Data Challenge Report

Protein Cellular Component Ontology Prediction

MVA - Advanced Learning for Text and Graph Data

Team: Peptide Bond 007

Denis Duval
denis.duval13@gmail.com

Erwan Fagnou
erwan.fagnou@gmail.com

January 2023

TABLE OF CONTENTS

Introduction

As part of the Advanced Learning for Text and Graph Data (ALTEGRAD) course, we took part in the Kaggle challenge on Cellular Component Ontology, made for the students following the course [1]. This challenge is an opportunity for us to apply and deepen the machine learning and artificial intelligence techniques learned throughout the course and test their performance on a data set made for this occasion. This challenge is also an opportunity for us students to manipulate data specific to bio-informatics i.e. protein data, and to explore their potential through state-of-the-art techniques.

In the field of bioinformatics, machine learning has been gaining attention for its potential to understand the complex nature of proteins. These biomolecules, made up of chains of amino acids, play a crucial role in the functioning of all living organisms. They act as catalysts for chemical reactions, provide structural support, and play a key role in the immune system's ability to distinguish self from non-self. However, the exact mechanisms behind how a protein's sequence, structure, and function are related are still unclear.

1 The dataset

1.1 Content

The challenge involved classifying proteins to 18 different functions, using both the sequence of amino acids and a graph representation of their 3D structure. Among the 6111 proteins of the dataset, 4888 are labeled and hence constitute the training data while the unlabeled remaining part constitute the test data on which we have to make predictions. The latter is the data our models are evaluated and ranked in the challenge's leaderboard.

For each of these proteins we are given the following information:

- The sequence of amino acids. Note that the order of the sequence is also an information and a permutation of the sequence may not result in the same function of the protein.
- The features of the amino acids (there are 86 in total): 3D position, type (as a one hot vector), hydrogen bond acceptor and donor status, and more features derived from the EXPASY protein scale [2].
- The edges of the protein graphs.
- The features of the edges of the graphs: the distance between the two connected nodes, and the type of edge (peptide bond, hydrogen bond, k-NN edge, or distance-based edge).

This information is contained in the raw .txt files we are given as data for the challenge. A preprocessing of these raw files has to be carried out in order to feed the models developed with the data. This preprocessing has been done only once (we saved the preprocessed data with pickle) in order to save some computational energy and time and hence be more efficient while testing our models.

1.2 Data analysis

Figure 1 below shows that the data set is strongly unbalanced. There are 5 of the 18 classes that represent 2/3 of the training data. We also observe in Figure 2 that amino acid types are also unbalanced, and that especially the "X" type appears in extremely few proteins, which may cause some overfitting.

Figure 1: Distribution of the samples in each class. Some classes are marginally represented in this dataset with a factor 20 with other classes.

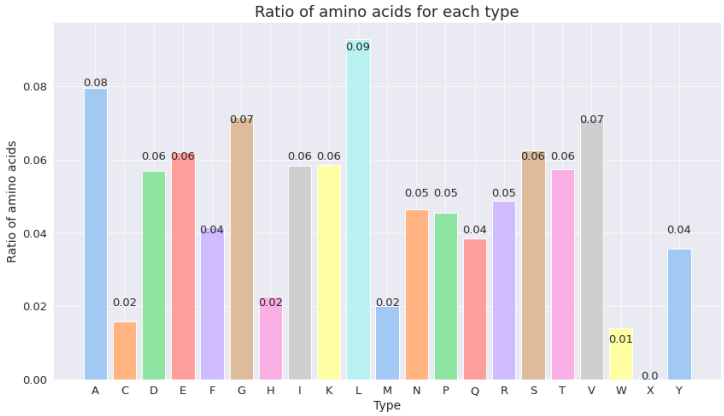

Figure 2: Ratio of amino acids types in the data set

	Min	0.25-quantile	Median	Mean	0.75-quantile	Max
Nodes	9	129	221	257	340.5	989
Edges	55	2187	3996	4721	6491	20417

Table 1: Statistics on the number of nodes and edges of the protein graphs.

When analysing the distribution of the number of nodes and edges in graphs (Table 1), we observe that there is a strong discrepancy in the dataset with graphs being very simple and others way more complex.

In spite of being able to visualize a protein graph in 3D with its attributes, we tried to retrieve an information on the shape taken by the protein by quantifying the volume that it takes in space. An interesting quantity is the radius of the smallest ball that contains all the nodes of the graph, which are shown in Figure 3. Overall the proteins have the same spatial dimension. There are non negligible variations but it is also due to the fact that some classes are more prevalent than others.

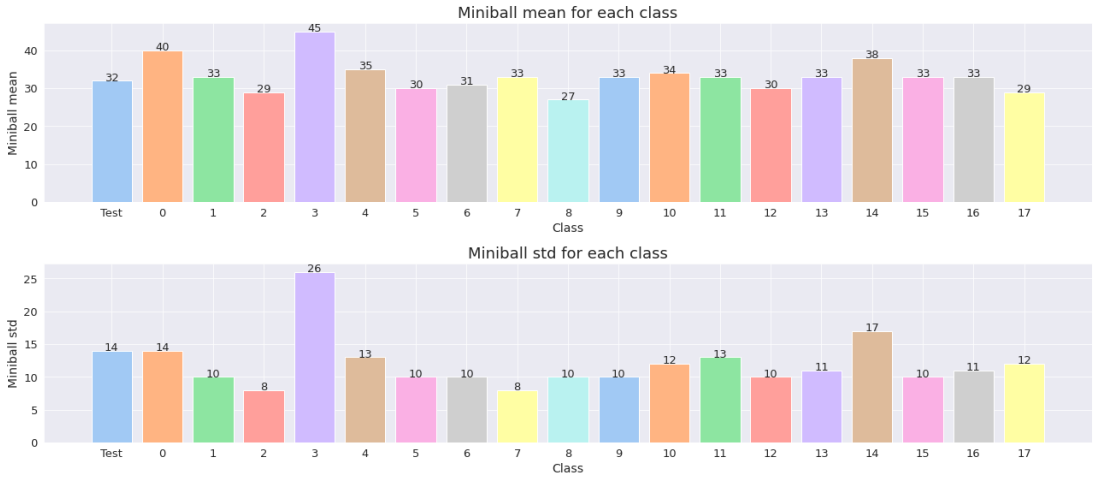

Figure 3: Mean and Standard deviation of the minimum radius of a ball containing the protein.

2 Our models

Because the dataset was very small, our models were prone to a lot of overfitting, so we took 500 proteins from the 4888 training samples to make a validation dataset. This helped us a lot to compare our different models, as the training loss was not representative of the real performance.

The size of our models were also reduced in order not to overfit, which may explain why some of the following methods did not work despite their state-of-the-art results in the literature.

2.1 Structure based methods

Our first attempts used simple graph convolution networks. As we were using the pytorch-geometric library [3], we were able to easily try more advanced state-of-the-art methods, like Graph Attention Networks [4], Attentive FP [5], and DeeperGCN [6], which are methods that also take edge features into account. This however didn't improve much the results.

We also experimented with the aggregation part of the model, which merges the states of every node to a single vector that is used for the final prediction. We observed that taking the mean was better than the sum, but more fancy aggregations didn't show great performance either.

The first improvements were obtained by enhancing the node features (which were directly taken from the dataset) with other features. Some were from the literature (Local Degree Profile [7], Laplacian Positional Embedding [8]) while others were designed by us for the special case of proteins. Indeed, after noticing that there often are some spirals in the amino acid chains, we added the torsion angle [9, 10] as a feature. We also added the distance to the center of mass of the protein (we didn't notice any difference between the Euclidean and Mahalanobis distances), and the position in the sequence. We observed improvements with every additional feature, and using all of them improved the validation loss by 8%, as shown in Table 2.

	Node features			
	default	default + extra features	LSTM	LSTM + extra features
Number of features	83	95	512	524
Validation loss	1.77	1.62	1.57	1.49

Table 2: Performance of the same GCN model using different node features. The default features from the dataset can be improved with additional features as described in section 2.1, while replacing them with a pretrained LSTM is even better.

2.2 Learning node features from the sequences

A standard approach to take the protein sequences into account is to pretrain a language model on them, to obtain embeddings for each amino acid. This has already been done, for instance with recurrent neural networks [11] or transformers [12].

We first tried to pretrain a model made of two bidirectional LSTMs superposed. We trained it on the masked language modelling task (some amino acids of the proteins were masked, and the model tried to predict them). This performed quite poorly. Then we trained a similar model on the task of predicting the next amino acid in the sequence, which led to more promising results. Indeed, not only was the model able to predict the next amino acid with good accuracy, but the embeddings it learned (the hidden states of the two LSTMs) significantly improved the loss of the GCN on the validation data by 11% (see Table 2). An interesting observation is that the LSTM was better when reading the sequences in one direction (left to right) compared to both directions (randomly reading from left to right or right to left), which suggests that proteins have a specific orientation.

A more recent approach for learning amino acid embeddings in proteins is ESM-2 [12], which is a modified BERT encoder. By reducing the dimensions and number of layers, we were able to train it (from scratch) on the dataset without overfitting. However the hidden states of this model did not seem to carry a lot of relevant information, as the validation loss only slightly improved when using these as node features in a GCN.

2.3 Using the pretrained ESM-2 embeddings

Despite the poor results of our own ESM-2 model, it ended up being a game-changer when we used the pretrained models published by its authors. Simply training a 2-layer MLP on the last state of the first token (which corresponds to the <bos> token) lowered the validation loss to 1.35, and using the concatenated hidden states of every layer as embeddings in a GCN led to 1.10. We also tried to train the parameters of ESM-2 for the classification, but this led to quick overfitting as expected.

It is important to understand that ESM-2 was initially used to predict the 3D structure of proteins [12]. This means that the hidden states are expected to contain all of the information about the spatial structure, which suggests that a GCN would not be able to extract more spatial information. We indeed observed that replacing the GCN with a multihead attention (MHA) layer performed even better. We believe using attention to aggregate the embeddings is very relevant for this task, as the function of a protein may depend on different patterns in different parts of the protein, and because the hidden states of ESM-2 are inputs of MHA layers, which guarantees a good compatibility between the hidden states and our classification head.

We managed to obtain a score of 0.68 on the Kaggle platform using this approach: the pretrained ESM-2 (the version with 650M parameters) to generate embeddings, followed by a multihead attention to aggregate the embeddings, and a 2-layer MLP to compute the prediction. A few interesting point can be noted about our approach and experiments:

- We observed that including the embeddings of the `<bos>` and `<eos>` tokens improved the performances.
- ESM-2 650M is made of 33 layers with a hidden dimension of 1280. We used the concatenation of the hidden states of the 2^{nd}, 12^{th}, 22^{nd} and 32^{nd} layers as the embeddings of the amino acids, but we didn't have the time to try many variations.
- Only selecting a few layers of ESM-2 for the embeddings allowed us to precompute the embeddings for the whole dataset, thus making the training itself really fast (between 10 and 15 seconds per epoch, with a convergence in around 15 epochs).

2.4 Preventing overfitting

Throughout our experiments we had to use diverse methods to prevent overfitting. We used standard techniques like dropout (with a 20% rate) and label smoothing (set at 0.05). Label smoothing is a regularization technique for multi-class classification tasks that helps to prevent overconfidence of the labels and hence allow a better generalization of the model [13]. Because our models always overfitted a lot, the models predicted extreme probabilities (close to 0 or 1), so a single mistake would cause a large loss; label smoothing addressed this issue and had a great impact on the validation accuracy. We did not manage to get better results with other methods like weight decay or Stochastic Weight Averaging [14], but they might be useful after a more careful hyperparameter tuning.

Because the node features often had a large number of dimensions, we tried reducing their size using PCA, or a learned projection (which performed better than PCA). The hidden dimension of our model played a big role its generalization abilities, and we chose 128 for all of the layers – further tuning the dimensions may have improved the results a little.

The final technique to improve the score was to train multiple models and average their predictions. We also tried to use the median instead of the average, but it was worse; once again, more fancy ensemble methods could probably boost the performance even more, but simply doing this was enough to gain a lot, as shown in Table 3. We can see that after a couple dozens of models, the performance doesn't increase a lot.

	Number of models			
	1	4	25	336
Average	0.684	0.638	0.628	0.625
Median	0.684	0.659	0.635	0.631

Table 3: Score on Kaggle after aggregating the predictions of multiple models using ESM-2 650M, either with an average or the median.

2.5 Last minute improvements

On the last day of the contest, we trained the same model using a larger version of ESM-2 (with 3 billion parameters). It has 36 layers so we used the hidden states of the 2^{nd}, 13^{th}, 24^{th} and 35^{th} for the embeddings of the amino acids. We didn't change anything except the learning rate which influenced a lot the validation loss of the model. We trained 113 such models and aggregated them as explained in the previous section. This improved the final score to 0.593 (compared to 0.625 with ESM-2 650M), and allowed us to gain the first position in the leaderboard.

Conclusion

We are very satisfied with our results, although with more time we could have tuned the hyperparameters more precisely. The use of the larger ESM-2 pretrained model with 15 billion parameters could also improve the score. Our final model is relatively simple, because more complex models overfitted too quickly, and we believe that using larger protein datasets would be necessary in order to use fancier models like our first attempts.

Bibliography

[1] Giannis Nikolentzos. Protein cellular component ontology prediction, 2022. URL https://kaggle.com/competitions/altegrad-2022.

[2] URL https://web.expasy.org/protscale.

[3] Matthias Fey and Jan Eric Lenssen. Fast graph representation learning with PyTorch Geometric, 2019. URL https://arxiv.org/abs/1903.02428.

[4] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks, 2017. URL https://arxiv.org/abs/1710.10903.

[5] Zhaoping Xiong, Dingyan Wang, Xiaohong Liu, Zhong Feisheng, Xiaozhe Wan, Xutong Li, Zhaojun Li, Xiaomin Luo, Kaixian Chen, H. Jiang, and Mingyue Zheng. Pushing the boundaries of molecular representation for drug discovery with graph attention mechanism. *Journal of Medicinal Chemistry*, 63, 08 2019. doi: 10.1021/acs.jmedchem.9b00959.

[6] Guohao Li, Chenxin Xiong, Ali Thabet, and Bernard Ghanem. Deepergcn: All you need to train deeper gcns, 2020. URL https://arxiv.org/abs/2006.07739.

[7] Chen Cai and Yusu Wang. A simple yet effective baseline for non-attributed graph classification, 2018. URL https://arxiv.org/abs/1811.03508.

[8] Vijay Prakash Dwivedi, Chaitanya K. Joshi, Anh Tuan Luu, Thomas Laurent, Yoshua Bengio, and Xavier Bresson. Benchmarking graph neural networks. 2020. doi: 10.48550/ARXIV.2003.00982. URL https://arxiv.org/abs/2003.00982.

[9] Arnaud Blondel and Martin Karplus. New formulation for derivatives of torsion angles and improper torsion angles in molecular mechanics: Elimination of singularities. *Journal of Computational Chemistry*, 17(9):1132–1141, 1996. doi: https://doi.org/10.1002/(SICI)1096-987X(19960715)17:9<1132::AID-JCC5>3.0.CO;2-T. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/%28SICI%291096-987X%2819960715%2917%3A9%3C1132%3A%3AAID-JCC5%3E3.0.CO%3B2-T.

[10] Wikipedia. Dihedral angle (In polymer physics) — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Dihedral%20angle&oldid=1131606823#In_polymer_physics, 2023.

[11] Vladimir Gligorijevic, P. Renfrew, Tomasz Kosciolek, Julia Koehler, Daniel Berenberg, Tommi Vatanen, Chris Chandler, Bryn Taylor, Ian Fisk, Hera Vlamakis, Ramnik Xavier, Rob Knight, Kyunghyun Cho, and Richard Bonneau. Structure-based protein function prediction using graph convolutional networks. *Nature Communications*, 12, 05 2021. doi: 10.1038/s41467-021-23303-9.

[12] Zeming Lin, Halil Akin, Roshan Rao, Brian Hie, Zhongkai Zhu, Wenting Lu, Nikita Smetanin, Robert Verkuil, Ori Kabeli, Yaniv Shmueli, Allan dos Santos Costa, Maryam Fazel-Zarandi, Tom Sercu, Salvatore Candido, and Alexander Rives. Evolutionary-scale prediction of atomic level protein structure with a language model. *bioRxiv*, 2022. doi: 10.1101/2022.07.20.500902. URL https://www.biorxiv.org/content/early/2022/12/21/2022.07.20.500902.

[13] Rafael Müller, Simon Kornblith, and Geoffrey Hinton. When does label smoothing help?, 2019. URL https://arxiv.org/abs/1906.02629.

[14] Pavel Izmailov, Dmitrii Podoprikhin, Timur Garipov, Dmitry Vetrov, and Andrew Gordon Wilson. Averaging weights leads to wider optima and better generalization, 2018. URL https://arxiv.org/abs/1803.05407.