Hugo DEBES
Rita-Mathilda KABRO
Vincent TCHOUMBA
first_name.last_name@polytechnique.edu

Challenge Report
ALTEGRAD 2022

01/22/23

Abstract

This document reports on the Cellular Component Ontology Prediction. The goal was to classify using its sequence structure a protein in one of the 18 classes. Each class represents a characteristic of the location where the protein performs its function obtained from the Cellular Component Ontology We aim to present how we dealt with each part of the pipeline from representation to critical analysis of our results.

1 Problem Understanding

A protein is a bio-molecule made up of one or more chains of amino acids. Proteins play a variety of roles in the body, including building and repairing tissues, transporting molecules, and catalyzing metabolic reactions. Proteins can be found in many different forms in the body, including as enzymes, hormones, and structural components of cells and tissues. Therefore, we can classify them in many different ways. For this challenge, the aim was to classify the proteins by their localization in the cell. But first, let's talk about the protein itself. Each protein is first described by sequences of amino acids. 20 different amino acids are commonly found in proteins, each with a unique side chain R.

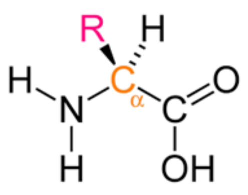


Figure 1: Amino acid

These 20 amino acids can be grouped into several categories based on their properties, including essential amino acids, non-essential amino acids, and conditional amino acids.

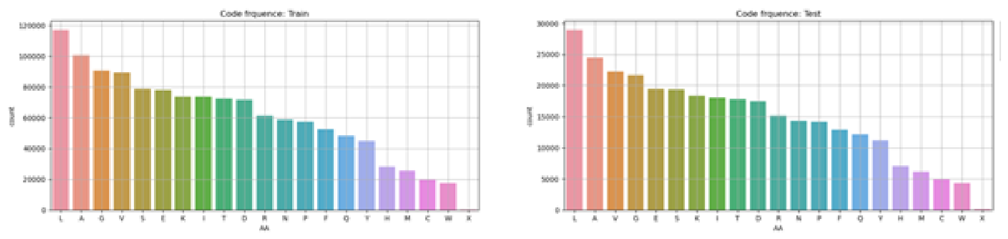


Figure 2: Distribution of amino acids in the training and the testing set

Some amino acids are a lot more represented than others, but the three different categories are well distributed in both of the datasets. Each amino acid will have a different impact on the propriety of the protein, that's why it's important to take the sequence into account when building the predictive model. The length of the sequences can vary between two proteins:

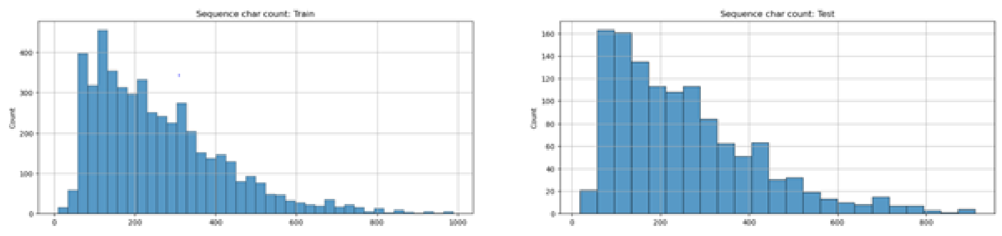


Figure 3: Distribution of sequences based on the length

In the dataset, we have proteins containing up to 900 amino acids, but most of them have a length of range 80-320 amino acids. The proteins are a very complex structure and therefore can be represented by many different features. Each amino acid is connected through peptide bonds [1]. Peptide bonds form between the carboxyl group (-COOH) of one amino acid and the amino group (-NH2) of another. When multiple amino acids are joined together in this way, they form a protein. The sequence of amino acids in a polypeptide chain determines its unique properties and functions, that's why it's also important to take into account those peptide bounds, and therefore the 3D structure of the proteins. Each protein can be classified into a class that represents its localization.

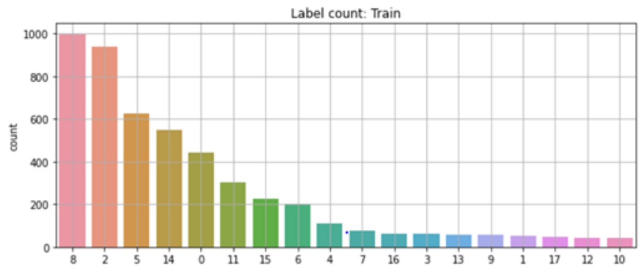


Figure 4: Distribution of classes

We can report that the dataset is quite imbalanced. Indeed, our training set is mostly composed of classes 8, 2, and 5. The problem with an imbalanced dataset is that it includes biases in our training. As we see there are a lot of aspects that we must take into consideration during the conception of our model.

2 Feature Extraction

2.1 Introduction

One of the ideas that seemed essential to us upon discovering the challenge was to use all the information we had about proteins, namely what their sequences look like and what structure they possess. Retrieving and working with one or the other was relatively simple given that we had access to two baselines, one based on the sequence and the other on the structure of the protein. The problem was to bring these two sources of information together for our model to be better able to distinguish between classes of proteins. To do this, we explored two ways: the first, starting from the protein sequence and trying to add variables corresponding to information related to its structure, and the second, starting from the structure and trying to add information related to the protein sequence.

2.2 Enhancing Representations on a Protein-Level

The first idea, whose goal is to improve the database by adding desirably significant variables in our multi-class classification problem, is purely Feature Engineering. The variables related to the structure that seemed the most important for our problem were the number of amino acids in the protein, i.e. the length of the sequence, the number of links within each protein using data from *edgelist.txt* and finally the data related to the length of each of these links contained in *edge_attributes.txt*.

This was a good start, but it did not seem sufficient to us, we wanted to add more information related to the sequence and structure of the protein. After some research, we came across a library called Protlearn whose sole purpose is to extract variables from the protein sequence and the structure of the amino acids that compose it. Among the variables that were added thanks to this library were:

- **Atc** (Atomic and bond composition): This function returns the sum of atomic and bond compositions for each amino acid sequence. The atomic features are comprised of five atoms (C, H, N, O, and S), and the bond features are comprised of total bonds, single bonds, and double bonds.
- **ctd** (Conjoint triad descriptor): Amino acids can be grouped into 7 different classes based on their dipoles and side chain volumes, which reflect their electrostatic and hydrophobic interactions.

The combination of all these variables adds expertise to our database and thus valuable information for the learning of our models.

2.3 Enhancing Representations on a Node-Level

On the other hand, we worked on the different features of each amino acid present in the protein. We already knew that several pieces of information were used by the Graph Neural Network baseline stored in the matrix *features_batch*. We noticed that from indices 4 to 23, the amino acids were one-hot encoded. It is a usual method to represent categorical data that nevertheless has some flaws:

- The vectors are highly sparse since for each of them, you have zeros except for one occurrence.
- This method does not capture the "semantic" similarity between each Amino Acid.

Thus, we decided to introduce better embeddings thanks to the pre-trained ProtBert [2] model. Trained in a self-supervised way on the UniRef dataset containing more than 200 million protein sequences, we could re-use it with Transfer learning via the Hugging Face API. The last step was to replace the one-hot encoding with the new embeddings in the features matrix while keeping an eye on the dimensionality. Limited by our computational power, we decided to apply a Kernel PCA, a dimensionality reduction algorithm, on the new vectors from a length of 1024 to 30.

3 Models selection

3.1 Integral Deep Learning Apporach

Deep learning methods represent the state of the art when it comes to many problems and biology is not an exception [4]. The availability of large banks of data has led to unprecedented improvements in protein structure prediction. For our challenge, we explored a full deep-learning approach from feature extraction to classification.

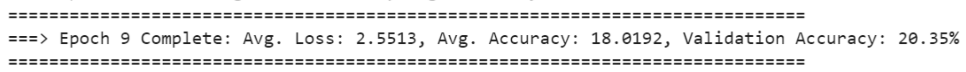
RNN - LSTM - AttentionBased

The protein is a sequence of amino acids, so it was important for us to try some sequential models. RNN (Recurrent Neural Network) and LSTM (Long Short-Term Memory) are types of neural network models that are used for processing sequential data. RNNs can utilize information from previous time steps to process current time steps. This makes them suitable for tasks such as natural language processing and speech recognition, but also study any kind of sequences.

LSTM is a variation of RNN that is specifically designed to handle the problem of vanishing gradients. It includes a memory cell, a hidden state, and gates that control the flow of information into and out of the cell. These gates allow LSTM to selectively preserve or discard information from previous time steps, thus allowing it to learn long-term dependencies in the data. The LSTM is on paper the perfect model to manage long sequences like the proteins.

The only difference between language processing and protein processing is vocabulary. Instead of doing embeddings on words, we did an embedding on characters (amino acids in this context). But before creating an embedding, it was important to do some preprocessing. The sequences can be of different sizes, the problem with that kind of model is that it's necessary to keep the same input size. That's why we created some padding on the data.

We tried different types of embeddings, from classical to pre-trained Bert, then we add some RNN and linear layers to conclude our prediction. The performances were not as good as we wanted:



```
===> Epoch 9 0/64: Avg. Loss: 2.5087, Avg. Accuracy: 21.8750%
===> Epoch 9 16/64: Avg. Loss: 2.5429, Avg. Accuracy: 17.6471%
===> Epoch 9 32/64: Avg. Loss: 2.5464, Avg. Accuracy: 17.1875%
===> Epoch 9 48/64: Avg. Loss: 2.5472, Avg. Accuracy: 17.7615%
==============================================================================
===> Epoch 9 Complete: Avg. Loss: 2.5513, Avg. Accuracy: 18.0192, Validation Accuracy: 20.35%
==============================================================================
```

Figure 5: Performances on the LSTM

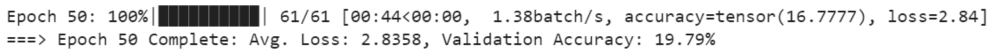
After a few epochs, the error and the accuracy have reached a plateau. The model was worse than random. We didn't want to move on from pure sequential models without trying other solutions, that's why we decide to add some attention to this model [8].

Attention models are a type of neural network that is used to process sequential data. They work by allowing the model to selectively focus on certain parts of the input sequence when making predictions.

In traditional RNNs and LSTM, the model processes the entire input sequence in a fixed order and at each time step, the hidden state of the model is updated based on the current input and the previous hidden state. However, in some situations, some parts of the input sequence may be more important than others for making a prediction. Attention models try to solve this problem by allowing the model to ”pay attention” to different parts of the input sequence at different times.

The main idea behind attention models is to assign a weight or ”attention score” to each element in the input sequence. These attention scores are used to weigh the contribution of each element to the final prediction. The model then ”attends” to different parts of the input sequence based on these attention scores.

We tried attention models on amino acids and part of the sequences. We thought it interesting to separate the sequences into three parts (beginning, middle, and ending) to see if some parts of the proteins are more important than others. Attention models can help us understand better the behavior of the proteins. Again, the performances were not as good as we expected.



```
Epoch 50: 100%|██████████| 61/61 [00:44<00:00,  1.38batch/s, accuracy=tensor(16.7777), loss=2.84]
===> Epoch 50 Complete: Avg. Loss: 2.8358, Validation Accuracy: 19.79%
```

Figure 6: Performances on the Attention Model

Models that take into account the amino acid sequences are not sufficient for our objective, that’s why we have to rely on other methods to complete our predictions.

GAT/GConv on structure

Having taken by storm the world of computer vision, CNN and more precisely Convolution have been revisited by several researchers to be used in a graph-based structure. We used the GConv layer from Kipf & Welling [5] implemented in the PyTorch Geometric library. Our goal was to use the previously computed node embeddings to extract valuable features of the structure with convolution. We used several layers to capture features on different scales taking inspiration from V. Gligorijevic’s [3] architecture. However, we did not concatenate the features matrix before as it produced erratic behavior. The classification head was composed of one layer accompanied by dropout. Unfortunately, the model overfitted and was unable to generalize to the eighteen classes. Nevertheless, we discovered many interesting approaches in the PyTorch Geometric.

Attention [6] can help the network to focus on specific parts of a sequence and as we’ve seen during the practical work specific parts of a graph. We used the Graph Attention Network developed by Veličković et al [7] Rather than the message passing layer where a node focuses equally on each of its neighbors, the GAT layer tuned attention coefficients. Using this network, we hoped to get valuable embeddings per node before aggregating them with a graph pooling operation. Our network contained three layers of attention with dropout summarised by a sum readout and batch normalization. The classification head was the same as the Graph convolutional architecture.

We used the Adam optimizer and fine-tuned its parameters to control the learning process. It was unfortunately not enough to beat the Machine Learning Approach we ran in parallel. In our opinion, several factors may explain why we struggle to tune the different deep learning algorithms. We did not balance the complexity of a model, i.e., the number of parameters, with the computational power at our disposal. This led to either hardly sustainable training time or oversimplification regarding our data that caused overfitting. We noticed how a relatively small and imbalanced nature of the dataset impacted our results. Knowing all its reasons, we focused on a different approach.

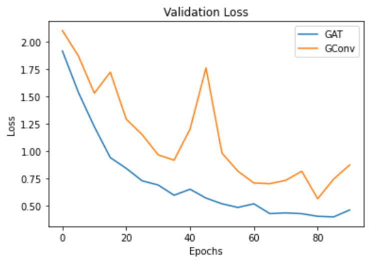


Figure 7: Validation Loss comparison between GAT and GConv

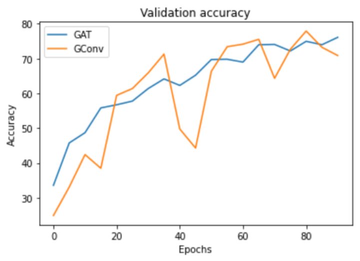


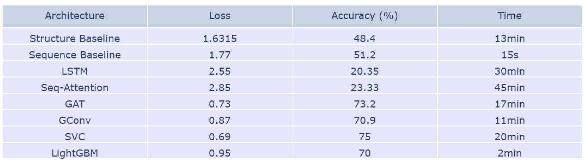
Figure 8: Validation Accuracy comparison between GAT and GConv

3.2 Mix with Machine Learning

In addition to neural network-based models, we wanted to test the performance of more classic Machine Learning methods. Given that our problem is that of classification, we wanted to test models that have proven themselves in this type of scenario. The base model used in sequence_baseline is a Logistic Regression without any optimization of the hyperparameters. The first idea was to try to modify these parameters to obtain better results. Using an L1 regularisation function seemed to be a good start because it allows removing variables considered non-significant. As we had more than 9400 due to the TFiDF and the variables added during feature engineering, it was important to make a selection. This small modification already allowed to improve the model's results but it was still insufficient.

The second idea was to use more complex models that are certainly more demanding in terms of computation time but have proven their effectiveness in multi-class classification problems. We tested kernel methods like SVC and ensemble methods like XGBoost and LightGBM. Their simple use allowed us to achieve much more convincing results and we still had to optimize their hyperparameters. To try to reduce the computation time, we wanted to project our database into a lower-dimensional space using a method such as PCA. However, it was difficult to drastically reduce the dimension without greatly impacting the performance. In the end, we decided to be patient and keep all the variables as they were. Thus, the hyperparameter optimization process was carried out using GridSearchCV which retains the best parameters using a Cross-Validation. SVC is the model that stood out the most during this phase and is the author of our best score on the Leaderboard.

One way to improve our performance was probably to take into account all the permutations of 4 amino acids when transforming the sequence Tfidf, but this potential performance would be produced at the expense of computation time.



Architecture	Loss	Accuracy (%)	Time
Structure Baseline	1.6315	48.4	13min
Sequence Baseline	1.77	51.2	15s
LSTM	2.55	20.35	30min
Seq-Attention	2.85	23.33	45min
GAT	0.73	73.2	17min
GConv	0.87	70.9	11min
SVC	0.69	75	20min
LightGBM	0.95	70	2min

Figure 9: Summary Table

Conclusion

To conclude, this challenge helps us to put into practice many approaches while completing practical courses. We tried a lot of preprocessing steps and architectures to achieve the lowest loss. One of our main difficulties was to identify the architecture with the best potential to optimize it to the fullest. We spent a lot of time coming back and forth between several tries. It is mostly due to our lack of experience as it was our first hackathon of the sort. We are convinced to achieve higher performance in our next opportunities.

Regarding the challenge, we dealt with proteins which are highly sensitive data linked to health. The margin of error is as tiny as one can imagine and even with a small loss, we would need a lot more time to optimize it for deployment on an application.

References

[1] Dennis J. Dietzen. Principles and applications of molecular diagnostics. 2018.

[2] Ahmed Elnaggar, Michael Heinzinger, Christian Dallago, Ghalia Rehawi, 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. *bioRxiv*, 2020.

[3] Vladimir Gligorijevic, P. Douglas Renfrew, Tomasz Kosciolek, Julia Koehler Leman, Kyunghyun Cho, Tommi Vatanen, Daniel Berenberg, Bryn Taylor, Ian M. Fisk, Ramnik J. Xavier, Rob Knight, and Richard Bonneau. Structure-based function prediction using graph convolutional networks. *bioRxiv*, 2019.

[4] John M. Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Zídek, Anna Potapenko, Alex Bridgland, Clemens Meyer, Simon A A Kohl, Andy Ballard, Andrew Cowie, Bernardino Romera-Paredes, Stanislav Nikolov, Rishub Jain, Jonas Adler, Trevor Back, Stig Petersen, David A. Reiman, Ellen Clancy, Michal Zielinski, Martin Steinegger, Michalina Pacholska, Tamas Berghammer, Sebastian Bodenstein, David Silver, Oriol Vinyals, Andrew W. Senior, Koray Kavukcuoglu, Pushmeet Kohli, and Demis Hassabis. Highly accurate protein structure prediction with alphafold. *Nature*, 596:583 – 589, 2021.

[5] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks, 2016.

[6] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017.

[7] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks, 2017.

[8] Chris Dyer Xiaodong He2 Alex Smola Eduard Hovy1 Zichao Yang, Diyi Yang. Hierarchical attention networks for document classification. 13, 2016.