Match removal in DPCfam dataset

Lorenzo Basile, project developed during internship at AREA Science Park.

The problem

The general aim of this project is to remove as many matches as possible from the DPCfam dataset while trying to maintain as many data points (hits) as possible.

A match happens when overlapping parts of the same protein are assigned to two different clusters (families). To be precise, this means that a match happens when two datapoints of the DPCfam dataset have equal protein name, different family name and alignment coordinates $from1$, $to1$, $from2$, $to2$ such that:

$$\frac{|[from1, to1] \cap [from2, to2]|}{|[from1, to1] \cup [from2, to2]|} > 0.8$$

The way chosen to remove matches is the setting of global, family-specific evalue thresholds. This means that if a certain threshold t_f is chosen for family f all data points clustered in family f with an evalue greater than t_f are discarded from the dataset.

With this choice, since the evalues range between 0 and 1, choosing a threshold of 0 for a certain family would lead to the removal of all matches involving that family (but, more generally, to the removal of all hits, even the non-matching ones), while setting the threshold to 1 leaves the family untouched.

It is clear that the two objectives (match removal and data conservation) are contrasting: match removal will lead inevitably to some data loss.

One possible way to find an optimal set of thresholds is by means of gradient-based optimization. This technique requires the definition of a loss function to minimize. A possible choice is the following:

$$\mathcal{L}(\mathbf{t}) = \mathcal{L}_{data}(\mathbf{t}) + \mathcal{L}_{matches}(\mathbf{t})$$

With:

$$\mathcal{L}_{data}(\mathbf{t}) = \sum_{f \in families} \frac{|evalues(f) > t_f|}{|f|}$$

and

$$\mathcal{L}_{matches}(\mathbf{t}) = \frac{|surviving\ matches(\mathbf{t})|}{|matches|} = \frac{1}{|matches|} \sum_{f,g \in families} |surviving\ matches(t_f, t_g)|$$

This loss definition by itself is still not sufficient to use gradient-based optimization. In fact, all these functions are stepwise functions of the thresholds. Another step is required to approximate these functions with smoother, differentiable functions, suitable for gradient descent.

Neural approximation of loss functions

Approximation of $\mathcal{L}_{data}$

$\mathcal{L}_{data}$ can be approximated with a set of $|families|$ identically constructed (but not sharing weights) simple neural networks. Each network has 3 hidden layers, takes one input (the threshold t_f for a certain family f) and returns one number (an approximation of $\frac{|evalues(f)>t_f|}{|f|}$).

Training of these networks is performed independently one from each other using randomly sampled input batches. Since the variability of the functions to approximate is mostly around 0, inputs x are sampled as follows:

$$u \sim U(0,1)$$

$$exp \sim Categorical(\{0,1,\dots,10\})$$

$$x = u \cdot 10^{-exp}$$

Learning is then performed using L_2 loss and `AdamW` optimizer (for some reason it works better than plain `Adam`) until convergence.

Convergence is assumed to be reached for a given family when the loss in one epoch is lower than a given threshold and the value of the learned approximation in 1 is reasonably close to 0. This last requirement is heuristic and comes from two facts: because of the sampling technique, requiring only the loss to be low does not guarantee convergence around 1 and the functions we are approximating take always value 0 when evaluated in 1.

Here, a possible improvement in terms of performance may come from the functional similarity between the family-specific losses to be approximated. Since $\frac{|evalues(f)>t_f|}{|f|}$ has a similar behaviour for all families, one could decide to train the approximator from scratch for the first family and then perform fine-tuning from the weights of the first approximator for the following ones, potentially speeding-up the process. However, experimentally, at first sight this speed-up looks negligible.

Approximation of $\mathcal{L}_{matches}$

Very similar considerations apply for the approximation of $\mathcal{L}_{matches}$. It is approximated using a set of potentially $|families|^2$ identically constructed neural networks (but in practise it is sufficient to only consider pairs of families which have many matches, greatly reducing the number of networks to build and train).

Sampling and training are conducted exactly as explained before apart from the following modifications:

- input is 2-dimensional (we need a threshold for each family of the pair), so two input numbers should be sampled at each epoch;
- the final heuristic requirement is switched: this time the approximation is required to be close to 1 when evaluated in the point $(1,1)$.

One thing to point out is that the two sets of neural networks can be trained simultaneously: there are no dependencies between the two approximation.

Gradient Descent

Finally, once the global loss can be approximated by means of neural networks, gradient descent can be performed to obtain an optimal set of thresholds. This can be done by means of a ("fake") neural network with just one linear layer and without biases, that takes a constant input of 1 and has $|families|$ outputs. Doing so, the thresholds are the weights of this neural network.

Running the codes

1. run `readfile.py`;
2. run `match_finder.py`;
3. run `family_evalues.py`;
4. run `ldata_approximation.py` and `lmatches_approximation.py`. This step is the training of the two sets of neural approximators and can be conducted in parallel.
5. run `descent.py`.