**Important Information**

1. Project 3 is due by **5:00pm PST on November 30, 2022**. Late policy applies up to 3 days as specified in the syllabus. Don't forget that you have two late days for the quarter.
2. Programming projects must be completed individually. You may discuss algorithms with others, but the coding should be done alone. You must explicitly name everyone with whom you discussed this project in the header comments of your code and in the collaboration attestation in the project quiz. Students must abide by the terms of the Stanford Honor Code.
3. Remember to consult Ed, as many common questions will be asked and answered there.
4. Prior to beginning the assignment, please read the [Code Policy](https://canvas.stanford.edu/courses/158755/pages/code-policy).
5. Read this project page carefully; your source code must run **exactly** as specified below. We will be unit testing your code on Gradescope.

**Technical Background**

This assignment will take you through common steps when performing a bioinformatic research project. We will walk you through using publicly available databases to get protein data, sequences, and functional annotation. We will then walk you through using existing models to construct your own analysis. We will focus on the utility of the ESM-2 model, which is a large language model trained on protein sequences and creates an embedding vector that captures the latent features of the sequence. (<https://www.science.org/doi/10.1126/science.ade2574>) We will then use a model published by Rao *et al.* (E2EATP) to use ESM embedding vectors to predict structural and functional features of the protein, specifically binding sites. (<https://pubs.acs.org/doi/10.1021/acs.jcim.3c01298> - can be accessed by logging in with Stanford University). Since the goal of this project is to become familiar with using such neural network models and analyzing the results, we will provide the model code for you and ask you to write scripts to systematically analyze the results. To run the provided code, certain packages are required, including: torch, numpy, fair-esm, and biopython. We provide the required packages and version in *requiremens.txt.* Run pip install -r requirements.txt to install.

**Background**

**Kinases: An Overview**

Kinases are a diverse family of enzymes that play critical roles in cellular signaling, metabolism, and regulation by catalyzing the transfer of phosphate groups from adenosine triphosphate (ATP) to specific substrates. This phosphorylation process is essential for modulating protein activity, altering protein interactions, and controlling various cellular processes such as cell growth, differentiation, and apoptosis. Kinases are classified into several families based on their structure and substrate specificity, including serine/threonine kinases, tyrosine kinases, and dual-specificity kinases, each with distinct regulatory mechanisms and physiological functions.

**ATP Binding Sites**

The ATP binding site is a highly conserved region within kinases, crucial for their enzymatic activity. ATP serves as the phosphate donor in the phosphorylation reaction, and its binding is fundamental for the kinase to exert its function. Typically, the binding site consists of several key residues that interact directly with ATP, ensuring proper recognition and specificity. The binding involves hydrogen bonds, ionic interactions, and hydrophobic contacts, which stabilize the ATP molecule in the active site.

**Importance of Predicting Binding Residues**

Predicting the binding residues of kinases, especially those that interact with ATP, is of paramount importance for several reasons:

1. **Understanding Enzyme Mechanism**: Identifying the specific residues that contribute to ATP binding and catalysis can provide insights into the mechanism of action of kinases. This knowledge can help elucidate how kinases regulate various biological pathways and processes.
2. **Drug Discovery**: Many kinases are key targets for drug development due to their roles in diseases, particularly cancer. Understanding the ATP binding site allows researchers to design small molecules that can specifically inhibit or activate kinases. Predicting binding residues aids in the identification of potential druggable pockets within the kinase structure, facilitating the rational design of inhibitors with improved efficacy and selectivity.
3. **Biomarker Development**: Mutations in kinase genes can lead to altered phosphorylation activity, contributing to disease progression. By predicting binding residues and understanding how mutations affect ATP binding, researchers can identify potential biomarkers for disease diagnosis and prognosis.
4. **Structural Biology and Computational Modeling**: Accurate prediction of binding residues contributes to structural models of kinases. These models are essential for simulating interactions and understanding the dynamic behavior of kinases in response to various stimuli. Knowledge of binding residues enhances the predictive power of computational approaches used in drug design and protein engineering.

Kinases are vital components of cellular regulation, and their ATP binding sites are critical for their function. Predicting the binding residues is essential for advancing our understanding of enzymatic mechanisms, aiding drug discovery efforts, and developing potential biomarkers. This foundational knowledge is particularly relevant for graduate students studying molecular biology, biochemistry, and bioinformatics, as it provides a framework for exploring the intricate relationships between protein structure, function, and therapeutic applications.

**Project Overview**

The objective of this project is to predict ATP binding sites in kinases using embedding representations generated by the ESM (Evolutionary Scale Modeling) framework. The project will begin by acquiring a FASTA file containing sequences of various kinase proteins, which will serve as the foundation for analysis and predictions. Next, we will utilize the ESM model to generate representation vectors for each kinase sequence, capturing essential features that reflect their structural and functional properties. Different pooling strategies will then be analyzed to determine their impact on the representation of kinase sequences and the accuracy of ATP binding site predictions. Using the processed embeddings, we will predict the ATP binding sites, employing machine learning models or existing tools designed for binding site identification. The effectiveness of these predictions will be assessed by calculating relevant metrics such as precision, recall, F1-score, and accuracy, providing insights into the performance of the pooling strategies and the predictive model. Following this, we will obtain Protein Data Bank (PDB) files corresponding to the kinase proteins of interest and utilize PyMOL to visualize their 3D structures alongside the predicted ATP binding sites, facilitating an understanding of the spatial arrangement of these critical sites. Finally, a qualitative analysis will be conducted to evaluate the predicted ATP binding sites, discussing their biological relevance and significance, including considerations such as conservation across species, interactions with ATP, and the implications of binding site mutations. Through this comprehensive approach, the project aims to enhance our understanding of ATP binding sites in kinases, contributing valuable insights to drug discovery efforts targeting kinase-related pathways and furthering knowledge of kinase regulation in various biological contexts.

**Data Formats**

We include data files in the following formats:

**.fasta files:** These files contain protein sequences, represented as a string of 1-letter amino acid codes. The lines starting with the symbol “>” simply denote the protein and chain that the sequence comes from.

**.pdb files:** This file format is used to store protein structures derived from X-ray crystallography data. These files are fixed-width and contain metadata information as well as the identity and xyz-coordinates of every atom in the protein. More information on the PDB file format is available on [WikipediaLinks to an external site.](https://en.wikipedia.org/wiki/Protein_Data_Bank_(file_format)), but this is not necessary to complete the project; we provide functions to interact with PDB files as necessary.

**Implementation**

**Part 1: Using ESM to get vector representation of protein sequences (40 points)**

ESM (Evolutionary Scale Modeling) is a deep learning framework developed by Facebook AI Research that leverages large-scale protein sequence data to generate high-quality embeddings for protein sequences. It uses transformer architectures to capture the intricate relationships between amino acids in a sequence, allowing the model to learn context-dependent representations that reflect the structural and functional properties of proteins. ESM is pretrained on diverse protein databases, enabling it to understand evolutionary information and generalize well across different protein families. By converting protein sequences into high-dimensional embeddings, ESM facilitates various downstream tasks, such as predicting protein structure, function, and interactions, making it a powerful tool in computational biology and bioinformatics.

1. Create an ESM\_model class in a file called esm\_model.py
   1. The arguments will be the input fasta file, output folder directory, and the protein family metadata file

Run as:

Python esm\_model.py <fasta file> <output\_dir> <protein family datafile>

* 1. The output directory will be where you output the calculated vector dictionaries and figures. Use ‘outputs’ as the directory name
  2. The protein family metadata file is a csv file with the protein id and a list of its protein family names in a row

1. ***load\_fasta(fasta\_file)***Write a function call load\_fasta that takes in the fasta file and outputs a list of tuples where the first element of the tuple is the UniProt protein id and the second element is the sequence.
   1. Use BioPython package’s SeqIO functions to do this (<https://biopython.org/wiki/SeqIO>)
   2. The fasta file we are working with is downloaded from UniProt. The header takes the format: >sp|<<UniProt Id>>|<<Protein Name>>\_<<organism>>
2. ***get\_vectors(list\_tuples\_protId\_seq)*** Write a function call get\_vectors that takes in the list of tuples and outputs a a dictionary with the UniProt protein id as the key and the representation vector from the ESM model as the value.
   1. We will use the smaller model for this project for ease of use: esm2\_t33\_650M\_UR50D (all possible model types listed here: <https://github.com/facebookresearch/esm?tab=readme-ov-file#available-models>). We will use the last layer (33) of the model to get the representation vectors.
   2. Pass only one sequence to the model at a time. When passing more than one sequence together to the model, the sequences are padded to match lengths. Since we want to perform downstream prediction tasks that are residue specific, we want to preserve the sequence length for each embedding.
   3. After you load the model, call model.eval() for deterministic results.
   4. Use the embedding returned by the model from the last layer (33). This will produce a vector per residue in a tensor object of shape (1, <sequence length+2>, <embedding length = 1280>). The model adds a start and stop token to the beginning and end of the sequence that is the first and last vectors produced, but our downstream prediction task does not use these two token, so remove them from your final dictionary values. This readme includes example usage: <https://github.com/facebookresearch/esm>
   5. Use pickle to write the final dictionary to a file in the output folder called: embeddings.pkl. Do not save inside the function get\_vectors.
3. ***pool\_representations(dict\_protId\_embedding, mean\_max\_param)*** Write a function called pool\_representations that takes in the dictionary of per token vectors and a parameter denoting the pooling regime (‘mean’ or ‘max) and outputs a dictionary with the UniProt protein id as keys and the pooled vector (as a numpy object) as the values (dimension = [1,1280]). The output vector from the ESM model is per residue, but some tasks require one vector per protein instead. Therefore, the per residue vectors are pooled together to create one representation vector for the protein. Two common pooling strategies are mean pooling and max pooling. Mean pooling takes the average value across all residues for each index and max pooling takes the max value across all residues for each index.

Mean Pooled

Max Pooled

* 1. Use this function to perform both mean pooling and max pooling. Save both the dictionaries in the output folder directory, as meanPooled\_emebddings.pkl and maxPooled\_embeddings.pkl, respectively. Do not save within the function
  2. Use these two dictionaries to create two TSNE plots for each pooling regime (t-SNE is from the scikit-learn package). t-Distributed Stochastic Neighbor Embedding (t-SNE) is a powerful dimensionality reduction technique primarily used for visualizing high-dimensional data in a lower-dimensional space, typically two or three dimensions. It works by modeling the similarities between data points as probabilities and aims to preserve these similarities when mapping the data to a lower-dimensional representation. The algorithm emphasizes retaining local structures, making it particularly effective for revealing clusters in data. To use t-SNE, you first need to prepare your high-dimensional data, usually represented as a matrix where rows correspond to samples and columns correspond to features. You can then apply the t-SNE algorithm using libraries such as scikit-learn in Python. After fitting the model, you can visualize the resulting low-dimensional embeddings using scatter plots to identify patterns, clusters, and relationships within the data.
     1. Use n\_components = 2, random\_state=42 when creating the TSNE
     2. Save the two figures in the outputs folder as maxPooled\_viz.png and meanPooled\_viz.png

1. ***calculate\_silhouette\_score (family\_data\_file, pooled\_vectors\_dict)*** Write a function called calculate\_silhouette\_score that takes in the family metadata file and a pooled vector dictionary, calculates the silhouette score for each family, and then returns a dataframe with the family name and corresponding score per row.
   1. Do this for both the mean pooled and max pooled dictionaries and save each dataframe as a csv in the outputs folder called meanPooled\_silhouette.csv or maxPooled\_silhouette.csv
   2. Create two TSNE plots (one for each pooling regime) such that the points are colored by their family membership. Save these figures in the outputs folder as meanPooled\_viz\_family.png and maxPooled\_viz\_family.png

**Part 2: Using E2EATP to predict ATP Binding sites (30 points)**

In this part, we will use a modified version of E2EATP to predict binding sites in our set of kinases, based on this paper: <https://pubs.acs.org/doi/10.1021/acs.jcim.3c01298>

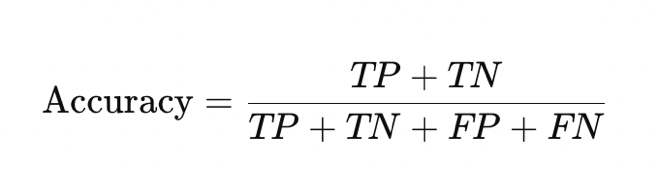
We will also evaluate this model. We will imagine that we are provided with a list of known ATP binding sites for protein that were held out of the model training, and so we will use these known sites to calculate metrics for the model.

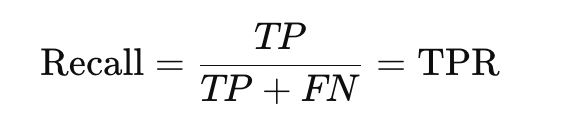
1. Use the provided e2eatp folder and predict.py script to predict the binding sites of the set of kinases in the fasta file.
   1. To run the script, cd into the e2eatp folder and type this into the terminal:

python predict.py -outfolder ../predict\_results -seq\_fa <fasta\_file> --embeddings ../outputs/embeddings.pkl

* 1. We will use the per residue embedding because it will create a more specific prediction than the pooled embeddings.
  2. The prediction script outputs a file for each protein with the Index sequence, Amino Acid (AA) at the index of the sequence, and the predicted probability that this AA is an ATP binding site.

1. Write a class called AnalyzePredictions in a file called analyze.py. We will use this class to analyze the files created by the prediction model. Write this class in the main directory, not inside the e2eatp directory.
2. ***load\_data(predicted\_results\_dir, cutoff)*** Write a function called load\_data that takes in the directory with the predicted results and a cutoff value.
   1. This function will iterate through the files in the predict\_results folder (use the function os.listdir(<folder path>)) and read the tsv file. Save the rows where the probability is higher than the cutoff in a pandas dictionary that has the header (protid, Index, AA, Prob) and return this dataframe
   2. Using a cutoff = 0.5, plot a histogram of the frequency that each amino acid is an ATP binding site across all proteins. Save this plot in the output directory as histogram.png. Write the call to save the figure outside of the load\_data function.
3. ***get\_bindingSite\_labels(protein\_bindingSite\_file)*** Write a function called get\_bindingSite\_labels that reads in the file protein\_bindingSite.csv into a dataframe and returns this dataframe, such that each row has a protein id and one of its binding sites. A protein will have multiple binding sites, so a protein id will appear in multiple rows of the dataframe.
4. ***calculate\_metrics(predicted\_results\_dir, cutoff, bindingSite\_dataframe)*** Write a function called calculate\_metrics that takes in a cutoff value and the predicted results directory, and the known bindingSites dataframe. The function will return the accuracy, precision, true positive rate, and false positive rate ( in that order).
   1. It may be helpful to call load\_data within this function.
   2. There are some predictions (protein in the predicted\_results directory) that are not in the known binding site dataframe. Be sure to include only proteins that are in the known binding site dataframe in these metric calculations.

A mathematical equation with black text

Description automatically generatedA black and white math equation

Description automatically generated

1. Run calculate metrics for at least 10 cutoff points within the range 0 to 1 and plot and ROC curve. Save this curve in the outputs folder, called ROC.png

**Submission**:

This is the directory structure for submission:

.root

|\_\_ outputs

|\_\_\_ embeddings.pkl

|\_\_\_ maxPooled\_embeddings.pkl

|\_\_\_ meanPooled\_embeddings.pkl

|\_\_\_ maxPooled\_silhouette.csv

|\_\_\_ meanPooled\_silhouette.csv

|\_\_\_ predict\_results

|\_\_\_ <<tsv file for each protein>>

|\_\_\_ analyze.py

|\_\_\_ esm\_model.py

\*\* Do not submit any other files.

**Rubric (Total 70pts):**

* Autograder total - 60 points
  + ESM model – 40 points
    - Correct protein embeddings (10 pts)
    - Correct mean pooling (10 pts)
    - Correct max pooling (10 pts)
    - Correct silhouette scores (10 pts)
  + Analyze – 30 points
    - Correctly used provided model to predict results (10 pts)
    - Correctly find residues above cutoff (10 pts)
    - Correctly calculate metrics at different cutoffs (10 pts)
* Style - 10 points

**Appendix: How to use PyMol for visualization**

Arguably the most interesting and satisfying part of any protein simulation is the visualization. Visually comparing the final folded structure of our protein to its native crystal structure allows us to better understand where our approach succeeds and where it struggles. It also provides a visual sanity check that a structure with a low RMSD and low energy actually does look similar to the real protein. Finally, visualizing how the structure changes over the course of the simulation helps us to understand how our simulation procedure works to find low-energy conformations.

One of the most popular software programs for visualizing protein structures is PyMol, which is free for students and available on all platforms (Linux, Mac, and Windows). You can download PyMol from [https://pymol.org. Links to an external site.](https://pymol.org/)

The PyMol interface consists of a large display screen, a console at the top (boxed in red), and an object panel on the right side (boxed in green).

A screenshot of a video game

Description automatically generated

The two main ways to interact with PyMol are by typing commands into the console and by using the buttons in the object panel. You will need to visualize some of your outputs for the project quiz. In addition to the tips listed below, there is a PyMol cheat sheet posted in the Project 3 files on Canvas.

Basic trackpad commands (for Mac, these may differ slightly on Windows):

* Rotate view: click and drag
* Zoom: two-finger pinch
* Pan: Option + click and drag

Some important console commands:

* To load a structure:

load /path/to/structure.pdb

* To load a structure into a named object MyProtein (in PyMol syntax, commands are structured as 'command, object'):

load /path/to/structure.pdb, MyProtein

* Loading multiple PDB files into one object results in **frames**(shown here as 1/100). If your files are labeled consistently, you can load multiple at once using the following command:
* To delete an object:

delete MyProtein

Some important notes about the object panel:

* Clicking the object name will toggle the object on/off
* The buttons ‘A’ (Action), ‘S’ (Show), ‘H’ (Hide), ‘L’ (Label), ‘C’ (Color) each reveal a menu that controls how the object is displayed. Play around with these menus a bit to get a sense for what each of them does.A screenshot of a computer

  Description automatically generated       A screenshot of a computer

  Description automatically generated
* Color is controlled using the ‘C’ menu. It can be useful to color by secondary structure (‘C’ > ‘by ss’) to visualize high-level fold, as shown below.

A structure of a protein

Description automatically generated with medium confidence