Background

- Competition Name: CAFA 5 Protein Function Prediction
- Team Name: U900
- Private Leaderboard Score: 0.582[1]
- Private Leaderboard Place: 2

Team:

Anton Vakhrushev, Russia, Moscow. Btbpanda@gmail.com

Background:

- Education: math methods in economics
- Experience: 10+ years of Data Science and RnD in banking
- Kaggle experience: 8 years

One of my research areas is applying GBDTs for extreme multioutput tasks, that motivates me to enter the competition.

Sergei Fironov, Russia, Saint-Petersburg. ifserge@gmail.com

Background:

- Education: Bachelor of Physics
- Experience: 10+ years of RnD in Mass Spectrometry field, 7 years of DS application experience for industry, e-commerce and user experience improvement
- Active kaggle experience: 8 years

Initially, I was interested in the competition, believing that I could develop large models for the protein sequence. But the approach didn't work.

Alexander Chervov, France, Paris, Institute Curie. al.chervov@gmail.com

Background:

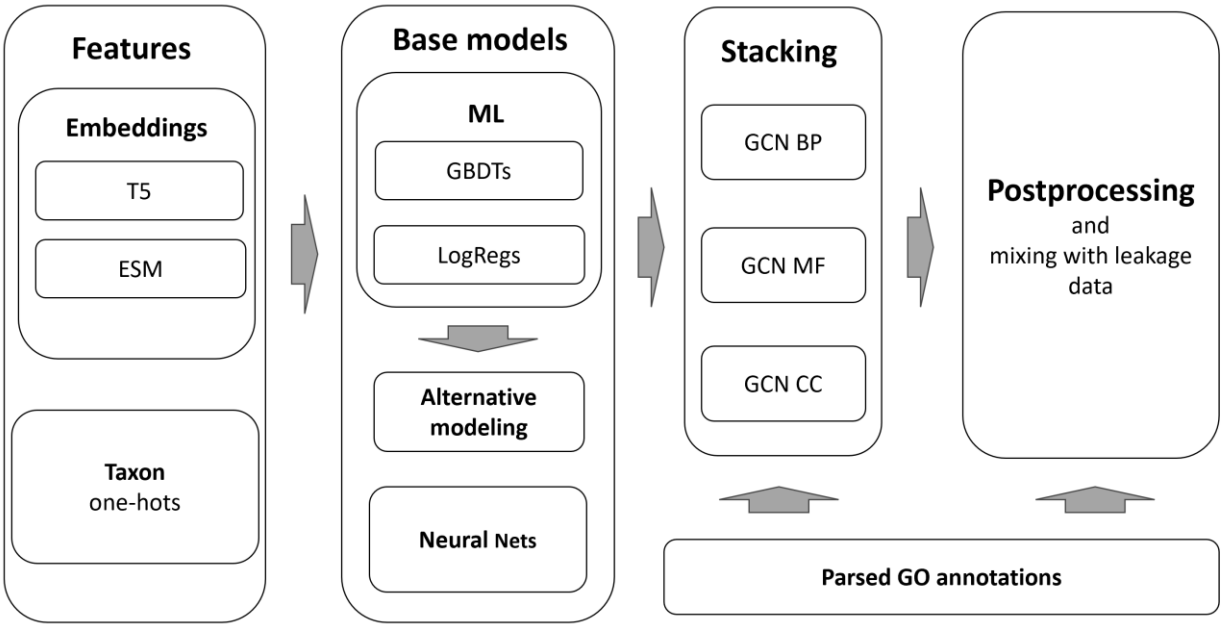
- Education: PhD in mathematics
- Experience: 10+ academia research in mathematics, 10 years of industrial data analysis and data science, 4 years academia research in bioinformatics and data science
- On kaggle experience: 5 years

[1] This is the score on the moment of competition end. While we were reproducing the results, we found a small error in one of pipelines (feature extractor in the stacking part), after fixing this issue score increased to 0.59 +- 0.002 (late submissions result) depends on random seed. Anyway, it does not affect the final position

I am interested in data science application to bioinformatics. I hope experience on that challenge can be applied to other tasks like prediction of kinase-sites phosphorylation by proteins, which is related to some my projects in Curie.

Solution description

General approach and used methods are represented on the Figure below:



Below we describe all the steps more detailed.

Sequence embedding and base models features

We tried to use the following embedding list:

- T5 (used for all models)
- esm2-small (used for some models)
- ankh-large (rejected)

Finally, the most of the models used only T5, but some of them used concat of T5+ESM. We used pre trained models only. To our surprise, fine-tuning was not helpful in this task.

In addition, we concatenated one-hot taxon features to embedding. We selected only taxons that are good enough represented in both train and test features, around 30 totally, other taxons where merged into single group. According to our estimation, adding one-hot taxon data improves the CAFA score of base models about 0.02-0.03

Validation

Our validation scheme could be separated on 2 parts:

Base models validation: we were not able to create a validation scheme better than a simple 5 fold CV. We had some experiments on the topic but other CV schemes lead to the models with less LB score.

Stacking models validation: stacking models were trained on the full train sample. The part of test sample was used to validate. We selected only the proteins from the test set that had annotations (not electronic) in GO database and used this small sample (about 500 protein/term pairs) as validation. Since stacking model is neural net, we used this sample to select the best checkpoints during the training process.

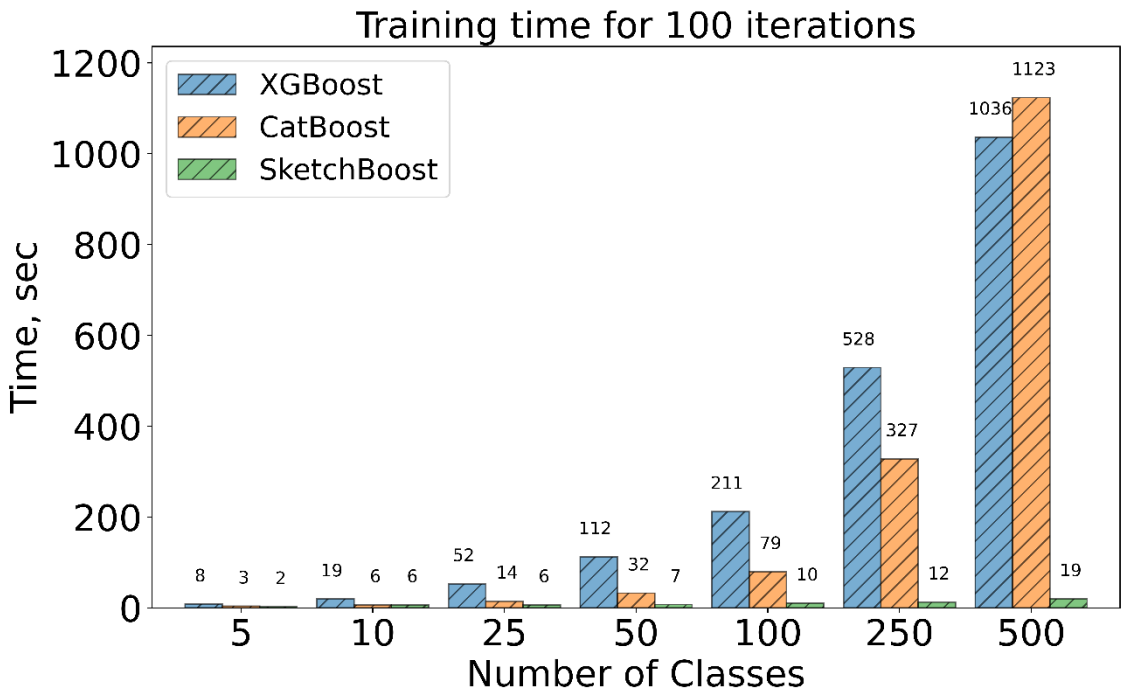
Base models

All models were trained to multi label task, where target matrix of shape n_proteins x n_terms is predicted using feature matrix n_protein x n_features using binary cross entropy loss. In addition, some of models were trained using alternative conditional approach described in the next section

To train each model we selected top frequent terms from each ontology, top amount depends on the model type and provided below.

Py-Boost

The best performed models on both CV and LB are from Gradient Boosting family. We used my own GBDT implementation called **py-boost.** I made it few years ago especially to deal with extreme multi-output datasets since it was my main research area at that time. It works on GPU only and it is able to train multi-label tens on even hundreds times faster than popular well known implementations. You can check **py-boost** on the github or read our NeurIPS paper, where we explain all the strategies to speed-up multi-output training. The following Figure illustrates the speed up of our method (**SketchBoost** refers to the fast algorithm implemented inside the the **py-boost** library).



Locally we were able to fit 4.5к output (3000/1000/500) **py-boost** on a single V100 32GB GPU and it takes about 2 hours for a single fold.

Logistic Regression

We also trained a simple 13.5к output Logistic Regression (10000/2000/1500). It shows much less performance than GBDT on the popular terms, but is able to perform on rare outputs.

Neural Networks

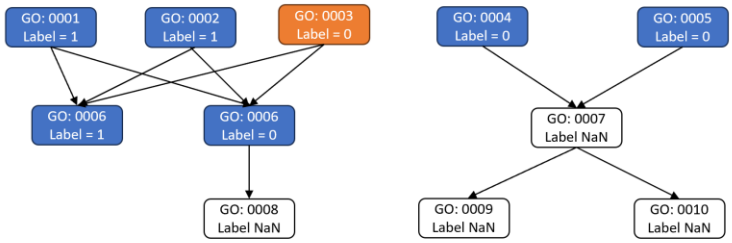
In the ensemble of models, we used a slightly modified version of the public notebook Pytorch,Keras,Etc 3 Blend, CAFA metric, etc. The only difference was the best cross-validated combination of hyper parameters averaged many times. Top 2000 terms from all ontologies were selected to train. T5 and ESM embedding were used as features (without one-hot taxon features)

Alternative modelling approach: predicting the conditional probabilities

This approach was discovered at the beginning of the competition that's why it makes some wrong assumptions about the data. But somehow it becomes useful for us for both CV and LB. The main advantage of this approach is utilising the OBO graph on the inference phase and it helps to make a prediction even for terms that were not used in training. Here are the main points:

1. We assume, that term can exist for the protein only **if at least one of its parents exist**. This is wrong. In real, if term exists, all its parents exist too because of the propagation rules.
2. We reformulate classic multi-label scheme where target matrix of shape (n_protein, n_terms) consists of 0 and 1 to the new scheme. Now targets can be 0, 1 and NaN. Term for the protein will have NaN value in matrix if there is no parent term with value 1.
3. During model training phase NaN cells in the target matrix are masked and ignored
4. Now, our model outputs **the conditional probabilities of term in case of at least one of its parents exist**. On the inference phase we need to transform it back to the raw probabilities
5. Transformation is made in the order defined by graph. When we process the term, all its parents are already processed and have raw probabilities. All terms are included to the scheme, even they are not used in training. For the terms that were not used for training, we used prior mean.
6. While processing the term, we make another wrong assumption, that parents probabilities for the term are independent. But if we assume that, according to [1], we can calculate raw probability for term as $P(GO{:}N) = P_{cond}(GO{:}N) * \left(1 - \prod_{K \in Par(N)} 1 - P(GO{:}K)\right)$ Remember that while we processing the term, all its parents already have raw probabilities calculated.

The following Figure represents the main difference between classic multi label and alternative conditional scheme:



Model type 1: Classic Multilabel, P(GO: N = 1)

	GO: 0001	GO:0002	GO:0003	GO: 0004	GO: 0005	GO: 0006	GO: 0007	GO: 0008	GO: 0009	GO: 0010
ProtID	1	1	0	0	0	1	0	0	0	0

Model type 2: Conditional, Pcond(GO: N = 1| (GO:Par0 = 1) or...or (GO: ParK = 1))

	GO: 0001	GO:0002	GO:0003	GO: 0004	GO: 0005	GO: 0006	GO: 0007	GO: 0008	GO: 0009	GO: 0010
ProtID	1	1	0	0	0	1	NaN	NaN	NaN	NaN

Key assumptions:

1) Label is 1 or 0 only if one of the parents is 1. Otherwise label is NaN

2) Events $(GO: K = 1)\ for\ all\ K$ are independent

GO:0003 could not be 0, because GO:0006 is 1. Some assumptions are wrong

Connection between Type 1 and Type 2

$$P(GO:N) = P_{cond}(GO:N) * \left(1 - \prod_{K \in Par(N)} 1 - P(GO:K)\right)$$

On the Figure term GO:0003 marked red, because it represents the situation, that could not be observed in the data – GO:0003 could not be 0 because its child GO:0006 is 1. The figure represents our view and motivation on the moment of discovering the approach, not the real situation.

That approach scores better on both CV and LB, but not too much. The main advantage of it is that models are very different from classic multi-label approach and combining it all together boosts our score a lot. We apply this technique for GBDTs and LogRegs, so finally we had 4 GBDTs (T5/T5 + ESM), 2 LogRegs (Only T5) and single blended NN as the base models.

Stacking with GCN

We used graph convolution network to aggregate all the predictions. It is trained for the node classification task where each node is a term and each protein is a graph (but all the proteins have the same adjacency matrix). As the node features we used base models predictions together with node embedding trained from scratch. We also added GO annotations features described in the next section.

Features are:

5 model predictions. 2GBDTs and 2 LogRegs based on T5 only and Single blended NN. Each of model prediction is transformed to 4 features. Important - all models assumed to have prediction for all the terms. If one model was not trained for particular term, prior probability of term is used, see Figure below for the details. One GO annotation feature added - 0/1 flag if term has electronic labelling in GOA database. In addition, 8 dims term id embedding is trained from scratch. So totally, each pair protein/term is represented by 29 features values.

Targets are:

Binary labels of protein/term pair. All terms are used in training. Models were trained using binary cross entropy loss.

TTA setup:

One interesting feature we discovered by just making a mistake. On the inference stage we have wrong model ids but still get a good score, so we made a kind of test time augmentations by averaging the predictions with shuffled models.

We used 2 extra GBDT models based on T5 + ESM to augment here. We made inference 4 times while replacing GBDT T5 classic and GBDT T5 conditional models with the following:

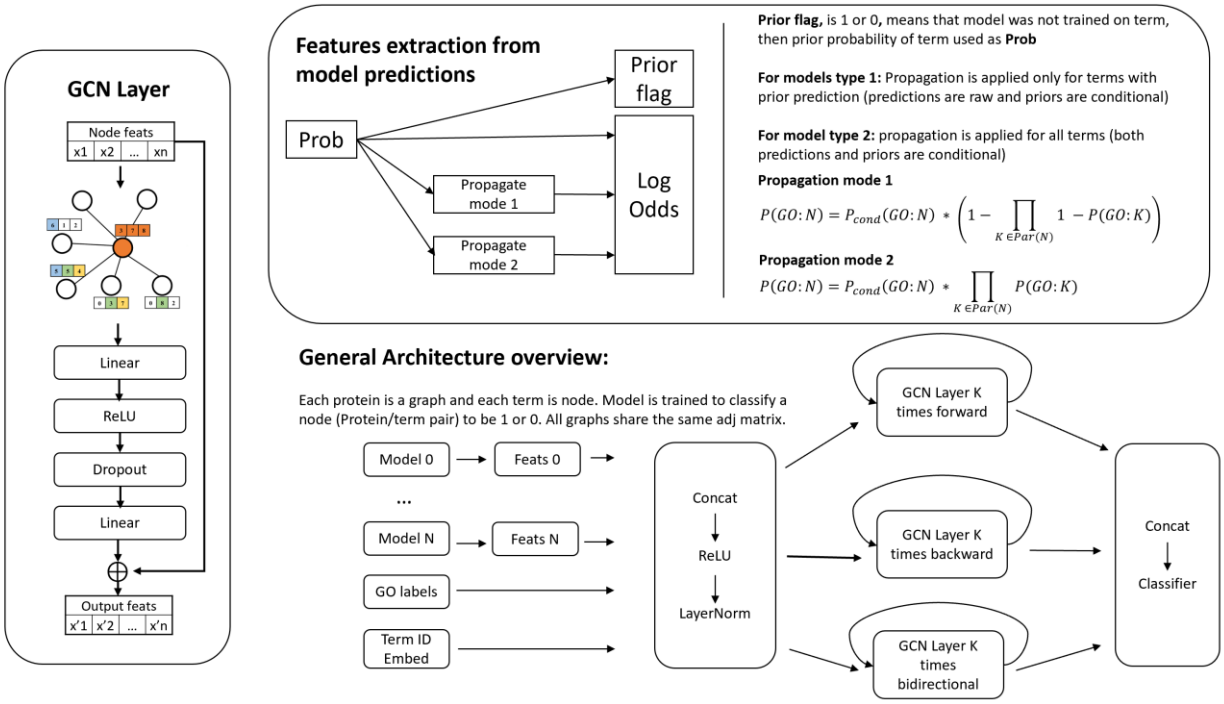
- GBDT T5 classic/GBDT T5 conditional
- GBDT T5 + ESM classic/GBDT T5 conditional

- GBDT T5 classic/GBDT T5 + ESM conditional
- GBDT T5 + ESM classic/GBDT T5 + ESM conditional

Train sample

Another important thing I should mention is about the metric. As far as I understand, we are evaluated only on protein/ontology pairs that are experimentally found. So, if ones we predict a term from ontology that does not exist, we will not get any penalty at all! That means, that we need actually to estimate **the conditional probability of term in case when its ontology exists.** So, the correct way to fit a final stacker model for ontology X is to truncate the sample and take only the proteins that contain the terms from ontology X. It gives some small boost to the score and speed-up the computations.

The following Figure represents the details of stacking model implementation:



GO annotations

We discovered the GO annotations dataset provided by the link. They provide not only labelling but also the evidence codes of each term. We separated the codes that Kaggle suppose to be experimental (hope we understood it correctly) from the electronic codes. So, we can use electronic labelling as the features to predict experimental given by Kaggle. From our analysis we discovered that about 30% of electronic labels becomes experimental, so using it as model feature performs better than just adding it as is. Experimental labelling was added as is for about 500 protein/term pairs, that we were able to find in this dataset. We also added raw labelling that MT provided if we can not observe it in our dataset. The last step almost didn't change our score, so that sources are almost the same.

After the competition finished, we restore those MT's data sources as the difference between different repot dates of GO annotations. We made it to make the solution exactly reproducible without using side datasets.

Postprocessing

We also used the OBO graph to make a post processing. The problem of ML model prediction of protein terms is that it is inconsistent. Following to propagation rule that is applied to the target (I checked it in the CAFA-Evaluator repo) if term exists, all its parents assumed to exist too. So, consistent model will never predict the probability for parent lower than term probability. But our models don't care about it at all. So we can manually fix this situation. Our final term prediction is **the average of term probability, maximum propagated children probability, and minimum propagated parents probability**. That trick boosts score just a little on the LB but hope it makes model a little bit more stable.

The figure below illustrated the postprocessing details:

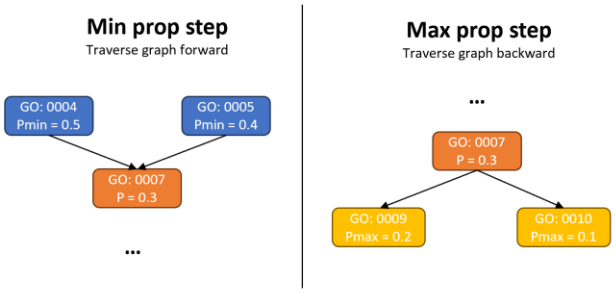
General rules:

Postprocessing is made in order defined by OBO Graph. After term N is processed, its probability is modified inplace and for next terms $P_{min}(N)$ will be used instead of $P(N)$. Min and max propagations are done separately and then results are averaged

$$P_{min}(N) = Min\{P_{min}(k), k \in Par(N)\} * 0.7 + P(N) * 0.3$$

$$P_{max}(N) = Max\{P_{max}(m), m \in Chld(N)\} * 0.7 + P(N) * 0.3$$

$$P_{post}(N) = \frac{P_{min}(N) + P_{max}(N)}{2}$$



Example:

For term GO:0007 we will calculate the final value as:

$$P_{min}(N) = Min\{0.5, 0.4\} * 0.7 + 0.3 * 0.3 = 0.37$$

$$P_{max}(N) = Max\{0.2, 0.1\} * 0.7 + 0.3 * 0.3 = 0.223$$

$$P_{post}(N) = (0.37 + 0.223) * 0.5 = 0.296$$

Ablation (based on public LB)

Here is brief evolution of our solution (scores are approximate):

1. Single py-boost over T5 + alternative modelling: 0.51
2. Step 1 + taxon data: 0.53
3. Blend of 2 py-boost and 2 LogRegs (classic and conditional): 0.55
4. Step 3 + GOA test leakage: 0.57
5. Step 4 + NN base model: 0.59
6. Stacking + GOA test leakage: 0.62

Simplified model

We believe, it does not make sense to choose one best and simple approach, since:

1. There is no need to make fast and efficient online inference for such kind of research task
2. Any single model scores much less that combination of multiple approaches

We consider to use all the models and data sources to evaluate all the potential protein functions, except adding the potential leakage data of course. But using GO electronic labelling as feature in stacker model is valid knowledge to predict, which terms will be experimentally found.

Model execution time

Here we provide the approximate runtime of each step. All time is provided for single V100 32GB GPU:

1. **Preparation step**. All the preprocessing takes about **1-2 hours**, excluding the time for downloading FTP GOA data sources (it takes maybe 10 hours)
2. **Embedding** inference takes about **3 hours for each model** type. If 2 GPUs are available, inference could be done in parallel
3. **Base models** (all time listed for 5 fold CV). All models could be trained in parallel. Inference is quite fast and takes much less time that train (few minutes for each model):
    a. **py-boost**: 4 models, each takes **15 hours**
    b. **LogReg**: one model **10 hours**, another **2 hours**
    c. **Neural Net**: one model blended 12 times **1 hour**
4. **Stacking train** (single model for each ontology), could be trained in parallel:
    a. **BP: 13 hours**
    b. **MF: 5 hours**
    c. **CC: 2 hours**
5. **Stacking inference** (4 times TTA), could be inferred in parallel:
    a. **BP: 1.5 hours**
    b. **MF: 40 mins**
    c. **CC: 20 mins**
6. **Postprocessing: 5 mins**

Source Code

Solution code is open source now and available at https://github.com/btbpanda/CAFA5-protein-function-prediction-2nd-place