BLAST My MLPs! Many Layers of Perseverence and The Predictable Power of Sequence Alignment

Agostini Flavio
flavio.agostini.1@studenti.unipd.it

Poli Mikael
mikael.poli@studenti.unipd.it

Quaglio Emanuele
emanuele.quaglio@studenti.unipd.it

Abstract

Protein function annotation is a crucial task in bioinformatics, particularly for understanding the molecular and cellular roles of uncharacterized proteins. Here, we present a multi-faceted approach combining protein embeddings derived from state-of-the-art models and data derived from similarity analysis obtained with BLAST. We developed two systems to integrate information gathered with BLAST in our Multi Layer Perceptron (MLP) model. In the first, we use BLAST results to compute TF-IDF-transformed features; in the second, we use BLAST information in post-processing. We developed a consistent pipeline for training and validation, including stratified train-test splits, careful synchronization of GO annotations and embeddings, and the application of CAFA evaluation metrics to a custom test set. We further constrained predictions to a maximum of 500 GO terms per protein per GO aspect to ensure computational efficiency and biological relevance. This approach demonstrates robust performance in annotating proteins with GO terms, with BLAST likely providing the most accurate predictions and neural networks supporting them by filling gaps when BLAST data is unavailable.

1. Introduction

The annotation of protein functions remains a key challenge in computational biology, driven by the growing volume of protein sequences generated by high-throughput sequencing technologies. Functional annotation often involves assigning Gene Ontology (GO) terms to proteins, providing a structured and hierarchical framework to describe their molecular functions, biological processes, and cellular components. Despite advances in experimental methods, the annotation of proteins, particularly in non-model organisms, relies heavily on computational approaches due to the prohibitive costs and time required for experimental characterization.

Traditional sequence alignment methods, such as BLAST, have been extensively used for functional annotation. However, these methods are limited by their reliance on sequence similarity, which may not capture the functional nuances of distantly related proteins. To address these limitations, recent developments in deep learning have enabled the generation of protein embeddings, which encode structural and functional information derived from large-scale sequence data. These embeddings have shown promise in a range of predictive tasks, including function annotation. We thought that complementing these embeddings with alignment-based features, such as term frequency-inverse document frequency (TF-IDF) representations of GO term occurrences, could offer a novel method to enhance the annotation process.

In this study, we propose two hybrid frameworks. The first integrates protein ProtT5 [1] embeddings and TF-IDF-transformed BLAST-derived GO term frequencies to train an MLP multi-label classifier. The second applies the knowledge gathered with BLAST in post-processing, modifying the results obtained by the MLP trained with just the sequence embeddings. The approach is applied independently to the three GO aspects (MF, BP, CC), leveraging the distinct characteristics of each. The performance of the classifiers is evaluated using *cafaeval* [2], ensuring a comprehensive assessment of their predictive capabilities.

By combining state-of-the-art embeddings with traditional alignment-based features, our method seeks to improve the accuracy and coverage of GO term predictions, providing a scalable and effective solution for protein function annotation in diverse datasets.

1.1. Datasets

- Training
  - **train_set.tsv:** training set containing protein IDs and their associated GO terms and aspects,
  - **train_ids.txt:** protein IDs for the proteins in the

training set,

  - **train_embeddings.h5:** ProtT5 protein embeddings.

- Test

  - **test_ids.txt:** protein IDs,
  - **test_embeddings.h5:** ProtT5 protein embeddings,
  - **test_blast_results.tsv:** BLAST results between the test (query) and training (target) proteins.

1.1.1 Exploratory Data Analysis

To get a bird's-eye view of our training set, we checked the distribution of GO terms and proteins by aspect:

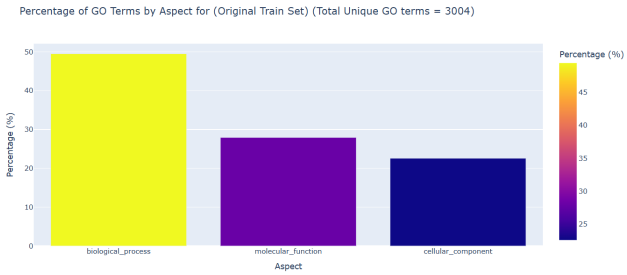

Figure 1. Distribution of GO terms by aspect (training set).

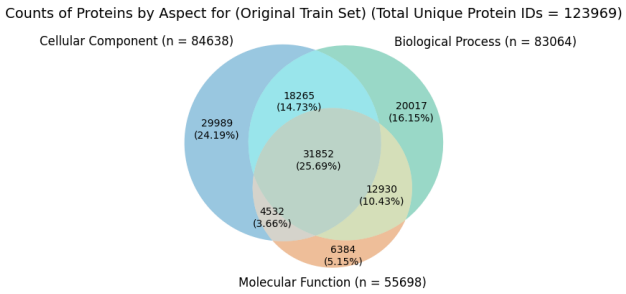

Figure 2. Counts of proteins by aspect (training set).

The training dataset featured 123,969 unique proteins. The most represented GO aspect was biological_process.

For each aspect, we then plotted the distribution of its 50 most frequent GO terms. The plots help visualize Gene Ontology's hierarchical structure. Every protein in each aspect-based subset of the training set is linked to the root term of its respective subontology, along with a few higher-level terms in the hierarchy. The deeper into the hierarchy, the fewer proteins are associated with the lower-level GO terms (leaf terms), following an almost uniform distribution across them.

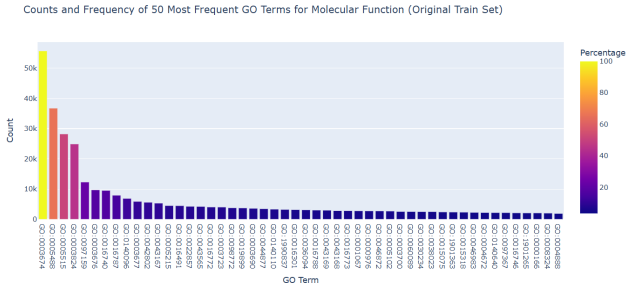

Figure 3. Distribution of the 50 most frequent GO terms for molecular_function (training set).

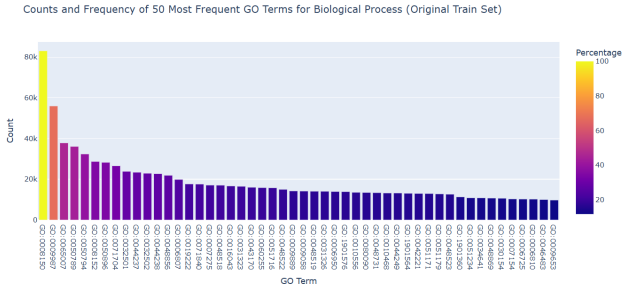

Figure 4. Distribution of the 50 most frequent GO terms for biological_process (training set).

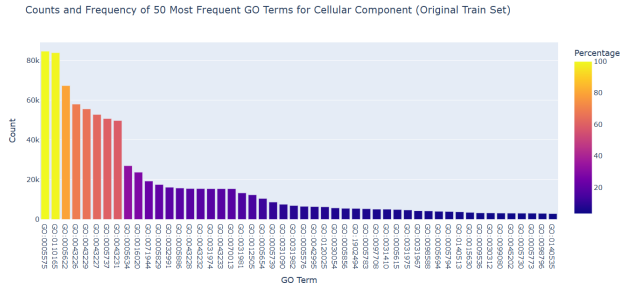

Figure 5. Distribution of the 50 most frequent GO terms for cellular_component (training set).

2. Methods

2.1. General preprocessing

To make use of sequential data, we created a BLAST database comprising all the training sequences. Due to computational issues, we split the original FASTA file in chunks of 5000 sequences each and used a custom bash script to run BLAST using the sequences of each chunk against the database of train sequences. Here the standalone version of blast has been used in local. As a result we obtained a file containing all BLAST alignments between train sequences. We didn't perform any preprocessing on embeddings data, as we deemed it already usable. We defined a 1000 protein test set by extracting a subset of the training set. We split the training set in three sets, one for each aspect, in order to train the three separate models.

2.2. Integration of BLAST with TF-IDF approach

To be able to integrate alignment information in our Deep Learning pipeline, we decided to convert the results of BLAST into a TF-IDF table. As a first step, we obtained a table of counts by counting for each protein how many times each GO term appeared in the proteins whose BLAST e-value was below a threshold (0.001), and thus considered significant for our purpose. We had to use a particularly low e-value, with the risk of omitting some relevant alignments, because of computational issues derived from our RAM availability. TF-IDF was then applied to this count table to account for the fact that GO terms that occur less often should have more weight during the training process. The idea was to obtain a "bag-of-GO-words" that allowed the classifier to make use of similarity information. The *term frequency* for term j in document i is defined as:

$$\text{TF}(i,j) = \frac{\text{Count}(i,j)}{\sum_{k=1}^{M} \text{Count}(i,k)}$$

where:

- $\text{Count}(i,j)$ is the raw count of term j in document i,
- M is the total number of terms in document i.

The *document frequency* for term j is the number of documents in which the term appears at least once:

$$\text{DF}(j) = \sum_{i=1}^{N} \mathbb{I}(\text{Count}(i,j) > 0)$$

where:

- N is the total number of documents,
- $\mathbb{I}(\cdot)$ is the indicator function, which returns 1 if $\text{Count}(i,j) > 0$, and 0 otherwise.

The *inverse document frequency* for term j is defined as:

$$\text{IDF}(j) = \ln\left(\frac{N+1}{\text{DF}(j)+1}\right)$$

where:

- N is the total number of documents,
- $\text{DF}(j)$ is the document frequency of term j,
- The terms $N+1$ and $\text{DF}(j)+1$ include a smoothing factor to avoid division by zero.

And finally *TF-IDF* weight for term j in document i is calculated as:

$$\text{TF-IDF}(i,j) = \text{TF}(i,j) \cdot \text{IDF}(j)$$

We obtained three TF-IDF frequency tables, one for each GO aspect, and concatenated with the protein embeddings to obtain the final features to be fed to the classifier. We then used the *Keras* [3] library to define the MLP architecture of the model. We used a 80/20 train validation split. We performed grid search to find the best architecture, which resulted in: 4 fully connected layers with dimension [512, 256, 128, 64] and ReLU activation function. The final layer implements a sigmoid function to be able to output probabilities for each GO term. Batch normalization and dropout were used as regularization techniques. The training procedure integrated early stopping and learning rate reduction.

2.3. BLAST Integration Through a Post Hoc GO Terms Transfer Approach

Our alternative approach involved training three separate MLP models (one for each GO aspect), and transferring GO terms between similar proteins based on BLAST results. These models had a deeper architecture than the previous ones, with 10 fully connected hidden layers, each containing 300 units. Each layer used leaky ReLU activation, batch normalization, and dropout. We employed early stopping and a decreasing learning rate, training each model for 50 epochs. All models were once again implemented in *Keras* and shared identical parameters, except for the BP model, which used a dropout rate of 0.3, while the MF and CC models used 0.2. The rationale for using a deeper architecture was that these models might require a more complex structure to learn the relationships between protein embeddings and functional annotations, especially since they couldn't access BLAST information during training.

2.3.1 Processing BLAST Results

In the BLAST results file, query proteins were drawn from our custom test set, while target proteins came from our training set. To generate predictions based on BLAST matches, we followed these steps:

1. *Filtering BLAST matches*

   We retained only matches with an e-value below 0.001. To avoid skewed results due to an imbalance in match distribution, we further limited each query protein to its top 50 matches.

2. *Assigning weights to matches*

   For each query-target pair, we calculated a weight based on the e-value using the transformation:

   $$y_i = min(-log_{10}(e_i + 10^{-300}), 10^6)$$

   - Adding 10^{-300} to e_i prevents numerical issues when $e_i = 0$,

- The base-10 logarithm emphasizes lower e-values (more significant matches),
- The result is capped at 10^6 to prevent overly large weights.

3. *Scaling weights*

   The weights were scaled to fall in the $[0, 10]$ range, ensuring compatibility with a sigmoid function (which we applied in a later step), preventing extreme probabilities.

4. *Encoding GO terms and aggregating scores*

   To aggregate the GO term scores, we first encoded each GO term as a binary vector of size $1 \times n$ (n = number of unique GO terms in the training set, in our case, 3004), where each element corresponds to a unique GO term. Then:

   - For each query protein, we initialized a score vector $\mathbf{s}_q$ of length n to a reasonably small negative value, i.e., -3.0,
   - For each match between a query q and a target t, we retrieved the GO terms associated with t,
   - We updated the score vector $\mathbf{s}_q$ using the match weight $\mathbf{w}_{q,t}$:

   $$\mathbf{s}_q = \mathbf{s}_q + \mathbf{w}_{q,t} \times \mathbf{e}_g$$

   where e_g is the encoding vector of the GO term g.

5. *Transforming scores into probabilities*

   After processing all matches for a query, we applied the sigmoid function element-wise to $\mathbf{s}_q$:

   $$s_q = \sigma(s_q), \quad \sigma(x) = \frac{1}{1 + e^{-x}}$$

   This ensured that all scores were normalized between 0 and 1.

2.3.2 Combining MLP and BLAST Predictions

We used the trained MLP models to predict GO terms for the test set and combined these predictions with the BLAST-based scores. The combination was controlled by a coefficient α, using the formula:

$$final_predictions = \alpha \times MLP + (1 - \alpha) \times BLAST$$

3. Results

We used *cafaeval* to obtain F-scores for both of our approaches and compare them to the InterPro and Naive methods baseline. We tested the post hoc approach at 11 different α levels, ranging from 0 to 1 in increments of 0.1. As shown in Figure 6, the highest F-scores were those obtained by setting α to 0.2. Finally, table 1 summarizes our results compared to the baselines.

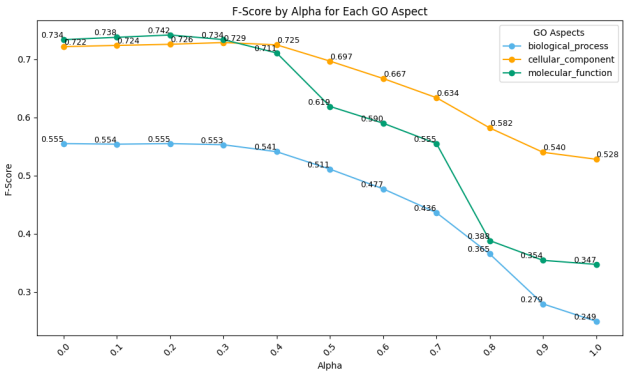

Figure 6. F-score by alpha level for each GO aspect.

F-scores			
Method	MF	BP	CC
Naive	0.450	0.343	0.597
InterPro	0.618	0.350	0.256
BLAST TF-IDF	0.766	0.51	0.756
BLAST Post Hoc	0.742	0.555	0.726

Table 1. Best F-scores for each approach.

Both methods showed improvements over the Naive and InterPro baselines, with a tradeoff emerging between their strengths. The TF-IDF approach performed better at predicting MF and CC terms, while the post hoc approach was more effective for BP terms. Overall, their performances were comparable..

4. Protein Function Prediction

Given the similarity in results between the TF-IDF and the post-hoc approach, and considering the lower computational cost of the latter, we chose to predict GO terms for the proteins in the original test set (whose correct labels we did not know) using the post-hoc method. This decision is also justified by the better performance of this approach in the biological processes aspect, which is the most difficult to predict. We set the α value to 0.2, as it performed best in our tests, meaning the MLP models contributed 20% to the final predictions. To ensure our predictions adhered to the instruction of having a maximum of 500 GO term predictions per aspect per protein in the final output, we:

1. Split the predictions DataFrame into three separate DataFrames, one for each GO aspect,
2. Sorted each DataFrame first alphabetically by protein ID, and then by prediction score in descending order,
3. Limited each DataFrame to the top 500 predictions for each protein, and rounded the predictions to three decimal places,
4. Further filtered the predictions by keeping only those above 0.2.

Next, we combined the three DataFrames, sorted alphabetically by protein ID and by aspect, into a single structure. We removed the 'aspect' column and exported the resulting DataFrame to a .tsv file. The final file contains a total of 74,864 predictions for the 1,000 test proteins.

5. Discussion

We found BLAST-based predictions to be the most accurate method for predicting protein function annotations. However, during the testing phase, predictions where 20% of the weight was assigned to the MLP models yielded the best performance in terms of F-score. We hypothesize that the MLP models may have learned coarser-grain association patterns between ProtT5 embeddings and GO terms, potentially focusing on root terms of each aspect. This would complement the BLAST-based predictions, especially in cases where BLAST data was incomplete. In fact, our custom test set contained 86 proteins (8.6%) that were not present in our training BLAST queries, suggesting that the MLP models helped fill in gaps when BLAST information was unavailable.

What is particularly noteworthy in our project is that we achieved nearly identical performance accuracy using two methods derived from complementary approaches. The TF-IDF method leverages the MLP models' ability to learn general and frequent associations during training, while the post hoc approach simply transfers GO terms between proteins without training. Despite their differing mechanisms, both methods produced comparable results.

In terms of computational requirements, the MLP models used in the post hoc approach required approximately 15 minutes of training time on Google Colab's T4 GPU, on top of added computational cost required to obtain and process the GO-counts feature table. While the training process is relatively quick, the computational resources needed should be considered if scaling up to larger datasets or more complex models. This concern is especially relevant when incorporating BLAST information during training, which is more computationally intensive. Similarly, performing BLAST alignment across large protein datasets is computationally demanding and must be considered when applying our methods to larger-scale projects.

That said, combining BLAST-based predictions with MLP-based predictions offered a balanced trade-off between accuracy and computational efficiency. The post hoc calculation and integration of BLAST-based predictions took only a few minutes, making it a practical option when BLAST data is readily available.

Although we focused on straightforward MLP architectures, a variety of other approaches are available in the literature and could be explored in future work. Furthermore, protein function predictions in the post hoc approach are influenced by several factors, such as thresholds for selecting target protein matches, normalizing BLAST weights, the alpha coefficient, and score initializations for probability calculations. Adjusting these parameters may lead to further improvements in performance.

6. Conclusion

In conclusion, BLAST-based predictions were likely the dominant factor driving protein function prediction accuracy, with MLP models playing a supportive role. The MLP models seemed to be particularly useful in cases where BLAST data was incomplete, helping to fill in gaps when necessary. While the combination of both methods provided some valuable insights, the results suggest that BLAST predictions are the primary contributor to performance. Further work could explore optimizing the MLP models and adjusting relevant parameters to enhance their utility in scenarios with missing data, while continuing to leverage the strengths of BLAST-based predictions for more better overall results.

7. Software

The code, data, and documentation for the post hoc approach to BLAST results integration can be found at https://github.com/mikaelpoli/protein-function-prediction.git. The .tsv file containing the predictions for the test set is in the `pfp/results/submission` folder. Code and documentation for the TF-IDF approach, including the pipeline for obtaining the training BLAST file, can be found at https://github.com/biosaio/Function-Prediction.git.

References

[1] Elnaggar, Ahmed / others u.a.(2022): *ProtTrans: Toward Understanding the Language of Life Through Self-Supervised Learning*, 10: 7112–7127.

[2] Piovesan, D. u.a.(2024): *CAFA-evaluator: A Python Tool for Benchmarking Ontological Classification Methods*.

[3] Chollet, François / others u.a. (2015): *Keras* `https://keras.io`.