Bootleg

Release v1.1.0dev1

Laurel Orr

May 28, 2022

GETTING STARTED

Bootleg is a named entity disambiguation (NED) system that links mentions in text to entities and produces contextual entity embeddings.

Bootleg is still *actively under development*, so feedback and contributions are welcome. Submit bugs in the Issues section or feel free to submit your contributions as a pull request.

CHAPTER

ONE

INSTALL

Bootleg requires Python 3.6 or later:

```
git clone git@github.com:HazyResearch/bootleg bootleg
cd bootleg
python3 setup.py install
```

Note: You will need at least 40 GB of disk space, 12 GB of GPU memory, and 35 GB of CPU memory to run our model.

CHAPTER

TWO

QUICKSTART

Getting started is easy. Run the following. This will download our default model.

Note: You will need at least 40 GB of disk space, 12 GB of GPU memory, and 35 GB of CPU memory to run our model. When running for the first time, it will take 10 plus minutes for everything to download and load correctly, depending on network speeds.

```
from bootleg.end2end.bootleg_annotator import BootlegAnnotator
ann = BootlegAnnotator()
ann.label_mentions("How many people are in Lincoln")["titles"]
```

You can also pass in multiple sentences:

```
ann.label_mentions(["I am in Lincoln", "I am Lincoln", "I am driving a Lincoln"])["titles
↪"]
```

Or, you can decide to use a different model (the choices are bootleg_cased, bootleg_uncased, bootleg_cased_mini, and bootleg_uncased_mini - default is bootleg_uncased):

```
ann = BootlegAnnotator(model_name="bootleg_uncased")
ann.label_mentions("How many people are in Lincoln")["titles"]
```

Other initialization parameters are at bootleg/end2end/bootleg_annotator.py.

Check out our tutorials for more help getting started.

2.1 Faster Inference

For improved speed, you can pass in a static matrix of all entity embeddings downloaded from here.

Then, our annotator can be run as:

```
ann = BootlegAnnotator(entity_embs_path=<PATH TO UNTARRED EMBEDDING FILE>)
ann.label_mentions("How many people are in Lincoln")["titles"]
```

Tip: If you have a larger amount of data to disambiguate, checkout out our end-to-end tutorial showing a more optimized end-to-end pipeline.

CHAPTER

THREE

EMMENTAL

We use the Emmental framework. Emmental is a framework for building multimodal multi-task learning systems. A key feature of Emmental is its task flow design where models are defined by the data flow through modules. By reusing modules in different tasks, you easy can extend your model to a multi-task setting.

We high encourage you to check out the Emmental docs and Emmental tutorials to understand the framework.

CHAPTER

FOUR

ENTITY PROFILES

Bootleg uses Wikipedia and Wikidata to collect and generate a entity database of metadata associated with an entity. We support both non-structural data (e.g., the title of an entity) and structural data (e.g., the type or relationship of an entity). We now describe how to generate entity profile data from scratch to be used for training and the structure of the profile data we already provide.

4.1 Generating Profiles

The database of entity data starts with a simple `jsonl` file of data associated with an entity. Specifically, each line is a JSON object

```
{
    "entity_id": "Q16240866",
    "mentions": [["benin national u20 football team",1],["benin national under20␣
↪football team",1]],
    "title": "Forbidden fruit",
    "description": "A fruit that once was considered not to be eaten",
    "types": {"hyena": ["<wordnet_football_team_108080025>"],
              "wiki": ["national association football team"],
              "relations":["country for sport","sport"]},
    "relations": [
        {"relation":"P1532","object":"Q962"},
    ],
}
```

The `entity_id` gives a unique string identifier of the entity. It does *not* have to start with a `Q`. As we normalize to Wikidata, our entities are referred to as QIDs. The `mentions` provides a list of known aliases to the entity and a prior score associated with that mention indicating the strength of association. The score is used to order the candidates. The `types` provides the different types and entity is and supports different type systems. In the example above, the two type systems are `hyena` and `wiki`. We also have a `relations` type system which treats the relationships an entity participates in as types. The `relations` JSON field provides the actual KG relationship triples where `entity_id` is the head.

Note: By default, Bootleg assigns the score for each mentions as being the global entity count in Wikipedia. We empirically found this was a better scoring method for incorporating Wikidata "also known as" aliases that did not appear in Wikipedia. This means the scores for the mentions for a single entity will be the same.

We provide a more complete sample of raw profile data to look at.

Once the data is ready, we provide an EntityProfile API to build and interact with the profile data. To create an entity profile for the model from the raw `jsonl` data, run

```
from bootleg.symbols.entity_profile import EntityProfile
path_to_file = "data/sample_raw_entity_data/raw_profile.jsonl"
# edit_mode means you are allowed to modify the profile
ep = EntityProfile.load_from_jsonl(path_to_file, edit_mode=True)
```

Note: By default, we assume that each alias can have a maximum of 30 candidates, 10 types, and 100 connections. You can change these by adding `max_candidates`, `max_types`, and `max_connections` as keyword arguments to `load_from_jsonl`. Note that increasing the number of maximum candidates increases the memory required for training and inference.

4.2 Profile API

Now that the profile is loaded, you can interact with the metadata and change it. For example, to get the title and add a type mapping, you'd run

```
ep.get_title("Q16240866")
# This is adding the type "country" to the "wiki" type system
ep.add_type("Q16240866", "sports team", "wiki")
```

Once ready to train or run a model with the profile data, simply save it

```
ep.save("data/sample_entity_db")
```

We have already provided the saved dump at `data/sample_entity_data`.

See our entity profile tutorial for a more complete walkthrough notebook of the API.

4.3 Training with a Profile

Inside the saved folder for the profile, all the mappings needed to run a Bootleg model are provided. There are three subfolders as described below. Note that we use the word `alias` and `mention` interchangeably.

- **`entity_mappings`: This folder contains non-structural entity data.**
  - `qid2eid`: This is a folder containing a Trie mapping from entity id (we refer to this as QID) to an entity index used internally to extract embeddings. Note that these entity ids start at 1 (0 index is reserved for a "not in candidate list" entity). We use Wikidata QIDs in our tutorials and documentation but any string identifier will work.
  - `qid2title.json`: This is a mapping from entity QID to entity Wikipedia title.
  - `qid2desc.json`: This is a mapping from entity QID to entity Wikipedia description.
  - `alias2qids`: This is a folder containing a RecordTrie mapping from possible mentions (or aliases) to a list possible candidates. We restrict our candidate lists to be a predefined max length, typically 30. Each item in the list is a pair of [QID, QID score] values. The QID score is used for sorting candidates before filtering to the top 30. The scores are otherwise not used in Bootleg. This mapping is mined from both Wikipedia and Wikidata (reach out with a github issue if you want to know more).

  - `alias2id`: This is a folder containing a Trie mapping from alias to alias index used internally by the model.
  - `config.json`: This gives metadata associated with the entity data. Specifically, the maximum number of candidates.

- **`type_mappings`: This folder contains type entity data for each type system subfolder. Inside each subfolder are the followi**
  - `qid2typenames`: Folder containing a RecordTrie mapping from entity QID to a list of type names.
  - `config.json`: Contains metadata of the maximum number of types allowed for an entity.
- **`kg_mappings`: This folder contains relationship entity data.**
  - `qid2relations`: Folder containing a RecordTrie mapping from entity QID to relations to list of tail QIDs associated with the entity QID.
  - `config.json`: Contains metadata of the maximum number of tail connections allowed for a particular head entity and relation.

Note: In Bootleg, we add types from a selected type system and add KG relationship triples to our entity encoder.

Note: In our public `entity_db` provided to run Bootleg models, we also provide `alias2qids_unfiltered.json` which provides our unfiltered, raw candidate mappings. We filter noisy aliases before running mention extraction.

Given this metadata, you simply need to specify the types, relation mappings and correct folder structures in a Bootleg training config. Specifically, these are the config parameters that need to be set to be associated with an entity profile.

```
data_config:
  entity_dir: data/sample_entity_data
  use_entity_desc: true
  entity_type_data:
    use_entity_types: true
    type_symbols_dir: type_mappings/wiki
  entity_kg_data:
    use_entity_kg: true
    kg_symbols_dir: kg_mappings
```

See our example config for a full reference, and see our entity profile tutorial for some methods to help modify configs to map to the entity profile correctly.

CHAPTER

FIVE

INPUTS

Given an input sentence, Bootleg outputs the entities that participate in the text. For example, given the sentence

`Where is Lincoln in Logan County`

Bootleg should output that Lincoln refers to Lincoln IL and Logan County to Logan County IL.

This disambiguation occurs in two parts. The first, described here, is mention extraction and candidate generation, where phrases in the input text are extracted to be disambiguation. For example, in the sentence above, the phrases "Lincoln" and "Logan County" should be extracted. Each phrase to be disambiguated is called a mention (or alias). Instead of disambiguating against all entities in Wikipedia, Bootleg uses predefined candidate maps that provide a small subset of possible entity candidates for each mention. The second step, described in Bootleg Model, is the disambiguation using Bootleg's neural model.

To understand how we do mention extraction and candidate generation, we first need to describe the profile data we have associated with an entity. Then we will describe how we perform mention extraction. Finally, we will provide details on the input data provided to Bootleg. Take a look at our tutorials to see it in action.

5.1 Entity Data

Bootleg uses Wikipedia and Wikidata to collect and generate a entity database of metadata associated with an entity. This is all located in `entity_db` and contains mappings from entities to structural data and possible mention. We describe the entity profiles in more details and how to generate them on our entity profile page. For reference, we have an EntityProfile class that loads and manages this metadata.

As our profile data does give us mentions that are associated with each entity, we now need to describe how we generate mentions.

5.2 Mention Extraction

Our mention extraction is a simple n-gram search over the input sentence (see bootleg/end2end/extract_mentions.py). Starting from the largest possible n-grams and working towards single word mentions, we iterate over the sentence and see if any n-gram is a hit in our `alias2qid` mapping. If it is, we extract that mention. This enusre that each mention has a set of candidates.

To prevent extracting noisy mentions, like the word "the", we filter our alias maps to only have words that appear approximately more that 1.5% of the time as mentions in our training data.

The input format is in `jsonl` format where each line is a json object of the form

- `sentence`: input sentence.

We output a jsonl with

- `sentence`: input sentence.
- `aliases`: list of extracted mentions.
- `spans`: list of word offsets [inclusive, exclusive) for each alias.

5.3 Textual Input

Once we have mentions and candidates, we are ready to run our Bootleg model. The raw input format is in `jsonl` format where each line is a json object. We have one json per sentence in our training data with the following files

- `sentence`: input sentence.
- `sent_idx_unq`: unique sentence index.
- `aliases`: list of extracted mentions.
- `qids`: list of gold entity id (if known). We use canonical Wikidata QIDs in our tutorials and documentation, but any id used in the entity metadata will work. The id can be `Q-1` if unknown, but you _must_ provide gold QIDs for training data.
- `spans`: list of word offsets [inclusive, exclusive) for each alias.
- `gold`: list of booleans if the alias is a gold anchor link from Wikipedia or a weakly labeled link.
- `slices`: list of json slices for evaluation. See advanced training for details.

For example, the input for the sentence above is

```
{
    "sentence": "Where is Lincoln in Logan County",
    "sent_idx_unq": 0,
    "aliases": ["lincoln", "logan county"],
    "qids": ["Q121", "Q???"],
    "spans": [[2,3], [4,6]],
    "gold": [True, True],
    "slices": {}
}
```

For more details on training, see our *training tutorial*.

CHAPTER

SIX

MODEL OVERVIEW

Given an input sentence, list of mentions to be disambiguated, and list of possible candidates for each mention (described in Input Data), Bootleg outputs the most likely candidate for each mention. Bootleg's model is a biencoder architecture and consists of two components: the entity encoder and context encoder. For each entity candidate, the entity encoder generates an embedding representing this entity from a textual input containing entity information such as the title, description, and types. The context encoder embeds the mention and its surrounded context. The selected candidate is the one with the highest dot product.

We now describe each step in detail and explain how to add/remove different parts of the entity encoder in our Bootleg Config.

6.1 Entity Encoder

The entity encoder is a BERT Transformer that takes a textual input for an entity and feeds it through BERT. During training, we take the `[CLS]` token as the entity embedding. There are four pieces of information we add to the textual input for an entity:

- `title`: Entity title. Comes from `qid2title.json`. This is always used.
- `description`: Entity description. Comes from `qid2desc.json`. This is toggled on/off.
- `type`: Entity type from one of the type systems specified in the config. If the entity has multiple types, we add them to the input as `<type_1> ; <type_2> ; ...`
- `KG`: Entity KG relations specified in the config. We add KG relations to the input as `<predicate_1> <object_1> ; <predicate_2> <object_2> ; ...` where the head of each triple is the entity in question.

The final entity input is `<title> [SEP] <types> [SEP] <relations> [SEP] <description>`.

You control what inputs are added by the following part in the input config. All the relevant entity encoder code is in bootleg/dataset.py.

```
data_config:
    ...
    use_entity_desc: true
    entity_type_data:
      use_entity_types: true
      type_symbols_dir: type_mappings/wiki
    entity_kg_data:
      use_entity_kg: true
      kg_symbols_dir: kg_mappings
    max_seq_len: 128
    max_seq_window_len: 64
    max_ent_len: 128
```

6.2 Context Encoder

Like the entity encoder, our context encode takes the context of a mention and feeds it through a BERT Transformer. The `[CLS]` token is used as th e relevant mention embedding. To allow BERT to understand where the mention is, we separate it by `[ENT_START]` and `[ENT_END]` clauses. As shown above, you can specify the maximum sequence length for the context encoder and the maximum window length. All the relevant context encoder code is in bootleg/dataset.py.

CHAPTER

SEVEN

BASIC TRAINING

We describe how to train a Bootleg model for named entity disambiguation (NED), starting from a new dataset. If you already have a dataset in the Bootleg format, you can skip to *Preparing the Config*. All commands should be run from the root directory of the repo.

7.1 Formatting the Data

We assume three components are available for input:

1. *Text datasets*
2. *Entity and alias data*
3. *Type and knowledge graph data*

For each component, we first describe the data requirements and then discuss how to convert the data to the expected format. Finally, we discuss the expected directory structure to organize the data components. We provide a small dataset sampled from Wikipedia in the directory `data` that we will use throughout this tutorial as an example.

7.1.1 Text Datasets

Requirements

1. Text data for training and dev datasets, and if desired, a test dataset, is available. For simplicity, in this tutorial, we just assume there is a dev dataset available.
2. Known aliases (also known as mentions) and linked entities are available. This information can be obtained for Wikipedia, for instance, by using anchor text on Wikipedia pages as aliases and the linked pages as the entity label.

Each dataset will need to follow the format described below.

We assume that the text dataset is formatted in a jsonlines file (each line is a dictionary) with the following keys:

- `sentence`: the text of the sentence.
- `sent_idx_unq`: a unique numeric identifier for each sentence in the dataset.
- `aliases`: the aliases in the sentence to disambiguate. Aliases serve as lookup keys into an alias candidate map to generate candidates, and may not actually appear in the text. For example, the phrase "Victoria Beckham" in the sentence may be weakly labelled as the alias "Victoria" by a simple heuristic.
- `spans`: the start and end word indices of the aliases in the text, where the end span is exclusive (like python slicing).

- `qids`: the id of the true entity for each alias. We use canonical Wikidata QIDs in this training tutorial, but any string indentifier will work. See Input Data for more information.
- `gold`: True if the entity label was an anchor link in the source dataset or otherwise known to be "ground truth"; False, if the entity label is from weak labeling techniques. While all provided alias-entity pairs can be used for training, only alias-entity pairs with a gold value of True are used for evaluation.
- (Optional) `slices`: indicates which alias-entity pairs are part of certain data subsets for evaluating performance on important subsets of the data (see the Advanced Training Tutorial for more details).

Using this format, an example line is:

```
{
    "sentence": "Heidi and her husband Seal live in Vegas . ",
    "sent_idx_unq": 0,
    "aliases": ["heidi", "seal", "vegas"],
    "spans": [[0,1], [4,5], [7,8]],
    "qids": ["Q60036", "Q218091", "Q23768"],
    "gold": [true, true, true]
}
```

We also provide sample training and dev datasets as examples of text datasets in the proper format.

7.1.2 Entities and Aliases

You need an entity profile dump for training. Our Entity Profile page details how to create the correct metadata for the entities and aliases and the structural files. The path is added to the config in the `entity_data_dir` param (see below).

7.1.3 Directory Structure

We assume the data above is saved in the following directory structure, where the specific directory and filenames can be set in the config discussed in *Preparing the Config*. We will also discuss how to generate the `prep` directories in *Preprocessing the Data*. The `emb_data` directory can be shared across text datasets and entity sets, and the `entity_data` directory can be shared across text datasets (if they use the same set of entities).

```
text_data/
    train.jsonl
    dev.jsonl
    prep/

entity_db/
     type_mappings/
         wiki/
             qid2typenames/
             config.json
     kg_mappings/
         config.json
         qid2relations/
         kg_adj.txt
     entity_mappings/
         alias2qids/
         qid2eid/
```

(continues on next page)

(continued from previous page)

```
        qid2title.json
        qid2desc.json
        alias2id/
        config.json
```

7.2 Preparing the Config

Once the data has been converted to the correct format, we are ready to prepare the config. We provide a sample config in configs/tutorial/sample_config.yaml. The full parameter options and defaults for the config file are explain in Configuring Bootleg. If values are not provided in the YAML config, the default values are used. We provide a brief overview of the configuration settings here.

The config parameters are organized into five main groups:

- `emmental`: Emmental parameters.
- `run_config`: run time settings that aren't set in Emmental; e.g., eval batch size and number of dataloader threads.
- `train_config`: training parameters of batch size.
- `model_config`: model parameters of hidden dimension.
- `data_config`: paths of text data, embedding data, and entity data to use for training and evaluation, as well as configuration details for the entity embeddings.

We highlight a few parameters in the `emmental`.

- `log_dir` should be set to specify where log output and model checkpoints should be saved. When a new model is trained, Emmental automatically generates a timestamp and saves output to a folder with the timestamp inside the `log_dir`.
- `evaluation_freq` indicates how frequently the evaluation on the dev set should be run. Steps corresponds to epochs by default (but can be configured to batches), such that 0.2 means 0.2 of an epoch has been processed.
- `checkpoint_freq` indicates when to save a model checkpoint after performing evaluation. If set to 1, then a model checkpoint will be saved every time dev evaluation is run.

See Emmental Config for more information.

We now focus on the `data_config` parameters as these are the most unique to Bootleg. We walk through the key parameters in the `data_config` to pay attention to.

7.2.1 Directories

We define the paths to the directories through the `data_dir`, `entity_dir`, and `entity_map_dir` config keys. The first three correspond to the top-level directories introduced in *Directory Structure*. The `entity_map_dir` includes the entity JSON mappings produced in *Entities and Aliases* and should be inside the `entity_dir`. For example, to follow the directory structure set up in the `data` directory, we would have:

```
"data_dir": "data/sample_text_data",
"entity_dir": "data/sample_entity_data",
"entity_map_dir": "entity_mappings"
```

7.2.2 Entity Encoder

As described in the _Bootleg Model, Bootleg generates an embedding entity from an Transformer encoder. The resources which go in to the encoder input are defined in the config as shown below.

```
data_config:
    ...
    use_entity_desc: true
    entity_type_data:
      use_entity_types: true
      type_symbols_dir: type_mappings/wiki
      max_ent_type_len: 20
    entity_kg_data:
      use_entity_kg: true
      kg_symbols_dir: kg_mappings
      max_ent_kg_len: 60
    max_seq_len: 128
    max_seq_window_len: 64
    max_ent_len: 128
```

In this example, the entity input will have descriptions, types, and relations. You can control the total length of each resource by a `max_ent_type_len` and `max_ent_kg_len` param and the maximum entity length by `max_ent_len`.

7.2.3 Entity Masking

A secret sauce to getting our Bootleg encoder to pay attention to the types and relationships is to apply masking of the mention and entity title. Without masking, the model will rely heavily on mention-title memorization and ignore more subtle structural cues required for the tail. To overcome this, we mask entity titles in the entity encoder and mentions in the context encoder. By default, we mask titles and mentions 50% of the time, with more popular entities being masked up to 95% of the time. To turn this off, in `data_config`, set `popularity_mask` to be `false`.

If desired, we also support MLM style masking of the context input. By default, we do not use this masking, but you can turn it on by setting `context_mask_perc` to be between 0.0 and 1.0 in `data_config`.

7.2.4 Candidates and Aliases

Candidate Not in List

Bootleg supports two types of candidate lists: (1) assume that the true entity must be in the candidate list, (2) use a NIL or "No Candidate" (NC) as another candidate, and does not require that the true candidate is the candidate list. Not that if using (1), during training, the gold candidate *must* be in the list or preprocessing with fail. The gold candidate does not have to be in the candidate set for evaluation. To switch between these two modes, we provide the `train_in_candidates` parameter (where True indicates (1)).

Multiple Candidate Maps

Within the `entity_map_dir` there may be multiple candidate maps for the same set of entities. For instance, a benchmark dataset may use a specific candidate mapping. To specify which candidate map to use, we set the `alias_cand_map` value in the config.

7.2.5 Datasets

We define the train, dev, and test datasets in `train_dataset`, `dev_dataset`, and `test_dataset` respectively. For each dataset, we need to specify the name of the file with the `file` key. We can also specify whether to use weakly labeled alias-entity pairs (pairs that are labeled heurisitcally during preprocessing). For training, if `use_weak_label` is True, these alias-entity pairs will contribute to the loss. For evaluation, the weakly labelled alias-entity pairs will only be used as more signal for other alias-entity pairs (e.g. for collective disambiguation), but will not be scored. As an example of a dataset entry, we may have:

```
train_dataset:
   file: train.jsonl
   use_weak_label: true
```

7.2.6 Word Embeddings

Bootleg leverages BERT Transformers to encode the entities and mention context. This type of BERT model and its size is configured in the `word_embedding` section of the config. You can change which HuggingFace BERT model by the `bert_model` param, change its cached direction by `cache_dir`, and the number of layers by `context_layers` and `entity_layers`.

Finally, in the `data_config`, we define a maximum word token length through `max_seq_len` and that max window length around a mention by `max_seq_window_len`.

7.3 Preprocessing the Data

Prior to training, if the data is not already prepared, we will preprocess or prep the data. This is where we convert the context and entity token data to a memory-mapped format for the dataloader to quickly load during training. If the data does not change, this preprocessing only needs to happen once.

Warning: errors may occur if the file contents change but the file names stay the same, since the preprocessed data uses the file name as a key and will be loaded based on the stale data. In these cases, we recommend removing the ``prep`` directories or assigning a new prep directory (by setting ``data_prep_dir`` or ``entity_prep_dir`` in the config) and repeating preprocessing.

7.3.1 Prep Directories

As the preprocessed knowledge graph and type embedding data only depends on the entities, we store it in a prep directory in the entity directory to be shared across all datasets that use the same entities and knowledge graph/type data. We store all other preprocessed data in a prep directory inside the data directory.

7.4 Training the Model

After the data is prepped, we are ready to train the model! As this is just a tiny random sample of Wikipedia sentences with sampled KG information, we do not expect the results to be good (for instance, we haven't seen most aliases in dev in training and we do not have an adequate number of examples to learn reasoning patterns). We recommend training on GPUs. To train the model on a single GPU, we run:

```
python3 bootleg/run.py --config_script configs/tutorial/sample_config.yaml
```

If a GPU is not available, we can also get away with training this tiny dataset on the CPU by adding the flag below to the command. Flags follow the same hierarchy and naming as the config, and the `cpu` parameter could also have been set directly in the config file in the `run_config` section:

```
python3 bootleg/run.py --config_script configs/tutorial/sample_config.json --emmental.
↪device -1
```

At each eval step, we see a json save of eval metrics. At the beginning end end of the model training, you should see a print out of the log direction. E.g.,

```
Saving metrics to logs/turtorial/2021_03_11/20_31_11/02b0bb73
```

Inside the log directory, you'll find all checkpoints, the `emmental.log` file, `train_metrics.txt`, and `train_disambig_metrics.csv`. The latter two files give final eval scores of the model. For example, after 10 epochs, `train_disambig_metrics.csv` shows

```
task,dataset,split,slice,mentions,mentions_notNC,acc_boot,acc_boot_notNC,acc_pop,acc_pop_
↪notNC
NED,Bootleg,dev,final_loss,70,70,0.8714285714285714,0.8714285714285714,0.
↪8714285714285714,0.8714285714285714
NED,Bootleg,test,final_loss,70,70,0.8714285714285714,0.8714285714285714,0.
↪8714285714285714,0.8714285714285714
```

The fields are

- `task`: the task name (will be NED for disambiguation metrics).
- `dataset`: dataset (if case of multi-modal training)
- `slice`: the subset of the dataset evaluated. `final_loss` is the slice which includes all mentions in the dataset. If you set `emmental.online_eval` to be True in the config, training metrics will also be reported and collected.
- `mentions`: the number of mentions (aliases) under evaluation.
- `mentions_notNC`: the number of mentions (aliases) under evaluation where the gold QID is in the candidate list.
- `acc_boot`: the accuracy of Bootleg.
- `acc_boot_notNC`: the accuracy of Bootleg for notNC mentions.
- `acc_boot`: the accuracy of a baseline where the first candidate is always selected as the answer.
- `acc_boot_notNC`: the accuracy of the baseline for notNC mentions.

As our data was very tiny, our model is not doing great, but the train loss is going down!

7.5 Evaluating the Model

After the model is trained, we can also run eval to get test scores or to save predictions. To eval the model on a single GPU, we run:

```
python3 bootleg/run.py --config_script configs/tutorial/sample_config.yaml --mode dump_
↪preds --emmental.model_path logs/turtorial/2021_03_11/20_31_11/02b0bb73/last_model.pth
```

This will generate a label file at `logs/turtorial/2021_03_11/20_38_09/c5e204dc/dev/last_model/bootleg_labels.jsonl` (path is printed). This can be read it for evaluation and error analysis. Check out the End-to-End Tutorial on our Tutorials Page for seeing how to do this and for evaluating pretrained Bootleg models.

7.6 Advanced Training

Bootleg supports distributed training using PyTorch's Distributed Data Parallel framework. This is useful for training large datasets as it parallelizes the computation by distributing the batches across multiple GPUs. We explain how to use distributed training in Bootleg to train a model on a large dataset (all of Wikipedia with 50 million sentences) in the Advanced Training Tutorial.

CHAPTER

EIGHT

CONFIGURING BOOTLEG

By default, Bootleg loads the default config from bootleg/utils/parser/bootleg_args.py. When running a Bootleg model, the user may pass in a custom JSON or YAML config via:

```
python3 bootleg/run.py --config_script <path_to_config>
```

This will override all default values. Further, if a user wishes to overwrite a param from the command line, they can pass in the value, using the dotted path of the argument. For example, to overwrite the data directory (the param `data_config.data_dir`, the user can enter:

```
python3 bootleg/run.py --config_script <path_to_config> --data_config.data_dir <path_to_
↪data>
```

Bootleg will save the run config (as well as a fully parsed verison with all defaults) in the log directory.

Finally, when evaluating Bootleg using the annotator, Bootleg processes possible mentions in text with three environment flags: `BOOTLEG_STRIP`, `BOOTLEG_LOWER`, `BOOTLEG_LANG_CODE`. The first sets the language to use for Spacy. The second is if the user wants to strip punctuation on mentions (set to False by default). The third is if the user wants to call `.lower()` (set to True by default).

8.1 Emmental Config

As Bootleg uses Emmental, the training parameters (e.g., learning rate) are set and handled by Emmental. We provide all Emmental params, as well as our defaults, at bootleg/utils/parser/emm_parse_args.py. All Emmental params are under the `emmental` configuration group. For example, to change the learning rate and number of epochs in a config, add

```
emmental:
  lr: 1e-4
  n_epochs: 10
run_config:
  ...
```

You can also change Emmental params by the command line with `--emmental.<emmental_param> <value>`.

8.2 Example Training Config

An example training config is shown below

```
emmental:
  lr: 2e-5
  n_epochs: 3
  evaluation_freq: 0.2
  warmup_percentage: 0.1
  lr_scheduler: linear
  log_path: logs/wiki
  l2: 0.01
  grad_clip: 1.0
  fp16: true
run_config:
  eval_batch_size: 32
  dataloader_threads: 4
  dataset_threads: 50
train_config:
  batch_size: 32
model_config:
  hidden_size: 200
data_config:
  data_dir: bootleg-data/data/wiki_title_0122
  data_prep_dir: prep
  use_entity_desc: true
  entity_type_data:
    use_entity_types: true
    type_symbols_dir: type_mappings/wiki
  entity_kg_data:
    use_entity_kg: true
    kg_symbols_dir: kg_mappings
  entity_dir: bootleg-data/data/wiki_title_0122/entity_db
  max_seq_len: 128
  max_seq_window_len: 64
  max_ent_len: 128
  overwrite_preprocessed_data: false
  dev_dataset:
    file: dev.jsonl
    use_weak_label: true
  test_dataset:
    file: test.jsonl
    use_weak_label: true
  train_dataset:
    file: train.jsonl
    use_weak_label: true
  train_in_candidates: true
  word_embedding:
    cache_dir: bootleg-data/embs/pretrained_bert_models
    bert_model: bert-base-uncased
```

8.3 Default Config

The default Bootleg config is shown below

```
"""Bootleg default configuration parameters.

In the json file, everything is a string or number. In this python file,
if the default is a boolean, it will be parsed as such. If the default
is a dictionary, True and False strings will become booleans. Otherwise
they will stay string.
"""
import multiprocessing

config_args = {
    "run_config": {
        "spawn_method": (
            "forkserver",
            "multiprocessing spawn method. forkserver will save memory but have slower␣
↪startup costs.",
        ),
        "eval_batch_size": (128, "batch size for eval"),
        "dump_preds_accumulation_steps": (
            1000,
            "number of eval steps to accumulate the output tensors for before saving␣
↪results to file",
        ),
        "dump_preds_num_data_splits": (
            1,
            "number of chunks to split the input file; helps with OOM issues",
        ),
        "overwrite_eval_dumps": (False, "overwrite dumped eval data"),
        "dataloader_threads": (16, "data loader threads to feed gpus"),
        "log_level": ("info", "logging level"),
        "dataset_threads": (
            int(multiprocessing.cpu_count() * 0.9),
            "data set threads for prepping data",
        ),
        "result_label_file": (
            "bootleg_labels.jsonl",
            "file name to save predicted entities in",
        ),
        "result_emb_file": (
            "bootleg_embs.npy",
            "file name to save contextualized embs in",
        ),
    },
    # Parameters for hyperparameter tuning
    "train_config": {
        "batch_size": (32, "batch size"),
    },
    "model_config": {
        "hidden_size": (300, "hidden dimension for the embeddings before scoring"),
        "normalize": (False, "normalize embeddings before dot product"),
```

(continues on next page)

(continued from previous page)

```
        "temperature": (1.0, "temperature for softmax in loss"),
    },
    "data_config": {
        "eval_slices": ([], "slices for evaluation"),
        "train_in_candidates": (
            True,
            "Train in candidates (if False, this means we include NIL entity)",
        ),
        "data_dir": ("data", "where training, testing, and dev data is stored"),
        "data_prep_dir": (
            "prep",
            "directory where data prep files are saved inside data_dir",
        ),
        "entity_dir": (
            "entity_data",
            "where entity profile information and prepped embedding data is stored",
        ),
        "entity_prep_dir": (
            "prep",
            "directory where prepped embedding data is saved inside entity_dir",
        ),
        "entity_map_dir": (
            "entity_mappings",
            "directory where entity json mappings are saved inside entity_dir",
        ),
        "alias_cand_map": (
            "alias2qids",
            "name of alias candidate map file, should be saved in entity_dir/entity_map_
↪dir",
        ),
        "alias_idx_map": (
            "alias2id",
            "name of alias index map file, should be saved in entity_dir/entity_map_dir",
        ),
        "qid_cnt_map": (
            "qid2cnt.json",
            "name of alias index map file, should be saved in data_dir",
        ),
        "max_seq_len": (128, "max token length sentences"),
        "max_seq_window_len": (64, "max window around an entity"),
        "max_ent_len": (128, "max token length for entire encoded entity"),
        "context_mask_perc": (
            0.0,
            "mask percent for context tokens in addition to tail masking",
        ),
        "popularity_mask": (
            True,
            "whether to use popularity masking for training in the entity and context␣
↪encoders",
        ),
        "overwrite_preprocessed_data": (False, "overwrite preprocessed data"),
        "print_examples_prep": (True, "whether to print examples during prep or not"),
```

(continues on next page)

(continued from previous page)

```
        "use_entity_desc": (True, "whether to use entity descriptions or not"),
        "entity_type_data": {
            "use_entity_types": (False, "whether to use entity type data"),
            "type_symbols_dir": (
                "type_mappings/wiki",
                "directory to type symbols inside entity_dir",
            ),
            "max_ent_type_len": (20, "max WORD length for type sequence"),
        },
        "entity_kg_data": {
            "use_entity_kg": (False, "whether to use entity type data"),
            "kg_symbols_dir": (
                "kg_mappings",
                "directory to kg symbols inside entity_dir",
            ),
            "max_ent_kg_len": (60, "max WORD length for kg sequence"),
        },
        "train_dataset": {
            "file": ("train.jsonl", ""),
            "use_weak_label": (True, "Use weakly labeled mentions"),
        },
        "dev_dataset": {
            "file": ("dev.jsonl", ""),
            "use_weak_label": (True, "Use weakly labeled mentions"),
        },
        "test_dataset": {
            "file": ("test.jsonl", ""),
            "use_weak_label": (True, "Use weakly labeled mentions"),
        },
        "word_embedding": {
            "bert_model": ("bert-base-uncased", ""),
            "context_layers": (12, ""),
            "entity_layers": (12, ""),
            "cache_dir": (
                "pretrained_bert_models",
                "Directory where word embeddings are cached",
            ),
        },
    },
}
```

CHAPTER

NINE

DISTRIBUTED TRAINING

We discuss how to use distributed training to train a Bootleg model on the full Wikipedia save. This tutorial assumes you have already completed the Basic Training Tutorial.

As Wikipedia has over 5 million entities and over 50 million sentences, training on the full Wikipedia save is computationally expensive. We recommend using a p4d.24xlarge instance on AWS to train on Wikipedia.

We provide a config for training Wikipedia here. Note this config is the config used to train the pretrained model provided in the End-to-End Tutorial.

9.1 1. Downloading the Data

We provide scripts to download:

1. Prepped Wikipedia data (training and dev datasets)
2. Wikipedia entity data and embedding data

To download the Wikipedia data, run the command below with the directory to download the data to. Note that the prepped Wikipedia data will require ~200GB of disk space and will take some time to download and decompress the prepped Wikipedia data (16GB compressed, ~150GB uncompressed).

```
bash download_wiki.sh <DOWNLOAD_DIRECTORY>
```

To download (2) above, run the command

```
bash download_data.sh <DOWNLOAD_DIRECTORY>
```

At the end, the directory structure should be

```
<DOWNLOAD_DIRECTORY>
  wiki_data/
      prep/
  entity_db/
      entity_mappings/
      type_mappings/
      kg_mappings/
      prep/
```

9.2 2. Setting up Distributed Training

Emmental, the training framework of Bootleg, supports distributed training using PyTorch's Data Parallel or Distributed Data Parallel framework. We recommend DDP for training.

There is nothing that needs to change to get distributed training to work. We do, however, recommend setting the following params

```
emmental:
    ...
    distributed_backend: nccl
    fp16: true
```

This allows for fp16 and making sure the `nccl` backend is used. Note that when training with DDP, the `batch_size` is **per gpu**. With standard data parallel, the `batch_size` is across all GPUs.

From the Basic Training Tutorial, recall that the directory paths should be set to where we want to save our models and read the data, including:

- `cache_dir` in `data_config.word_embedding`
- `data_dir` and `entity_dir` in `data_config`

We have already set these directories in the provided Wikipedia config, but you will need to update `data_dir` and `entity_dir` to where you downloaded the data in step 1 and may want to update `log_dir` to where you want to save the model checkpoints and logs.

9.3 3. Training the Model

As we provide the Wikipedia data already prepped, we can jump immediately to training. To train the model with 8 gpus using DDP, we simply run:

```
python3 -m torch.distributed.run --nproc_per_node=8  bootleg/run.py --config_script␣
↪configs/tutorial/wiki_uncased_ft.yaml
```

To train using DP, simply run

```
python3 bootleg/run.py --config_script configs/tutorial/wiki_uncased_ft.yaml
```

and Emmental will automatically using distributed training (you can turn this off by `dataparallel: false` in the `emmental` config block.

Once the training begins, we should see all GPUs being utilized.

If we want to change the config (e.g. change the maximum number of aliases or the maximum word token len), we would need to re-prep the data and would run the command below. Note it takes several hours to perform Wikipedia pre-processing on a 56-core machine:

9.4 4. Evaluating with Slices

We use evaluation slices to understand the performance of Bootleg on important subsets of the dataset. To use evaluation slices, alias-entity pairs are labelled as belonging to specific slices in the `slices` key of the dataset.

In the Wikipedia data in this tutorial, we provide three "slices" of the dev dataset in addition to the "final_loss" (all examples) slice. For each of these three slices, the alias being scored must have more than one candidate. This filters trivial examples all models get correct.

- `unif_NS_TS`: The gold entity does not occur in the training dataset (toes).
- `unif_NS_TL`: The gold entity occurs globally 10 or fewer times in the training dataset (tail).
- `unif_NS_TO`: The gold entity occurs globally between 11-1000 times in the training dataset (torso).
- `unif_NS_HD`: The gold entity occurs globally greater than 1000 times in the training dataset (head).
- `unif_NS_all`: All gold entities.

To use the slices for evaluation, they must also be specified in the `eval_slices` section of the `run_config` (see the Wikipedia config as an example).

When the dev evaluation occurs during training, we should see the performance on each of the slices that are specified in `eval_slices`. These slices help us understand how well Bootleg performs on more challenging subsets. The frequency of dev evaluation can be specified by the `evaluation_freq` parameter in the `emmental` block.

CHAPTER

TEN

CHANGELOG

All notable changes to this project will be documented in this file.

The format is based on Keep a Changelog and this project adheres to Semantic Versioning 2.0.0 conventions. The maintainers will create a git tag for each release and increment the version number found in bootleg/_version.py accordingly. We release tagged versions to PyPI automatically using GitHub Actions.

Note: Bootleg is still under active development and APIs may still change rapidly. Until we release v1.0.0, changes in MINOR version indicate backward incompatible changes.

10.1 Unreleased 1.1.1dev0

10.1.1 Fixed

- Corrected training with `train_in_candidates` set to False.

10.2 1.1.0 - 2022-04-12

10.2.1 Changed

- We did an architectural change and switched to a biencoder model. This changes our task flow and dataprep. This new model uses less CPU storage and uses the standard BERT architecture. Our entity encoder now takes a textual input of an entity that contains its title, description, KG relationships, and types.
- To support larger files for dumping predictions over, we support adding an `entity_emb_file` to the model (extracted from `extract_all_entities.py`. This will make evaluation faster. Further, we added `dump_preds_num_data_splits` to split a file before dumping. As each file pass gets a new dataload object, this can mitiage any torch dataloader memory issues that happens over large files.
- Renamed `eval_accumulation_steps` to `dump_preds_accumulation_steps`.
- Removed option to `dump_embs`. Users should use `dump_preds` instead. The output file will have `entity_ids` attribute that will index into the extracted entity embeddings.
- Restructured our `entity_db` data for faster loading. It uses Tries rather than jsons to store the data for read only mode. The KG relations are not backwards compatible.
- Moved to character spans for input data. Added utils.preprocessing.convert_to_char_spans as a helper function to convert from word offsets to character offsets.

10.2.2 Added

- `BOOTLEG_STRIP` and `BOOTLEG_LOWER` environment variables for `get_lnrm`.
- `extract_all_entities.py` as a way to extract all entity embeddings. These entity embeddings can be used in eval and be used downstream. Uses can use `get_eid` from the `EntityProfile` to extract the row id for a specific entity.

10.3 1.0.5 - 2021-08-20

10.3.1 Fixed

- Fixed -1 command line argparse error
- Adjusted requirements

10.4 1.0.4 - 2021-07-12

10.4.1 Added

- Tutorial to generate contextualized entity embeddings that perform better downstream

10.4.2 Fixed

- Bump version of Pydantic to 1.7.4

10.5 1.0.3 - 2021-06-29

10.5.1 Fixed

- Corrected how custom candidates were handled in the BootlegAnnotator when using `extracted_examples`
- Fixed memory leak in BooltegAnnotator due to missing `torch.no_grad()`

10.6 1.0.2 - 2021-04-28

10.6.1 Added

- Support for `min_alias_len` to `extract_mentions` and the `BootlegAnnotator`.
- `return_embs` flag to pass into `BootlegAnnotator` that will return the contextualized embeddings of the entity (using key `embs`) and entity candidates (using key `cand_embs`).

10.6.2 Changed

- Removed condition that aliases for eval must appear in candidate lists. We now allow for eval to not have known aliases and always mark these as incorrect. When dumping predictions, these get "-1" candidates and null probabilities.

10.6.3 Fixed

- Corrected `fit_to_profile` to rebuild the title embeddings for the new entities.

10.7 1.0.1 - 2021-03-22

Note: If upgrading to 1.0.1 from 1.0.0, you will need to re-download our models given the links in the README.md. We altered what keys were saved in the state dict, but the model weights are unchanged.

10.7.1 Added

- `data_config.print_examples_prep` flag to toggle data example printing during data prep.
- `data_config.dump_preds_accumulation_steps` to support subbatching dumping of predictings. We save outputs to separate files of size approximately `data_config.dump_preds_accumulation_steps*data_config.eval_batch_size` and merge into a final file at the end.
- Entity Profile API. See the docs. This allows for modifying entity metadata as well as adding and removing entities. We profile methods for refitting a model with a new profile for immediate inference, no finetuning needed.

10.7.2 Changed

- Support for not using multiprocessing if use sets `data_config.dataset_threads` to be 1.
- Added better argument parsing to check for arguments that were misspelled or otherwise wouldn't trigger anything.
- Code is now Flake8 compatible.

10.7.3 Fixed

- Fixed readthedocs so the BootlegAnnotator was loaded correctly.
- Fixed logging in BootlegAnnotator.
- Fixed `use_exact_path` argument in Emmental.

10.8 1.0.0 - 2021-02-15

We did a major rewrite of our entire codebase and moved to using Emmental for training. Emmental allows for each multi-task training, FP16, and support for both DataParallel and DistributedDataParallel.

The overall functionality of Bootleg remains unchanged. We still support the use of an annotator and bulk mention extraction and evaluation. The core Bootleg model has remained largely unchanged. Checkout our documentation for more information on getting started. We have new models trained as described in our README.

Note: This branch os **not** backwards compatible with our old models or code base.

Some more subtle changes are below

10.8.1 Added

- Support for data parallel and distributed data parallel training (through Emmental)
- FP16 (through Emmental)
- Easy install with `BootlegAnnotator`

10.8.2 Changed

- Mention extraction code and alias map has been updated
- Models trained on October 2020 save of Wikipedia
- Have uncased and cased models

10.8.3 Removed

- Support for slice-based learning
- Support for `batch prepped` KG embeddings (only use `batch on the fly`)

CHAPTER

ELEVEN

INSTALLATION

To test changes in the package, you install it in editable mode locally in your virtualenv by running:

```
$ make dev
```

This will also install our pre-commit hooks and local packages needed for style checks.

Tip: If you need to install a locally edited version of bootleg in a separate location, such as an application, you can directly install your locally modified version:

```
$ pip install -e path/to/bootleg/
```

in the virtualenv of your application.

Note, you can test the *pip* downloadable version using TestPyPI. To handle dependencies, run

```
pip install --index-url https://test.pypi.org/simple/ --extra-index-url https://pypi.org/
↪simple bootleg
```

CHAPTER

TWELVE

TESTING

We use pytest to run our tests. Our tests are all located in the `test` directory in the repo, and are meant to be run *after* installing Bootleg locally.

To run our tests, just run:

```
$ make test
```

CHAPTER

THIRTEEN

CODE STYLE

For code consistency, we have a pre-commit configuration file so that you can easily install pre-commit hooks to run style checks before you commit your files. You can setup our pre-commit hooks by running:

```
$ pip install .[dev]
$ pre-commit install
```

Or, just run:

```
$ make dev
```

Now, each time you commit, checks will be run using the packages explained below.

We use black as our Python code formatter with its default settings. Black helps minimize the line diffs and allows you to not worry about formatting during your own development. Just run black on each of your files before committing them.

Tip: Whatever editor you use, we recommend checking out black editor integrations to help make the code formatting process just a few keystrokes.

For sorting imports, we reply on isort. Our repository already includes a `.isort.cfg` that is compatible with black. You can run a code style check on your local machine by running our checks:

```
$ make check
```

CHAPTER

FOURTEEN

BOOTLEG

14.1 bootleg package

14.1.1 Subpackages

bootleg.end2end package

Submodules

bootleg.end2end.annotator_utils module

Annotator utils.

class bootleg.end2end.annotator_utils.**DownloadProgressBar**

Bases: object

Progress bar.

bootleg.end2end.bootleg_annotator module

BootlegAnnotator.

class bootleg.end2end.bootleg_annotator.**BootlegAnnotator**(*config: Optional[Union[str, Dict[str, Any]]] = None*, *device: Optional[int] = None*, *min_alias_len: int = 1*, *max_alias_len: int = 6*, *threshold: float = 0.0*, *cache_dir: Optional[str] = None*, *model_name: Optional[str] = None*, *entity_emb_file: Optional[str] = None*, *return_embs: bool = False*, *return_ctx_embs: bool = False*, *extract_method: str = 'spacy'*, *verbose: bool = False*)

Bases: object

Bootleg on-the-fly annotator.

BootlegAnnotator class: convenient wrapper of preprocessing and model eval to allow for annotating single sentences at a time for quick experimentation, e.g. in notebooks.

Parameters

- **config** – model config or path to config (default None)
- **device** – model device, -1 for CPU (default None)
- **min_alias_len** – minimum alias length (default 1)
- **max_alias_len** – maximum alias length (default 6)
- **threshold** – probability threshold (default 0.0)
- **cache_dir** – cache directory (default None)
- **model_name** – model name (default None)
- **entity_emb_file** – entity embedding file (default None)
- **return_embs** – whether to return entity embeddings or not (default False)
- **return_ctx_embs** – whether to return context embeddings or not (default False)
- **extract_method** – mention extraction method
- **verbose** – verbose boolean (default False)

extract_mentions(*text*)

Mention extraction wrapper.

Parameters **text** – text to extract mentions from

Returns: JSON object of sentence to be used in eval

get_entity_tokens(*qid*)

Get entity tokens.

Parameters **qid** – entity QID

Returns Dict of input tokens for forward pass.

get_forward_batch(*input_ids*, *token_type_ids*, *attention_mask*, *entity_token_ids*, *entity_type_ids*, *entity_attention_mask*, *entity_cand_eid*, *generate_entity_inputs*)

Generate emmental batch.

Parameters

- **input_ids** – word token ids
- **token_type_ids** – word token type ids
- **attention_mask** – work attention mask
- **entity_token_ids** – entity token ids
- **entity_type_ids** – entity type ids
- **entity_attention_mask** – entity attention mask
- **entity_cand_eid** – entity candidate eids
- **generate_entity_inputs** – whether to generate entity id inputs

Returns: X_dict for emmental

get_sentence_tokens(*sample*, *men_idx*)

Get context tokens.

Parameters

- **sample** – Dict sample after extraction

- **men_idx** – mention index to select

Returns: Dict of tokenized outputs

label_mentions(*text_list=None*, *extracted_examples=None*)

Extract mentions and runs disambiguation.

If user provides extracted_examples, we will ignore text_list.

Parameters

- **text_list** – list of text to disambiguate (or single string) (can be None if extracted_examples is not None)
- **extracted_examples** – List of Dicts of keys "sentence", "aliases", "spans", "cands" (QIDs) (optional)

Returns: Dict of

- `qids`: final predicted QIDs,
- `probs`: final predicted probs,
- `titles`: final predicted titles,
- `cands`: all entity candidates,
- `cand_probs`: probabilities of all candidates,
- `char_spans`: final extracted char spans,
- `aliases`: final extracted aliases,
- `embs`: final entity contextualized embeddings (if return_embs is True)
- `cand_embs`: final candidate entity contextualized embeddings (if return_embs is True)

set_threshold(*value*)

Set threshold.

Parameters **value** – threshold value

bootleg.end2end.bootleg_annotator.**create_config**(*model_path*, *data_path*, *model_name*)

Create Bootleg config.

Parameters

- **model_path** – model directory
- **data_path** – data directory
- **model_name** – model name

Returns: updated config

bootleg.end2end.bootleg_annotator.**create_sources**(*model_path*, *data_path*, *model_name*)

Download Bootleg data and saves in log dir.

Parameters

- **model_path** – model directory
- **data_path** – data directory
- **model_name** – model name to download

bootleg.end2end.bootleg_annotator.**get_default_cache**()

Get default cache directory for saving Bootleg data.

bootleg.end2end.extract_mentions module

Extract mentions.

This file takes in a jsonlines file with sentences and extract aliases and spans using a pre-computed alias table.

bootleg.end2end.extract_mentions.**chunk_text_data**(*input_src*, *chunk_files*, *chunk_size*, *num_lines*)

Chunk text input file into chunk_size chunks.

Parameters

- **input_src** – input file
- **chunk_files** – list of chunk file names
- **chunk_size** – chunk size in number of lines
- **num_lines** – total number of lines

bootleg.end2end.extract_mentions.**create_out_line**(*sent_obj*, *final_aliases*, *final_spans*, *found_char_spans*)

Create JSON output line.

Parameters

- **sent_obj** – input sentence JSON
- **final_aliases** – list of final aliases
- **final_spans** – list of final spans
- **found_char_spans** – list of final char spans

Returns: JSON object

bootleg.end2end.extract_mentions.**extract_mentions**(*in_filepath*, *out_filepath*, *entity_db_dir*, *extract_method='ngram_spacy'*, *min_alias_len=1*, *max_alias_len=6*, *num_workers=8*, *num_chunks=None*, *verbose=False*)

Extract mentions from file.

Parameters

- **in_filepath** – input file
- **out_filepath** – output file
- **entity_db_dir** – path to entity db
- **extract_method** – mention extraction method
- **min_alias_len** – minimum alias length (in words)
- **max_alias_len** – maximum alias length (in words)
- **num_workers** – number of multiprocessing workers
- **num_chunks** – number of subchunks to feed to workers
- **verbose** – verbose boolean

bootleg.end2end.extract_mentions.**main**()

Run.

bootleg.end2end.extract_mentions.**merge_files**(*chunk_outfiles*, *out_filepath*)

Merge output files.

Parameters

- **chunk_outfiles** – list of chunk files
- **out_filepath** – final output file path

bootleg.end2end.extract_mentions.**parse_args**()

Generate args.

bootleg.end2end.extract_mentions.**subprocess**(*args*)

Extract mentions single process.

Parameters **args** – subprocess args

Module contents

End2End init.

bootleg.layers package

Submodules

bootleg.layers.alias_to_ent_encoder module

AliasEntityTable class.

class bootleg.layers.alias_to_ent_encoder.**AliasEntityTable**(*data_config*, *entity_symbols*)

Bases: torch.nn.modules.module.Module

Stores table of the K candidate entity ids for each alias.

Parameters

- **data_config** – data config
- **entity_symbols** – entity symbols

classmethod build_alias_table(*data_config*, *entity_symbols*)

Construct the alias to EID table.

Parameters

- **data_config** – data config
- **entity_symbols** – entity symbols

Returns: numpy array where row is alias ID and columns are EID

forward(*alias_indices*)

Model forward.

Parameters **alias_indices** – alias indices (B x M)

Returns: entity candidate EIDs (B x M x K)

get_alias_eid_priors(*alias_indices*)

Return the prior scores of the given alias_indices.

Parameters **alias_indices** – alias indices (B x M)

Returns: entity candidate normalized scores (B x M x K x 1)

classmethod prep(*data_config*, *entity_symbols*, *num_aliases_with_pad_and_unk*, *num_cands_K*)

Preps the alias to entity EID table.

Parameters

- **data_config** – data config
- **entity_symbols** – entity symbols
- **num_aliases_with_pad_and_unk** – number of aliases including pad and unk
- **num_cands_K** – number of candidates per alias (aka K)

Returns: torch Tensor of the alias to EID table, save pt file

training: bool

bootleg.layers.bert_encoder module

BERT encoder.

class bootleg.layers.bert_encoder.**Encoder**(*transformer*, *out_dim*)

Bases: torch.nn.modules.module.Module

Encoder module.

Return the CLS token of Transformer.

Parameters

- **transformer** – transformer
- **out_dim** – out dimension to project to

forward(*token_ids*, *segment_ids=None*, *attention_mask=None*)

BERT Encoder forward.

training: bool

Module contents

Layer init.

bootleg.slicing package

Submodules

bootleg.slicing.slice_dataset module

Bootleg slice dataset.

class bootleg.slicing.slice_dataset.**BootlegSliceDataset**(*main_args*, *dataset*, *use_weak_label*, *entity_symbols*, *dataset_threads*, *split='train'*)

Bases: object

Slice dataset class.

Our dataset class for holding data slices (or subpopulations).

Each mention can be part of 0 or more slices. When running eval, we use the SliceDataset to determine which mentions are part of what slices. Importantly, although the model "sees" all mentions, only GOLD anchor links are evaluated for eval (splits of test/dev).

Parameters

- **main_args** – main arguments
- **dataset** – dataset file
- **use_weak_label** – whether to use weak labeling or not
- **entity_symbols** – entity symbols
- **dataset_threads** – number of processes to use
- **split** – data split

classmethod build_data_dict(*save_dataset_name*, *storage*)

Build the slice dataset from saved file.

Loads the memmap slice dataset and create a mapping from sentence index to row index.

Parameters

- **save_dataset_name** – saved memmap file name
- **storage** – storage type of memmap file

Returns: numpy memmap data, Dict of sentence index to row in data

contains_sentidx(*sent_idx*)

Return true if the sentence index is in the dataset.

Parameters **sent_idx** – sentence index

Returns: bool whether in dataset or not

get_slice_incidence_arr(*sent_idx*, *alias_orig_list_pos*)

Get slice incident array.

Given the sentence index and the list of aliases to get slice indices for (may have -1 indicating no alias), return a dictionary of slice_name -> 0/1 incidence array of if each alias in alias_orig_list_pos was in the slice or not (-1 for no alias).

Parameters

- **sent_idx** – sentence index
- **alias_orig_list_pos** – list of alias positions in input data list (due to sentence splitting, aliases may be split up)

Returns: Dict of slice name -> 0/1 incidence array

class bootleg.slicing.slice_dataset.**InputExample**(*sent_idx*, *subslice_idx*, *anchor*, *num_alias2pred*, *slices*)

Bases: object

A single training/test example.

classmethod **from_dict**(*in_dict*)

Create object from dictionary.

to_dict()

Turn object to dictionary.

class bootleg.slicing.slice_dataset.**InputFeatures**(*sent_idx*, *subslice_idx*, *alias_slice_incidence*, *alias2pred_probs*)

Bases: object

A single set of features of data.

classmethod **from_dict**(*in_dict*)

Create object from dictionary.

to_dict()

Object to dictionary.

bootleg.slicing.slice_dataset.**convert_examples_to_features_and_save**(*meta_file*, *dataset_threads*, *slice_names*, *save_dataset_name*, *storage*)

Convert the prepped examples into input features.

Saves in memmap files. These are used in the __get_item__ method.

Parameters

- **meta_file** – metadata file where input file paths are saved
- **dataset_threads** – number of threads
- **slice_names** – list of slice names to evaluation on
- **save_dataset_name** – data file name to save
- **storage** – data storage type (for memmap)

bootleg.slicing.slice_dataset.**convert_examples_to_features_and_save_hlp**(*input_dict*)

Convert to features helper.

bootleg.slicing.slice_dataset.**convert_examples_to_features_and_save_initializer**(*save_dataset_name*, *storage*)

Convert to features multiprocessing initializer.

bootleg.slicing.slice_dataset.**convert_examples_to_features_and_save_single**(*input_dict*, *mmap_file*)

Convert examples to features multiprocessing helper.

bootleg.slicing.slice_dataset.**create_examples**(*dataset*, *create_ex_indir*, *create_ex_outdir*, *meta_file*, *data_config*, *dataset_threads*, *slice_names*, *use_weak_label*, *split*)

Create examples from the raw input data.

Parameters

- **dataset** – dataset file
- **create_ex_indir** – temporary directory where input files are stored
- **create_ex_outdir** – temporary directory to store output files from method
- **meta_file** – metadata file to save the file names/paths for the next step in prep pipeline
- **data_config** – data config
- **dataset_threads** – number of threads
- **slice_names** – list of slices to evaluate on
- **use_weak_label** – whether to use weak labeling or not
- **split** – data split

bootleg.slicing.slice_dataset.**create_examples_hlp**(*args*)

Create examples wrapper helper.

bootleg.slicing.slice_dataset.**create_examples_initializer**(*data_config*, *slice_names*, *use_weak_label*, *split*, *train_in_candidates*)

Create example multiprocessing initialiezr.

bootleg.slicing.slice_dataset.**create_examples_single**(*in_file_name*, *in_file_lines*, *out_file_name*, *constants_dict*)

Create examples multiprocessing helper.

bootleg.slicing.slice_dataset.**get_slice_values**(*slice_names*, *line*)

Results a dictionary of all slice values for an input example.

Any mention with a slice value of > 0.5 gets assigned that slice. If some slices are missing from the input, we assign all mentions as not being in that slice (getting a 0 label value). We also check that slices are formatted correctly.

Parameters

- **slice_names** – slice names to evaluate on
- **line** – input data json line

Returns: Dict of slice name to alias index string to float value of if mention is in a slice or not.

Module contents

Slicing initializer.

bootleg.symbols package

Submodules

bootleg.symbols.constants module

Constants.

bootleg.symbols.constants.**check_qid_exists**(*func*)

Check QID exists.

bootleg.symbols.constants.**edit_op**(*func*)

Edit op.

bootleg.symbols.entity_profile module

Entity profile.

class bootleg.symbols.entity_profile.**EntityObj**(*, *entity_id: str*, *mentions: List[Tuple[str, float]]*, *title: str*, *description: str*, *types: Dict[str, List[str]] = None*, *relations: List[Dict[str, str]] = None*)

Bases: pydantic.main.BaseModel

Base entity object class to check types.

description: str

entity_id: str

mentions: List[Tuple[str, float]]

relations: Optional[List[Dict[str, str]]]

title: str

types: Optional[Dict[str, List[str]]]

class bootleg.symbols.entity_profile.**EntityProfile**(*entity_symbols*, *type_systems=None*, *kg_symbols=None*, *edit_mode=False*, *verbose=False*)

Bases: object

Entity Profile object to handle and manage entity, type, and KG metadata.

add_entity(*entity_obj*)

Add entity to our dump.

Parameters **entity_obj** – JSON object of entity metadata

add_mention(*qid: str*, *mention: str*, *score: float*)

Add the mention with its score to the QID.

Parameters

- **qid** – QID
- **mention** – mention
- **score** – score

add_relation(*qid*, *relation*, *qid2*)

Add the relation triple.

Parameters

- **qid** – head QID
- **relation** – relation
- **qid2** – tail QID

add_type(*qid*, *type*, *type_system*)

Add type to QID in for the given type system.

Parameters

- **qid** – QID
- **type** – type name
- **type_system** – type system

get_all_mentions()

Return list of all mentions.

Returns: List of strings

get_all_qids()

Return all entity QIDs.

Returns: List of strings

get_all_types(*type_system*)

Return list of all type names for a type system.

Parameters **type_system** – type system

Returns: List of strings

get_all_typesystems()

Return list of all type systems.

Returns: List of strings

get_desc(*qid*)

Get the description of an entity QID.

Parameters **qid** – entity QID

Returns: string

`get_eid`(*qid*)

Get the entity EID (internal number) of an entity QID.

Parameters `qid` – entity QID

Returns: integer

`get_entities_of_type`(*typename*, *type_system*)

Get all entities of type `typename` for type system `type_system`.

Parameters

- `typename` – type name
- `type_system` – type system

Returns: List of QIDs

`get_mentions`(*qid*)

Get the mentions for the QID.

Parameters `qid` – QID

Returns: List of mentions

`get_mentions_with_scores`(*qid*)

Get the mentions with thier scores associated with the QID.

Parameters `qid` – QID

Returns: List of tuples [mention, score]

`get_qid_cands`(*mention*)

Get the entity QID candidates of the mention.

Parameters `mention` – mention

Returns: List of QIDs

`get_qid_count_cands`(*mention*)

Get the entity QID candidates with their scores of the mention.

Parameters `mention` – mention

Returns: List of tuples [QID, score]

`get_relations_between`(*qid*, *qid2*)

Check if two QIDs are connected in KG and returns their relation.

Parameters

- `qid` – QID one
- `qid2` – QID two

Returns: string relation or None

`get_relations_tails_for_qid`(*qid*)

Get dict of relation to tail qids for given qid.

Parameters `qid` – QID

Returns: Dict relation to list of tail qids for that relation

`get_title`(*qid*)

Get the title of an entity QID.

Parameters **qid** – entity QID

Returns: string

`get_type_typeid`(*type*, *type_system*)

Get the type type id for the type of the `type_system` system.

Parameters

- **type** – type
- **type_system** – type system

Returns: type id

`get_types`(*qid*, *type_system*)

Get the type names associated with the given QID for the `type_system` system.

Parameters

- **qid** – QID
- **type_system** – type system

Returns: list of typename strings

`classmethod load_from_cache`(*load_dir*, *edit_mode=False*, *verbose=False*, *no_kg=False*, *no_type=False*, *type_systems_to_load=None*)

Load a pre-saved profile.

Parameters

- **load_dir** – load directory
- **edit_mode** – edit mode flag, default False
- **verbose** – verbose flag, default False
- **no_kg** – load kg or not flag, default False
- **no_type** – load types or not flag, default False. If True, this will ignore type_systems_to_load.
- **type_systems_to_load** – list of type systems to load, default is None which means all types systems

Returns: entity profile object

`classmethod load_from_jsonl`(*profile_file*, *max_candidates=30*, *max_types=10*, *max_kg_connections=100*, *edit_mode=False*)

Load an entity profile from the raw jsonl file.

Each line is a JSON object with entity metadata.

Example object:

```
{
    "entity_id": "C000",
    "mentions": [["dog", 10.0], ["dogg", 7.0], ["animal", 4.0]],
    "title": "Dog",
    "types": {"hyena": ["animal"], "wiki": ["dog"]},
```

(continues on next page)

(continued from previous page)

```
    "relations": [
        {"relation": "sibling", "object": "Q345"},
        {"relation": "sibling", "object": "Q567"},
    ],
}
```

Parameters

- **profile_file** – file where jsonl data lives
- **max_candidates** – maximum entity candidates
- **max_types** – maximum types per entity
- **max_kg_connections** – maximum KG connections per entity
- **edit_mode** – edit mode

Returns: entity profile object

mention_exists(*mention*)

Check if mention exists.

Parameters **mention** – mention

Returns: Boolean

property num_entities_with_pad_and_nocand

Get the number of entities including a PAD and UNK entity.

Returns: integer

prune_to_entities(*entities_to_keep*)

Remove all entities except those in `entities_to_keep`.

Parameters **entities_to_keep** – List or Set of entities to keep

qid_exists(*qid*)

Check if QID exists.

Parameters **qid** – entity QID

Returns: Boolean

reidentify_entity(*qid*, *new_qid*)

Rename `qid` to `new_qid`.

Parameters

- **qid** – old QID
- **new_qid** – new QID

remove_mention(*qid*, *mention*)

Remove the mention from being associated with the QID.

Parameters

- **qid** – QID
- **mention** – mention

remove_relation(*qid*, *relation*, *qid2*)

Remove the relation triple.

Parameters

- **qid** – head QID
- **relation** – relation
- **qid2** – tail QID

remove_type(*qid*, *type*, *type_system*)

Remove the type from QID in the given type system.

Parameters

- **qid** – QID
- **type** – type to remove
- **type_system** – type system

save(*save_dir*)

Save the profile.

Parameters **save_dir** – save directory

save_to_jsonl(*profile_file*)

Dump the entity dump to jsonl format.

Parameters **profile_file** – file to save the data

update_entity(*entity_obj*)

Update the metadata associated with the entity.

The entity must already be in our dump to be updated.

Parameters **entity_obj** – JSON of entity metadata.

bootleg.symbols.entity_symbols module

Entity symbols.

class bootleg.symbols.entity_symbols.**EntitySymbols**(*alias2qids: Union[Dict[str, list], bootleg.utils.classes.nested_vocab_tries.TwoLayerVocabularyScoreTrie]*, *qid2title: Dict[str, str]*, *qid2desc: Optional[Dict[str, str]] = None*, *qid2eid: Optional[bootleg.utils.classes.nested_vocab_tries.VocabularyTrie] = None*, *alias2id: Optional[bootleg.utils.classes.nested_vocab_tries.VocabularyTrie] = None*, *max_candidates: int = 30*, *alias_cand_map_dir: str = 'alias2qids'*, *alias_idx_dir: str = 'alias2id'*, *edit_mode: Optional[bool] = False*, *verbose: Optional[bool] = False*)

Bases: object

Entity Symbols class for managing entity metadata.

`add_entity`(*qid*, *mentions*, *title*, *desc=''*)

Add entity QID to our mappings with its mentions and title.

Parameters

- **qid** – QID
- **mentions** – List of tuples [mention, score]
- **title** – title
- **desc** – description

`add_mention`(*qid: str*, *mention: str*, *score: float*)

Add mention to QID with the associated score.

The mention already exists, error thrown to call `set_score` instead. If there are already max candidates to that mention, the last candidate of the mention is removed in place of QID.

Parameters

- **qid** – QID
- **mention** – mention
- **score** – score

`alias_exists`(*alias*)

Check alias existance.

Parameters **alias** – alias string

Returns: boolean

`get_alias2qids_dict`()

Get the alias2qids mapping.

Key is alias, value is list of candidate tuple of length two of [QID, sort_value].

Returns: Dict alias2qids mapping

`get_alias_from_idx`(*alias_idx*)

Get the alias from the numeric index.

Parameters **alias_idx** – alias numeric index

Returns: alias string

`get_alias_idx`(*alias*)

Get the numeric index of an alias.

Parameters **alias** – alias

Returns: integer representation of alias

`get_all_alias_vocabtrie`()

Get a trie of all aliases.

Returns: Vocab trie of all aliases.

`get_all_aliases`()

Get all aliases.

Returns: Dict_keys of all aliases

`get_all_qids()`
Get all QIDs.

Returns: Dict_keys of all QIDs

`get_all_titles()`
Get all QID titles.

Returns: Dict_values of all titles

`get_desc(id)`
Get description for QID.

Parameters `id` – QID string

Returns: title string

`get_eid(id)`
Get the QID for the EID.

Parameters `id` – EID int

Returns: QID string

`get_eid_cands(alias, max_cand_pad=False)`
Get the EID candidates for an alias.

Parameters

- `alias` – alias
- `max_cand_pad` – whether to pad with -1 or not if fewer than max_candidates candidates

Returns: List of EID ints

`get_mentions(qid)`
Get the mentions for the QID.

Parameters `qid` – QID

Returns: List of mentions

`get_mentions_with_scores(qid)`
Get the mentions and the associated score for the QID.

Parameters `qid` – QID

Returns: List of tuples [mention, score]

`get_qid(id)`
Get the QID associated with EID.

Parameters `id` – EID

Returns: QID string

`get_qid2eid_dict()`
Get the qid2eid mapping.

Returns: Dict qid2eid mapping

`get_qid2title_dict()`
Get the qid2title mapping.

Returns: Dict qid2title mapping

`get_qid_cands`(*alias*, *max_cand_pad=False*)

Get the QID candidates for an alias.

Parameters

- **alias** – alias
- **max_cand_pad** – whether to pad with '-1' or not if fewer than max_candidates candidates

Returns: List of QID strings

`get_qid_count_cands`(*alias*, *max_cand_pad=False*)

Get the [QID, sort_value] candidates for an alias.

Parameters

- **alias** – alias
- **max_cand_pad** – whether to pad with ['-1',-1] or not if fewer than max_candidates candidates

Returns: List of [QID, sort_value]

`get_title`(*id*)

Get title for QID.

Parameters **id** – QID string

Returns: title string

`classmethod load_from_cache`(*load_dir*, *alias_cand_map_dir='alias2qids'*, *alias_idx_dir='alias2id'*, *edit_mode=False*, *verbose=False*)

Load entity symbols from load_dir.

Parameters

- **load_dir** – directory to load from
- **alias_cand_map_dir** – alias2qid directory
- **alias_idx_dir** – alias2id directory
- **edit_mode** – edit mode flag
- **verbose** – verbose flag

`prune_to_entities`(*entities_to_keep*)

Remove all entities except those in `entities_to_keep`.

Parameters **entities_to_keep** – Set of entities to keep

`qid_exists`(*qid*)

Check QID existance.

Parameters **alias** – QID string

Returns: boolean

`reidentify_entity`(*old_qid*, *new_qid*)

Rename `old_qid` to `new_qid`.

Parameters

- **old_qid** – old QID
- **new_qid** – new QID

remove_mention(*qid*, *mention*)

Remove the mention from those associated with the QID.

Parameters

- **qid** – QID
- **mention** – mention to remove

save(*save_dir*)

Dump the entity symbols.

Parameters **save_dir** – directory string to save

set_desc(*qid: str*, *desc: str*)

Set the description for a QID.

Parameters

- **qid** – QID
- **desc** – description

set_score(*qid: str*, *mention: str*, *score: float*)

Change the mention QID score and resorts candidates.

Highest score is first.

Parameters

- **qid** – QID
- **mention** – mention
- **score** – score

set_title(*qid: str*, *title: str*)

Set the title for a QID.

Parameters

- **qid** – QID
- **title** – title

bootleg.symbols.kg_symbols module

KG symbols class.

class bootleg.symbols.kg_symbols.**KGSymbols**(*qid2relations: Union[Dict[str, Dict[str, List[str]]], bootleg.utils.classes.nested_vocab_tries.ThreeLayerVocabularyTrie]*, *max_connections: Optional[int] = 50*, *edit_mode: Optional[bool] = False*, *verbose: Optional[bool] = False*)

Bases: object

KG Symbols class for managing KG metadata.

add_entity(*qid*, *relation_dict*)

Add a new entity to our relation mapping.

Parameters

- **qid** – QID

- **relation_dict** – dictionary of relation -> list of connected other_qids by relation

add_relation(*qid*, *relation*, *qid2*)

Add a relationship triple to our mapping.

If the QID already has max connection through `relation`, the last `other_qid` is removed and replaced by `qid2`.

Parameters

- **qid** – head entity QID
- **relation** – relation
- **qid2** – tail entity QID:

get_all_relations()

Get all relations in our KG mapping.

Returns: Set

get_qid2relations_dict()

Return a dictionary form of the relation to qid mappings object.

Returns: Dict of relation to head qid to list of tail qids

get_relations_between(*qid1*, *qid2*)

Check if two QIDs are connected in KG and returns the relations between then.

Parameters

- **qid1** – QID one
- **qid2** – QID two

Returns: string relation or empty set

get_relations_tails_for_qid(*qid*)

Get dict of relation to tail qids for given qid.

Parameters **qid** – QID

Returns: Dict relation to list of tail qids for that relation

classmethod load_from_cache(*load_dir*, *prefix=''*, *edit_mode=False*, *verbose=False*)

Load type symbols from load_dir.

Parameters

- **load_dir** – directory to load from
- **prefix** – prefix to add to beginning to file
- **edit_mode** – edit mode
- **verbose** – verbose flag

Returns: TypeSymbols

prune_to_entities(*entities_to_keep*)

Remove all entities except those in `entities_to_keep`.

Parameters **entities_to_keep** – Set of entities to keep

reidentify_entity(*old_qid*, *new_qid*)

Rename `old_qid` to `new_qid`.

Parameters

- **old_qid** – old QID
- **new_qid** – new QID

remove_relation(*qid*, *relation*, *qid2*)

Remove a relation triple from our mapping.

Parameters

- **qid** – head entity QID
- **relation** – relation
- **qid2** – tail entity QID

save(*save_dir*, *prefix=''*)

Dump the kg symbols.

Parameters

- **save_dir** – directory string to save
- **prefix** – prefix to add to beginning to file

bootleg.symbols.type_symbols module

Type symbols class.

class bootleg.symbols.type_symbols.**TypeSymbols**(*qid2typenames: Union[Dict[str, List[str]], bootleg.utils.classes.nested_vocab_tries.TwoLayerVocabularyScoreTrie]*, *max_types: Optional[int] = 10*, *edit_mode: Optional[bool] = False*, *verbose: Optional[bool] = False*)

Bases: `object`

Type Symbols class for managing type metadata.

add_entity(*qid*, *types*)

Add an entity QID with its types to our mappings.

Parameters

- **qid** – QID
- **types** – list of type names

add_type(*qid*, *typename*)

Add the type to the QID.

If the QID already has maximum types, the last type is removed and replaced by `typename`.

Parameters

- **qid** – QID
- **typename** – type name

`get_all_types()`

Return all typenames.

`get_entities_of_type(typename)`

Get all entity QIDs of type `typename`.

Parameters `typename` – typename

Returns: List

`get_qid2typename_dict()`

Return dictionary of qid to typenames.

Returns: Dict of QID to list of typenames.

`get_types(qid)`

Get the type names associated with the given QID.

Parameters `qid` – QID

Returns: list of typename strings

`classmethod load_from_cache(load_dir, prefix='', edit_mode=False, verbose=False)`

Load type symbols from load_dir.

Parameters

- `load_dir` – directory to load from
- `prefix` – prefix to add to beginning to file
- `edit_mode` – edit mode flag
- `verbose` – verbose flag

Returns: TypeSymbols

`prune_to_entities(entities_to_keep)`

Remove all entities except those in `entities_to_keep`.

Parameters `entities_to_keep` – Set of entities to keep

`reidentify_entity(old_qid, new_qid)`

Rename `old_qid` to `new_qid`.

Parameters

- `old_qid` – old QID
- `new_qid` – new QID

`remove_type(qid, typename)`

Remove the type from the QID.

Parameters

- `qid` – QID
- `typename` – type name to remove

`save(save_dir, prefix='')`

Dump the type symbols.

Parameters

- `save_dir` – directory string to save

- **prefix** – prefix to add to beginning to file

Module contents

Symbols init.

bootleg.tasks package

Submodules

bootleg.tasks.entity_gen_task module

Entity gen task definitions.

class bootleg.tasks.entity_gen_task.**EntityGenOutput**(*normalize*)

Bases: object

Entity gen for output.

entity_output_func(*intermediate_output_dict*)

Entity output func.

bootleg.tasks.entity_gen_task.**create_task**(*args*, *len_context_tok*)

Return an EmmentalTask for entity encoder only.

Parameters

- **args** – args
- **len_context_tok** – number of tokens in the tokenizer

Returns: EmmentalTask for entity embedding extraction

bootleg.tasks.ned_task module

NED task definitions.

class bootleg.tasks.ned_task.**DisambigLoss**(*normalize*, *temperature*, *entity_encoder_key*)

Bases: object

Disambiguation loss.

batch_cands_disambig_loss(*intermediate_output_dict*, *Y*)

Return the entity disambiguation loss on prediction heads.

Parameters

- **intermediate_output_dict** – output dict from the Emmental task flor
- **Y** – gold labels

Returns: loss

batch_cands_disambig_output(*intermediate_output_dict*)

Return the probs for a task in Emmental.

Parameters **intermediate_output_dict** – output dict from Emmental task flow

Returns: NED probabilities for candidates (B x M x K)

disambig_loss(*intermediate_output_dict*, *Y*)

Return the entity disambiguation loss on prediction heads.

Parameters

- **intermediate_output_dict** – output dict from the Emmental task flor
- **Y** – gold labels

Returns: loss

disambig_output(*intermediate_output_dict*)

Return the probs for a task in Emmental.

Parameters **intermediate_output_dict** – output dict from Emmental task flow

Returns: NED probabilities for candidates (B x M x K)

bootleg.tasks.ned_task.**create_task**(*args*, *use_batch_cands*, *len_context_tok*, *slice_datasets=None*, *entity_emb_file=None*)

Return an EmmentalTask for named entity disambiguation (NED).

Parameters

- **args** – args
- **use_batch_cands** – use batch candidates for training
- **len_context_tok** – length of the context tokenizer
- **slice_datasets** – slice datasets used in scorer (default None)
- **entity_emb_file** – file for pretrained entity embeddings - used for EVAL only

Returns: EmmentalTask for NED

Module contents

Task init.

bootleg.utils package

Subpackages

bootleg.utils.classes package

Submodules

bootleg.utils.classes.aliasmention_trie module

bootleg.utils.classes.comment_json module

JSON with comments class.

An example of how to remove comments and trailing commas from JSON before parsing. You only need the two functions below, *remove_comments()* and *remove_trailing_commas()* to accomplish this. This script serves as an example of how to use them but feel free to just copy & paste them into your own code/projects. Usage:: json_cleaner.py some_file.json Alternatively, you can pipe JSON into this script and it'll clean it up:: cat some_file.json | json_cleaner.py Why would you do this? So you can have human-generated .json files (say, for configuration) that include comments and, really, who wants to deal with catching all those trailing commas that might be present? Here's an example of a file that will be successfully cleaned up and JSON-parseable:

FYI: This script will also pretty-print the JSON after it's cleaned up (if using it from the command line) with an indentation level of 4 (that is, four spaces).

bootleg.utils.classes.comment_json.**remove_comments**(*json_like*)

Remove C-style comments from *json_like* and returns the result.

Example:

```
>>> test_json = '''\
{
    "foo": "bar", // This is a single-line comment
    "baz": "blah" /* Multi-line
    Comment */
}'''
>>> remove_comments('{"foo":"bar","baz":"blah",}')
'{\n    "foo":"bar",\n    "baz":"blah"\n}'
```

bootleg.utils.classes.comment_json.**remove_trailing_commas**(*json_like*)

Remove trailing commas from *json_like* and returns the result.

Example:

```
>>> remove_trailing_commas('{"foo":"bar","baz":["blah",],}')
'{"foo":"bar","baz":["blah"]}'
```

bootleg.utils.classes.dotted_dict module

Dotted dict class.

class bootleg.utils.classes.dotted_dict.**DottedDict**(**args*, ***kwargs*)

Bases: `dict`

Dotted dictionary.

Override for the dict object to allow referencing of keys as attributes, i.e. dict.key.

copy()

Ensure copy object is DottedDict, not dict.

to_dict()

Recursive conversion back to dict.

class bootleg.utils.classes.dotted_dict.**PreserveKeysDottedDict**(**args*, ***kwargs*)

Bases: `dict`

Override auto correction of key names to safe attr names.

Can result in errors when using attr name resolution.

copy()

Ensure copy object is DottedDict, not dict.

to_dict()

Recursive conversion back to dict.

bootleg.utils.classes.dotted_dict.**create_bool_dotted_dict**(*d_dict*)

Create boolean Dotted Dict.

bootleg.utils.classes.dotted_dict.**is_json**(*value*)

Return true if is json.

bootleg.utils.classes.dotted_dict.**is_number**(*s*)

Return True is string is a number.

Module contents

Classes init.

bootleg.utils.parser package

Submodules

bootleg.utils.parser.bootleg_args module

Bootleg default configuration parameters.

In the json file, everything is a string or number. In this python file, if the default is a boolean, it will be parsed as such. If the default is a dictionary, True and False strings will become booleans. Otherwise they will stay string.

bootleg.utils.parser.emm_parse_args module

Overrides the Emmental parse_args.

bootleg.utils.parser.emm_parse_args.**parse_args**(*parser: Optional[argparse.ArgumentParser] = None*) → Tuple[argparse.ArgumentParser, Dict]

Parse args.

Overrides the default Emmental parser to add the "emmental." level to the parser so we can parse it correctly with the Bootleg config.

Parameters **parser** – Argument parser object, defaults to None.

Returns The updated argument parser object.

bootleg.utils.parser.emm_parse_args.**parse_args_to_config**(*args:* bootleg.utils.classes.dotted_dict.DottedDict) → Dict[str, Any]

Parse the Emmental arguments to config dict.

Parameters **args** – parsed namespace from argument parser.

Returns: Emmental config dict.

bootleg.utils.parser.parser_utils module

Bootleg parser utils.

Parses a Booleg input config into a DottedDict of config values (with defaults filled in) for running a model.

bootleg.utils.parser.parser_utils.**add_nested_flags_from_config**(*parser*, *config_dict*, *parser_hierarchy*, *prefix*)

Add flags from config file, keeping the hierarchy the same.

When a lower level is needed, parser.add_argument_group is called. Note, we append the parent key to the –param option (via prefix parameter).

Parameters

- **parser** – arg parser to add options to
- **config_dict** – raw config dictionary
- **parser_hierarchy** – Dict to add parser hierarhcy to
- **prefix** – prefix to add to arg parser

bootleg.utils.parser.parser_utils.**flatten_nested_args_for_parser**(*args*, *new_args*, *groups*, *prefix*)

Flatten all parameters to be passed as a single list to arg parse.

bootleg.utils.parser.parser_utils.**get_boot_config**(*config*, *parser_hierarchy=None*, *parser=None*, *unknown=None*)

Return a parsed Bootleg config from config.

Config can be a path to a config file or an already loaded dictionary.

The high level work flow

1. Reads Bootleg default config (config_args) and addes params to a arg parser, flattening all hierarchical values into "." values

   E.g., data_config -> word_embeddings -> layers becomes –data_config.word_embedding.layers

2. Flattens the given config values into the "." format
3. Adds any unknown values from the first arg parser that parses the config script. Allows the user to add –data_config.word_embedding.layers to command line that overwrite values in file
4. Parses the flattened args w.r.t the arg parser
5. Reconstruct the args back into their hierarchical form

Parameters

- **config** – model specific config
- **parser_hierarchy** – Dict of hierarchy of config (or None)
- **parser** – arg parser (or None)
- **unknown** – unknown arg values passed from command line to be added to config and overwrite values in file

bootleg.utils.parser.parser_utils.**is_json**(*value*)

Return True if json.

bootleg.utils.parser.parser_utils.**is_number**(*s*)

Return True is string is a number.

bootleg.utils.parser.parser_utils.**load_commented_json_file**(*file*)

Load commented json file.

bootleg.utils.parser.parser_utils.**merge_configs**(*config_l*, *config_r*, *new_config=None*)

Merge two dotted dict configs.

bootleg.utils.parser.parser_utils.**or_none**(*default*)

Return or None function.

bootleg.utils.parser.parser_utils.**parse_boot_and_emm_args**(*config_script*, *unknown=None*)

Merge the Emmental config with the Bootleg config.

As we have an emmental: ... level in our config for emmental commands, we need to parse those with the Emmental parser and then merge the Bootleg only config values with the Emmental ones.

Parameters

- **config_script** – config script for Bootleg and Emmental args
- **unknown** – unknown arg values passed from command line to overwrite file values

Returns: parsed merged Bootleg and Emmental config

bootleg.utils.parser.parser_utils.**reconstructed_nested_args**(*args*, *names*, *parser_hierarchy*, *prefix*)

Reconstruct the arguments and pass them to the necessary subparsers.

bootleg.utils.parser.parser_utils.**recursive_keys**(*dictionary*)

Recursively yields all keys of dict.

Module contents

Parser init.

bootleg.utils.preprocessing package

Submodules

bootleg.utils.preprocessing.compute_statistics module

Compute statistics over data.

Helper file for computing various statistics over our data such as mention frequency, mention text frequency in the data (even if not labeled as an anchor), ...

etc.

bootleg.utils.preprocessing.compute_statistics.**chunk_text_data**(*input_src*, *chunk_files*, *chunk_size*, *num_lines*)

Chunk text data.

bootleg.utils.preprocessing.compute_statistics.**compute_histograms**(*save_dir*, *entity_symbols*)

Compute histogram.

bootleg.utils.preprocessing.compute_statistics.**compute_occurrences**(*save_dir*, *data_file*, *entity_dump*, *lower*, *strip*, *num_workers=8*)

Compute statistics.

bootleg.utils.preprocessing.compute_statistics.**compute_occurrences_single**(*args*, *max_alias_len=6*)

Compute statistics single process.

bootleg.utils.preprocessing.compute_statistics.**get_num_lines**(*input_src*)

Get number of lines.

bootleg.utils.preprocessing.compute_statistics.**main**()

Run.

bootleg.utils.preprocessing.compute_statistics.**parse_args**()

Parse args.

bootleg.utils.preprocessing.count_body_part_size module

bootleg.utils.preprocessing.gen_alias_cand_map module

bootleg.utils.preprocessing.gen_entity_mappings module

bootleg.utils.preprocessing.get_train_qid_counts module

Compute QID counts.

Helper function that computes a dictionary of QID -> count in training data.

If a QID is not in this dictionary, it has a count of zero.

bootleg.utils.preprocessing.get_train_qid_counts.**get_counts**(*num_processes*, *file*)

Get true anchor slice counts.

bootleg.utils.preprocessing.get_train_qid_counts.**get_counts_hlp**(*line*)

Get count helper.

bootleg.utils.preprocessing.get_train_qid_counts.**main**()

Run.

bootleg.utils.preprocessing.get_train_qid_counts.**parse_args**()

Parse args.

bootleg.utils.preprocessing.sample_eval_data module

Sample eval data.

This will sample a jsonl train or eval data based on the slices in the data. This is useful for subsampling a smaller eval dataset.py.

The output of this file is a files with a subset of sentences from the input file samples such that for each slice in –args.slice, a minimum of args.min_sample_size mentions are in the slice (if possible). Once that is satisfied, we sample to get approximately –args.sample_perc of mentions from each slice.

bootleg.utils.preprocessing.sample_eval_data.**get_slice_stats**(*num_processes*, *file*)

Get true anchor slice counts.

bootleg.utils.preprocessing.sample_eval_data.**get_slice_stats_hlp**(*args*)

Get slice count helper.

bootleg.utils.preprocessing.sample_eval_data.**main**()

Run.

bootleg.utils.preprocessing.sample_eval_data.**parse_args**()

Parse args.

Module contents

Preprocessing init.

Submodules

bootleg.utils.data_utils module

Bootleg data utils.

bootleg.utils.data_utils.**add_special_tokens**(*tokenizer*)

Add special tokens.

Parameters

- **tokenizer** – tokenizer
- **data_config** – data config
- **entitysymbols** – entity symbols

bootleg.utils.data_utils.**correct_not_augmented_dict_values**(*gold*, *dict_values*)

Correct gold label dict values in data prep.

Modifies the dict_values to only contain those mentions that are gold labels. The new dictionary has the alias indices be corrected to start at 0 and end at the number of gold mentions.

Parameters

- **gold** – List of T/F values if mention is gold label or not
- **dict_values** – Dict of slice_name -> Dict[alias_idx] -> slice probability

Returns: adjusted dict_values such that only gold = True aliases are kept (dict is reindexed to start at 0)

bootleg.utils.data_utils.**generate_slice_name**(*data_args*, *slice_names*, *use_weak_label*, *dataset*)

Generate name for slice datasets, taking into account the config eval slices.

Parameters

- **data_args** – data args
- **slice_names** – slice names
- **use_weak_label** – if using weak labels or not
- **dataset** – dataset name

Returns: dataset name for saving slice data

bootleg.utils.data_utils.**get_chunk_dir**(*prep_dir*)

Get directory for saving data chunks.

Parameters **prep_dir** – prep directory

Returns: directory path

bootleg.utils.data_utils.**get_data_prep_dir**(*data_config*)

Get data prep directory for saving prep files.

Parameters **data_config** – data config

Returns: directory path

bootleg.utils.data_utils.**get_emb_prep_dir**(*data_config*)

Get embedding prep directory for saving prep files.

Parameters **data_config** – data config

Returns: directory path

bootleg.utils.data_utils.**get_eval_slices**(*eval_slices*)

Get eval slices in data prep.

Given input eval slices (passed in config), ensure FINAL_LOSS is in the eval slices. FINAL_LOSS gives overall metrics.

Parameters **eval_slices** – list of input eval slices

Returns: list of eval slices to use in the model

bootleg.utils.data_utils.**get_save_data_folder**(*data_args*, *use_weak_label*, *dataset*)

Get save data folder for the prepped data.

Parameters

- **data_args** – data config
- **use_weak_label** – whether to use weak labelling or not
- **dataset** – dataset name

Returns: folder string path

bootleg.utils.data_utils.**get_save_data_folder_candgen**(*data_args*, *use_weak_label*, *dataset*)

Give save data folder for the prepped data.

Parameters

- **data_args** – data config
- **use_weak_label** – whether to use weak labelling or not
- **dataset** – dataset name

Returns: folder string path

bootleg.utils.data_utils.**read_in_akas**(*entitysymbols*)

Read in alias to QID mappings and generates a QID to list of alternate names.

Parameters **entitysymbols** – entity symbols

Returns: dictionary of QID to type names

bootleg.utils.eval_utils module

Bootleg eval utils.

bootleg.utils.eval_utils.**batched_pred_iter**(*model*, *dataloader*, *dump_preds_accumulation_steps*, *sent_idx2num_mens*)

Predict from dataloader.

Predict from dataloader taking into account eval accumulation steps. Will yield a new prediction set after each set accumulation steps for writing out.

If a sentence or batch doesn't have any mentions, it will not be returned by this method.

Recall that we split up sentences that are too long to feed to the model. We use the sent_idx2num_mens dict to ensure we have full sentences evaluated before returning, otherwise we'll have incomplete sentences to merge together when dumping.

Parameters

- **model** – model
- **dataloader** – The dataloader to predict
- **dump_preds_accumulation_steps** – Number of eval steps to run before returning
- **sent_idx2num_mens** – list of sent index to number of mentions

Returns Iterator over result dict.

bootleg.utils.eval_utils.**check_and_create_alias_cand_trie**(*save_folder*, *entity_symbols*)

Create a mmap memory trie object for storing the alias-candidate mappings.

Parameters

- **save_folder** – save folder for alias trie
- **entity_symbols** – entity symbols

bootleg.utils.eval_utils.**collect_and_merge_results**(*unmerged_entity_emb_file*, *emb_file_config*, *config*, *sent_idx2num_mens*, *sent_idx2row*, *save_folder*, *entity_symbols*)

Merge mentions, filtering non-gold labels, and saves to file.

Parameters

- **unmerged_entity_emb_file** – memmap file from dump step
- **emb_file_config** – config file for loading memmap file
- **config** – model config
- **res_dict** – result dictionary from Emmental predict
- **sent_idx2num_mens** – Dict sentence idx to number of mentions
- **sent_idx2row** – Dict sentence idx to row of eval data
- **save_folder** – folder to save results
- **entity_symbols** – entity symbols

Returns: saved prediction file, total mentions seen

bootleg.utils.eval_utils.**dump_model_outputs**(*model*, *dataloader*, *config*, *sentidx2num_mentions*, *save_folder*, *entity_symbols*, *task_name*, *overwrite_data*)

Dump model outputs.

Parameters

- **model** – model
- **dataloader** – data loader
- **config** – config
- **sentidx2num_mentions** – Dict from sentence idx to number of mentions
- **save_folder** – save folder
- **entity_symbols** – entity symbols
- **task_name** – task name
- **overwrite_data** – overwrite saved mmap files

Returns: mmemp file name for saved outputs, dtype file name for loading memmap file

bootleg.utils.eval_utils.**get_emb_file**(*save_folder*)

Get the embedding numpy file for the batch.

Parameters **save_folder** – save folder

Returns: string

bootleg.utils.eval_utils.**get_eval_folder**(*file*)

Return eval folder for the given evaluation file.

Stored in log_path/filename/model_name.

Parameters **file** – eval file

Returns: eval folder

bootleg.utils.eval_utils.**get_result_file**(*save_folder*)

Get the jsonl label file for the batch.

Parameters **save_folder** – save folder

Returns: string

bootleg.utils.eval_utils.**get_sent_idx2num_mens**(*data_file*)

Get the map from sentence index to number of mentions and to data.

Used for calculating offsets and chunking file.

Parameters **data_file** – eval file

Returns: Dict of sentence index -> number of mention per sentence, Dict of sentence index -> input line

bootleg.utils.eval_utils.**get_sental2embid**(*merged_entity_emb_file*, *merged_storage_type*)

Get sent_idx, alias_idx mapping to emb idx for quick lookup.

Parameters

- **merged_entity_emb_file** – memmap file after merge sentences
- **merged_storage_type** – file storage type

Returns: Dict of f"{sent_idx}_{alias_idx}" -> index in merged_entity_emb_file

bootleg.utils.eval_utils.**map_aliases_to_candidates**(*train_in_candidates*, *max_candidates*, *alias_cand_map*, *aliases*)

Get list of QID candidates for each alias.

Parameters

- **train_in_candidates** – whether the model has a NC entity or not (assumes all gold QIDs are in candidate lists)
- **alias_cand_map** – alias -> candidate qids in dict or TwoLayerVocabularyScoreTrie format
- **aliases** – list of aliases

Returns: List of lists QIDs

bootleg.utils.eval_utils.**map_candidate_qids_to_eid**(*candidate_qids*, *qid2eid*)

Get list of EID candidates for each alias.

Parameters

- **candidate_qids** – list of list of candidate QIDs
- **qid2eid** – mapping of qid to entity id

Returns: List of lists EIDs

bootleg.utils.eval_utils.**masked_class_logsoftmax**(*pred*, *mask*, *dim=2*, *temp=1.0*, *zero_delta=1e-45*)

Masked logsoftmax.

Mask of 0/False means mask value (ignore it)

Parameters

- **pred** – input tensor
- **mask** – mask
- **dim** – softmax dimension
- **temp** – softmax temperature
- **zero_delta** – small value to add so that vector + (mask+zero_delta).log() is not Nan for all 0s

Returns: masked softmax tensor

bootleg.utils.eval_utils.**merge_subsentences**(*num_processes*, *subset_sent_idx2num_mens*, *cache_folder*, *to_save_file*, *to_save_storage*, *to_read_file*, *to_read_storage*)

Merge and flatten sentence over sub-sentences.

Flatten all sentences back together over sub-sentences; removing the PAD aliases from the data I.e., converts from sent_idx -> array of values to (sent_idx, alias_idx) -> value with varying numbers of aliases per sentence.

Parameters

- **num_processes** – number of processes
- **subset_sent_idx2num_mens** – Dict of sentence index to number of mentions for this batch
- **cache_folder** – cache directory
- **to_save_file** – memmap file to save results to
- **to_save_storage** – save file storage type
- **to_read_file** – memmap file to read predictions from

- **to_read_storage** – read file storage type

bootleg.utils.eval_utils.**merge_subsentences_hlp**(*args*)

Merge subsentences multiprocessing subprocess helper.

bootleg.utils.eval_utils.**merge_subsentences_initializer**(*to_write_file*, *to_write_storage*, *to_read_file*, *to_read_storage*, *sentidx2offset_file*)

Merge subsentences initializer for multiprocessing.

Parameters

- **to_write_file** – file to write
- **to_write_storage** – mmap storage type
- **to_read_file** – file to read
- **to_read_storage** – mmap storage type
- **sentidx2offset_file** – sentence index to offset in mmap data

bootleg.utils.eval_utils.**merge_subsentences_single**(*K*, *hidden_size*, *r_idx_set*, *filt_emb_data*, *full_pred_data*, *sentidx2offset*)

Merge subsentences single process.

Will flatted out the results from *full_pred_data* so each line of *filt_emb_data* is one alias prediction.

Parameters

- **K** – number candidates
- **hidden_size** – hidden size
- **r_idx_set** – batch result index
- **filt_emb_data** – mmap embedding file to write
- **full_pred_data** – mmap result file to read
- **sentidx2offset** – sentence to emb data offset

bootleg.utils.eval_utils.**write_data_labels**(*num_processes*, *merged_entity_emb_file*, *merged_storage_type*, *sent_idx2row*, *cache_folder*, *out_file*, *entity_dump*, *train_in_candidates*, *max_candidates*, *trie_candidate_map_folder=None*, *trie_qid2eid_file=None*)

Take the flattened data from merge_sentences and write out predictions.

Parameters

- **num_processes** – number of processes
- **merged_entity_emb_file** – input memmap file after merge sentences
- **merged_storage_type** – input file storage type
- **sent_idx2row** – Dict of sentence idx to row relevant to this subbatch
- **cache_folder** – folder to save temporary outputs
- **out_file** – final output file for predictions
- **entity_dump** – entity dump
- **train_in_candidates** – whether NC entities are not in candidate lists

- **max_candidates** – maximum number of candidates
- **trie_candidate_map_folder** – folder where trie of alias->candidate map is stored for parallel proccessing
- **trie_qid2eid_file** – file where trie of qid->eid map is stored for parallel proccessing

bootleg.utils.eval_utils.**write_data_labels_hlp**(*args*)

Write data labels multiprocess helper function.

bootleg.utils.eval_utils.**write_data_labels_initializer**(*merged_entity_emb_file*, *merged_storage_type*, *sental2embid_file*, *train_in_candidates*, *max_cands*, *trie_candidate_map_folder*, *trie_qid2eid_file*)

Write data labels multiprocessing initializer.

Parameters

- **merged_entity_emb_file** – flattened embedding input file
- **merged_storage_type** – mmap storage type
- **sental2embid_file** – sentence, alias -> embedding id mapping
- **train_in_candidates** – train in candidates flag
- **max_cands** – max candidates
- **trie_candidate_map_folder** – alias trie folder
- **trie_qid2eid_file** – qid to eid trie file

bootleg.utils.eval_utils.**write_data_labels_single**(*sentidx2row*, *output_file*, *filt_emb_data*, *sental2embid*, *alias_cand_map*, *qid2eid*, *train_in_cands*, *max_cands*)

Write data labels single subprocess function.

Will take the alias predictions and merge them back by sentence to be written out.

Parameters

- **sentidx2row** – sentence index to raw eval data row
- **output_file** – output file
- **filt_emb_data** – mmap embedding data (one prediction per row)
- **sental2embid** – sentence index, alias index -> embedding row id
- **alias_cand_map** – alias to candidate map
- **qid2eid** – qid to entity id map
- **train_in_cands** – training in candidates flag
- **max_cands** – maximum candidates

bootleg.utils.eval_utils.**write_disambig_metrics_to_csv**(*file_path*, *dictionary*)

Save disambiguation metrics in the dictionary to file_path.

Parameters

- **file_path** – file path
- **dictionary** – dictionary of scores (output of Emmental score)

bootleg.utils.model_utils module

Model utils.

bootleg.utils.model_utils.**count_parameters**(*model*, *requires_grad*, *logger*)

Count the number of parameters.

Parameters

- **model** – model to count
- **requires_grad** – whether to look at grad or no grad params
- **logger** – logger

bootleg.utils.model_utils.**get_max_candidates**(*entity_symbols*, *data_config*)

Get max candidates.

Returns the maximum number of candidates used in the model, taking into account train_in_candidates If train_in_canddiates is False, we add a NC entity candidate (for null candidate)

Parameters

- **entity_symbols** – entity symbols
- **data_config** – data config

bootleg.utils.utils module

Bootleg utils.

bootleg.utils.utils.**chunk_file**(*in_file*, *out_dir*, *num_lines*, *prefix='out_'*)

Chunk a file into num_lines chunks.

Parameters

- **in_file** – input file
- **out_dir** – output directory
- **num_lines** – number of lines in each chunk
- **prefix** – prefix for output files in out_dir

Returns: total number of lines read, dictionary of output file path -> number of lines in that file (for tqdms)

bootleg.utils.utils.**chunks**(*iterable*, *n*)

Chunk data.

chunks(ABCDE,2) => AB CD E.

Parameters

- **iterable** – iterable input
- **n** – number of chunks

Returns: next chunk

bootleg.utils.utils.**create_single_item_trie**(*in_dict*, *out_file=''*)

Create marisa trie.

Creates a marisa trie from the input dictionary. We assume the dictionary has string keys and integer values.

Parameters

- **in_dict** – Dict[str] -> Int
- **out_file** – marisa file to save (useful for reading as memmap) (optional)

Returns: marisa trie of in_dict

bootleg.utils.utils.**dump_json_file**(*filename*, *contents*, *ensure_ascii=False*)

Dump dictionary to json file.

Parameters

- **filename** – file to write to
- **contents** – dictionary to save
- **ensure_ascii** – ensure ascii

bootleg.utils.utils.**dump_yaml_file**(*filename*, *contents*)

Dump dictionary to yaml file.

Parameters

- **filename** – file to write to
- **contents** – dictionary to save

bootleg.utils.utils.**ensure_dir**(*d*)

Check if a directory exists. If not, it makes it.

Parameters **d** – path

bootleg.utils.utils.**exists_dir**(*d*)

Check if directory exists.

Parameters **d** – path

bootleg.utils.utils.**get_lnrm**(*s*, *strip=1*, *lower=1*)

Convert to lnrm form.

Convert a string to its lnrm form We form the lower-cased normalized version l(s) of a string s by canonicalizing its UTF-8 characters, eliminating diacritics, lower-casing the UTF-8 and throwing out all ASCII- range characters that are not alpha-numeric.

from http://nlp.stanford.edu/pubs/subctackbp.pdf Section 2.3

Parameters

- **s** – input string
- **strip** – boolean for stripping alias or not
- **lower** – boolean for lowercasing alias or not

Returns: the lnrm form of the string

bootleg.utils.utils.**load_json_file**(*filename*)

Load dictionary from json file.

Parameters **filename** – file to read from

Returns: Dict

bootleg.utils.utils.**load_single_item_trie**(*file*)

Load a marisa trie with integer values from memmap file.

Parameters **file** – marisa input file

Returns: marisa trie

bootleg.utils.utils.**load_yaml_file**(*filename*)

Load dictionary from yaml file.

Parameters **filename** – file to read from

Returns: Dict

bootleg.utils.utils.**recurse_redict**(*d*)

Cast all DottedDict values in a dictionary to be dictionaries.

Useful for YAML dumping.

Parameters **d** – Dict

Returns: Dict with no DottedDicts

bootleg.utils.utils.**strip_nan**(*input_list*)

Replace float('nan') with nulls.

Used for ujson loading/dumping.

Parameters **input_list** – list of items to remove the Nans from

Returns: list or nested list where Nan is not None

bootleg.utils.utils.**try_rmtree**(*rm_dir*)

Try to remove a directory tree.

In the case a resource is open, rmtree will fail. This retries to rmtree after 1 second waits for 5 times.

Parameters **rm_dir** – directory to remove

bootleg.utils.utils.**write_jsonl**(*filepath*, *values*, *ensure_ascii=False*)

Write List[Dict] data to jsonlines file.

Parameters

- **filepath** – file to write to
- **values** – list of dictionary data to write
- **ensure_ascii** – ensure_ascii for json

bootleg.utils.utils.**write_to_file**(*filename*, *value*)

Write generic value to a file.

If value is not string, will cast to str().

Parameters

- **filename** – file to write to
- **value** – context to write

Returns: Dict

Module contents

Util init.

14.1.2 Submodules

14.1.3 bootleg.data module

Bootleg data creation.

bootleg.data.**bootleg_collate_fn**(*batch: Union[List[Tuple[Dict[str, Any], Dict[str, torch.Tensor]]], List[Dict[str, Any]]]*) → Union[Tuple[Dict[str, Any], Dict[str, torch.Tensor]], Dict[str, Any]]

Collate function (modified from emmental collate fn).

The main difference is our collate function merges candidates from across the batch for disambiguation. :param batch: The batch to collate.

Returns The collated batch.

bootleg.data.**get_dataloaders**(*args*, *tasks*, *use_batch_cands*, *load_entity_data*, *splits*, *entity_symbols*, *tokenizer*, *dataset_offsets: Optional[Dict[str, List[int]]] = None*)

Get the dataloaders.

Parameters

- **args** – main args
- **tasks** – task names
- **use_batch_cands** – whether to use candidates across a batch (train and eval_batch_cands)
- **load_entity_data** – whether to load entity data
- **splits** – data splits to generate dataloaders for
- **entity_symbols** – entity symbols
- **dataset_offsets** – [start, end] offsets for each split to index into the dataset. Dataset len is end-start. If end is None, end is the length of the dataset.

Returns: list of dataloaders

bootleg.data.**get_entity_dataloaders**(*args*, *tasks*, *entity_symbols*, *tokenizer*)

Get the entity dataloaders.

Parameters

- **args** – main args
- **tasks** – task names
- **entity_symbols** – entity symbols

Returns: list of dataloaders

bootleg.data.**get_slicedatasets**(*args*, *splits*, *entity_symbols*)

Get the slice datasets.

Parameters

- **args** – main args

- **splits** – splits to get datasets for
- **entity_symbols** – entity symbols

Returns: Dict of slice datasets

14.1.4 bootleg.dataset module

Bootleg NED Dataset.

class bootleg.dataset.**BootlegDataset**(*main_args*, *name*, *dataset*, *use_weak_label*, *load_entity_data*, *tokenizer*, *entity_symbols*, *dataset_threads*, *split='train'*, *is_bert=True*, *dataset_range=None*)

Bases: emmental.data.EmmentalDataset

Bootleg Dataset class.

Parameters

- **main_args** – input config
- **name** – internal dataset name
- **dataset** – dataset file
- **use_weak_label** – whether to use weakly labeled mentions or not
- **load_entity_data** – whether to load entity data or not
- **tokenizer** – sentence tokenizer
- **entity_symbols** – entity database class
- **dataset_threads** – number of threads to use
- **split** – data split
- **is_bert** – is the tokenizer a BERT or not
- **dataset_range** – offset into dataset

classmethod build_data_dicts(*save_dataset_name*, *save_labels_name*, *X_storage*, *Y_storage*)

Return the X_dict and Y_dict of inputs and labels.

Parameters

- **save_dataset_name** – memmap file name with inputs
- **save_labels_name** – memmap file name with labels
- **X_storage** – memmap storage for inputs
- **Y_storage** – memmap storage labels

Returns: X_dict of inputs and Y_dict of labels for Emmental datasets

classmethod build_data_entity_dicts(*save_dataset_name*, *X_storage*)

Return the X_dict for the entity data.

Parameters

- **save_dataset_name** – memmap file name with entity data
- **X_storage** – memmap storage type

Returns: Dict of labels

get_sentidx_to_rowids()

Get mapping from sent idx to row id in X_dict.

Returns: Dict of sent idx to row id

class bootleg.dataset.**BootlegEntityDataset**(*main_args*, *name*, *dataset*, *tokenizer*, *entity_symbols*, *dataset_threads*, *split='test'*)

Bases: emmental.data.EmmentalDataset

Bootleg Dataset class for entities.

Parameters

- **main_args** – input config
- **name** – internal dataset name
- **dataset** – dataset file
- **tokenizer** – sentence tokenizer
- **entity_symbols** – entity database class
- **dataset_threads** – number of threads to use
- **split** – data split

classmethod build_data_entity_dicts(*save_dataset_name*, *X_storage*)

Return the X_dict for the entity data.

Parameters

- **save_dataset_name** – memmap file name with entity data
- **X_storage** – memmap storage type

Returns: Dict of labels

class bootleg.dataset.**InputExample**(*sent_idx*, *subsent_idx*, *alias_list_pos*, *alias_to_predict*, *span*, *phrase*, *alias*, *qid*, *qid_cnt_mask_score*)

Bases: object

A single training/test example for prediction.

classmethod from_dict(*in_dict*)

Create pobject from dictionary.

to_dict()

Return dictionary of object.

class bootleg.dataset.**InputFeatures**(*alias_idx*, *word_input_ids*, *word_token_type_ids*, *word_attention_mask*, *word_qid_cnt_mask_score*, *gold_eid*, *for_dump_gold_eid*, *gold_cand_K_idx*, *for_dump_gold_cand_K_idx_train*, *alias_list_pos*, *sent_idx*, *subsent_idx*, *guid*)

Bases: object

A single set of features of data.

classmethod from_dict(*in_dict*)

Create pobject from dictionary.

to_dict()

Return dictionary of object.

bootleg.dataset.**build_and_save_entity_inputs**(*save_entity_dataset_name*, *X_entity_storage*, *data_config*, *dataset_threads*, *tokenizer*, *entity_symbols*)

Create entity features.

Parameters

- **save_entity_dataset_name** – memmap filename to save the entity data
- **X_entity_storage** – storage type for memmap file
- **data_config** – data config
- **dataset_threads** – number of threads
- **tokenizer** – tokenizer
- **entity_symbols** – entity symbols

bootleg.dataset.**build_and_save_entity_inputs_hlp**(*input_qids*)

Create entity features multiprocessing helper.

bootleg.dataset.**build_and_save_entity_inputs_initializer**(*constants*, *data_config*, *save_entity_dataset_name*, *X_entity_storage*, *tokenizer*)

Create entity features multiprocessing initializer.

bootleg.dataset.**build_and_save_entity_inputs_single**(*input_qids*, *constants*, *memfile*, *type_symbols*, *kg_symbols*, *tokenizer*, *entity_symbols*)

Create entity features.

bootleg.dataset.**convert_examples_to_features_and_save**(*meta_file*, *guid_dtype*, *data_config*, *dataset_threads*, *use_weak_label*, *split*, *is_bert*, *save_dataset_name*, *save_labels_name*, *X_storage*, *Y_storage*, *tokenizer*, *entity_symbols*)

Create features from examples.

Converts the prepped examples into input features and saves in memmap files. These are used in the __get_item__ method.

Parameters

- **meta_file** – metadata file where input file paths are saved
- **guid_dtype** – unique identifier dtype
- **data_config** – data config
- **dataset_threads** – number of threads
- **use_weak_label** – whether to use weak labeling or not
- **split** – data split
- **is_bert** – is the tokenizer a BERT tokenizer
- **save_dataset_name** – data features file name to save
- **save_labels_name** – data labels file name to save
- **X_storage** – data features storage type (for memmap)

- **Y_storage** – data labels storage type (for memmap)
- **tokenizer** – tokenizer
- **entity_symbols** – entity symbols

bootleg.dataset.**convert_examples_to_features_and_save_hlp**(*input_dict*)

Convert examples to features multiprocessing initializer.

bootleg.dataset.**convert_examples_to_features_and_save_initializer**(*tokenizer*, *data_config*, *save_dataset_name*, *save_labels_name*, *X_storage*, *Y_storage*)

Create examples multiprocessing initializer.

bootleg.dataset.**convert_examples_to_features_and_save_single**(*input_dict*, *tokenizer*, *entitysymbols*, *mmap_file*, *mmap_label_file*)

Convert examples to features multiprocessing helper.

bootleg.dataset.**create_examples**(*dataset*, *create_ex_indir*, *create_ex_outdir*, *meta_file*, *data_config*, *dataset_threads*, *use_weak_label*, *split*, *is_bert*, *tokenizer*)

Create examples from the raw input data.

Parameters

- **dataset** – data file to read
- **create_ex_indir** – temporary directory where input files are stored
- **create_ex_outdir** – temporary directory to store output files from method
- **meta_file** – metadata file to save the file names/paths for the next step in prep pipeline
- **data_config** – data config
- **dataset_threads** – number of threads
- **use_weak_label** – whether to use weak labeling or not
- **split** – data split
- **is_bert** – is the tokenizer a BERT one
- **tokenizer** – tokenizer

bootleg.dataset.**create_examples_hlp**(*args*)

Create examples multiprocessing helper.

bootleg.dataset.**create_examples_initializer**(*constants_dict*, *tokenizer*)

Create examples multiprocessing initializer.

bootleg.dataset.**create_examples_single**(*in_file_idx*, *in_file_name*, *in_file_lines*, *out_file_name*, *constants_dict*, *tokenizer*)

Create examples.

bootleg.dataset.**extract_context**(*span*, *sentence*, *max_seq_window_len*, *tokenizer*)

Extract the left and right context window around a span.

Parameters

- **span** – character span (left and right values)
- **sentence** – sentence

- **max_seq_window_len** – maximum window length around a span
- **tokenizer** – tokenizer

Returns: context window

bootleg.dataset.**get_entity_string**(*qid*, *constants*, *entity_symbols*, *kg_symbols*, *type_symbols*)

Get string representation of entity.

For each entity, generates a string that is fed into a language model to generate an entity embedding. Returns all tokens that are the title of the entity (even if in the description)

Parameters

- **qid** – QID
- **constants** – Dict of constants
- **entity_symbols** – entity symbols
- **kg_symbols** – kg symbols
- **type_symbols** – type symbols

Returns: entity strings, number of types over max length, number of relations over max length

bootleg.dataset.**get_structural_entity_str**(*items*, *max_tok_len*, *sep_tok*)

Return sep_tok joined list of items of strucutral resources.

Parameters

- **items** – list of structural resources
- **max_tok_len** – maximum token length
- **sep_tok** – token to separate out resources

Returns result string, number of items that went beyond `max_tok_len`

14.1.5 bootleg.extract_all_entities module

Bootleg run command.

bootleg.extract_all_entities.**parse_cmdline_args**()

Parse command line.

Takes an input config file and parses it into the correct subdictionary groups for the model.

Returns model run mode of train, eval, or dumping parsed Dict config path to original config path

bootleg.extract_all_entities.**run_model**(*config*, *run_config_path=None*)

Run Emmental Bootleg model.

Parameters

- **config** – parsed model config
- **run_config_path** – original config path (for saving)

bootleg.extract_all_entities.**setup**(*config*, *run_config_path=None*)

Set distributed backend and save configuration files.

Parameters

- **config** – config

- **run_config_path** – path for original run config

14.1.6 bootleg.run module

Bootleg run command.

bootleg.run.**configure_optimizer**()

Configure the optimizer for Bootleg.

Parameters **config** – config

bootleg.run.**parse_cmdline_args**()

Take an input config file and parse it into the correct subdictionary groups for the model.

Returns model run mode of train, eval, or dumping parsed Dict config path to original config path

bootleg.run.**run_model**(*mode*, *config*, *run_config_path=None*, *entity_emb_file=None*)

Run Emmental Bootleg models.

Parameters

- **mode** – run mode (train, eval, dump_preds)
- **config** – parsed model config
- **run_config_path** – original config path (for saving)
- **entity_emb_file** – file for dumped entity embeddings

bootleg.run.**setup**(*config*, *run_config_path=None*)

Set distributed backend and save configuration files.

Parameters

- **config** – config
- **run_config_path** – path for original run config

14.1.7 bootleg.scorer module

Bootleg scorer.

class bootleg.scorer.**BootlegSlicedScorer**(*train_in_candidates*, *slices_datasets=None*)

Bases: object

Sliced NED scorer init.

Parameters

- **train_in_candidates** – are we training assuming that all gold qids are in the candidates or not
- **slices_datasets** – slice dataset (see slicing/slice_dataset.py)

bootleg_score(*golds: numpy.ndarray*, *probs: numpy.ndarray*, *preds: Optional[numpy.ndarray]*, *uids: Optional[List[str]] = None*) → Dict[str, float]

Scores the predictions using the gold labels and slices.

Parameters

- **golds** – gold labels

- **probs** – probabilities
- **preds** – predictions (max prob candidate)
- **uids** – unique identifiers

Returns: dictionary of tensorboard compatible keys and metrics

get_slices(*uid*)

Get slices incidence matrices.

Get slice incidence matrices for the uid Uid is dtype (np.dtype([('sent_idx', 'i8', 1), ('subsent_idx', 'i8', 1), ("alias_orig_list_pos", 'i8', max_aliases)]) where alias_orig_list_pos gives the mentions original positions in the sentence.

Parameters **uid** – unique identifier of sentence

Returns: dictionary of slice_name -> matrix of 0/1 for if alias is in slice or not (-1 for no alias)

14.1.8 bootleg.task_config module

Emmental task constants.

14.1.9 Module contents

Print functions for distributed computation.

bootleg.**log_rank_0_debug**(*logger*, *message*)

If distributed is initialized log debug only on rank 0.

bootleg.**log_rank_0_info**(*logger*, *message*)

If distributed is initialized log info only on rank 0.

PYTHON MODULE INDEX

b

INDEX

C

D

E

F

G

I

K

L

M

N

T

U

W