Protein Family Classification Report

Yvann Vincent

August 5, 2024

Contents

1 Introduction

The goal of this project is to classify protein sequences into their respective families using various machine learning models. This report details the methodology, technical stack, and results obtained from my experiments.

2 Existing Methods and Documentation

2.1 Baseline NLP Techniques

Initial approaches to protein classification leveraged symbolic rule-based methods [MZK09] to baseline NLP methods. Techniques such as Bag of Words and n-grams were popular due to their simplicity and effectiveness in capturing local patterns. On one hand, bag of words models represent sequences as a collection of individual amino acid occurrences, disregarding order but capturing the presence of key residues [CSM11]. On the other, N-grams extend this by considering contiguous sequences of amino acids, thus incorporating some degree of sequential information. These methods, while useful, were limited in their ability to capture complex dependencies and long-range interactions within protein sequences.

2.2 Recurrent Neural Networks

RNNs and their variants like Long Short-Term Memory (LSTMs, GRU too although not as prominent), are designed to handle sequential data by maintaining hidden states that capture information from previous time steps. This allows them to model dependencies across longer sequences, making them well-suited for protein classification tasks where contextual information is crucial [Liu17]. However, they often struggle with very long sequences due to issues like vanishing gradients and computational inefficiency.

2.3 Transformers

Transformers have revolutionized both NLP and sequence processing by addressing the limitations of RNNs. The Transformer architecture relies on self-attention mechanisms to capture dependencies across entire sequences in parallel. This approach not only improves computational efficiency but also enables the modeling of long-range interactions more effectively. The success of Transformers in NLP naturally extended to bioinformatics, leading to the development of specialized models for protein sequences [EHD$^+$21]. Notable among these are ProtBert and the ESM series of models [ZLWF24].

While ProtBert is relatively large compared to the compute power I have at my disposal, ESM-2 exists in a 8M parameters version that could be quantized and fine-tuned using QLoRA, making it a suitable solution to the problem.

3 Dataset Analysis Summary

The dataset used for this project is the Pfam seed random split from the Google AI Kaggle page, containing protein sequences categorized into various families. Here is a summary of my analysis of the dataset, that you can take a look at in detail in the data analysis notebbok:

3.1 Class Distribution

The dataset exhibits a highly imbalanced class distribution among 17930 classes, with some having significantly more samples than others. The most represented types of classes in the dataset only have 2 representatives, making a three-way split delicate for proper training. In the same line of thought, the most common types of classes only have few representatives, which could make convergence to a local minimum long in training.

I made sure to regroup the data and use a stratified split to ensure that at minima the validation set and testing have have one representative of each class. I took the position that it is most important to evaluate the model in real life conditions (i.e on all possible classes).

3.2 Sequence Lengths

Although median and average length is in the low one-hundreds, protein sequences vary in length greatly, necessitating padding or truncation during preprocessing. We used a maximum length of 256 for uniformity across models. This allowed to both capture enough context for the classification task, as well as have computationally efficient training.

4 Suggested Methods and Technical Stack Choice

Given the complexity of protein sequences and the imbalance in the dataset, we opted for a combination of transformer-based models and simpler baseline models.

With these suggested methods, my main goal is to determine wheter:

- Using more computationally expensive methods such as fine-tuning a large langage model on this task will yield significantly better results than a simpler baseline, justifying the use of such methods.
- Contextually compare both methods with compute times of similar magnitude. Performance at the cost of significantly higher compute demand and training time would weigh down the relevance or more complex methods for low compute environments.

4.1 Model Selection

- **Baseline**: A simple linear classifier trained through gradient descent using bag of words features.
- **ESM2-8M, 8-bit quantized**: Fine-tuned using QLoRA to adapt the pretrained ESM2 model to our specific classification task.
- **Custom Transformer with Rotary Position Embeddings**: I have customly implemented this model as a compromise solution between the two previous methods, but unfortunately haven't had time to full integrate it with my DataModules and train it. Therefore I won't be able to report results on this model.

4.2 Technical Stack

- **PyTorch Lightning**: Chosen for its flexibility, ease of use and modular nature, allowing us to manage training loops efficiently and leverage advanced features such as gradient clipping and gradient accumulation with little effort in implementation. Although I used the transformers library to download ESM-2, I fine-tuned it using a custom pytorch lightning wrapper for better code homogeneity.
- **Hydra**: Managing training hyperparameters.

Each model was trained while monitoring results on the validation set to prevent overfitting, and the training results were logged to my personnal WandB account.

5 Results

As a thumb rule, I defined 1h of training loop compute time to be the ceiling of acceptable compute before comparing results between methods.

For context, I arbitrarily set this upper bound after trying to fine-tune ProtBert for the sake of the project, until I realized that even quantized the training loop would take 35h alone to complete for a single epoch on a T4 GPU from Google Colab.

This prompted my switch to the ESM models from META, whose smallest 8M variant showed an acceptable 55 minutes training loop time for an epoch once quantized.

Here are the results I obtained from evaluating each method on the testing set with a compute training of 1h for the training loop :

Model	**Accuracy**	**F1**	**Precision**
Linear Classifier	0,136	0.193	0.598
ESM-2 QLoRA	0.848	0.893	0.971

Table 1: Model Performance on the testing set

Although the ESM-2 fine-tune performs significantly better for comparable compute, the surprisingly high precision of both models seems striking at first. Precision being the ratio of true positives among all positive inferences for a class, this indicates that the model make few mistakes when they assign class labels, but it doesn't necessarily mean it's identifying all instances of each class. In other words it tends to be conservative in its predictions.

Intuitively, I believe this inflated precision is a good indicator of the model performance but should also be considered in the context of the severe dataset imbalance shown in the data analysis with large class count. Indeed, a minority of classes (those with only 2 representatives) were not present in my training dataset because I chose to prioritize model evaluation. Thus, the model rarely predicts the minority of classes outside it's hasn't seen during training, which is reflected in the high precision score.

F1-score, as an average of precision and recall, should be considered first to evaluate the performance of the model on the testing set.

6 Conclusion

Based on results, the absolute prediction quality of the models can be discussed because of the gap in the training set indicated by inflated precision. However, results also undeniably show better performance for the ESM-2 fine-tune compared to my baseline linear classifier (0.89 vs 0.19 F1).

Therefore, for protein classification tasks such as the one from the PFAM dataset, the fine-tuning of predictive models pretrained on similar tasks is both more cost-effective and effective than the use of simpler baseline techniques. If compute ressources allow it, standing on the shoulders of giants is strongly advised.

References

[CSM11] Cornelia Caragea, Adrian Silvescu, and Prasenjit Mitra. Protein sequence classification using feature hashing, 2011.

[EHD$^+$21] Ahmed Elnaggar, Michael Heinzinger, Christian Dallago, Ghalia Rihawi, Yu Wang, Llion Jones, Tom Gibbs, Tamas Feher, Christoph Angerer, Martin Steinegger, Debsindhu Bhowmik, and Burkhard Rost. Prottrans: Towards cracking the language of life's code through self-supervised deep learning and high performance computing, 2021.

[Liu17] Xueliang Liu. Deep recurrent neural network for protein function prediction from sequence, 2017.

[MZK09] Eghbal G. Mansoori, Mansoor J. Zolghadri, and Seraj D. Katebi. Protein superfamily classification using fuzzy rule-based classifier, 2009.

[ZLWF24] Fan Zhang, Jinfeng Li, Zhenguo Wen, and Chun Fang. Fuspb-esm2: Fusion model of protbert and esm-2 for cell-penetrating peptide prediction. *Computational Biology and Chemistry*, 111:108098, 2024.