Protein Stability Prediction (MLB S24) Kaggle Challenge Report

Macrina Lobo

1 Data Processing

Exploratory data analysis was performed. This included plotting the distribution of sequence length as well as the histogram of average occurrence of each amino acid in training and test sets. I observed similar distribution on train and test set with each amino acid having good coverage (except U).

I decided to use protein language models (PLMs) for this task. My workflow consisted of tokenization and embedding generation utilizing the last hidden layer of the PLM encoder. Mean pooling was applied at the final layer resulting in a 320 dimension embedding for each input sequence. I tried several esm2 models[2] and decided to go with $esm2\_t6\_8M\_UR50D$ due to having lower number of layers, parameters and embedding dimension thereby leading to faster runtime and lower computational resource requirements. PLM models like ESM2 typically apply padding and truncate the length of the input protein sequences. I truncated my sequence length to a sequence of 1024 amino acids. Alternatively, mean aggregation over every set of 1024 consecutive amino acid sequences in the input instead of truncation was another option. However, the frequency distribution of the first 1024 bases is representative of the entire dataset and consistent on training and test data so truncation might not drastically reduce performance (Figure 1).

Finally, I visualized the embeddings using the first two components of PCA, UMAP computed with 50 principal components, 15 neighbors and a min distance of 0.1. Though there is some separation between high and low stability values on the embeddings, it warrants further downstream transfer learning models.

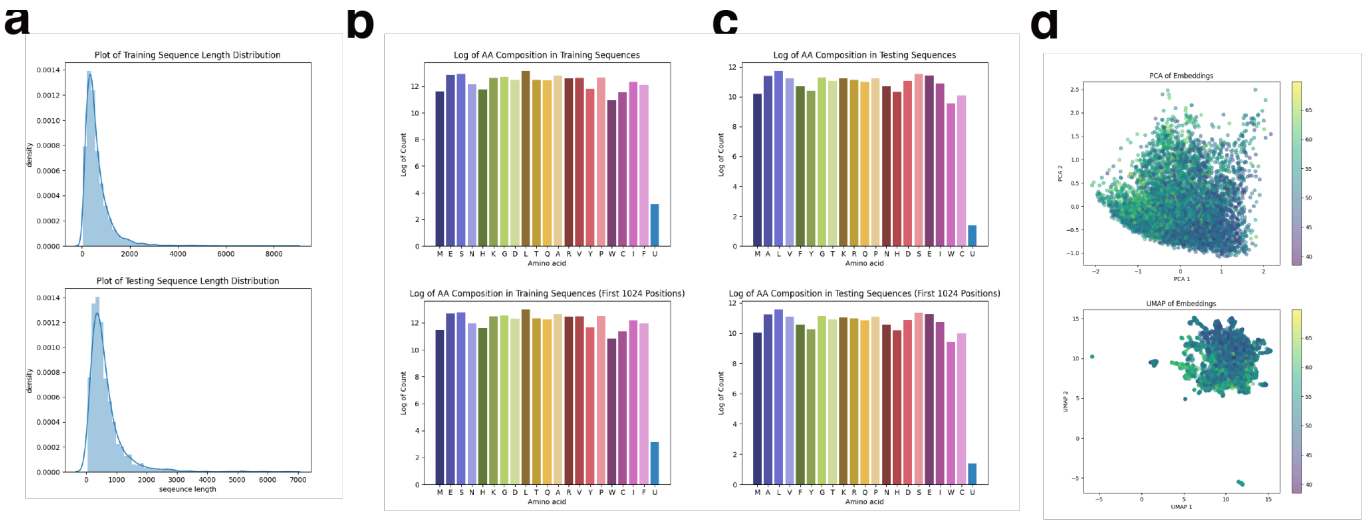

Figure 1: Approaches for Data PreProcessing and Visualization.(a) Histogram showing the sequence length distribution on the training (top) and test (bottom) sets (b-c) Barplot showing the log number of times the amino acid occurs in the whole set (top) and the first 1024 bases which were used here computed on the training set (b) and test set (c)

2 Machine Learning Model

From the previous section, there is a need for machine learning models to be trained on the embeddings generated from ESM2. Several downstream transfer learning models were run on the PCA embeddings by tuning the number of principal components. This includes the following:

- KNeighborsRegressor with varying neighbors, weights, algorithm (ball tree, kd tree and brute), and leaf size
- Support Vector Regression with varying C, kernel, degree and gamma.

- RandomForestRegression with varying number of estimators, model criterion, maximum features, split and samples in a leaf and
- Multi Layer Perceptron regression with 1-3 hidden layers of various sizes, tuning the number of hidden nodes in each layer, utilizing relu and tanh activation functions and the sgd / adam optimizer. I also tuned the regularization term, model learning rate (constant or adaptive) and the maximum number of iterations.

  Models were run with scikit-learn.

3 ML model training

I initally split my training data into 80% training, 20% validation. However, I later decided to use scikit-learn's GridSearch Cross Validation and ran it on the entire training set with the models described in the previous section. I created a pipeline for PCA and the model, defined my grid comprising machine learning models described in the previous section and their tunable parameters and fit the training data and fitness labels with cross validation. I subsequently used the best estimator. All steps were performed locally without gpu on an Apple M2 Max MacBook Pro with memory of 32GB. Use of gpu would lower the runtime.

4 Hyperparameter selection

I tried 60 and 150 principal components. For the other hyperparameters, I tuned them with GridSearch Cross Validation from scikitlearn.model_selection and using a pipeline. I set the evaluation to be Spearmann's correlation since that was used in the Kaggle evaluation. A total of 108 candidate models alongwith their parameters were tested with 5 fold cross validation totalling 540 fits. A summary of all the tests is available in the all_results.csv file submitted along with the code. Table 1 contains an overview of the top performing methods for each algorithm tried and their mean performance on the test set during cross validation.

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_model	param_model	param_model	param_model	param_model	param_model	param_model	param_model	param_model	param_model	param_model	param_model	param_model	param_model	param_model	param_model	param_model	param_model	params	split0_test_score	split1_test_score	split2_test_score	split3_test_score	split4_test_score	mean_test_score	std_test_score	rank_test_score
7	0.13612837	0.0051514	0.8070769	71 0.0082054	KNeighborsR	ball_tree	15	10	2	distance													{'model': 'KNe	0.69827477	0.73529475	0.71180451	0.71151141	0.92985485	0.75734806	0.0870745	1
23	0.15455365	0.0106856	0.7957596	0.0283284	KNeighborsR	kd_tree	15	10	2	distance													{'model': 'KNe	0.69807329	0.73501940	0.71232782	0.71281704	0.93013335	0.75727419	0.0871435	2
39	0.12735438	0.0105421	0.1098592	71 0.0080532	KNeighborsR	brute	15	10	2	distance													{'model': 'KNe	0.69809140	0.73506276	0.71127186	0.71244420	0.92929598	0.75723324	0.0868486	3
58	5.67708668	2.68869587	0.4721852	77 0.5147181	3 SVR()						10	3	scale	rbf										0.67516666	0.70941784	0.71554369	0.71721230	0.82754595	0.72897729	0.0516	25
60	1.35452680	0.12473013	0.0157904	1 0.0034255	RandomForestRegressor()										squared_erro	sqrt	1	5	20				{'model': 'Ran	0.67873749	0.69521105	0.69359831	0.67776852	0.89440161	0.72784339	0.08354411	26
61	2.28799142	0.11611109	0.0130335	3 0.0013346	RandomForestRegressor()										squared_erro	sqrt	1	10	20				{'model': 'Ran	0.67473617	0.70785249	0.69087577	0.68343377	0.87335041	0.72604972	0.0744514	27
72	24.2139629	2.64551532	0.0623242	3 0.0951269	RandomForestRegressor()										absolute_erro	log2	1	5	20				{'model': 'Ran	0.67624868	0.69161258	0.69073419	0.67654059	0.88693291	0.72441379	0.08152844	28
57	0.96355366	0.85811073	0.7191894	60 0.0647249	SVR()						10	3	scale	poly									{'model': 'SVR	0.6331098	0.66010682	0.66618987	0.67989191	0.80703265	0.68926621	0.0608164	42
54	5.51172223	5.38493058	5.3825715	5 1.1183822	3 SVR()						1	3	scale	rbf									{'model': 'SVR	0.65675892	0.6789529	0.69212826	0.68770207	0.72345284	0.68780101	0.02159815	43
89	0.85576844	0.71203066	0.0067424	0.0012559	MLPRegressor()															relu	0.001	(32, 16)	constant	500	sgd	{'model': 'MLP	0.64604387	0.65251880	0.67387926	0.66759257	0.780507	0.6841083	0.0492278	44
78	0.96614332	0.48879636	0.0118798	2 0.008393	MLPRegressor()															relu	0.0001	(32,)	adaptive	500	adam	{'model': 'MLP	0.65259933	0.66561558	0.6788818	0.66888575	0.75222677	0.68364186	0.0353069	45
84	5.84662208	0.0786784	0.0067109	0.0027781	MLPRegressor()															relu	0.001	(32,)	constant	500	adam	{'model': 'MLP	0.6508825	0.66915582	0.65878349	0.68164020	0.75651649	0.68339570	0.03798782	46

Figure 2: Cross validation results of the top 3 ranked models for each machine learning method applied, namely, KNN-Regression, Suport Vector Regression, Random Forest Regression and Multi Layer Perceptron Regression

5 Results

I achieve high cross-validation score (0.76 spearmann correlation) with 60 component PCA and a KNeighborsRegressor using ball tree algorithm, leaf size = 15, 10 neighbors, and weights = distance. I initially got but very poor test correlation (0.02 with the best fit model) but realized this was because my test results were permuted. After fixing this, I got 0.64 test set spearmann correlation.

6 Discussion

It seems that transfer learning on the ESM2 PLM generated embeddings have generated reasonable test set Spearmann Correlation on a simple k-nearest neighbor regression model which even outperformed multi-layer perceptron regression during cross validation. It would be interesting to investigate more advanced models such as VAEs, CNNs, or LSTMs trained on the ESM2 embeddings.

I briefly explored the possibility of using a pre-trained thermostability model EsmTherm[1] on this data. EsmTherm was trained by fine-tuning ESM2 using data from [3]. Applying EsmTherm directly on our Kaggle challenge data yielded poor performance. This is because EsmTherm is trained to predict delta_g which is different from the stability measurement in our data. Further, the data in [3] is generated differently from our data. I explored the possibility of fine-tuning $esm2\_t6\_8M\_UR50D$ with data from [3] using a LoRA based approach and then using transfer leaning on our dataset with these fine-tuned emebeddeings. However, due to computational and runtime constraints, I could not complete this step though I think it would considerably improve performance.

Code and embeddings have also been uploaded to github at https github•commacrinalobo2024MLCB_thermostability.

References

[1] Chu, K. S., and Siegel, J. B. Protein stability prediction by fine-tuning a protein language model on a mega-scale dataset. *bioRxiv* (2023), 2023–11.

[2] Lin, Z., Akin, H., Rao, R., Hie, B., Zhu, Z., Lu, W., Smetanin, N., dos Santos Costa, A., Fazel-Zarandi, M., Sercu, T., Candido, S., et al. Language models of protein sequences at the scale of evolution enable accurate structure prediction. *bioRxiv* (2022).

[3] Tsuboyama, K., Dauparas, J., Chen, J., Laine, E., Mohseni Behbahani, Y., Weinstein, J. J., Mangan, N. M., Ovchinnikov, S., and Rocklin, G. J. Mega-scale experimental analysis of protein folding stability in biology and design. *Nature 620*, 7973 (2023), 434–444.