TEGym manual

Tiago Minuzzi

March, 2024

Summary

What is TEGym?

TEGym is a program written in python to help people without deep learning expertise create a data driven transposable elements classifier, *i.e.*, a model to classify transposable elements of species lacking enough data to train a classifier, using the data from a more closely related species. It automatizes preprocessing, hyperparameters testing and model training, resulting in a classifier suited for the needs of the user. Although TEGym was developed with transposable elements in mind, it can probably be used for other sequence classification tasks.

TEGym is a work in progress to the date of this writing. We are trying to add more options and improvements as soon as possible.

Installation

TEGym uses python version 3.11. Preferentially use python version >= 3.10 in a python virtual enviroment or a conda enviroment. Install the required packages using:

```
pip install -r requirements.txt
```

Basic usage

The most basic usage is simple. You only need a `FASTA` file or a `CSV` file contaning the sequences and the labels. The `CSV` must be like the following:

label	sequence
TE	ctagctagtgac...
TE	gctagctagcat...
NonTE	atcgtagctgct...
NonTE	tacgtgatctag...

In the case of the `FASTA` file, the headers/identifiers must be in the `RepeatMasker` format (`>sequenceId#label`). Your `FASTA` must look something like:

```
>seq01#TE
actactgatgcatga...
```

```
>seq02#NonTE
tgactgtagcttgtat...
```

The comand for running using a FASTA file is:

```
python gym.py -f my_file.fasta
```

If using a CSV file just change the flag -f to -c as in:

```
python gym.py -c my_file.csv
```

Both of the aforementioned commands will initiate the program with default settings. The initial phase involves searching for the optimal hyperparameters to train the model based on the input dataset. This step is the most time-consuming. Upon completion, a CSV file will be generated containing a table with the resultant hyperparameters. Subsequently, the model will be trained using the best combination of hyperparameters, determined by the lowest validation loss.

Recommendations

We recommend a minimum of 2000 sequences for each label to achieve a better learning performance of the model. Ideally, each label should have an equal number of sequences. However, if this is not possible, TEGym adjusts automatically by giving weights to the labels based on their availability. The label with the highest number of sequences should not exceed 4x times the total of the label with the lowest number of sequences. Otherwise, the model may exhibit bias towards the majority, even when using class weights. Your dataset can contain as many labels as desired. However, it's important to note that too few sequences for a given label will likely lead to poor learning outcomes.

Running steps independentely

Instead of running hyperparameter search and model training all at once, you can run the steps independentely. Just call the script hyperparameters.py to generate the CSV to be used for model training later. Then, when calling gym.py use the flag -p to indicate the path to the hyperparameter's CSV.

Example:

```
python hyperparameter.py -f my_file.fasta

python gym.py -f my_file.fasta \
              -p TEGym_hyperparameters.csv
```

Customizing hyperparameters search

If you want to set-up other values for hyperparameter searching different than the default values used by TEGym, you just need to modify the values in the `TOML` file `my_config_hyperparameters.toml`. Do NOT change the name of values before the = sign, just the values inside square brackets, which must be comma separated.

Other flags

Here are some other options allowed by the program. Below we show the commands using each flag individually, but you can use any of them together.

Metric

As stated in the section **`Basic usage`**, combination of hyperparameters used to train the model is sorted by the lowest validation loss, but this can be changed to use the highest validation accuracy. Just pass the flag `-m` with the value `val_accuracy`.

Example:

```
python gym.py -f my_file.fasta -m val_accuracy
```

Title

This flag just adds a customized prefix to the output files generated by the program. Use `-t` to modify the default value (TEgym).

Example:

```
python gym.py -f my_file.fasta -t my_model
```

Runs

The flag `-r` changes how many combinations of hyperparameters will be tested. The default value is `20`, use the flag as follows to change it.

Example:

```
python gym.py -f my_file.fasta -r 50
```

Split

Split is the percentage, in decimal form, of the dataset that will be used as validation set to evaluate the accuracy of the model. The default value is `0.1` (10%). This means that 90% of the dataset is used to train the model and 10% to test it. Change it using the command as in the example below.

Example:

```
python gym.py -f my_file.fasta -s 0.2
```

Help

To show the program usage and what each flag is just use the flag `-help`. It can be used for the modules `gym.py` and `hyperparameters.py`.

Example:

```
python gym.py --help
```

FASTA to CSV

When using a `FASTA` file as input, the program will convert it to a `CSV` file. Depending on the size of your `FASTA`, it may be time-consuming. You can convert you `FASTA` to `CSV` prior to running the program using the script `fasta_to_csv.py` as follows:

```
python fasta_to_csv.py my_file.fasta
```

Predict

After your model is trained using `gym.py`, you can use it as a classifier by running the script `predict.py`. It has three mandatory arguments: an `FASTA` file with sequences to be classified, the path to the trained model and the path to the `TOML` file with model info.

Example:

```
python predict.py -f file.fasta \
                  -m my_model.keras \
                  -i my_model_info.toml
```

The output is a `CSV` file containing the classification prediction for each sequence and the classication score ranging from 0 to 1.

Example:

id	prediction	TE_score	NonTE_score
Seq01	TE	0.98	0.02
Seq02	TE	0.72	0.28
Seq03	NonTE	1.0	0.0
Seq04	NonTE	0.63	0.37
Seq05	TE	0.85	0.15

Create negative class

If your dataset has only one class, for instance, only sequences labeled as TE (transposable element), you can create a negative class, *i.e.*, a NonTE class by running the script `create_negative_class.py`. It will create random sequences using the value `random` or, using the value `shuffled`, it will shuffle the sequences of your dataset, creating non-sense sequences that can be used as a negative value. Choose one type and pass the value with the flag `-c`.

Example:

```
python create_negative_class.py -f my_file.fasta \
                                -c shuffled
```

The output is a `CSV` file with the prefix `TDS` (standing for Training Data Set) with your sequences and the generated sequences. Then, you can use it with the main program.