Biological Data Project

Lorenzo Caprini, Stefano Della Morte, Riccardo Minzoni

10/02/2021

1 Introduction

All the code mentioned in this report can be consulted at a Git repository hosted on GitHub[1]. The whole project can be reproduced with the main Python 3 Jupyter Notebook `project.ipynb` in the `code` folder. However, some parts are coded to be executed on a Linux x64 machine. More detailed instructions can be found in the `README` file of the repository. As pointed out also in the `README` file, the code is also hosted on DeepNote[2] and can be viewed with an enhanced graphic interface.

2 Domain Model Definition

Building the models In this first part of the project we have built a PSSM and HMM model representing the assigned domain, starting from a input sequence.

First of all, we performed a BLAST search against Uniprot using our assigned sequence in order to retrieve homologous proteins. We did not do it against *UniRef50* or *UniRef90* because we found enough significant hits through UniProt, setting up an E-value threshold of 10^{-20}. To generate a MSA from our hits we decided to use a Python wrapper to query the online EBI T-Coffee service via HTTP request and to analyse it we exploited JalView but we did not detect a notable presence of noise so the MSA remained the same.

The next step was focus on the generation of two models: a PSSM model with PSI-BLAST and a HMM model with HMMER. We made the decision to keep a E-value threshold of 10^{-5} on both model in order to have a fair comparison of the results. In particular, we set up the PSSM's number of

[1] `https://github.com/pterodattero/biodata2020`

[2] `https://deepnote.com/project/dab01843-4697-4e38-8601-caa4706bd153`

iterations of 4 to avoid the *drift problem*. PSSM and HMM were able to retrieve, respectively, 518 and 437 sequences.

Models Evaluation Thanks to InterPro API[3], we defined our ground truth retrieving the proteins belonging to the PFAM domain of our assignment sequence and we have filtered it removing the unnecessary information. We evaluated our models with two distinct approaches:

- an evaluation on the sequences, computing the confusion matrix on the matching/mismatching accession codes between the ground truth set and our hits;
- an evaluation on the domain position, computing a confusion matrix as shown in Figure 3 for each matching accession code based on the matching/mismatching residues and then considering the average of these matrices. Notice that the criterion we chose is arbitrary, nevertheless it can be useful to compare models.

To perform the analysis we used accuracy, precision, sensitivity, specificity, F1 score, and MCC as metrics for both approaches.

As Figure 1 shows, when we are looking at matching/mismatching sequences approach, PSSM works slightly better than HMM, with higher F1 score and MCC. On the other hand, in the Figure 2 the two histograms are balanced but also in this case PSSM has higher values. So we decided to keep PSSM model as final model.

3 Structural characterization

Database construction The first step in the Structural Characterization part of the project was to retrieve the required Databases to work with.

From SIFTS[4] a mapping table for all the UniProt PDBs was downloaded, in order to generate the `family_structures` dataset. To do so, every accession code found with the initial chosen model was mapped to this database, to extract every PDB successfully matching our model.

From this point, a `pdb_coverage` function was implemented and the retrieved dataset was filtered to keep only the PDBs matching our model with a coverage value higher than a threshold, set to 81%, choice made in

[3] `https://www.ebi.ac.uk/interpro/api/`

[4] `https://www.ebi.ac.uk/pdbe/docs/sifts/quick.html`

order to keep the number of retrieved PDBs under the limit of the *mTM-Align online service*, later to be used.

After these actions were performed, `family_structures` dataset consisted in just 31 PDBs.

Pairwise structural alignment A pairwise structural alignment was performed exploiting a locally downloaded `TM-Align` script, in order to confront the structure of each PDB previously downloaded and to calculate each TM-Score and *RMSD*.

Two heatmaps were then plotted, in order to display the results obtained with the calculations of *TM-score* in Figure 5 and *RMSD* Figure 4. Here it is easy to see that there is at least one outlier in the collection of domains, which has PDB ID `5ikn`.

Outliers of the distribution of minima *RMSD*, since their structure is significantly different, were removed from `family_structures`. To correctly visualize them, both a histogram (Figure 6) and a dendrogram (Figure 7) representing the hierarchical clustering has been created, exploiting the `scipy.cluster.hierarchy` package.

At the end of this analysis, one singular PDB was removed from the collection (`5ikn`) as expected from the previously plotted heatmaps.

Multiple structural alignment The Multiple structural alignment was performed exploiting the *mTM-Align online service*[5], as previously stated, giving as input a compressed `.tar` archive containing all the successfully downloaded PDBs (minus the outliers). The results were saved into two separate files:

- `result.pdb` containing the complete alignment of all the PDBs
- `cc.pdb` containing the common core alignment

After retrieving the multiple structural alignment, a distance matrix for every domain structure was produced (Figure 8), considering only the aligned common core and imposing a threshold on the sequence separation (threshold = 12 sequence positions), in order to ignore in-sequence closeness in favour of in-structure closeness, not considering couples of residuals distant less than the threshold.

A similar procedure was exploited to produce a contact map, averaging over all the distance matrices computed before and considering as contacts only areas with a structural separation lower than 8 *Ångström* (Figure 9).

[5] `https://yanglab.nankai.edu.cn/mTM-align/`

CATH superfamily retrieval Once again, from SIFTS, mappings between PDB IDs and CATH IDs were retrieved, in pair with a *CATH superfamily database* (retrieved from the CATH website[6]).

Filtering with the accession codes obtained from the chosen model, 549 CATH IDs associated with model sequences were found, and just 57 unique sequences matching the model, as expected, considering that CATH is not completely automatized and a lot of sequences are not present in the database.

As a result of this analysis it was clear that CATH superfamily `3.40.30.10` was the most important one, with 514 entries, against the 11 of `2.60.40.1250` and the 8 of `3.30.70.330`, respectively in second and third place Figure 11

4 Taxonomy

For a start, we retrieved the `family_sequences` dataset in `.xml` format. It is composed by the data of each *UniRef90* clusters containing our model proteins. We collected it using the Batch Retrieval service[7]. We did it directly in Python using the `requests` library.

Extraction of taxonomic data All the taxonomic information we need consists in NCBI taxonomic IDs, which is an identifier defined for every taxonomic category, from domain to species. In particular each cluster has a taxonomic ID which is common to all the cluster components and a taxonomic ID for each member. Furthermore, each cluster has also a representative member with its own taxonomic ID. To perform the next steps we decided parse from the `.xml` file the taxonomic ID of each representative member, in order to work with a handy number of IDs, but at the same time to work with species instead of general taxon.

Lineage retrieval Since we had only a set of taxonomic IDs, we had to retrieve a taxonomic DB with more exhaustive data. The `ete3` library offers the powerful interface `NCBITaxa`, that we use to build a taxonomic tree from our NCBI IDs and to gather each one's associated lineage. We wrote this information in `results/lineages.csv` file.

Taxonomic tree representation We decided to use the `toytree` convenient API for `ete3` to plot the taxonomic tree of `family_sequences`. For

[6] `http://cathdb.info/wiki?id=data:index`

[7] `https://www.uniprot.org/uploadlists/`

each taxonomic ID of the dataset we computed its abundance and we proportionally set the size of the corresponding node. The plot is shown in Figure 12.

5 Functional characterization

Similarly as we did for taxon IDs, we parsed GO terms for each cluster and we built a dictionary that maps accession ID of each cluster's representative member into a list of the GO annotations of the cluster. Furthermore, in order to compute the enrichment of our terms with SwissProt as reference, we computed a dictionary with the same structure for the entire database. We dumped the latter to a `.json` file, since its extraction from SwissProt `.xml` file takes a while. At last, we retrieved also the hierarchical structure of GO terms from geneontology.org[8].

Enrichment of GO terms For each GO term of the dataset and their ancestors we first computed the confusion matrix. Then we were able to compute the Fisher exact test to see if the presence of each examined GO term was significantly higher with respect to the reference database. Moreover, we also computed the fold increase. Finally, we put all this information in a dataframe.

Due to the construction of convention of the confusion matrix and since we are looking for enrichment and not for the opposite, we can compute an enrichment score[9] after right tail p-value. The wordcloud in Figure 13 shows most enriched terms with size proportional to their score.

Most significantly enriched branches To see enriched high level GO terms, that can be thought as branches, we can apply two filters. On one hand consider only GO terms that have a certain significativity, introducing a threshold for right tail p-value. On the other hand keep only terms that have a number of descendants which is above a certain quantile.

Notice that we are considering a term to be *high level* if it has a considerable number of descendants. Other approaches can involve, for example, the distance from the root, the distance from the furthest or closest leaf, and so on.

[8] `http://geneontology.org/docs/download-ontology/`

[9] $s(i) = \log \frac{1}{\varepsilon + p_i}$, with a small $\varepsilon > 0$

6 List of figures and tables

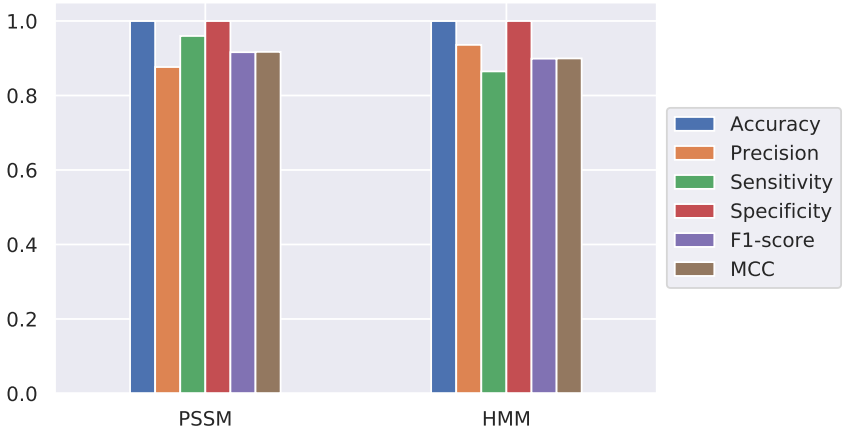

Figure 1: Sequences metrics approach comparison.

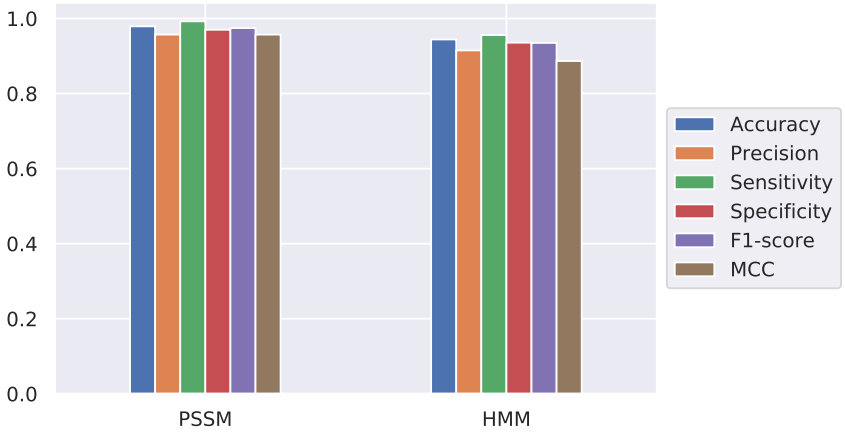

Figure 2: Domain position approach comparison.

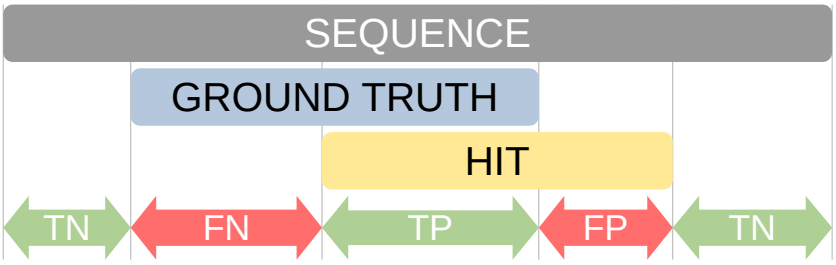

Figure 3: Confusion matrix for position metrics.

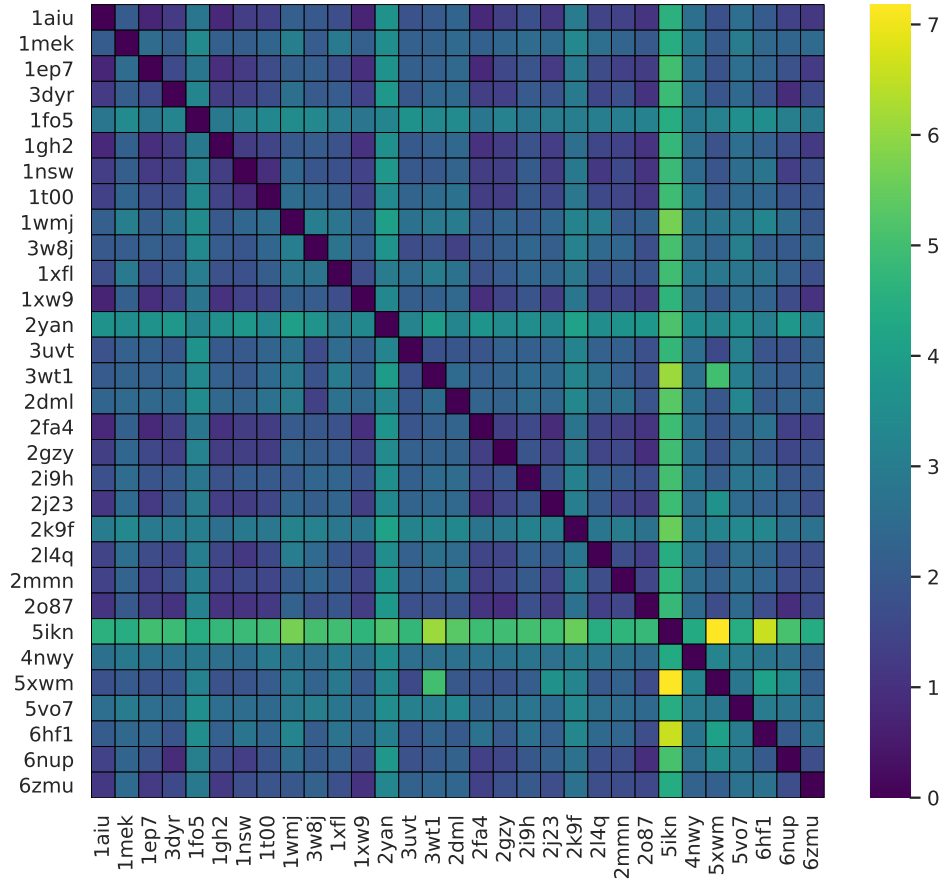

Figure 4: RMSD distance matrix.

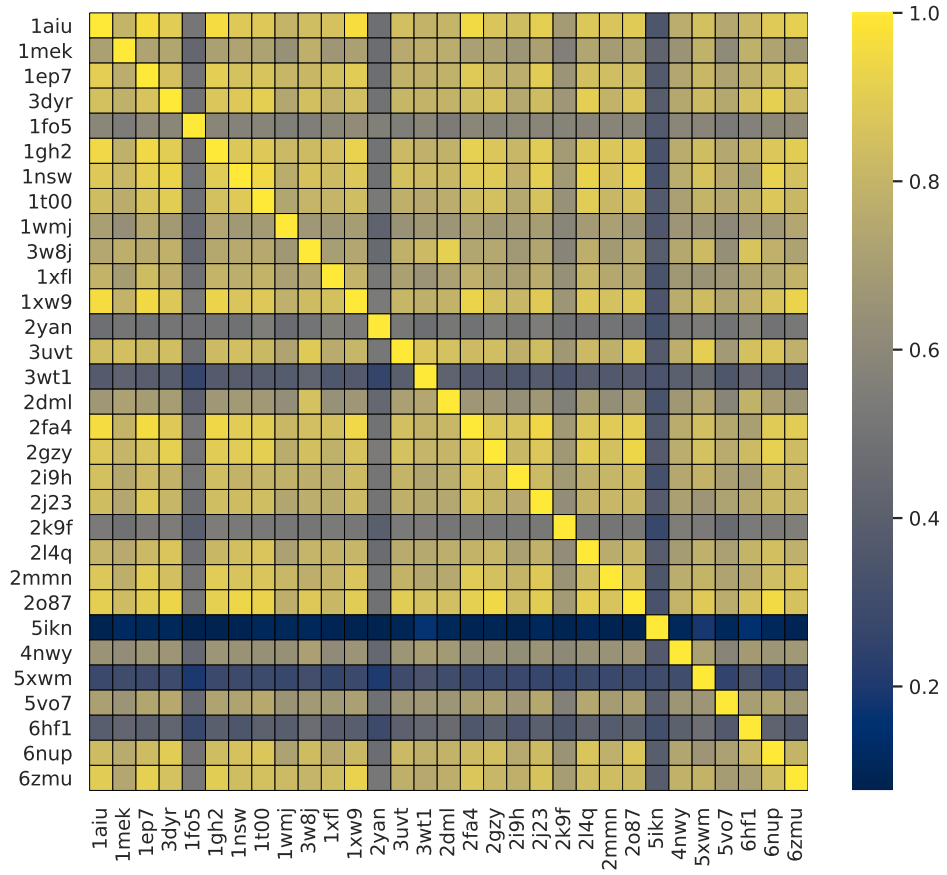

Figure 5: TM-score distance matrix.

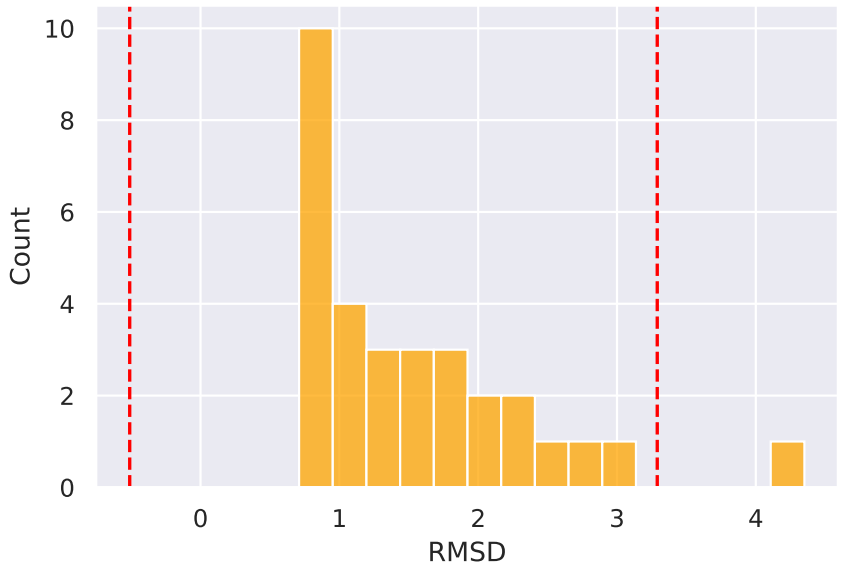

Figure 6: RMSD histogram employed to spot the singular outlier.

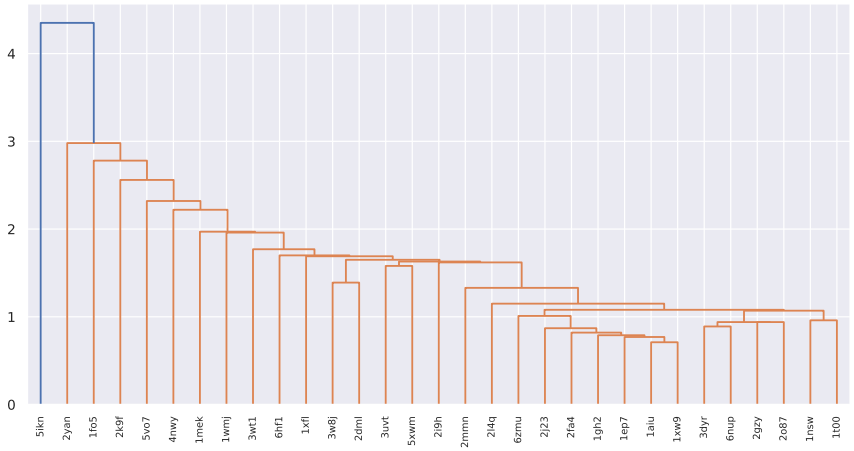

Figure 7: dendrogram employed to spot the singular outlier.

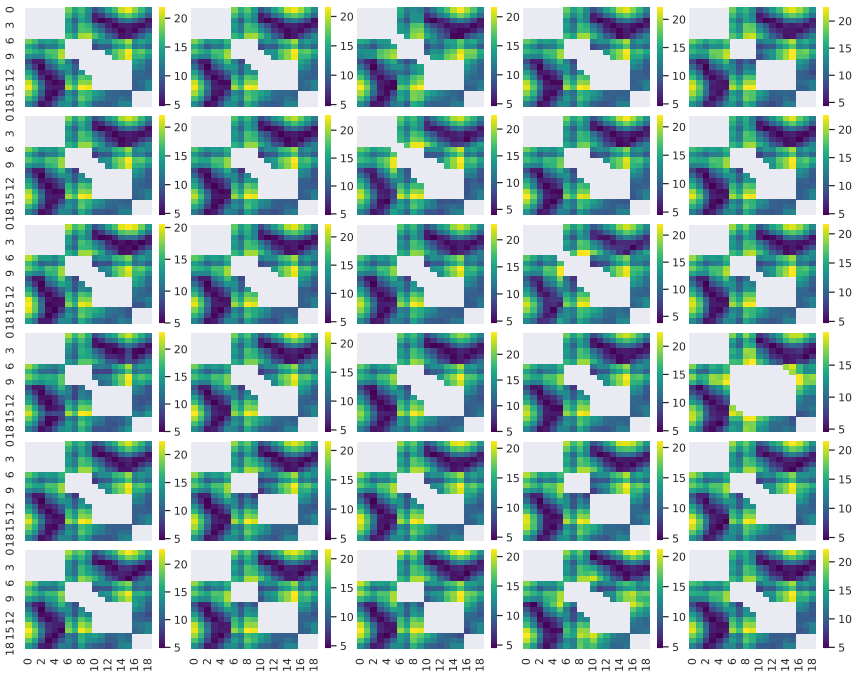

Figure 8: Distance matrices for every retrieved PDB.

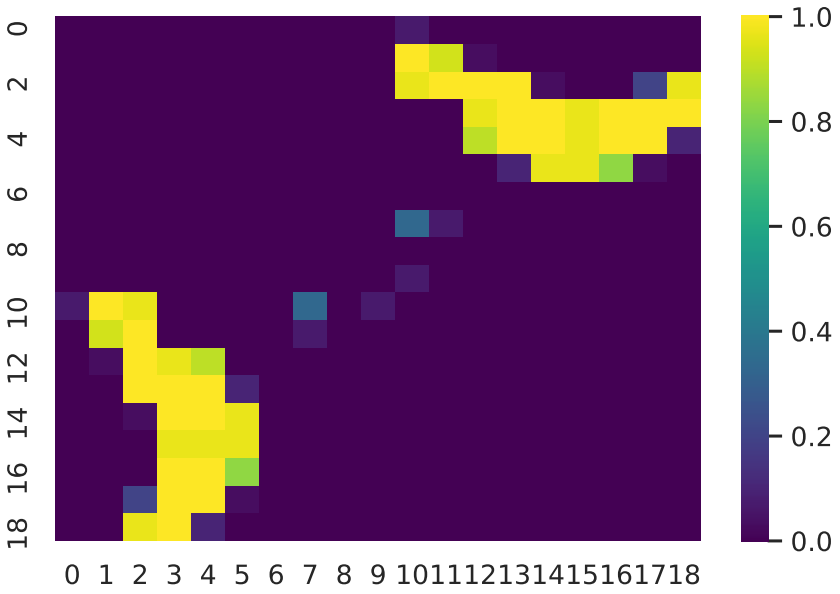

Figure 9: contact map obtained by averaging over all the distance matrices previously produced.

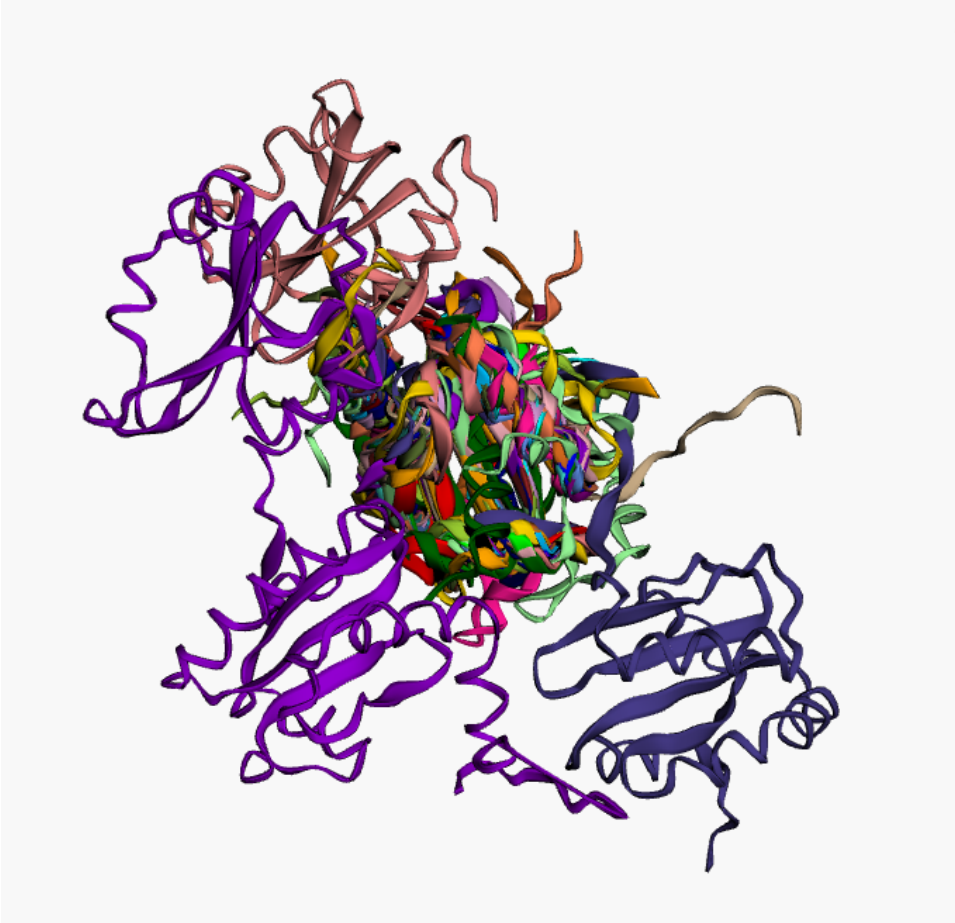

Figure 10: PyMol 3D rendering of mTM-align common core.

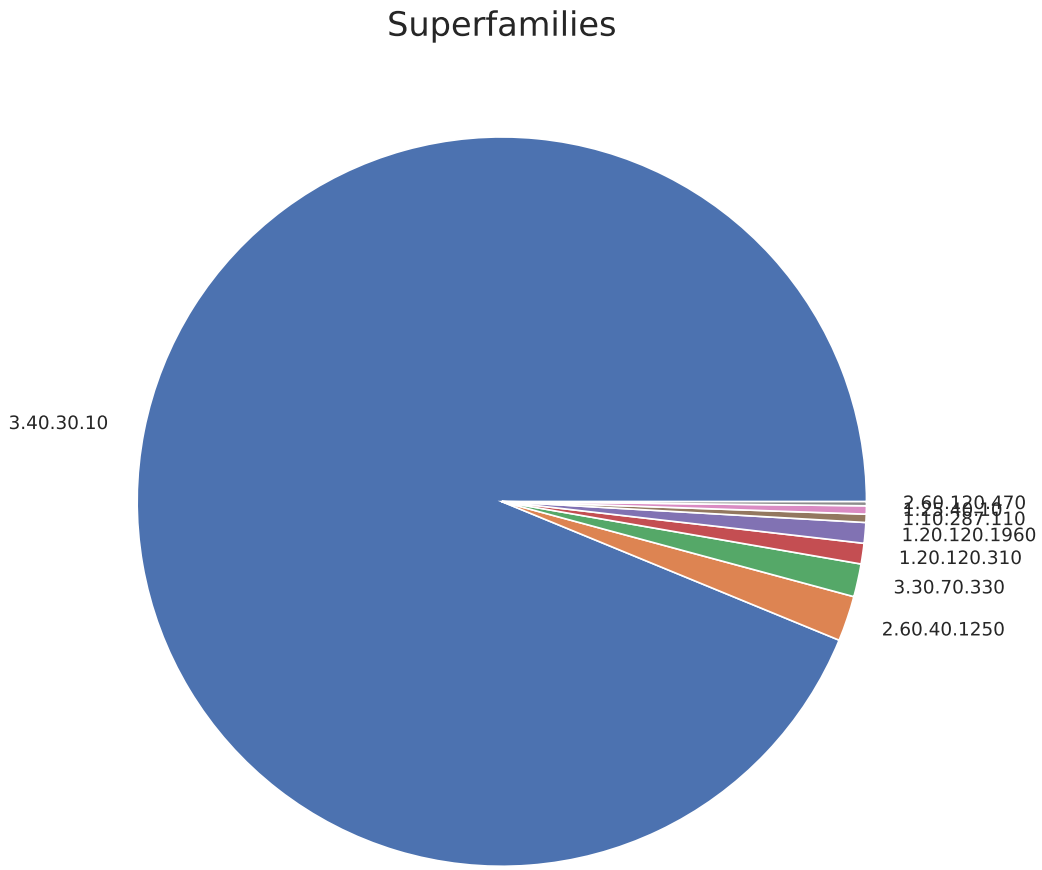

Figure 11: Pieplot representing the distributions of retrieved CATH superfamilies.

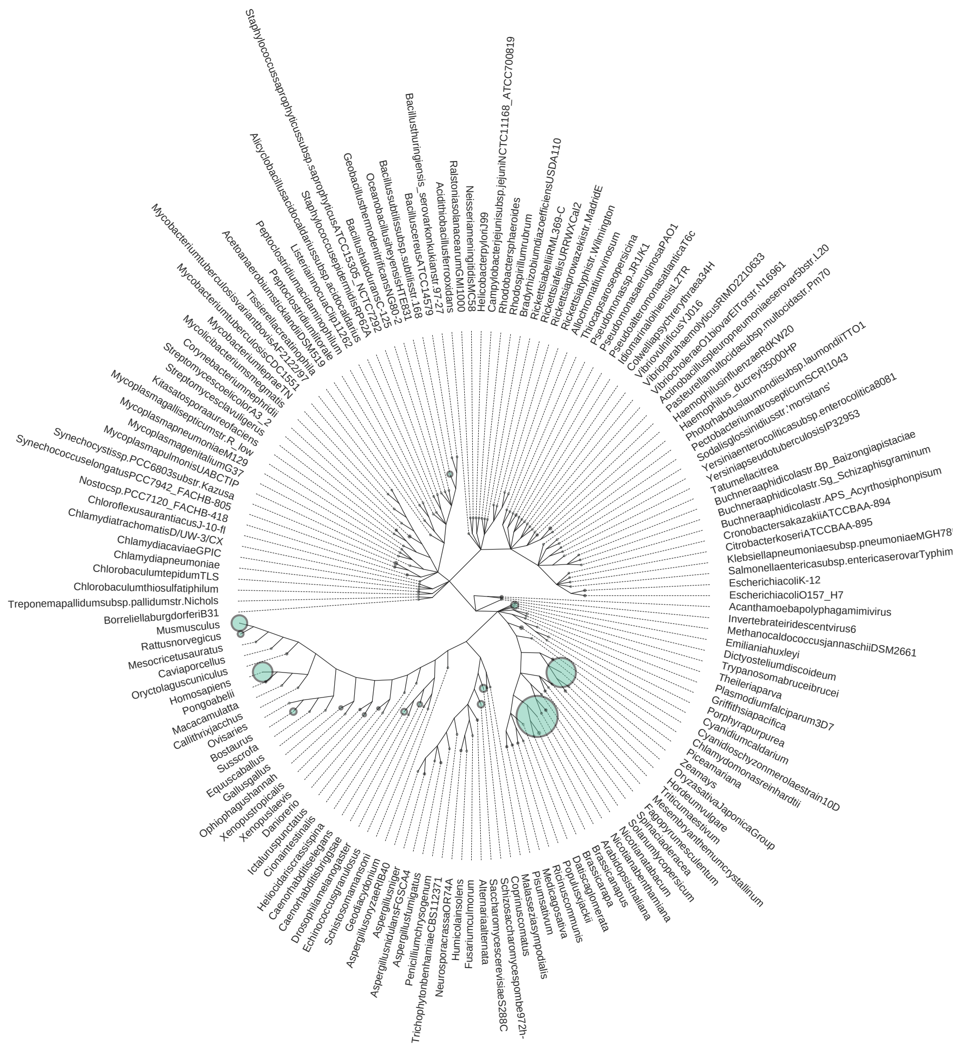

Figure 12: Taxonomic tree of `family_sequences`.

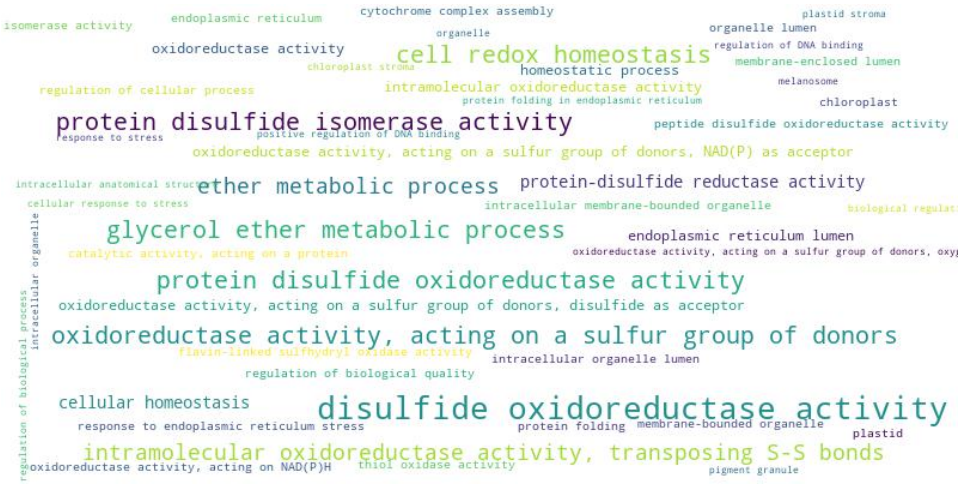

Figure 13: Most enriched GO terms in a wordcloud.