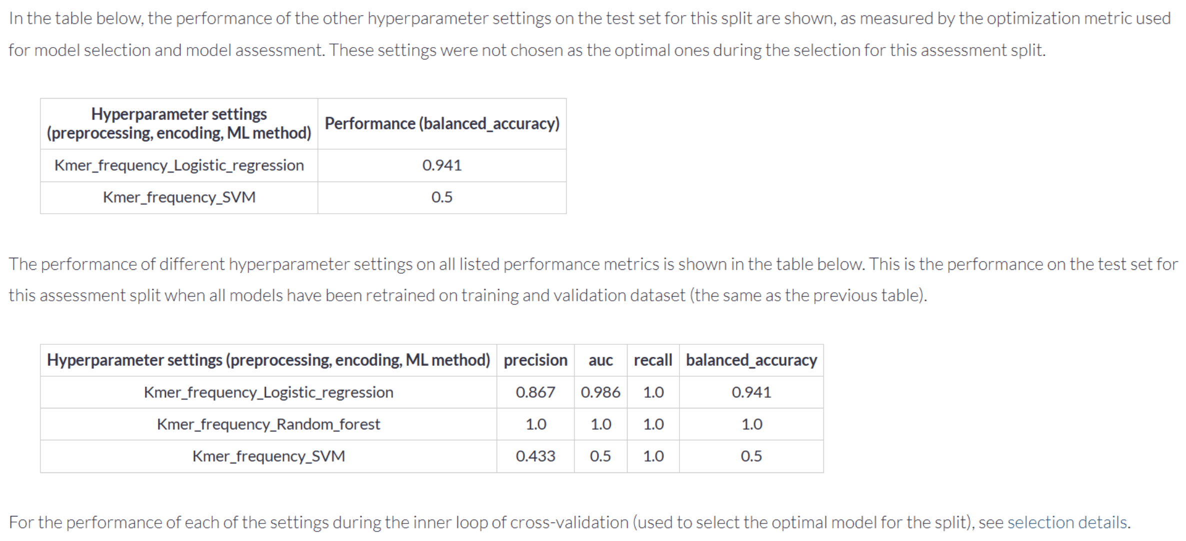

Standard Nested Cross Validation (nCV)
A
Outer Training Fold 1
Test fold 1
2nd outer fold
3rd outer fold
4th outer fold
B
Split Outer Training Fold 1 for feature selection and parameter tuning
Inner Training Fold 1
In Test 1
1st inner model
2nd inner model
3rd inner model
C
Apply inner model with minimum overfitting to Outer Test Fold 1
D
1st Outer Fold model accuracy and parameters
2nd Outer Model
3rd Outer Model
4th Outer Model
E
Choose best outer model features and parameters and train on full data to create final model.
F
Validate final model on independent data set.
Split Outer Training Fold 2
Split Outer Training Fold 3
Split Outer Training Fold 4
Inner Training Fold 1
In Test 1
1st inner fold
2nd inner fold
3rd inner fold

Figure 3: (A) The data is split into four outer identical folds. Each fold's data is split into training and testing. The split is handled in a way that there is no overlap between the testing data across the different folds.
(B) The training data from outer training fold 1 (red box from step A) is split into three inner folds. The data in the inner folds are then split again into a training and testing data, as it was in step A. This is done for feature selection and hyperparameter tuning by using grid search.
(C) The model with least overfitting (green box from step B) is used for testing on 1st outer fold's testing data (blue box in step A).
(D) The model with the best results from this outer fold is saved, as are the features and test accuracies. Then the process is repeated with the remaining outer folds.
(E) The outer model with the least overfitting is selected and is used to train on the full data, to create the final model.
(F) The last step is to validate the model on an independent data set.
Figure by Parvandeh et al (2020). [30]

The user is also able to select which encoding should be used for machine learning. Encoding is a way to translate the raw data, which might be in the form of text to a numerical representation of it. Which encoding that is used for machine learning has a definite impact on the results of the run. Different encodings and its settings will expose different parts of the data. For example, k-mer encoding, which encodes a dataset based on the presence of subsequences of length k, can have different values of k. A k-value of 1 is simply just a single letter, which will make the total number of edges 4 in a DNA sequence (the distinct letters in a DNA sequence are "ATCG" [31]) regardless of the length of the sequence.
While a k-value of 3 will create substantially more edges, because it will create an edge for every distinct three letter combination of ATCG.

These encodings and machine learning algorithms can be used to make machine learning models. immuneML supports multiple combinations of encodings and machine learning algorithms in a single run, combined with the nested cross validation, the user can see which encoding, machine learning algorithm, and setting gives the best suited results. The results are provided in a html file that aggregates the data and shows it in a clear way.


In the table below, the performance of the other hyperparameter settings on the test set for this split are shown, as measured by the optimization metric used for model selection and model assessment. These settings were not chosen as the optimal ones during the selection for this assessment split.

Hyperparameter settings (preprocessing, encoding, ML method)	Performance (balanced_accuracy)
Kmer_frequency_Logistic_regression	0.941
Kmer_frequency_SVM	0.5

The performance of different hyperparameter settings on all listed performance metrics is shown in the table below. This is the performance on the test set for this assessment split when all models have been retrained on training and validation dataset (the same as the previous table).

Hyperparameter settings (preprocessing, encoding, ML method)	precision	auc	recall	balanced_accuracy
Kmer_frequency_Logistic_regression	0.867	0.986	1.0	0.941
Kmer_frequency_Random_forest	1.0	1.0	1.0	1.0
Kmer_frequency_SVM	0.433	0.5	1.0	0.5

For the performance of each of the settings during the inner loop of cross-validation (used to select the optimal model for the split), see selection details.

Figure 4: This is an excerpt from an html file that is generated by immuneML after a run with the kmer encoding and the machine learning algorithms: logistic regression, random forest, and SVM (support vector machine). The selected metrics were precision, auc, recall and balanced accuracy.

immuneML also support certain "exploratory analysis" functionalities. These functionalities are not related to machine learning algorithms but provides ways to analyse the data without training machine learning models. With exploratory analysis the user can select multiple combinations of encodings and reports. Reports generates information about the dataset, for example the SequenceLengthDistribution report creates a histogram of the lengths of the sequences in the repertoire. Certain reports require specific encodings for them to work. Others reports like DesignMatrixReport will work with all encodings, since it will just output the design matrix from the encoding, in the form of a tabular file.

3 Goals

3.1 Adding 3D structure support to immuneML

immuneML supports several machine learning algorithms and encodings. In addition, it also provides exploratory analysis reports that can aggregate information about the dataset. All these functionalities only work on datasets of 2D sequences. Since the antibody-antigen binding takes place in 3D space, it might prove beneficial to utilize the properties that 3D structure data provides [27], for example coordinates and distance. This project will focus on adding 3D structure support to immuneML, which will make it possible to use immuneML's functionalities on 3D data in the future. The focus will be on creating support of 3D antibody data, not TCR data.

The goals and sub-goals of this thesis are as follows:

Implementing 3D structure support:

- **Importing 3D data from files**: immuneML will be able to find and parse the 3D files.
- **Internal representation of 3D data**: immuneML will use the parsed 3D files to create a dataset object that is suited for storage of 3D structures.

Implementation of some first basic analyses to show what is possible with the new 3D structure data:

- **A proof of principle exploratory analysis application**: The provided 3D structures will be used in an encoding that calculates the distance between antibody and antigen, and a corresponding report that will display it in a heatmap.
- **A proof of principle machine learning application:** The provided 3D structures will be used with an encoding that compares structure similarity, and then use machine learning classification algorithms to classify the different structures, based on similarity. The goal is not to get good machine learning results, but rather to show that with the added functionality, machine learning can be done on 3D data.

4 Methods

4.1 immuneML – YAML & workflow

4.1.1 YAML specifications

immuneML takes an AIRR dataset and a YAML specification file as input, and then outputs the trained machine learning models and a summary in the form of an HTML file.

YAML[32] is a data serialization file format that was made to be human readable. It is mostly used as a configuration file for applications. The syntax is a simple key-value format. Where the user specifies the key first, and then a value with a colon separating them. The placements of the key are space sensitive, which means that they need to be placed with the correct length of whitespace in front.

The YAML specification file must follow a specific format to be used in immuneML. The YAML file consists of two main sections and each of them have several sub-sections.
The first main section is the definition section. This section contains the user's selection of dataset, simulation, encoding, machine learning methods, reports, and pre-processing. Each one of these selections have their own settings that the user can define as well.

The second main section is the instruction section. This section the user will define what should be done with the definitions the user specified in the earlier section. immuneML currently have these instructions available for use:

- TrainMLModel – Trains machine learning models and provides hyperparameter optimization through nested cross validation.

- DatasetExport – Exports the dataset to a specific format such as AIRR. The user is also able to apply preprocessing steps on the initial dataset before export.

- Exploratory analysis – Enables the user to combine multiple encodings and reports for an analysis of the dataset, without machine learning.

- MLApplication – Uses already trained ML models and encoders on new datasets. The new dataset does not need to be labelled.

- Simulation – Implants a synthetic signal into the dataset. Outputs a new dataset with the implantation of the synthetic signal.

- Subsampling – Splits the dataset into multiple smaller datasets based on user provided parameters.

All of these instructions have their own settings that the user can specify to get a run specific to their own requirements and needs. See figure 5 for an example of a YAML file.

```yaml
definitions:
  datasets:
    my_dataset: # user-defined dataset name
      format: AIRR
      params:
        is_repertoire: true # we are importing a repertoire dataset
        path: path/to/files/
        metadata_file: path/to/metadata.csv # metadata file for RepertoireDataset

  encodings:
    my_distance_encoder: Distance

  ml_methods:
    my_knn: PrecomputedKNN

instructions:
  my_training_instruction: # user-defined instruction name
    type: TrainMLModel

    dataset: my_dataset # use the same dataset name as in definitions
    labels:
    - signal_disease    # use a label available in the metadata.csv file

    settings: # which combinations of ML settings to run
    - encoding: my_distance_encoder
      ml_method: my_knn

    assessment: # parameters in the assessment (outer) cross-validation loop
      split_strategy: random   # how to split the data - here: split randomly
      split_count: 1           # how many times (here once - just to train and test)
      training_percentage: 0.7 # use 70% of the data for training

    selection: # parameters in the selection (inner) cross-validation loop
      split_strategy: random
      split_count: 1
      training_percentage: 1 # use all data for training

    optimization_metric: balanced_accuracy # the metric to optimize during nested cross-validation when comparing multiple models
    metrics: # other metrics to compute for reference
    - auc
    - precision
    - recall

    strategy: GridSearch # strategy for hyperparameter optimization, GridSearch is currently the only available option

    number_of_processes: 4     # processes for parallelization
    refit_optimal_model: false # whether to retrain the model on the whole dataset after optimizing hyperparameters
```

Figure 5: YAML file that specifies a TrainMLModel instruction with PrecomputedKNN on an AIRR dataset. The two main sections are “definition” and “instructions”. Both main sections have sub sections that define the run and its properties.

4.1.2 immuneML order of execution

immuneML requires a YAML file and a path to the output directory to run. When run; it will first parse the YAML file, which specifies what the user wants to do. The parsing is divided into two parts. The first is parsing the definition section of the YAML file and the second is parsing the instruction section.

After parsing the definition and instructions the specified run will be executed it. The order of specifications defined in the definition section does not matter for the execution order.
If we follow the example YAML from figure 5, it starts with defining the dataset. Here the dataset is specified as AIRR, which is the most common datasets used by immuneML. For immuneML to be able to read the local files, the user needs to specify the path to the folder containing the files, and if it is a repertoire dataset it also requires a metadata file.
The metadata file is a csv file containing information about the datasets. The only requirement for a metadata file is to have a column which contains unique values for each of the files, such as id or filename. The user is free to add more fields to represent specific attributes of the dataset, like diseases. The additionally added fields can be used as labels for machine learning.

This is followed by specifying the encoding and machine learning method to use with the dataset. In figure 5 the specified encoding is Distance, which is an encoder that calculates the distance between each of the repertoires. The calculated distances are stored in a distance matrix.
The specified machine learning method is precomputed KNN, which is a variant of regular KNN, where the distance matrix is already calculated. Different machine learning algorithms have different settings that the user can specify. If it is not specified, it is run with default settings.

The instruction section defines what the user wants to run. Here the user wants to train a ML model. The TrainMLModel instruction has several settings that can be specified by the user to get a run best suited to their needs. If we continue with the YAML from figure 5, the specified settings are:

- Dataset – The dataset that will be used to train the machine learning model. Here it is specified as "my_dataset", which is the AIRR dataset that was defined in the definition section above.

- Labels – The label that will be used for machine learning classification. In repertoire datasets the label is a column in the metadata file, while for receptor and sequence datasets, the label is a column in the data file. The values usually used for labels are TRUE and FALSE. In the example this is set to the label "signal_disease", which is a column in the metadata file.

- Settings – This is where the user specifies what kind of machine learning model that is going to be trained, and which encoding to be used on the dataset. In the example this is set to the encoding "my_distance_encoder" and machine learning model is set to "my_knn".

- Assessment – The settings for the assessment of the outer loop in the nested cross validation. The outer loop is focusing on the model's performance. In the example the split_strategy is set to "random". Split_count is 1, which means that it will only split once. Training_percentage is 0.7, which means that it will use 70% of the data for training the model.

- Selection – This sub-section has the same settings as the assessment section above, but these settings will be used in the inner loop of the nested cross validation.

- Optimization_metric – The metric that immuneML will try to optimize when running nested cross validation. In the example it is set to balanced_accuracy.

- Metrics – Other metrics that also will be computed and presented in the output. This is set to auc, precision and recall in the example.

- Strategy – How immuneML will search for the different hyperparameters. In the example this is set to Gridsearch, which will try all combinations.

- Number_of_processes – The number of processes that will run concurrently to speed up the analysis.

- Refit_optimal_model – This is a Boolean that decides if the model created from running the instructions should be refitted on the entire dataset.

After running all the specified instructions, immuneML outputs the results in the form of a HTML file. See figure 6 for a visual representation of the immuneML order of execution.

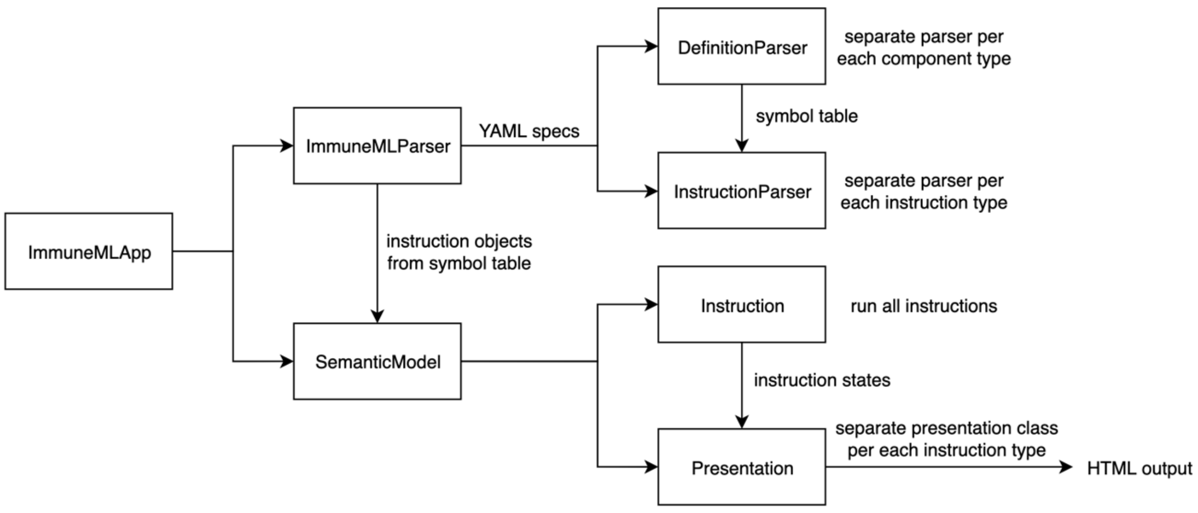

ImmuneMLApp
ImmuneMLParser
YAML specs
DefinitionParser
separate parser per each component type
symbol table
InstructionParser
separate parser per each instruction type
instruction objects from symbol table
SemanticModel
Instruction
run all instructions
instruction states
Presentation
separate presentation class per each instruction type
HTML output

Figure 6: immuneMLApp calls on the immuneMLParser which parses what the user wants to run, then sends the order to the SemanticModel who runs the instructions and presents the results.

4.2 Implementation of 3D structure support

3D structures can be stored in PDB files, which are commonly used within the biological scientific community, and also what will be used in this project.

The PDB file format[2] is a format for storing 3D structural data of biological macromolecules such as proteins. It stands for Protein Data Bank which is a data bank with over 200 thousand structures. The PDB file format has a central role in this project, which is why the structure will be explained in-depth.

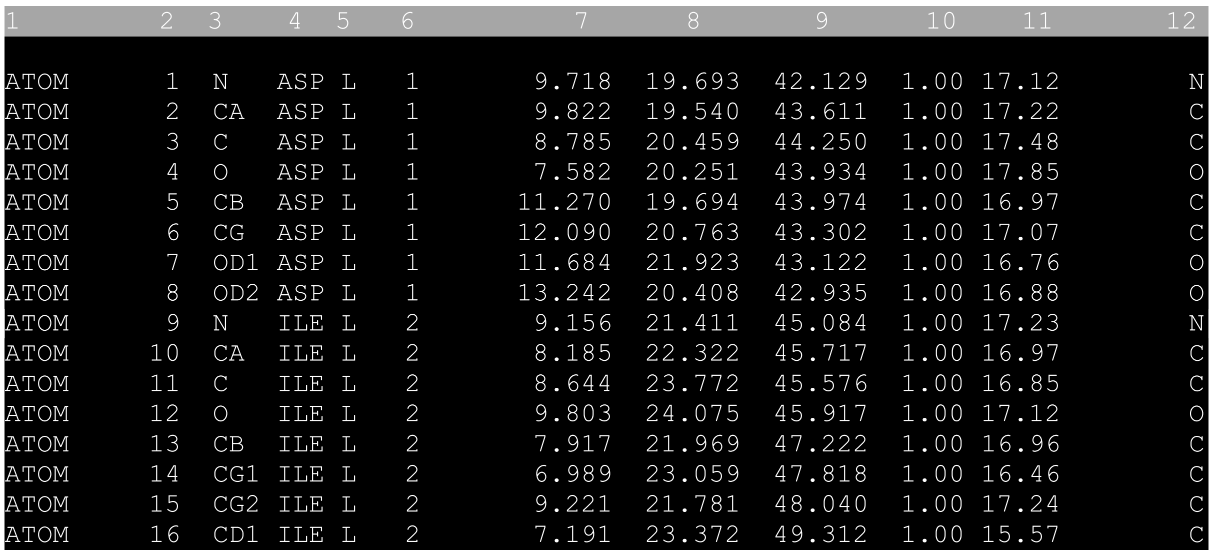
PDB is a text file format that contains information about the atoms and bonds that make up the 3D structure. The format also includes metadata such as the name of the molecule, the author(s) who submitted the structure, and the date the structure was deposited.
See excerpt of the start of a PDB file below.

```
HEADER    COMPLEX(ANTIBODY-ANTIGEN)               11-AUG-88   3HFM
TITLE     STRUCTURE OF AN ANTIBODY-ANTIGEN COMPLEX. CRYSTAL STRUCTURE
TITLE    2 OF THE HY/HEL-10 FAB-LYSOZYME COMPLEX
COMPND    MOL_ID: 1;
COMPND   2 MOLECULE: HYHEL-10 IGG1 FAB (LIGHT CHAIN);
COMPND   3 CHAIN: L;
COMPND   4 ENGINEERED: YES;
COMPND   5 MOL_ID: 2;
COMPND   6 MOLECULE: HYHEL-10 IGG1 FAB (HEAVY CHAIN);
COMPND   7 CHAIN: H;
COMPND   8 ENGINEERED: YES;
COMPND   9 MOL_ID: 3;
COMPND  10 MOLECULE: HEN EGG WHITE LYSOZYME;
COMPND  11 CHAIN: Y;
COMPND  12 EC: 3.2.1.17;
COMPND  13 ENGINEERED: YES
SOURCE    MOL_ID: 1;
SOURCE   2 ORGANISM_SCIENTIFIC: MUS MUSCULUS;
SOURCE   3 ORGANISM_COMMON: HOUSE MOUSE;
SOURCE   4 ORGANISM_TAXID: 10090;
SOURCE   5 MOL_ID: 2;
SOURCE   6 ORGANISM_SCIENTIFIC: MUS MUSCULUS;
SOURCE   7 ORGANISM_COMMON: HOUSE MOUSE;
SOURCE   8 ORGANISM_TAXID: 10090;
SOURCE   9 MOL_ID: 3;
SOURCE  10 ORGANISM_SCIENTIFIC: GALLUS GALLUS;
SOURCE  11 ORGANISM_COMMON: CHICKEN;
SOURCE  12 ORGANISM_TAXID: 9031
```

Figure 7: Excerpt from “3HFM.pdb”. PDB files start with the name of the file, followed with naming of the chains and the source of them.

In the excerpt of the pdb file “3HFM” in figure 7, it shows that the file contains a: “STRUCTURE OF AN ANTIBODY-ANTIGEN COMPLEX. CRYSTAL STRUCTURE OF THE HY/HEL-10 FAB-LYSOZYME COMPLEX”. This structure has the three chains: L, H, and Y. Where the L chain corresponds to “HYHEL-10 IGG1 FAB (LIGHT CHAIN)”. The H chain corresponds to “HYHEL-10 IGG1 FAB (HEAVY CHAIN)”, and the Y chain corresponds to “HEN EGG WHITE LYSOZYME”.
The light and heavy chain are from a mouse while the lysozyme is from a hen egg.

The main part of a PDB file is the ATOM section, which contains a list of all the atoms in the structure; their coordinates, which amino acid it is a part of, which chain it is a part of, and their atom id.



```
1           2  3    4  5   6          7       8        9        10     11        12

ATOM        1  N    ASP L   1      9.718  19.693  42.129  1.00 17.12           N
ATOM        2  CA   ASP L   1      9.822  19.540  43.611  1.00 17.22           C
ATOM        3  C    ASP L   1      8.785  20.459  44.250  1.00 17.48           C
ATOM        4  O    ASP L   1      7.582  20.251  43.934  1.00 17.85           O
ATOM        5  CB   ASP L   1     11.270  19.694  43.974  1.00 16.97           C
ATOM        6  CG   ASP L   1     12.090  20.763  43.302  1.00 17.07           C
ATOM        7  OD1  ASP L   1     11.684  21.923  43.122  1.00 16.76           O
ATOM        8  OD2  ASP L   1     13.242  20.408  42.935  1.00 16.88           O
ATOM        9  N    ILE L   2      9.156  21.411  45.084  1.00 17.23           N
ATOM       10  CA   ILE L   2      8.185  22.322  45.717  1.00 16.97           C
ATOM       11  C    ILE L   2      8.644  23.772  45.576  1.00 16.85           C
ATOM       12  O    ILE L   2      9.803  24.075  45.917  1.00 17.12           O
ATOM       13  CB   ILE L   2      7.917  21.969  47.222  1.00 16.96           C
ATOM       14  CG1  ILE L   2      6.989  23.059  47.818  1.00 16.46           C
ATOM       15  CG2  ILE L   2      9.221  21.781  48.040  1.00 17.24           C
ATOM       16  CD1  ILE L   2      7.191  23.372  49.312  1.00 15.57           C
```

Figure 8: The beginning of the atom part of the PDB file “3HFM”

The atom part has in total 12 columns that contain information about each of the atoms in the structure.

Column number from left	What the column contains
1	This is the section identifier. “ATOM” means that this is the atom section containing information about all the atoms in the structure.
2	This is the atom identification number; each atom has their own unique id number
3	See explanation after the table.
4	What type of amino acid it forms. Every amino acid has their own unique three letter identifier.
5	Which chain it is a part of. Normally L = light chain, and H = heavy chain.
6	This is the amino identifier in the PDB file. Every amino acid in the file will have their own unique id number.
7	X coordinate of the atom
8	Y coordinate of the atom
9	Z coordinate of the atom
10	This column corresponds to Occupancy. Occupancy indicates how certain it is that the atom is located at that location. Since the atoms are always moving, it might not be easy to distinguish atoms of different amino acids from each other. Having an occupancy score of 1 means that it is certain that the specific atom is located at the given position. While a score of 0.5 means that the atom was not present at the location in half of the times it was checked.
11	This is the temperature column. Given that the atoms are constantly moving, it is important to know the temperature of the different atoms. Higher temperature means that the atoms are moving faster and vice versa.

12	The last column simply shows the atom element symbol. This doesn’t contain any Greek letters or numbers.

Table 1: Table explaining the columns in the atom section of a PDB file.

Column 3:

This column contains the atom type identifier. For example: N means the atom nitrogen. Note that there might be multiple occurrences of the same atom in the same amino acid. To differentiate them from each other, they can have numbers and Greek letters after the normal atom letter. For example: in row 2 the atom is a “CA”, which corresponds to carbon alpha. The order of atoms within an amino acid residue is determined by their location within the amino acid, whether it is in the main chain or in the side chain.

The main chain consists of the carboxyl group, amino group and a carbon alpha that is bound to a hydrogen atom.

The atoms in the main chain will not have any Greek letters or numbers, just simply the atom, except the carbon alpha. The carbon alpha is often considered to be the centre of the amino acid, because it is the atom connecting the main chain to the side chain.

Carbon betas and gammas corresponds to carbon atoms in the side chain. This is a naming convention widely used in the biological community. That’s also the reason why there are instances of “OD1” and “CG2” in the PDB example above. These names correspond to atoms at specific locations with certain properties. For example, OD1 and OD2 are both oxygen atoms bonded to carbon gamma, but it is important to differentiate between them.

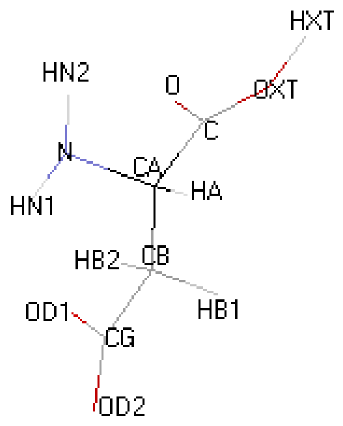


Figure 9: Composition of the amino acid Aspartic acid (ASP). [33]

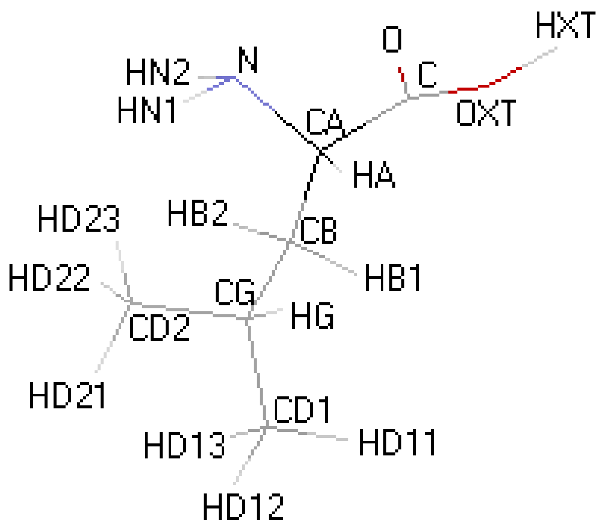


Figure 10: Composition of Leucine (Leu). This amino acid contains both CD1 and CD2, who has their own individual chain. This figure shows the importance of the naming conventions, otherwise the atoms could get mixed up. [33]

The PDB file is great for storing 3D structures but are insufficient for storing specific information about immune receptors. This is why the IMGT numbering scheme has been made. The IMGT numbering scheme is especially useful since it keeps the different CDRs at specific positions. For example, a basic length CDR3 is always at the positions 105-117. The IMGT numbering scheme is not a file format, but rather a standard for where to place the different amino acids. This numbering scheme is usually used in combination with the PDB file format. IMGT numbered PDB files are the best suited files for this project.

Currently immuneML only supports sequence, receptor, and repertoire datasets. These datasets cannot properly store the data in a PDB file. This makes it necessary to create a new dataset; the PDBDataset.

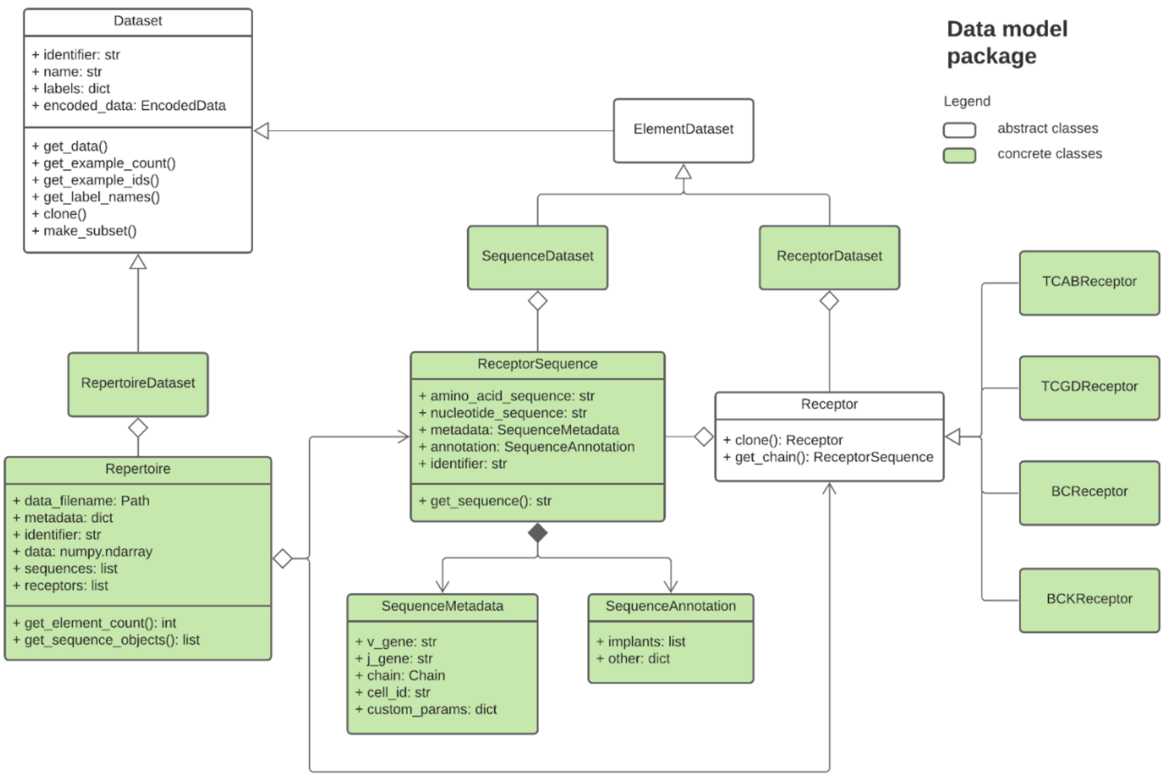


Figure 11: This is a UML class diagram of the current immuneML data model. The repertoire, sequence, and receptor datasets are all sub-classes of the «Dataset» object.

In immuneML each dataset has its own importer class that handles the import of the local files to the corresponding dataset object. In this case there needs to be a PDBImporter that gathers the files from the user and generates a PDBDataset object with the file paths. It should also handle the metadata file and labels. The metadata file will contain the filename and the label as columns.

Every dataset class in immuneML has these attributes:

Attribute	**Value/type and why**
Identifier	Is a random UUID used to differentiate between datasets in the memory, if multiple instances are run simultaneously.
Name	User can name the dataset, otherwise it is set to the same as identifier
Labels	Is a dictionary where the key is the columns in the metadata file, and the value is a list of all the distinct values in the column. This dictionary is used to access the labels specified by the user.
Encoded_data	Is an EncodedData object, which is generated by the different encoders. If immuneML is run without an encoder, this will be a “None” object. EncodedData class contains a two-dimensional matrix and information about the data in it, such as: labels, ids, and feature names.

Table 2: The attribute column describes the variable names. The value/type and why column describes the type of the attribute and the purpose of the variable.

In addition to these attributes the PDBDataset has these as well:

File_names	List of names of the pdb files. The names are used as row values in eventually generated two-dimensional matrices.
Pdb_file_paths	List of the entire path to the pdb files. These file paths are used to access each of the pdb files.
List_of_pdb_structures	List of PDBStructure objects. Its these objects that are accessed for performing calculations.
Metadata_file	Is a Path object from the pathlib module. This is used to access the entire metadata file.

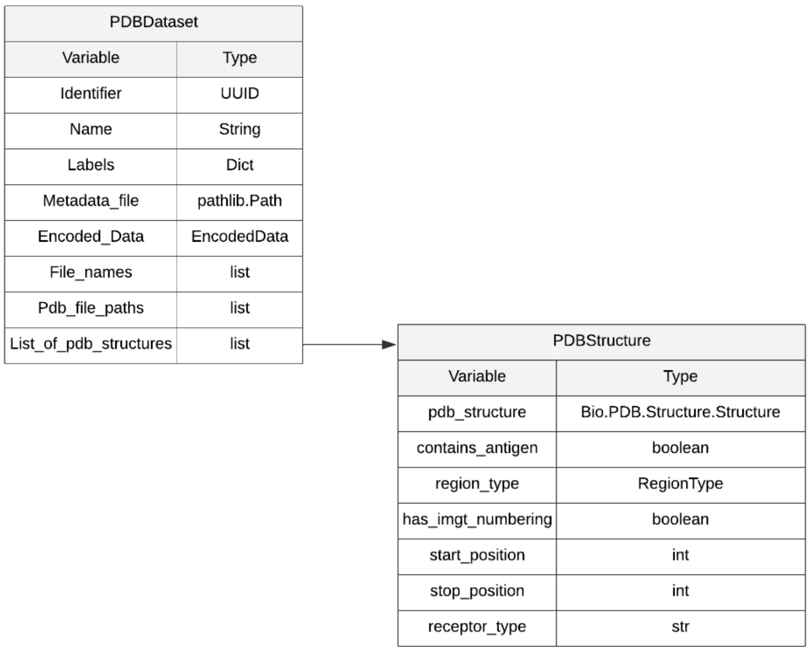
Table 3: This table has the same columns as Table 2 and follows the same style.

The PDBDataset contains many lists. There is a list of all the PDB filenames, paths to these files, and a list of PDBStructures. Which is another class made specifically in this project for handling the data stored in a PDB file.

There already exists a python module for parsing pdb files. This parser is called PDBParser[34] and is a part of the PDB package[35] made by BioPython [36]. The PDB parser is made specifically for converting the information in a PDB file, to a Structure object[37], which is another class from the PDB package. It only requires a user defined name for the structure, and a path to the PDB file as arguments.

The Structure class consists of a "Model" object. This object consists of a dictionary of chains, which in turn consists of residues and atoms. Each chain has a key that corresponds to the name given in the PDB file. With this key, the user can specify which chain it wants to access. The model, chains, residues, and atoms are all iterable.

The classes provided by the PDB package are great assets, but they cannot contain all the necessary information that immuneML would like. Information such as, if the PDB file contains an antigen, which part of the region it is, and if it is IMGT numbered, are useful information for immuneML. Which is why the object PDBStructure was created. This object keeps tracks of variables that can be used by immuneML. The new PDBStructure object also contains a Structure object from the PDB package.



PDBDataset	
Variable	Type
Identifier	UUID
Name	String
Labels	Dict
Metadata_file	pathlib.Path
Encoded_Data	EncodedData
File_names	list
Pdb_file_paths	list
List_of_pdb_structures	list

PDBStructure	
Variable	Type
pdb_structure	Bio.PDB.Structure.Structure
contains_antigen	boolean
region_type	RegionType
has_imgt_numbering	boolean
start_position	int
stop_position	int
receptor_type	str

Figure 12: The classes PDBDataset and PDBStructure; their variables and types.

PDBDataset uses the list of file paths to generate a list of PDBStructures. When generating this list, the first thing the PDBDataset does is to check whether the PDB file is made with IMGT-numbering[38] or not. With IMGT-numbering it's much easier to determine the CDRs. Especially if the user wants specific parts of the 3D structure in the analysis. The user can select which CDR they want, but only if the PDB file has IMGT-numbering, otherwise the user needs to specify the start and stop position of the amino acids to be included in the analysis. In the PDBStructure class the CDRs are specified as a RegionType object. The reason the CDRs are so important is because these are the regions that are in direct contact with the antigen. This is the most interesting part of the 3D structure. Some analyses may want to focus on a specific CDR and not the entire antibody structure.

The PDBDataset makes it possible to process the data contained in the PDB file. In this project we want to make some basic encodings and reports, that will run on the new dataset, as a proof of concept.

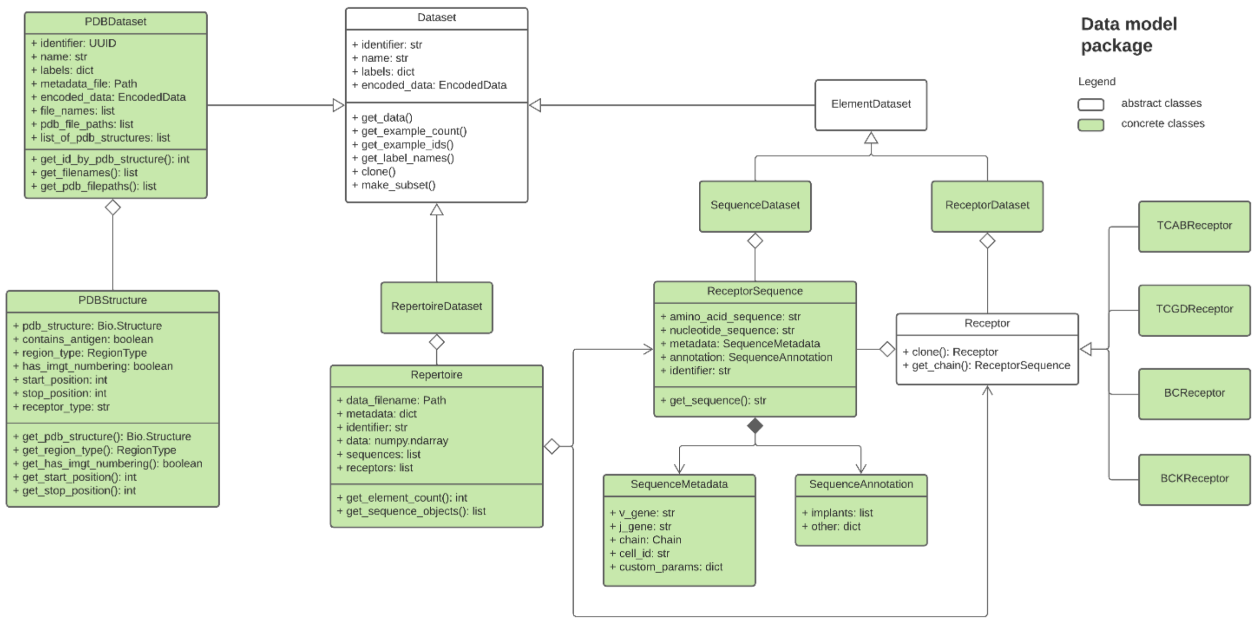


Figure 13: The new updated immuneML data model with the new data structures “PDBDataset” and “PDBStructure”.

4.3 Generating distance heatmap

4.3.1 DistanceToAntigenMatrixEncoder

The main goal by creating a distance heatmap generator is to show the information that the PDBDataset can provide. This will be an ExploratoryAnalysis functionality, which means that it will give an analysis of the data, but it will not run machine learning.
The DistanceToAntigenMatrixEncoder is an encoder that will take a PDBDataset and calculate the distance between each carbon alpha from the antibody chains to the antigen chain in each of the PDB files.

PDB files containing an antibody-antigen binding usually consists of a light, a heavy and an antigen chain. The light chain is usually given the chain id “L” or “A” and the heavy chain given the id “H” or “B”. The antigen has a larger variety of ids, but the most commons ones are “A”, “E”, “P”, and “C”. But this isn’t always the case, as authors of PDB files are free to choose whatever character they want. It is therefore important that the distance calculations are done on the correct chains, specifically between the light and the antigen, and between the heavy and the antigen, thus avoiding calculations between the light and the heavy chain.
This is why the encoder starts with trying to detect which chain is which. The detection is based on the light chain is named “L”, and the heavy chain is named “H”. If both the light and the heavy chain are detected, the last remaining chain is then designated as the antigen. If the detection fails to detect the chains, it will initiate the fallback routine which is selecting the first chain as the light chain, the second as heavy, and the last as the antigen.
Note that there is an overlap on the id “A”, both the light chain and the antigen tend to be given that id. The detection will encounter problems when faced with a PDB file where the antigen is listed first and given the id “A”. A solution to this problem could be a provisional detection first, and then a final one based on all the three detected ids. This solution was not implemented because of time constraints.
With the implemented detection of chains, it decreases the chances of distance calculations between the light and the heavy chain. Furthermore, the detection will not work when the chains are named something else than the ids listed above. For example, if a PDB file lists the

light chain as G, the heavy as Y, and the antigen as T, the detection will go with the fallback routine.

There are many ways to calculate distance between the antigen and the antibody chains. One approach is calculating the distance between the carbon alphas in the antibody to the carbon alphas in the antigen. Carbon alphas are considered the “centre” of the amino acids.
Other approaches could be calculating the distance from the centre of mass in each amino acid, or closest atom to closest atom. In this project carbon alphas are used. Although in the future, the algorithm could be extended to support the other types of distances as well.

The encoder class calculates the distances from the carbon alphas of the amino acids in the light and heavy chains to the antigen. The distances are put in a two-dimensional matrix, both the light and the heavy chain have a two-dimensional matrix each. These two-dimensional matrices are stacked, where each 3D structure has their own entry. Which means that the data structure is now a four-dimensional matrix. See figure below for a visualized representation.

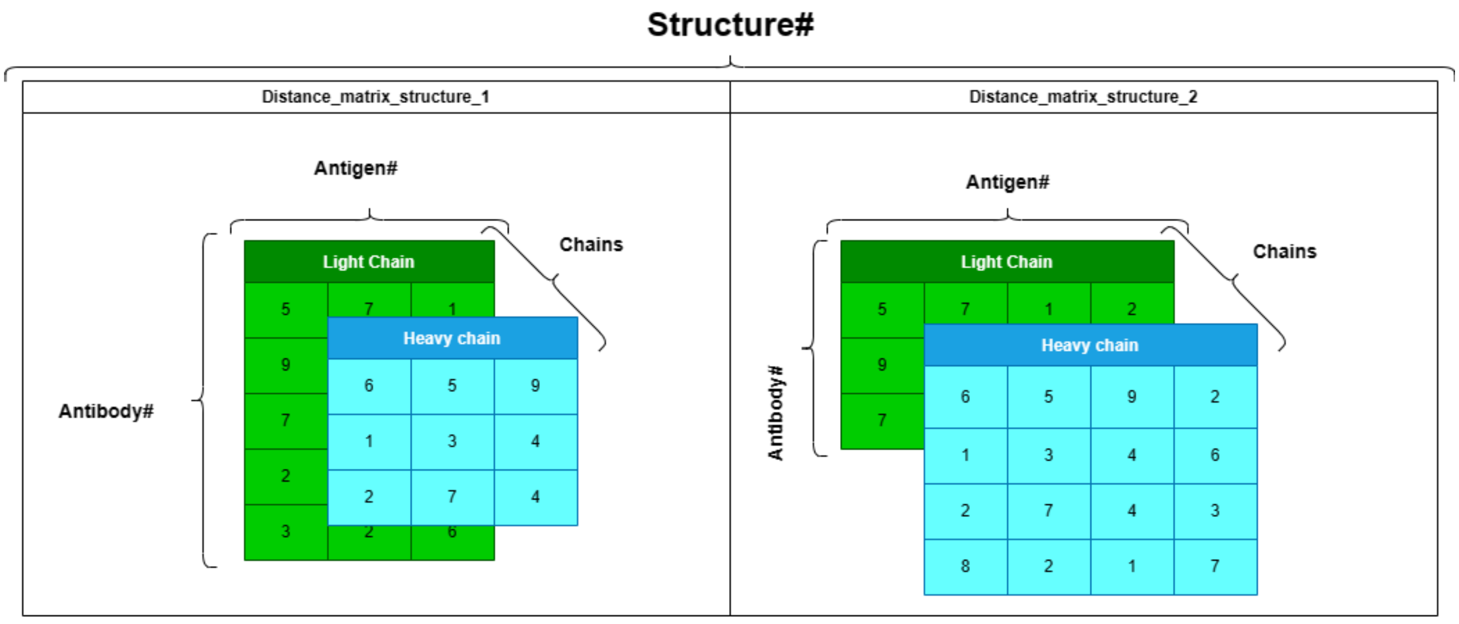


Figure 14: The structure array consists of two matrices for each structure. The matrices contain the distance between the chain and the antigen. The four dimensions are: Structure#, chains, antigen#, and antibody#.

Notice how different structures have different length of antigens and chains. This makes it a three-dimensional matrix where the sides have notches. Although these are the actual shapes of the different structures, the matrix can’t have notches since it needs to square. Because of this, the matrix needs to be padded to the length of the longest chain in that axis. In this encoder the matrix is filled with the numpy.inf value. This number was preferable than filling it with zeroes because that could potentially be a valid distance. The numpy.inf value is later removed for the generation of heatmaps. In the future there could be a suitable machine learning model that can directly use the matrix, but for generation of heatmaps the numpy.inf value is not useable.

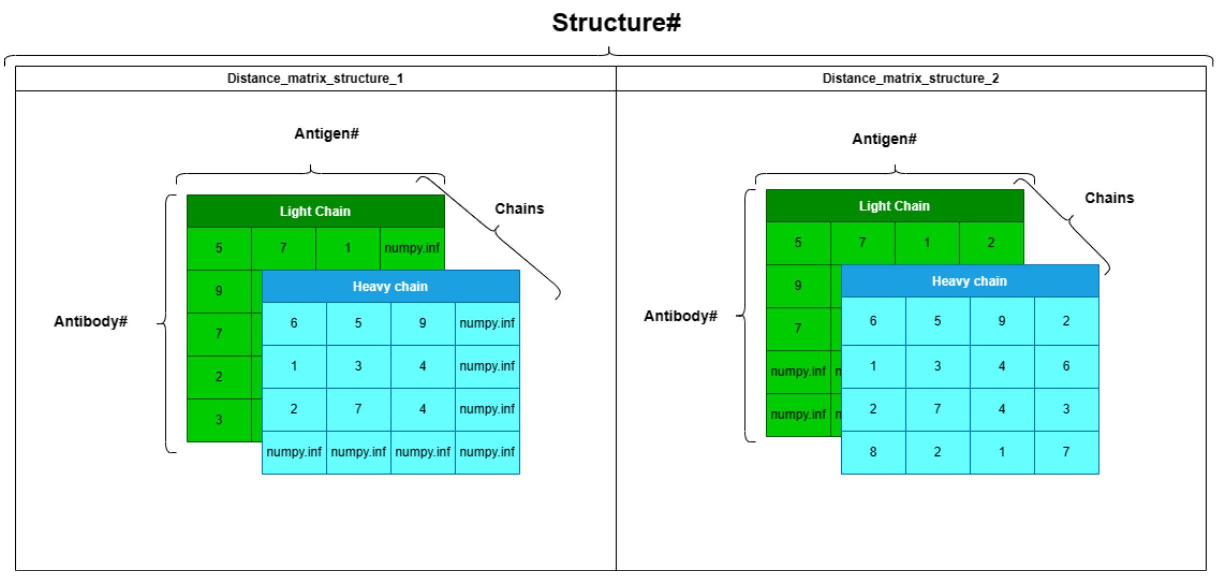


Figure 15: The light- and heavy chains of both structures are padded to the same shape of their respective chain.

The padded four-dimensional matrix is used to create an EncodedData object, which is added to the PDBDataset.

4.3.2 DistanceToAntigenHeatmapReport

After encoding the dataset with the distance from the light and the heavy chain to the antigen for every structure, we need to display it in the form of a heatmap. This is simply done by iterating over the four-dimensional array stored in the EncodedData object.
The numpy.inf values are removed before the heatmaps are created for each of the structures.

The encoder also supports selection of specific regions. By doing so, the user can reduce the amount of carbon alphas in the graph, to only the parts the user wants. Currently the options are the three different CDRs and the full sequence, which is the entire PDB file. If the user selects CDR3, the encoder will only use the carbon alphas in the CDR3 region. This is only possible if the PDB file is IMGT numbered. IMGT numbered PDB files have the specific CDRs at specific amino acid placements. For example, CDR3 are located with the amino acid placements between 105 and 117.
If the PDB file is not IMGT numbered, the encoder will use the start and stop positions provided by the user in the metadata file.

4.4 Training a machine learning model with a PDBDataset

The previous encoder and report show the basic functionalities that the PDBDataset can provide. In the future it can be used for machine learning as well, but only when a suitable machine learning method has been implemented.
immuneML is fundamentally a machine learning platform, therefore the PDBDataset should also work with machine learning algorithms. That is why the next encoder was made with utilizing KNN for classification (K-nearest neighbours) in mind. This encoder will produce a design matrix that is more similar to the already existing design matrices, that are created by the other dataset types.

The KNN classification algorithm is a machine learning algorithm that tries to categorize the data based on which neighbour it’s close to. It is a supervised algorithm, meaning that it requires labelled data. In immuneML the label is provided in the metadata file.

The goal is to see if the KNN algorithm can categorize different PDB files based on the structure of the antibody.

4.4.1 DistanceBetweenStructuresEncoder

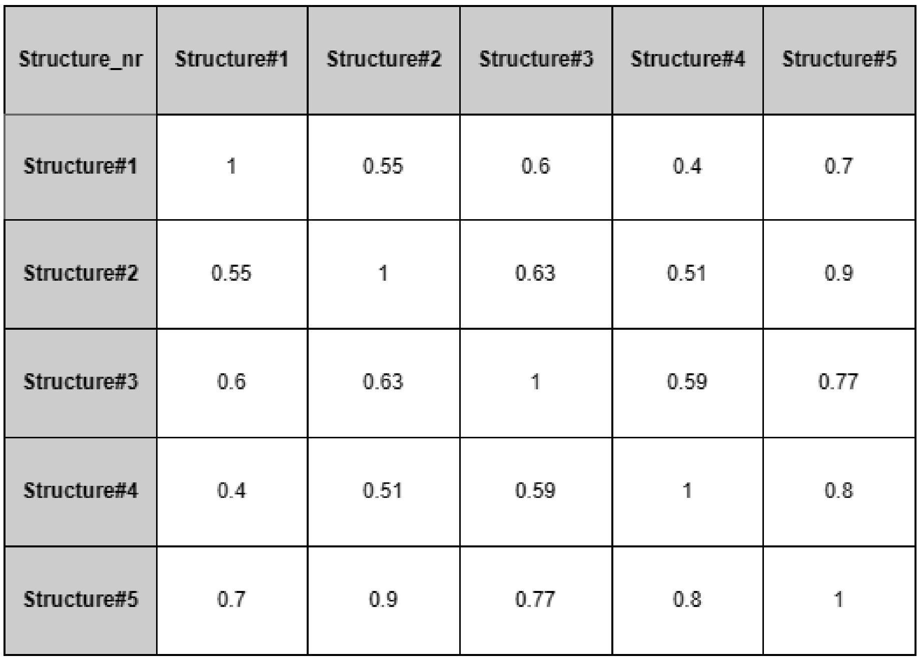
The purpose of this encoder is to categorize the different antibodies based on their structural similarities. For example, are binding antibodies structurally similar? Are non-binding antibodies? Are antibodies that bind to a specific antigen? These are a few of the categorizations that could occur when training a machine learning model, based on structural similarities.
Therefore, for the purpose of this encoder, only the antibody is used, and not the antigen. The encoder will take the PDB dataset and remove all the antigens. To do this, we use the PDB.Select[39] module, which also is a part of the PDB package by BioPython. This module will take a PDB structure and create a new one based on specific rules. In this case, it will only accept chains that are named “L” or “R”. These letters usually correspond to the light and the heavy chain in the antibody. Although, since the author of the PDB file is free to name the chains whatever they want, there are cases where they are named something else. The light and the heavy chain are also named “A” and “B” quite often, but the probability of an antigen chain named “A” is higher. Accepting an antigen chain could ruin the machine learning model, which is why only “L” and “R” are accepted.

After generating the new PDB files without the antigen, they are re-read by the PDB parser and treated as new PDB structures. This is preferable to overwriting the existing local files.

The next step is to determine how similar the structures are to each other. An approach for structure comparison could be to compare the root mean square deviation (RMSD) of the two structures. This is calculated by taking the average of the distances between the atoms of the different structures. An issue with RMSD is that all residue pairs are weighted evenly. This makes it heavily influenced by the magnitudes of errors, especially when the RMSD value is large [40]. For example, if two structures have a similar topology, but there are local areas where there are significant differences, the score in the local areas will skew the total score towards dissimilarity [40].

This is one of the motivations behind the creation of TM-score(Template modelling score)[41]. The TM-score is based on RMSD but correcting some of the limitations. TM-score is made to capture the global structure similarity. It also takes the protein size into account, by using scale factor for differences between protein sizes.
TM-score has been used in previous studies, such as Polychronidou et al (2018) [42], and is the distance metric that will be used in this encoder. Although multiple distance metrics could be added in future implementations.

TM-score mainly works on structures of the same length. In this project the structures might have very different lengths, which means that only using TM-score won't be possible. This is where TM-align[43] will be useful. This algorithm will align the structures in a way that will get the highest TM-score, even when the structures have different length. The TM-score generated by the TM-align algorithm is put in a two-dimensional array. KNN will use this array and the labels from the metadata file to a train machine learning model.



Structure_nr	Structure#1	Structure#2	Structure#3	Structure#4	Structure#5
Structure#1	1	0.55	0.6	0.4	0.7
Structure#2	0.55	1	0.63	0.51	0.9
Structure#3	0.6	0.63	1	0.59	0.77
Structure#4	0.4	0.51	0.59	1	0.8
Structure#5	0.7	0.9	0.77	0.8	1

Figure 16: Example of the two-dimensional distance matrix of tm-scores. The TM-scores range from 0 to 1, where 1 means that the structures are exactly alike, and 0 means the structures are significantly different. A TM-score of ≥0.5 signifies that the structures are in the same fold, and ≤0.5 signifies the opposite[44].

Since the encoder calculates the TM-score between every structure, the encoding's time complexity is quadratic (N*N), which is quite long. Normally when working with native K-fold cross validations, the encoder would have to perform these calculations multiple times.

This is not the case for immuneML since it provides caching functionalities, ensuring that the TM-score calculations are only computed once, and then saved in a pandas.Dataframe that is accessible between different folds. The usage of caching drastically reduces the runtime of the encoder.
K-Fold cross validation is a technique in machine learning where you create multiple folds of the dataset, and then dividing the usual training, testing and validation data differently in all the folds.

After calculating and creating the two-dimensional distance matrix of tm-scores, it is passed to the next step, namely the training of the machine learning model. The two-dimensional array is used as the distance matrix for the precomputed KNN machine learning algorithm.

After implementing these mentioned classes, they are ready to be used. The first step in starting an immuneML PDB run is gathering the PDB files. There are certain databanks where there are PDB files available for download.

4.5 Data selection and handling

The most used databank is the RCSB Protein Data Bank(https://www.rcsb.org/). This databank has over 200,000 PDB structures. These structures could be anything, from sperm whale DNA to parts of cancer cells. Even though it is called a protein data bank, it doesn’t necessarily mean that every structure in it is a protein either. It simply means that the structure is stored in the PDB file format. For this immuneML application, antibody-antigen structure data is required, which is a small fraction of what is in the RCSB databank.

Another databank is the IMGT 3D-structure databank (https://www.imgt.org/3Dstructure-DB/). This is the same IMGT that created the PDB IMGT-numbering system. The PDB files in this databank are all IMGT numbered and related to immunology. This databank is ideal for finding files for the DistanceToAntigenMatrixEncoder. The reason is because these files contain the CDR at specific locations.

Even if the PDB files are IMGT numbered, they might not be suited for the encoder. The files need to have a light, heavy and antigen chain. And it also needs to have at least one carbon alpha in the region specified. The search system on their webpage isn't that extensive which makes it harder to gather files with specific properties. A better approach would be to use RCSB's search fields and then try to find that same file in IMGT's database. This is easier because RCSB has more precise search options, but also because PDB files have the same name between databases. It usually is a combination of numbers and letters with a total length of four. PDB files in the IMGT database are normally files from RCSB's database, just renumbered with their own numbering system.

Finding enough suitable files for DistanceBetweenStructuresEncoder is a much more challenging task. Not because the encoder can't handle it, but because it will be used with KNN for classification. The data for this encoder needs to be in different large groups, so KNN can try to categorize them.
PDB files are usually created when researching a specific antigen-antibody binding. There are rarely created a larger amount of PDB files that share a common property or behaviour. A property could for example be that they all bind to a specific antigen, or that they all are associated with sick individuals. A way to gather a dataset would be to first decide on a category and then manually find the PDB files in that category.
It would be ideal to have datasets of different antigens, and then see if KNN is able to distinguish them from each other. Unfortunately, there are very few larger datasets of specific diseases. In addition, there are certain requirements of the PDB files to be used with this encoder. The previous encoder needed the PDB files to have a light, a heavy, and an antigen chain. The current one has the same requirements, but it also needs to be a protein.

A database that might have enough data in a category is Oxford Protein Informatics Group's CoV-AbDab [45]. This is a database of coronavirus antibodies. After the covid outbreak, there has been a drastic increase of coronavirus samples, which makes it an ideal target for the machine learning use case.
On their site there is a downloadable csv file containing information about every entry in the database. The csv file has over 12000 lines of different antibodies, but only some of them correspond to PDB files.

It would be rather time consuming going through every single entry marked with the find function. A better approach would be to convert the information in the csv file to a pandas.Dataframe and then use the functionalities it provides. The converted dataframe has many columns, but the ones relevant in this run are these:

Column	Explanation
Name	Unique ID for every entry
Ab or Nb	Antibody or Nanobody
Binds to	Which antigen it binds to
Protein + Epitope	What type of protein it is, and where it binds itself to the antigen
Origin	Where the antibody originated from
Structures	Link to a database where the structure is stored
Sources	Which research paper the structure is a part of

Table 4: The different columns and what information it contains.

Using the functionalities pandas provides we can extract all the rows that fulfil these conditions:

Column	Value	Why
Name	*	This is only to get the row ID. The value can be anything.
Ab or Nb	Ab	Antibodies would be better used for this project than Nanobodies. Since nanobodies is only a part of an antibody.
Binds to	SARS-CoV2	There are many types of SARS-CoV2. For this KNN run, it doesn’t matter which version it is.
Protein + Epitope	S	S is short for spike protein, which is the protein that contains the payload that starts the viral infection.
Origin	*	For this run it doesn’t matter where the antibody originated from.
Structures	RCSB	This is the most important requirement for the row. Only the rows containing RCSB has an obtainable PDB file.
Sources	*/RCSB	Some rows contain the RCSB link here instead of the structure.

Table 5: This table shows what values are being filtered and why.

These requirements give us a much smaller dataframe with a length of around 500 rows. Since we want KNN to classify two different categories, we should find two groups of data that the algorithm will try to classify.
Figure 17 shows the different unique values in the “Protein + Epitope” column.

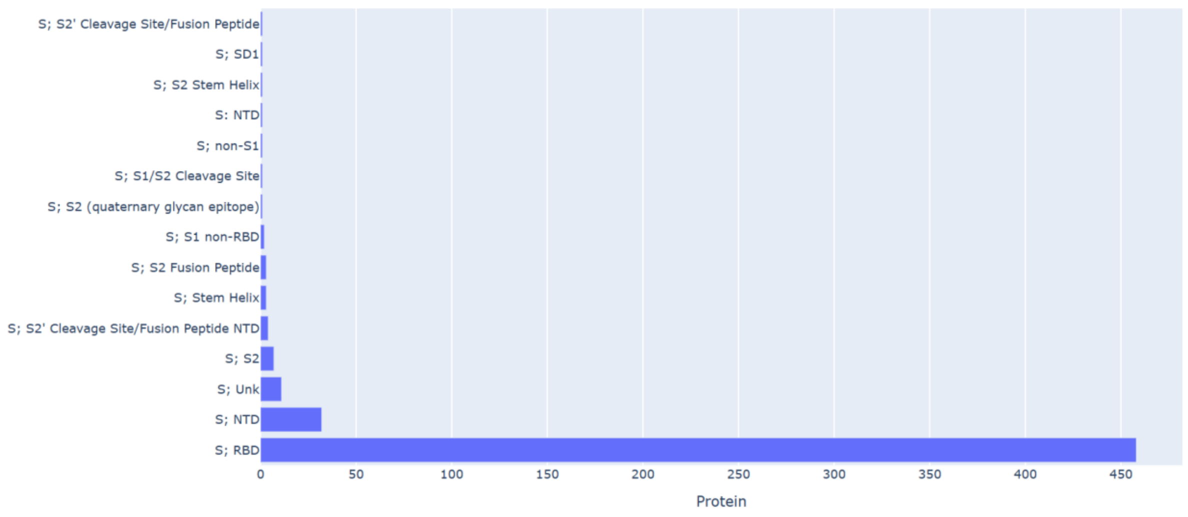
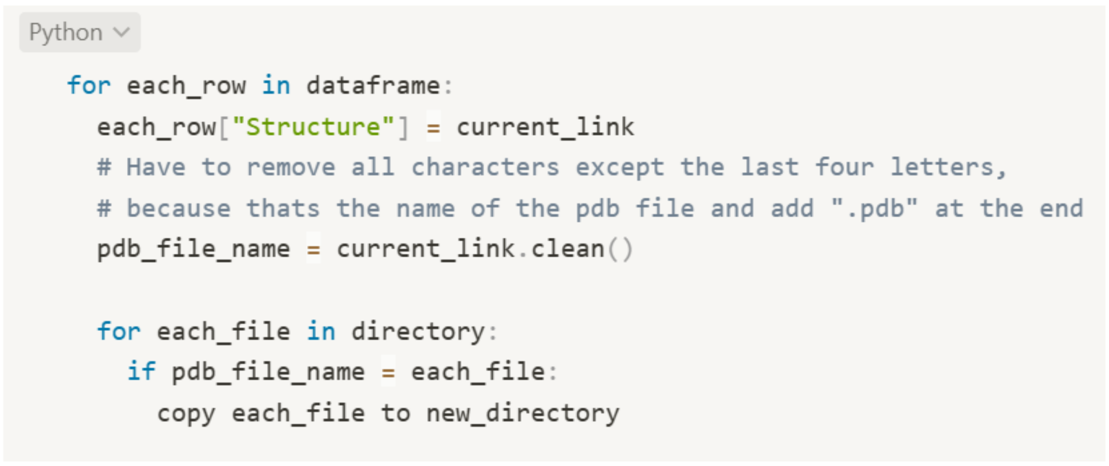


Figure 17: Protein is “spike” in all of the rows. The main difference is the Epitope part. Epitope is the region on the antigen where the antibody binds to. NTD means N-terminal domain and is a specific part of the antibody that binds to the epitope. RBD is receptor binding domain, which also is a specific part of the antibody that binds to the epitope.

There is a clear abundance of RBD files, and the second highest value is NTD. The ideal run with this data would be an NTD vs RBD classification run with KNN. There are over 450 RBD files and around 30 NTD files.

The CoV-AbDab site also provides the option to download all the pdb files as several zipped folders. Not all of the files are usable for the project, which means that we need to find the files listed by our pandas’ requirements. Manually finding each file listed would take quite an extensive amount of time. The best approach would be to write a python script that uses the list generated in the pandas dataframe and iterate through it, finding the corresponding PDB file and moving it to a folder.



```
for each_row in dataframe:
  each_row["Structure"] = current_link
  # Have to remove all characters except the last four letters,
  # because thats the name of the pdb file and add ".pdb" at the end
  pdb_file_name = current_link.clean()

  for each_file in directory:
    if pdb_file_name = each_file:
      copy each_file to new_directory
```

Figure 18: Pseudo code of the file handling operation. The rows are all stripped of all characters except the last four, because that’s the PDB id. Then the script will use the id and iterate through the directory until it finds the corresponding PDB file. After finding the file it is copied to a new directory.

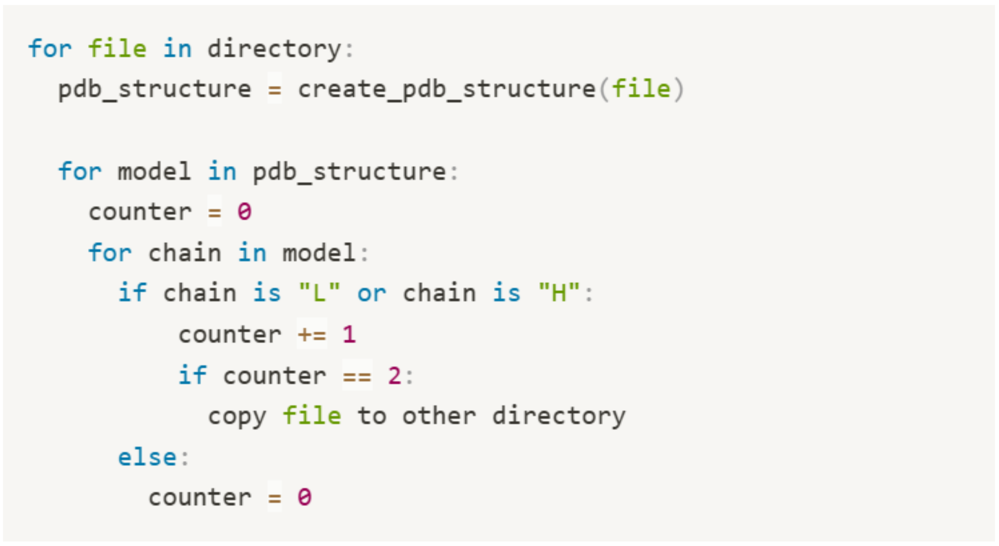
The new directory now contains all the files fulfilling the earlier requirements. To find the NTD files, either run the previous script again, with a list of the NTD structures, or just manually find them since there are fewer files.

After the PDB files are split between the RBD and the NTD, they need to be examined again to see if they fulfil the requirements given by the DistanceBetweenStructures encoder. The encoder currently has these requirements:

- Must contain a light and a heavy chain.
- The light and heavy chain must be named “L”, and “H” respectively.

There is no requirement of the PDB file to contain an antigen. Since the encoder will only accept chains named “L” and “H”, it makes no difference if the file contains an antigen or not, since it is removed either way. On the other hand, there will be issues if the antigen is named “L” or “H”.

To find the PDB files that fulfil these requirements, another python script was made.



```
for file in directory:
  pdb_structure = create_pdb_structure(file)

  for model in pdb_structure:
    counter = 0
    for chain in model:
      if chain is "L" or chain is "H":
          counter += 1
          if counter == 2:
            copy file to other directory
      else:
        counter = 0
```

Figure 19: Pseudo code of the script used for finding the PDB files fulfilling the encoder requirements. It is creating a PDB structure for every PDB file, and then iterating through the chains in each of the structures. The variable counter is used to ensure that the chains “L” and “H” are encountered in succession. There have been instances where there were more than two chains, and the “L” and “H” did not correspond to a light and a heavy chain.

After running the python script, there are 25 NTD PDB files and 160 RBD PDB files, making it a total of 185 files ready to be used for this machine learning use case.

4.5.1 Creating the metadata file

The PDBDataset requires that the PDB files are accompanied by a metadata file. Creating a metadata file is rather simple and can be written by the user themselves for smaller datasets. However, this becomes challenging when working with larger datasets.

For the DistanceToAntigenMatrixEncoder the metadata file is very simple. It only needs the column “filenames”, and then list each of the PDB files under.

When it comes to creating a metadata file for the DistanceBetweenStructures encoder, it is recommended to create a python script. This encoder usually requires larger datasets, which is why it might be tedious for the user to manually create the metadata file.
The metadata file for this encoder requires the columns “filename” and one that represents the

label, in this case the label is RBD or NTD. After running the previous scripts, the PDB files are split between two folders, one containing the NTD files and one containing the RBD files. There are multiple ways to generate a metadata file for a new combined dataset.
One option is to first generate a metadata file for each of the folders, and then merge the two metadata files together with pandas and output it to csv. Then move the PDB files to one large folder. Remember that the operating system usually sorts the folder alphabetically, which means that the metadata also needs to be sorted alphabetically. This is easily done with panda's utility functions. The appendix section contains some of the python scripts made for handling the PDB files.

With a suitable dataset and a metadata file, immuneML is ready to start its first run on 3D structures.

5 Results

5.1 Classes created

immuneML now has the necessary classes for providing functionalities on 3D data. The supported 3D data format is the PDB[2] file format. immuneML can read PDB files and create a PDBDataset object that stores the data contained in a PDB file. It can also utilize two different encodings, one machine learning algorithm (Precomputed KNN), and one report function on 3D data.
These classes have been implemented to support 3D structure data:

Class	**What it does**
PDBImport	• Gathers the user provided PDB files to a list. • Gets the user provided label from the metadata file. • Creates a PDBDataset object.
PDBDataset	• Stores information about the dataset. • Checks if the PDB files are IMGT numbered or not. • Generates a PDBStructure object for each of the PDB files.
PDBStructure	• Stores the data in the PDB file. • Stores other relevant information

Table 6: Every class that has been created for supporting 3D structure files, and what the classes do.

These classes have been made to provide the distance to antigen heatmap functionality:

Class	What it does
DistanceToAntigenMatrixEncoder	• Automatic detection of the chains • Gathers all the carbon alphas in the user specified location of the structure (default is CDR3) • Calculates the distance between carbon alphas in the antibody to the carbon alphas in the antigen. • Returns a dataset object containing the 3D array with structures and their distances
DistanceToAntigenHeatmapReport	• Uses the 3D array with structures and their distances to create a heatmap for each structure. • Returns a list of heatmaps to be displayed in the html file.

Table 7: Classes created for the distance to antigen heatmap functionality, and what the classes do.

This class was made to make it possible to run the precomputed KNN machine learning algorithm on 3D data:

Class	**What it does**
DistanceBetweenStructuresEncoder	• Removes the antigen from the file and creates a new PDB file without it. • Calculates the TM-score between every structure. • Returns a distance matrix with all the TM-scores.

Table 8: Classes created for the distance to antigen heatmap functionality, and what the classes do.

With the created classes the user can now specify a PDBDataset in the YAML file. The format is closer to specifying an AIRR repertoire dataset than a receptor/sequence dataset. This is because the repertoire data format is more like the PDB data format than the receptor/sequence data format. Even though the data inside the receptor/sequence files are closer to the data inside the PDB files. The PDB data format consists of a folder with many PDB structures, the same way the repertoire data format consists of a folder of repertoires. They both also require a metadata file. The metadata file needs to have a column that contains the PDB filenames. The DistanceBetweenStructuresEncoder also requires a label column in the metadata file.

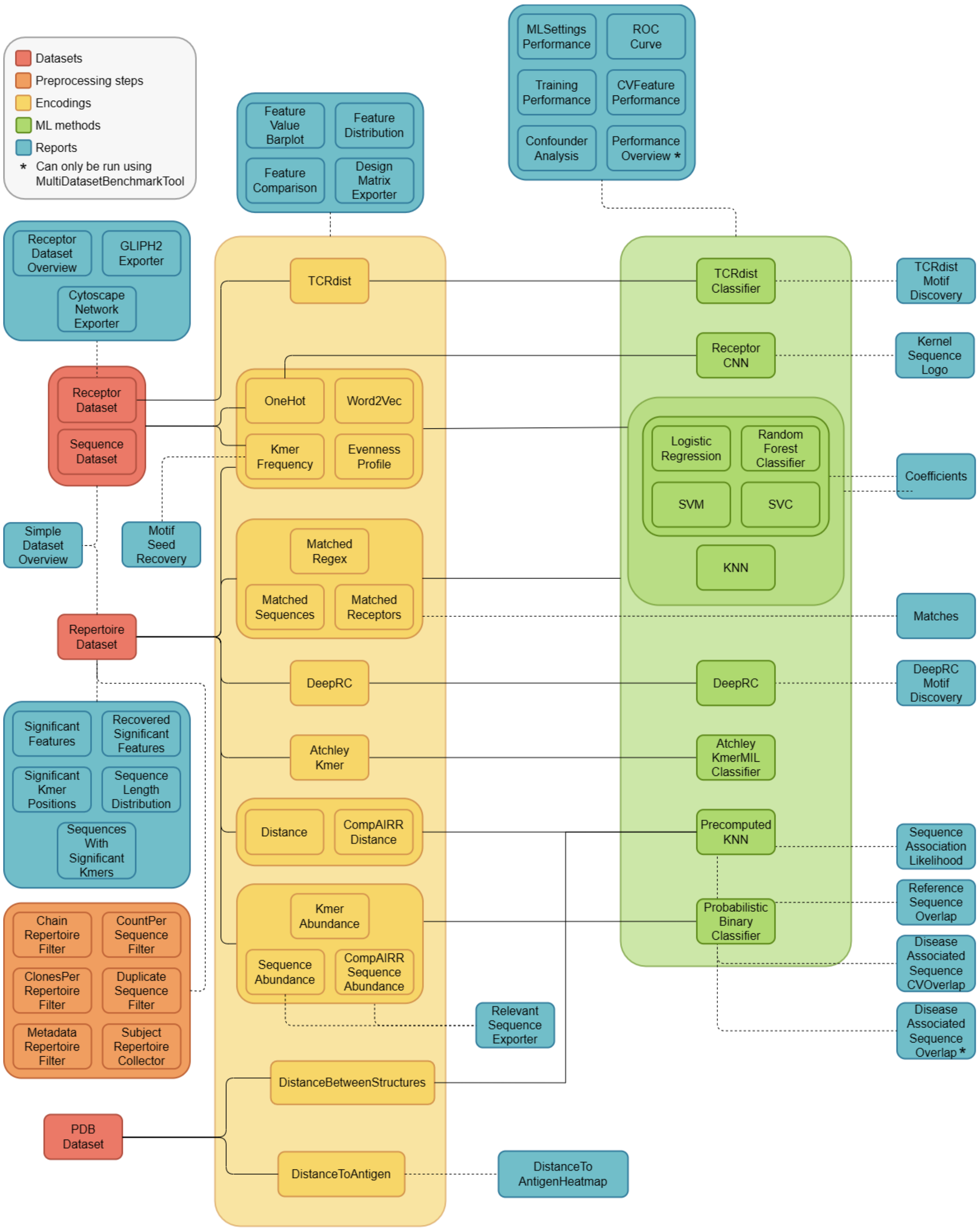


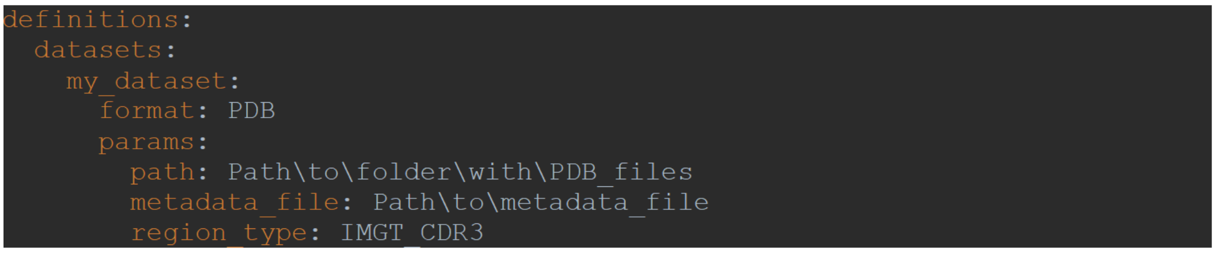
Figure 20: This figure shows all the functionalities that immuneML can provide. Updated with the new PDB functionalities.

5.2 Starting a run with the new classes

5.2.1 Generating heatmaps with distances to the antigen

To define a PDB dataset, simply write “PDB” as format in the dataset section.
The dataset section requires the user to specify the path to the folder of PDB files, and the path to the metadata file. In addition, the user can specify a specific region with the region_type variable. The possible values for region_type are: IMGT_CDR1 -> IMGT_CDR3, and FULL_SEQUENCE. Currently the region_type variable is unused by the other classes. The only class that has region slicing capabilities is the DistanceToAntigenMatrixEncoder, which has its own parameter for specifying. This is because the region shouldn’t be locked by the dataset, but rather the encoding which is the class performing actions on the dataset.

This is the format for defining a PDB dataset:



```
definitions:
  datasets:
    my_dataset:
      format: PDB
      params:
        path: Path\to\folder\with\PDB_files
        metadata_file: Path\to\metadata_file
        region_type: IMGT_CDR3
```

Figure 21: Example of how to define a PDB dataset. The required params are path and metadata_file path. Region_type is a parameter that the user can use to specify the region of the protein to be used. Here it is set to IMGT_CDR3, which is the default value. The region_type parameter in the Dataset is not currently in use by any classes.

To generate a distance to antigen heatmap, the user needs to first ensure that the PDB files are acceptable for the encoder. This encoder requires that the PDB file contains a light chain, a heavy chain, and an antigen.

The DistanceToAntigenMatrix encoding can be defined like this:

```yaml
encodings:
  my_encoding:
    DistanceToAntigenMatrix:
      region_type: FULL_SEQUENCE
```

Figure 22: Example of how to define the encoding “DistanceToAntigenMatrix”, with the chosen region “FULL_SEQUENCE”

The user can select a region_type for which region of the protein it should collect carbon alphas from. If region_type is left blank by the user, it is assigned to IMGT_CDR3. FULL_SEQUENCE means that the entire structure is being used for the encoding.

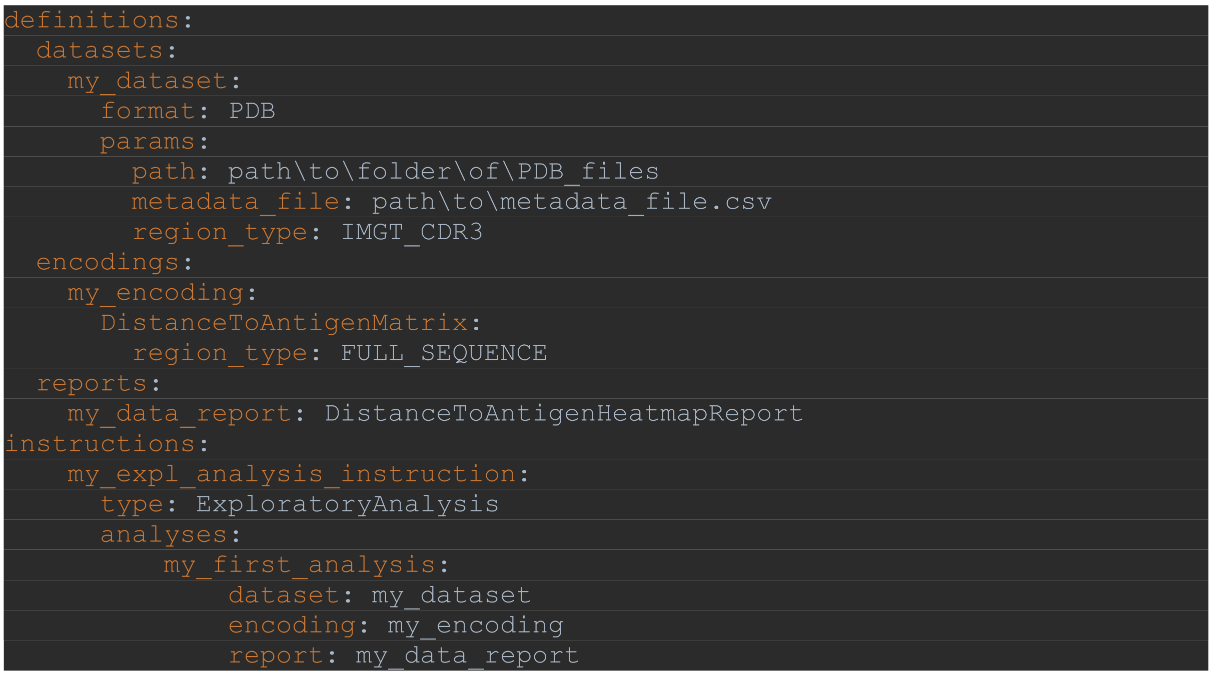
After defining the encoding, the user needs to define the report, which is the DistanceToAntigenHeatmapReport. This is the end of the definition section of the YAML file. The user should end up with a YAML file close to the one in figure 23.

```yaml
definitions:
  datasets:
    my_dataset:
      format: PDB
      params:
        path: path\to\folder\of\PDB_files
        metadata_file: path\to\metadata_file.csv
        region_type: IMGT_CDR3
  encodings:
    my_encoding:
      DistanceToAntigenMatrix:
        region_type: FULL_SEQUENCE
  reports:
    my_data_report: DistanceToAntigenHeatmapReport
```

Figure 23: The definition section of a distance to antigen heatmap generation run in immuneML.

After the definition section is the instruction section. This section the user needs to specify what should be used and with what.

The distance to antigen heatmap generation is an immuneML ExploratoryAnalysis functionality, which means that no machine learning will be used. To run an exploratory analysis the user simply needs to specify it under “type”. Then specify which dataset, encoding and report to use. See figure 24 for the entire YAML file specifying this run.



```yaml
definitions:
  datasets:
    my_dataset:
      format: PDB
      params:
        path: path\to\folder\of\PDB_files
        metadata_file: path\to\metadata_file.csv
        region_type: IMGT_CDR3
  encodings:
    my_encoding:
      DistanceToAntigenMatrix:
        region_type: FULL_SEQUENCE
  reports:
    my_data_report: DistanceToAntigenHeatmapReport
instructions:
    my_expl_analysis_instruction:
      type: ExploratoryAnalysis
      analyses:
          my_first_analysis:
              dataset: my_dataset
              encoding: my_encoding
              report: my_data_report
```

Figure 24: This YAML file specifies a DistanceToAntigenHeatmapReport with the encoding DistanceToAntigenMatrix on PDB data. The region_type in the encoding is set to FULL_SEQUENCE

immuneML will then generate heatmaps with the distances from the light and the heavy chain to the antigen. See figure 25 and 26 for example heatmaps generated with region selected as FULL_SEQUENCE, and figure 27, and 28 for example heatmaps with region IMGT_CDR3. The x-axis is the amino acid number in the light/heavy chain, and the y-axis is the amino acid number in the antigen.

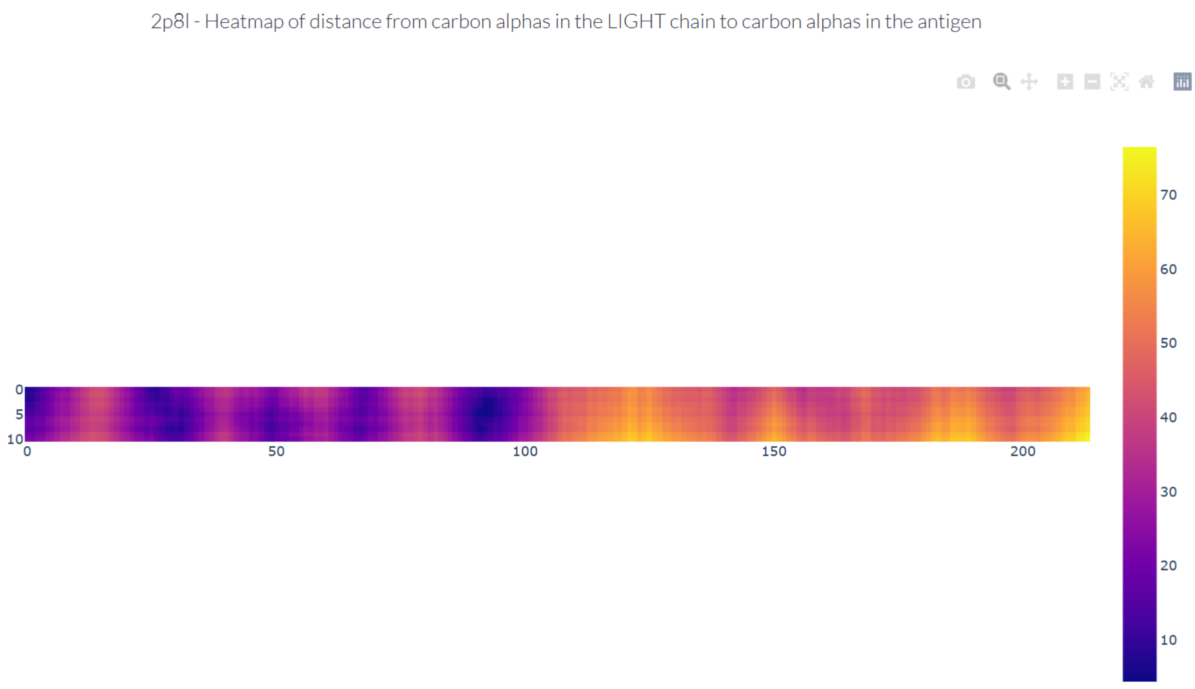


Figure 25: This is the heatmap generated with the distances from the carbon alphas in light chain, to the carbon alphas in the antigen. The carbon alphas are closest in the blue areas. This is from the PDB structure “2p8l”.

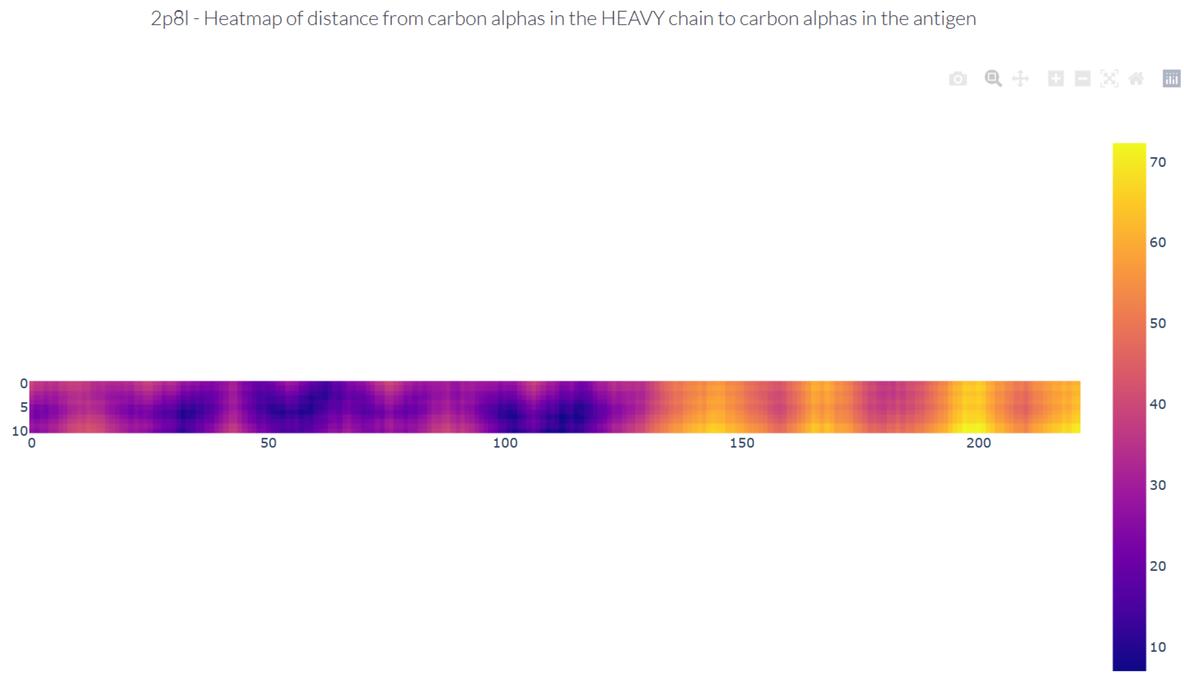


Figure 26: This is generated from the same PDB structure as the figure above, but with the distances from the heavy chain instead of the light.

2p8l - Heatmap of distance from carbon alphas in the LIGHT chain to carbon alphas in the antigen

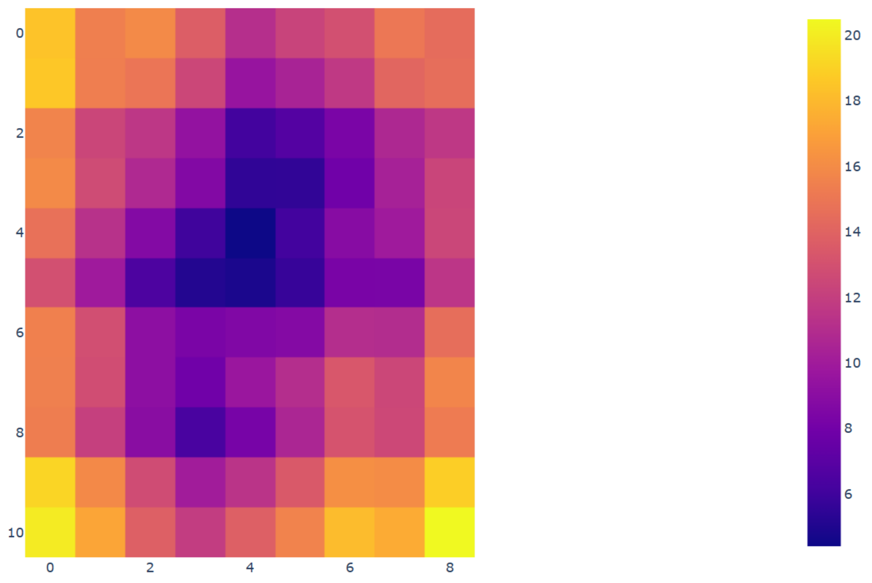


Figure 27: Heatmap generated with the distances from the carbon alphas in the CDR3 region in the light chain, to the antigen in the PDB file “2p8l”.

2p8l - Heatmap of distance from carbon alphas in the HEAVY chain to carbon alphas in the antigen

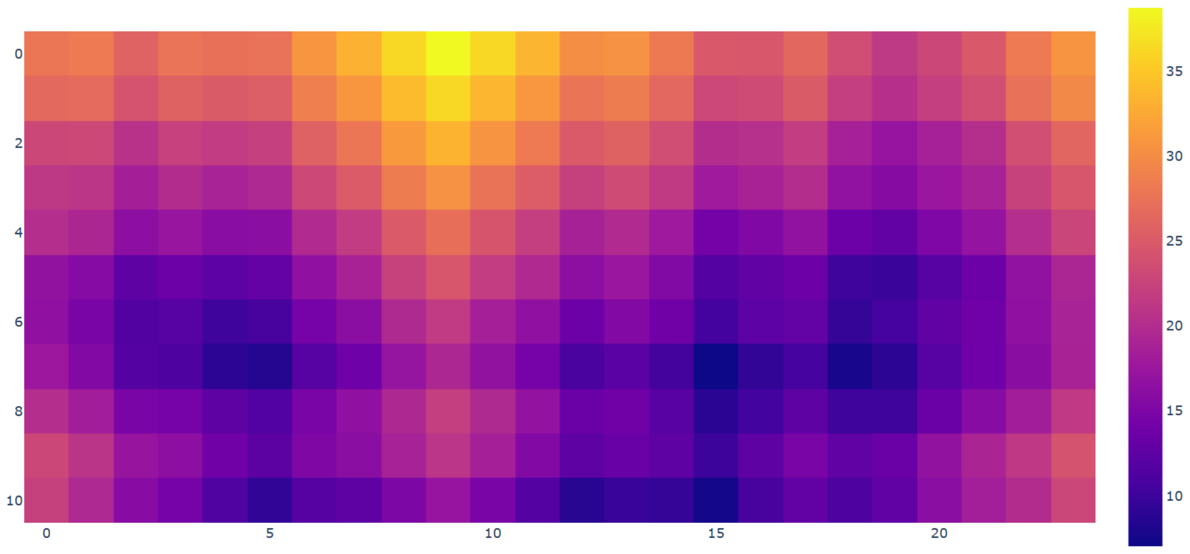


Figure 28: Heatmap generated with the distances from the carbon alphas in the CDR3 region in the heavy chain, to the antigen in the PDB file “2p8l”.

The generated heatmaps show that there are certain areas in the light and heavy chains that are closer to the antigen. This is especially clear from the heatmaps with FULL_SEQUENCE as selected region. The areas that are closer to the antigen correspond with the IMGT numbering scheme of the location of the CDRs. The CDRs is where the antigen binding is located [46].
The IMGT numbering scheme places CDR1 at 27-38, CDR2 at 56-65, and CDR3 at 105-117.
Figure 26 have the lowest distance around these locations as well.

5.2.2 Starting a precomputed KNN run on a PDB dataset

Specifying and starting an immuneML TrainMLModel instruction on a PDB dataset is quite simple. All the user needs to do is select the PDB dataset, the encoding DistanceBetweenStructures, and PrecomputedKNN as ML method. See figure 28 for the complete YAML file.

During this project it has been trained three machine learning models, with different sizes of PDB datasets.
The first one is named knn_quickstart. This model was trained on a small dataset of 24 PDB files. Where there were 12 RBD files, and 12 NTD files. The runtime for training this machine learning model was around 15 minutes.
The first model ended up with these metric scores:

Metric	Score
Balanced accuracy	0.733
AUC (Area under ROC curve)	0.833
Precision	0.667
Recall	0.667

Table 9: The performance of the knn_quickstart machine learning model with the metrics balanced accuracy, auc, precision, and recall.

The model ended up with a decent balanced accuracy and auc score. However, the precision and recall were not that impressive.

The second one is named knn_run_normal. This model was trained on a medium data set of 50 PDB files, which were divided into 25 RBD files and 25 NTD files. The runtime for training this machine learning model was around 45 minutes.
The second model ended up with these metric scores:

Metric	Score
Balanced accuracy	0.611
AUC (Area under ROC curve)	0.768
Precision	0.462
Recall	1

Table 10: The performance of the knn_run_normal machine learning model with the metrics balanced accuracy, auc, precision, and recall.

The model has quite poor balanced accuracy and precision, but a decent auc score. A recall score of 1 means that the model correctly identified all the positives.

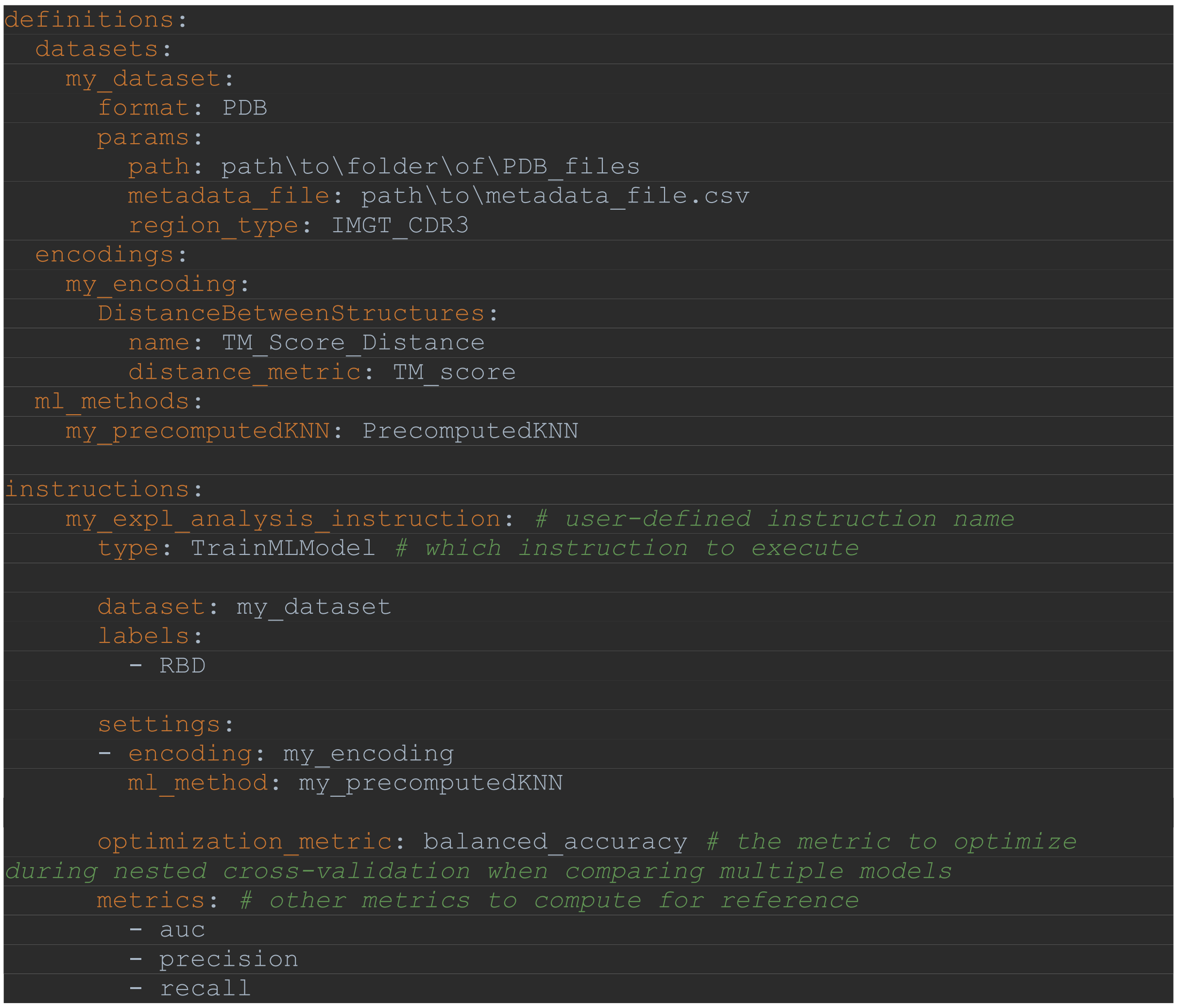
The third and last one is named knn_large_run. This model was trained on a large dataset of 185 PDB files, which were divided into 160 RBD files and 25 NTD files. There is a significant imbalance in the data between the two categories. This is caused by the lack of NTD files. The runtime for training this machine learning model was around 8 hours.
The third model ended up with these metric scores:

Metric	Score
Balanced accuracy	0.58
AUC (Area under ROC curve)	0.588
Precision	0.925
Recall	0.961

Table 11: The performance of the knn_large_run machine learning model with the metrics balanced accuracy, auc, precision, and recall.

The balanced accuracy and auc from the third model are quite poor. While the precision and recall score are quite high. This was expected as the dataset was heavily imbalanced from the beginning.

In the end it is important to remember that the main goal of this project was to add 3D structure support to immuneML, and not well performing machine learning models.



```
definitions:
  datasets:
    my_dataset:
      format: PDB
      params:
        path: path\to\folder\of\PDB_files
        metadata_file: path\to\metadata_file.csv
        region_type: IMGT_CDR3
  encodings:
    my_encoding:
      DistanceBetweenStructures:
        name: TM_Score_Distance
        distance_metric: TM_score
  ml_methods:
    my_precomputedKNN: PrecomputedKNN

instructions:
    my_expl_analysis_instruction: # user-defined instruction name
      type: TrainMLModel # which instruction to execute

      dataset: my_dataset
      labels:
        - RBD

      settings:
      - encoding: my_encoding
        ml_method: my_precomputedKNN

      optimization_metric: balanced_accuracy # the metric to optimize
during nested cross-validation when comparing multiple models
      metrics: # other metrics to compute for reference
        - auc
        - precision
        - recall
```

Figure 29: A complete YAML file for running the encoding DistanceBetweenStructures with the ML method PrecomputedKNN on a PDB dataset.

All of the outputs can be found here:

https://drive.google.com/drive/folders/1pZ9Lk1HGH-Qo5scQaBJki8ShZIzQfYht

6 Discussion

6.1 Results from this project

PDBDataset

immuneML can now take PDB files as input and run some analyses with them. Although the functionalities do not provide an extensive analysis of the immune receptors, it still proves that immuneML can handle 3D structure files.

DistanceBetweenStructures

Using the distance between structures encoder with precomputed KNN classification produces a machine learning model with an average balanced accuracy of 0.64, which is not a great score. The result from the classification run shows that it is not substantially capable to correctly classify the two groups of 3D structures.

DistanceToAntigenMatrixEncoder + HeatmapReport

The DistatanceToAntigenMatroxEncoder used with the heatmap report produces heatmaps, that usually show the light and the heavy chain closer to the antigen in the CDRs. Which is expected since that is where the binding is located.
Ideally the colours in the heatmap should be inverted. Warmer colours should be used when the distances are short, and colder colours should be used when the distances are far. But this proved harder to implement than anticipated on heatmaps in Plotly express.

6.2 PDB files and numbering scheme

The KNN classification run could be improved with larger amounts of suitable data. Suitable data are any protein structures that can be grouped together based on their properties. For example, a classification run with immune receptors that bind to a specific antigen vs immune receptors that don’t bind to it, would require many PDB files of different antibodies that are known to bind to the same antigen. It would also require research into if the non-binders actually don’t bind to the specified antigen, since there are cases where some immune receptors can bind to multiple[47]. This would be a better dataset to run machine learning on, compared to the one that was used in this project, since it might learn what features determine that it binds to the antigen.

Normally PDB files are created by authors when they are researching a specific topic or structure. The research is usually on the binding between the antibody and the antigen. This causes the databanks to be filled with single antibody-antigen bindings, if other antibodies bind to the same antigen is not known since it is not tested.

In numbers alone it can seem like there are a lot of PDB files of immune receptors available for machine learning, but that is unfortunately not the case. These files can be useful when conducting research into a specific antibody-antigen binding, but for machine learning the data is not enough. Only recently there has been a large amount of PDB files gathered related to the same antigen. This was caused by the Sars-Cov2 pandemic, which many started researching and creating PDB structures. These files all have the same protein spike, but possibly different bindings.

The PDB format itself is also not extensive enough to accurately display the information in the antibody-antigen binding structures. Usually when working with immune receptors the focus is on the CDRs [48], this is the section that contains the binding to the antigen. While the PDB format contains information about the entire structure, it is insufficient for storing all the details of the variable regions in an antibody. This is one the reason there have been created many numbering schemes that are specifically made for storing antibody variable regions.

Dondelinger et al. (2018) [49] explains the difference between the numbering systems Kabat[50], Chotia[51], Martin[52] (Based on Chotia numbering), and IMGT[53].
Kabat is a numbering scheme standard for antibodies, specifically sequence based[52] ones. The Kabat scheme is widely used, but there are some limitations.
Firstly, it ignores antibody chains that has unconventional lengths[49], or chains that has been altered with insertions[49] or deletions[49].
Secondly, the scheme doesn’t translate the 3D structure well enough [49]. There are also regions defined in the Kabat scheme that do not exactly match the binding regions [49].

While Kabat is a sequence based numbering scheme, Chothia is a structure based[52] one, which also brings the advantage that topologically aligned residues across different antibodies are assigned the same position number. [49] Additionally, the CDR definition in the Chothia scheme closely matches the structural antigen-binding loops in most antibody sequences. This scheme also has the same limitation as Kabat, where it ignores antibody chains of unconventional lengths.
The Martin numbering scheme makes changes to the Chotia numbering scheme[52], by changing the cut offs for designating amino acids as framework region and the CDRs[52]. These changes were made based on analyses of large amount sequences and structures [52]. The Martin numbering scheme can be viewed as an updated version of the Chotia scheme [49].

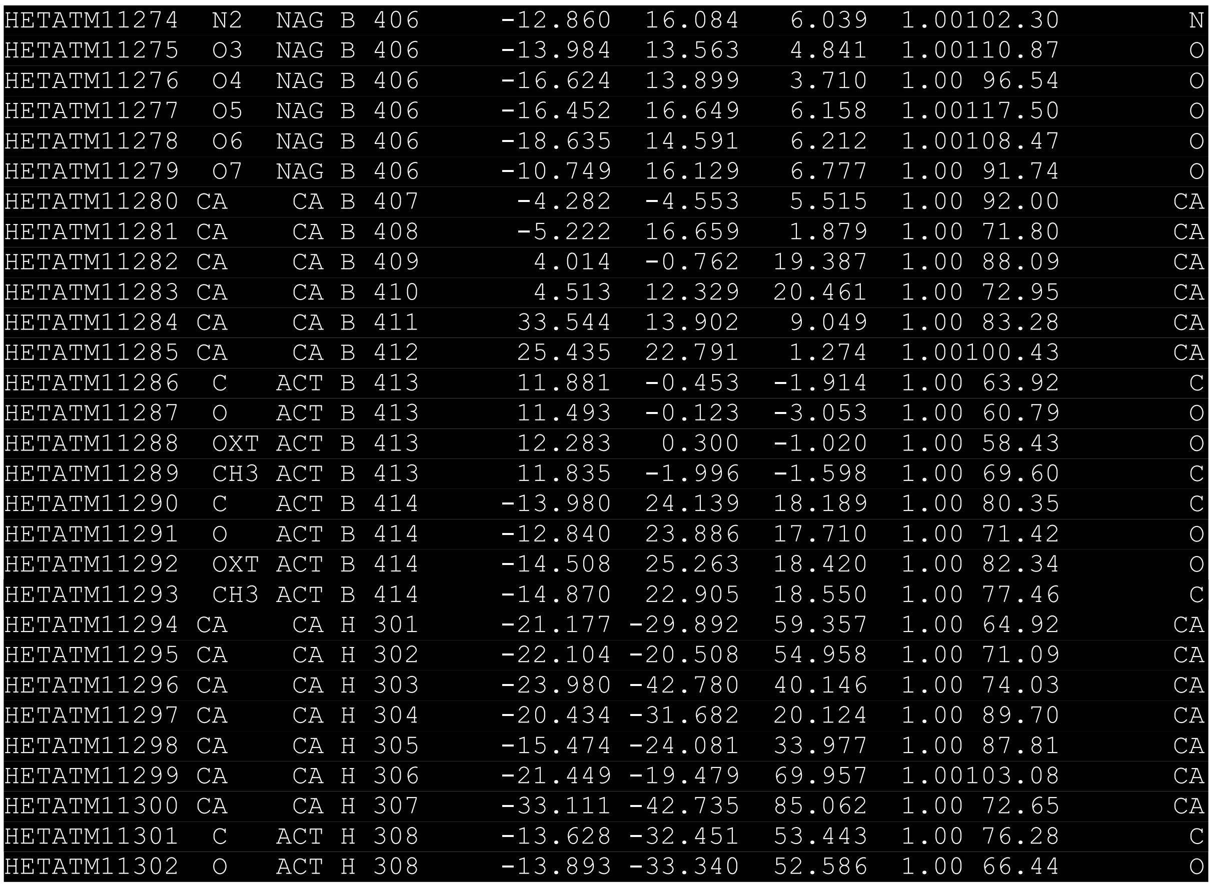
IMGT is a numbering scheme that was made for numbering protein sequences of antibodies, as well as their variable domains and chains [49]. The numbering scheme is based on the amino acid sequence alignment of the germ-line V gene [49]. The germ-line V is significant for antibody research because it is located in the CDRs [54]. IMGT uses the germ-line V as an alignment and then counts residues from position 1 to 128 [49].
IMGT is a primary reference for work in immunogenetics and immune-informatics [49], and is currently in use[49] and endorsed[55] by the World Health Organization - International Union of Immunological Societies.
A drawback with IMGT is that it is quite rigid with its structure, in the sense that it does not allow insertions directly in the structure, but rather places them at the end [49].

Each of these numbering schemes also have different definitions of the CDR lengths. Where one CDR ends and where the next one begins are different between the mentioned numbering schemes. Kabat's and IMGT's definition of the CDRs are based on sequence alignment [49]. Chotia, on the other hand, is based on the actual structure of the loops in the antibody [49].

What each numbering system defines as CDR might have an impact on how well machine learning algorithms perform on the data. This is something that needs to be taken into consideration when selecting a numbering scheme for usage.
The three mentioned numbering schemes are also not built on the PDB file format, but they are often used together. The numbering scheme is simply a standard for deciding which amino acid gets which position number. Combined with the PDB file format, additional information can be stored about each of the atoms, such as coordinates and position in the amino acid.
In this project the IMGT numbering scheme was used, but the other mentioned numbering schemes could perform better. immuneML will benefit from supporting all of the mentioned numbering schemes. Especially since every one of them have their own way of defining the CDRs. By supporting multiple numbering schemes, the user can compare and see which performs best.

The PDB file format is a great file format for storing biological 3D structures. However, there aren't strict rules for what to name the different chains. There are guidelines that many follow, but no requirement. Although the light and heavy chain are usually named L and H respectively, there are instances where they have been named A and B as well. This is especially confusing since the alpha and beta chains in the T-cell receptors are usually named A and B. Even though every PDB file has a description in the beginning explaining what each chain is, it might still be confusing. The author's description might not be sufficient for a complete understanding of the different chains, which might cause misunderstandings. Since there can be a high variation of PDB files, there have been created tools that handle parsing. The PDB parser [34]module by Biopython[36] is one such tool. This module parses information in the PDB file and stores it in a python object called PDB.Structure [37].
In the work of adding 3D structure support to immuneML, the PDB Parser is a central piece. It handles all the parsing of the PDB files, and the PDB.Structure object is the object accessed for data in the use of encodings. immuneML now uses numerous classes from the PDB package [35], which is a sub-package of the main Bio[36] package. These tools by Biopython are

updated regularly, providing further functionality and fixes for bugs, and issues. Throughout this project, there have been PDB files that are formatted so differently, that even the PDB Parser couldn’t handle it. Or the PDB Parser did handle it, but the object created doesn’t follow the expected structure. Every one of these different formatted PDB files are an edge case that needs to be dealt with, either by catching the error, rectifying it, or simply discarding the PDB file. See figure 30 for an example of a PDB file where the PDB Parser encountered a file it couldn’t parse.



```
HETATM11274  N2  NAG B 406     -12.860  16.084   6.039  1.00102.30           N
HETATM11275  O3  NAG B 406     -13.984  13.563   4.841  1.00110.87           O
HETATM11276  O4  NAG B 406     -16.624  13.899   3.710  1.00 96.54           O
HETATM11277  O5  NAG B 406     -16.452  16.649   6.158  1.00117.50           O
HETATM11278  O6  NAG B 406     -18.635  14.591   6.212  1.00108.47           O
HETATM11279  O7  NAG B 406     -10.749  16.129   6.777  1.00 91.74           O
HETATM11280 CA    CA B 407      -4.282  -4.553   5.515  1.00 92.00          CA
HETATM11281 CA    CA B 408      -5.222  16.659   1.879  1.00 71.80          CA
HETATM11282 CA    CA B 409       4.014  -0.762  19.387  1.00 88.09          CA
HETATM11283 CA    CA B 410       4.513  12.329  20.461  1.00 72.95          CA
HETATM11284 CA    CA B 411      33.544  13.902   9.049  1.00 83.28          CA
HETATM11285 CA    CA B 412      25.435  22.791   1.274  1.00100.43          CA
HETATM11286  C   ACT B 413      11.881  -0.453  -1.914  1.00 63.92           C
HETATM11287  O   ACT B 413      11.493  -0.123  -3.053  1.00 60.79           O
HETATM11288  OXT ACT B 413      12.283   0.300  -1.020  1.00 58.43           O
HETATM11289  CH3 ACT B 413      11.835  -1.996  -1.598  1.00 69.60           C
HETATM11290  C   ACT B 414     -13.980  24.139  18.189  1.00 80.35           C
HETATM11291  O   ACT B 414     -12.840  23.886  17.710  1.00 71.42           O
HETATM11292  OXT ACT B 414     -14.508  25.263  18.420  1.00 82.34           O
HETATM11293  CH3 ACT B 414     -14.870  22.905  18.550  1.00 77.46           C
HETATM11294 CA    CA H 301     -21.177 -29.892  59.357  1.00 64.92          CA
HETATM11295 CA    CA H 302     -22.104 -20.508  54.958  1.00 71.09          CA
HETATM11296 CA    CA H 303     -23.980 -42.780  40.146  1.00 74.03          CA
HETATM11297 CA    CA H 304     -20.434 -31.682  20.124  1.00 89.70          CA
HETATM11298 CA    CA H 305     -15.474 -24.081  33.977  1.00 87.81          CA
HETATM11299 CA    CA H 306     -21.449 -19.479  69.957  1.00103.08          CA
HETATM11300 CA    CA H 307     -33.111 -42.735  85.062  1.00 72.65          CA
HETATM11301  C   ACT H 308     -13.628 -32.451  53.443  1.00 76.28           C
HETATM11302  O   ACT H 308     -13.893 -33.340  52.586  1.00 66.44           O
```

Figure 30: This is an excerpt from the PDB file “7L2C.pdb”, which the PDB parser couldn’t handle. The issue is in the HETATM section, where the author added “CA” to the carbon alphas in the far-right column. This column is supposed to list the element symbol of the atom, which for carbon is “C”. The author has listed it as “CA”, which causes the PDB Parser to stop with exceptions.

6.3 Other related 3D structure methods

Polychronidou et al (2018) [42] shows one method for clustering a set of proteins based on their 3D structure. In this paper they also used TM-align for alignment of the 3D structures[42]. After aligning them with TM-align, they extracted several different 3D descriptors from them[42]. 3D descriptors are properties that describe the conformation of the structure [56]. The 3D descriptors were then used for calculating the distance between the structures. The chosen distance metric was RMSD (Root Mean Square Deviation).
With the RMSD score they started clustering the data. There were three clustering methods used: k-medoids, hierarchical agglomerative, and DBScan.

The k-medoids clustering method is quite like the k-means method, but instead of using vectorial representations of the objects, it uses a distance matrix as input [42]. The k-means algorithm guesses the cluster centre positions within the feature space and updates them when the available data provides a better fit. In contrast, the k-medoids method does not define a feature space and has no concept of cluster centre positions. Instead, it uses objects to represent the set of objects, called medoids, to act as the cluster centre. Initially the medoids are set randomly but are updated based on their similarity to other objects in the same cluster, and their dissimilarity to objects in other clusters.

The hierarchical agglomerative clustering algorithm is a method that starts with every data point being a cluster itself. Then merges two and two clusters together based on their distance. The distance in this paper[42] is the mean value between each cluster. The algorithm stops when all the clusters are merged into one large cluster. At the end there is one large cluster, but the main advantage from this algorithm is the tree-like structure that forms after completing each step/merge. This tree-like structure can be cut at different heights to show the clustering at different stages.

The last clustering algorithm is DBScan which is a density-based clustering method[42]. This algorithm works by selecting a random data point and if there are N data points close, it is considered dense, then forming a cluster[42]. N is the lowest number of data points to be classified as dense. The algorithm repeats this action until there are no points that can be considered as close to a cluster.

Utilizing these three machine learning algorithms on the combined 3D descriptor, the Rand index ended up at 89.8% for clustering labelled data to six, and 92.2% for optimal numbers of clusters.[42] These results are promising for research using 3D structures and should be further explored.

immuneML could benefit from recreating the methods used in Polychronidou et al (2018) [42]. Especially the part with the usage of 3D descriptors instead of the raw PDB file. In addition, implementing RMSD could also prove beneficial for immuneML.

7 Conclusion

immuneML is a machine learning platform for analysis of adaptive immune receptor repertoires. The main objective of this project was to add 3D structure support to immuneML. This has been achieved with the implementation of the PDBDataset and the PDBStructure classes. These classes use BioPython's PDB package, modules such as PDB Parser, and the class Structure, are used to read and store the PDB files. In addition, the users are now able to run exploratory analysis and machine learning on 3D structures.
The main challenge for training machine learning models on 3D structures, is the lack of larger datasets of suitable data with the same properties. Classification algorithms require the data to be split into different categories, usually based on their properties, for example if the antibody binds to a specific antigen or that it does not bind to a specific antigen. In the future when more of such data is available, immuneML will already have support for them.

7.1 Further work

7.1.1 Improvements to the PDBDataset

Here are some ways forward for improving the PDBDataset:

- Currently PDBDatasets only support BCRs, which means that implementing TCR support is a clear way forward.
- Implement support for multiple numbering schemes in addition to IMGT, such as Chotia, Martin and Kabat. These numbering schemes also have different definitions of CDR, which could give new perspectives of the data, or better performance.

7.1.2 DistanceToAntigen

The DistanceToAntigen encoder could be improved with better detection of chains. The current implementation works for most of the PDB files, that fulfil the requirements specified. This can be further improved by implementing a dynamic detection of the chains based on all the possible names of chains. For example, if the chain “A” and “B” are detected, that should mean that “A” is the light chain, and “B” is the heavy chain. Thus, the last chain will be the antigen. But if “A” is detected with no “B”, it probably means that the antigen is named “A” and the other two chains are the light and heavy chain.

Another approach could be to let the user specify the chains themselves in the metadata file. This approach would fix all of the issues with the detection of chains but will be tedious for the user.

A small improvement could be to reverse the colours in the heatmap. Closer distances will have a warmer colour, while longer distances will have a colder one. Adding titles to the axis’ will also be an improvement.

7.1.3 DistanceBetweenStructures

The DistanceBetweenStructures encoder could be improved with implementation of the 3D structure descriptors that were used in Polychronidou et al (2018) [42]. Using 3D structure descriptors instead of the PDB file itself, could prove beneficial in clustering performance. Other ways to improve the encoder:

- Implementation of multiple numbering schemes gives the user the possibility to select one that is best suited for the user’s situation.

- Optimization of code will improve the runtime. Currently it is quite long for larger datasets, which should be decreased.

Literature references

1. Breden, F. *et al.* Reproducibility and Reuse of Adaptive Immune Receptor Repertoire Data. *Front. Immunol.* **8**, (2017).
2. Berman, H. M. *et al.* The Protein Data Bank. *Nucleic Acids Res.* **28**, 235–242 (2000).
3. Jasim, S. A. *et al.* The deciphering of the immune cells and marker signature in COVID-19 pathogenesis: An update. *J. Med. Virol.* **94**, 5128–5148 (2022).
4. Ye, C., Hu, W. & Gaeta, B. Prediction of Antibody-Antigen Binding via Machine Learning: Development of Data Sets and Evaluation of Methods. *JMIR Bioinform Biotech* **3**, e29404 (2022).
5. Mane, S. U. & Pangu, K. H. Disease Diagnosis Using Pattern Matching Algorithm from DNA Sequencing: A Sequential and GPGPU Based Approach. in *Proceedings of the International Conference on Informatics and Analytics* (Association for Computing Machinery, 2016). doi:10.1145/2980258.2980392.
6. Ayyadevara, V. *Pro Machine Learning Algorithms*. (2018). doi:10.1007/978-1-4842-3564-5.
7. Pavlović, M. *et al.* The immuneML ecosystem for machine learning analysis of adaptive immune receptor repertoires. *Nat. Mach. Intell.* **3**, 936–944 (2021).
8. Punt, J., Stranford, S. A., Jones, P. P. & Owen, J. A. *Kuby immunology*. (Macmillan Education, 2019).
9. *The innate and adaptive immune systems*. *InformedHealth.org [Internet]* (Institute for Quality and Efficiency in Health Care (IQWiG), 2020).
10. Alcover, A., Alarcón, B. & Di Bartolo, V. Cell Biology of T Cell Receptor Expression and Regulation. *Annu. Rev. Immunol.* **36**, 103–125 (2018).

11. Chapter 1.9 - The structure of the antibody molecule illustrates the central puzzle of adaptive immunity. in *Immunobiology: The Immune System in Health and Disease. 5th edition* (Garland Science, 2001).
12. Charles A Janeway, J., Travers, P., Walport, M. & Shlomchik, M. J. T-cell receptor gene rearrangement. *Immunobiol. Immune Syst. Health Dis. 5th Ed.* (2001).
13. Antibody Combining Site - an overview | ScienceDirect Topics. https://www.sciencedirect.com/topics/biochemistry-genetics-and-molecular-biology/antibody-combining-site.
14. Wong, W. K., Leem, J. & Deane, C. M. Comparative Analysis of the CDR Loops of Antigen Receptors. *Front. Immunol.* **10**, 2454 (2019).
15. Kuroda, D., Shirai, H., Kobori, M. & Nakamura, H. Systematic classification of CDR-L3 in antibodies: implications of the light chain subtypes and the VL-VH interface. *Proteins* **75**, 139–146 (2009).
16. Wang, Y. Identifying CDRs by Antibody Sequencing. *Rapid Novor* https://www.rapidnovor.com/identifying-cdrs-antibody-sequencing/ (2022).
17. Alberts, B. *et al. Molecular Biology of the Cell*. (Garland Science, 2002).
18. Nelson, P. N. *et al.* Demystified … Monoclonal antibodies. *Mol. Pathol.* **53**, 111–117 (2000).
19. Little, M., Kipriyanov, S. M., Le Gall, F. & Moldenhauer, G. Of mice and men: hybridoma and recombinant antibodies. *Immunol. Today* **21**, 364–370 (2000).
20. Maggon, K. Monoclonal antibody ‘gold rush’. *Curr. Med. Chem.* **14**, 1978–1987 (2007).
21. Monoclonal Antibody Therapy Market Size, Growth, Trends, 2028. https://www.fortunebusinessinsights.com/monoclonal-antibody-therapy-market-102734.

22. Urquhart, L. Top companies and drugs by sales in 2021. *Nat. Rev. Drug Discov.* **21**, 251–251 (2022).

23. HUMIRA® (adalimumab) | A Biologic Treatment Option. https://www.humira.com/.

24. Trück, J. *et al.* Biological controls for standardization and interpretation of adaptive immune receptor repertoire profiling. *eLife* **10**, e66274 (2021).

25. Vander Heiden, J. A. *et al.* AIRR Community Standardized Representations for Annotated Immune Repertoires. *Front. Immunol.* **9**, (2018).

26. Mason, D. M. *et al.* Optimization of therapeutic antibodies by predicting antigen specificity from antibody sequence via deep learning. *Nat. Biomed. Eng.* **5**, 600–612 (2021).

27. Akbar, R. *et al.* A compact vocabulary of paratope-epitope interactions enables predictability of antibody-antigen binding. *Cell Rep.* **34**, 108856 (2021).

28. Bank, R. P. D. RCSB PDB - 2DD8: Crystal Structure of SARS-CoV Spike Receptor-Binding Domain Complexed with Neutralizing Antibody. https://www.rcsb.org/structure/2DD8.

29. sklearn.neighbors.KNeighborsClassifier. *scikit-learn* https://scikit-learn/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html.

30. Parvandeh, S., Yeh, H.-W., Paulus, M. P. & McKinney, B. A. Consensus features nested cross-validation. *Bioinformatics* **36**, 3093–3098 (2020).

31. Szathmáry, E. Why are there four letters in the genetic alphabet? *Nat. Rev. Genet.* **4**, 995–1001 (2003).

32. Yet Another Markup Language (YAML) 1.0. https://yaml.org/spec/history/2001-05-26.html.

33. IMGT Education. https://www.imgt.org/IMGTeducation/Aide-memoire/_UK/aminoacids/formuleAA/#MLformula.
34. Bio.PDB.PDBParser module — Biopython 1.76 documentation. https://biopython.org/docs/1.76/api/Bio.PDB.PDBParser.html.
35. Bio.PDB package — Biopython 1.75 documentation. https://biopython.org/docs/1.75/api/Bio.PDB.html.
36. Cock, P. J. *et al.* Biopython: freely available Python tools for computational molecular biology and bioinformatics. *Bioinformatics* **25**, 1422–1423 (2009).
37. Bio.PDB.Structure module — Biopython 1.76 documentation. https://biopython.org/docs/1.76/api/Bio.PDB.Structure.html.
38. Lefranc, M.-P. *et al.* IMGT, the international ImMunoGeneTics information system. *Nucleic Acids Res.* **37**, D1006-1012 (2009).
39. Bio.PDB.PDBIO module — Biopython 1.75 documentation. https://biopython.org/docs/1.75/api/Bio.PDB.PDBIO.html?highlight=select#Bio.PDB.PDBIO.Select.
40. Kufareva, I. & Abagyan, R. Methods of Protein Structure Comparison. in *Homology Modeling: Methods and Protocols* (eds. Orry, A. J. W. & Abagyan, R.) 231–257 (Humana Press, 2012). doi:10.1007/978-1-61779-588-6_10.
41. Zhang, Y. & Skolnick, J. Scoring function for automated assessment of protein structure template quality. *Proteins Struct. Funct. Bioinforma.* **57**, 702–710 (2004).
42. Polychronidou, E. *et al.* Automated shape-based clustering of 3D immunoglobulin protein structures in chronic lymphocytic leukemia. *BMC Bioinformatics* **19**, 414 (2018).
43. Zhang, Y. & Skolnick, J. TM-align: a protein structure alignment algorithm based on the TM-score. *Nucleic Acids Res.* **33**, 2302–2309 (2005).

44. Xu, J. & Zhang, Y. How significant is a protein structure similarity with TM-score = 0.5? *Bioinformatics* **26**, 889–895 (2010).

45. Raybould, M. I. J., Kovaltsuk, A., Marks, C. & Deane, C. M. CoV-AbDab: the Coronavirus Antibody Database. *Bioinformatics* **37**, 734–735 (2021).

46. Kunik, V., Peters, B. & Ofran, Y. Structural Consensus among Antibodies Defines the Antigen Binding Site. *PLoS Comput. Biol.* **8**, e1002388 (2012).

47. Chapter 4 - Specificity and Cross-Reactivity. in *Immunology and Evolution of Infectious Disease* (Princeton University Press, 2002).

48. Xu, J. L. & Davis, M. M. Diversity in the CDR3 Region of VH Is Sufficient for Most Antibody Specificities. *Immunity* **13**, 37–45 (2000).

49. Dondelinger, M. *et al.* Understanding the Significance and Implications of Antibody Numbering and Antigen-Binding Surface/Residue Definition. *Front. Immunol.* **9**, 2278 (2018).

50. Kabat, E. A. *Sequences of Proteins of Immunological Interest*. (DIANE Publishing, 1992).

51. Chothia, C. & Lesk, A. M. Canonical structures for the hypervariable regions of immunoglobulins. *J. Mol. Biol.* **196**, 901–917 (1987).

52. Abhinandan, K. R. & Martin, A. C. R. Analysis and improvements to Kabat and structurally correct numbering of antibody variable domains. *Mol. Immunol.* **45**, 3832–3839 (2008).

53. Lefranc, M.-P. *et al.* IMGT unique numbering for immunoglobulin and T cell receptor variable domains and Ig superfamily V-like domains. *Dev. Comp. Immunol.* **27**, 55–77 (2003).

54. Chapter 5 - Organization and Expression of Immunoglobulin Genes. in *Kuby immunology* (Macmillan Education, 2019).

55. Lefranc, M.-P. *et al.* IMGT®, the international ImMunoGeneTics information system® 25 years on. *Nucleic Acids Res.* **43**, D413–D422 (2015).

56. Bergström, C. A. S. In silico Predictions of Drug Solubility and Permeability: Two Rate-limiting Barriers to Oral Drug Absorption. *Basic Clin. Pharmacol. Toxicol.* **96**, 156–161 (2005).

Appendix A – How to run immuneML with PDB functionalities

Easiest: Access the UiO VM that has been setup for this. (The tutorial is for Windows based systems)

Other: Install it on your own device.

Accessing the UiO virtual machine

Simply download the private key used for connecting to the VM from this link:

https://drive.google.com/file/d/1T75rE-XsHH6S83yrnE1QLqss-KviU75H/view?usp=sharing

If you have windows the command to connect to the VM is:

```
$ ssh -i nrec_vm debian@158.39.75.182
```

Gain root access with these commands:

```
[debian@testvm ~]$ sudo -i
[root@testvm ~]# whoami
root
```

If there are any issues, follow the guide here: https://uh-iaas.readthedocs.io/ssh.html#connecting-to-the-instance

IP of VM: 158.39.75.182

After gaining access to the VM use this command to go to the correct directory:

```
[root@testvm ~]# cd /usr/immuneML/immuneML/immuneML/Run_files
```

From here you can use these four commands. Remember to delete the "Output" folder between each run. The KNN classification run also generate new PDB files in the current directory, these files can be deleted but do not need to. They can easily be deleted with the command at the end of this guide.

1. Generate heatmaps of the distance between the chains and the antigen. 7 PDB structures (Fast runtime)

```
$ immune-ml heatmap.yaml Output
```

2. Starts a KNN classification run on the TM-score distance between 24 different PDB structures. 12 RBD and 12 NTD (Runtime is around 15 minutes)

```
$ immune-ml KNN_quickstart.yaml Output
```

3. Starts a KNN classification run on the TM-score distance between 50 different PDB structures. 25 RBD vs 25 NTD(Runtime is around 50 minutes)

```
$ immune-ml KNN_run_normal.yaml Output
```

Starts a KNN classification run on the TM-score distance between 185 different PDB structures. 160 RBD and 25 NTD (Runtime is around 9 hours)

```
$ immune-ml KNN_LARGE_run.yaml Output
```

The commands above will generate a folder called Output and store the results there.

If you want to see the results and open the html file, you need to send the results to your local computer. To do this, you need to zip the folder, and then open another terminal and request the zipped folder from the VM.

Use this command to zip the Output folder:

```
$ zip -r Results.zip Output
```

Then send the zipped folder with this command (Not the same terminal that is connected to the VM):

```
$ scp -i nrec_vm debian@158.39.75.182://usr/immuneML/immuneML/immuneML/Run_files/R
esults.zip Results.zip
```

The zipped folder should appear in your current directory.

How to delete the generated PDB files (Use in the current directory):

```
$ rm *.pdb
```

Installing immuneML with PDB support on your own device

You can install immuneML with PDB support by cloning this branch:

https://github.com/uio-bmi/immuneML/tree/pdb_support_final

Simply install the setup.py file with python

```
Then run:

$ pip install biopython
$ pip install tmtools
```

To check if immuneML is installed run:

```
$ immune-ml -h
```

Running immuneML with PDB files

The necessary files for running immuneML with PDB files are already provided in the branch. You are free to add or remove files as you see fit, but remember to update the metadata file to correctly match the data. The files are located in the “Run_files” folder.

You need to update the YAML files with the correct path to the metadata file and the folder of PDB files.

immuneML takes in two arguments. The first is the path to the YAML file, and the second is the path to the folder where the output will be stored. The arguments are separated by a single whitespace.
Remember to delete the “Output” folder between each run. The KNN classification runs also generate new PDB files in the current directory, these files can be deleted but does not need to.

The command for running the heatmap.yaml is (replace the path with your own):

```
$ immune-ml immuneML/Run_files/heatmap.yaml immuneML/Run_files/Output
```

1. Select which YAML file you want to run by replacing the path with the path to YAML file you want to run.

2. Run the command above with your choice and an output path.

There are currently four yaml files in this example branch.

The first is the heatmap.yaml. This file should generate heatmaps with the distance from the antigen to the different chains. There are in total 7 PDB files in this folder, which makes the total of heatmaps 14. The runtime is quite low for this run.

The second one is KNN_quickstart.yaml. This run uses the DistanceBetweenStructuresEncoder, and calculates the TM-score between the different structures. Then it runs the KNN classification algorithm on the distance matrix. There are in total 24 PDB files used for this run. The data is split between 12 RBD files, and 12 NTD files. The runtime is around 20 minutes for this run.

The second one is KNN_run_normal.yaml. This run is the same as the second one, but has a larger dataset. There are in total 50 PDB files used for this run, 25 RBD files and 25 NTD files. The runtime is around 40 minutes for this run.

The last one is KNN_LARGE_run. This is the same as the previous run, but here there are in total 185 PDB files. The dataset contains 25 NTD files and 160 RBD files. The runtime is around 8 hours.

Appendix B – Code

Code used to data filtration:

```
import os
from Bio.PDB import PDBParser
import shutil



directory = “Directory\of\files”



files_to_keep []


for filename in os.listdir(directory):
    pdb_parser = PDBParser(
        PERMISSIVE=True
    )

    pdb_structure = pdb_parser.get_structure("pdbStructure", directory +
"\\" + filename)

    for model in pdb_structure.get_models():
        counter = 0

        for chains in model:
            if str(chains) == "<Chain id=L>" or str(chains) == "<Chain
id=H>":
                counter = counter + 1
                if counter == 2:
                    files_to_keep(filename)
                    break

            else:
                counter = 0


print(len(files_to_keep))
for file in files_to_keep:
    shutil.copyfile(directory + "\\" + file, "new_folder\\" + file)
```

Code used for generating the metadata file

```python
import csv
import os

directory = 'RBD_PDB'

with open("metadata.csv", 'w', newline='') as file:
    writer = csv.writer(file)
    writer.writerow(["RBD","filename"])
    id_counter = 0
    for filename in os.listdir(directory):
        writer.writerow(["True",filename])
        id_counter = id_counter+1



    for filename in os.listdir("NTD_PDB"):
        writer.writerow(["False",filename])
        id_counter = id_counter+1
```