ctpf Documentation

David Cortes

May 30, 2022

Contents:

This is the documentation page for the python package *ctpfrec*. For more details, see the project's GitHub page:

https://www.github.com/david-cortes/ctpfrec/

CHAPTER 1

Installation

Package is available on PyPI, can be installed with

```
pip install ctpfrec
```

class ctpfrec.**CTPF**(*k=50*, *a=0.3*, *b=0.3*, *c=0.3*, *d=0.3*, *e=0.3*, *f=0.3*, *g=0.3*, *h=0.3*, *stop_crit='train-llk'*, *stop_thr=0.001*, *reindex=True*, *miniter=25*, *maxiter=70*, *check_every=10*, *verbose=True*, *use_float=True*, *random_seed=None*, *ncores=-1*, *initialize_hpf=True*, *standardize_items=False*, *rescale_factors=False*, *missing_items='include'*, *step_size=<function CTPF.<lambda>>*, *allow_inconsistent_math=False*, *full_llk=False*, *keep_data=True*, *save_folder=None*, *produce_dicts=True*, *sum_exp_trick=False*, *keep_all_objs=True*)

Bases: `object`

Collaborative Topic Poisson Factorization

Model for recommending items based on probabilistic Poisson factorization on sparse count data (e.g. number of times a user viewed different items) along with count data on item attributes (e.g. bag-of-words representation of text descriptions of items), using mean-field variational inference with coordinate-ascent. Can also accommodate user attributes in addition to item attributes (see note below for more information).

Can use different stopping criteria for the opimization procedure:

1) Run for a fixed number of iterations (stop_crit='maxiter').
2) Calculate the Poisson log-likelihood every N iterations (stop_crit='train-llk' and check_every) and stop once {1 - curr/prev} is below a certain threshold (stop_thr)
3) Calculate the Poisson log-likelihood in a user-provided validation set (stop_crit='val-llk', val_set, and check_every) and stop once {1 - curr/prev} is below a certain threshold. For this criterion, you might want to lower the default threshold (see Note).
4) Check the the difference in the Theta matrix after every N iterations (stop_crit='diff-norm', check_every) and stop once the *l2-norm* of this difference is below a certain threshold (stop_thr). Note that this is **not a percent** difference as it is for log-likelihood criteria, so you should put a larger value than the default here. This is a much faster criterion to calculate and is recommended for larger datasets.

If passing reindex=True, it will internally reindex all user and item IDs. Your data will not require reindexing if the IDs for users, items, and words (or other countable item attributes) in counts_df and words_df meet the following criteria:

1) Are all integers.
2) Start at zero.
3) Don't have any enumeration gaps, i.e. if there is a user '4', user '3' must also be there.

If you only want to obtain the fitted parameters and use your own API later for recommendations, you can pass produce_dicts=False and pass a folder where to save them in csv format (they are also available as numpy arrays in this object's Theta, Eta and Epsilon attributes). Otherwise, the model will create Python dictionaries with entries for each user, item, and word, which can take quite a bit of RAM memory. These can speed up predictions later through this package's API.

Passing verbose=True will also print RMSE (root mean squared error) at each iteration. For slighly better speed pass verbose=False once you know what a good threshold should be for your data.

Note: DataFrames and arrays passed to '.fit' might be modified inplace - if this is a problem you'll need to pass a copy to them, e.g. 'counts_df=counts_df.copy()'.

Note: If 'check_every' is not None and stop_crit is not 'diff-norm', it will, every N iterations, calculate the Poisson log-likelihood of the data. By default, this is NOT the full likelihood, (not including a constant that depends on the data but not on the parameters and which is quite slow to compute). The reason why it's calculated by default like this is because otherwise it can result in overflow (number is too big for the data type), but be aware that if not adding this constant, the number can turn positive and will mess with the stopping criterion for likelihood.

Note: If you pass a validation set, it will calculate the Poisson log-likelihood *of the non-zero observations only*, rather than the complete Poisson log-likelihood that includes also the combinations of users and items not present in the data (assumed to be zero), thus it's more likely that you might see positive numbers here. Compared to ALS, iterations from this algorithm are a lot faster to compute, so don't be scared about passing large numbers for maxiter.

Note: In some unlucky cases, the parameters will become NA in the first iteration, in which case you should see weird values for log-likelihood and RMSE. If this happens, try again with a different random seed.

Note: As this model will fit the parameters to both user-item interactions and item attributes, you might see the Poisson log-likelihood decreasing during iterations. This doesn't necessarily mean that it failed, but in such cases you might want to try increasing K or decreasing the number of item attributes.

Note: Can also fit a model that includes user attributes in the same format as the bag-of-words representations of items - in this case the new variables will be called Omega (user-factor matrix), Kappa (user_attribute-factor), X (multinomial for user attributes). The Y variables will be associated as follows: Ya (Omega, Theta) - Yb (Eta, Theta) - Yc (Omega, Epsilon) - Yd (Eta, Epsilon). The priors for these new parameters will be taken to be the same ones as their item counterparts.

Parameters

- **k** (*int*) – Number of latent factors (topics) to use.
- **a** (*float*) – Shape parameter for the word-topic prior (Beta). If fitting the model with user attributes, will also be taken as prior for the user_attribute-factor matrix.
- **b** (*float*) – Rate parameter for the word-topic prior (Beta). If fitting the model with user attributes, will also be taken as prior for the user_attribute-factor matrix.
- **c** (*float*) – Shape parameter document-topic prior (Theta).
- **d** (*float*) – Rate parameter document-topic prior (Theta).
- **e** (*float*) – Shape parameter for the user-topic prior (Eta).
- **f** (*float*) – Rate parameter for the user-topic prior (Eta).
- **g** (*float*) – Shape parameter for the document-topic offset prior (Epsilon). If fitting the model with user attributes, will also be taken as prior for the user-factor offset matrix.
- **h** (*float*) – Rate parameter for the document-topic offset prior (Epsilon). If fitting the model with user attributes, will also be taken as prior for the user-factor offset matrix.
- **stop_crit** (*str, one of 'maxiter', 'train-llk', 'val-llk', 'diff-norm'*) – Stopping criterion for the optimization procedure.
- **stop_thr** (*float*) – Threshold for proportion increase in log-likelihood or l2-norm for difference between matrices.
- **reindex** (*bool*) – Whether to reindex data internally. Will be forced to 'False' if passing sparse COO matrices to 'fit'.
- **miniter** (*int or None*) – Minimum number of iterations for which to run the optimization procedure. When using likelihood as a stopping criterion, note that as the model is fit to both user-item interactions and item attributes, the Poisson likelihood for the interactions alone can decrease during iterations as the complete model likelihood increases (and this Poisson likelihood might start increasing again later). Thus, a minimum number of iterations will avoid stopping when the Poisson likelihood decreases.
- **maxiter** (*int or None*) – Maximum number of iterations for which to run the optimization procedure. This corresponds to epochs when fitting in batches of users. Recommended to use a lower number when passing a batch size.
- **check_every** (*None or int*) – Calculate log-likelihood every N iterations.
- **verbose** (*bool*) – Whether to print convergence messages.
- **use_float** (*bool*) – Whether to use the C float type (typically `np.float32`). Using float types (as compared to double) results in less memory usage and faster operations, but it has less numeric precision and the results will be slightly worse compared to using double. If passing `False`, will use C double (typically `np.float64`).
- **random_seed** (*int or None*) – Random seed to use when starting the parameters.
- **ncores** (*int*) – Number of cores to use to parallelize computations. If set to -1, will use the maximum available on the computer.
- **initialize_hpf** (*bool*) – Whether to initialize the Theta and Beta matrices using hierarchical Poisson factorization on the bag-of-words representation only (words_df passed to 'fit'). This can provide better results than a random initialization, but it takes extra time to fit.
- **standardize_items** (*bool*) – Whether to standardize the item bag-of-words representations passed to '.fit' ('words_df') so that all the items have the same sum of words (set to the

mean sum of word counts across items). Will also apply to user attributes if passing them to '.fit'.

- **rescale_factors** (*bool*) – Whether to rescale the resulting item-factor matrix (Theta) after fitting to have its rows sum to 1. This decreases the model susceptibility to have items with more words be considered more popular, but it can also result in way worse rankings. Will also be applied to user factors if fitting the model with user attributes. (Not recommended)
- **missing_items** (*str, one of 'include' or 'exclude'*) – If there are items in the 'words_df' object to be passed to '.fit' that are not present in 'counts_df', shall they be considered as having all their user-item interactions with a count of zero (when passing 'include'), or shall they be considered to be censored (e.g. missing because the model is fit to bag-of-words of articles that are not available to users). In the second case, these items will be included when initializing with 'initialize_hpf=True', but will be excluded afterwards. In the second case, note that the model won't be able to make predictions for these items, but you can add them afterwards using the '.add_items' method. Same for user attributes when fitting the model with user side information. Note that this **only applies to extra items/users with side info but no interaction**, while any user-item interaction not present in the data is taken as include. Forced to 'include' when passing 'initialize_hpf=False' or 'reindex=False'.
- **step_size** (*None or function -> float in (0, 1)*) – Function that takes the iteration/epoch number as input (starting at zero) and produces the step size for the update to Beta and Theta. When initializing these through hierarchical Posisson factorization, it can be beneficial to have the first steps change them less or not change them at all, while the user and offset matrices start getting shaped towards these initialized topics, with later iterations being allowed to change them more (so it starts at zero and tends towards 1 as the iteration number increases). When using 'stop_crit=diff-norm', it will not stop if step_size(iteration)<=1e-2. You can also pass a function that always returns zero if you do not wish to update the Theta and Beta parameters obtained from HPF, but in that case you'll also need to change the stopping criterion. Will also apply to the Kappa parameter in the model with user attributes. Forced to None when passing 'initialize_hpf=False'.
- **allow_inconsistent_math** (*bool*) – Whether to allow inconsistent floating-point math (producing slightly different results on each run) which would allow parallelization of the updates for all of the shape parameters.
- **full_llk** (*bool*) – Whether to calculate the full log-likehood, including terms that don't depend on the model parameters (thus are constant for a given dataset).
- **keep_data** (*bool*) – Whether to keep information about which user was associated with each item in the training set, so as to exclude those items later when making Top-N recommendations.
- **save_folder** (*str or None*) – Folder where to save all model parameters as csv files.
- **produce_dicts** (*bool*) – Whether to produce Python dictionaries for users and items, which are used to speed-up the prediction API of this package. You can still predict without them, but it might take some additional miliseconds (or more depending on the number of users and items).
- **sum_exp_trick** (*bool*) – Whether to use the sum-exp trick when scaling the multinomial parameters - that is, calculating them as exp(val - maxval)/sum_{val}(exp(val - maxval)) in order to avoid numerical overflow if there are too large numbers. For this kind of model, it is unlikely that it will be required, and it adds a small overhead, but if you notice NaNs in the results or in the likelihood, you might give this option a try. Forced to True when passing 'initialize_hpf=True'. Will also be forced to True when passing user side information.
- **keep_all_objs** (*bool*) – Whether to keep intermediate objects/variables in the object that are not necessary for predictions - these are: Gamma_shp, Gamma_rte, Lambda_shp,

Lambda_rte, k_rte, t_rte (when passing True here, the model object will have these extra attributes too). Without these objects, it's not possible to call functions that alter the model parameters given new information after it's already fit.

Variables

- **`Theta`** (*`array (nitems, k)`*) – Item-topic matrix.
- **`Beta`** (*`array (nwords, k)`*) – Word-topic matrix. Only kept when passing 'keep_all_objs=True'
- **`Eta`** (*`array (nusers, k)`*) – User-topic matrix
- **`Epsilon`** (*`array (nitems, k)`*) – Item-topic offset matrix
- **`user_mapping`** (*`array (nusers,)`*) – ID of the user (as passed to .fit) of each row of Eta.
- **`item_mapping`** (*`array (nitems,)`*) – ID of the item (as passed to .fit) of each row of Beta.
- **`word_mapping`** (*`array (nwords,)`*) – ID of the word (as passed to .fit) of each row of Theta and Epsilon.
- **`user_dict`** (*`dict (nusers)`*) – Dictionary with the mapping between user IDs (as passed to .fit) and rows of Eta.
- **`item_dict`** (*`dict (nitems)`*) – Dictionary with the mapping between item IDs (as passed to .fit) and rows of Theta and Epsilon.
- **`word_dict`** (*`dict (nwords)`*) – Dictionary with the mapping between item IDs (as passed to .fit) and rows of Beta.
- **`is_fitted`** (*`bool`*) – Whether the model has been fit to some data.
- **`niter`** (*`int`*) – Number of iterations for which the fitting procedure was run.

References

[1] Content-based recommendations with poisson factorization (Gopalan, P.K., Charlin, L. and Blei, D., 2014)

`add_items` (*words_df*, *maxiter=10*, *stop_thr=0.001*, *ncores=1*, *random_seed=10*)

Adds new items to an already fit model

Adds new items without refitting the model from scratch. Note that this will not modify any of the user or word parameters.

For better results, refit the model from scratch including the data from these new items.

Note: This function is prone to producing all NaNs values. Adding both users and items to already-fit model might cause very bad quality results for both.

Parameters

- **words_df** (*data frame or array (n_samples, 3)*) – DataFrame with the bag-of-words representation of the new items only. Must contain columns 'ItemId', 'WordId', 'Count'. If passing a numpy array, columns will be assumed to be in that order. When using 'reindex=False', the numeration must start right after the last item ID that was present in the training data.
- **maxiter** (*int*) – Maximum number of iterations for which to run the procedure.

- **stop_thr** (*float*) – Will stop if the norm of the difference between the shape parameters after an iteration is below this threshold.
- **ncores** (*int*) – Number of threads/core to use. When there is few data, it's unlikely that using multiple threads would give a significant speed-up, and it might even end up making the function slower due to the overhead.
- **random_seed** (*int or None:*) – Random seed to be used for the initialization of the new shape parameters.

Returns **True** – Will return True if the procedure terminates successfully.

Return type bool

`add_users`(*counts_df=None*, *user_df=None*, *maxiter=10*, *stop_thr=0.001*, *ncores=1*, *random_seed=10*)

Adds new users to an already fit model

Adds new users without refitting the model from scratch. Note that this will not modify any of the item or word parameters. In the regular model, you will need to provide "counts_df" as input, and the parameters will be determined according to the user-item interactions. If fitting the model with user attributes, you will also need to provide "user_df". Not providind a 'counts_df' object will assume that all the interactions for this user are zero (only supported in the model with user attributes).

For better results, refit the model from scratch including the data from these new users.

Note: This function is prone to producing all NaNs values. Adding both users and items to already-fit model might cause very bad quality results for both.

Parameters

- **counts_df** (*data frame or array (n_samples, 3)*) – DataFrame with the user-item interactios for the new users only. Must contain columns 'UserId', 'ItemId', 'Count'. If passing a numpy array, columns will be assumed to be in that order.
- **user_df** (*data frame or array (n_samples, 3)*) – DataFrame with the user attributes for the new users only. Must contain columns 'UserId', 'AttributeId', 'Count'. If passing a numpy array, columns will be assumed to be in that order. Only for models with to user side information.
- **maxiter** (*int*) – Maximum number of iterations for which to run the procedure.
- **stop_thr** (*float*) – Will stop if the norm of the difference between the shape parameters after an iteration is below this threshold.
- **ncores** (*int*) – Number of threads/core to use. When there is few data, it's unlikely that using multiple threads would give a significant speed-up, and it might even end up making the function slower due to the overhead.
- **random_seed** (*int or None:*) – Random seed to be used for the initialization of the new shape parameters.

Returns **True** – Will return True if the procedure terminates successfully.

Return type bool

`eval_llk`(*counts_df*, *full_llk=False*)

Evaluate Poisson log-likelihood (plus constant) for a given dataset

Note: This log-likelihood is calculated only for the combinations of users and items provided here, so it's not a complete likelihood, and it might sometimes turn out to be a positive number because of this. Will filter out the input data by taking only combinations of users and items that were present in the training set.

Parameters

- **counts_df** (*pandas data frame (nobs, 3)*) – Input data on which to calculate log-likelihood, consisting of IDs and counts. Must contain one row per non-zero observaion, with columns 'UserId', 'ItemId', 'Count'. If a numpy array is provided, will assume the first 3 columns contain that info.
- **full_llk** (*bool*) – Whether to calculate terms of the likelihood that depend on the data but not on the parameters. Ommitting them is faster, but it's more likely to result in positive values.

Returns **llk** – Dictionary containing the calculated log-likelihood and the number of observations that were used to calculate it.

Return type dict

`fit` (*counts_df*, *words_df*, *user_df=None*, *val_set=None*)
Fit Collaborative Topic Poisson Factorization model to sparse count data

Note: DataFrames and arrays passed to '.fit' might be modified inplace - if this is a problem you'll need to pass a copy to them, e.g. 'counts_df=counts_df.copy()'.

Note: Forcibly terminating the procedure should still keep the last calculated shape and rate parameter values, but is not recommended. If you need to make predictions on a forced-terminated object, set the attribute 'is_fitted' to 'True'.

Parameters

- **counts_df** (*DatFrame(n_samples, 3) or sparse COO(n_users, n_items)*) – User-Item interaction data with one row per non-zero observation, consisting of triplets ('UserId', 'ItemId', 'Count'). Must containin columns 'UserId', 'ItemId', and 'Count'. Combinations of users and items not present are implicitly assumed to be zero by the model. If passing a COO matrix, will set `self.reindex=False`.
- **words_df** (*DatFrame(n_samples, 3) or sparse COO(n_items, n_words)*) – Bag-of-word representation of items with one row per present unique word, consisting of triplets ('ItemId', 'WordId', 'Count'). Must contain columns 'ItemId', 'WordId', and 'Count'. Combinations of items and words not present are implicitly assumed to be zero. Must be of the same type ('DataFrame' or 'coo_matrix') as `counts_df`.
- **user_df** (*DatFrame(n_samples, 3), or sparse COO(n_users, n_attr)*) – User attributes, same format as 'words_df'. Must contain columns 'UserId', 'AttributeId', 'Count'. Must be of the same type ('DataFrame' or 'coo_matrix') as `counts_df`.
- **val_set** (*DatFrame(n_samples, 3), or sparse COO(n_users, n_items)*) – Validation set on which to monitor log-likelihood. Same format as `counts_df`.

Returns **self** – Copy of this object

Return type obj

`predict` (*user*, *item*)
Predict count for combinations of users and items

Note: You can either pass an individual user and item, or arrays representing tuples (UserId, ItemId) with the combinatinons of users and items for which to predict (one row per prediction).

Parameters

- **user** (*array-like (npred,) or obj*) – User(s) for which to predict each item.
- **item** (*array-like (npred,) or obj*) – Item(s) for which to predict for each user.

`predict_item_factors` (*words_df*, *maxiter=10*, *ncores=1*, *random_seed=1*, *stop_thr=0.001*, *return_all=False*)
Obtain latent factors/topics for items given their bag-of-words representation alone

Note: For better results, refit the model again including these items.

Note: If passing more than one item, the resulting rows will be in the sorted order of the item IDs from user_df (e.g. if users are 'b', 'a', 'c', the first row will contain the factors for item 'a', second for 'b', third for 'c'.

Note: This function is prone to producing all NaNs values.

Parameters

- **words_df** (*DataFrame (n_samples, 3)*) – Bag-of-words representation of the items to predict. Same format as the one passed to '.fit'.
- **maxiter** (*int*) – Maximum number of iterations for which to run the inference procedure.
- **ncores** (*int*) – Number of threads/cores to use. With data for only one user, it's unlikely that using multiple threads would give a significant speed-up, and it might even end up making the function slower due to the overhead. If passing -1, it will determine the maximum number of cores in the system and use that.
- **random_seed** (*int*) – Random seed used to initialize parameters.
- **stop_thr** (*float*) – If the l2-norm of the difference between values of Theta_{i} between interations is less than this, it will stop. Smaller values of 'k' should require smaller thresholds.
- **return_all** (*bool*) – Whether to return also the intermediate calculations (Theta_shp, Theta_rte). When passing True here, the output will be a tuple containing (Theta, Theta_shp, Theta_rte, Phi)

Returns **factors** – Obtained latent factors/topics for these items.

Return type array (nitems, k)

predict_user_factors(*user_df*, *maxiter=10*, *ncores=1*, *random_seed=1*, *stop_thr=0.001*, *return_all=False*)

Obtain latent factors/topics for users given their attributes alone

Note: For better results, refit the model again including these users.

Note: If passing more than one user, the resulting rows will be in the sorted order of the user IDs from user_df (e.g. if users are 'b', 'a', 'c', the first row will contain the factors for user 'a', second for 'b', third for 'c'.

Note: This function is prone to producing all NaNs values.

Parameters

- **user_df** (*DataFrame (n_samples, 3)*) – Attributes of the items to predict. Same format as the one passed to '.fit'.
- **maxiter** (*int*) – Maximum number of iterations for which to run the inference procedure.
- **ncores** (*int*) – Number of threads/cores to use. With data for only one user, it's unlikely that using multiple threads would give a significant speed-up, and it might even end up making the function slower due to the overhead. If passing -1, it will determine the maximum number of cores in the system and use that.
- **random_seed** (*int*) – Random seed used to initialize parameters.
- **stop_thr** (*float*) – If the l2-norm of the difference between values of Theta_{i} between interations is less than this, it will stop. Smaller values of 'k' should require smaller thresholds.
- **return_all** (*bool*) – Whether to return also the intermediate calculations (Z). When passing True here, the output will be a tuple containing (Theta_shp, Z)
- **only_shape** (*bool*) – Whether to return only the shape parameter for Theta, instead of dividing it by the rate parameter.

Returns **factors** – Obtained latent factors/topics for these items.

Return type array (nitems, k)

topN(*user*, *n=10*, *exclude_seen=True*, *items_pool=None*)

Recommend Top-N items for a user

Outputs the Top-N items according to score predicted by the model. Can exclude the items for the user that were associated to her in the training set, and can also recommend from only a subset of user-provided items.

Parameters

- **user** (*obj*) – User for which to recommend.
- **n** (*int*) – Number of top items to recommend.
- **exclude_seen** (*bool*) – Whether to exclude items that were associated to the user in the training set.
- **items_pool** (*None or array*) – Items to consider for recommending to the user.

Returns **rec** – Top-N recommended items.

Return type array (n,)

`topN_cold`(*user_df*, *n=10*, *items_pool=None*, *maxiter=10*, *ncores=1*, *random_seed=1*, *stop_thr=0.001*)

Recommend Top-N items for a user that was not in the training set.

Note: This function is only available if fitting a model that uses user attributes.

Note: The data passed to this function might be modified inplace. Be sure to pass a copy of the 'user_df' object if this is a problem.

Parameters

- **attributes** (*data frame (n_samples, 2)*) – Attributes of the user. Must have columns 'AttributeId', 'Count'.
- **n** (*int*) – Number of top items to recommend.
- **items_pool** (*None or array*) – Items to consider for recommending to the user.

Returns **rec** – Top-N recommended items.

Return type array (n,)

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Python Module Index

Index