ALTEGRAD Project Report
Cellular Component Ontology Prediction

CECCHINI Alessandro
ale.cecchini.valette@gmail.com

CHAUVIN Paul
paulchauvin97@gmail.com

LOISEAU Thibaut
thibaut.loiseau@gmail.com

January 2023

Abstract

This project investigates the use of machine and deep learning techniques for cellular component ontology prediction. The dataset consisted of proteins from 18 different classes, leading to an imbalanced dataset which required adaptation. Two approaches were used to tackle the problem. The first approach was based on the protein sequence and involved the use of a decision tree and support vector machines (SVMs), as well as protCNN, bidirectional LSTM and transformer models such as ProtBert. The second approach focused on the protein's structure by using a graph attention net. Although the final results were not acceptable mainly due to overfitting, several interesting paths of investigation were explored.

1 Introduction

Proteins are essential biomolecules that play a vital role in the functioning of all living organisms. They act as enzymes, provide structural support, and play a key role in the immune system's ability to distinguish self from invaders. The structure of proteins is determined by their sequence of amino acids, which fold to form a 3D structure that determines their specific chemical functionality.

However, the exact details of this process are not yet fully understood. In this project, we aim to study and apply machine learning/artificial intelligence techniques to a classification problem from the field of bio-informatics. Specifically, we will be using the sequence and structure of $6,111$ proteins and classify them into 18 different classes, each representing a characteristic of the location where the protein performs its function.

2 Methodology

2.1 Data Used

In this project, we will be using a dataset of $6,111$ proteins to classify them into 18 different classes, each representing a characteristic of the location where the protein performs its function. The dataset includes the sequence of each protein in a text file, edges and attributes of the edges in separate text files, attributes of the nodes in another text file, graph indicators and labels for the proteins in text files. The final evaluation of the methods will be done on proteins that have not been labeled and the goal is to predict the class label of each one of those proteins.

2.2 Data Analysis

As this challenge is a classification problem, we first want to analyze the data to find potential issues that could lead to bad model training. We first look at the distribution of classes in the training data (See Figure 1).

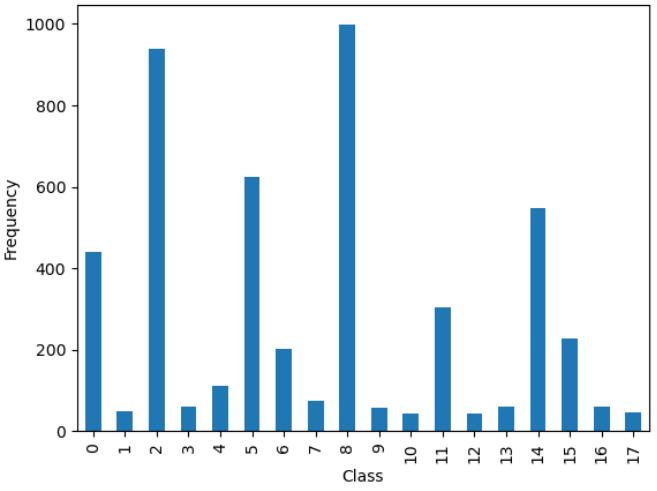

Figure 1: Class distribution over training set

As we can see, the class distribution over the training set is very unbalanced. Indeed, the less represented class has only 43 examples, whereas the most represented one has 998 examples. This can be a big problem as the model could completely ignore the less represented classes.
There exists several techniques to solve this problem, mainly :

- Subsampling : we compute the number of examples in the less represented class and sample that same number of random examples from other classes to form a new dataset. The drawback is that we lose a lot of information, as we discard some of the most represented examples.
- Oversampling : we create a new dataset by having the same examples from the less represented class several times. We then have a new dataset that is bigger than with subsampling, but this method could lead to overfitting.
- Weighted sampling : whether than subsampling or oversampling, we can sample the original dataset by weighting each

example according to its class. The less represented examples will have a greater probability to be sampled than the ones being more represented. By doing this, we keep the whole dataset as is. We are still doing oversampling of the less represented examples, but the model will almost see all examples during training.

2.3 Sequence based models

2.3.1 Modifications made on the sequence baseline

In this section, we tried to modify the baseline model using TF-IDF and logistic regression, using support vector machines (SVM), and a further modification using decision trees. The goal of the study was to determine which model would provide the highest accuracy and smallest loss for the task at hand.

The baseline model using TF-IDF and logistic regression was a simple yet effective approach for text classification. TF-IDF, which stands for term frequency-inverse document frequency, is a common technique for representing text data in a numerical format that is suitable for machine learning algorithms. Logistic regression is a widely used linear model for classification problems.

The modification of the baseline model using SVM improved the performance of the model by introducing a non-linear decision boundary. SVM, which stands for support vector machines, is a powerful algorithm that can handle complex and high-dimensional data. By using SVM, we were able to achieve a higher accuracy and smaller loss than the baseline model.

Finally, we also tested a modification of the model using decision trees. Decision trees are a popular method for classification tasks and are known for their interpretability. However, compared to SVM, decision tree didn't perform as well. It resulted in a lower accuracy and larger loss than the SVM model.

2.3.2 Protein Computational Neural Network (ProtCNN)

In this section, we present a deep learning model for predicting the label of a protein based on its amino acid sequence. It uses protein computational neural network (ProtCNN) [1]. The first step in our model is to create a dictionary that maps each amino acid code to an integer. We then use this dictionary to convert the amino acid sequences into integer sequences, which are then padded to a fixed length. Next, the integer sequences are one-hot encoded, which is a process of converting integers to binary vectors. The output variable (the label of the protein) is also encoded using a label encoder, and one-hot encoded.

We first used a complex architecture, which was then replace by a simpler one in order to intend to reduce overfitting. The architecture is a CNN model that uses dilated convolution and a dilation rate. We used cross-validation to evaluate the model on unseen data.

We also implemented two alternative techniques in our model: one using downsampling and another one using upsampling. However, none of these techniques improved the model's performance. This is why we chose to work with the 'weight class' argument which is a technique used to handle class imbalance in a dataset, as described in the 'Data Analysis' part.

To prevent overfitting, we used L2 regularization and dropout. L2 regularization penalizes large weights, making the model more robust to overfitting. Dropout is used to randomly drop out a certain percentage of neurons during training. This prevents the model from relying too heavily on any one feature or neuron.

This model performed poorly, we didn't manage to overcome overfit, we had good accuracy and a low loss on the training dataset that we splitted between train and evaluation. However performances on evaluation and test were not sufficient. We can see clearly that the model is overfitting in the figures 3) and 2). Another explanation for poor results compared to the one in the paper [1] is the few data we had compared to the data used in the original paper.

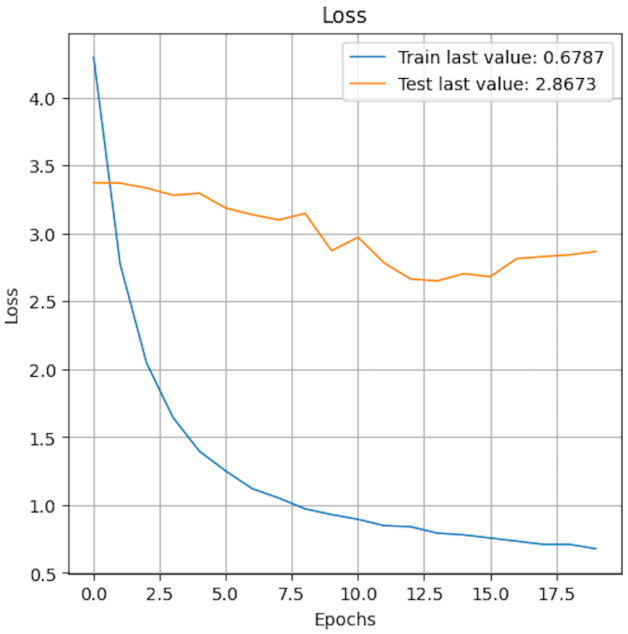

Figure 2: Loss for train and test on ProtCNN

2.3.3 Bidirectional Long Short Term Memory (BiLSTM)

Then, we implemented a model using bidirectional Long Short Term Memory (BiLSTM).[2]

Just like in the ProtCNN model, in order to prevent overfitting, we used cross-validation and regularization techniques in the BiLSTM model. We defined the number of folds as 5 and used the KFold object to split the data into training and validation sets. We also used class weight to balance the dataset.

For the encoder, we used an Embedding layer with 128 dimensions and a maximum input length of 100 for the sequences. We then used a bidirectional LSTM layer with 64 units and applied dropout with a rate of 0.3. We then used a dense layer with

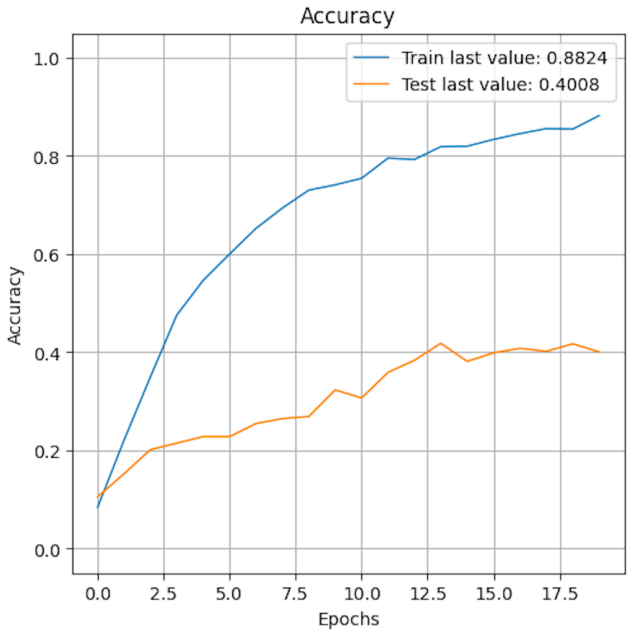

Figure 3: Accuracy for train and test on ProtCNN

18 units (because we target 18 different labels) and a softmax activation function as the classifier. We used the Adam optimizer and categorical cross-entropy as the loss function. We also used early stopping with a patience of 3 to stop the training when the validation loss did not improve.

Same as before here, this model performed poorly, we didn't manage to overcome overfit, we had good accuracy and a low loss on the training dataset that we splitted between train and eval. However performances on eval and test were really lower.

2.3.4 Fine-tuning pretrained models

The last model we tried on sequence-based approach is based on the idea to fine-tune pretrained models. We are using ProtBERT, a model from [3] available on the Hugging Face Hub, on which we add a classifier head to adapt to the desired output. We fine-tune the model first by freezing the pretrained layers.

We first have to tokenize the sequences. The length ditribution over all the dataset is presented Figure 4.

We therefore choose a length of 512 for the tokenizer. The dataset was also not well distributed, so we tried to have a weighted sampler to have an evenly distributed dataset, but we did not manage to train a network with this kind of sampling. The raw training data was used as is.

We train the head of the network for 25 epochs with a batch size of 128. We start with a learning rate of 10^{-3}, with a step decay of 10 every 5 epochs, as the loss is not decreasing.

We did not manage to beat the sequence baseline with this model and only got about 20% test accuracy for a loss of 2.35.

We could have tried to do ensembling of our method to

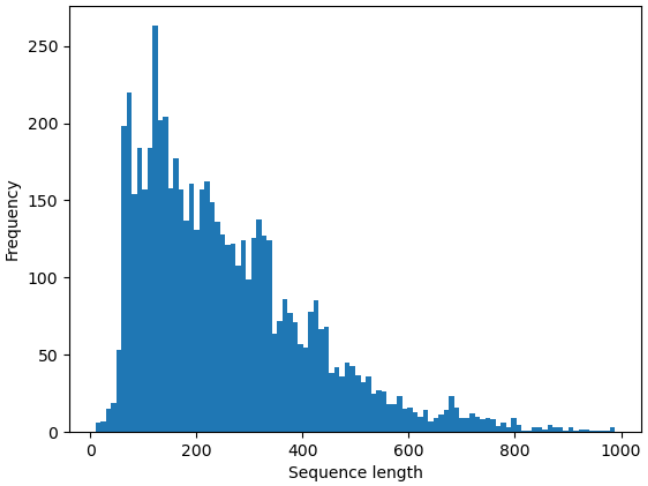

Figure 4: Sequence length distribution over all dataset

2.4 Structure-based models

In this section, we aim at using graph models that could use the intrisic structure and interactions within the protein.

In this section, we aim at looking only at the sequences using language models or more classical models to try to predict the classes of the test dataset.

Inspired by the paper Universal Graph Transformer Self-Attention Networks [4] we propose a novel architecture incorporating edge features. Given a graph $\mathcal{G} = (V, E, \{h_v^0\}_{v\in V}, \{q_e^0\}_{e\in E})$, where V is the set of nodes, E the set of edges, h_v^0 is the node attribute of node v and q_e^0 is the attribute of edge e, the goal is to classify $\mathcal{G}$ to its appropriate class label. We define a transformer architecture composed of K blocks. Each block is composed of T transformer encoder blocks. The output $h_{t,v}^k$ of node v at block k, layer t is given by :

$$x_{t,v}^k = \text{GroupNorm}(h_{t,v}^{(k-1)} + \text{ATT}(h_{t,v}^{(k-1)}, \{h_{t,u}^{(k-1)}||e_{u,v}\}_{u\in\bar{N}_v\cup\{v\}}) \quad (1)$$

$$h_{t,v}^{(k)} = \text{GroupNorm}(x_{t,v}^{(k)} + \text{MLP}(x_{t,v}^{(k)})) \quad (2)$$

Where the MLP is a classical multi-layer perceptron with a dropout layer and a Mish activation function. The ATT(.) layer consists in a cross-attention operation between query $h_{t,v}^{(k-1)}$ and key-values $\{h_{t,u}^{(k-1)}||e_{u,v}\}_{u\in\bar{N}_v\cup\{v\}}$ where $\bar{N}_v$ is a uniform sampled set of neighbors (with replacement), $e_{u,v}$ is the edge feature between node u and v and $||$ is the concatenation operator along feature dimension. Let's explicit the operation :

$$\text{ATT}(h_{t,v}^{(k-1)}, \{h_{t,u}^{(k-1)}||e_{u,v}\}_{u\in\bar{N}_v\cup\{v\}}) = \sum_{u\in\bar{N}_v\cup\{v\}} \alpha_{t,(v,u)}^{(k)} \cdot (V_t^{(k)}[h_{t,u}^{(k-1)}||e_{u,v}]) \quad (3)$$

$$\alpha_{t,(v,u)}^{(k)} = \text{SoftMax}\left(\frac{(Q_t^{(k)}h_{t,v}^{(k-1)}) \cdot (K_t^{(k)}[h_{t,u}^{(k-1)}||e_{u,v}])}{\sqrt{d}}\right) \quad (4)$$

The data is preprocessed in the following way :

- We extract for each training, validation and test dataset the graphs labels, the nodes features and edge connections and features of each graph.
- We construct two arrays A and B with both $|\mathscr{G}|$ rows and 2 columns. The first row gives the indices of the first nodes and edges respectively where a new graph begins. The second row gives the length of the nodes and edges sequences which composes each graph.
- Similarly we construct another array C of size $[|V|, 2]$ where $|V|$ is the total number of nodes, indicating the first edge at which every new node starts (edges are given in ascending order with respect to first node), and the number of neighbors a node has.
- We construct an array D of size $[E, K, |V|, L]$ where E is the total number of epochs in training and L is the number of neighbors we want to sample for every node at every new pass in block $k \in K$. At every node entry v, we allocate a tensor where each entry is sampled from a uniform distribution over $0, ..., B[v, 1]$, where $B[v, 1]$ represents the number of neighbors of node v.
- We then add $D[...] + B[:, 0]$ to obtain the positions of the generated node neighbors in B. These indices allow us to extract the neighbor's node positions in A by accessing the value of the edge connections array for these indices. Similarly, we also extract edge features of all generated neighbors.
- We finally construct a batch with nodes and edge features, the graph corresponding labels for train and validation set, the neighbor's node positions for every node and transformer block, the corresponding neighbors edge features, with the total number of nodes being inferior to some fixed batch size. At every epoch we shuffle the node and edges concatenated indices.

3 Results

3.1 Metrics used

The performance is assessed using the logarithmic loss measure on the Kaggle competition. This metric is defined as the negative log-likelihood of the true class labels given a probabilistic classifier's predictions. Specifically, the multi-class log loss is defined as:

$$-\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{C} y_{ij}\log(p_{ij})$$

where N is the number of samples (i.e., proteins), C is the number of classes (i.e., the 18 categories), y_{ij} is 1 if sample i belongs to class j and 0 otherwise, and p_{ij} is the predicted probability that sample i belongs to class j.

We also used the accuracy metric in order to help us better understand our data and results. Eventhough it is not the metric used for the model evaluation on Kaggle, it helped us for training our models.

3.2 Results on sequence-based models

The quantitative results on sequence-based models are summarized in Table 1.

We did manage to beat the baseline with only one of our models. All others failed to demonstrate real improvements on the problem.

3.3 Results on structure-based models

We finished coding the model and preprocessing the data but we did not had the time to implement the training loop neither to train the model. We consider having results for day of the presentation.

4 Conclusion

In conclusion, this project has highlighted the potential of machine learning technologies for cellular component ontology prediction. While not achieving satisfactory results due to overfitting, significant insights were gathered through two different approaches: sequence-based and structure-based models. With further investigation and refinement, these approaches could form the basis of future successful models in predicting protein classes. We could also improve our performances by combining both sequence and structure-based models.

References

[1] Maxwell L. Bileschi, David Belanger, Drew Bryant, Theo Sanderson, Brandon Carter, D. Sculley, Mark A. DePristo, and Lucy J. Colwell. Using deep learning to annotate the protein universe. May 2019.

[2] Protein sequence classification (2022). https://deepnote.com/@ronakv/protein-sequence-classification-33797a33-7311-48d3-8794-79a1ee26f406.

[3] 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.

[4] Dai Quoc Nguyen, Tu Dinh Nguyen, and Dinh Phung. Universal graph transformer self-attention networks, 2019.

Sequence-Based models						
	Sequence baseline	Modified sequence baseline - SVM	Modified sequence baseline - decision tree	ProtCNN	BiLSTM	ProtBERT
Loss	1.67	1.61	1.92	2.2	2.1	2.35
Accuracy	50.7%	48.3%	38.2%	38.5%	35%	20%

Table 1: Quantitative results for sequence-based models