genomeNLP

Tyrone Chen

Apr 17, 2024

CONTENTS:

Visit our lab website here. Contact Sonika Tyagi at sonika.tyagi@monash.edu.

Note: The main repository is on github but also mirrored on gitlab. Please submit any issues to the main github repository only.

CHAPTER

ONE

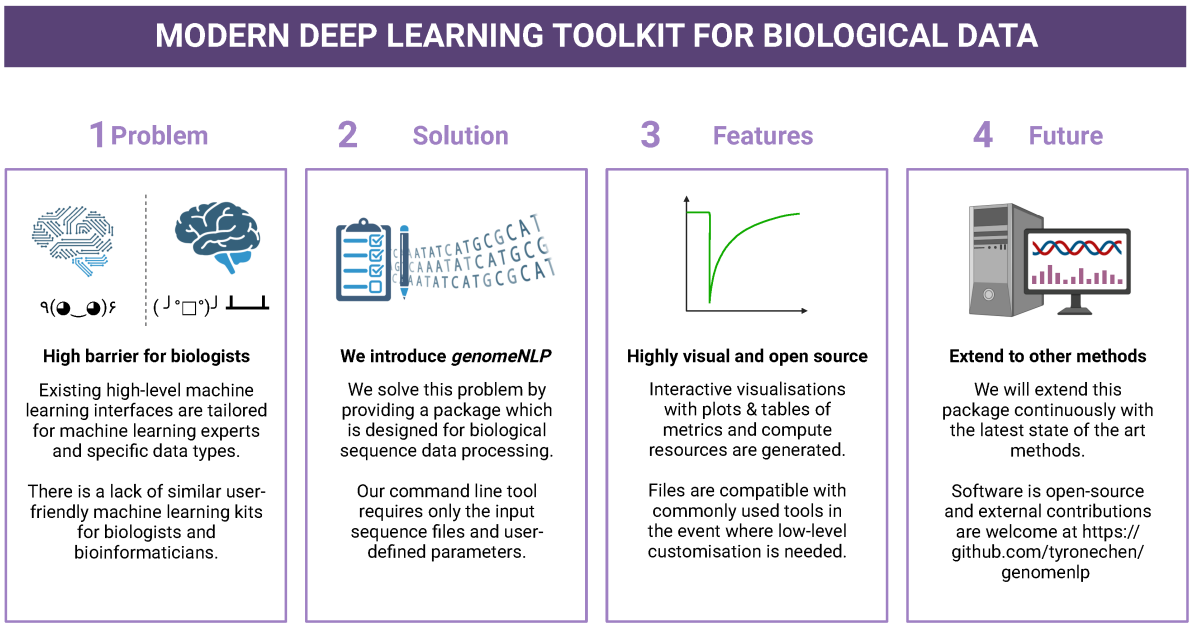
GENOMENLP: GENOME RECODING FOR MACHINE LEARNING USAGE INCORPORATING GENOMICBERT

Visit our lab website here. Contact Sonika Tyagi at sonika.tyagi@monash.edu.

Note: The main repository is on github but also mirrored on gitlab. Please submit any issues to the main github repository only.

1.1 Highlights

- We provide a comprehensive classification of genomic data tokenisation and representation approaches for ML applications along with their pros and cons.
- Using our `genomicBERT` deep learning pipeline, we infer k-mers directly from the data and handle out-of-vocabulary words. At the same time, we achieve a significantly reduced vocabulary size compared to the conventional k-mer approach reducing the computational complexity drastically.
- Our method is agnostic to species or biomolecule type as it is data-driven.
- We enable comparison of trained model performance without requiring original input data, metadata or hyperparameter settings.
- We present the first publicly available, high-level toolkit that infers the grammar of genomic data directly through artificial neural networks.
- Preprocessing, hyperparameter sweeps, cross validations, metrics and interactive visualisations are automated but can be adjusted by the user as needed.



1.2 Cite us with:

Manuscript:

```
@article{chen2023genomicbert,
  title={genomicBERT and data-free deep-learning model evaluation},
  author={Chen, Tyrone and Tyagi, Navya and Chauhan, Sarthak and Peleg, Anton Y and
↪Tyagi, Sonika},
  journal={bioRxiv},
  month={jun},
  pages={2023--05},
  year={2023},
  publisher={Cold Spring Harbor Laboratory},
  doi={10.1101/2023.05.31.542682},
  url={https://doi.org/10.1101/2023.05.31.542682}
```

}

Software:

```
@software{chen_tyrone_2023_8143218,
author       = {Chen, Tyrone and
                Tyagi, Navya and
                Chauhan, Sarthak and
                Peleg, Anton Y. and
                Tyagi, Sonika},
title        = {{genomicBERT and data-free deep-learning model
                 evaluation}},
```

(continues on next page)

(continued from previous page)

```
month        = jul,
year         = 2023,
note         = {If you use this software, please cite it as below.},
publisher    = {Zenodo},
version      = {latest},
doi          = {10.5281/zenodo.8135590},
url          = {https://doi.org/10.5281/zenodo.8135590}
}
```

1.3 Install

1.3.1 Mamba (automated)

This is the recommended install method as it automatically handles dependencies. Note that this has only been tested on a linux operating system.

Note: Installing with `mamba` is highly recommended. Installing with `pip` will not work. Installing with `conda` will be slow. You can find instructions for setting up mamba here. Please submit any issues to the main github repository only.

First try this:

```
mamba install -c conda-forge -c tyronechen genomenlp
```

If there are any errors with the previous step (especially if you are on a cluster with GPU access), try this first and then repeat the previous step:

```
mamba install -c anaconda cudatoolkit
```

If neither works, please submit an issue with the full stack trace and any supporting information.

1.3.2 Mamba (manual)

Clone the git repository. This will also allow you to manually run the python scripts.

Then manually install the following dependencies with `mamba`. Installing with `pip` will not work as some distributions are not available on `pip`.:

```
gensim==4.2.0
hyperopt==0.2.7
matplotlib==3.5.2
pandas==1.4.2
pytorch==1.10.0
ray==1.13.0
scikit-learn==1.1.1
screed==1.0.5
seaborn==0.11.2
sentencepiece==0.1.96
tokenizers==0.12.1
```

(continues on next page)

(continued from previous page)

```
tqdm==4.64.0
transformers==4.30.0
wandb==0.13.4
weightwatcher==0.5.9
xgboost==1.7.1
yellowbrick==1.3.post1
```

You should then be able to run the scripts manually from `src/genomenlp`. As with the automated step, `cudatoolkit` may be required.

1.4 Usage

1.4.1 0. Available commands

If installed correctly using the automated `mamba` method, each of these commands will be available directly on the command line:

```
create_dataset_bio
create_dataset_nlp
create_embedding_bio_sp
create_embedding_bio_kmers
cross_validate
embedding_pipeline
fit_powerlaw
freq_pipeline
generate_synthetic
interpret
kmerise_bio
parse_sp_tokens
summarise_metrics
sweep
tokenise_bio
train
```

If installed correctly using the manual `mamba` method, each of the above commands can be called from their corresponding python script, for example:

```
python create_dataset_bio.py
```

1.4.2 1. Preprocessing

Tokenise the biological sequence data into segments using either empirical tokenisation or conventional k-mers. Provide input data as gzipped fasta files. Empirical tokenisation is a two-step process, while in k-merisation the tokenisation and dataset creation is performed in the same operation. Both methods generate data compatible with the `genomicBERT` pipeline.

Empirical tokenisation pathway:

```
tokenise_bio -i [INFILE_PATH ... ] -t TOKENISER_PATH
create_dataset_bio <INFILE_SEQS_1> <INFILE_SEQS_2> <TOKENISER_PATH> -c CHUNK -o OUTFILE_
↪DIR
```

Conventional k-mers pathway:

```
# LABEL must match INFILE_PATH! assume that one fasta file has one seq class
kmerise_bio -i [INFILE_PATH ... ] -t TOKENISER_PATH -k KMER_SIZE -l [LABEL ... ] -c␣
↪CHUNK -o OUTFILE_DIR
create_dataset_bio <INFILE_SEQS_1> <INFILE_SEQS_2> <TOKENISER_PATH> -c CHUNK -o OUTFILE_
↪DIR
```

Embedding pathway (input files here are `csv` created from previous step):

```
# after the empirical tokenisation pathway::
create_embedding_bio_sp -i [INFILE_PATH ... ] -t TOKENISER_PATH -o OUTFILE_DIR

# after the conventional k-mers pathway::
create_embedding_bio_kmers -i [INFILE_PATH ... ] -t TOKENISER_PATH  -o OUTFILE_DIR
```

Note: The `CHUNK` flag can be used to partition individual sequences into smaller chunks. `512` is a good starting point. The `--no_reverse_complement` flag should be used where non-DNA sequences are used. Vocabulary size can be set with the `--vocab_size` flag. For generating embeddings, number of threads can be set with `--njobs`.

1.4.3 2. Classification

Feed the data preprocessed in the previous step into the classification pipeline. Set `freq_method` and `model` combination as needed. Hyperparameter sweeping is performed by default. For non-deep learning methods, cross-validation is performed in the same operation.

Deep learning with the `genomicBERT` pipeline requires a `wandb` account set up and configured to visualise interactive plots in real time. Please follow the instructions on wandb to configure your own account.

Frequency-based approaches:

```
freq_pipeline -i [INFILE_PATH ... ] --format "csv" -t TOKENISER_PATH --freq_method [␣
↪cvec | tfidf ] --model [ rf | xg ] --kfolds N --sweep_count N --metric_opt [ accuracy␣
↪| f1 | precision | recall | roc_auc ] --output_dir OUTPUT_DIR
```

Embedding:

```
embedding_pipeline -i [INFILE_PATH ... ] --format "csv" -t TOKENISER_PATH --freq_method␣
↪[ cvec | tfidf ] --model [ rf | xg ] --kfolds N --sweep_count N --metric_opt [␣
↪accuracy | f1 | precision | recall | roc_auc ] --output_dir OUTPUT_DIR
```

Note: `--model_features` can be set to reduce the number of features used. Number of threads can be set with `--njobs`. `--sweep_method` can be set to change search method between `[ grid | random ]`.

`genomicBERT` deep learning pipeline:

```
sweep <TRAIN_DATA> <FORMAT> <TOKENISER_PATH> --test TEST_DATA --valid VALIDATION_DATA --
↪hyperparameter_sweep PARAMS.JSON --entity_name WANDB_ENTITY_NAME --project_name WANDB_
↪PROJECT_NAME --group_name WANDB_GROUP_NAME --sweep_count N --metric_opt [ eval/
↪accuracy | eval/validation | eval/loss | eval/precision | eval/recall ] --output_dir␣
↪OUTPUT_DIR

# use the WANDB_ENTITY_NAME, WANDB_PROJECT_NAME and the best run id corresponding to the␣
↪sweep
# WANDB_GROUP_NAME should be changed to reflect the new category of runs (eg "cval")
cross_validate <TRAIN_DATA> <FORMAT> --test TEST_DATA --valid VALIDATION_DATA --entity_
↪name WANDB_ENTITY_NAME --project_name WANDB_PROJECT_NAME --group_name WANDB_GROUP_NAME␣
↪--kfolds N --config_from_run WANDB_RUN_ID --output_dir OUTPUT_DIR
```

Note: You can provide the hyperparameter search space with a `json` file to `--hyperparameter_sweep`. The `label_names` argument here is different from previous steps and refers to the column name containing labels, not a list of class labels. Set `--device cuda:0` if you have `cuda` installed and want to use GPU.

```
{
  "name" : "random",
  "method" : "random",
  "metric": {
    "name": "eval/f1",
    "goal": "maximize"
  },
  "parameters" : {
    "epochs" : {
      "values" : [1, 2, 3]
    },
    "batch_size": {
        "values": [8, 16, 32, 64]
        },
    "learning_rate" :{
      "distribution": "log_uniform_values",
      "min": 0.0001,
      "max": 0.1
      },
    "weight_decay": {
        "values": [0.0, 0.1, 0.2, 0.3, 0.4, 0.5]
      }
  },
  "early_terminate": {
      "type": "hyperband",
      "s": 2,
      "eta": 3,
      "max_iter": 27
  }
}
```

1.4.4 3. Comparing deep learning models trained by genomicBERT

The included method only works on deep learning models, including those trained through the `genomicBERT` pipeline. For more information on the method, including interpretation, please refer to the publication (https://arxiv.org/pdf/2202.02842.pdf).

```
fit_powerlaw -i [ INFILE_PATH ... ] -t OUTPUT_DIR -a N
```

1.4.5 4. Case study

A detailed case study is available for reference.

1.5 Background

To be written

1.6 Acknowledgements

TC was supported by an Australian Government Research Training Program (RTP) Scholarship and Monash Faculty of Science Dean's Postgraduate Research Scholarship. ST acknowledges support from Early Mid-Career Fellowship by Australian Academy of Science and Australian Women Research Success Grant at Monash University. AP and ST acnowledge MRFF funding for the SuperbugAI flagship. This work was supported by the MASSIVE HPC facility (https://www.massive.org.au) and the authors thank the Monash Bioinformatics Platform as well as the HPC team at Monash eResearch Centre for their continuous personnel support. We thank Yashpal Ramakrishnaiah for helpful suggestions on package management, code architecture and documentation hosting. We thank Jane Hawkey for advice on recovering deprecated bacterial protein identifier mappings in NCBI. We thank Andrew Perry and Richard Lupat for helping resolve an issue with the python package building process. Biorender was used to create many figures in this publication. We acknowledge and pay respects to the Elders and Traditional Owners of the land on which our 4 Australian campuses stand (https://www.monash.edu/indigenous-australians/about-us/recognising-traditional-owners).

CHAPTER

TWO

GENOMENLP: CASE STUDY OF DEEP LEARNING

2.1 Outline

The primary focus of this tutorial is application of NLP in a genomic context by introducing our package `genomenlp`. In this tutorial, we cover a wide range of topics from introduction to field of GenomeNLP to practical application skills of our `conda` package, divided into various sections:

1. Introduction to GenomeNLP
2. Connection to a remote server
3. Installing conda and genomenlp package
4. Setting up a Biological Dataset
5. Format a dataset as input for genomenlp
6. Preparing a hyperparameter sweep
7. Selecting optimal parameters
8. With the selected hyperparameters, train the full dataset
9. Performing cross-validation
10. Comparing performance of different models
11. Obtain model interpretability scores

For detailed usage of individual functions, please refer to the latest documentation.

2.1.1 Learning objectives

- Describe the unique challenges in biological NLP compared to conventional NLP
- Understand common representations of biological data
- Understand common biological data preprocessing steps
- Investigate biological sequence data for use in machine learning
- Perform a hyperparameter sweep, training and cross-validation
- Identify what the model is focusing on

- Compare trained model performances to each other

Note: *This is **not** an introductory machine learning workshop. Readers of this tutorial are assumed to be familiar with the use of the command line and of the basics of machine learning.*

2.1.2 Potential/preferred prerequisite knowledge

- [required] CLI (e.g. bash shell) usage
- [optional] Connecting and working on a remote server (e.g. `ssh`)
- [optional] Basic knowledge of machine learning
- [optional] Machine learning dashboards (e.g. `tensorboard`, `wandb`)
- [optional] Package/environment managers (e.g. `conda`, `mamba`)

Length: Half-day, 4.0 - 4.5 hours Intended audience: machine learning practitioners OR computational biologists

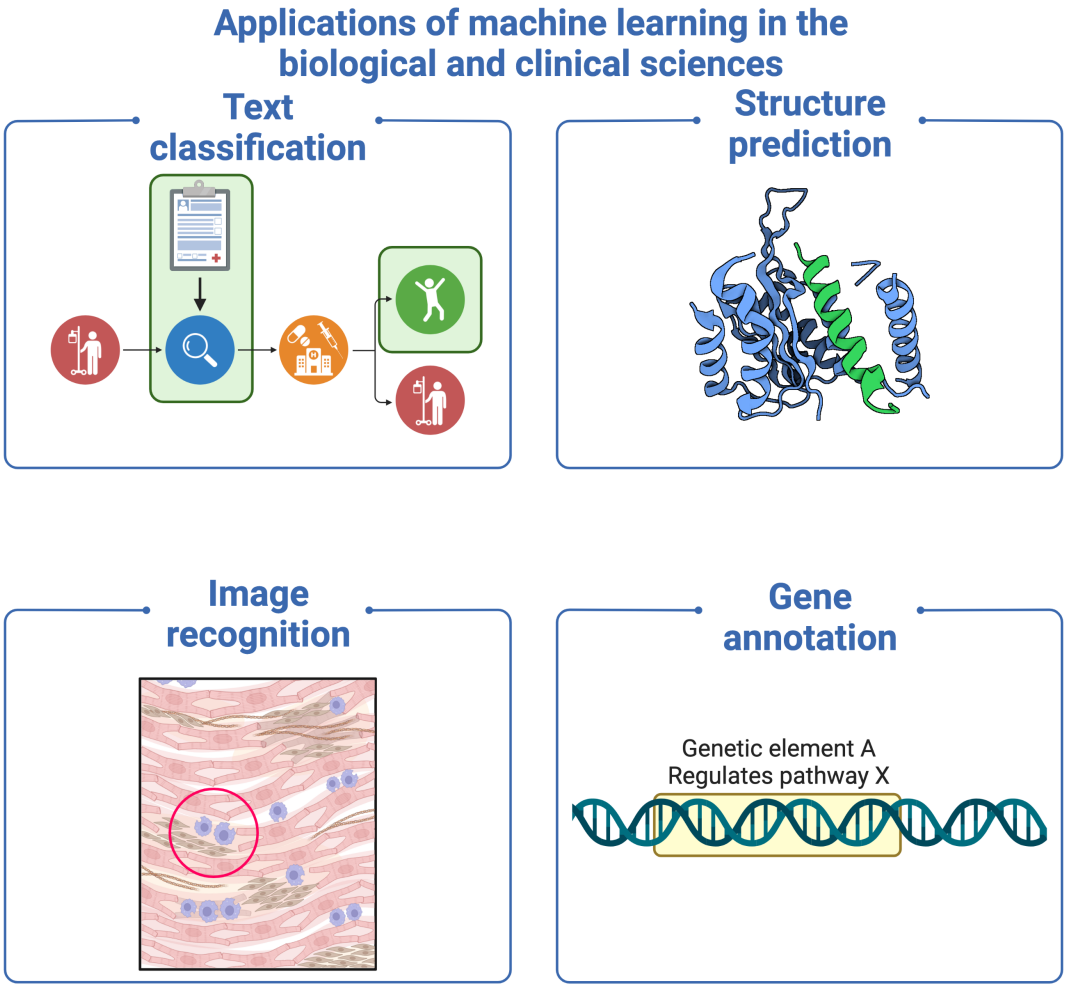
2.1.3 Glossary

- BERT - Bidirectional Encoder Representations from Transformers, a family of deep learning architectures used for NLP.
- DL - Deep Learning
- k-mers - Identical to tokens
- k-merisation - A process where a biological sequence is segmented into substrings. Commonly performed as a sliding window.
- ML - Machine Learning
- NLP - Natural Language Processing
- OOV - Out-of-vocabulary words
- Sliding window - ABCDEF: [ABC, BCD, CDE, DEF] instead of [ABC, DEF]
- Tokenisation - A process where a string is segmented into substrings
- Tokens - Subunits of a string used as input into conventional NLP algorithms

2.2 1. Introduction

2.2.1 What is NLP and genomics

Natural Language Processing (NLP) is a branch of computer science focused around the understanding of and the processing of human language. Such a task is non-trivial, due to the high variation in meaning of words found embedded in different contexts. Nevertheless, NLP is applied with varying degrees of success in various fields, including speech recognition, machine translation and information extraction. A recent well-known example is ChatGPT.

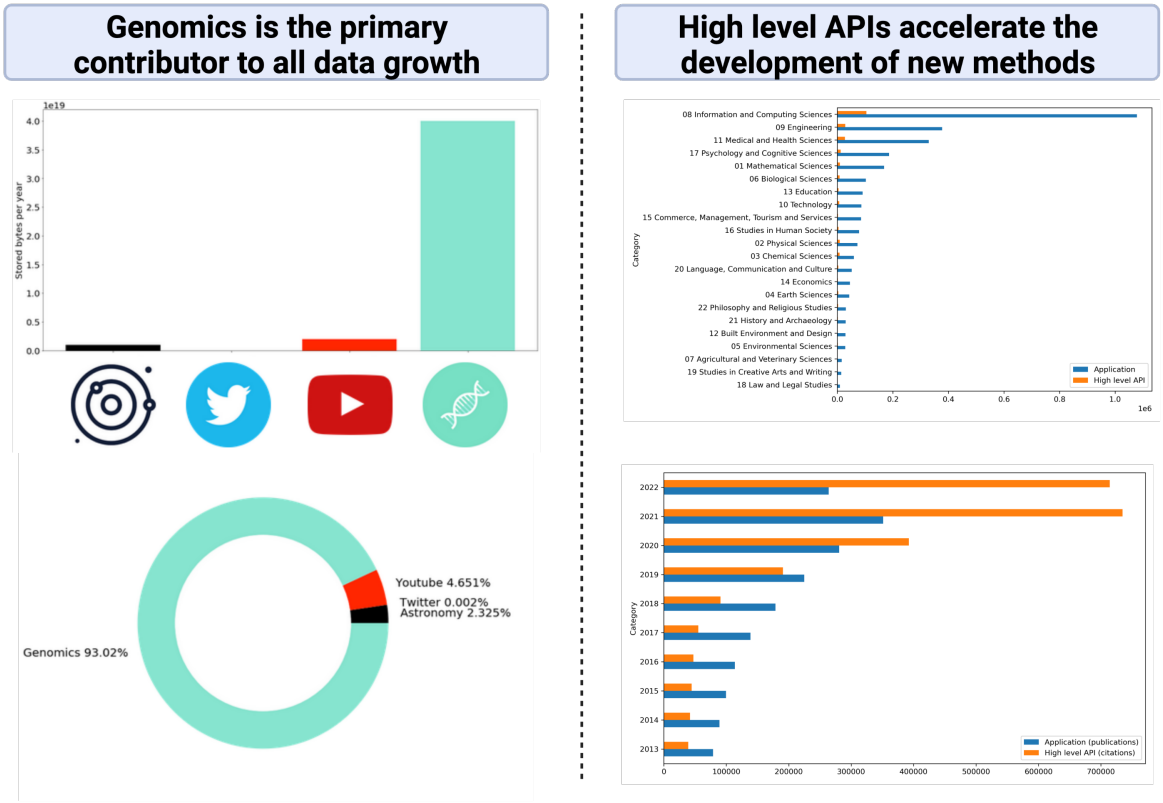


Meanwhile, genomics involves the study of the genome, which contains the entire genetic content of an organism. As the primary blueprint, it is an important source of information and underpins all biological experiments, directly or indirectly.

2.2.2 Why apply NLP in genomics

Although NLP has been shown to effectively preprocess and extract "meaning" from human language, until recently, its application in biology was mostly centred around biological literature and electronic health record mining. However, we note the striking similarities between genomic sequence data and human languages that make it well-suited to NLP. (A) DNA can be directly expressed as human language, being composed of text strings such as A, C, T, G, and having its own semantics as well as grammar. (B) Large quantities of biological data are available in the public domain, with a growth rate exponentially exceeding astronomy and social media platforms combined. (C) Recent advances in machine learning which improve the scalability of deep learning (DL) make computational analysis of genomic data feasible.

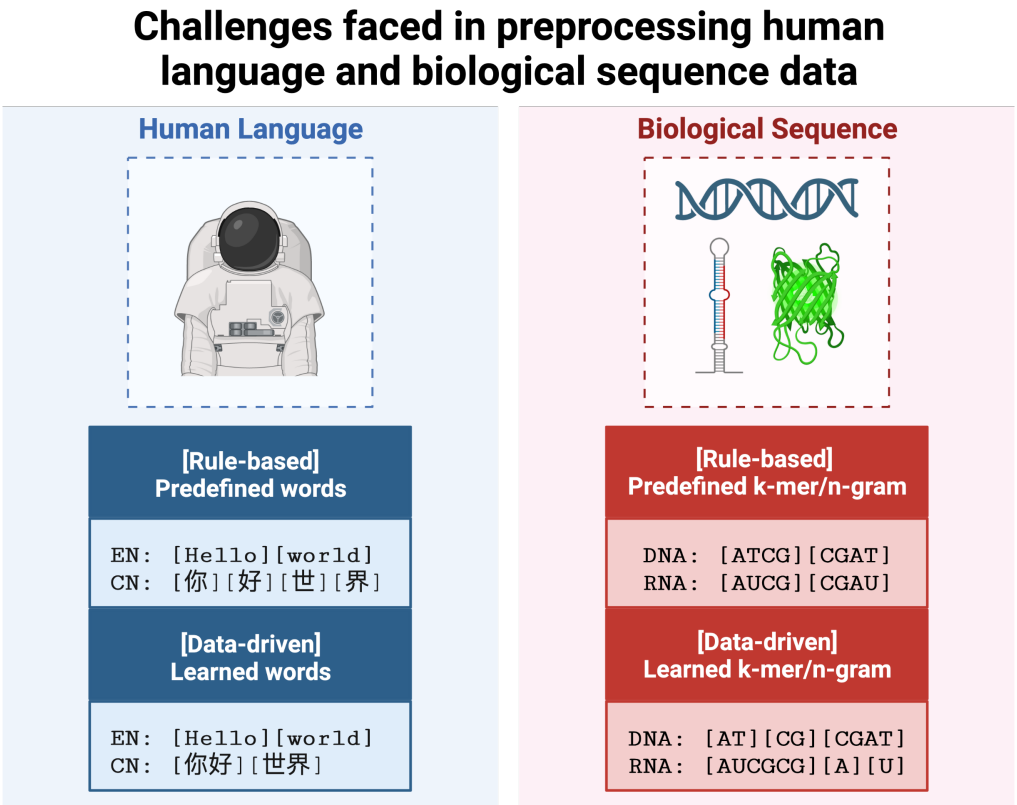
Note: *The same is true for protein sequences, and nucleic acid data such as transcripts. While our pipeline can process any of these, the scope of this tutorial is for genomic data only.*



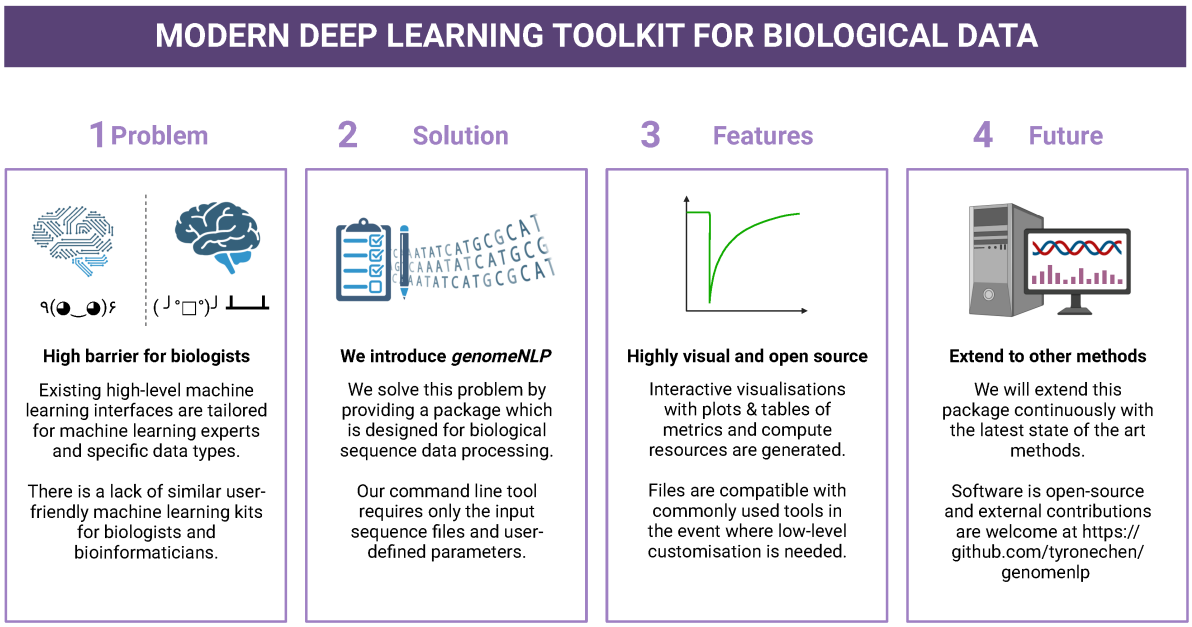
We therefore make a distinction between the field of conventional literature or electronic health record mining and the application of NLP concepts and methods to the genome. We call this field *genome NLP*. The aim of *genome NLP* would be to extract relevant information from the large corpora of biological data generated by experiments, such as gene names, point mutations, protein interactions and biological pathways. Applying concepts used in NLP can potentially enhance the analysis and interpretation of genomic data, with implications for research in personalised medicine, drug discovery and disease diagnosis.

2.2.3 Distinction between conventional NLP and genome NLP

Several key differences need to be accounted for for implementing NLP on the genome. (A) The first challenge is the tokenisation of long biological sequences into smaller subunits. While some natural languages have subunits separated by spaces, enabling easy segmentation, this is not true in biological sequence data, and also to an extent in many languages such as Arabic, Mandarin or Sanskrit characters. (B) A second challenge is the diversity and high degree in nuance of biological experiments. As a result, interpretability and interoperability of biological data is highly restricted in scope, even within a single experiment. (C) The third challenge is the difficulty in comparing models, partly due to the second challenge, and partly due to the lack of accessible data in the biomedical field for privacy reasons, and partly because of the limited enforcement of biological data integrity as well as metadata by journals. In addition, the large volume of biological data in a single experiment makes re-training time consuming.



To address the challenges in genome-NLP, we used a new semi-automated workflow. This workflow integrates feature engineering and machine learning techniques and is designed to be adaptable across different species and biological sequences, including nucleic acids and proteins. The workflow includes the introduction of a (1) new tokeniser for biological sequence data which effectively tokenises contiguous genomic sequences while retaining biological context. This minimises manual preprocessing, reduces vocabulary sizes, and (2) handles unknown biological terms, conceptually identical to the out-of-vocabulary (OOV) problem in natural languages. (3) Passing the preprocessed data to a `genomicBERT` algorithm allows for direct biological sequence input to a state-of-the-art deep learning algorithm. (4) We also enable model comparison by weights, removing the need for computationally expensive re-training or access to raw data. To promote collaboration and adoption, `genomicBERT` is available as part of the publicly accessible conda package called `genomeNLP`. Successful case studies have demonstrated the effectiveness of `genomeNLP` in `genome NLP` applications.



2.3 2. Connect to a remote server

To standardise the compute environment for all participants, we will be establishing a network connection with a remote server. Data and a working install of `genomenlp` is provided. Secure Shell (SSH) is a common method for remote server connection, providing secure access and remote command execution through encrypted connections between the client and server.

To use `ssh` (Secure Shell) for remote server access, please follow these steps:

1. Open a Terminal or Command Prompt on your local machine. SSH is typically available on Unix-like systems (e.g. Linux, macOS) and can also be installed on Windows systems using tools like PuTTY or MobaXterm.
2. Determine the `ssh` command syntax. Generally the format is: `ssh username@hostname` or the IP address of the remote server.
3. Enter your password or passphrase when prompted. Once authenticated, you should be connected to the remote server via SSH.

Note: *Details for (2) and (3) will be provided on the day of the workshop.*

2.4 3. Installing conda, mamba and genomenlp

Note: *This step is already performed for you. Information is provided as a guide for those who are reading this document outside of the tutorial, or if for some reason the installation is not working.*

A package/environment manager is a software tool that automates the installation, update, and removal of packages and allows for the creation of isolated environments with specific configurations. This simplifies software setup, reduces compatibility issues, and improves software development workflows. Popular examples include `apt` and `anaconda`. We will use `conda` and `mamba` in this case study.

Note: *The same is true for protein sequences, and nucleic acid data such as transcripts. While our pipeline can process any of these, the scope of this tutorial is for genomic data only.*

To install `conda` using the command line, you can follow these steps:

1. Open your command prompt. Use the `curl` or `wget` command to download the installer directly from the command line using its URL.

```
$ wget 'https://repo.anaconda.com/miniconda/Miniconda3-py39_23.3.1-0-Linux-x86_64.sh'
```

2. Run the installer script using the following command:

```
$ bash Miniconda3-py39_23.3.1-0-Linux-x86_64.sh
```

3. Follow the on-screen prompts to proceed with the installation. (In the prompt asking for the location for `conda` installation, please specify the directory as `foo/bar`)
4. Reload your `shell` as shown below OR exit and return to complete the install.

```
$ source ~/.bashrc
$ source ~/.bash_profile
```

5. To install `mamba`, which is a faster alternative to Conda for package management, run the following command:

```
$ conda install mamba -n base -c conda-forge
```

Note: *`pip` does not work due to a missing pytorch dependency. `conda` was found to be very slow due to the large dependency tree.*

6. As with Step 4, reload your shell as below OR exit and return to complete the install.

```
$ source ~/.bashrc
$ source ~/.bash_profile
```

7. To install and activate `genomenlp`, run the following commands:

```
$ mamba create -n genomenlp -c tyronechen -c conda-forge genomenlp -y
$ mamba activate genomenlp
# after the above completes
$ sweep -h
# you should see some output
```

2.5 Case studies per molecule type

Please select the case study relevant to your use case:

2.5.1 DNA case study

genomeNLP: Case study of DNA

2.5.2 RNA case study

coming soon

2.5.3 Protein case study

genomeNLP: Case study of Protein

2.6 Citation

Cite our manuscript here:

```
@article{chen2023genomicbert,
    title={genomicBERT and data-free deep-learning model evaluation},
    author={Chen, Tyrone and Tyagi, Navya and Chauhan, Sarthak and Peleg, Anton Y and␣
↪Tyagi, Sonika},
    journal={bioRxiv},
    month={jun},
    pages={2023--05},
    year={2023},
    publisher={Cold Spring Harbor Laboratory},
    doi={10.1101/2023.05.31.542682},
    url={https://doi.org/10.1101/2023.05.31.542682}
}
```

Cite our software here:

```
@software{tyrone_chen_2023_8135591,
  author       = {Tyrone Chen and
                  Navya Tyagi and
                  Sarthak Chauhan and
                  Anton Y. Peleg and
                  Sonika Tyagi},
  title        = {{genomicBERT and data-free deep-learning model
                  evaluation}},
  month        = jul,
  year         = 2023,
  publisher    = {Zenodo},
  version      = {latest},
  doi          = {10.5281/zenodo.8135590},
```

(continues on next page)

(continued from previous page)

```
  url          = {https://doi.org/10.5281/zenodo.8135590}
}
```

CHAPTER

THREE

GENOMENLP: CASE STUDY OF DNA

3.1 4. Setting up a biological dataset

Understanding of the data and experimental design is a necessary first step to analysis. In our case study, we perform a simple two case classification, where the dataset consists of a corpora of biological sequence data belonging to two categories. Genomic sequence associated with promoters and non-promoter regions are available. In the context of biology, promoters are important modulators of gene expression, and most are relatively short as well as information rich. Motif prediction is an active, on-going area of research in biology, since many of these signals are weak and difficult to detect, as well as varying in frequency and distribution across different species. **Therefore, our aim is to classify sequences into promoter and non-promoter sequence categories**.

Note: A more detailed description of the data is available here.

Our data is available in the form of `fasta` files. `fasta` files are a common format for storing biological sequence data. They typically contain headers that provide information about the sequence, followed by the sequence itself. They can also store other nucleic acid data, as well as protein. The `fasta` format contains headers with a leading >. Lines without > contain biological sequence data and can be newline separated. In our simple example, the full set of characters are the DNA nucleotides adenine `A`, thymine `T`, cytosine `C` and guanine `G`. These are the building blocks of the genetic code.

The files can be downloaded here for non promoter sequences and promoter sequences.

```
# create the directory structure
cd ~
mkdir -p data src results
cd data
curl -L -O "https://raw.githubusercontent.com/khanhlee/bert-promoter/main/data/non_
↪promoter.fasta"
curl -L -O "https://raw.githubusercontent.com/khanhlee/bert-promoter/main/data/promoter.
↪fasta"
gzip non_promoter.fasta
gzip promoter.fasta
```

```
HEADER:   >PCK12019 FORWARD 639002 STRONG
SEQUENCE: TAGATGTCCTTGATTAACACCAAAAT
HEADER:   >ECK12066 REVERSE 3204175 STRONG
SEQUENCE: AAAGAAAATAATTAATTTTACAGCTG
```

Note: *In real world data, other characters are available which refer to multiple possible nucleotides, for example ``W`` indicates either an ``A`` or a ``T``. RNA includes the character ``U``, and proteins include additional letters of the alphabet.*

Tokenisation in genomics involves segmenting biological sequences into smaller units, called tokens (or k-mers in biology) for further processing. In the context of genomics, tokens can represent individual nucleotides, k-mers, codons, or other biologically meaningful segments. Just as in conventional NLP, tokenisation is required to facilitate most downstream operations.

Here, we provide gzipped fasta file(s) as input. While conventional biological tokenisation splits a sequence into arbitrary-length segments, empirical tokenisation derives the resulting tokens directly from the corpus, with vocabulary size as the only user-defined parameter. Data is then split into training, testing and/or validation partitions as desired by the user and automatically reformatted for input into the deep learning pipeline.

Note: *We provide the conventional k-merisation method as well as an option for users. In our pipeline specifically, the empirical tokenisation and data object creation is split into two steps, while k-merisation combines both in one operation. This is due to the empirical tokenisation process having to "learn" tokens from the data.*

```
# Empirical tokenisation pathway
cd ~/src
tokenise_bio \
  -i ../data/promoter.fasta.gz \
     ../data/non_promoter.fasta.gz \
  -t ../data/tokens.json
# -i INFILE_PATHS path to files with biological seqs split by line
# -t TOKENISER_PATH path to tokeniser.json file to save or load data
```

This generates a `json` file with tokens and their respective weights or IDs. You should see some output like this.

```
[00:00:00] Pre-processing sequences
[00:00:00] Suffix array seeds
[00:00:14] EM training
Sample input sequence: AACCGGTT
Sample tokenised: [156, 2304]
Token: : k-mer map: 156  : : AA
Token: : k-mer map: 2304 : : CCGGTT
```

3.2 5. Format a dataset for input into genomeNLP

In this section, we reformat the data to meet the requirements of our pipeline which takes specifically structured inputs. This intermediate data structure serves as the foundation for downstream analyses and facilitates seamless integration with the pipeline. Our pipeline contains a method that performs this automatically, generating a reformatted dataset with the desired structure.

Note: *The data format is identical to that used by the HuggingFace ``datasets`` and ``transformers`` libraries.*

```
# Empirical tokenisation pathway
create_dataset_bio \
```

(continues on next page)

(continued from previous page)

```
  ../data/promoter.fasta.gz \
  ../data/non_promoter.fasta.gz \
  ../data/tokens.json \
  -o ../data/
# -o OUTFILE_DIR write dataset to directory as
#   [ csv \| json \| parquet \| dir/ ] (DEFAULT:"hf_out/")
# default datasets split: train 90%, test 5% and validation set 5%
```

The output is a reformatted dataset containing the same information. Properties required for a typical machine learning pipeline are added, including labels, customisable data splits and token identifiers.

```
DATASET AFTER SPLIT:
DatasetDict ({
  train: Dataset ({
  features: ['idx', 'feature', 'labels', 'input_ids', 'token_type_ids', 'attention_
↪mask'],
  num_rows: 12175 })
  test: Dataset ({
  features: ['idx', 'feature', 'labels', 'input_ids', 'token_type_ids', 'attention_
↪mask'],
  num_rows: 677 })
  valid: Dataset ({
  features: ['idx', 'feature', 'labels', 'input_ids', 'token_type_ids', 'attention_
↪mask'],
  num_rows: 676 })
})
```

Note: *The column ``token_type_ids`` is not actually needed in this specific case study, but it is safely ignored in such cases.*

```
SAMPLE TOKEN MAPPING FOR FIRST 5 TOKENS IN SEQ:
TOKEN ID: 858  | TOKEN: TCA
TOKEN ID: 2579 | TOKEN: GCATCAC
TOKEN ID: 111  | TOKEN: TATT
TOKEN ID: 99   | TOKEN: CAGG
TOKEN ID: 777  | TOKEN: AGGCT
```

3.3 6. Preparing a hyperparameter sweep

In machine learning, achieving optimal model performance often requires finding the right combination of hyperparameters (assuming the input data is viable). Hyperparameters vary depending on the specific algorithm and framework being used, but commonly include learning rate, dropout rate, batch size, number of layers and optimiser choice. These parameters heavily influence the learning process and subsequent performance of the model.

For this reason, hyperparameter sweeps are normally carried out to systematically test combinations of hyperparameters, with the end goal of identifying the configuration that produces the best model performance. Usually, sweeps are carried out on a small partition of the data only to maximise efficiency of compute resources, but it is not uncommon to perform sweeps on entire datasets. Various strategies, such as grid search, random search, or bayesian optimisation, can be employed during a hyperparameter sweep to sample parameter values. Additional strategies such as early stopping can also be used.

To streamline the hyperparameter optimization process, we use the `wandb` (Weights & Biases) platform which has a user-friendly interface and powerful tools for tracking experiments and visualising results.

First, sign up for a wandb account at: https://wandb.ai/site and login by pasting your API key.

```
wandb login
wandb: Paste an API key from your profile, and hit enter and hit enter or press ctrl+c␣
↪to quit:
```

Now, we use the `sweep` tool to perform hyperparameter sweep. Search strategy, parameters and search space are passed in as a `json` file. An example is below. If no sweep configuration is provided, default configuration will apply.

```
{
    "name": "random",
    "method": "random",
    "metric": {
        "name": "eval/f1",
        "goal": "maximize"
        },
    "parameters": {
        "epochs": {
            "values": [1, 2, 3, 4, 5]
            },
        "dropout": {
          "values": [0.15, 0.2, 0.25, 0.3, 0.4]
        },
        "batch_size": {
            "values": [8, 16, 32, 64]
            },
        "learning_rate": {
            "distribution": "log_uniform_values",
            "min": 1e-5,
            "max": 1e-1
        },
        "weight_decay": {
            "values": [0.0, 0.1, 0.2, 0.3, 0.4, 0.5]
        },
        "decay": {
            "values": [1e-5, 1e-6, 1e-7]
        },
        "momentum": {
            "values": [0.8, 0.9, 0.95]
        }
    },
    "early_terminate": {
        "type": "hyperband",
        "s": 2,
        "eta": 3,
        "max_iter": 27
    }
}
```

```
sweep \
  ../data/train.parquet \
```

(continues on next page)

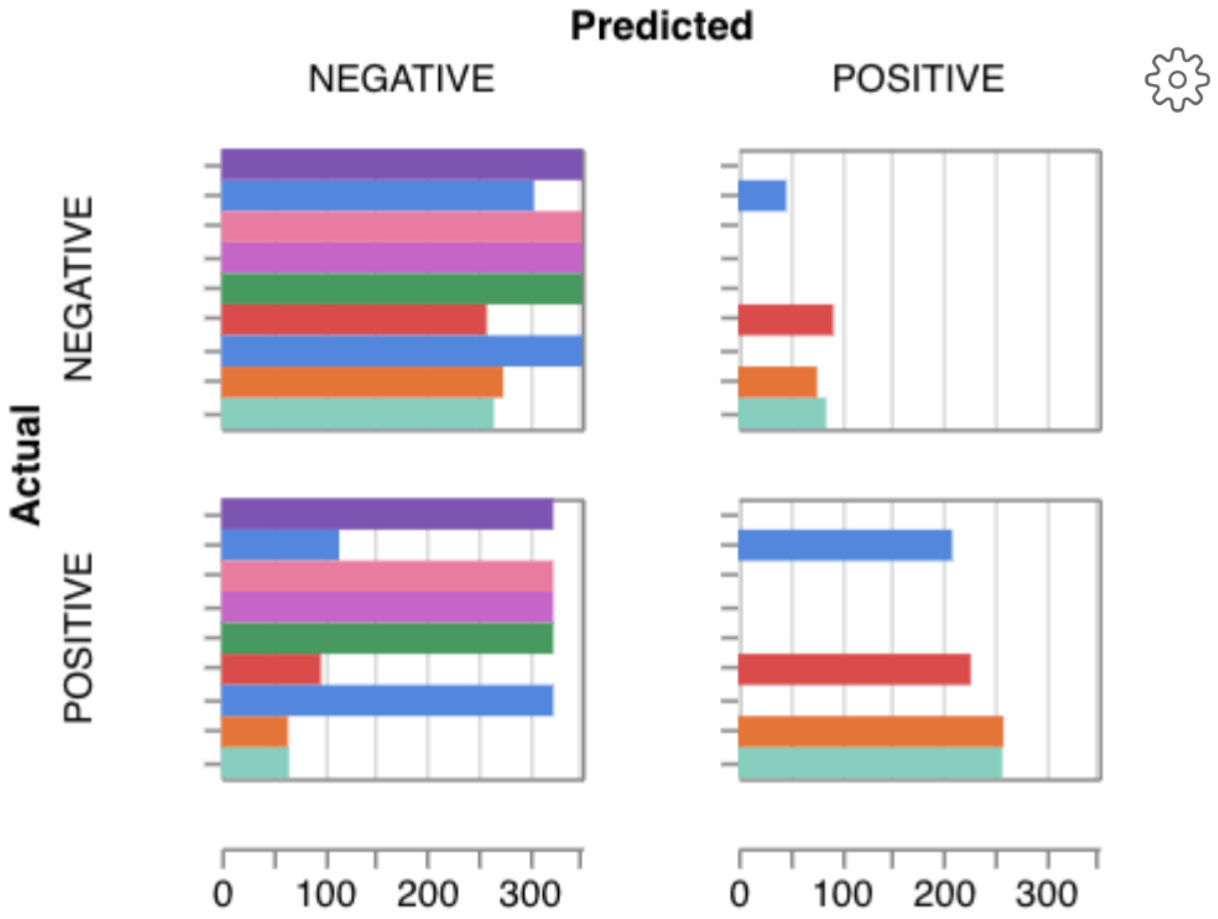
(continued from previous page)

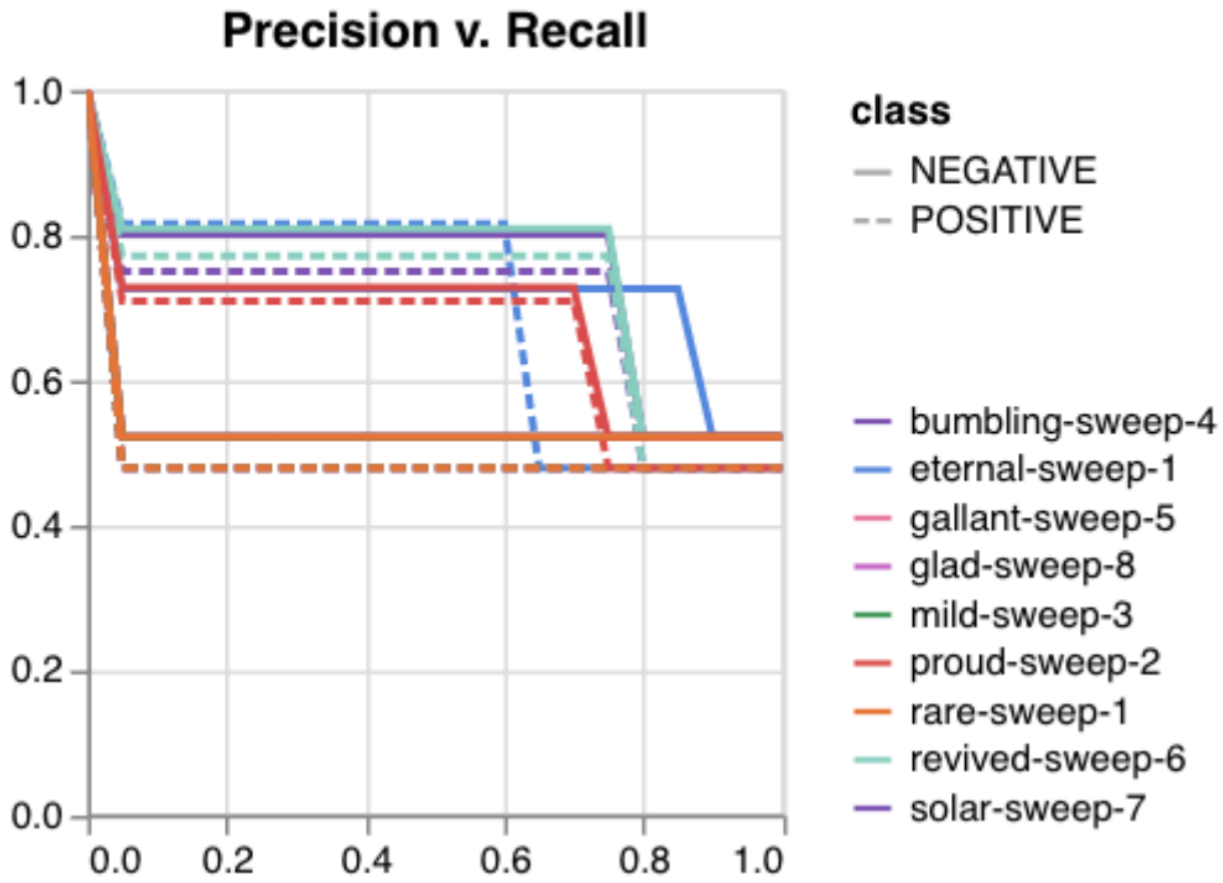
```
  parquet \
  ../data/tokens.json \
  -t ../data/test.parquet \
  -v ../data/valid.parquet \
  -w ../data/hyperparams.json \   # optional
  -e entity_name \       # <- edit as needed
  -p project_name \      # <- edit as needed
  -l labels \
  -n 3
# -t TEST, path to [ csv \| csv.gz \| json \| parquet ] file
# -v VALID, path to [ csv \| csv.gz \| json \| parquet ] file
# -w HYPERPARAMETER_SWEEP, run a hyperparameter sweep with config from file
# -e ENTITY_NAME, wandb team name (if available).
# -p PROJECT_NAME, wandb project name (if available)
# -l LABEL_NAMES, provide column with label names (DEFAULT: "").
# -n SWEEP_COUNT, run n hyperparameter sweeps
```

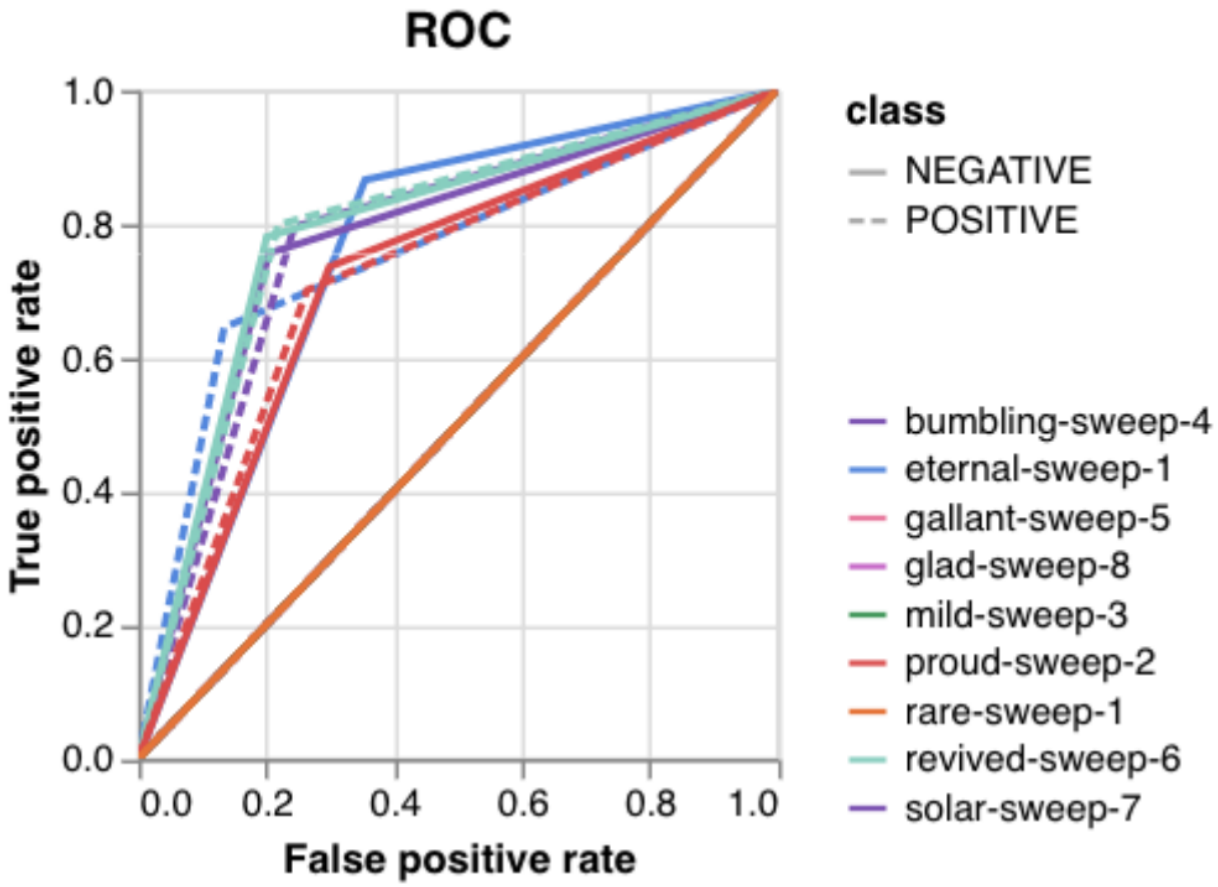
```
*****Running training*****
Num examples = 12175
Num epochs= 1
Instantaneous batch size per device = 64
Total train batch size per device = 64
Gradient Accumulation steps= 1
Total optimization steps= 191
```

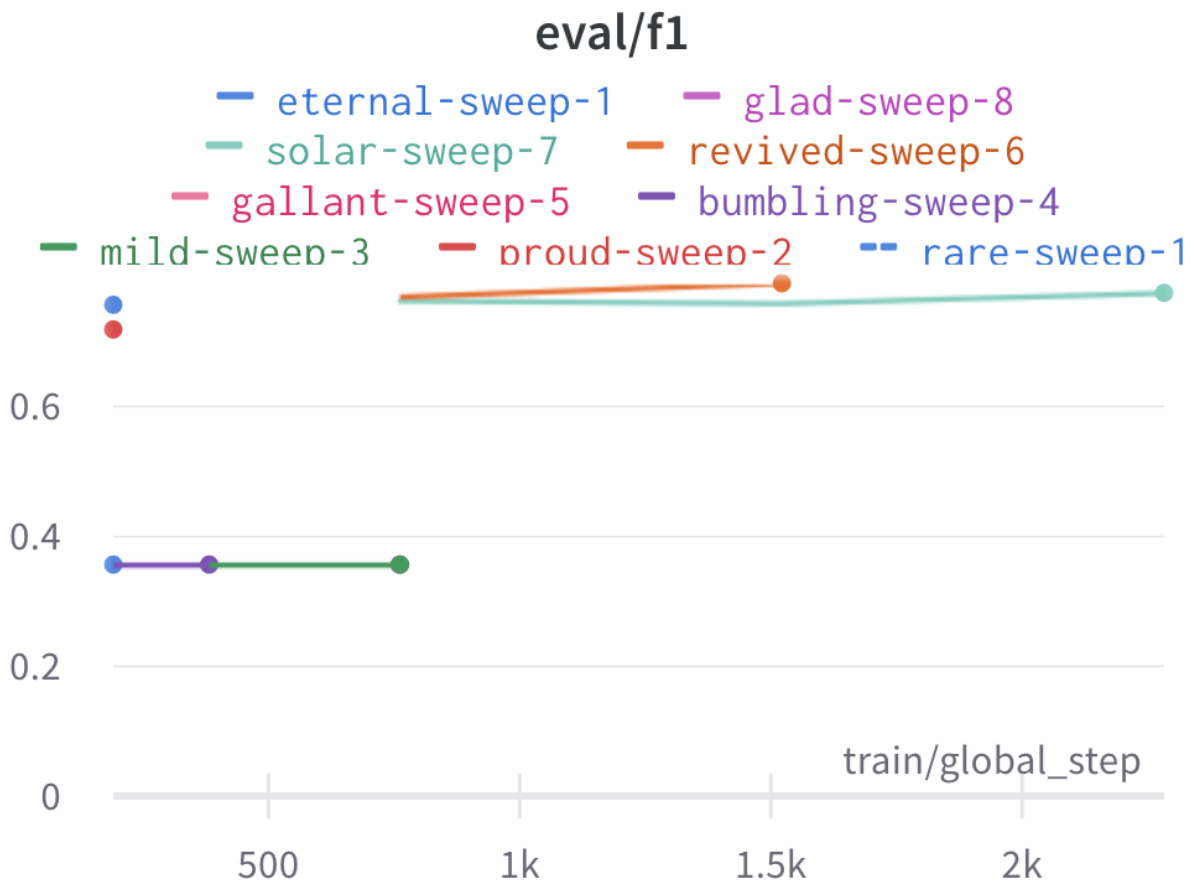
The output is written to the specified directory, in this case `sweep_out` and will contain the output of a standard `pytorch` saved model, including some `wandb` specific output.

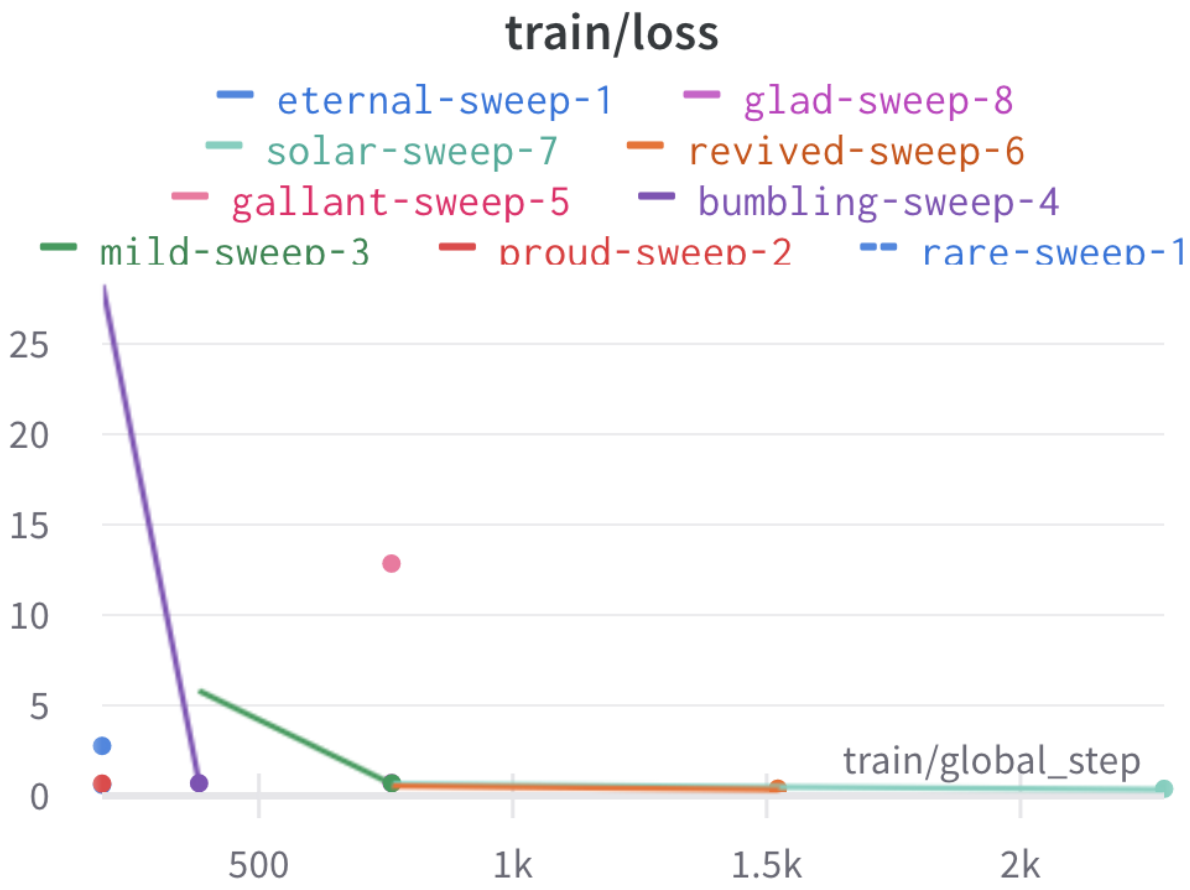
The sweeps gets synced to the `wandb` dashboard along with various interactive custom charts and tables which we provide as part of our pipeline. A small subset of plots are provided for reference. Interactive versions of these and more plots are available on wandb.

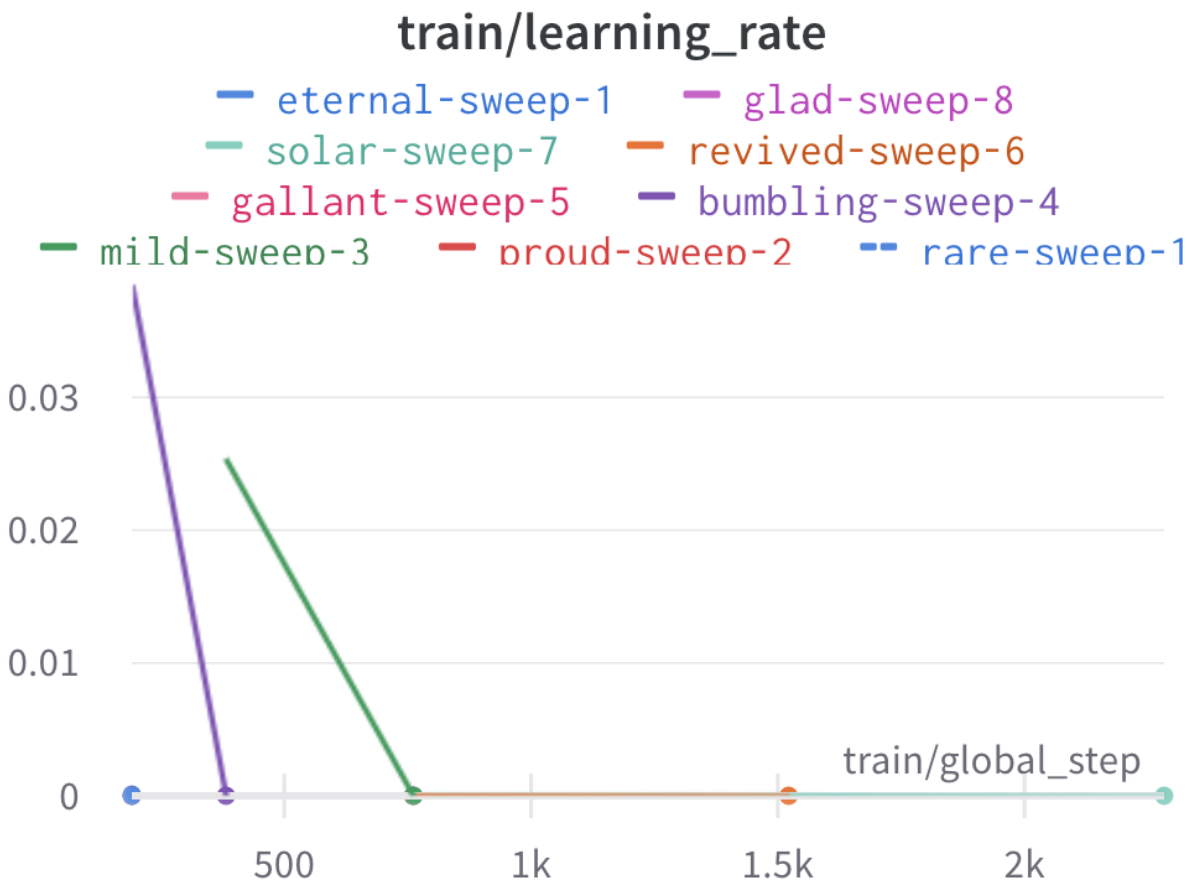

Predicted
NEGATIVE
POSITIVE
Actual
NEGATIVE
POSITIVE
0
100
200
300
0
100
200
300


Precision v. Recall
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
class
NEGATIVE
POSITIVE
bumbling-sweep-4
eternal-sweep-1
gallant-sweep-5
glad-sweep-8
mild-sweep-3
proud-sweep-2
rare-sweep-1
revived-sweep-6
solar-sweep-7


ROC
1.0
0.8
0.6
0.4
0.2
0.0
True positive rate
0.0
0.2
0.4
0.6
0.8
1.0
False positive rate
class
NEGATIVE
POSITIVE
bumbling-sweep-4
eternal-sweep-1
gallant-sweep-5
glad-sweep-8
mild-sweep-3
proud-sweep-2
rare-sweep-1
revived-sweep-6
solar-sweep-7


eval/f1
eternal-sweep-1
glad-sweep-8
solar-sweep-7
revived-sweep-6
gallant-sweep-5
bumbling-sweep-4
mild-sweep-3
proud-sweep-2
rare-sweep-1
0.6
0.4
0.2
0
train/global_step
500
1k
1.5k
2k


train/loss
eternal-sweep-1
glad-sweep-8
solar-sweep-7
revived-sweep-6
gallant-sweep-5
bumbling-sweep-4
mild-sweep-3
proud-sweep-2
rare-sweep-1
25
20
15
10
5
0
train/global_step
500
1k
1.5k
2k



Here is an example of a full wandb generated report:

You may inspect your own generated reports after they complete.

3.4 7. Selecting optimal hyperparameters for training

Having completed a sweep, we next identified the best set of parameters for model training. We do this by examining training metrics. These serve as quantitative measures of a model's performance during training. These metrics provide insights into the model's accuracy and generalisation capabilities. We explore commonly used training metrics, including accuracy, loss, precision, recall, and f1 score to inform us of a model's performance

A key event we want to avoid is overfitting. Overfitting occurs when a learning model performs exceptionally well on the training data but fails to generalise to unseen data, making it unfit for use outside of the specific scope of the experiment. This can be detected by observing performance metrics, if the accuracy decreases and later increases an overfit event has occurred. In real world applications, this can lead to adverse events that directly impact us, considering that such models are used in applications such as drug prediction or self-driving cars. Here, we use the f1 score calculated on the testing set as the main metric of interest. We showed that we obtain a best `f1` score of `0.79`.

```
Best run revived-sweep-6 with eval/f1=0.7900291349379833
BEST MODEL AND CONFIG FILES SAVED TO: *./sweep_out/model_files*
HYPERPARAMETER SWEEP END
```

Here is an example of a full wandb generated report for the "best" run.

You may inspect your own generated reports after they complete.

3.5 8. With the selected hyperparameters, train the full dataset

In a conventional workflow, the sweep is performed on a small subset of training data. The resulting parameters are then recorded and used in the actual training step on the full dataset. Here, we perform the sweep on the entire dataset, and hence remove the need for further training. If you perform this on your own data and want to use a small subset, you can do so and then pass the recorded hyperparameters with the same input data to the `train` function of the pipeline. We include an example of this below for completeness, but you can skip this for our specific case study. Note that the input is almost identical to `sweep`.

```
train \
  ../data/train.parquet \
  parquet \
  ../data/tokens.json \
  -t ../data/test.parquet \
  -v ../data/valid.parquet \
  --output_dir ../results/train_out \
  -f ../data/hyperparams.json \  # <- you can pass in hyperparameters
  -c entity_name/project_name/run_id \  # <- wandb overrides hyperparameters
  -e entity_name \   # <- edit as needed
  -p project_name    # <- edit as needed
# -t TEST, path to [ csv \| csv.gz \| json \| parquet ] file
# -v VALID, path to [ csv \| csv.gz \| json \| parquet ] file
# -w HYPERPARAMETER_SWEEP, run a hyperparameter sweep with config from file
# -e ENTITY_NAME, wandb team name (if available).
# -p PROJECT_NAME, wandb project name (if available)
# -l LABEL_NAMES, provide column with label names (DEFAULT: "").
```

Note: *Remove the ``-e entity_name`` line if you do not have a group setup in wandb*

```
{
  "output_dir": "./sweep_out/random",
  "overwrite_output_dir": false,
  "do_train": false,
  "do_eval": true,
  "do_predict": false,
  "evaluation_strategy": "epoch",
  "prediction_loss_only": false,
  "per_device_train_batch_size": 16,
  "per_device_eval_batch_size": 16,
  "per_gpu_train_batch_size": null,
  "per_gpu_eval_batch_size": null,
  "gradient_accumulation_steps": 1,
  "eval_accumulation_steps": null,
  "eval_delay": 0,
  "learning_rate": 7.796477400405317e-05,
  "weight_decay": 0.5,
```

(continues on next page)

(continued from previous page)

```
  "adam_beta1": 0.9,
  "adam_beta2": 0.999,
  "adam_epsilon": 1e-08,
  "max_grad_norm": 1.0,
  "num_train_epochs": 2,
  "max_steps": -1,
  "lr_scheduler_type": "linear",
  "warmup_ratio": 0.0,
  "warmup_steps": 0,
  "log_level": "passive",
  "log_level_replica": "passive",
  "log_on_each_node": true,
  "logging_dir": "./sweep_out/random/runs/out",
  "logging_strategy": "epoch",
  "logging_first_step": false,
  "logging_steps": 500,
  "logging_nan_inf_filter": true,
  "save_strategy": "epoch",
  "save_steps": 500,
  "save_total_limit": null,
  "save_on_each_node": false,
  "no_cuda": false,
  "use_mps_device": false,
  "seed": 42,
  "data_seed": null,
  "jit_mode_eval": false,
  "use_ipex": false,
  "bf16": false,
  "fp16": false,
  "fp16_opt_level": "O1",
  "half_precision_backend": "auto",
  "bf16_full_eval": false,
  "fp16_full_eval": false,
  "tf32": null,
  "local_rank": -1,
  "xpu_backend": null,
  "tpu_num_cores": null,
  "tpu_metrics_debug": false,
  "debug": [],
  "dataloader_drop_last": false,
  "eval_steps": null,
  "dataloader_num_workers": 0,
  "past_index": -1,
  "run_name": "./sweep_out/random",
  "disable_tqdm": false,
  "remove_unused_columns": false,
  "label_names": null,
  "load_best_model_at_end": true,
  "metric_for_best_model": "loss",
  "greater_is_better": false,
  "ignore_data_skip": false,
  "sharded_ddp": [],
```

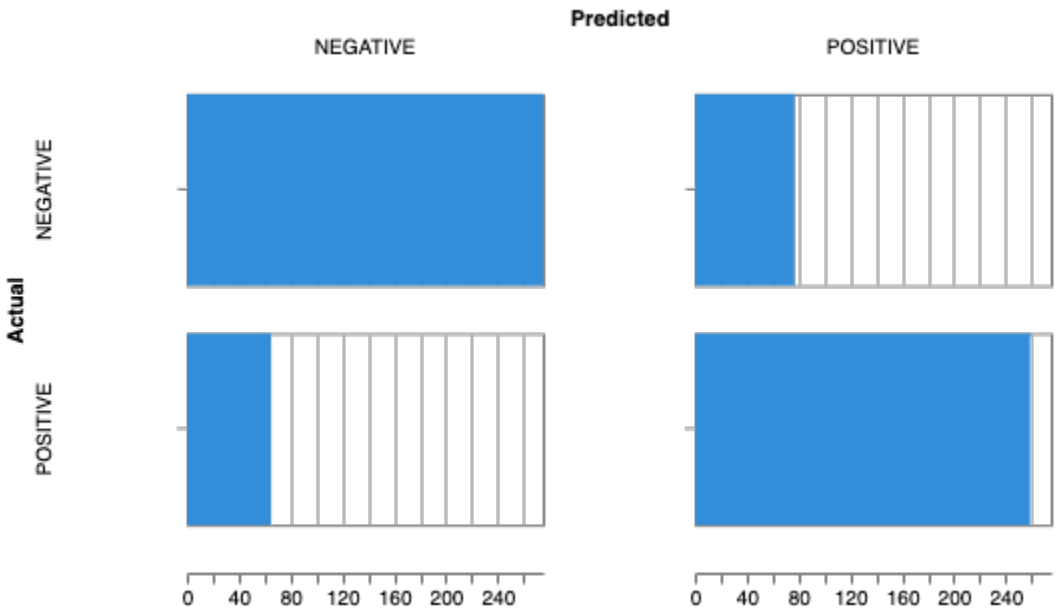
(continues on next page)

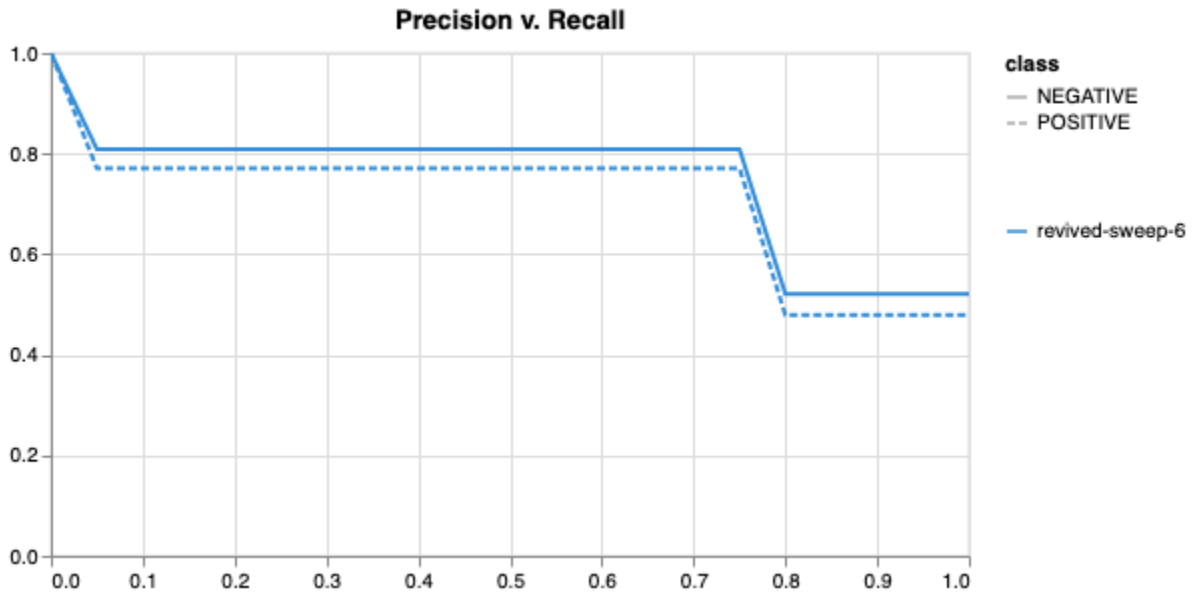
(continued from previous page)

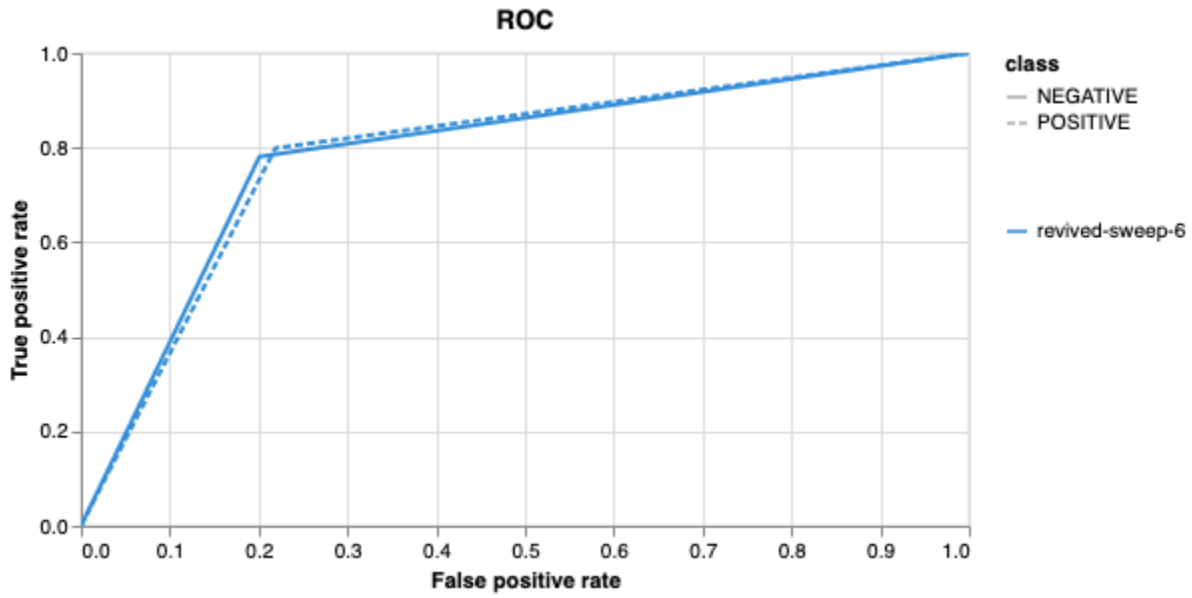
```
  "fsdp": [],
  "fsdp_min_num_params": 0,
  "fsdp_transformer_layer_cls_to_wrap": null,
  "deepspeed": null,
  "label_smoothing_factor": 0.0,
  "optim": "adamw_hf",
  "adafactor": false,
  "group_by_length": false,
  "length_column_name": "length",
  "report_to": [
    "wandb"
  ],
  "ddp_find_unused_parameters": null,
  "ddp_bucket_cap_mb": null,
  "dataloader_pin_memory": true,
  "skip_memory_metrics": true,
  "use_legacy_prediction_loop": false,
  "push_to_hub": false,
  "resume_from_checkpoint": null,
  "hub_model_id": null,
  "hub_strategy": "every_save",
  "hub_token": "<HUB_TOKEN>",
  "hub_private_repo": false,
  "gradient_checkpointing": false,
  "include_inputs_for_metrics": false,
  "fp16_backend": "auto",
  "push_to_hub_model_id": null,
  "push_to_hub_organization": null,
  "push_to_hub_token": "<PUSH_TO_HUB_TOKEN>",
  "mp_parameters": "",
  "auto_find_batch_size": false,
  "full_determinism": false,
  "torchdynamo": null,
  "ray_scope": "last",
  "ddp_timeout": 1800
}
```

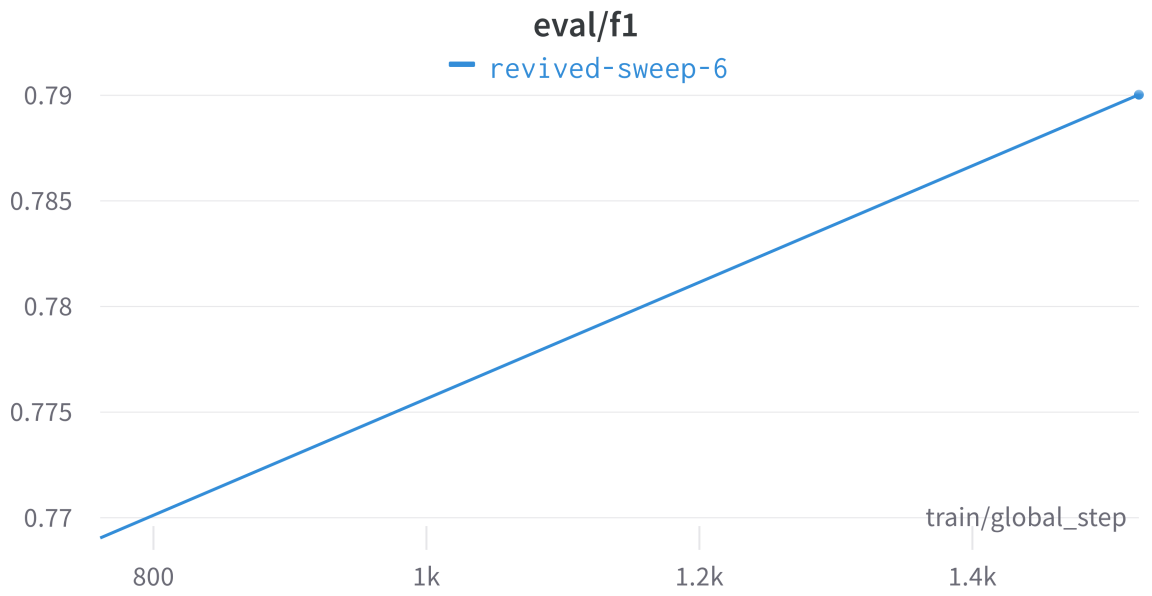
The output is written to the specified directory, in this case `train_out` and will contain the output of a standard `pytorch` saved model, including some `wandb` specific output.

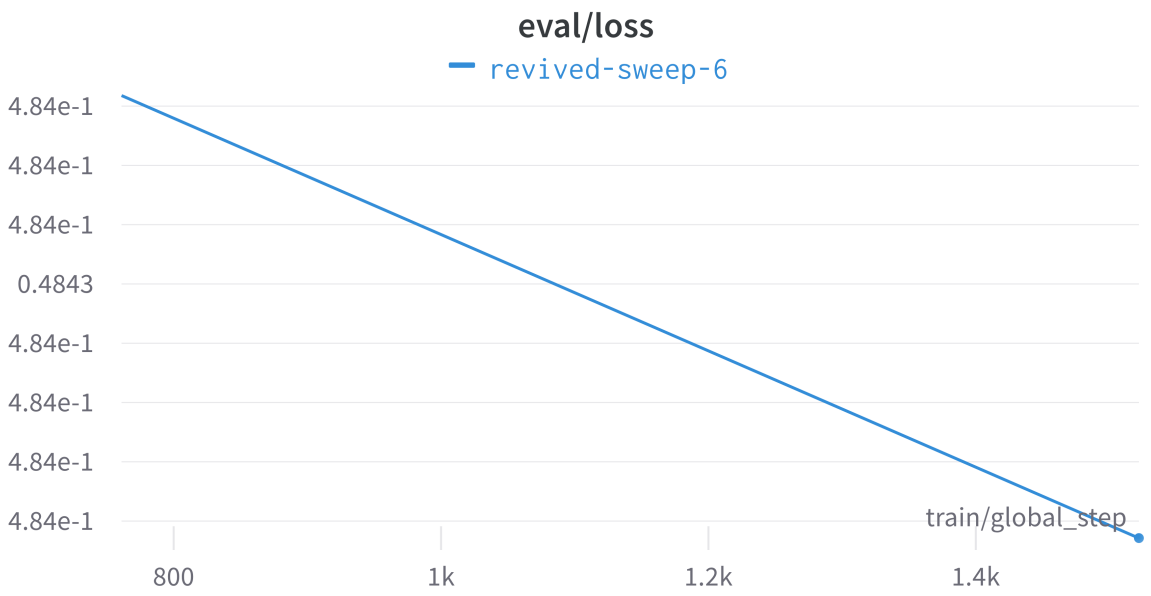
The trained model gets synced to the `wandb` dashboard along with various interactive custom charts and tables which we provide as part of our pipeline. A small subset of plots are provided for reference. Interactive versions of these and more plots are available on wandb.

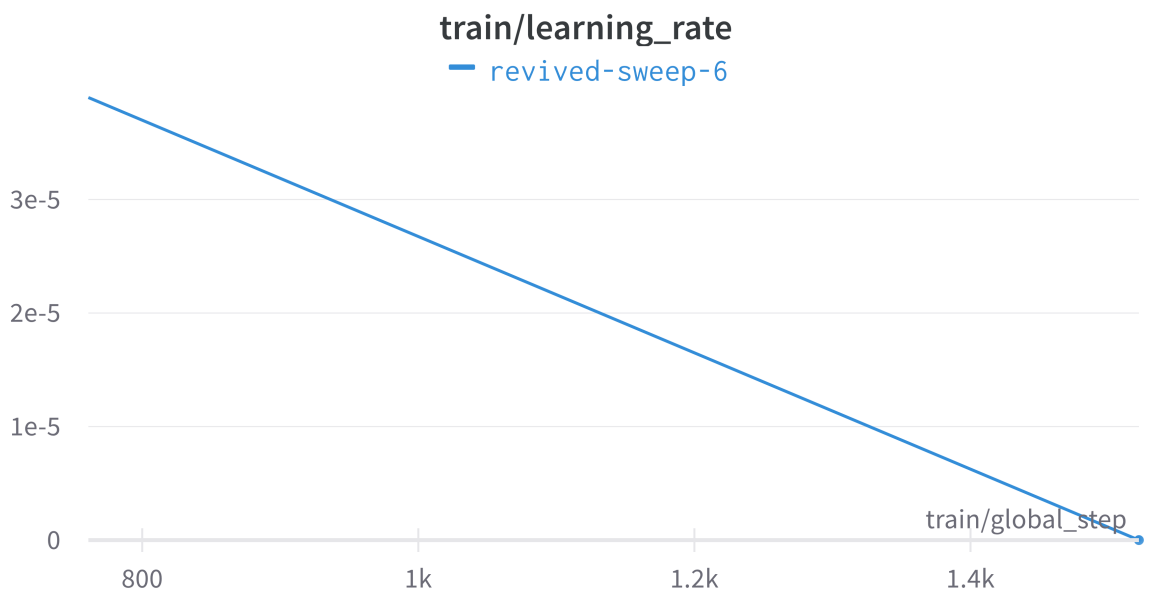

Predicted
NEGATIVE
POSITIVE
Actual
NEGATIVE
POSITIVE
0
40
80
120
160
200
240
0
40
80
120
160
200
240


Precision v. Recall
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0
class
NEGATIVE
POSITIVE
revived-sweep-6


ROC
True positive rate
False positive rate
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0
class
NEGATIVE
POSITIVE
revived-sweep-6


eval/f1
revived-sweep-6
0.79
0.785
0.78
0.775
0.77
train/global_step
800
1k
1.2k
1.4k





Here is an example of a full wandb generated report:

You may inspect your own generated reports after they complete.

3.6 9. Perform cross-validation

Having identified the best set of parameters and trained the model, we next want to conduct a comprehensive review of data stability, and we do this by evaluating model performance across different data slices. This assessment is known as cross-validation. We make use of k-fold cross-validation in which data is divided into k subsets and the model is trained and tested on these individual subsets.

```
cross_validate \
  ../data/train.parquet parquet \
  -t ../data/test.parquet \
  -v ../data/valid.parquet \
  -e entity_name \              # <- edit as needed
  -p project_name \             # <- edit as needed
  --config_from_run p9do3gzl \  # id OR directory of best performing run
  --output_dir ../results/cv \
  -m ../results/sweep_out \     # <- overridden by --config_from_run
  -l labels \
  -k 8
# --config_from_run WANDB_RUN_ID, *best run id*
# --output_dir OUTPUT_DIR
# -l label_names
# -k KFOLDS, run n number of kfolds

cross_validate \
  ../data/train.parquet parquet \
  -t ../data/test.parquet \
  -v ../data/valid.parquet \
  -e tyagilab \
  -p foobar \
  -c tyagilab/foobar/kixu82co \
  -o ../results/cv \
  -m ../results/sweep_out \
  -l labels \
  -k 8
```

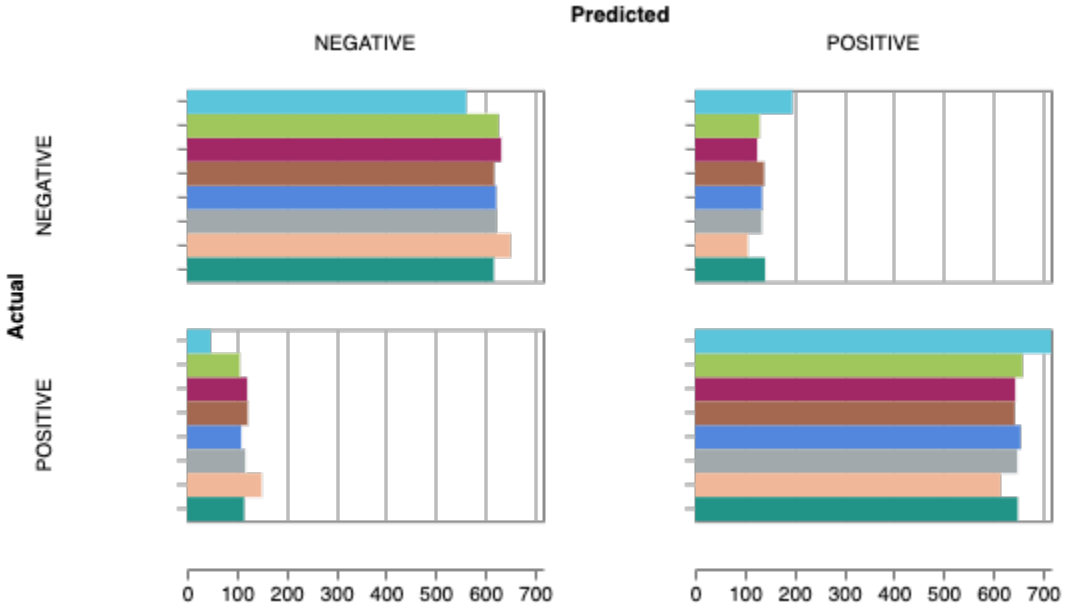
Note: *If both ``model_path`` and ``config_from_run`` are specified, ``config_from_run`` overrides*

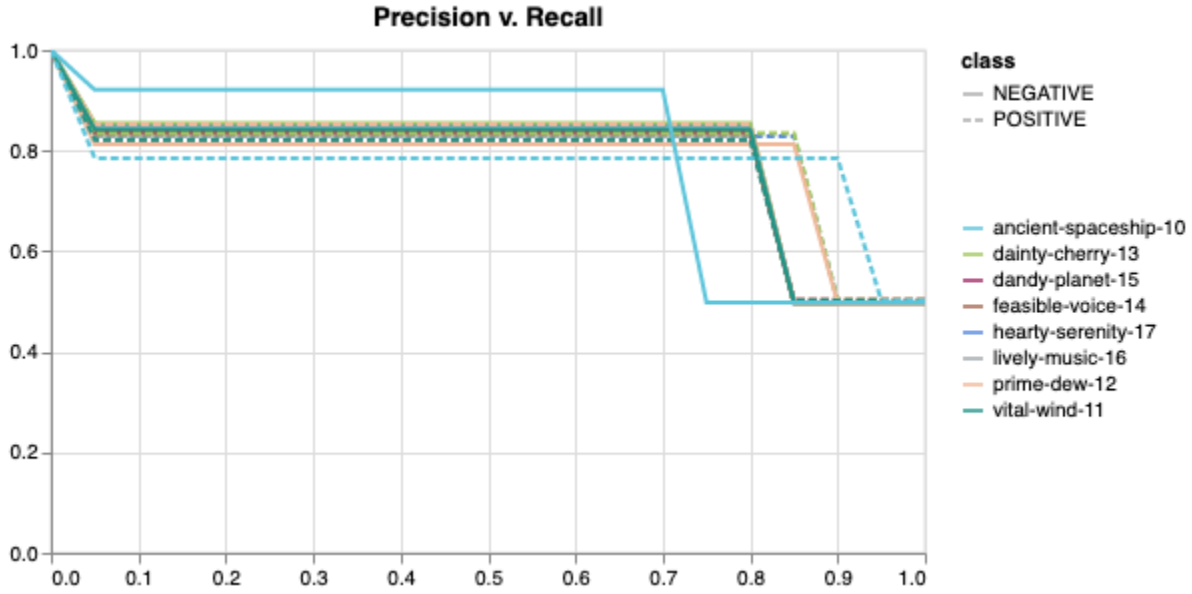
Note: *Remove the ``-e entity_name`` line if you do not have a group setup in wandb*

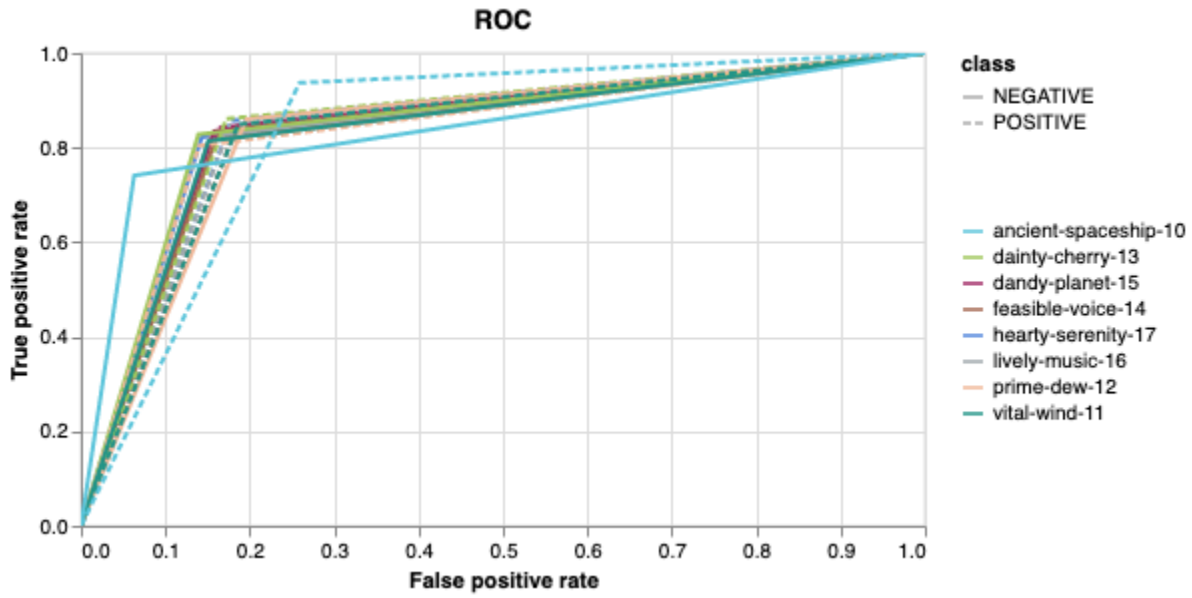
```
*****Running training*****
Num examples = 10653
Num epochs= 2
Instantaneous batch size per device = 16
Total train batch size (w, parallel, distributed & accumulation)= 16
Gradient Accumulation steps= 1
Total optimization steps= 1332
Automatic Weights & Biases logging enabled
```

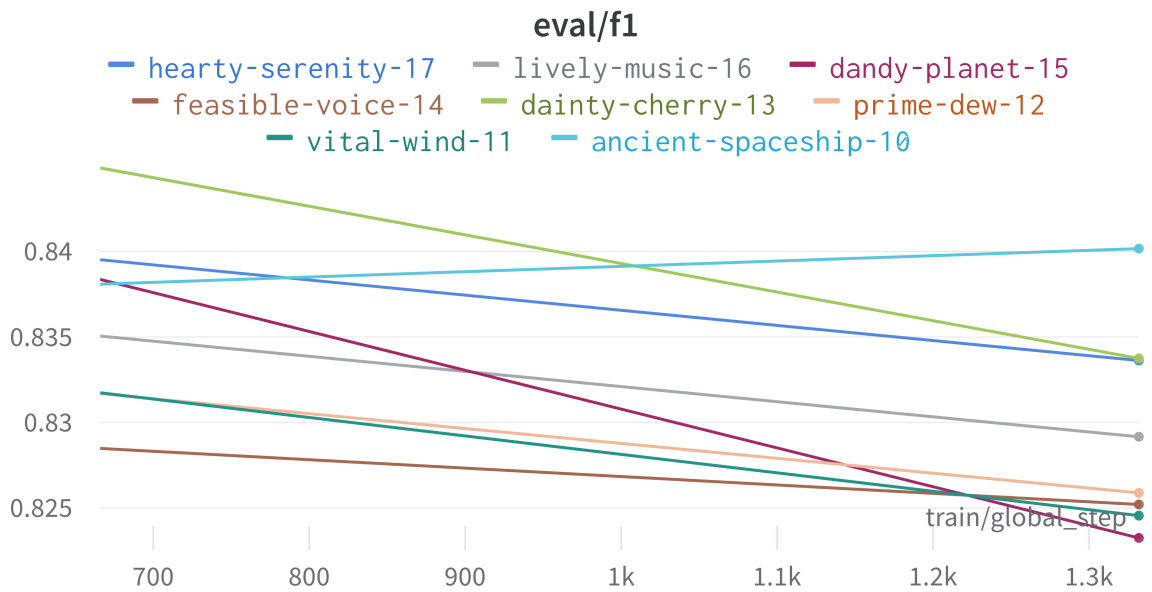
The cross-validation runs are uploaded to the `wandb` dashboard along with various interactive custom charts and tables which we provide as part of our pipeline. These are conceptually identical to those generated by `sweep` or `train`. A

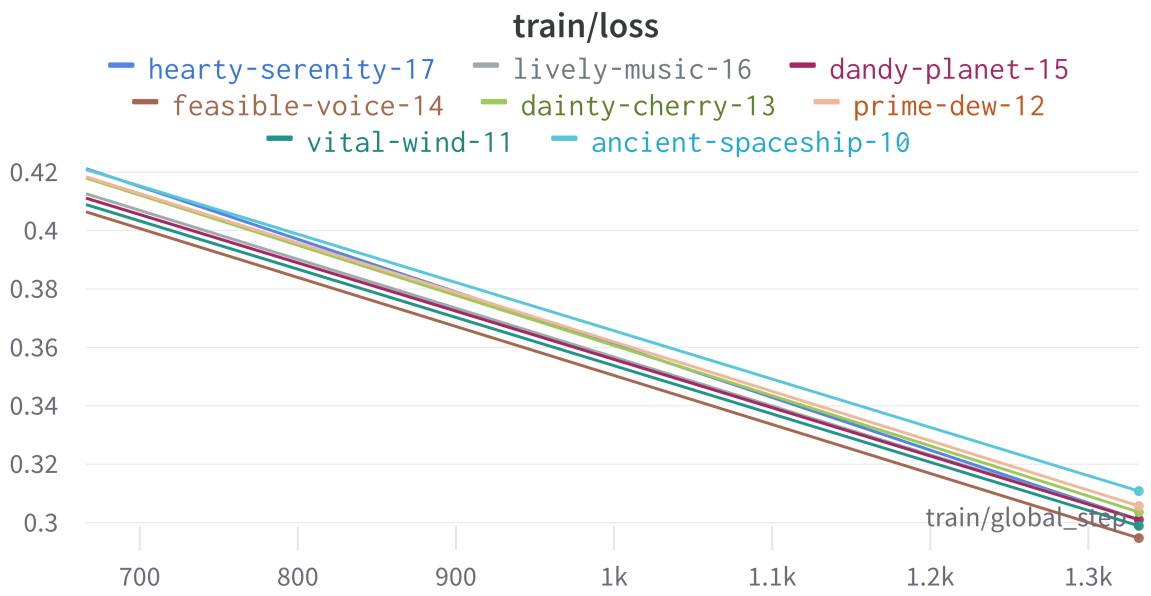
small subset of plots are provided for reference. Interactive versions of these and more plots are available on wandb.

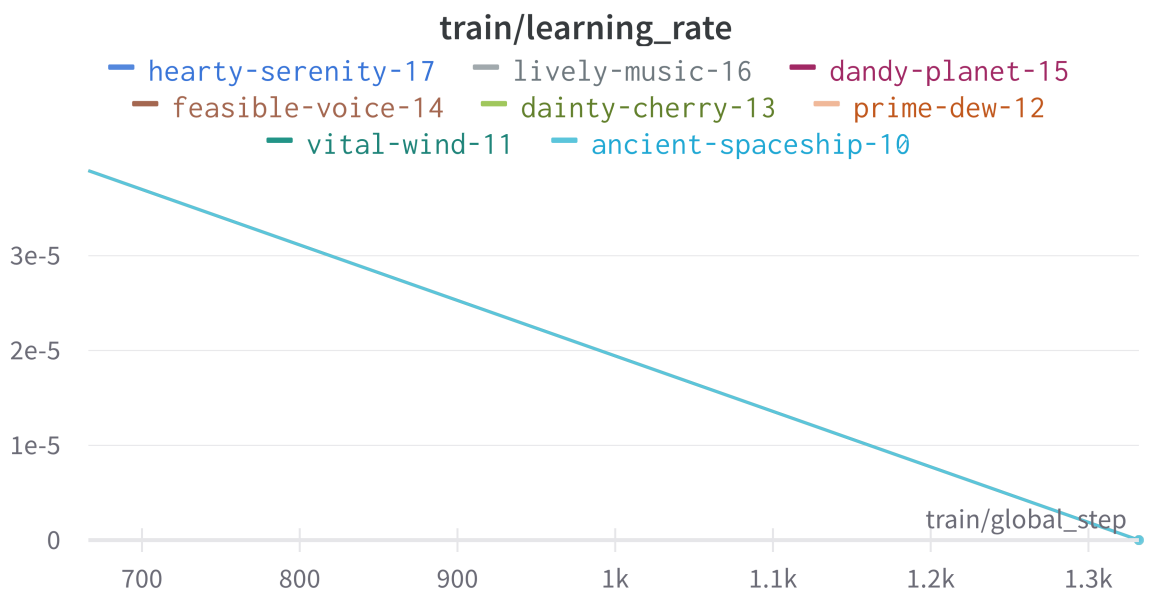





ROC
True positive rate
False positive rate
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0
class
NEGATIVE
POSITIVE
ancient-spaceship-10
dainty-cherry-13
dandy-planet-15
feasible-voice-14
hearty-serenity-17
lively-music-16
prime-dew-12
vital-wind-11


eval/f1
hearty-serenity-17
lively-music-16
dandy-planet-15
feasible-voice-14
dainty-cherry-13
prime-dew-12
vital-wind-11
ancient-spaceship-10
0.84
0.835
0.83
0.825
train/global_step
700
800
900
1k
1.1k
1.2k
1.3k





Here is an example of a full wandb generated report:

You may inspect your own generated reports after they complete.

3.7 10. Compare different models

The aim of this step is to compare performance of different deep learning models efficiently while avoiding computationally expensive re-training and data download in conventional model comparison. In the case of patient data, they are often inaccessible for privacy reasons, and in other cases they are not uploaded by the authors of the experiment.

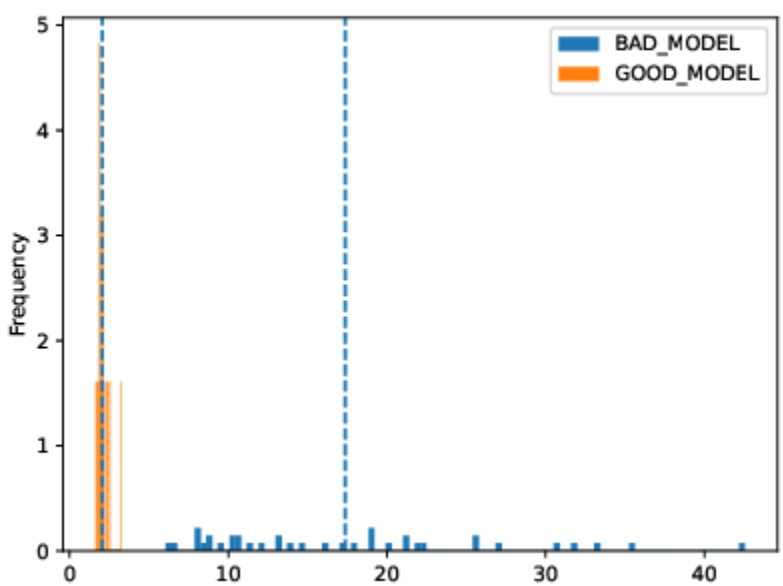
For the purposes of this simple case study, we compare multiple sweeps of the same dataset as a demonstration. In a real life application, existing biological models can be compared against the user-generated one.

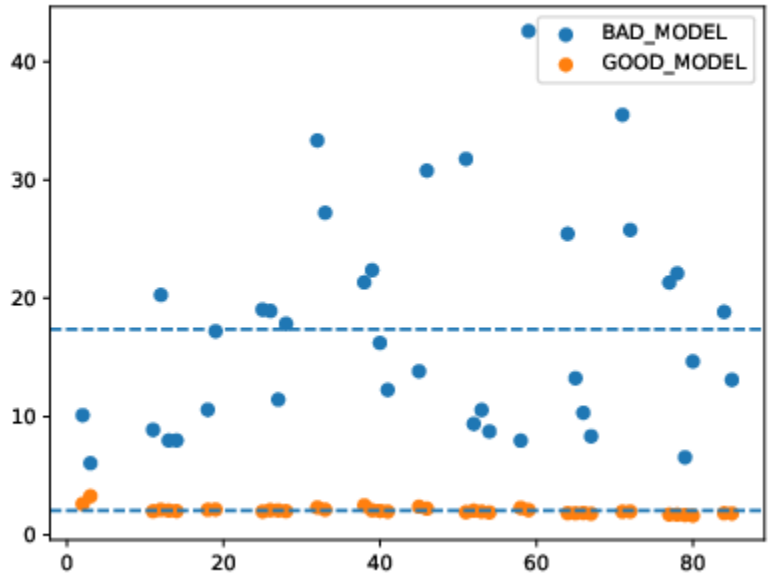
```
fit_powerlaw \
  ../results/sweep_out/model_files \
  -o ../results/fit
# -m MODEL_PATH, path to trained model directory
# -o OUTPUT_DIR, path to output metrics directory
```

This tool outputs a variety of plots in the specified directory.

```
ls ../results/fit
# alpha_hist.pdf  alpha_plot.pdf  model_files/
```

Very broadly, the overlaid bar plots allow the user to compare the performance of different models on the same scale. A narrow band around 2-5 with few outliers is in general cases an indicator of good model performance. This is a general guideline and will differ depending on context! For a detailed explanation of these plots, please refer to the original publication.





3.8 11. Obtain model interpretability scores

Model interpretability is often used for debugging purposes, by allowing the user to "see" (to an extent) what a model is focusing on. In this case, the tokens which contribute to a certain classification are highlighted. The green colour indicates a classification towards the target category, while the red colour indicates a classification away from the target category. Colour intensity indicates the classification score.

In some scenarios, we can exploit this property by identifying regulatory regions or motifs in DNA sequences, or discovering amino acid residues in protein structure critical to its function, leading to a deeper understanding of the underlying biological system.

```
gzip -cd ../data/promoter.fasta.gz | \
  head -n10 > ../data/subset.fasta
interpret \
  ../results/sweep_out/model_files \
  ../data/subset.fasta \
  -l PROMOTER NON-PROMOTER \
  -o ../results/model_interpret
# -t TOKENISER_PATH, path to tokeniser.json file to load data
# -o OUTPUT_DIR, specify path for output
```

```
ECK120010480 CSGDP1 REVERSE 1103344 SIGMA38.html
ECK120010489 OSMCP2 FORWARD 1556606 SIGMA38.html
ECK120010491 TOPAP1 FORWARD 1330980 SIGMA32 STRONG.html
ECK120010496 YJAZP  FORWARD 4189753 SIGMA32 STRONG.html
ECK120010498 YADVP2 REVERSE 156224  SIGMA38.html
```

Legend: Negative Neutral Positive

True Label	Predicted Label	Attribution Label	Attribution Score	Word Importance
0	PROMOTER (0.96)	PROMOTER	0.77	AAAGA AAATAATT AATTTTA CAGCT GTTAA ACC AAACGGT TAT AACC TG GTCATACGC AGTA GTT CGGAC AA GCGGTA CAT

Legend: Negative Neutral Positive

True Label	Predicted Label	Attribution Label	Attribution Score	Word Importance
0	PROMOTER (0.64)	PROMOTER	-0.57	TA GATGTCC TTGATTAA CACCAA AATTAAACC TTTT AAAAAC CAGG CATTC AAAAAC GGC GAATTCATCGAA ATCACC GAA

3.9 Citation

Cite our manuscript here:

```
@article{chen2023genomicbert,
    title={genomicBERT and data-free deep-learning model evaluation},
    author={Chen, Tyrone and Tyagi, Navya and Chauhan, Sarthak and Peleg, Anton Y and
↪Tyagi, Sonika},
    journal={bioRxiv},
    month={jun},
    pages={2023--05},
    year={2023},
    publisher={Cold Spring Harbor Laboratory},
```

(continues on next page)

(continued from previous page)

```
    doi={10.1101/2023.05.31.542682},
    url={https://doi.org/10.1101/2023.05.31.542682}
}
```

Cite our software here:

```
@software{tyrone_chen_2023_8135591,
  author       = {Tyrone Chen and
                  Navya Tyagi and
                  Sarthak Chauhan and
                  Anton Y. Peleg and
                  Sonika Tyagi},
  title        = {{genomicBERT and data-free deep-learning model
                  evaluation}},
  month        = jul,
  year         = 2023,
  publisher    = {Zenodo},
  version      = {latest},
  doi          = {10.5281/zenodo.8135590},
  url          = {https://doi.org/10.5281/zenodo.8135590}
}
```

CHAPTER

FOUR

GENOMENLP: CASE STUDY OF PROTEIN

4.1 4. Setting up a biological dataset

Understanding of the data and experimental design is a necessary first step to analysis. In our case study, we perform a simple two case classification, where the dataset consists of a corpora of biological sequence data belonging to two categories. Protein sequence associated with DNA binding proteins (DBP) and RNA binding proteins (RBP) are available. In the context of biology, DBP interact with DNA molecules, playing roles in gene regulation, DNA replication, repair, and structural organization while RBP interact with various RNA types and function in processes like splicing, RNA stability, translation regulation, and ribosome function. Aberrations in DBP and RBP are implicated in various diseases, including cancer and neurodegenerative disorders. Identifying and classifying these proteins helps in studying disease mechanisms and developing potential therapeutic strategies. **Therefore, our aim is to classify sequences into DNA binding protein and RNA binding protein categories.**

Our data is available in the form of `fasta` files. `fasta` files are a common format for storing biological sequence data. They typically contain headers that provide information about the sequence, followed by the sequence itself. They can also store other nucleic acid data, as well as protein. The `fasta` format contains headers with a leading >. Lines without > contain biological sequence data and can be newline separated. In this example, the full set of characters are the 20 naturally occurring amino acids Alanine `A`, Cysteine `C`, Aspartic Acid `D`, Glutamic Acid `E`, Phenylalanine `F`, Glycine `G`, Histidine `H`, Isoleucine `I`, Lysine `K`, Leucine `L`, Methionine `M`, Asparagine `N`, Proline `P`, Glutamine `Q`, Arginine `R`, Serine `S`, Threonine `T`, Valine `V`, Tryptophan `W` and Tyrosine `Y`. These are the building blocks of proteins.

```
#!/bin/bash
# download and preprocess protein data for RNA and DNA binding proteins
# original article: https://doi.org/10.1016/j.jmb.2020.09.008
wget 'http://bliulab.net/iDRBP_MMC/static/dataset/training_dataset.txt'
wget 'http://bliulab.net/iDRBP_MMC/static/dataset/test_dataset_TEST474.txt'
wget 'http://bliulab.net/iDRBP_MMC/static/dataset/test_dataset_PDB255.txt'

csplit --digits=2 --quiet --prefix=outfile training_dataset.txt "/-----------------------
↪-----------------------------------/+1" "{*}"
sed '$d' outfile02 | sed '$d' > train_dna_binding.fa
sed '$d' outfile04 | sed '$d' > train_rna_binding.fa
rm outfile0*

csplit --digits=2 --quiet --prefix=outfile test_dataset_TEST474.txt "/-------------------
↪----------------------------------------/+1" "{*}"
sed '$d' outfile02 | sed '$d' > test_TEST474_dna_binding.fa
sed '$d' outfile04 | sed '$d' > test_TEST474_rna_binding.fa
rm outfile0*

csplit --digits=2 --quiet --prefix=outfile test_dataset_PDB255.txt "/--------------------
```

(continues on next page)

(continued from previous page)

```
↪----------------------------------------/+1" "{*}"
sed '$d' outfile02 | sed '$d' > test_PDB255_dna_binding.fa
sed '$d' outfile04 | sed '$d' > test_PDB255_rna_binding.fa
rm outfile0*

# we combine the full dataset and later repartition it with the pipeline
cat train_dna_binding.fa test_TEST474_dna_binding.fa test_PDB255_dna_binding.fa > dna_
↪binding.fa
cat train_rna_binding.fa test_TEST474_rna_binding.fa test_PDB255_rna_binding.fa > rna_
↪binding.fa
gzip dna_binding.fa
gzip rna_binding.fa
```

The files can be downloaded using the above script. The original publication is accessible here.

```
HEADER:   >Q7YU81
SEQUENCE: MATLIPVNGGHPAASGQSSNVEATYEDMFKEITRKLYGEETGNGLHTLGTPVAQVATSGP
          TAVPEGEQRSFTNLQQLDRSAAPSIEYESSAAGASGNNVATTQANVIQQQQQQQQQAESG
          NSVVVTASSGATVVPAPSVAAVGGFKSEDHLSTAFGLAALMQNGFAAGQAGLLKAGEQQQ
          RWAQDGSGLVAAAAAEPQLVQWTSGGKLQSYAHVNQQQQQQQQPHQSTPKSKKHRQEHAA..
```

Note: *In real world data, other characters are available which refer to multiple possible nucleotides, for example ``W`` indicates either an ``A`` or a ``T``. RNA includes the character ``U``, and proteins include additional letters of the alphabet.*

Tokenisation in genomics involves segmenting biological sequences into smaller units, called tokens (or k-mers in biology) for further processing. In the context of proteins, tokens can represent individual amino acids, k-mers or other biologically meaningful segments. Just as in conventional NLP, tokenisation is required to facilitate most downstream operations.

Here, we provide gzipped fasta file(s) as input. While conventional biological tokenisation splits a sequence into arbitrary-length segments, empirical tokenisation derives the resulting tokens directly from the corpus, with vocabulary size as the only user-defined parameter. Data is then split into training, testing and/or validation partitions as desired by the user and automatically reformatted for input into the deep learning pipeline.

Note: *We provide the conventional k-merisation method as well as an option for users. In our pipeline specifically, the empirical tokenisation and data object creation is split into two steps, while k-merisation combines both in one operation. This is due to the empirical tokenisation process having to "learn" tokens from the data.*

```
# Empirical tokenisation pathway
$ tokenise_bio  -i dna_binding.fa.gz rna_binding.fa.gz  -t prot.2000.json -v 2000
# -i INFILE_PATHS path to files with biological seqs split by line
# -t TOKENISER_PATH path to tokeniser.json file to save or load data
# -v VOCAB_SIZE select vocabulary size (DEFAULT: 32000)
```

This generates a `json` file with tokens and their respective weights or IDs. You should see some output like this.

```
[00:00:00] Pre-processing sequences
[00:00:00] Suffix array seeds
[00:00:14] EM training
```

4.2 5. Format a dataset for input into genomeNLP

In this section, we reformat the data to meet the requirements of our pipeline which takes specifically structured inputs. This intermediate data structure serves as the foundation for downstream analyses and facilitates seamless integration with the pipeline. Our pipeline contains a method that performs this automatically, generating a reformatted dataset with the desired structure.

Note: *The data format is identical to that used by the HuggingFace ``datasets`` and ``transformers`` libraries.*

```
# Empirical tokenisation pathway
$ create_dataset_bio \
    dna_binding.fa.gz  \
    rna_binding.fa.gz \
    prot.2000.json \
    -o prot.2000.512 \
    --no_reverse_complement \
    -c 512
# -o OUTFILE_DIR write dataset to directory as
# [ csv \| json \| parquet \| dir/ ] (DEFAULT:"hf_out/")
# --no_reverse_complement  turn off reverse complement (DEFAULT: ON)
# -c CHUNK  split seqs into n-length blocks (DEFAULT: None)
# default datasets split: train 90%, test 5% and validation set 5%
```

The output is a reformatted dataset containing the same information. Properties required for a typical machine learning pipeline are added, including labels, customisable data splits and token identifiers.

```
DATASET AFTER SPLIT:
DatasetDict ({
  train: Dataset ({
  features: ['idx', 'feature', 'labels', 'input_ids', 'token_type_ids', 'attention_
↪mask'],
  num_rows: 9719 })
  test: Dataset ({
  features: ['idx', 'feature', 'labels', 'input_ids', 'token_type_ids', 'attention_
↪mask'],
  num_rows: 540 })
  valid: Dataset ({
  features: ['idx', 'feature', 'labels', 'input_ids', 'token_type_ids', 'attention_
↪mask'],
  num_rows: 540 })
})
```

Note: *The column ``token_type_ids`` is not actually needed in this specific case study, but it is safely ignored in such cases.*

```
SAMPLE TOKEN MAPPING FOR FIRST 5 TOKENS IN SEQ:
TOKEN ID: 400  | TOKEN: MA
TOKEN ID: 533  | TOKEN: SQS
TOKEN ID: 1742 | TOKEN: EPG
```

(continues on next page)

(continued from previous page)

```
TOKEN ID: 296  | TOKEN: YL
TOKEN ID: 346  | TOKEN: AAA
```

4.3 6. Preparing a hyperparameter sweep

In machine learning, achieving optimal model performance often requires finding the right combination of hyperparameters (assuming the input data is viable). Hyperparameters vary depending on the specific algorithm and framework being used, but commonly include learning rate, dropout rate, batch size, number of layers and optimiser choice. These parameters heavily influence the learning process and subsequent performance of the model.

For this reason, hyperparameter sweeps are normally carried out to systematically test combinations of hyperparameters, with the end goal of identifying the configuration that produces the best model performance. Usually, sweeps are carried out on a small partition of the data only to maximise efficiency of compute resources, but it is not uncommon to perform sweeps on entire datasets. Various strategies, such as grid search, random search, or bayesian optimisation, can be employed during a hyperparameter sweep to sample parameter values. Additional strategies such as early stopping can also be used.

To streamline the hyperparameter optimization process, we use the `wandb` (Weights & Biases) platform which has a user-friendly interface and powerful tools for tracking experiments and visualising results.

First, sign up for a wandb account at: https://wandb.ai/site and login by paste your API key.

```
$ wandb login
$ wandb: Paste an API key from your profile, and hit enter and hit enter or press ctrl+c␣
↪to quit :
```

Now, we use the `sweep` tool to perform hyperparameter sweep. Search strategy, parameters and search space are passed in as a `json` file.

```
# sweep parameters
{
  "method": "random",
  "name": "sweep",
  "metric": {
    "goal": "maximize",
    "name": "eval/f1"
  },
  "parameters": {
    "batch_size": {"values": [5, 10, 15]},
    "epochs": {"values": [1, 2, 3, 4, 5]},
    "learning_rate": {"max": 0.1, "min": 0.0001}
  }
}
```

```
$ sweep \
    prot.2000.512/train.parquet \
    parquet \
    prot.2000.json \
    --test prot.2000.512/test.parquet \
    --valid prot.2000.512/valid.parquet \
    --hyperparameter_sweep random.json \
```

(continues on next page)

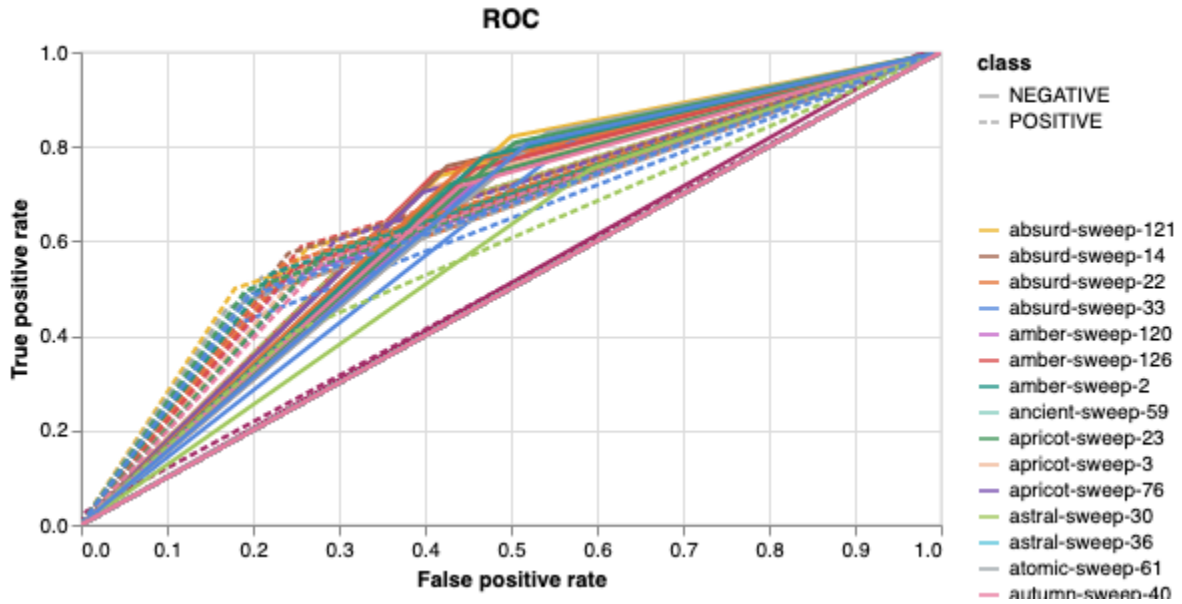
(continued from previous page)

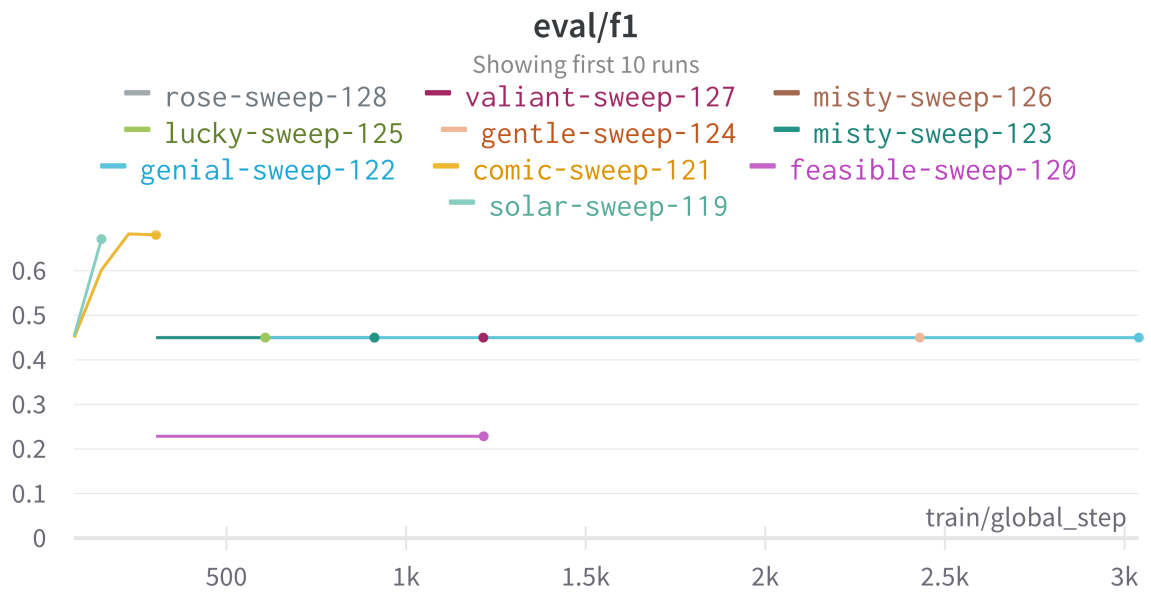
```
    --entity_name tyagilab \ # <- edit as needed
    --project_name p_sweep \ # <- edit as needed
    --group_name prot.2000 \
    --output_dir sweep.2000 \
    --label_names "labels" \
    -n 3

# --test, path to [ csv \| csv.gz \| json \| parquet ] file
# --valid, path to [ csv \| csv.gz \| json \| parquet ] file
# --hyperparameter_sweep, run a hyperparameter sweep with config from file
# --entity_name, wandb team name (if available).
# --project_name, wandb project name (if available)
# --group_name, provide wandb group name (if desired)
# --label_names, provide column with label names (DEFAULT: "")
# -n SWEEP_COUNT, run n hyperparameter sweeps
# -o OUTPUT_DIR, specify path for output (DEFAULT: ./sweep_out)
```

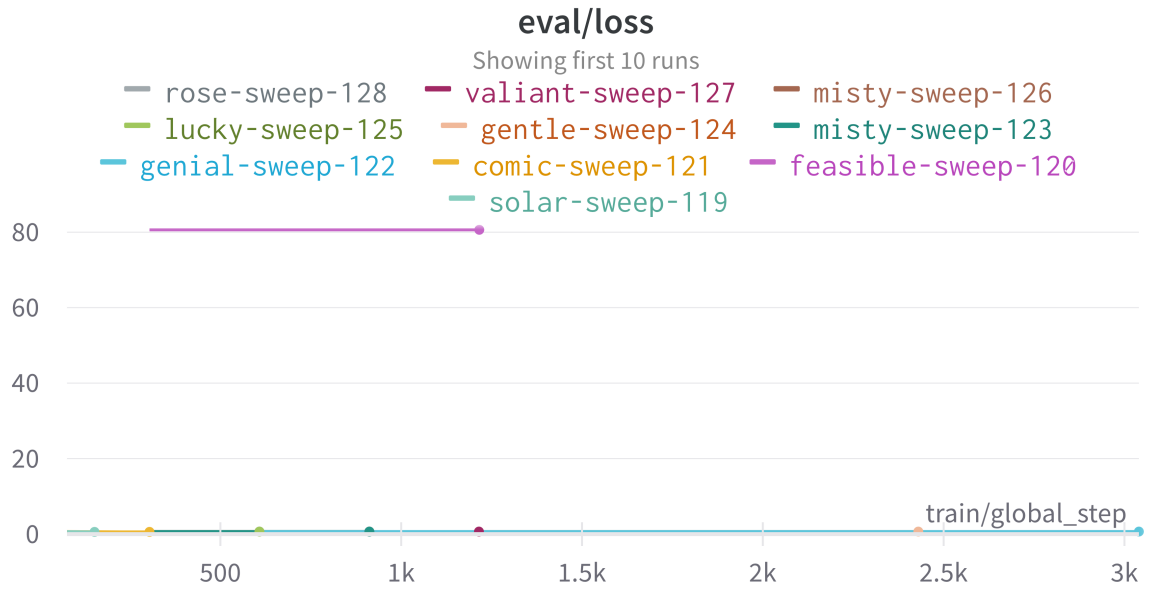
```
*****Running training*****
Num examples = 9719
Num epochs= 1
Instantaneous batch size per device = 5
Total train batch size per device = 5
Gradient Accumulation steps= 1
Total optimization steps= 1944
```

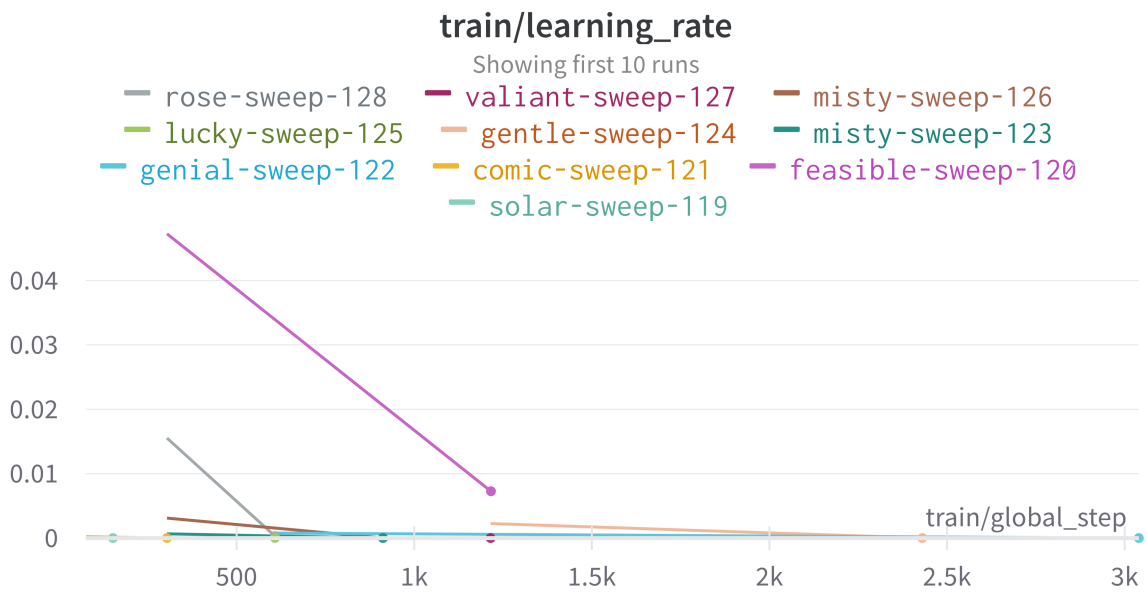
The output is written to the specified directory, in this case `sweep_out` and will contain the output of a standard `pytorch` saved model, including some `wandb` specific output.

The sweeps gets synced to the `wandb` dashboard along with various interactive custom charts and tables which we provide as part of our pipeline. A small subset of plots are provided for reference. Interactive versions of these and more plots are available on wandb.


ROC
True positive rate
False positive rate
0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0
class
NEGATIVE
POSITIVE
absurd-sweep-121
absurd-sweep-14
absurd-sweep-22
absurd-sweep-33
amber-sweep-120
amber-sweep-126
amber-sweep-2
ancient-sweep-59
apricot-sweep-23
apricot-sweep-3
apricot-sweep-76
astral-sweep-30
astral-sweep-36
atomic-sweep-61


eval/f1
Showing first 10 runs
rose-sweep-128
valiant-sweep-127
misty-sweep-126
lucky-sweep-125
gentle-sweep-124
misty-sweep-123
genial-sweep-122
comic-sweep-121
feasible-sweep-120
solar-sweep-119
0.6
0.5
0.4
0.3
0.2
0.1
0
train/global_step
500
1k
1.5k
2k
2.5k
3k


eval/loss
Showing first 10 runs
rose-sweep-128
valiant-sweep-127
misty-sweep-126
lucky-sweep-125
gentle-sweep-124
misty-sweep-123
genial-sweep-122
comic-sweep-121
feasible-sweep-120
solar-sweep-119
80
60
40
20
0
train/global_step
500
1k
1.5k
2k
2.5k
3k



Here is an example of a full wandb generated report:

You may inspect your own generated reports after they complete.

4.4 7. Selecting optimal hyperparameters for training

Having completed a sweep, we next identified the best set of parameters for model training. We do this by examining training metrics. These serve as quantitative measures of a model's performance during training. These metrics provide insights into the model's accuracy and generalisation capabilities. We explore commonly used training metrics, including accuracy, loss, precision, recall, and f1 score to inform us of a model's performance

A key event we want to avoid is overfitting. Overfitting occurs when a learning model performs exceptionally well on the training data but fails to generalise to unseen data, making it unfit for use outside of the specific scope of the experiment. This can be detected by observing performance metrics, if the accuracy decreases and later increases an overfit event has occurred. In real world applications, this can lead to adverse events that directly impact us, considering that such models are used in applications such as drug prediction or self-driving cars. Here, we use the f1 score calculated on the testing set as the main metric of interest. We showed that we obtain a best `f1` score of `0.677488189237731`.

```
Best run kind-sweep-18 with eval/f1=0.677488189237731
BEST MODEL AND CONFIG FILES SAVED TO: protein_sweep/model_files
HYPERPARAMETER SWEEP END
```

`Here is an example of a full wandb generated report for the "best" run. <https://api.wandb.ai/links/tyagilab/58zmy653`__

You may inspect your own generated reports after they complete.

4.5 8. With the selected hyperparameters, train the full dataset

In a conventional workflow, the sweep is performed on a small subset of training data. The resulting parameters are then recorded and used in the actual training step on the full dataset. Here, we perform the sweep on the entire dataset, and hence remove the need for further training. If you perform this on your own data and want to use a small subset, you can do so and then pass the recorded hyperparameters with the same input data to the `train` function of the pipeline. We include an example of this below for completeness, but you can skip this for our specific case study. Note that the input is almost identical to `sweep`.

```
$ train \
    prot.2000.512/train.parquet \
    "parquet" \
    prot.2000.json \
    --test prot.2000.512/test.parquet \
    --valid prot.2000.512/valid.parquet \
    --entity_name tyagilab \
    --project_name prot \
    --group_name train.2000 \
    --config_from_run tyagilab/prot/2niwyeqs \
    --output_dir train.out \
    --label_names "labels" \
    --overwrite_output_dir
# -t TEST, path to [ csv \| csv.gz \| json \| parquet ] file
# -v VALID, path to [ csv \| csv.gz \| json \| parquet ] file
# -w HYPERPARAMETER_SWEEP, run a hyperparameter sweep with config from file
# -e ENTITY_NAME, wandb team name (if available).
# -p PROJECT_NAME, wandb project name (if available)
# -l LABEL_NAMES, provide column with label names (DEFAULT: "").
# -n SWEEP_COUNT, run n hyperparameter sweeps
```

```
{
  "output_dir": "./sweep_out/random",
  "overwrite_output_dir": false,
  "do_train": false,
  "do_eval": true,
  "do_predict": false,
  "evaluation_strategy": "epoch",
  "prediction_loss_only": false,
  "per_device_train_batch_size": 32,
  "per_device_eval_batch_size": 32,
  "per_gpu_train_batch_size": null,
  "per_gpu_eval_batch_size": null,
  "gradient_accumulation_steps": 1,
  "eval_accumulation_steps": null,
  "eval_delay": 0,
  "learning_rate": 0.00000017248305228664,
  "weight_decay": 0.5,
  "adam_beta1": 0.9,
  "adam_beta2": 0.999,
  "adam_epsilon": 1e-08,
  "max_grad_norm": 1.0,
  "num_train_epochs": 2,
```

(continues on next page)

(continued from previous page)


```
  "max_steps": -1,
  "lr_scheduler_type": "linear",
  "warmup_ratio": 0.0,
  "warmup_steps": 0,
  "log_level": "passive",
  "log_level_replica": "passive",
  "log_on_each_node": true,
  "logging_dir": "./sweep_out/random/runs/out",
  "logging_strategy": "epoch",
  "logging_first_step": false,
  "logging_steps": 500,
  "logging_nan_inf_filter": true,
  "save_strategy": "epoch",
  "save_steps": 500,
  "save_total_limit": null,
  "save_on_each_node": false,
  "no_cuda": false,
  "use_mps_device": false,
  "seed": 42,
  "data_seed": null,
  "jit_mode_eval": false,
  "use_ipex": false,
  "bf16": false,
  "fp16": false,
  "fp16_opt_level": "O1",
  "half_precision_backend": "auto",
  "bf16_full_eval": false,
  "fp16_full_eval": false,
  "tf32": null,
  "local_rank": -1,
  "xpu_backend": null,
  "tpu_num_cores": null,
  "tpu_metrics_debug": false,
  "debug": [],
  "dataloader_drop_last": false,
  "eval_steps": null,
  "dataloader_num_workers": 0,
  "past_index": -1,
  "run_name": "./sweep_out/random",
  "disable_tqdm": false,
  "remove_unused_columns": false,
  "label_names": null,
  "load_best_model_at_end": true,
  "metric_for_best_model": "loss",
  "greater_is_better": false,
  "ignore_data_skip": false,
  "sharded_ddp": [],
  "fsdp": [],
  "fsdp_min_num_params": 0,
  "fsdp_transformer_layer_cls_to_wrap": null,
  "deepspeed": null,
  "label_smoothing_factor": 0.0,
```

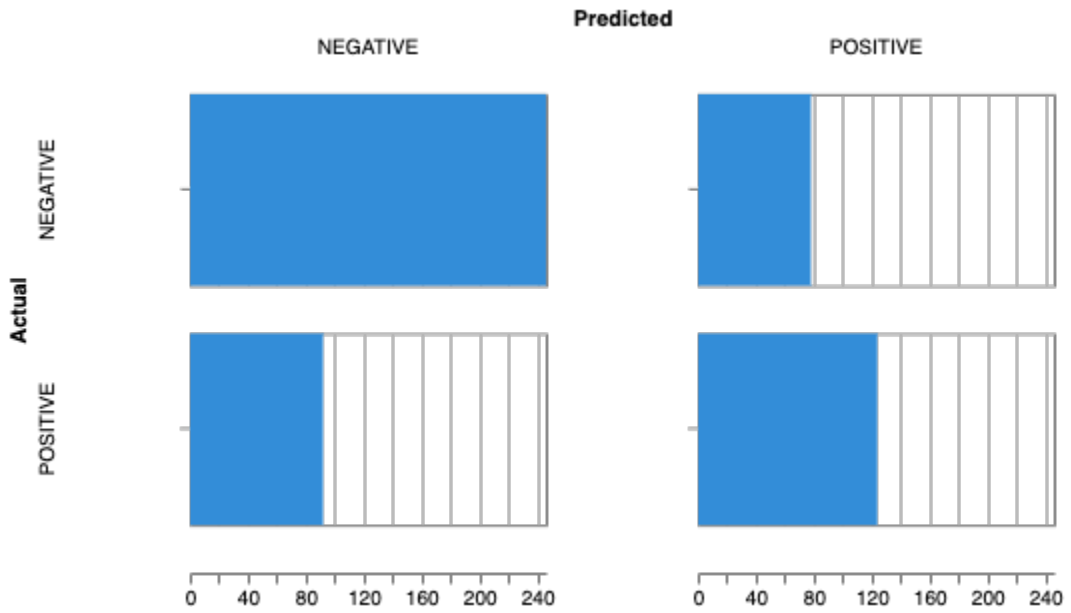

(continues on next page)

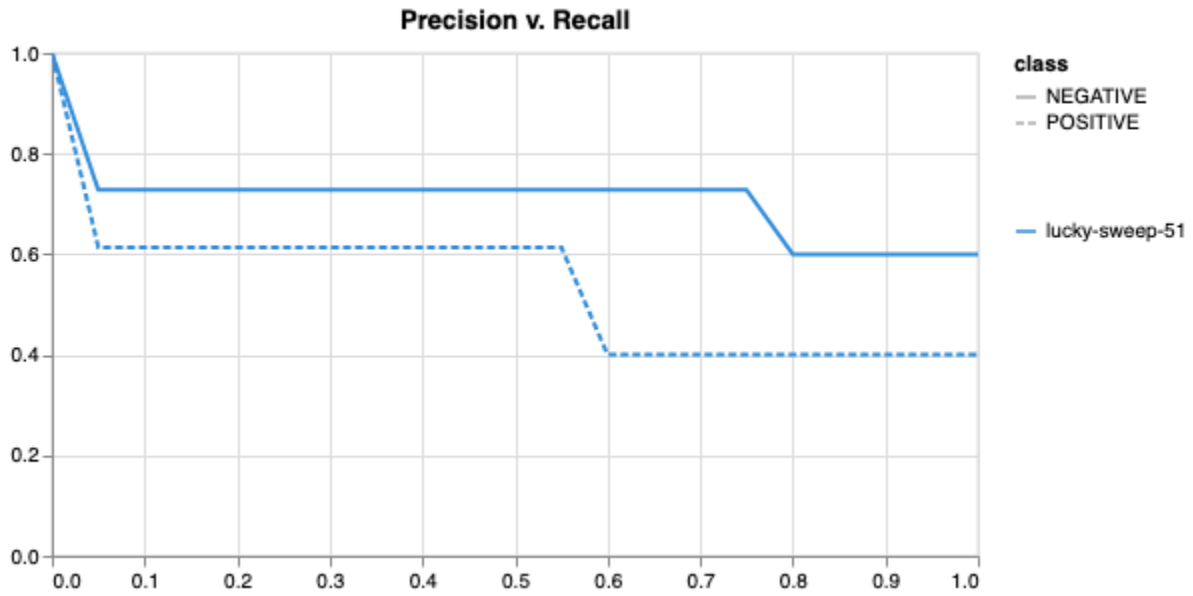
(continued from previous page)

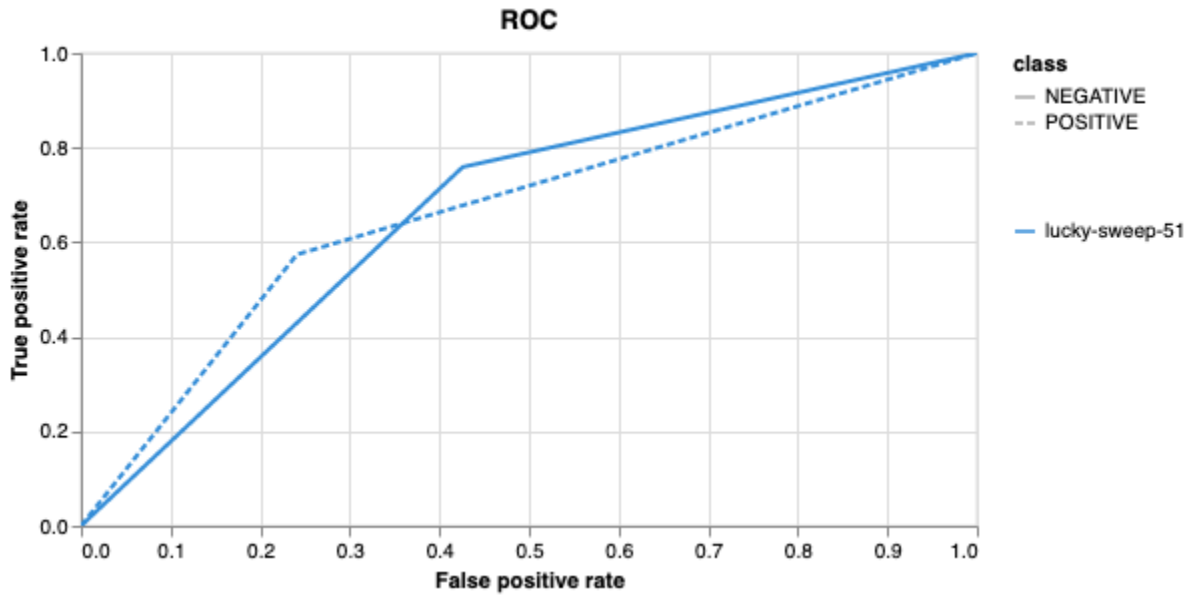
```
  "optim": "adamw_hf",
  "adafactor": false,
  "group_by_length": false,
  "length_column_name": "length",
  "report_to": [
    "wandb"
  ],
  "ddp_find_unused_parameters": null,
  "ddp_bucket_cap_mb": null,
  "dataloader_pin_memory": true,
  "skip_memory_metrics": true,
  "use_legacy_prediction_loop": false,
  "push_to_hub": false,
  "resume_from_checkpoint": null,
  "hub_model_id": null,
  "hub_strategy": "every_save",
  "hub_token": "<HUB_TOKEN>",
  "hub_private_repo": false,
  "gradient_checkpointing": false,
  "include_inputs_for_metrics": false,
  "fp16_backend": "auto",
  "push_to_hub_model_id": null,
  "push_to_hub_organization": null,
  "push_to_hub_token": "<PUSH_TO_HUB_TOKEN>",
  "mp_parameters": "",
  "auto_find_batch_size": false,
  "full_determinism": false,
  "torchdynamo": null,
  "ray_scope": "last",
  "ddp_timeout": 1800
}
```

The output is written to the specified directory, in this case `train_out` and will contain the output of a standard `pytorch` saved model, including some `wandb` specific output.

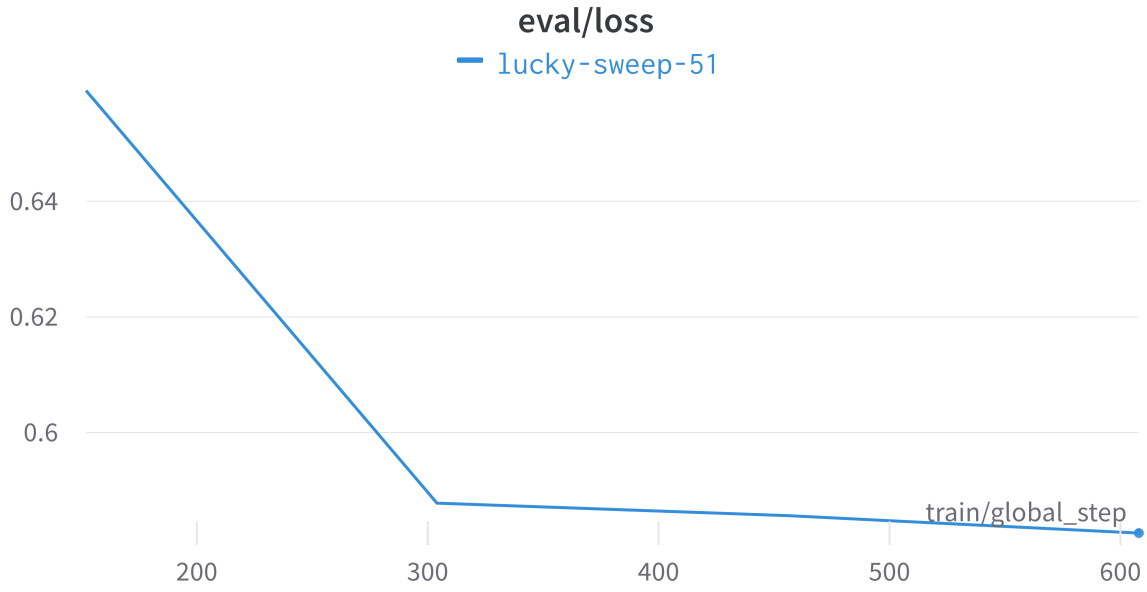
The trained model gets synced to the `wandb` dashboard along with various interactive custom charts and tables which we provide as part of our pipeline. A small subset of plots are provided for reference. Interactive versions of these and more plots are available on wandb.

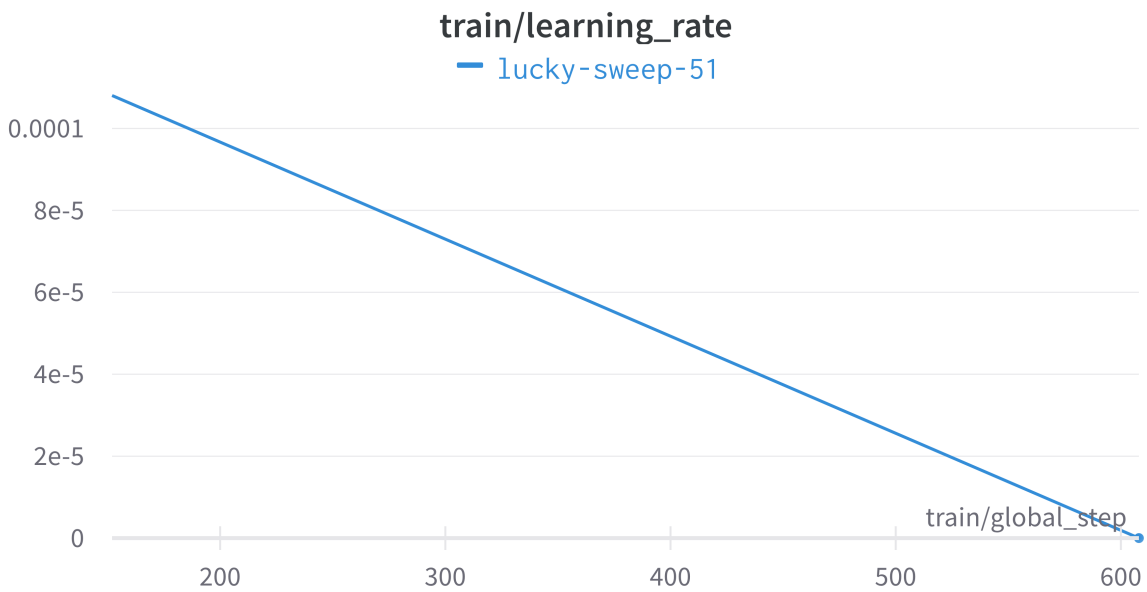

Predicted
NEGATIVE
POSITIVE
Actual
NEGATIVE
POSITIVE
0
40
80
120
160
200
240


Precision v. Recall
1.0
0.8
0.6
0.4
0.2
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
class
NEGATIVE
POSITIVE
lucky-sweep-51



fig/protein/train_f1.png





Here is an example of a full wandb generated report:

You may inspect your own generated reports after they complete.

4.6 9. Perform cross-validation

Having identified the best set of parameters and trained the model, we next want to conduct a comprehensive review of data stability, and we do this by evaluating model performance across different data slices. This assessment is known as cross-validation. We make use of k-fold cross-validation in which data is divided into k subsets and the model is trained and tested on these individual subsets.

```
$ cross_validate \
    data.csv/train.parquet parquet \
    -t data.csv/test.parquet \
    -v data.csv/valid.parquet \
    -e tyagilab \
    -p testm3 \
    --config_from_run p9do3gzl \  # id of best performing run
    --output_dir cv \
    -m sweep_out \
    -l labels \
    -k 3
# --config_from_run WANDB_RUN_ID, *best run id*
# --output_dir OUTPUT_DIR
# -l label_names
# -k KFOLDS, run n number of kfolds
```

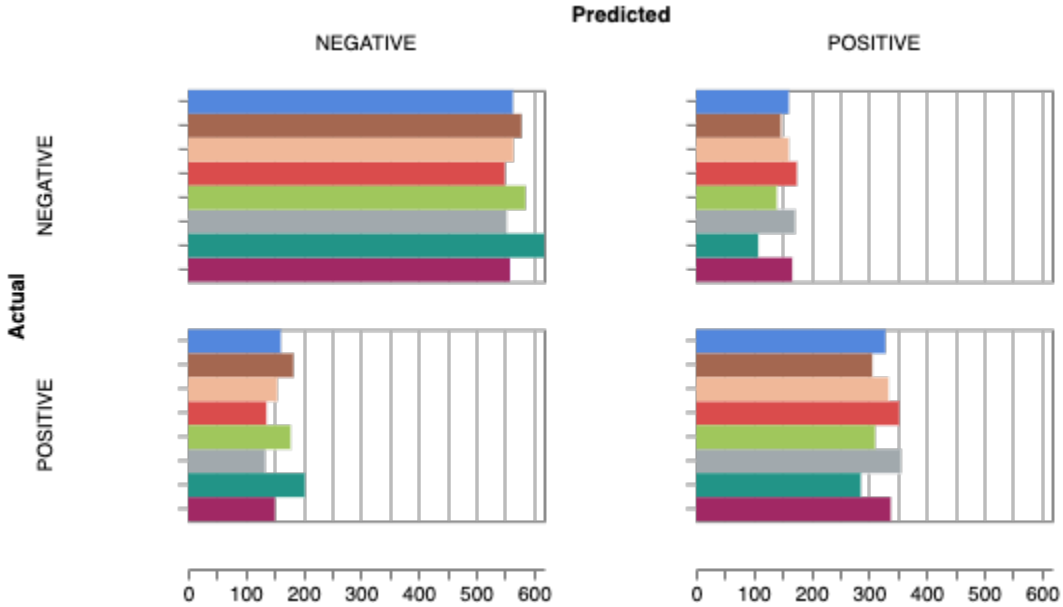
```
*****Running training*****
Num examples = 8504
Num epochs= 4
Instantaneous batch size per device = 64
```

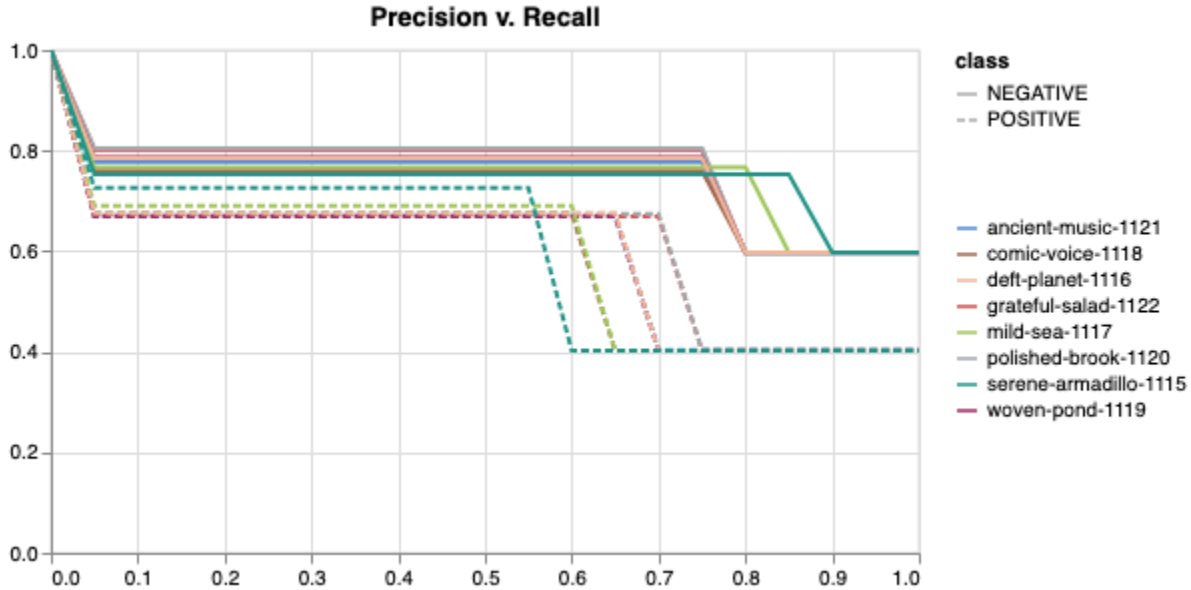
(continues on next page)

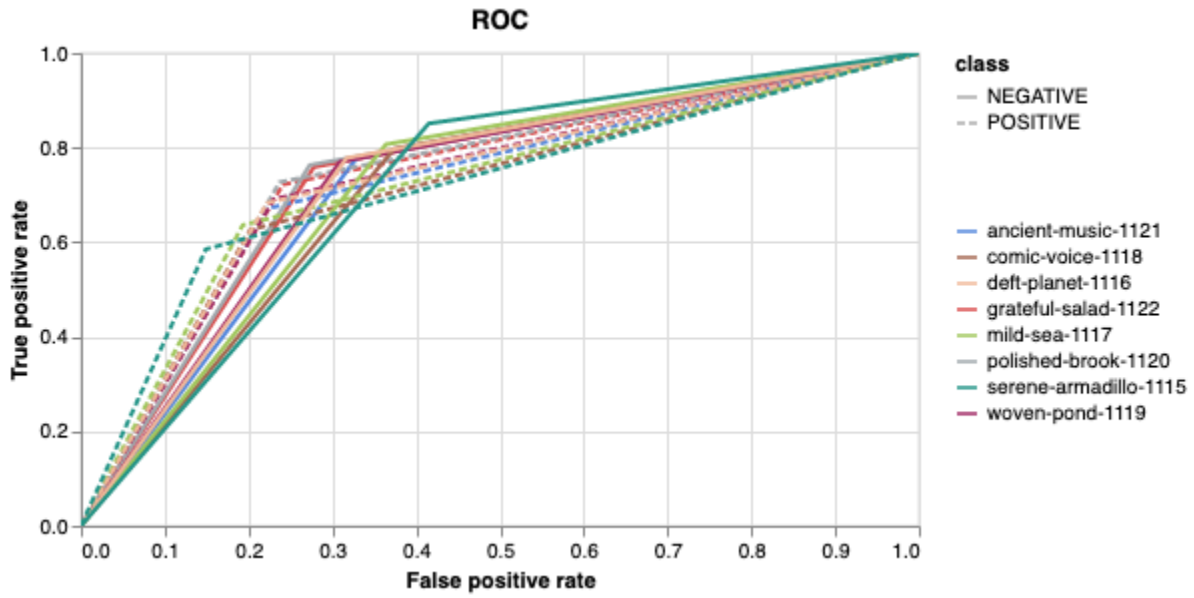
(continued from previous page)

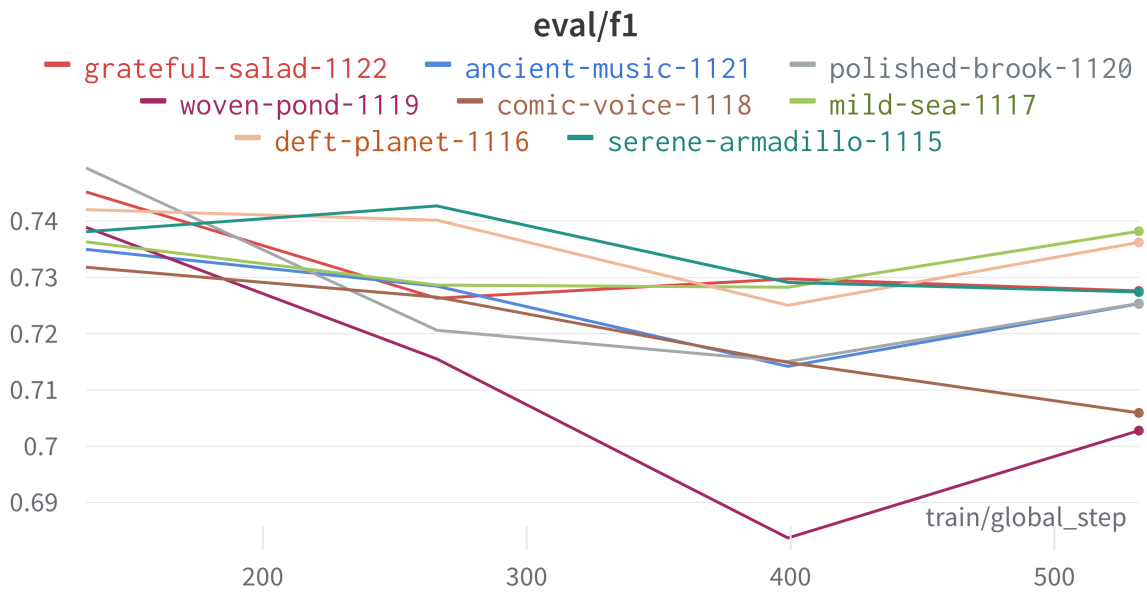
```
Total train batch size (w, parallel, distributed & accumulation)= 64
Gradient Accumulation steps= 1
Total optimization steps= 532
Automatic Weights & Biases logging enabled
```

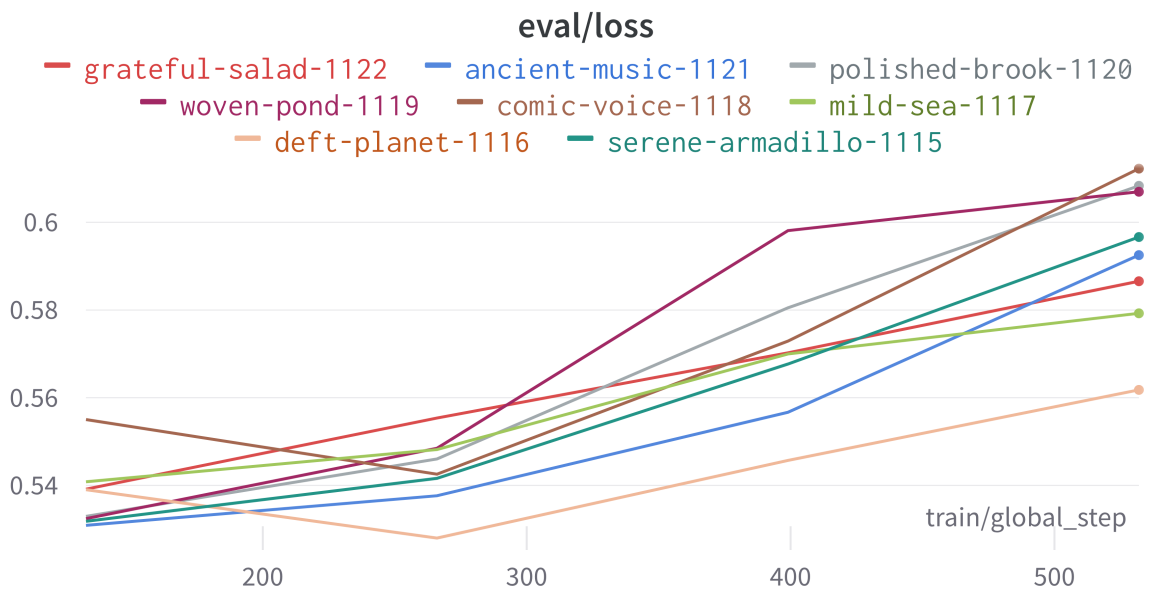
The cross-validation runs are uploaded to the `wandb` dashboard along with various interactive custom charts and tables which we provide as part of our pipeline. These are conceptually identical to those generated by `sweep` or `train`. A small subset of plots are provided for reference. Interactive versions of these and more plots are available on wandb.

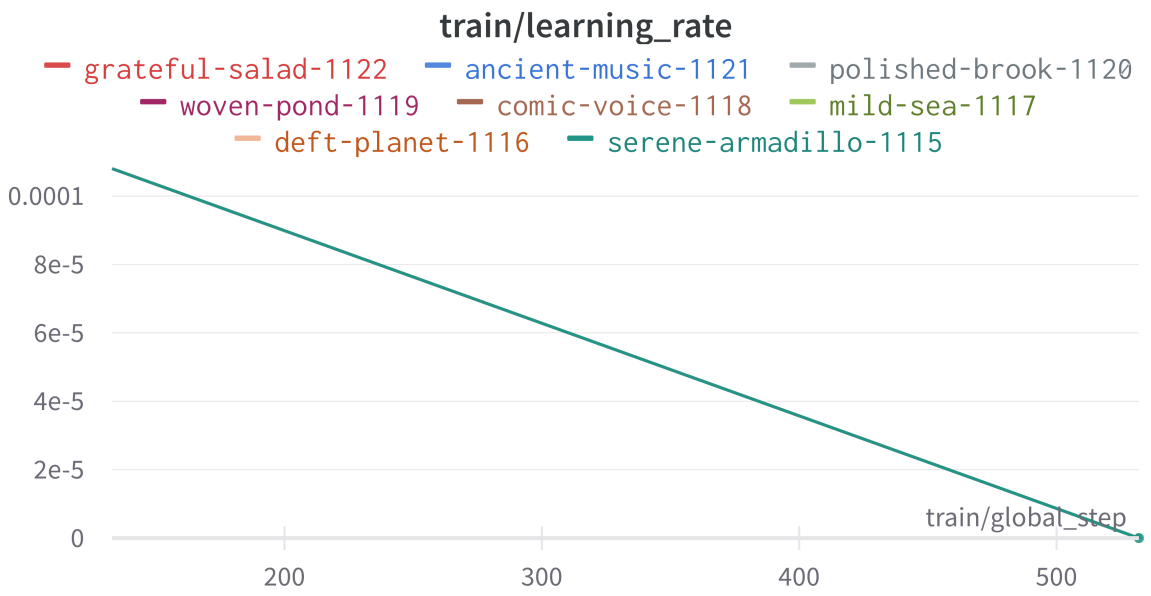





ROC
True positive rate
False positive rate
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0
class
NEGATIVE
POSITIVE
ancient-music-1121
comic-voice-1118
deft-planet-1116
grateful-salad-1122
mild-sea-1117
polished-brook-1120
serene-armadillo-1115
woven-pond-1119


eval/f1
grateful-salad-1122
ancient-music-1121
polished-brook-1120
woven-pond-1119
comic-voice-1118
mild-sea-1117
deft-planet-1116
serene-armadillo-1115
0.74
0.73
0.72
0.71
0.7
0.69
train/global_step
200
300
400
500





Here is an example of a full wandb generated report:

You may inspect your own generated reports after they complete.

4.7 10. Compare different models

The aim of this step is to compare performance of different deep learning models efficiently while avoiding computationally expensive re-training and data download in conventional model comparison. In the case of patient data, they are often inaccessible for privacy reasons, and in other cases they are not uploaded by the authors of the experiment.

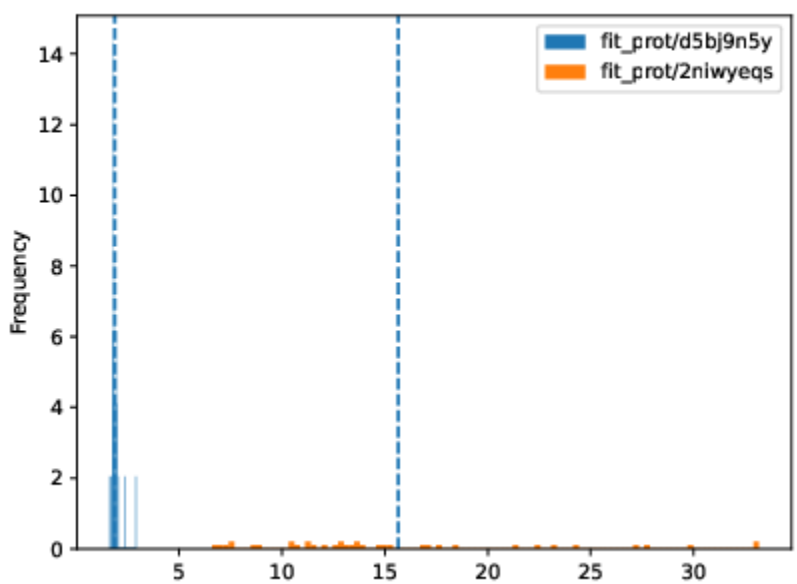
For the purposes of this simple case study, we compare multiple sweeps of the same dataset as a demonstration. In a real life application, existing biological models can be compared against the user-generated one.

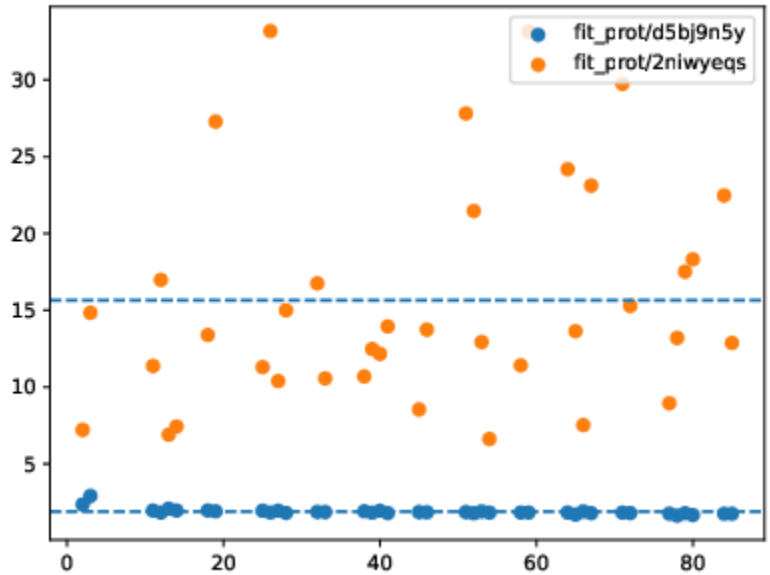
```
$ fit_powerlaw tyagilab/prot/d5bj9n5y tyagilab/prot/2niwyeqs -o fit_prot
# -m MODEL_PATH, path to trained model directory
# -o OUTPUT_DIR, path to output metrics directory
```

This tool outputs a variety of plots in the specified directory.

```
$ ls fit_prot
> alpha_hist.pdf  alpha_plot.pdf  model_files/
```

Very broadly, the overlaid bar plots allow the user to compare the performance of different models on the same scale. A narrow band around 2-5 with few outliers is in general cases an indicator of good model performance. This is a general guideline and will differ depending on context! For a detailed explanation of these plots, please refer to the original publication.





4.8 11. Obtain model interpretability scores

Model interpretability is often used for debugging purposes, by allowing the user to "see" (to an extent) what a model is focusing on. In this case, the tokens which contribute to a certain classification are highlighted. The green colour indicates a classification towards the target category, while the red colour indicates a classification away from the target category. Colour intensity indicates the classification score.

In some scenarios, we can exploit this property by identifying regulatory regions or motifs in DNA sequences, or discovering amino acid residues in protein structure critical to its function, leading to a deeper understanding of the underlying biological system.

```
$ gzip -cd dna.binding.fa.gz | head -n22 > dna_subset.fasta
$ interpret tyagilab/prot/d5bj9n5y dna_subset.fasta -o prot_interpret
# -o OUTPUT_DIR, specify path for output
```

4.9 Citation

Cite our manuscript here:

```
@article{chen2023genomicbert,
    title={genomicBERT and data-free deep-learning model evaluation},
    author={Chen, Tyrone and Tyagi, Navya and Chauhan, Sarthak and Peleg, Anton Y and␣
↪Tyagi, Sonika},
    journal={bioRxiv},
    month={jun},
```

(continues on next page)

(continued from previous page)

```
    pages={2023--05},
    year={2023},
    publisher={Cold Spring Harbor Laboratory},
    doi={10.1101/2023.05.31.542682},
    url={https://doi.org/10.1101/2023.05.31.542682}
}
```

Cite our software here:

```
@software{tyrone_chen_2023_8135591,
  author       = {Tyrone Chen and
                  Navya Tyagi and
                  Sarthak Chauhan and
                  Anton Y. Peleg and
                  Sonika Tyagi},
  title        = {{genomicBERT and data-free deep-learning model
                  evaluation}},
  month        = jul,
  year         = 2023,
  publisher    = {Zenodo},
  version      = {latest},
  doi          = {10.5281/zenodo.8135590},
  url          = {https://doi.org/10.5281/zenodo.8135590}
}
```

CHAPTER

FIVE

CREATE A TOKEN SET FROM SEQUENCES

This explains the use of `kmerise_bio.py` and `tokenise_bio.py`. In `tokenise_bio.py` we empirically derive tokens from biological sequence data which can be used in downstream applications such as `genomicBERT`.

5.1 Source data

Any `fasta` file can be used (nucleic acid or protein).

5.2 Results

Note: Entry points are available if this is installed using the automated conda method. You can then use the command line argument directly, for example: `create_dataset_bio`. If not, you will need to use the script directly, which follows the same naming pattern, for example: `python create_dataset_bio.py`.

Running the code as below:

5.2.1 Empirical tokenisation

```
python tokenise_bio.py -i [ INFILE_PATH ... ] -t TOKENISER_PATH
```

You will obtain a `json` file with weights for each token. Any special tokens you add will also be present. This will be used in the next step of creating a `HuggingFace` compatible dataset object.

5.2.2 Conventional k-mers

```
python kmerise_bio.py -i [INFILE_PATH ... ] -t TOKENISER_PATH -k KMER_SIZE -l [LABEL ...␣
↪] -c CHUNK -o OUTFILE_DIR
```

For k-mers, `HuggingFace`-like dataset files will be written to disk in the same operation. This can be loaded directly into a "conventional" deep learning pipeline.

However, the file is not split into partitions. You can use it directly if you already have other partitions corresponding to training, testing and validation data. If not, you will need to create a dataset in the next stage, using the `tokeniser.json` file generated in this step.

5.3 Notes

Please refer to HuggingFace tokenisers for more detailed information:

5.4 Usage

5.4.1 Empirical tokenisation

For empirical tokenisation, the next step is to run `create_dataset_bio.py`. Reverse complementing `Y/R` is supported.

```
python tokenise.py -h
usage: tokenise.py [-h] [-i INFILE_PATHS [INFILE_PATHS ...]] [-t TOKENISER_PATH]
                   [-s SPECIAL_TOKENS [SPECIAL_TOKENS ...]] [-e EXAMPLE_SEQ]

Take gzip fasta file(s), run empirical tokenisation and export json.

options:
  -h, --help            show this help message and exit
  -i INFILE_PATHS [INFILE_PATHS ...], --infile_paths INFILE_PATHS [INFILE_PATHS ...]
                        path to files with biological seqs split by line
  -t TOKENISER_PATH, --tokeniser_path TOKENISER_PATH
                        path to tokeniser.json file to save or load data
  -v VOCAB_SIZE, --vocab_size VOCAB_SIZE
                        select vocabulary size (DEFAULT: 32000)
  -b BREAK_SIZE, --break_size BREAK_SIZE
                        split long reads, keep all by default (DEFAULT: None)
  -c CASE, --case CASE  change case, retain original by default (DEFAULT: None)
  -s SPECIAL_TOKENS [SPECIAL_TOKENS ...], --special_tokens SPECIAL_TOKENS [SPECIAL_
↪TOKENS ...]
                        assign special tokens, eg space and pad tokens
                        (DEFAULT: ["<s>", "</s>", "<unk>", "<pad>", "<mask>"])
  -e EXAMPLE_SEQ, --example_seq EXAMPLE_SEQ
                        show token to seq map for a sequence (DEFAULT: None)

      usage: compare_empirical_tokens.py [-h] [-t TOKENISER_PATH] [-w TOKEN_WEIGHT] [-m␣
↪MERGE_STRATEGY]
                                  [-p POOLING_STRATEGY] [-o OUTFILE_PATH]
                                  infile_paths [infile_paths ...]

# supplementary script for evaluating tokenisation performance across contig lengths or␣
↪sequence subsets
python compare_empirical_tokens.py -h
usage: compare_empirical_tokens.py [-h] [-t TOKENISER_PATH] [-w TOKEN_WEIGHT] [-m MERGE_
↪STRATEGY]
                                  [-p POOLING_STRATEGY] [-o OUTFILE_PATH]
                                  infile_paths [infile_paths ...]
Take token json files, show intersection and weight variance.

positional arguments:
  infile_paths          path to tokeniser files generated by tokenise_bio
```

(continues on next page)

(continued from previous page)

```
optional arguments:
  -h, --help            show this help message and exit
  -t TOKENISER_PATH, --tokeniser_path TOKENISER_PATH
                        path to pooled tokeniser (DEFAULT: pooled.json)
  -w TOKEN_WEIGHT, --token_weight TOKEN_WEIGHT
                        path to output file showing token weights status
  -m MERGE_STRATEGY, --merge_strategy MERGE_STRATEGY
                        merge tokens using [ inner | outer ] (DEFAULT: None)
  -p POOLING_STRATEGY, --pooling_strategy POOLING_STRATEGY
                        pool tokens using [ mean | median | max | min ] (DEFAULT: None)
  -o OUTFILE_PATH, --outfile_path OUTFILE_PATH
                        path to output boxplot showing token weights distribution
```

Handling long reads

The original word segmentation algorithm was designed for sentences in human language. In biology, a chromosome can be formulated as a single sentence. In such cases, empirical tokenisation breaks if a sequence length of greater than ~3-4 Mbp is provided.

Since there is a limit to sequence length, here we explore the feasability of subsampling sequences as a workaround. We obtain a small genome which can be tokenised fully as a control, and split its genome into different contig lengths to get a collection of smaller sequences. We then compare the (a) empirical token weights and (b) token identity across different contig lengths.

We choose the *Haemophilus influenzae* genome since it can be fully tokenised:

```
#!/bin/sh
# download Haemophilus influenzae genome
curl -OJX GET "https://api.ncbi.nlm.nih.gov/datasets/v2alpha/genome/accession/GCF_
↪000931575.1/download?include_annotation_type=GENOME_FASTA,GENOME_GFF,RNA_FASTA,CDS_
↪FASTA,PROT_FASTA,SEQUENCE_REPORT&filename=GCF_000931575.1.zip" -H "Accept: application/
↪zip"
unzip GCF_000931575.1.zip
cp ncbi_dataset/data/GCF_000931575.1/GCF_000931575.1_ASM93157v1_genomic.fna ./
gzip GCF_000931575.1_ASM93157v1_genomic.fna
```

We split the genome into different contig lengths spanning 2**9 to 2**20, and retain the full genome as a control:

```
# NOTE: this will take some time!
# 0 for ground truth
for len in 0 512 1024 2048 4096 8192 16384 32768 65536 131072 262144 524288 1048576; do
  tokenise_bio \
  -i GCF_000931575.1_ASM93157v1_genomic.fna.gz \
  -v 10000 \
  -t tokens_contigs.${len}.10000.json \
  -c upper \
  -b ${len}

# you will see the tokeniser files generated as a result
ls *10000.json
```

Next, we examine the token weights for each contig length compared to the whole genome. Token weights and outliers are exported, along with a boxplot showing variance of the weight distribution:

```
# compare the ground truth tokens vs each contig length
for i in *json; do
  compare_empirical_tokens \
    tokens_contigs.0.10000.json \
    $i \
    -t ${i}.tsv \
    -o ${i}.pdf
done

# compare all contig lengths together
compare_empirical_tokens *json -t all.tsv -o all.pdf
```

Note: You can use `compare_empirical_tokens` with any combination of `json` files as a quality control metric on your own data. We suggest a contig length of 1M as an upper limit.:

We examine the results and observe two key patterns: - Token weight variance from ground truth decreases with longer contigs - Token identity overlap with ground truth increases with longer contigs

However, the variance in weight and identity overlap is not extremely large, even with very short contigs. Across different contig lengths, lower weighted tokens tend to be more variable, while highly weighted tokens are more stable.

Token set identity per contig length:

Contig length	Token overlap	Percentage identity
0	10000 (control)	100.00
512	7570	75.70
1024	8131	81.31
2048	8585	85.85
4096	8849	88.49
8192	9057	90.57
16384	9196	91.96
32768	9349	93.49
65536	9422	94.22
131072	9512	95.12
262144	9593	95.93
524288	9618	96.18
1048576	9674	96.74

Due to size, only a subset of plots are shown for reference. The full plots can be generated from the above code.

Full genome length (highest weighted tokens):

fig/contig_0_high.png

Full genome length (lowest weighted tokens):

fig/contig_0_low.png

Long contigs 1048576 bp (highest weighted tokens):

fig/contig_1048576_high.png

Long contigs 1048576 bp (lowest weighted tokens):

fig/contig_1048576_low.png

Short contigs 512 bp (highest weighted tokens):

fig/contig_512_high.png

Short contigs 512 bp (lowest weighted tokens):

fig/contig_512_low.png

Pooling multiple tokenisers

If you want to pool multiple tokenisers, you can use `compare_empirical_tokens` with additional options `-m` for `inner` or `outer` merge, and `-p` for `min`, `max`, `mean`, `median` weight pooling.

5.4.2 Conventional k-mers

Note that this step also generates a dataset object in the same operation. Reverse complementing `Y/R` is supported.

Here we take a list of infile paths, and a list of matching labels. Eg `--infile_path file1.fasta file2.fasta`, then `--label 0 1`.

```
python ../src/kmerise_bio.py -h
usage: kmerise_bio.py [-h] [-i INFILE_PATH [INFILE_PATH ...]]
                      [-o OUTFILE_PATH] [-c CHUNK] [-m MAPPINGS]
                      [-t TOKENISER_PATH] [-k KMER_SIZE]
                      [-l LABEL [LABEL ...]] [--no_reverse_complement]

Take gzip fasta file(s), kmerise reads and export csv.

options:
  -h, --help            show this help message and exit
  -i INFILE_PATH [INFILE_PATH ...], --infile_path INFILE_PATH [INFILE_PATH ...]
                        path to files with biological seqs split by line
  -o OUTFILE_PATH, --outfile_path OUTFILE_PATH
                        path to output huggingface-like dataset.csv file
  -c CHUNK, --chunk CHUNK
                        split seqs into n-length blocks (DEFAULT: None)
  -m MAPPINGS, --mappings MAPPINGS
                        path to output mappings file
  -t TOKENISER_PATH, --tokeniser_path TOKENISER_PATH
                        path to tokeniser.json file to save data
  -k KMER_SIZE, --kmer_size KMER_SIZE
                        split seqs into n-length blocks (DEFAULT: None)
  -l LABEL [LABEL ...], --label LABEL [LABEL ...]
                        provide integer label for seqs (order must match
                        infile_path!)
  --no_reverse_complement
                        turn off reverse complement (DEFAULT: ON)
```

CHAPTER

SIX

CREATE A DATASET OBJECT FROM SEQUENCES

This explains the use of `create_dataset_bio.py`. We generate a `HuggingFace` dataset object given a `fasta` file containing sequences, a `fasta` file containing control sequences, and a pretrained `tokeniser` from `tokeniser.py`. The dataset can then enter the `genomicBERT` pipeline.

6.1 Source data

Any `fasta` file can be used, with each `fasta` file representing a sequence collection of one category. Sample input data files will be available in `data/`. If needed, control data can be generated with `generate_synthetic.py`. Tokeniser can be generated with `tokenise.py`.

6.2 Results

Note: Entry points are available if this is installed using the automated conda method. You can then use the command line argument directly, for example: `create_dataset_bio`. If not, you will need to use the script directly, which follows the same naming pattern, for example: `python create_dataset_bio.py`.

Running the code as below:

```
python create_dataset_bio.py <INFILE_SEQS_1> <INFILE_SEQS_2> <TOKENISER_PATH> -c CHUNK -
↪o OUTFILE_DIR
```

`HuggingFace`-like dataset files will be written to disk. This can be loaded directly into a "conventional" deep learning pipeline.

6.3 Notes

It is possible to split the dataset into chunks of n-length. This is useful when the length of individual sequences become too large to fit in memory. A sequence length of 256-512 units can effectively fit on most modern GPUs. Sequence chunks are treated as independent samples of the same class and no merging of weights is performed in this implementation. Note that `create_dataset_bio.py` and `create_dataset_nlp.py` workflows are structured differently to account for the differences in conventional biological vs human language corpora, but the processes are conceptually identical.

More information on the HuggingFace `Dataset` object is available online.

6.4 Usage

```
python create_dataset_bio.py -h
usage: create_dataset_bio.py [-h] [-o OUTFILE_DIR] [-s SPECIAL_TOKENS [SPECIAL_TOKENS ...
↪]] [-c CHUNK]
                             [--split_train SPLIT_TRAIN] [--split_test SPLIT_TEST]
                             [--split_val SPLIT_VAL] [--no_reverse_complement] [--no_
↪shuffle]
                             infile_path control_dist tokeniser_path

Take control and test fasta files, tokeniser and convert to HuggingFace dataset object.␣
↪Fasta files
can be .gz. Sequences are reverse complemented by default.

positional arguments:
  infile_path           path to fasta/gz file
  control_dist          supply control seqs
  tokeniser_path        load tokeniser file

optional arguments:
  -h, --help            show this help message and exit
  -o OUTFILE_DIR, --outfile_dir OUTFILE_DIR
                        write  dataset to directory as [ csv | json | parquet | dir/ ]␣
↪(DEFAULT:
                        "hf_out/")
  -s SPECIAL_TOKENS [SPECIAL_TOKENS ...], --special_tokens SPECIAL_TOKENS [SPECIAL_
↪TOKENS ...]
                        assign special tokens, eg space and pad tokens (DEFAULT: ["<s>",
↪"</s>",
                        "<unk>", "<pad>", "<mask>"])
  -c CHUNK, --chunk CHUNK
                        split seqs into n-length blocks (DEFAULT: None)
  --split_train SPLIT_TRAIN
                        proportion of training data (DEFAULT: 0.90)
  --split_test SPLIT_TEST
                        proportion of testing data (DEFAULT: 0.05)
  --split_val SPLIT_VAL
                        proportion of validation data (DEFAULT: 0.05)
  --no_reverse_complement
                        turn off reverse complement (DEFAULT: ON)
  --no_shuffle          turn off shuffle for data split (DEFAULT: ON)
```

CHAPTER

SEVEN

CREATE EMBEDDINGS FROM A TOKENISED DATASET

This explains the use of `create_embedding_bio_sp.py` and `create_embedding_bio_kmers.py`. Only use this if you plan to use embeddings directly.

7.1 Source data

Use `csv` files created from either `create_dataset_bio.py` or `kmerise_bio.py`.

7.2 Results

Note: Entry points are available if this is installed using the automated conda method. You can then use the command line argument directly, for example: `create_dataset_bio`. If not, you will need to use the script directly, which follows the same naming pattern, for example: `python create_dataset_bio.py`.

7.2.1 Empirical tokenisation

```
create_embedding_bio_sp.py -i [INFILE_PATH ... ] -t TOKENISER_PATH -o OUTFILE_DIR
```

7.2.2 Conventional k-mers

```
create_embedding_bio_kmers.py -i [INFILE_PATH ... ] -t TOKENISER_PATH  -o OUTFILE_DIR
```

The resulting output will be used in `embedding_pipeline.py`.

7.3 Notes

Embeddings are generated for each individual token. For example:

```
# original seq of category X
AAAAACCCCCTTTTTGGGGG

# split into tokens using desired method
[AAAAA]
[CCCCC]
...

# each token gets projected onto an embedding
[0.1 0.2 0.3 ...]
[0.3 0.4 0.5 ...]
...
```

7.4 Usage

7.4.1 Empirical tokenisation

```
python create_embedding_bio_sp.py -h
usage: create_embedding_bio_sp.py [-h] [-i INFILE_PATH [INFILE_PATH ...]]
                                  [-o OUTPUT_DIR] [-c COLUMN_NAMES]
                                  [-l LABELS] [-x COLUMN_NAME] [-m MODEL]
                                  [-t TOKENISER_PATH]
                                  [-s SPECIAL_TOKENS [SPECIAL_TOKENS ...]]
                                  [-n NJOBS] [--w2v_min_count W2V_MIN_COUNT]
                                  [--w2v_sg W2V_SG]
                                  [--w2v_vector_size W2V_VECTOR_SIZE]
                                  [--w2v_window W2V_WINDOW]
                                  [--no_reverse_complement]
                                  [--sample_seq SAMPLE_SEQ]

Take fasta files, tokeniser and generate embedding. Fasta files can be .gz.
Sequences are reverse complemented by default.

options:
  -h, --help            show this help message and exit
  -i INFILE_PATH [INFILE_PATH ...], --infile_path INFILE_PATH [INFILE_PATH ...]
                        path to fasta/gz file
  -o OUTPUT_DIR, --output_dir OUTPUT_DIR
                        write embeddings to disk (DEFAULT: "embed/")
  -c COLUMN_NAMES, --column_names COLUMN_NAMES
                        column name for sp tokenised data (DEFAULT:
                        input_str)
  -l LABELS, --labels LABELS
                        column name for data labels (DEFAULT: labels)
  -x COLUMN_NAME, --column_name COLUMN_NAME
                        column name for extracting embeddings (DEFAULT:
```

(continues on next page)

(continued from previous page)

```
                        input_str)
  -m MODEL, --model MODEL
                        load existing model (DEFAULT: None)
  -t TOKENISER_PATH, --tokeniser_path TOKENISER_PATH
                        load tokenised data
  -s SPECIAL_TOKENS [SPECIAL_TOKENS ...], --special_tokens SPECIAL_TOKENS [SPECIAL_
↪TOKENS ...]
                        assign special tokens, eg space and pad tokens
                        (DEFAULT: ["<s>", "</s>", "<unk>", "<pad>",
                        "<mask>"])
  -n NJOBS, --njobs NJOBS
                        set number of threads to use
  --w2v_min_count W2V_MIN_COUNT
                        set minimum count for w2v (DEFAULT: 1)
  --w2v_sg W2V_SG       0 for bag-of-words, 1 for skip-gram (DEFAULT: 1)
  --w2v_vector_size W2V_VECTOR_SIZE
                        set w2v matrix dimensions (DEFAULT: 100)
  --w2v_window W2V_WINDOW
                        set context window size for w2v (DEFAULT: -/+10)
  --no_reverse_complement
                        turn off reverse complement (DEFAULT: ON)
  --sample_seq SAMPLE_SEQ
                        project sample sequence on embedding (DEFAULT: None)
```

7.4.2 Conventional k-mers

```
python create_embedding_bio_kmers.py -h
usage: create_embedding_bio_kmers.py [-h] [-i INFILE_PATH [INFILE_PATH ...]]
                                     [-o OUTPUT_DIR] [-m MODEL] [-k KSIZE]
                                     [-w SLIDE] [-c CHUNK] [-n NJOBS]
                                     [-s SAMPLE_SEQ] [-v VOCAB_SIZE]
                                     [--w2v_min_count W2V_MIN_COUNT]
                                     [--w2v_sg W2V_SG]
                                     [--w2v_vector_size W2V_VECTOR_SIZE]
                                     [--w2v_window W2V_WINDOW]
                                     [--no_reverse_complement]

Take tokenised data, parameters and generate embedding. Note that this takes
output of kmerise_bio.py, and NOT raw fasta files.

options:
  -h, --help            show this help message and exit
  -i INFILE_PATH [INFILE_PATH ...], --infile_path INFILE_PATH [INFILE_PATH ...]
                        path to input tokenised data file
  -o OUTPUT_DIR, --output_dir OUTPUT_DIR
                        write embeddings to disk (DEFAULT: "embed/")
  -m MODEL, --model MODEL
                        load existing model (DEFAULT: None)
  -k KSIZE, --ksize KSIZE
                        set size of k-mers
```

(continues on next page)

(continued from previous page)

```
  -w SLIDE, --slide SLIDE
                        set length of sliding window on k-mers (min 1)
  -c CHUNK, --chunk CHUNK
                        split seqs into n-length blocks (DEFAULT: None)
  -n NJOBS, --njobs NJOBS
                        set number of threads to use
  -s SAMPLE_SEQ, --sample_seq SAMPLE_SEQ
                        set sample sequence to test model (DEFAULT: None)
  -v VOCAB_SIZE, --vocab_size VOCAB_SIZE
                        vocabulary size for model config (DEFAULT: all)
  --w2v_min_count W2V_MIN_COUNT
                        set minimum count for w2v (DEFAULT: 1)
  --w2v_sg W2V_SG       0 for bag-of-words, 1 for skip-gram (DEFAULT: 1)
  --w2v_vector_size W2V_VECTOR_SIZE
                        set w2v matrix dimensions (DEFAULT: 100)
  --w2v_window W2V_WINDOW
                        set context window size for w2v (DEFAULT: -/+10)
  --no_reverse_complement
                        turn off reverse complement (DEFAULT: ON)
```

CHAPTER

EIGHT

PERFORM A HYPERPARAMETER SWEEP

This explains the use of `sweep.py` for machine and deep learning through `genomicBERT`. If you already know what hyperparameters are needed, you can use `train_model.py`. For conventional machine learning, the sweep, train and cross validation steps are combined in one operation.

8.1 Source data

Source data is a HuggingFace `dataset` object as a `csv`, `json` or `parquet` file. Specify `--format` accordingly. `csv` only for non-deep learning.

8.2 Results

Note: Entry points are available if this is installed using the automated conda method. You can then use the command line argument directly, for example: `create_dataset_bio`. If not, you will need to use the script directly, which follows the same naming pattern, for example: `python create_dataset_bio.py`.

Running the code as below:

8.2.1 Deep learning

```
python sweep.py <TRAIN_DATA> <FORMAT> <TOKENISER_PATH> --test TEST_DATA --valid␣
↪VALIDATION_DATA --hyperparameter_sweep PARAMS.JSON --entity_name WANDB_ENTITY_NAME --
↪project_name WANDB_PROJECT_NAME --group_name WANDB_GROUP_NAME --sweep_count N --metric_
↪opt [ eval/accuracy | eval/validation | eval/loss | eval/precision | eval/recall ] --
↪output_dir OUTPUT_DIR
```

8.2.2 Frequency-based approaches

```
python freq_pipeline.py -i [INFILE_PATH ... ] --format "csv" -t TOKENISER_PATH --freq_
↪method [ cvec | tfidf ] --model [ rf | xg ] --kfolds N --sweep_count N --metric_opt [␣
↪accuracy | f1 | precision | recall | roc_auc ] --output_dir OUTPUT_DIR
```

8.2.3 Embedding

```
python embedding_pipeline.py -i [INFILE_PATH ... ] --format "csv" -t TOKENISER_PATH --
↪freq_method [ cvec | tfidf ] --model [ rf | xg ] --kfolds N --sweep_count N --metric_
↪opt [ accuracy | f1 | precision | recall | roc_auc ] --output_dir OUTPUT_DIR
```

8.3 Notes

The original documentation to specify training arguments is available here.

8.4 Usage

8.4.1 genomicBERT: Deep learning

Sweep parameters and search space should be passed in as a `json` file.

```
{
  "name" : "random",
  "method" : "random",
  "metric": {
    "name": "eval/f1",
    "goal": "maximize"
  },
  "parameters" : {
    "epochs" : {
      "values" : [1, 2, 3]
    },
    "batch_size": {
        "values": [8, 16, 32, 64]
        },
    "learning_rate" :{
      "distribution": "log_uniform_values",
      "min": 0.0001,
      "max": 0.1
      },
    "weight_decay": {
        "values": [0.0, 0.1, 0.2, 0.3, 0.4, 0.5]
      }
  },
  "early_terminate": {
      "type": "hyperband",
```

(continues on next page)

(continued from previous page)

```
      "s": 2,
      "eta": 3,
      "max_iter": 27
  }
}
```

```
usage: sweep.py [-h] [-t TEST] [-v VALID] [-m MODEL]
                [--model_features MODEL_FEATURES] [-o OUTPUT_DIR] [-d DEVICE]
                [-s VOCAB_SIZE] [-w HYPERPARAMETER_SWEEP]
                [-l LABEL_NAMES [LABEL_NAMES ...]] [-n SWEEP_COUNT]
                [-e ENTITY_NAME] [-p PROJECT_NAME] [-g GROUP_NAME]
                [-c METRIC_OPT] [-r RESUME_SWEEP] [--fp16_off] [--wandb_off]
                train format tokeniser_path

Take HuggingFace dataset and perform parameter sweeping.

positional arguments:
  train                 path to [ csv | csv.gz | json | parquet ] file
  format                specify input file type [ csv | json | parquet ]
  tokeniser_path        path to tokeniser.json file to load data from

options:
  -h, --help            show this help message and exit
  -t TEST, --test TEST  path to [ csv | csv.gz | json | parquet ] file
  -v VALID, --valid VALID
                        path to [ csv | csv.gz | json | parquet ] file
  -m MODEL, --model MODEL
                        choose model [ distilbert | longformer ] distilbert
                        handles shorter sequences up to 512 tokens longformer
                        handles longer sequences up to 4096 tokens (DEFAULT:
                        distilbert)
  --model_features MODEL_FEATURES
                        number of features in data to use (DEFAULT: ALL)
                        NOTE: this is separate from the vocab_size argument.
                        under normal circumstances (eg a tokeniser generated
                        by tokenise_bio), setting this is not necessary
  -o OUTPUT_DIR, --output_dir OUTPUT_DIR
                        specify path for output (DEFAULT: ./sweep_out)
  -d DEVICE, --device DEVICE
                        choose device [ cpu | cuda:0 ] (DEFAULT: detect)
  -s VOCAB_SIZE, --vocab_size VOCAB_SIZE
                        vocabulary size for model configuration
  -w HYPERPARAMETER_SWEEP, --hyperparameter_sweep HYPERPARAMETER_SWEEP
                        run a hyperparameter sweep with config from file
  -l LABEL_NAMES [LABEL_NAMES ...], --label_names LABEL_NAMES [LABEL_NAMES ...]
                        provide column with label names (DEFAULT: "").
  -n SWEEP_COUNT, --sweep_count SWEEP_COUNT
                        run n hyperparameter sweeps (DEFAULT: 64)
  -e ENTITY_NAME, --entity_name ENTITY_NAME
                        provide wandb team name (if available).
  -p PROJECT_NAME, --project_name PROJECT_NAME
                        provide wandb project name (if available).
```

(continues on next page)

(continued from previous page)

```
  -g GROUP_NAME, --group_name GROUP_NAME
                        provide wandb group name (if desired).
  METRIC_OPT, --metric_opt METRIC_OPT
                        score to maximise [ eval/accuracy | eval/validation |
                        eval/loss | eval/precision | eval/recall ] (DEFAULT:
                        eval/f1)
  -r RESUME_SWEEP, --resume_sweep RESUME_SWEEP
                        provide sweep id to resume sweep.
  --fp16_off            turn fp16 off for precision / cpu (DEFAULT: ON)
  --wandb_off           run hyperparameter tuning using the wandb api and log
                        training in real time online (DEFAULT: ON)
```

8.4.2 Frequency based approach

```
python freq_pipeline.py -h
usage: freq_pipeline.py [-h] [--infile_path INFILE_PATH [INFILE_PATH ...]]
                        [--format FORMAT] [--embeddings EMBEDDINGS]
                        [--chunk_size CHUNK_SIZE] [-t TOKENISER_PATH]
                        [-f FREQ_METHOD] [--column_names COLUMN_NAMES]
                        [--column_name COLUMN_NAME] [-m MODEL]
                        [-e MODEL_FEATURES] [-k KFOLDS]
                        [--ngram_from NGRAM_FROM] [--ngram_to NGRAM_TO]
                        [--split_train SPLIT_TRAIN] [--split_test SPLIT_TEST]
                        [--split_val SPLIT_VAL] [-o OUTPUT_DIR]
                        [-s VOCAB_SIZE]
                        [--special_tokens SPECIAL_TOKENS [SPECIAL_TOKENS ...]]
                        [-w HYPERPARAMETER_SWEEP]
                        [--sweep_method SWEEP_METHOD] [-n SWEEP_COUNT]
                        [-c METRIC_OPT] [-j NJOBS] [-d PRE_DISPATCH]

Take HuggingFace dataset and perform parameter sweeping.

options:
  -h, --help            show this help message and exit
  --infile_path INFILE_PATH [INFILE_PATH ...]
                        path to [ csv | csv.gz | json | parquet ] file
  --format FORMAT       specify input file type [ csv | json | parquet ]
  --embeddings EMBEDDINGS
                        path to embeddings model file
  --chunk_size CHUNK_SIZE
                        iterate over input file for these many rows
  -t TOKENISER_PATH, --tokeniser_path TOKENISER_PATH
                        path to tokeniser.json file to load data from
  -f FREQ_METHOD, --freq_method FREQ_METHOD
                        choose dist [ cvec | tfidf ] (DEFAULT: tfidf)
  --column_names COLUMN_NAMES
                        column name for sp tokenised data (DEFAULT:
                        input_str)
  --column_name COLUMN_NAME
                        column name for extracting embeddings (DEFAULT:
```

(continues on next page)

(continued from previous page)

```
                        input_str)
  -m MODEL, --model MODEL
                        choose model [ rf | xg ] (DEFAULT: rf)
  -e MODEL_FEATURES, --model_features MODEL_FEATURES
                        number of features in data to use (DEFAULT: ALL)
  -k KFOLDS, --kfolds KFOLDS
                        number of cross validation folds (DEFAULT: 8)
  --ngram_from NGRAM_FROM
                        ngram slice starting index (DEFAULT: 1)
  --ngram_to NGRAM_TO   ngram slice ending index (DEFAULT: 1)
  --split_train SPLIT_TRAIN
                        proportion of training data (DEFAULT: 0.90)
  --split_test SPLIT_TEST
                        proportion of testing data (DEFAULT: 0.05)
  --split_val SPLIT_VAL
                        proportion of validation data (DEFAULT: 0.05)
  -o OUTPUT_DIR, --output_dir OUTPUT_DIR
                        specify path for output (DEFAULT: ./results_out)
  -s VOCAB_SIZE, --vocab_size VOCAB_SIZE
                        vocabulary size for model configuration
  --special_tokens SPECIAL_TOKENS [SPECIAL_TOKENS ...]
                        assign special tokens, eg space and pad tokens
                        (DEFAULT: ["<s>", "</s>", "<unk>", "<pad>",
                        "<mask>"])
  -w HYPERPARAMETER_SWEEP, --hyperparameter_sweep HYPERPARAMETER_SWEEP
                        run a hyperparameter sweep with config from file
  --sweep_method SWEEP_METHOD
                        specify sweep search strategy [ bayes | grid | random
                        ] (DEFAULT: random)
  -n SWEEP_COUNT, --sweep_count SWEEP_COUNT
                        run n hyperparameter sweeps (DEFAULT: 8)
  -c METRIC_OPT, --metric_opt METRIC_OPT
                        score to maximise [ accuracy | f1 | precision |
                        recall ] (DEFAULT: f1)
  -j NJOBS, --njobs NJOBS
                        run on n threads (DEFAULT: -1)
  -d PRE_DISPATCH, --pre_dispatch PRE_DISPATCH
                        specify dispatched jobs (DEFAULT: 0.5*n_jobs)
```

8.4.3 Embedding based approach

```
python embedding_pipeline.py -h
usage: embedding_pipeline.py [-h]
                             [--infile_path INFILE_PATH [INFILE_PATH ...]]
                             [--format FORMAT] [--embeddings EMBEDDINGS]
                             [--chunk_size CHUNK_SIZE] [-t TOKENISER_PATH]
                             [-f FREQ_METHOD] [--column_names COLUMN_NAMES]
                             [--column_name COLUMN_NAME] [-m MODEL]
                             [-e MODEL_FEATURES] [-k KFOLDS]
                             [--ngram_from NGRAM_FROM] [--ngram_to NGRAM_TO]
```

(continues on next page)

(continued from previous page)

```
                              [--split_train SPLIT_TRAIN]
                              [--split_test SPLIT_TEST]
                              [--split_val SPLIT_VAL] [-o OUTPUT_DIR]
                              [-s VOCAB_SIZE]
                              [--special_tokens SPECIAL_TOKENS [SPECIAL_TOKENS ...]]
                              [-w HYPERPARAMETER_SWEEP]
                              [--sweep_method SWEEP_METHOD] [-n SWEEP_COUNT]
                              [-c METRIC_OPT] [-j NJOBS] [-d PRE_DISPATCH]

Take HuggingFace dataset and perform parameter sweeping.

options:
  -h, --help            show this help message and exit
  --infile_path INFILE_PATH [INFILE_PATH ...]
                        path to [ csv | csv.gz | json | parquet ] file
  --format FORMAT       specify input file type [ csv | json | parquet ]
  --embeddings EMBEDDINGS
                        path to embeddings model file
  --chunk_size CHUNK_SIZE
                        iterate over input file for these many rows
  -t TOKENISER_PATH, --tokeniser_path TOKENISER_PATH
                        path to tokeniser.json file to load data from
  -f FREQ_METHOD, --freq_method FREQ_METHOD
                        choose dist [ embed ] (DEFAULT: embed)
  --column_names COLUMN_NAMES
                        column name for sp tokenised data (DEFAULT:
                        input_str)
  --column_name COLUMN_NAME
                        column name for extracting embeddings (DEFAULT:
                        input_str)
  -m MODEL, --model MODEL
                        choose model [ rf | xg ] (DEFAULT: rf)
  -e MODEL_FEATURES, --model_features MODEL_FEATURES
                        number of features in data to use (DEFAULT: ALL)
  -k KFOLDS, --kfolds KFOLDS
                        number of cross validation folds (DEFAULT: 8)
  --ngram_from NGRAM_FROM
                        ngram slice starting index (DEFAULT: 1)
  --ngram_to NGRAM_TO   ngram slice ending index (DEFAULT: 1)
  --split_train SPLIT_TRAIN
                        proportion of training data (DEFAULT: 0.90)
  --split_test SPLIT_TEST
                        proportion of testing data (DEFAULT: 0.05)
  --split_val SPLIT_VAL
                        proportion of validation data (DEFAULT: 0.05)
  -o OUTPUT_DIR, --output_dir OUTPUT_DIR
                        specify path for output (DEFAULT: ./results_out)
  -s VOCAB_SIZE, --vocab_size VOCAB_SIZE
                        vocabulary size for model configuration
  --special_tokens SPECIAL_TOKENS [SPECIAL_TOKENS ...]
                        assign special tokens, eg space and pad tokens
                        (DEFAULT: ["<s>", "</s>", "<unk>", "<pad>",
```

(continues on next page)

(continued from previous page)

```
                        "<mask>"])
  -w HYPERPARAMETER_SWEEP, --hyperparameter_sweep HYPERPARAMETER_SWEEP
                        run a hyperparameter sweep with config from file
  --sweep_method SWEEP_METHOD
                        specify sweep search strategy [ bayes | grid | random
                        ] (DEFAULT: random)
  -n SWEEP_COUNT, --sweep_count SWEEP_COUNT
                        run n hyperparameter sweeps (DEFAULT: 8)
  -c METRIC_OPT, --metric_opt METRIC_OPT
                        score to maximise [ accuracy | f1 | precision |
                        recall ] (DEFAULT: f1)
  -j NJOBS, --njobs NJOBS
                        run on n threads (DEFAULT: -1)
  -d PRE_DISPATCH, --pre_dispatch PRE_DISPATCH
                        specify dispatched jobs (DEFAULT: 0.5*n_jobs)
```

CHAPTER

NINE

GENOMICBERT: TRAIN A DEEP LEARNING CLASSIFIER

This explains the use of `train.py`. Use this if you already know what hyperparameters are needed. Otherwise use `sweep.py`. For conventional machine learning, the sweep, train and cross validation steps are combined in one operation.

9.1 Source data

Source data is a HuggingFace `dataset` object as a `csv`, `json` or `parquet` file. Specify `--format` accordingly.

9.2 Results

Note: Entry points are available if this is installed using the automated conda method. You can then use the command line argument directly, for example: `create_dataset_bio`. If not, you will need to use the script directly, which follows the same naming pattern, for example: `python create_dataset_bio.py`.

Running the code as below:

```
python train_model.py <TRAIN_DATA> <FORMAT> <TOKENISER_PATH> --test TEST_DATA --valid␣
↪VALIDATION_DATA --hyperparameter_file PARAMS.JSON --entity_name WANDB_ENTITY_NAME --
↪project_name WANDB_PROJECT_NAME --group_name WANDB_GROUP_NAME --sweep_count N --metric_
↪opt [ eval/accuracy | eval/validation | eval/loss | eval/precision | eval/recall ] --
↪output_dir OUTPUT_DIR --label_names labels
```

Note: Remember to provide the `--label_names` argument! This is `labels` by default (if this wasn't changed in any previous part of the pipeline).

You will obtain a `json` file with weights for each token. Any special tokens you add will also be present. This will be used in the next step of creating a `HuggingFace` compatible dataset object.

9.3 Notes

The original documentation to specify training arguments is available here.

9.4 Usage

The full list of arguments is truncated, and only arguments added by this package are shown. These are available on the corresponding HuggingFace `transformers.TrainingArguments` documentation shown above.

```
python train.py -h

Take HuggingFace dataset and train. Arguments match that of
TrainingArguments, with the addition of [ train, test, valid, tokeniser_path,
vocab_size, model, device, entity_name, project_name, group_name,
config_from_run, metric_opt, hyperparameter_file, no_shuffle, wandb_off,
override_output_dir ]. See: https://huggingface.co/docs/transformers/v4.19.4/
en/main_classes/trainer#transformers.TrainingArguments

positional arguments:
  train                 path to [ csv | csv.gz | json | parquet ] file
  format                specify input file type [ csv | json | parquet ]
  tokeniser_path        path to tokeniser.json file to load data from

options:
  -h, --help            show this help message and exit
  --output_dir OUTPUT_DIR
                        The output directory where the model predictions and
                        checkpoints will be written. (default: None)
  --overwrite_output_dir [OVERWRITE_OUTPUT_DIR]
                        Overwrite the content of the output directory. Use
                        this to continue training if output_dir points to a
                        checkpoint directory. (default: False)
  -t TEST, --test TEST  path to [ csv | csv.gz | json | parquet ] file
                        (default: None)
  -v VALID, --valid VALID
                        path to [ csv | csv.gz | json | parquet ] file
                        (default: None)
  -m MODEL, --model MODEL
                        choose model [ distilbert | longformer ] distilbert
                        handles shorter sequences up to 512 tokens longformer
                        handles longer sequences up to 4096 tokens (DEFAULT:
                        distilbert) (default: distilbert)
  -d DEVICE, --device DEVICE
                        choose device [ cpu | cuda:0 ] (DEFAULT: detect)
                        (default: None)
  -s VOCAB_SIZE, --vocab_size VOCAB_SIZE
                        vocabulary size for model configuration (default:
                        32000)
  -f HYPERPARAMETER_FILE, --hyperparameter_file HYPERPARAMETER_FILE
                        provide torch.bin or json file of hyperparameters.
                        NOTE: if given, this overrides all
```

(continues on next page)

(continued from previous page)

```
                        HfTrainingArguments! This is overridden by
                        --config_from_run! (default: )
  -e ENTITY_NAME, --entity_name ENTITY_NAME
                        provide wandb team name (if available). NOTE: has no
                        effect if wandb is disabled. (default: )
  -p PROJECT_NAME, --project_name PROJECT_NAME
                        provide wandb project name (if available). NOTE: has
                        no effect if wandb is disabled. (default: )
  -g GROUP_NAME, --group_name GROUP_NAME
                        provide wandb group name (if desired). (default:
                        train)
  -c CONFIG_FROM_RUN, --config_from_run CONFIG_FROM_RUN
                        load arguments from existing wandb run. NOTE: if
                        given, this overrides --hyperparameter_file!
                        (default: None)
  METRIC_OPT, --metric_opt METRIC_OPT
                        score to maximise [ eval/accuracy | eval/validation |
                        eval/loss | eval/precision | eval/recall ] (DEFAULT:
                        eval/f1) (default: eval/f1)
  --override_output_dir
                        override output directory (DEFAULT: OFF) (default:
                        False)
  --no_shuffle          turn off random shuffling (DEFAULT: SHUFFLE)
                        (default: True)
  --wandb_off           log training in real time online (DEFAULT: ON)
                        (default: True)

  [ADDITIONAL ARGUMENTS TRUNCATED]
```

CHAPTER

TEN

PERFORM CROSS-VALIDATION

This explains the use of `cross_validate.py` for deep learning through the `genomicBERT` pipeline. For conventional machine learning, the sweep, train and cross validation steps are combined in one operation.

10.1 Source data

Source data is a HuggingFace `dataset` object as a `csv`, `json` or `parquet` file. Specify `--format` accordingly. `csv` only for non-deep learning.

10.2 Results

Note: Entry points are available if this is installed using the automated conda method. You can then use the command line argument directly, for example: `create_dataset_bio`. If not, you will need to use the script directly, which follows the same naming pattern, for example: `python create_dataset_bio.py`.

Running the code as below:

10.2.1 Deep learning

Specify the same data, wandb project, entity and group names as used for sweeping or training. Once the best run is identified by the user, passing the run id into `--config_from_run` will automatically load config of the best run from `wandb`.

```
# use the WANDB_ENTITY_NAME, WANDB_PROJECT_NAME and the best run id corresponding to the␣
↪sweep
# WANDB_GROUP_NAME should be changed to reflect the new category of runs (eg "cval")
python cross_validate.py <TRAIN_DATA> <FORMAT> --test TEST_DATA --valid VALIDATION_DATA -
↪-entity_name WANDB_ENTITY_NAME --project_name WANDB_PROJECT_NAME --group_name WANDB_
↪GROUP_NAME --kfolds N --config_from_run WANDB_RUN_ID --output_dir OUTPUT_DIR
```

10.2.2 Frequency-based approaches

Cross-validation is carried out within the main pipeline:

```
python freq_pipeline.py -i [INFILE_PATH ... ] --format "csv" -t TOKENISER_PATH --freq_
↪method [ cvec | tfidf ] --model [ rf | xg ] --kfolds N --sweep_count N --metric_opt [␣
↪accuracy | f1 | precision | recall | roc_auc ] --output_dir OUTPUT_DIR
```

10.2.3 Embedding

Cross-validation is carried out within the main pipeline:

```
python embedding_pipeline.py -i [INFILE_PATH ... ] --format "csv" -t TOKENISER_PATH --
↪freq_method [ cvec | tfidf ] --model [ rf | xg ] --kfolds N --sweep_count N --metric_
↪opt [ accuracy | f1 | precision | recall | roc_auc ] --output_dir OUTPUT_DIR
```

10.3 Notes

The original documentation to specify training arguments is available here.

10.4 Usage

10.4.1 Deep learning

Sweep parameters and search space should be passed in as a `json` file.

```
python ../src/cross_validate.py -h
usage: cross_validate.py [-h] [--tokeniser_path TOKENISER_PATH] [-t TEST] [-v VALID] [-m␣
↪MODEL_PATH] [-o OUTPUT_DIR]
                         [-d DEVICE] [-s VOCAB_SIZE] [-f HYPERPARAMETER_FILE] [-l LABEL_
↪NAMES [LABEL_NAMES ...]]
                         [-k KFOLDS] [-e ENTITY_NAME] [-g GROUP_NAME] [-p PROJECT_NAME] [-
↪c CONFIG_FROM_RUN]
                         [-o METRIC_OPT] [--overwrite_output_dir] [--no_shuffle] [--wandb_
↪off]
                         train format

Take HuggingFace dataset and perform cross validation.

positional arguments:
  train                 path to [ csv | csv.gz | json | parquet ] file
  format                specify input file type [ csv | json | parquet ]

optional arguments:
  -h, --help            show this help message and exit
  --tokeniser_path TOKENISER_PATH
                        path to tokeniser.json file to load data from
  -t TEST, --test TEST  path to [ csv | csv.gz | json | parquet ] file
```

(continues on next page)

(continued from previous page)

```
  -v VALID, --valid VALID
                        path to [ csv | csv.gz | json | parquet ] file
  -m MODEL_PATH, --model_path MODEL_PATH
                        path to pretrained model dir. this should contain files such as␣
↪[ pytorch_model.bin,
                        config.yaml, tokeniser.json, etc ]
  -o OUTPUT_DIR, --output_dir OUTPUT_DIR
                        specify path for output (DEFAULT: ./cval_out)
  -d DEVICE, --device DEVICE
                        choose device [ cpu | cuda:0 ] (DEFAULT: detect)
  -s VOCAB_SIZE, --vocab_size VOCAB_SIZE
                        vocabulary size for model configuration
  -f HYPERPARAMETER_FILE, --hyperparameter_file HYPERPARAMETER_FILE
                        provide torch.bin or json file of hyperparameters. NOTE: if␣
↪given, this overrides all
                        HfTrainingArguments! This is overridden by --config_from_run!
  -l LABEL_NAMES [LABEL_NAMES ...], --label_names LABEL_NAMES [LABEL_NAMES ...]
                        provide column with label names (DEFAULT: "").
  -k KFOLDS, --kfolds KFOLDS
                        run n number of kfolds (DEFAULT: 8)
  -e ENTITY_NAME, --entity_name ENTITY_NAME
                        provide wandb team name (if available).
  -g GROUP_NAME, --group_name GROUP_NAME
                        provide wandb group name (if desired).
  -p PROJECT_NAME, --project_name PROJECT_NAME
                        provide wandb project name (if available).
  -c CONFIG_FROM_RUN, --config_from_run CONFIG_FROM_RUN
                        load arguments from existing wandb run. NOTE: if given, this␣
↪overrides --hyperparameter_file!
  METRIC_OPT, --metric_opt METRIC_OPT
                        score to maximise [ eval/accuracy | eval/validation | eval/loss␣
↪| eval/precision |
                        eval/recall ] (DEFAULT: eval/f1)
  --overwrite_output_dir
                        override output directory (DEFAULT: OFF)
  --no_shuffle          turn off random shuffling (DEFAULT: SHUFFLE)
  --wandb_off           run hyperparameter tuning using the wandb api and log training␣
↪in real time online (DEFAULT:
                        ON)
```

Note: If using the `--config_from_run` option, note that this inherits the original output directory paths. Make sure you specify a new `--output_dir` and enable the `--overwrite_output_dir` flag. This also inherits the device specifications (gpu or cpu).

CHAPTER

ELEVEN

COMPARE PERFORMANCE OF DIFFERENT DEEP LEARNING MODELS

This explains the use of `fit_powerlaw.py`. Only works on deep learning models through the `genomicBERT` pipeline. For more information on the method, including interpretation, please refer to the publication (https://arxiv.org/pdf/2202.02842.pdf).

11.1 Source data

Directories containing trained models from a standard `huggingface` or `pytorch` workflow can be passed in as input.

11.2 Results

Note: Entry points are available if this is installed using the automated conda method. You can then use the command line argument directly, for example: `create_dataset_bio`. If not, you will need to use the script directly, which follows the same naming pattern, for example: `python create_dataset_bio.py`.

Running the code as below:

```
python fit_powerlaw.py -i [ INFILE_PATH ... ] -t OUTPUT_DIR -a N
```

Plots will be output to the directory. A combined plot with all performance overlaid and individual performances will be generated.

11.3 Notes

Interpreting the plots may not be straightforward. Please refer to the publication for more information (https://arxiv.org/pdf/2202.02842.pdf).

11.4 Usage

```
python fit_powerlaw.py -h
usage: fit_powerlaw.py [-h] [-m MODEL_PATH [MODEL_PATH ...]] [-o OUTPUT_DIR]
                       [-a ALPHA_MAX]

Take trained model dataset and apply power law fit. Acts as a performance
metric which is independent of data. For more information refer here:
https://arxiv.org/pdf/2202.02842.pdf

optional arguments:
  -h, --help            show this help message and exit
  -m MODEL_PATH [MODEL_PATH ...], --model_path MODEL_PATH [MODEL_PATH ...]
                        path to trained model directory
  -o OUTPUT_DIR, --output_dir OUTPUT_DIR
                        path to output metrics directory (DEFAULT: same as
                        model_path)
  -a ALPHA_MAX, --alpha_max ALPHA_MAX
                        max alpha value to plot (DEFAULT: 8)
```

Note: If you are intending to download a model and the directory path matches the one on your disk, you will need to rename or remove it since it will first use local files as a priority!

CHAPTER

TWELVE

GENERATE SYNTHETIC SEQUENCES FOR USE IN CLASSIFICATION

This explains the use of `generate_synthetic.py`. Generates synthetic sequences given a `fasta` file.

12.1 Source data

Any `fasta` file can be used.

12.2 Results

Note: Entry points are available if this is installed using the automated conda method. You can then use the command line argument directly, for example: `create_dataset_bio`. If not, you will need to use the script directly, which follows the same naming pattern, for example: `python create_dataset_bio.py`.

Running the code as below:

```
python generate_synthetic.py \
  path/to/infile.fa \
  -o path/to/outfile.fa
```

You will obtain a `fasta` file with synthetic sequences generated according to your settings. By default, dinucleotide frequency is calculated **for each sequence** and used to generate a corresponding null sequence. Reverse complement is possible if needed. This can be used in two-step classification in cases where you do not have a control set.

12.3 Notes

The input file can be provided in `gzip` format. However, output will be a plain `text` file as sequences are read and written line by line.

12.4 Usage

```
python generate_synthetic.py -h
usage: generate_synthetic.py [-h] [-b BLOCK_SIZE] [-c CONTROL_DIST] [-o OUTFILE]
                             [--do_reverse_complement]
                             infile_path

Take fasta files, generate synthetic sequences. Accepts .gz files.

positional arguments:
  infile_path           path to fasta/gz file

options:
  -h, --help            show this help message and exit
  -b BLOCK_SIZE, --block_size BLOCK_SIZE
                        size of block to generate synthetic sequences from as
                        negative control (DEFAULT: 2)
  -c CONTROL_DIST, --control_dist CONTROL_DIST
                        generate control distribution by [ bootstrap | frequency
                        | /path/to/file ] (DEFAULT: frequency)
  -o OUTFILE, --outfile OUTFILE
                        write synthetic sequences (DEFAULT: "out.fa")
  --do_reverse_complement
                        turn on reverse complement (DEFAULT: OFF)
```

CHAPTER

THIRTEEN

GET CLASS ATTRIBUTION FOR DEEP LEARNING MODELS

This explains the use of `interpret.py` for deep learning through `genomicBERT`.

13.1 Source data

Source data is a path to a trained `pytorch` classifier model directory OR a `wandb` run.

13.2 Results

Note: Entry points are available if this is installed using the automated conda method. You can then use the command line argument directly, for example: `create_dataset_bio`. If not, you will need to use the script directly, which follows the same naming pattern, for example: `python create_dataset_bio.py`.

Running the code as below:

13.2.1 Deep learning

Input sequences can be provided as multiple strings and/or fasta files. If a string is provided, the file name will be the first 16 characters of the string followed by a unique string. If a fasta file is provided, the file name(s) will be the fasta header. Label names must be sorted in the order of labels, eg category 1, category 2.

```
python interpret.py <MODEL_PATH> <INPUT_SEQS ...> [TOKENISER_PATH] [OUTPUT_DIR] [LABEL_
↪NAMES ...]
```

13.3 Notes

More information on transformers interpretability is available here.

13.4 Usage

13.4.1 genomicBERT: Deep learning

Sequences to test for class attribution can be provided directly or as fasta files.

```
python interpret.py -h
usage: interpret.py [-h] [-t TOKENISER_PATH] [-o OUTPUT_DIR] [-l LABEL_NAMES [LABEL_
↪NAMES ...]]
                    model_path input_seqs [input_seqs ...]

Take complete classifier and calculate feature attributions.

positional arguments:
    model_path            path to local model directory OR wandb run
    input_seqs            input sequence(s) directly and/or fasta files

optional arguments:
    -h, --help            show this help message and exit
    -t TOKENISER_PATH, --tokeniser_path TOKENISER_PATH
                            path to tokeniser.json file to load data from
    -o OUTPUT_DIR, --output_dir OUTPUT_DIR
                            specify path for output (DEFAULT: ./interpret_out)
    -l LABEL_NAMES [LABEL_NAMES ...], --label_names LABEL_NAMES [LABEL_NAMES ...]
                            provide label names matching order (DEFAULT: None).
```

CHAPTER

FOURTEEN

INDICES AND TABLES

- genindex
- modindex
- search