Protein Family Accession Prediction Using Sequences from the PFAM Dataset

Harry Wilkinson

March 2023

The goal of this project was to classify proteins based on their sequences. We explored different architectures, including feed-forward networks, RNNs, LSTMs, and transformers, and found that using an embedding layer and LSTMs resulted in significant performance gains. Due to computational resource limitations, we used a reduced dataset with fewer classes, but we also investigated methods to deal with class imbalance through oversampling. We found that oversampling with a square root of the inverse popularity was the most effective method.

1 Data Analysis

The PFAM dataset [1] was organized into three datasets: train, dev (validation), and test, each containing information about protein sequences and their corresponding family accession. We conducted data analysis in the accompanying Jupyter notebook, where we examined the dataset splits, class imbalances, sequence lengths, and amino acid distributions. The notebook is annotated, so we will not delve into further detail here.

2 Method

Our goal is to develop a model capable of accurately classifying proteins based on their sequences. However, the provided dataset consists of over 17,000 unique classes, which are unevenly distributed. Even with a balanced distribution, handling this many classes would require computational resources beyond the scope of our project. Therefore, we decided to limit the number of classes to 100, which allows us to explore different methods and models without excessive hardware usage. Nonetheless, the imbalanced class distribution within the dataset remains a challenge to overcome and we will explore methods for dealing with imbalanced distributions later.

2.1 Feature Engineering

To obtain the data in a usable format, we began by removing samples from the training dataset whose classes were absent in the validation or test sets. This step ensured that the model learned to classify within the available classes. This led to a 2% reduction in the training dataset, which was an acceptable tradeoff. Consequently, we obtained a total of 13,071 classes, equally distributed across all datasets.

Next, we trimmed the sequence lengths to 128. Research suggests that protein domains are typically between 50 and

200 amino acids (AA) long, with an average length of 100 AAs, and 90% of domains being less than 200 AAs long [2]. By using a sequence length of 128, we enabled our models to learn sufficient information about the structure without requiring excessively large models.

The distribution of amino acids in the sequences was imbalanced, with five of them - "X", "U", "B", "O", and "Z" - appearing very rarely in the data. To account for this, we combined these amino acids into a single representation.

2.2 Dataset Creation

The goal was not only to build a model that accurately classifies proteins based on their sequences, but also to address the issue of imbalanced data. When the distribution of classes in the dataset is heavily skewed, machine learning models are prone to misclassification [3]. To overcome this, various methods are available to artificially balance the distribution. We experimented with different oversampling techniques, which we will discuss later.

To create datasets with varying levels of imbalance we selected 100 classes of varying levels of "popularity" for each of the five train|val|test dataset splits used in training. In this sense, popularity is a measure of the number of samples per class in the dataset. The classes selected for inclusion into each dataset split were picked starting with the most popular, moving to the least popular, with the increment between classes being values of 1, 3, 10, 30 and 100. Therefore, the dataset split with an increment of 1 simply had the 100 most popular classes, whereas the dataset split with the increment of 100 had the most popular, the 101st most popular, all the way through to the 10,001st most popular class.

These will henceforth be known as Iter1, Iter3, Iter10, Iter30 and Iter100, where the iteration factors were 1, 3, 10, 30 and 100, respectively.

2.3 Oversampling

Oversampling refers to the practice of sampling underrepresented data more frequently, which helps to balance the dataset artificially. However, oversampling can also result in overfitting for the less common datasets, since to match the frequency of common classes, the samples in an uncommon class must be resampled many times. We examined the impact of overbalancing using different weightings.

Firstly, we applied a simple form of oversampling called "regular" weighting. This method involves selecting samples inversely proportional to their frequency, such that a less common class is more likely to be selected for training. Specifically, if a class has 10 times fewer samples than another class, it is 10 times more likely to be selected for training. Let W_i denote the weighting for class i, S_i denote the number of samples for class i, and n denote the total number of classes.

$$W_i = \frac{S_i}{\sum_{j}^{n} S_j} \tag{1}$$

Secondly, we apply a square root to the calculation above. This "flattens" the weightings, making it more likely that less common classes will be sampled, while also reducing the potential overfitting that comes with the weighting method described above. This method will be referred to as "square root oversampling. Mathematically, the square root weighting can be calculated as follows:

$$W_i = \sqrt{\frac{S_i}{\sum_j^n S_j}} \tag{2}$$

Finally, we employed an equation from a research paper presented at CVPR'19 by Google [4]. This re-weighting approach depends on the "effective number of samples" concept, and it involves introducing a parameter β in the following equation:

$$W_i = \frac{1}{E_{S_i}} \tag{3}$$

$$E_{S_i} = \frac{1 - \beta^{S_i}}{1 - \beta} \tag{4}$$

3 Models

As we progressed with our training, we transitioned from using simple feed-forward neural networks to more appropriate architectures for our task. It was an interesting experiment to see how different models performed, rather than assuming that the best model would be the one suggested by the literature and going straight for that option.

Fully Connected Feed-Forward Network

The simplest model we used was a fully connected feed-forward network. We explored different hyperparameters and we were able to show that the model was learning patterns in the data and making predictions. However, this type of model isn't suitable for processing sequence information because they lack the ability to maintain temporal information. Feed-forward networks process input data in a fixed order, with no feedback or memory.

Recurrent Neural Network (RNN)

The next model we used was a recurrent neural network (RNN). Unlike feed-forward networks, RNNs are designed to process sequence information and maintain an internal state that captures information about previous inputs. This state is updated at each time step and can be used to inform the processing of subsequent inputs, making them more suitable for capturing long-term dependencies in the input sequence.

Long Short-Term Memory (LSTM)

LSTMs are a type of RNN designed to address the problem of vanishing gradients [5]. The memory cell introduced in LSTMs can store information over time, and three gates control the flow of information out of the cell. This design allows LSTMs to better handle long-term dependencies compared to standard RNNs. We explored both one-directional and bi-directional LSTMs with the assumption that the bi-directional model would be superior since sequences themselves are bi-directional.

Transformers

Transformers were initially introduced in 2017 for machine translation [6], but have since been successfully applied across various fields in deep learning, and currently represent the state-of-the-art in many domains. The self-attention mechanism in transformers allows them to focus on the most relevant elements within the input. However, unlike LSTMs and RNNs, transformers do not receive inputs sequentially, which requires the use of positional embeddings to pay attention to the order of the input.

As transformers are the current state-of-the-art in NLP, they were the natural choice for our best architecture. We used only transformer encoder blocks with a classification head. Typically, encoder architectures are more suited to classification tasks, while decoder architectures are used for generation tasks, such as in alphafold, which models the structure of a protein.

3.1 Embedding

In NLP, embedding allows us to represent words or phrases as dense, continuous vectors in a high-dimensional space. This allows us to capture semantic relationships between words, such as their relatedness or similarity. Embeddings have been shown to be superior to one-hot encoding or other traditional sparse representations [7].

Similarly, in the classification of proteins using their sequences, embedding allows us to capture the functional and structural relationships between amino acids. We first include embeddings in the RNN architecture and then in all architectures following that.

There are pre-trained protein embeddings such as UniProt and ProtBert, however, we opted not to use pre-trained embeddings and instead trained our embeddings as part of the model.

4 Results

4.1 Architectures

We trained a series of architectures, including a fully connected feed-forward network, an RNN with embedding, an RNN with one-hot encoding, an LSTM, a Bi-LSTM, and a transformer-based model. All models were fine-tuned, and their test loss and accuracy scores were evaluated on Iter1.

Table 1: Architecture Results on Iter1

	FC	***RNN***	***RNN (Embed)***	***LSTM***	***Bi-LSTM***	***Transformer***
Loss (e-3)	30.78	17.06	10.77	0.95	**0.79**	1.05
Accuracy (%)	50.66	75.25	83.08	98.21	**98.73**	98.22

The Bi-LSTM model achieved the highest accuracy of 98.72% on the test dataset, while the transformer and LSTM models had similar performance at 92.22% and 98.21% accuracy respectively.

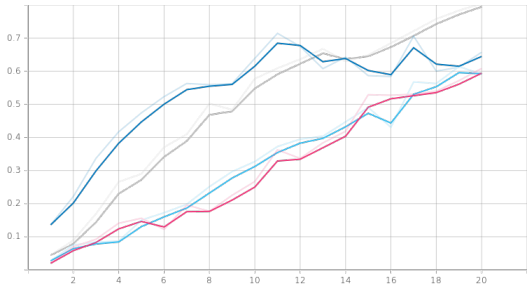
4.2 Oversampling Methods

We evaluated all oversampling methods using a transformer model with consistent hyperparameters. For each experiment, we used a dataset consisting of 100 unique classes, with varying class distributions in each dataset: Iter1, Iter3, Iter10, Iter30, and Iter100. As the classes become less common, the dataset sizes decrease, with Iter100 being roughly seven times smaller than Iter1. All models were trained for 20 epochs and validated using the "regular" oversampling method, which ensures that all classes are equally likely to appear. This method is suitable for evaluating the model's performance in learning to classify all classes.

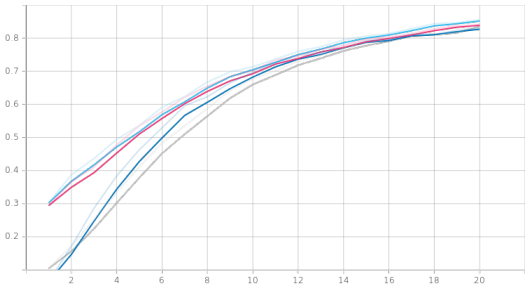
Table 2: Oversampling Results with a Transformer (loss is e-3)

	None		**Regular**		**Square Root**		**Effective**	
	Loss	**Accuracy (%)**	**Loss**	**Accuracy (%)**	**Loss**	**Accuracy (%)**	**Loss**	**Accuracy (%)**
Iter1	**0.99**	96.19	1.29	**97.50**	1.43	97.29	1.43	**97.50**
Iter3	1.72	96.64	1.86	96.42	**1.34**	**97.56**	1.50	97.21
Iter10	4.45	91.91	4.95	91.51	**4.37**	**92.23**	4.80	91.21
Iter30	14.78	71.95	**12.25**	**78.83**	12.61	77.66	17.25	65.93
Iter100	25.76	54.50	19.58	64.00	**14.61**	**75.92**	21.38	61.51

The square root oversampling method proved to be the most consistently effective among all oversampling techniques tested. This method significantly outperformed the others when applied to the Iter100 dataset, where the transformer-based model achieved impressive results. Notably, this model's performance surpassed that of the RNN model trained on the Iter1 dataset, despite the significant differences in class distribution and dataset size.



Validation Accuracy



Train Accuracy

Figure 3: The x-axis is epochs. The y-axis is the percent of predictions that were correct. The light blue line is without oversampling, the dark blue line is with regular oversampling, the grey line is with square root oversampling, and the pink line is with effective oversampling.

In figure 3, we have graphs showing the validation loss and validation accuracy as the models were trained on Iter100. Arguably, the results from this were the most interesting due to this being the dataset with the greatest imbalance.

5 Discussion

The results of our experiments show that the Bi-LSTM architecture with embedding was the most effective in accurately classifying protein sequences, but the transformer-based model also performed well, especially when trained on a dataset with

a highly imbalanced class distribution. The oversampling experiments showed that the square root oversampling method was the most consistently effective in improving the model's performance, especially on the highly imbalanced Iter100 dataset.

One limitation of our study is the reduced number of classes in the final dataset, which was necessary due to computational resource restrictions. However, the study provides valuable insights into the performance of different neural network architectures in the domain of protein sequence classification, and demonstrates the effectiveness of oversampling techniques in mitigating class imbalance issues. Future research could focus on developing more advanced oversampling methods, as well as exploring the use of other neural network architectures and incorporating additional features or data sources to improve classification performance.

Model Architectures

The LSTM, Bi-LSTM, and transformer models showed similar performance, but the Bi-LSTM had a slightly lower test loss and higher test accuracy. Notably, two architecture changes resulted in significant performance increases: using an embedding layer and incorporating LSTMs in the RNN. This suggests that protein sequence information can be effectively learned through embedding, and the LSTM's ability to capture long-range dependencies can greatly improve model performance in this domain.

Given more time and computational resources, it would be interesting to further explore the performance differences between Bi-LSTM and transformer architectures using a larger dataset with more classes.

Dealing with Imbalanced Distributions

There are various approaches to address class imbalance, and we investigated several oversampling techniques on datasets with different levels of imbalance. Our experiments showed that oversampling using the square root of the inverse popularity, where popularity is the number of occurrences divided by the total sample size, resulted in the most accurate model.

In future studies, it would be interesting to explore the use of loss functions that account for the scarcity of a class in a dataset and penalize the model for misclassifying less common classes. Additionally, to utilize other evaluation metrics beyond accuracy, such as the F1 score, to better assess the model's performance, especially with respect to the lesser seen classes.

References

[1] Robert D. Finn et al. "Pfam: the protein families database". In: *Nucleic Acids Research* 42.D1 (Nov. 2013), pp. D222–D230. ISSN: 0305-1048. DOI: 10.1093/nar/gkt1223. eprint: https://academic.oup.com/nar/article-pdf/42/D1/D222/3643441/gkt1223.pdf. URL: https://doi.org/10.1093/nar/gkt1223.

[2] Milo M. Lin and Ahmed H. Zewail. "Hydrophobic forces and the length limit of foldable protein domains". In: *Proceedings of the National Academy of Sciences* 109.25 (2012), pp. 9851–9856. DOI: 10.1073/pnas.1207382109. eprint: https://www.pnas.org/doi/pdf/10.1073/pnas.1207382109. URL: https://www.pnas.org/doi/abs/10.1073/pnas.1207382109.

[3] J.M. Johnson and T.M. Khoshgoftaar. "Survey on deep learning with class imbalance". In: *Journal of Big Data* (2019). DOI: 10.1186/s40537-019-0192-5. URL: https://doi.org/10.1186/s40537-019-0192-5.

[4] Yin Cui et al. *Class-Balanced Loss Based on Effective Number of Samples*. 2019. arXiv: 1901.05555 [cs.CV].

[5] Sepp Hochreiter and Jürgen Schmidhuber. *Long Short-Term Memory*. 1997. URL: https://www.bioinf.jku.at/publications/older/2604.pdf.

[6] Ashish Vaswani et al. "Attention Is All You Need". In: *Advances in Neural Information Processing Systems*. 2017, pp. 5998–6008.

[7] Dahouda Mwamba and Inwhee Joe. "A Deep-Learned Embedding Technique for Categorical Features Encoding". In: *IEEE Access* PP (Aug. 2021), pp. 1–1. DOI: 10.1109/ACCESS.2021.3104357.