==========
====== **README** ==========
==========

This package contains the code underlying the SCAN model as described in
Frermann and Lapata (2016). The code can be used to (a) create binary files
containing time-stamped input documents (the required input to SCAN); and (b)
to train scan models.

This bundle contains three parts
(1) main.zip
    the main functions. Run code from this directory. Also contains example
    input and output
(2) scan_lib.zip
    the core code underlying SCAN (my code)
(3) other_lib.zip
    libraries that are used but not part of the core go libraries. Each of these must
    be placed in the same directory as scan_lib.zip

To run the code, navigate into your 'main' directory and run

EITHER the pre-compiled binary (should work out of the box)
./tracking-meaning -parameter_file=/path/to/parameters/ [-create_corpus] -store={true,false}

OR the code itself (requires to have go installed)
go run *.go -parameter_file=/path/to/parameters/ [-create_corpus] -store={true,false}

The command-line parameters are:

- parameter_file
    path to a text file containing all parameters (see below)

-create_corpus
    an optional parameter. If it is set a binary corpus will be created. Otherwise a model will be trained

-store
    indicates whether target word-specific corpora should be stored

To run the code itself (second option above i.e., not just the binary) golang (https://golang.org/doc/)
must be installed. I use version go1.7.4 (might be safest to use the same). All required
packages to run this code should be part of this bundle.

----- **running out of the box** -----

I include an example for all necessary test files:
- a parameters.txt (which has my own hard-coded paths, MUST BE CHANGED)
- corpus file under main/test_input/corpus.txt
- a file with target words under main/test_input/targets.txt

You need to change the paths in the parameters.txt to your own paths (to the corpus / targets fies),
and run

(a) to create a binary corpus
./tracking-meaning -parameter_file=path/to/parameters.txt -create_corpus -store=true

(b) to train models
./tracking-meaning -parameter_file=path/to/parameters.txt -store=true

---
--- **the parameters file** ---

All parameters are specified in a parameters which must be passed as input to
the program. Best see the included 'parameters.txt' examples file. This includes

- paths to underlying text corpora, target word sets, etc
- model parameters (see paper for explanation)
- sampler parameters (number of iterations)
- parameters regarding the time start/end/intervals of interest
- parameters to optionally restrict the minimum number of available documents (to ignore highly
  infrequent words) and / or maximum number of available documents per time interval
  (to get manageable-size corpora for very high-frequent words)

Creating binary input corpora ---

Takes a text file and a list of target words, and a 'document length' specification. Outputs
a binary corpus with target-word specific, time-stamped documents. Document length refers to
the size of the context window considered around the target word, e.g., 5 words.
[The corpus has words mapped to unique ID identifiers, and contains dictionaries mapping from
 word strings to IDs and back]

It takes the following parameters (all specified in parameters.txt)

- text_corpus
    path to a text file which in each line contains a number indicating
    a year (of origin) followed by a \tab\ character followed by the corresponding text
    from that year of origin. The same year can be listed multiple times:

    YEAR \tab\ text ....
    YEAR \tab\ text ....
    ....

- target_words
    path to a text file containing all target words of interest
    whose meaning should be tracked, one word per line.

- window_size
    It also requires a specification of the window size (i.e, the context window to consider, as
    explained above)

- bin_corpus_store
    path to the location the binary corpus should be stored

--- Training a SCAN model ---

Once we have a binary corpus of time-stamped documents as explained above full_corpus_path, we can train SCAN
models that track meaning change of individual target words. To do this we

(1) extract a target word-specific corpus from the underlying binary corpus. It contains only time-tagged documents with the specified target word. It converts the absolute times in the underlying corpus (e.g, 1764, 1993, ...)
to time intervals (0, 1, ..., T) based on the start_time, end_time and time_interval parameters (see below).
It takes the following parameters (all specified in parameters.txt):

- full_corpus_path
  path to the underlying binary corpus

- start_time
  the earliest time stamp in the underlying corpus to be considered

- end_time
  the latest time stamp in the underlying corpus to be considered

- time_interval
  the length of time intervals into which the span [start_time, end_time] is to be split

  e.g., if start_time=1700 , end_time=2000 , time_interval=10 then documents are binned into 10-year bins
  and all documents from before 1700 and after 2000 are ignored. Documents from 1700-1709 are assigned to
  bin 0, documents from 1710-1719 are assigned to bin 1 and so on.

- word_corpus_path
  path to a location where word-specific corpora are stored.

  The filename reflects the choice of start_time / end_time / time_interval,
  e.g., corpus_s1700_e2009_i10.bin for the example above

(2) pass this corpus to the model and train the model with MCMC inference. It creates a model and human-readable output:

- output.dat the trained model binary and
- model.bin human-readable model output, namely for each time slice its distribution over senses, and for each sense in each time slice, it's distribution over words (as the set of most highly associated words).

It takes the following parameters (all specified in parameters.txt)

- output_path
  directory in which files output.dat and model.bin are to be stored

  If the output directory and files (a) and (b) already exist (from a previous run) it moves the old files to output_old.dat and model_old.bin

- kappaF, kappaK, a0, b0, num_top
  model parameters; check paper (or ask me!) for explanations

- iterations
  number of training iterations

- min_doc_per_word
  the model doesn't work well if there are very few documents for a target word available. You may want to
  only learn models per target words that occur at least N (~ 500?) times in the data

- max_docs_per_slice
  some words occur extremely often. To get a manageable size input corpus you can restrict the number of documents
  to consider per time interval

--- Understanding the output ---

The program creates a human-readable output file for each target word in [outptut_path]/word/output.dat

The included directories under main/test_input/output/ contain output for models trained on the corpus in main/test_input/corpus.txt for the target words in main/test_input/targets.txt

Models were trained to learn
 -- from the corpus.txt file (containing some language from between 1700 and 2009)
 -- K=8 senses per word
 -- Start_time=1700, end_time=2009, intervals=20 --> obtaining 16 20-year time intervals in total

In each output.dat file:
 -- K indexes the sense ID
 -- T indexes the time slice ID
 -- After each sense / time: The top 10 words with highest probability under each sense are listed
 -- The bottom line of each block (p(w|s)....) sums over all senses, i.e., shows the most highly associated words
    for a particular time, ignoring sense information.

The file output.dat contains the same information in two ways.

*** per type***
First, I list *by sense* the representation of the
same sense (k=0...K) for each time slice. The first number indicates the sense's prevalence at that particular time (as a probability, between 0 and 1). Look for example in main/example_input/output/power/output.dat.
We can see that sense 2 (the block with K=2) seems to relate to the 'electricity' sense of power (it has a highly associated words 'battery', 'plant', etc especially towards later times, and its prevalence increases towards later time slices.

*** per time ***
This lists the senses associated with each time interval (the content of the lines is the same as above). E.g., the times associated with T=0 (first block under 'per_time') shows that senses K=7 has high probability and sense K=2 (the 'power' sense including e.g., the word 'dynamo') has low probability. Senses K=2 refers to
the 'mental' power.

Output for the targets 'battery' and 'transport' is also included.