Andrew Smith

Documentation for Esophageal Cancer Protein Expression Exploration Tool

Background

Esophageal cancer affects more than 20,000 people a year, and 1 in 200 people will be diagnosed with esophageal cancer in their lifetime (SEER Cancer Stat Facts). The 5-year survival rate for esophageal cancer is only 21%, which makes it an especially severe form of cancer (the 5-year survival rate across all cancers is nearly 70%). Additional efforts to identify effective treatments for this deadly disease are critical. Esophageal cancer has affected 3 individuals close to me, and learning more about it was one of my primary motivations to study bioinformatics. The purpose of this project is to develop a new tool to explore and find new insights in existing esophageal cancer protein expression datasets.

The basis of this tool is 125 samples of esophageal cancer cells with protein expression data for nearly 500 proteins from each sample. I chose to work with protein expression data for a few reasons. Proteins play an important role in cancer because they control the normal functioning of a cell and the regulation of its metabolism and normal lifecycle. Cancer can manifest in drastic changes in protein levels. Oncoproteins that drive growth, like MYC, can become highly expressed and tumor suppressor genes, like p53, will be downregulated. Some of these proteins are actually driving oncogenesis, while others are merely up or downregulated as a side effect. Identifying the affected protein pathways and finding what is driving the dysregulation is a good starting point for developing new therapies.

I have chosen to visualize two things about a target protein: the other proteins that have correlated expression and the expression level across samples. I chose these because I think they are useful in identifying the key oncogenic proteins involved in esophageal cancer. The correlation graphs can help identify protein networks and groups of functionally related proteins. This gives context for what proteins might be good drug targets, and the possible compensatory mechanisms exist that could lead to treatment resistance. The expression heatmap can distinguish between up and downregulated genes, allows us to see how similar or different each sample is, and could identify distinct esophageal cancer pathologies. This type of analysis could also lead to the identification of biomarkers that are relevant for diagnosis, treatment, or prognosis.

Design and Development

The first step was to acquire data to populate the tool. I generated a manifest of the data that I need for the project on the NIH Genomic Data Commons (GDC) website, then used the NIH GDC Data Transfer Tool to download it. Data from each of the 125 samples was stored as a tab separated value file within its own directory, which was named after the sample's ID. These were parsed into the SQL database as described

below. The second table, with correlations, was calculated from this data and did not involve any additional information retrieval.

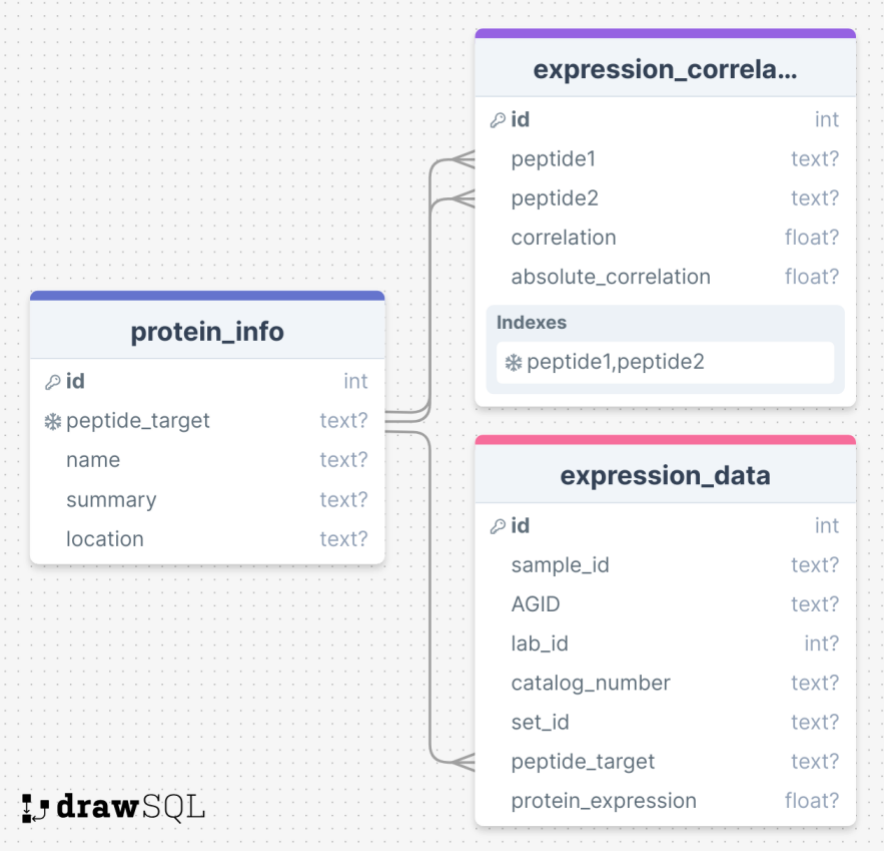
To obtain the metadata for each protein, I took a list of proteins that were present in the samples and searched for each on one NCBI Protein using the Entrez API with Biopython. From the protein results, I used regular expressions to find the gene that encodes that protein. I searched the gene name (limited to humans) in NCBI Gene and found the most relevant result, then retrieved the summary information, which was stored in the SQL database.

There is a directory of Python scripts, populate_database that was used to parse the data downloaded from these sources and store it in a normalized fashion in a relational database. For each one, I had to create a table with an appropriate schema and data types. For the GDC data, I created a named tuple class for an expression level data point and iterated through each directory and tsv file to create lists of these objects to store in the database. When designing this database I considered two possible schemas. In one, each protein could be a column, then there would be a single row for each sample that could contain all the expression levels. This would make the table more compact, but at the expense of flexibility and generalization. The problem is that if we encounter a sample that has a protein we didn't anticipate, maybe by adding more samples later, you must change the schema for the entire table, which is bad practice. That's why I instead opted to have each row be a single expression level for a single protein. This makes it easier to change which proteins I want to consider later and makes the table more flexible. The tradeoff is there are many more rows, since each sample creates nearly 500 new rows.

I decided to pre-calculate the correlations and store the results in a table because finding each correlation requires putting the entire expression levels table into memory and doing expensive computations. The downside to this approach is that I need to invalidate these cached calculations if more data is added. To make this table, I retrieved expression levels from the expression levels table and made a pivot table, like the first schema I described above. I removed any proteins from the table that had missing expression data. The data was stored in a pandas DataFrame, and the .corr() method was used to calculate the Pearson Correlation Coefficients between each possible pairing of proteins. I iterated through each of these pairings and added them to the table in both orders (ie as protein_1 vs. protein_2 and as protein_2 vs. protein_1), which makes the search easier and more intuitive. I made sure to include identity correlations (ie protein_1 vs. protein_1), because verifying that these equal 1 is a good sanity check that my procedure is working as intended. I added a constraint that each combination protein_1, protein_2 is unique, so new correlation calculations will have to override old ones. This is a sort of simplified cache invalidation.

Storing the protein metadata was relatively straightforward. Since I used Biopython with Entrez, the data had already been parsed into Python objects which are easy to work with.

The database schema was visualized with drawSQL:



The autocomplete JavaScript function is essentially the same as the one I developed for unit 8. When the form is submitted on the website, an event listener is triggered that calls two JavaScript functions. The first retrieves data from the protein_info table and populates the information box at the top of the page. The second uses promise to asynchronously retrieve the two images that are the data visualizations. Generating the visualizations is relatively slow, so I used promise to prevent the first API call from blocking the start of the second one and allow them to be generated simultaneously. The normal method of returning data from a CGI script by printing it doesn't work for images, so I must write the binary data to stdout to transmit it. This blob data is passed to URL.createObjectURL() to create a string URL for the object, which is set as the image's source attribute. Each section is shown only after the data is retrieved. If there is an error while retrieving data, an alert is shown to the user.

Each of the sections of the webpage that have content updated for the selected protein have their own CGI script. I could have a single CGI endpoint that calls the different functions, but this setup allows me to handle the asynchronous programming with the JavaScript event loop rather than in Python, which seems more natural (although the inverse is certainly possible). The protein info is retrieved with a simple SQL query, and JSONified. The graphs are made with matplotlib then saved as BytesIO objects. The protein name is searched for in a case-insensitive way, but in cases where I want to

display the proper capitalization (i.e. the title of the correlations graph), I retrieve the original name with an additional SQL query.

The heatmap takes the 125 expression levels and reshapes the vector into a 12x11 array, because the squarer shape fits better in the page layout. The min and max values for the color scale are fixed based on the min and max of the entire dataset to allow for better comparison between proteins. The viridis colormap is used to because it is perceptually uniform as well as pretty. For the correlations, I retrieve the 10 proteins with the highest absolute correlation (positive or negative) because this should represent the most important relationships. They are drawn on a vertical bar graph, with red bars for the negative values and green bars for the positive values. Labels with the protein names are placed in white on top of the bars, and the value labels are set to just outside of the bars. I experimented with different label layouts and found this to be the clearest and most readable.

Conclusion

Overall, this project allowed me to incorporate a wide range of skills from the term and practice using them in combination to create a useful tool. The client side was made with HTML, CSS, and JavaScript, which interacted with a backend with a Python layer that retrieved and formatted data stored in SQL relational databases. I also incorporated skills from other courses and professional experience like statistical analysis, asynchronous programming, advanced Entrez queries, and data visualization with matplotlib.

Sources

Gene [Internet]. Bethesda (MD): National Library of Medicine (US), National Center for Biotechnology Information; 2004 – [cited 2024 July 28]. Available from: https://www.ncbi.nlm.nih.gov/gene/

Grossman, Robert L., Heath, Allison P., Ferretti, Vincent, Varmus, Harold E., Lowy, Douglas R., Kibbe, Warren A., Staudt, Louis M. (2016) Toward a Shared Vision for Cancer Genomic Data. New England Journal of Medicine375:12, 1109-1112

SEER Cancer Stat Facts: Esophageal Cancer. National Cancer Institute. Bethesda, MD, https://seer.cancer.gov/statfacts/html/esoph.html

Screenshots of Features in Use

Figure 1. Autocomplete in Search Box

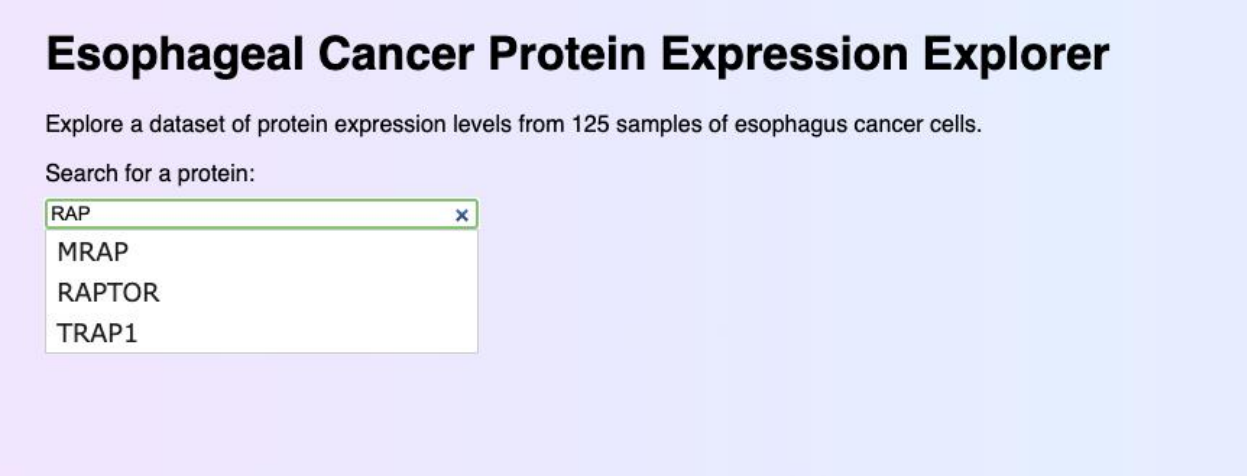


Figure 2. Protein Info Box for EMA

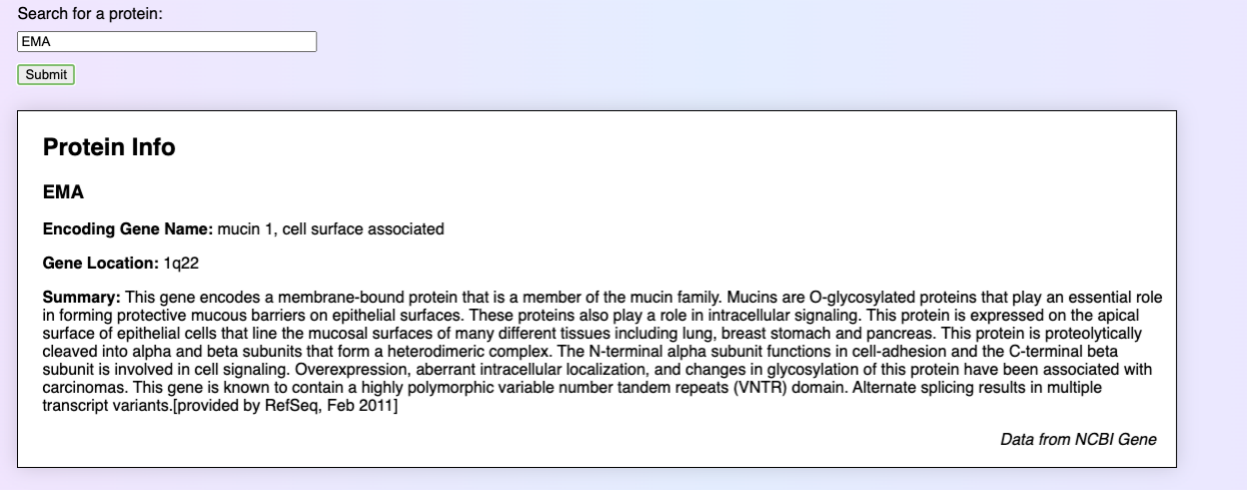


Figure 3. Expression Visualizations for EMA

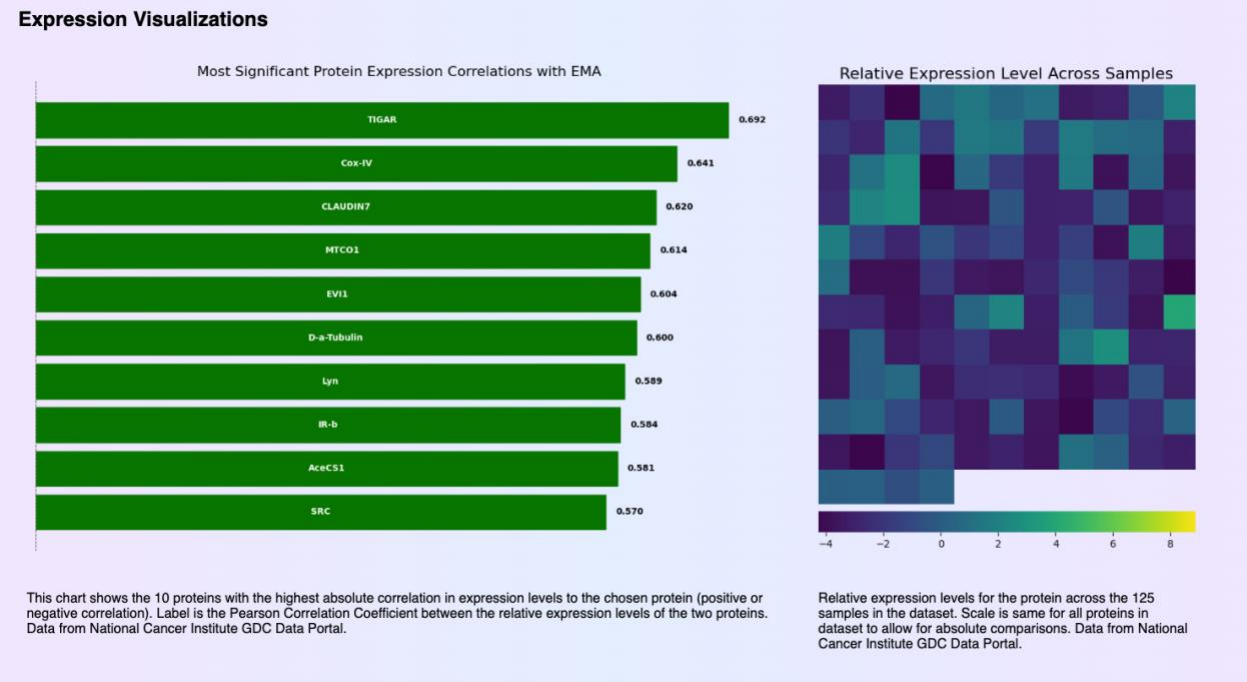


Figure 4. Correlation Graph with Positive and Negative Values