John Barker – Final Project

When choosing a problem for this project, one of the first questions that was already on my mind was the amount of sequence variation necessary to meaningfully change binding affinity. This is hardly an easy or unknown problem, but I was particularly inclined to think about it given the concern about the “Delta” variant of SARS-CoV2 and the worry that it might evade antibodies raised against vaccine provided spike proteins. Predicting structural similarity is a tremendous challenge that I cannot hope to address in a short project such as this, and virologists are unlikely to have all the possible variants of a novel spike protein on file, but the question was enough for me to think about structural prediction and the value of being able to clearly distinguish binding relationships or interactions from non-interacting ones.

As tools like the SWISS-PROT database allow for structural prediction and can try to model binding interactions, being able to tell when a sequence has changed enough to alter its interacting behavior is important for deciding whether a particular structural motif can be reasonably modeled as resembling a known structure for these predictions. Most specifically for this tool – as many others likely account for some of these considerations – I hoped that it would provide a quick visualization of gaps in the current knowledge space. Proteins that are either surrounded by similar and interacting sequences or adjacent to marginally higher similarity-scoring proteins that do interact with a particular ligand while not interacting themselves can provide useful insights into the limitations of a given binding interaction, as well as potential directions for future research or training data for binding prediction tools that are seeking to predict interactions of potential drug compounds with known biological targets.

The development of the back-end tool was reasonably straightforward. Already being loosely familiar with the IntAct database, I looked at some of its data to see what references it was in the habit of using. The first identifiers used were UniProt ID numbers, and the UniRef database already offered a pre-sorted list of similar protein sequences for a given cutoff – for this project, I began with the UniRef 90 database that catalogues all proteins with >90% sequence similarity, then set up a simple script using the python requests library to pull unformatted text for a given UniProt ID (the given ID to be later configured for user input). The script then loads the UniProt identifiers, UniProt names, and plain text names into a list object for later retrieval and display in the web template.

I then began searching through the IntAct DB for information on how to download its contents, and found several subsets of the whole database as well as a complete record of its contents available for download. I got about a dozen lines of code into designing a parser for the downloaded files when I began to worry about having to load a 20 GB database onto the class server. As IntAct does not appear to support programmatic access, I was concerned that the user would need to manually load in sections of the relevant database in order to display its results within the tool.

Pivoting slightly, I turned to the RESTful API for the BindingDB system that allows for GET requests returning unformatted XML of all of the ligands with interactions with a particular UniProt ID. The core concept of this tool may be demonstrated using either chemical ligands and small molecules recorded in BindingDB or in the larger proteins of IntAct DB. This would reduce the tool’s footprint on the class server while still performing the key function of visualizing the interaction between two databases and showing how small monomeric ligands can illustrate the effects of structural similarity on binding affinity. Unfortunately, many of the variables referencing Binding Database entries or responsible for pulling information from BDB are still abbreviated as IA for the original IntAct plan.

At this point, having built the bare bones of the system to collect what data would be displayed in my table, I began creating the HTML templates to display its results and consider the CSS necessary to format it as desired. I reasoned that I would need two critical elements in the template – a HTML form to pass user input to the database scraping program and a big extensible table down below that could accommodate additional inputs. I would also need the table itself to be interactive, allowing the user to pick from one of the SMILES values known by BindingDB to interact with the user’s query UniProt and get a coded output of which similar UniRef sequences did and did not have recorded interactions.

The workflow for using the tool is laid out more officially in the README, however the essence of its use is arranged as follows:

User opens page -> User inputs UniProt ID -> Table populates with all known BindingDB ligands for that UniProt ID on the left column in descending order of binding affinity and the first 100 UniRef90 sequences appear on the right side, thus far greyed out and not marked in any way -> User clicks on one of the BindingDB ligands (rendered as hyperlinks) and the program scrapes the database by that ligand, then checks each of the UniRef entries on the right against the output and flags them as having or not having a documented interaction. The ligands are referred to in the displayed output by their monomer IDs as these are the most generalized markers for the various compounds, many of which are small molecules – I have also included the SMILES molecular structure when available, although no special formatting is included beyond the rudimentary text representation.

At this point I began to set up the HTML template to integrate with the first three phases of this process, hard-coding some queries and passing the output to the table to check its formatting and the speed of accessing the various databases. This was the most challenging part of the assignment, attempting to connect up the various components to ensure that they could pass data smoothly back and forth to one another and that the web template would update correctly in response to user input. Given the limitations of the BindingDB API, I decided to have the user manually copy and paste the SMILES value for a given monomer into the search bar at the top as a way of filtering the results.

Because the handoff of form input data from html template to CGI script was giving me trouble, I instead turned to designing the CSS for the final output. Although I was not surprised to discover that CSS did not have functionality to change formatting in response to specific content, as that seems like a security and performance problem waiting to happen, I was pleased to see that I could tag <td> entries in the HTML output as having a particular data-content element that would trigger a change in formatting and could accomplish the green highlight I wanted for quick visual identification of matched bindings.