multi_task_NLP Documentation

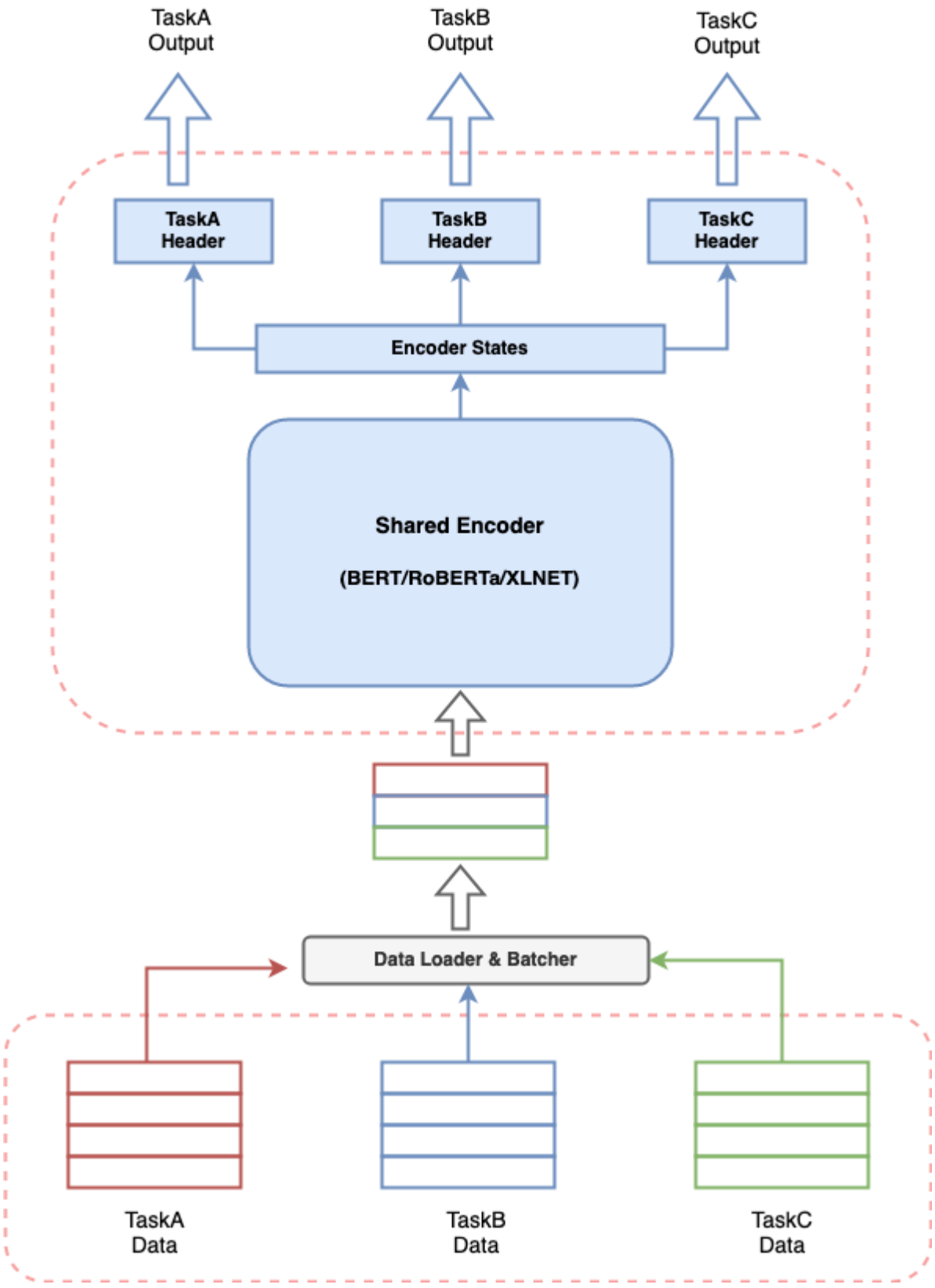
Release 0.0.1

saransh mehta

Jun 15, 2020

Contents

multi_task_NLP is a utility toolkit enabling NLP developers to easily train and infer a single model for multiple tasks. We support various data formats for majority of NLI tasks and multiple transformer-based encoders (eg. BERT, Distil-BERT, ALBERT, RoBERTa, XLNET etc.)


TaskA
Output
TaskB
Output
TaskC
Output
TaskA
Header
TaskB
Header
TaskC
Header
Encoder States
Shared Encoder
(BERT/RoBERTa/XLNET)
Data Loader & Batcher
TaskA
Data
TaskB
Data
TaskC
Data

CHAPTER 1

What is multi_task_NLP about?

Any conversational AI system involves building multiple components to perform various tasks and a pipeline to stitch all components together. Provided the recent effectiveness of transformer-based models in NLP, it's very common to build a transformer-based model to solve your use case. But having multiple such models running together for a conversational AI system can lead to expensive resource consumption, increased latencies for predictions and make the system difficult to manage. This poses a real challenge for anyone who wants to build a conversational AI system in a simplistic way.

multi_task_NLP gives you the capability to define multiple tasks together and train a single model which simultaneously learns on all defined tasks. This means one can perform multiple tasks with latency and resource consumption equivalent to a single task.

CHAPTER 2

Installation

To use multi-task-NLP, you can clone the repository into the desired location on your system with the following terminal command.

```
$ cd /desired/location/
$ git clone https://github.com/hellohaptik/multi-task-NLP.git
$ cd multi-task-NLP
$ pip install -r requirements.txt
```

NOTE:- The library is built and tested using `Python 3.7.3`. It is recommended to install the requirements in a virtual environment.

CHAPTER 3

Quickstart Guide

A quick guide to show how a single model can be trained for multiple NLI tasks in just 3 simple steps and with **no requirement to code!!**

3.1 Quickstart

Follow these 3 simple steps to train your multi-task model!

3.1.1 Step 1 - Define your task file

Task file is a YAML format file where you can add all your tasks for which you want to train a multi-task model.

```
TaskA:
  model_type: BERT
  config_name: bert-base-uncased
  dropout_prob: 0.05
  label_map_or_file:
  -label1
  -label2
  -label3
  metrics:
  - accuracy
  loss_type: CrossEntropyLoss
  task_type: SingleSenClassification
  file_names:
  - taskA_train.tsv
  - taskA_dev.tsv
  - taskA_test.tsv

TaskB:
  model_type: BERT
  config_name: bert-base-uncased
```

(continues on next page)

(continued from previous page)

```
  dropout_prob: 0.3
  label_map_or_file: data/taskB_train_label_map.joblib
  metrics:
  - seq_f1
  - seq_precision
  - seq_recall
  loss_type: NERLoss
  task_type: NER
  file_names:
  - taskB_train.tsv
  - taskB_dev.tsv
  - taskB_test.tsv
```

For knowing about the task file parameters to make your task file, refer *here*.

3.1.2 Step 2 - Run data preparation

After defining the task file in *Step 1*, run the following command to prepare the data.

```
$ python data_preparation.py \
    --task_file 'sample_task_file.yml' \
    --data_dir 'data' \
    --max_seq_len 50
```

For knowing about the `data_preparation.py` script and its arguments, refer *here*.

3.1.3 Step 3 - Run train

Finally you can start your training using the following command.

```
$ python train.py \
    --data_dir 'data/bert-base-uncased_prepared_data' \
    --task_file 'sample_task_file.yml' \
    --out_dir 'sample_out' \
    --epochs 5 \
    --train_batch_size 4 \
    --eval_batch_size 8 \
    --grad_accumulation_steps 2 \
    --log_per_updates 25 \
    --save_per_updates 1000 \
    --eval_while_train True \
    --test_while_train True \
    --max_seq_len 50 \
    --silent True
```

For knowing about the `train.py` script and its arguments, refer *here*.

CHAPTER 4

Examples Guide

We provide exemplar notebooks to demonstrate some conversational AI tasks which can be perfomed using our library. You can follow along the notebooks to understand and train a multi-task model for the tasks.

4.1 Examples

Here you can find various NLP (especially conversational AI) tasks as examples and can train them either in multi-task or single-task manner, using some simple steps mentioned in the notebooks.

4.1.1 Example-1 Intent detection, NER, Fragment detection

Tasks Description

`Intent Detection` :- This is a single sentence classification task where an *intent* specifies which class the data sample belongs to.

`NER` :- This is a Named Entity Recognition/ Sequence Labelling/ Slot filling task where individual words of the sentence are tagged with an entity label it belongs to. The words which don't belong to any entity label are simply labeled as "O".

`Fragment Detection` :- This is modeled as a single sentence classification task which detects whether a sentence is incomplete (fragment) or not (non-fragment).

Conversational Utility :- Intent detection is one of the fundamental components for conversational system as it gives a broad understand of the category/domain the sentence/query belongs to.

NER helps in extracting values for required entities (eg. location, date-time) from query.

Fragment detection is a very useful piece in conversational system as knowing if a query/sentence is incomplete can aid in discarding bad queries beforehand.

Intent Detection

Query: I need a reservation for a bar in bangladesh on feb the 11th 2032

Intent: BookRestaurant

NER

Query: [‘book’, ‘a’, ‘spot’, ‘for’, ‘ten’, ‘at’, ‘a’, ‘top-rated’, ‘caucasian’, ‘restaurant’, ‘not’, ‘far’, ‘from’, ‘selmer’]

NER tags: [‘O’, ‘O’, ‘O’, ‘O’, ‘B-party_size_number’, ‘O’, ‘O’, ‘B-sort’, ‘B-cuisine’, ‘B-restaurant_type’, ‘B-spatial_relation’, ‘I-spatial_relation’, ‘O’, ‘B-city’]

Fragment Detection

Query: a reservation for

Label: fragment

Notebook :- intent_ner_fragment

Transform file :- transform_file_snips

Tasks file :- tasks_file_snips

4.1.2 Example-2 Recognising Textual Entailment

Tasks Description

`Entailment` :- This is a sentence pair classification task which determines whether the second sentence in a sample can be inferred from the first.

Conversational Utility :- In conversational AI context, this task can be seen as determining whether the second sentence is similar to first or not. Additionally, the probability score can also be used as a similarity score between the sentences.

Query1: An old man with a package poses in front of an advertisement.

Query2: A man poses in front of an ad.

Label: entailment

Query1: An old man with a package poses in front of an advertisement.

Query2: A man poses in front of an ad for beer.

Label: non-entailment

Notebook :- entailment_snli

Transform file :- transform_file_snli

Tasks file :- tasks_file_snli

4.1.3 Example-3 Answerability detection

Tasks Description

`answerability` :- This is modeled as a sentence pair classification task where the first sentence is a query and second sentence is a context passage. The objective of this task is to determine whether the query can be answered from the context passage or not.

Conversational Utility :- This can be a useful component for building a question-answering/ machine comprehension based system. In such cases, it becomes very important to determine whether the given query can be answered with

given context passage or not before extracting/abstracting an answer from it. Performing question-answering for a query which is not answerable from the context, could lead to incorrect answer extraction.

> Query: how much money did evander holyfield make
>
> Context: Evander Holyfield Net Worth. How much is Evander Holyfield Worth? Evander Holyfield Net Worth: Evander Holyfield is a retired American professional boxer who has a net worth of $500 thousand. A professional boxer, Evander Holyfield has fought at the Heavyweight, Cruiserweight, and Light-Heavyweight Divisions, and won a Bronze medal a the 1984 Olympic Games.
>
> Label: answerable

Notebook :- answerability_detection_msmarco

Transform file :- transform_file_answerability

Tasks file :- tasks_file_answerability

4.1.4 Example-4 Query type detection

Tasks Description

`querytype` :- This is a single sentence classification task to determine what type (category) of answer is expected for the given query. The queries are divided into 5 major classes according to the answer expected for them.

Conversational Utility :- While returning a response for a query, knowing what kind of answer is expected for the query can help in both curating and cross-verfying an answer according to the type.

> Query: what's the distance between destin florida and birmingham alabama?
>
> Label: NUMERIC
>
> Query: who is suing scott wolter
>
> Label: PERSON

Notebook :- query_type_detection

Transform file :- transform_file_querytype

Tasks file :- tasks_file_querytype

4.1.5 Example-5 POS tagging, NER tagging

Tasks Description

`NER` :-This is a Named Entity Recognition task where individual words of the sentence are tagged with an entity label it belongs to. The words which don't belong to any entity label are simply labeled as "O".

`POS` :- This is a Part of Speech tagging task. A part of speech is a category of words that have similar grammatical properties. Each word of the sentence is tagged with the part of speech label it belongs to. The words which don't belong to any part of speech label are simply labeled as "O".

Conversational Utility :- In conversational AI context, determining the syntactic parts of the sentence can help in extracting noun-phrases or important keyphrases from the sentence.

> Query: ['Despite', 'winning', 'the', 'Asian', 'Games', 'title', 'two', 'years', 'ago', ',', 'Uzbekistan', 'are', 'in', 'the', 'finals', 'as', 'outsiders', '.']
>
> NER tags: ['O', 'O', 'O', 'I-MISC', 'I-MISC', 'O', 'O', 'O', 'O', 'O', 'I-LOC', 'O', 'O', 'O', 'O', 'O', 'O', 'O']

POS tags: ['I-PP', 'I-VP', 'I-NP', 'I-NP', 'I-NP', 'I-NP', 'B-NP', 'I-NP', 'I-ADVP', 'O', 'I-NP', 'I-VP', 'I-PP', 'I-NP', 'I-NP', 'I-SBAR', 'I-NP', 'O']

Notebook :- ner_pos_tagging_conll

Transform file :- transform_file_conll

Tasks file :- tasks_file_conll

4.1.6 Example-6 Query correctness

Tasks Description

`querycorrectness` :- This is modeled as single sentence classification task identifying whether or not a query is structurally well formed. can enhance query un-derstanding.

Conversational Utility :- Determining how much the query is structured would help in enhancing query understanding and improve reliability of tasks which depend on query structure to extract information.

Query: What places have the oligarchy government ?

Label: well-formed

Query: What day of Diwali in 1980 ?

Label: not well-formed

Notebook :- query_correctness

Transform file :- transform_file_query_correctness

Tasks file :- tasks_file_query_correctness

4.1.7 Example-7 Query similarity

Tasks Description

`Query similarity` :- This is a sentence pair classification task which determines whether the second sentence in a sample can be inferred from the first.

Conversational Utility :- In conversational AI context, this task can be seen as determining whether the second sentence is similar to first or not. Additionally, the probability score can also be used as a similarity score between the sentences.

Query1: What is the most used word in Malayalam?

Query2: What is meaning of the Malayalam word “”thumbatthu”“?

Label: not similar

Query1: Which is the best compliment you have ever received?

Query2: What's the best compliment you've got?

Label: similar

Notebook :- query_similarity

Transform file :- transform_file_qqp

Tasks file :- tasks_file_qqp

4.1.8 Example-8 Sentiment Analysis

Tasks Description

`sentiment` :- This is modeled as single sentence classification task to determine where a piece of text conveys a positive or negative sentiment.

Conversational Utility :- To determine whether a review is positive or negative.

> Review: What I enjoyed most in this film was the scenery of Corfu, being Greek I adore my country and I liked the flattering director's point of view. Based on a true story during the years when Greece was struggling to stand on her own two feet through war, Nazis and hardship. An Italian soldier and a Greek girl fall in love but the times are hard and they have a lot of sacrifices to make. Nicholas Cage looking great in a uniform gives a passionate account of this unfulfilled (in the beginning) love. I adored Christian Bale playing Mandras the heroine's husband-to-be, he looks very very good as a Greek, his personality matched the one of the Greek patriot! A true fighter in there, or what! One of the movies I would like to buy and keep it in my collection... for ever!
>
> Label: positive

Notebook :- IMDb_sentiment_analysis

Transform file :- transform_file_imdb

Tasks file :- tasks_file_imdb

CHAPTER 5

Step by Step Guide

A complete guide explaining all the components of multi-task-NLP in sequential order.

5.1 Task Formats

- To standardize the data input files, all the tasks require `tsv` format files as input data files.
- The tsv data files shouldn't contain any headers. Detailed tsv formats required for specific task types are mentioned in following subsection.

5.1.1 Task types

Input data formats for different NLI tasks can vary from task to task. We support the following three task types. Majority of the NLI tasks can be modeled using one of these task types.

- `SingleSenClassification`: This task type is to be used for classification of single sentences. The data files needs to have following columns separated by **"\t"** in the order as mentioned below.
  1. **Unique id** :- an id to uniquely identify each row/sample.
  2. **Label** :- label for the sentence. Labels can be numeric or strings. In case labels are strings, label mapping needs to be provided.
  3. **Sentence** :- The sentence which needs to be classified.
- `SentencePairClassification`: This task type is to be used for classification of sentence pairs (two sentences). The data files needs to have following columns separated by **"\t"** in the order as mentioned below.
  1. **Unique id** :- an id to uniquely identify each row/sample.
  2. **Label** :- label for the sentence. Labels can be numeric or strings. In case labels are strings, label mapping needs to be provided.
  3. **SentenceA** :- First sentence of the sentence pair.
  4. **SentenceB** :- Second sentence of the sentence pair.

- `NER` : This task type is to be used for sequence labelling tasks like Named Entity Recognition , entity mention detection, keyphrase extraction etc. The data files need to have following columns separated by **"\t"** in the order as mentioned below.
    1. **Unique id** :- an id to uniquely identify each row/sample.
    2. **Label** :- List of tags for words in sentence.
    3. **Sentence** :- List of words in sentence.

NOTE:- The tsv data files must not have the header names.

5.2 Data transformations

It is very likely that the data you have is not in the format as required by the library. Hence, data transformations provide a way to convert data in raw form to standard tsv format required.

5.2.1 Transform functions

Transform functions are the functions which can be used for performing transformations. Each function is defined to take raw data in certain format, perform the defined transformation steps and create the respective `tsv` file.

Sample transform functions

`utils.tranform_functions.`**`snips_intent_ner_to_tsv`**(*dataDir*, *readFile*, *wrtDir*, *transParamDict*, *isTrainFile=False*)

This function transforms the data present in snips_data/. Raw data is in BIO tagged format with the sentence intent specified at the end of each sentence. The transformation function converts the each raw data file into two separate tsv files, one for intent classification task and another for NER task. Following transformed files are written at wrtDir

- NER transformed tsv file.
- NER label map joblib file.
- intent transformed tsv file.
- intent label map joblib file.

For using this transform function, set `transform_func` : **snips_intent_ner_to_tsv** in transform file.

Parameters

- **`dataDir`** (`str`) – Path to the directory where the raw data files to be read are present..
- **`readFile`** (`str`) – This is the file which is currently being read and transformed by the function.
- **`wrtDir`** (`str`) – Path to the directory where to save the transformed tsv files.
- **`transParamDict`** (`dict`, defaults to `None`) – Dictionary of function specific parameters. Not required for this transformation function.

`utils.tranform_functions.`**`snli_entailment_to_tsv`**(*dataDir*, *readFile*, *wrtDir*, *transParamDict*, *isTrainFile=False*)

This function transforms the SNLI entailment data available at SNLI for sentence pair entailment task. Contradiction and neutral labels are mapped to 0 representing non-entailment scenario. Only entailment label is mapped to 1, representing an entailment scenario. Following transformed files are written at wrtDir

- Sentence pair transformed tsv file for entailment task

For using this transform function, set `transform_func` : **snli_entailment_to_tsv** in transform file.

Parameters

- **dataDir** (`str`) – Path to the directory where the raw data files to be read are present..
- **readFile** (`str`) – This is the file which is currently being read and transformed by the function.
- **wrtDir** (`str`) – Path to the directory where to save the transformed tsv files.
- **transParamDict** (`dict`, defaults to `None`) – Dictionary of function specific parameters. Not required for this transformation function.

utils.tranform_functions.**create_fragment_detection_tsv**(*dataDir*, *readFile*, *wrtDir*, *transParamDict*, *isTrainFile=False*)

This function transforms data for fragment detection task (detecting whether a sentence is incomplete/fragment or not). It takes data in single sentence classification format and creates fragment samples from the sentences. In the transformed file, label 1 and 0 represent fragment and non-fragment sentence respectively. Following transformed files are written at wrtDir

- Fragment transformed tsv file containing fragment/non-fragment sentences and labels

For using this transform function, set `transform_func` : **create_fragment_detection_tsv** in transform file. :param dataDir: Path to the directory where the raw data files to be read are present.. :type dataDir: `str` :param readFile: This is the file which is currently being read and transformed by the function. :type readFile: `str` :param wrtDir: Path to the directory where to save the transformed tsv files. :type wrtDir: `str` :param transParamDict: Dictionary requiring the following parameters as key-value

- `data_frac` (defaults to 0.2) : Fraction of data to consider for making fragments.
- `seq_len_right` : (defaults to 3) : Right window length for making n-grams.
- `seq_len_left` (defaults to 2) : Left window length for making n-grams.
- `sep` (defaults to " ") : column separator for input file.
- `query_col` (defaults to 2) : column number containing sentences. Counting starts from 0.

utils.tranform_functions.**msmarco_answerability_detection_to_tsv**(*dataDir*, *readFile*, *wrtDir*, *transParamDict*, *isTrainFile=False*)

This function transforms the MSMARCO triples data available at triples

The data contains triplets where the first entry is the query, second one is the context passage from which the query can be answered (positive passage) , while the third entry is a context passage from which the query cannot be answered (negative passage). Data is transformed into sentence pair classification format, with query-positive context pair labeled as 1 (answerable) and query-negative context pair labeled as 0 (non-answerable)

Following transformed files are written at wrtDir

- Sentence pair transformed downsampled file.
- Sentence pair transformed train tsv file for answerability task
- Sentence pair transformed dev tsv file for answerability task

- Sentence pair transformed test tsv file for answerability task

For using this transform function, set `transform_func` : **msmarco_answerability_detection_to_tsv** in transform file.

Parameters

- **dataDir** (`str`) – Path to the directory where the raw data files to be read are present..
- **readFile** (`str`) – This is the file which is currently being read and transformed by the function.
- **wrtDir** (`str`) – Path to the directory where to save the transformed tsv files.
- **transParamDict** (`dict`, defaults to `None`) – Dictionary of function specific parameters. Not required for this transformation function.
  - `data_frac` (defaults to 0.01) : Fraction of data to keep in downsampling as the original data size is too large.

`utils.tranform_functions.`**msmarco_query_type_to_tsv**(*dataDir*, *readFile*, *wrtDir*, *transParamDict*, *isTrainFile=False*)

This function transforms the MSMARCO QnA data available at MSMARCO_QnA for query-type detection task (given a query sentence, detect what type of answer is expected). Queries are divided into 5 query types - NUMERIC, LOCATION, ENTITY, DESCRIPTION, PERSON. The function transforms the json data to standard single sentence classification type tsv data. Following transformed files are written at wrtDir

- Query type transformed tsv data file.
- Query type label map joblib file.

For using this transform function, set `transform_func` : **msmarco_query_type_to_tsv** in transform file.

Parameters

- **dataDir** (`str`) – Path to the directory where the raw data files to be read are present..
- **readFile** (`str`) – This is the file which is currently being read and transformed by the function.
- **wrtDir** (`str`) – Path to the directory where to save the transformed tsv files.
- **transParamDict** (`dict`, defaults to `None`) – Dictionary requiring the following parameters as key-value
  - `data_frac` (defaults to 0.05) : Fraction of data to consider for downsampling.

`utils.tranform_functions.`**bio_ner_to_tsv**(*dataDir*, *readFile*, *wrtDir*, *transParamDict*, *isTrainFile=False*)

This function transforms the BIO style data and transforms into the tsv format required for NER. Following transformed files are written at wrtDir,

- NER transformed tsv file.
- NER label map joblib file.

For using this transform function, set `transform_func` : **bio_ner_to_tsv** in transform file.

Parameters

- **dataDir** (`str`) – Path to the directory where the raw data files to be read are present..
- **readFile** (`str`) – This is the file which is currently being read and transformed by the function.
- **wrtDir** (`str`) – Path to the directory where to save the transformed tsv files.

- **transParamDict** (dict, defaults to None) – Dictionary requiring the following parameters as key-value
  - save_prefix (defaults to 'bio_ner') : save file name prefix.
  - col_sep : (defaults to " ") : separator for columns
  - tag_col (defaults to 1) : column number where label NER tag is present for each row. Counting starts from 0.
  - sen_sep (defaults to " ") : end of sentence separator.

utils.tranform_functions.**coNLL_ner_pos_to_tsv**(*dataDir*, *readFile*, *wrtDir*, *transParamDict*, *isTrainFile=False*)

This function transforms the data present in coNLL_data/. Raw data is in BIO tagged format with the POS and NER tags separated by space. The transformation function converts the each raw data file into two separate tsv files, one for POS tagging task and another for NER task. Following transformed files are written at wrtDir

- NER transformed tsv file.
- NER label map joblib file.
- POS transformed tsv file.
- POS label map joblib file.

For using this transform function, set transform_func : **snips_intent_ner_to_tsv** in transform file.

Parameters

- **dataDir** (str) – Path to the directory where the raw data files to be read are present..
- **readFile** (str) – This is the file which is currently being read and transformed by the function.
- **wrtDir** (str) – Path to the directory where to save the transformed tsv files.
- **transParamDict** (dict, defaults to None) – Dictionary of function specific parameters. Not required for this transformation function.

utils.tranform_functions.**qqp_query_similarity_to_tsv**(*dataDir*, *readFile*, *wrtDir*, *transParamDict*, *isTrainFile=False*)

This function transforms the QQP (Quora Question Pairs) query similarity data available at QQP

If the second query is similar to first query in a query-pair, the pair is labeled -> 1 and if not, then labeled -> 0. Following transformed files are written at wrtDir

- Sentence pair transformed train tsv file for query similarity task
- Sentence pair transformed dev tsv file for query similarity task
- Sentence pair transformed test tsv file for query similarity task

For using this transform function, set transform_func : **snli_entailment_to_tsv** in transform file.

Parameters

- **dataDir** (str) – Path to the directory where the raw data files to be read are present..
- **readFile** (str) – This is the file which is currently being read and transformed by the function.
- **wrtDir** (str) – Path to the directory where to save the transformed tsv files.
- **transParamDict** (dict, defaults to None) – Dictionary of function specific parameters. Not required for this transformation function.

- `train_frac` (defaults to 0.8) : Fraction of data to consider for training. Remaining will be divided into dev and test.

`utils.tranform_functions.`**`query_correctness_to_tsv`**(*dataDir*, *readFile*, *wrtDir*, *transParamDict*, *isTrainFile=False*)

- Query correctness transformed file

For using this transform function, set `transform_func` : **query_correctness_to_tsv** in transform file.

Parameters

- **`dataDir`** (`str`) – Path to the directory where the raw data files to be read are present..
- **`readFile`** (`str`) – This is the file which is currently being read and transformed by the function.
- **`wrtDir`** (`str`) – Path to the directory where to save the transformed tsv files.
- **`transParamDict`** (`dict`, defaults to `None`) – Dictionary of function specific parameters. Not required for this transformation function.

`utils.tranform_functions.`**`imdb_sentiment_data_to_tsv`**(*dataDir*, *readFile*, *wrtDir*, *transParamDict*, *isTrainFile=False*)

This function transforms the IMDb moview review data available at IMDb after accepting the terms.

The data is having total 50k samples labeled as *positive* or *negative*. The reviews have some html tags which are cleaned by this function. Following transformed files are written at wrtDir

- IMDb train transformed tsv file for sentiment analysis task
- IMDb test transformed tsv file for sentiment analysis task

For using this transform function, set `transform_func` : **imdb_sentiment_data_to_tsv** in transform file.

Parameters

- **`dataDir`** (`str`) – Path to the directory where the raw data files to be read are present..
- **`readFile`** (`str`) – This is the file which is currently being read and transformed by the function.
- **`wrtDir`** (`str`) – Path to the directory where to save the transformed tsv files.
- **`transParamDict`** (`dict`, defaults to `None`) – Dictionary of function specific parameters. Not required for this transformation function.
  - `train_frac` (defaults to 0.05) : Fraction of data to consider for train/test split.

Your own transform function

In case, you need to convert some custom format data into the standard tsv format, you can do that by writing your own transform function. You must keep the following points in mind while writing your function

- The function must take the standard input arguments like *sample transform functions* Any extra function specific parameter can be added to the `transParamDict` argument.
- You should add the function in `utils/tranform_functions.py` file.
- You should add a name map for the function in `utils/data_utils.py` file under `TRANSFORM_FUNCS` map. This step is required for transform file to recognize your function.
- You should be able to use your function in the *transform file*.

5.2.2 Transform File

You can easily use the sample transformation functions or your own transformation function, by defining a YAML format `transform_file`. Say you want to perform these transformations - **sample_transform1**, **sample_transform2**, . . . , **sample_transform5**. Following is an example for the transform file,

```
sample_transform1:
  transform_func: snips_intent_ner_to_tsv
  read_file_names:
    - snips_train.txt
    - snips_dev.txt
    - snips_test.txt
  read_dir: snips_data
  save_dir: demo_transform


sample_transform2:
  transform_func: snli_entailment_to_tsv
  read_file_names:
    - snli_train.jsonl
    - snli_dev.jsonl
    - snli_test.jsonl
  read_dir : snli_data
  save_dir: demo_transform

sample_transform3:
  transform_func: bio_ner_to_tsv
  transform_params:
    save_prefix : sample
    tag_col : 1
    col_sep : " "
    sen_sep : "\n"
  read_file_names:
    - coNLL_train.txt
    - coNLL_testa.txt
    - coNLL_testb.txt

  read_dir: coNLL_data
  save_dir: demo_transform

sample_transform4:
  transform_func: fragment_detection_to_tsv
  transform_params:
    data_frac : 0.2
    seq_len_right : 3
    seq_len_left : 2
    sep : "\t"
    query_col : 2
  read_file_names:
    - int_snips_train.tsv
    - int_snips_dev.tsv
    - int_snips_test.tsv

  read_dir: data
  save_dir: demo_transform

sample_transform5:
  transform_func: msmarco_query_type_to_tsv
```

(continues on next page)

(continued from previous page)

```
  transform_params:
    data_frac : 0.2
  read_file_names:
    - train_v2.1.json
    - dev_v2.1.json
    - eval_v2.1_public.json

  read_dir: msmarco_qna_data
  save_dir: demo_transform
```

NOTE:- The transform names (sample_transform1, sample_transform2, ...) are unique identifiers for the transform, hence the transform names must always be distinct.

Transform file parameters

Detailed description of the parameters available in the transform file.

- `transform_func` *(required)* : Name of the *transform function* to use.
- `transform_params` *(optional)* : Dictionary of function specific parameters which will go in `transParamDict` parameter of function.
- `read_file_names` *(required)* : List of raw data files for transformations. The first file will be considered as **train file** and will be used to create label map file when required.
- `read_dir` *(required)* : Directory containing the input files.
- `save_dir` *(required)* : Directory to save the transformed tsv/label map files.

5.2.3 Running data transformations

Once you have made the *transform file* with all the transform operations, you can run data transformations with the following terminal command.

```
$ python data_transformations.py \
      --transform_file 'transform_file.yml'
```

5.3 Shared Encoder

5.3.1 What is a shared encoder?

The concept of this library is to provide a single model for multiple tasks. To achieve this we place a transformer-based encoder at centre. Data for all tasks will go through this centre encoder. This encoder is called shared as it is responsible for majority of learnings on all the tasks. Further, task specific headers are formed over the shared encoder.

5.3.2 Task specific headers

The encoder hidden states are consumed by task specific layers defined to output logits in the format required by the task. Forward pass for a data batch belonging to say taskA occurs through the shared encoder and header for taskA. The computed loss (which is called as 'task loss') is back-propagated through the same path.

5.3.3 Choice of shared encoder

We support multiple transformer-based encoder models. For ease of use, we've integrated the encoders from the transformers library. Available encoders with their config names are mentioned below.

Model type	Config name	Default config
DISTILBERT	distilbert-base-uncased	distilbert-base-uncased
	distilbert-base-cased	
BERT	bert-base-uncased	bert-base-uncased
	bert-base-cased	
	bert-large-uncased	
	bert-large-cased	
ROBERTA	roberta-base	roberta-base
	roberta-large	
ALBERT	albert-base-v1	albert-base-v1
	albert-large-v1	
	albert-xlarge-v1	
	albert-xxlarge-v1	
	albert-base-v2	
	albert-large-v2	
	albert-xlarge-v2	
	albert-xxlarge-v2	
XLNET	xlnet-base-cased	xlnet-base-cased
	xlnet-large-cased	

5.3.4 Losses

We support following two types of loss functions.

class `models.loss.`**CrossEntropyLoss**(*alpha=1.0*, *name='Cross Entropy Loss'*)

forward(*inp*, *target*, *attnMasks=None*)

This is the standard cross entropy loss as defined in pytorch. This loss should be used for single sentence or sentence pair classification tasks.

To use this loss for training, set `loss_type` : **CrossEntropyLoss** in task file

class `models.loss.`**NERLoss**(*alpha=1.0*, *name='Cross Entropy Loss'*)

forward(*inp*, *target*, *attnMasks=None*)

This loss is a modified version of cross entropy loss for NER/sequence labelling tasks. This loss ignores extra 'O' values through attention masks.

To use this loss for training, set `loss_type` : **NERLoss** in task file

5.3.5 Metrics

For evaluating the performance on dev and test sets during training, we provide the following standard metrics.

File for creating metric functions

`utils.eval_metrics.`**classification_accuracy**(*yTrue*, *yPred*)

Accuracy score for classification tasks using the label provided in file and predictions from multi-task model. It takes a batch of predictions and labels.

To use this metric, add **classification_accuracy** into list of `metrics` in task file.

Parameters

- **yPred** (`list`) – [0, 2, 1, 3]
- **yTrue** (`list`) – [0, 1, 2, 3]

`utils.eval_metrics.`**`classification_f1_score`**(*yTrue*, *yPred*)

Standard f1 score from sklearn for classification tasks. It takes a batch of predictions and labels.

To use this metric, add **classification_f1_score** into list of `metrics` in task file.

Parameters

- **yPred** (`list`) – [0, 2, 1, 3]
- **yTrue** (`list`) – [0, 1, 2, 3]

`utils.eval_metrics.`**`seqeval_f1_score`**(*yTrue*, *yPred*)

f1 score for NER/sequence labelling tasks taken from the seqeval library.

To use this metric, add **seqeval_f1_score** into list of `metrics` in task file.

Parameters

- **yTrue** (`list of list`) – [['O', 'O', 'O', 'B-MISC', 'I-MISC', 'I-MISC', 'O'], ['B-PER', 'I-PER', 'O']]
- **yPred** (`list of list`) – [['O', 'O', 'B-MISC', 'I-MISC', 'I-MISC', 'I-MISC', 'O'], ['B-PER', 'I-PER', 'O']]

`utils.eval_metrics.`**`seqeval_precision`**(*yTrue*, *yPred*)

Precision score for NER/sequence labelling tasks taken from the seqeval library.

To use this metric, add **seqeval_precision** into list of `metrics` in task file.

Parameters

- **yTrue** (`list of list`) – [['O', 'O', 'O', 'B-MISC', 'I-MISC', 'I-MISC', 'O'], ['B-PER', 'I-PER', 'O']]
- **yPred** (`list of list`) – [['O', 'O', 'B-MISC', 'I-MISC', 'I-MISC', 'I-MISC', 'O'], ['B-PER', 'I-PER', 'O']]

`utils.eval_metrics.`**`seqeval_recall`**(*yTrue*, *yPred*)

Recall score for NER/sequence labelling tasks taken from the seqeval library.

To use this metric, add **seqeval_recall** into list of `metrics` in task file.

Parameters

- **yTrue** (`list of list`) – [['O', 'O', 'O', 'B-MISC', 'I-MISC', 'I-MISC', 'O'], ['B-PER', 'I-PER', 'O']]
- **yPred** (`list of list`) – [['O', 'O', 'B-MISC', 'I-MISC', 'I-MISC', 'I-MISC', 'O'], ['B-PER', 'I-PER', 'O']]

`utils.eval_metrics.`**`snips_f1_score`**(*yTrue*, *yPred*)

f1 score for SNIPS NER/Slot filling task taken from the MiuLab library.

To use this metric, add **snips_f1_score** into list of `metrics` in task file.

Parameters

- **yTrue** (`list of list`) – [['O', 'O', 'O', 'B-MISC', 'I-MISC', 'I-MISC', 'O'], ['B-PER', 'I-PER', 'O']]

- **yPred** (list of list) – [['O', 'O', 'B-MISC', 'I-MISC', 'I-MISC', 'I-MISC', 'O'], ['B-PER', 'I-PER', 'O']]

utils.eval_metrics.**snips_precision**(*yTrue*, *yPred*)

Precision score for SNIPS NER/Slot filling task taken from the MiuLab library.

To use this metric, add **snips_precision** into list of metrics in task file.

Parameters

- **yTrue** (list of list) – [['O', 'O', 'O', 'B-MISC', 'I-MISC', 'I-MISC', 'O'], ['B-PER', 'I-PER', 'O']]
- **yPred** (list of list) – [['O', 'O', 'B-MISC', 'I-MISC', 'I-MISC', 'I-MISC', 'O'], ['B-PER', 'I-PER', 'O']]

utils.eval_metrics.**snips_recall**(*yTrue*, *yPred*)

Recall score for SNIPS NER/Slot filling task taken from the MiuLab library.

To use this metric, add **snips_recall** into list of metrics in task file.

Parameters

- **yTrue** (list of list) – [['O', 'O', 'O', 'B-MISC', 'I-MISC', 'I-MISC', 'O'], ['B-PER', 'I-PER', 'O']]
- **yPred** (list of list) – [['O', 'O', 'B-MISC', 'I-MISC', 'I-MISC', 'I-MISC', 'O'], ['B-PER', 'I-PER', 'O']]

5.4 How to define your multi-task model?

Let's consider you have three tasks - **TaskA**, **TaskB** and **TaskC** to train together. TaskA is single sentence classification type, TaskB is NER type and TaskC is sentence pair classification type. You can define a task file mentioning the required details about the task in following YAML format.

```
TaskA:
  model_type: BERT
  config_name: bert-base-uncased
  dropout_prob: 0.05
  label_map_or_file:
  -label1
  -label2
  -label3
  metrics:
  - accuracy
  loss_type: CrossEntropyLoss
  task_type: SingleSenClassification
  file_names:
  - taskA_train.tsv
  - taskA_dev.tsv
  - taskA_test.tsv

TaskB:
  model_type: BERT
  config_name: bert-base-uncased
  dropout_prob: 0.3
  label_map_or_file: data/taskB_train_label_map.joblib
  metrics:
  - seq_f1
```

(continues on next page)

(continued from previous page)

```
  - seq_precision
  - seq_recall
  loss_type: NERLoss
  task_type: NER
  file_names:
  - taskB_train.tsv
  - taskB_dev.tsv
  - taskB_test.tsv

TaskC:
  model_type: BERT
  config_name: bert-base-uncased
  dropout_prob: 0.05
  metrics:
  - accuracy
  loss_type: CrossEntropyLoss
  class_num: 2
  task_type: SentencePairClassification
  file_names:
  - taskC_train.tsv
  - taskC_dev.tsv
  - taskC_test.tsv
```

Few points to keep in mind while making the task file,

- You can keep the tasks which you want to train a single model for in this file.
- The file can have either a single task or multiple tasks. In case only a single task is mentioned, the model will act like single-task model.
- The task names (TaskA, TaskB and TaskC) are unique identifiers for the task, hence the task names must always be distinct.
- The model type for all the tasks mentioned in the file must be the same, as the library uses a single shared encoder model for all these tasks.

5.4.1 Task file parameters

Detailed description of the parameters available in the task file.

- `task_type` *(required)* : Format of the task as described in *Task types*
- `file_names` *(required)* : List of standard data tsv file names required for task. The first file is considered as **train** file, second file as **dev** file and the third file as **test** file.
- `model_type` *(required)* : Type of shared encoder model to use. The model type for all the tasks mentioned in the file must be the same. You can refer *Model type* for selecting model type.
- `config_name` *(optional)* : Config of the encoder model. You can refer *Model type* for selecting the model type config. In case this parameter is not present, default config will be used.
- `class_num` *(required/optional)* : Number of classes present for classification. This parameter is optional if label_map_or_file is provided, required otherwise.
- `label_map_or_file` *(required/optional)* :
  - In case labels are strings, this is the list of unique labels.
  - You can also give a joblib dumped dictionary map file like {'label1':0, 'label2':1, ..}.

  - If you're using *Data Transformations* to create the data files, path to the label_map file created along with transformed files is to be given here.

- `loss_type` *(required)* : Type of loss for training as defined in *Losses*.
- `dropout_prob` *(optional)*: Dropout probability to use between encoder hidden outputs and task specific headers.
- `metrics` *(optional)* : List of metrics to use during evaluation as defined in *Metrics*.
- `loss_weight` *(optional)*: Loss weight value (between 0 to 1) for individual task.

5.5 How to train?

Once you have made the task file with the tasks you want to train for, the next step is to run `data_preparation.py` and `train.py`.

5.5.1 Running data preparation

- The job of this script is to convert the given tsv data files to model inputs such as **Token Ids**, **Attention Masks** and **Token Type Ids** based on the shared encoder type.
- The script uses **multi-processing** which effectively reduces the data preparation time for large data files.
- It stores the prepared data in json files under the directory name **prepared_data** prefixed with the shared encoder config name.

The script takes the following arguments,

- `task_file` *(required)* :- Path to the created task file for which you want to train.
- `data_dir` *(required)* :- Path to the directory where the data files mentioned in task file are present.
- `do_lower_case` *(optional, default True)* :- Set this to False in case you are using a *cased* config for model type.
- `max_seq_len` *(required, default 128)* :- Maximum sequence length for inputs. Truncating or padding will occur accordingly.

You can use the following terminal command with your own argument values to run.

```
$ python data_preparation.py \
      --task_file 'sample_task_file.yml' \
      --data_dir 'data' \
      --max_seq_len 50
```

5.5.2 Running train

After `data_preparation.py` has finished running, it will store the respective prepared files under the directory name 'prepared_data' prefixed with the shared encoder config name. The `train.py` can be run from terminal to start the training. Following arguments are available

- `data_dir` *(required)* :- Path to the directory where prepared data is stored. (eg. bert_base_uncased_prepared_data)
- `task_file` *(required)* :- Path to task file for training.
- `out_dir` *(required)* :- Path to save the multi-task model checkpoints.

- `epochs` *(required)* :- Number of epochs to train.
- `train_batch_size` *(optional, default 8)* :- Batch size for training.
- `eval_batch_size` *(optional, default 32)* :- Batch size for evaluation.
- `grad_accumulation_steps` *(optional, default 1)* :- Number of batches to accumulate before update.
- `log_per_updates` *(optional, default 10)* :- Number of updates after which to log loss.
- `silent` *(optional, default True)* :- Set to False for logs to be shown on terminal output as well.
- `max_seq_len` *(optional, default 128)* :- Maximum sequence length which was used during data preparation.
- `save_per_updates` *(optional, default 0)* :- Number of update steps after which model checkpoint to be saved. Model is always saved at the end of every epoch.
- `load_saved_model` *(optional, default None)* :- Path to the saved model in case of loading.
- `resume_train` *(optional, default False)* :- Set to True for resuming training from the saved model. Training will resume from the step at which the loaded model was saved.

You can use the following terminal command with your own argument values to run.

```
$ python train.py \
      --data_dir 'data/bert-base-uncased_prepared_data' \
      --task_file 'sample_task_file.yml' \
      --out_dir 'sample_out' \
      --epochs 5 \
      --train_batch_size 4 \
      --eval_batch_size 8 \
      --grad_accumulation_steps 2 \
      --max_seq_len 50 \
      --log_per_updates 25 \
      --save_per_updates 1000 \
      --eval_while_train \
      --test_while_train  \
      --silent
```

5.5.3 Logs and tensorboard

- Logs for the training should be saved in a time-stamp named directory (eg. 05_05-17_30).
- The tensorboard logs are also present in the same directory and tensorboard can be started with the following command

```
$ tensorboard --logdir 05_05-17_30/tb_logs
```

5.6 How to Infer?

Once you have a multi-task model trained on your tasks, we provide a convenient and easy way to use it for getting predictions on samples through the **inference pipeline**.

class `infer_pipeline.`**`inferPipeline`**(*modelPath*, *maxSeqLen=128*)

For running inference on samples using a trained model for say TaskA, TaskB and TaskC, you can import this class and load the corresponding multi-task model by making an object of this class with the following arguments

Parameters

- **modelPath** (str) – Path to the trained multi-task model for required tasks.
- **maxSeqLen** (int, defaults to 128) – maximum sequence length to be considered for samples. Truncating and padding will happen accordingly.

Example:

```
>>> from infer_pipeline import inferPipeline
>>> pipe = inferPipeline(modelPath = 'sample_out_dir/multi_task_model.pt',␣
↪maxSeqLen = 50)
```

infer(*dataList*, *taskNamesList*, *batchSize=8*, *seed=42*)

This is the function which can be called to get the predictions for input samples for the mentioned tasks.

- Samples can be packed in a list of lists manner as the function processes inputs in batch.
- In case, an input sample requires sentence pair, the two sentences can be kept as elements of the list.
- In case of single sentence classification or NER tasks, only the first element of a sample will be used.
- For NER, the infer function automatically splits the sentence into tokens.
- All the tasks mentioned in taskNamesList are performed for all the input samples.

Parameters

- **dataList** (list of lists) – A batch of input samples. For eg.

  [[<sentenceA>, <sentenceB>],

  [<sentenceA>, <sentenceB>],

  ]

  or in case all the tasks just require single sentence inputs,

  [[<sentenceA>],

  [<sentenceA>],

  ]

- **taskNamesList** (list) – List of tasks to be performed on dataList samples. For eg.

  ['TaskA', 'TaskB', 'TaskC']

  You can choose the tasks you want to infer. For eg.

  ['TaskB']

- **batchSize** (int, defaults to 8) – Batch size for running inference.

Returns

List of dictionary objects where each object contains one corresponding input sample and it's tasks outputs. The task outputs can also contain the confidence scores. For eg.

[{'Query' : [<sentence>],

'TaskA' : <TaskA output>,

'TaskB' : <TaskB output>,

'TaskC' : <TaskC output>},

]

Return type outList (list of objects)

Example:

```
>>> samples = [ ['sample_sentence_1'], ['sample_sentence_2'] ]
>>> tasks = ['TaskA', 'TaskB']
>>> pipe.infer(samples, tasks)
```

5.7 License

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

   (a) You must give any other recipients of the Work or Derivative Works a copy of this License; and

   (b) You must cause any modified files to carry prominent notices stating that You changed the files; and

   (c) You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

   (d) If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

   You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work.

> To apply the Apache License to your work, attach the following boilerplate notice, with the fields enclosed by brackets "[]" replaced with your own identifying information. (Don't include the brackets!) The text should be enclosed in the appropriate comment syntax for the file format. We also recommend that a file or class name and description of purpose be included on the same "printed page" as the copyright notice for easier identification within third-party archives.

Copyright [yyyy] [name of copyright owner]

Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at

> http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

Python Module Index

u

Index