ENT Rank-Lips

Laura Dietz

August 2, 2020

1 Introduction

Recently, the ENT rank model lead to successful improvements on Entity Ranking task by combining features of entities, with features of neighboring entities and the relevance of the mention context. At the heart of this method is a new learning-to-rank method that allows to learn relevance weights not just for the entities, but also relevance weights for neighbor relationships and the context—which are optimized together for a learning to rank metric such as MAP or NDCG using coordinate ascent.

However, the underlying machine learning method is much more general than entity ranking, and can be applied to any ranking problem where target entries e of a type E (e.g. entities) are to be ranked with respect to a query q, and entries of contextual types such as $x \in X$ and $y \in Y$ are pertinent for the ranking decision. Where traditional learning to rank methods only accept features to the target entity type, $\vec{f}(e)$, ENT-rank-lips will additionally accept features for contextual types $x \in X$ or $y \in Y$ such as $\vec{f}(x)$ or $\vec{f}(y)$. The relevance as described in the features of x and y is transfered to the target type e through the provided associations.

Features describing the relevance of combinations of contextual types x, y with a target type such as $\vec{f}(\{e\}, \{x\}, \{y\})$ are accepted. In fact any subset of contextual types and target entities are accepted, such as $\vec{f}(e, \{x_1, x_2\}, \emptyset)$, where two elements of type X are pertinent to rank one target entry e, while no statement about contextual types Y is made for this feature.

The following inputs need to be available for ENT-rank-lips to be applicable:
For a set of training/test queries $q \in Q$:

- query-specific candidate pool of target entries e of type E
- for training only: a query-specific ground truth of relevance for target type E.
- query-specific associations A_q of target entries $e \in E$ with contextual entries X or Y that are pertinent for the ranking decision. In particular: subsets of pertinent associations $A \subset \wp(E) \times \wp(X) \times \wp(Y)$,[1] so that for $(\{e\}, \{x\}, \{y\}) \in A$ the relevance of $x \in X$ and the relevance of $y \in Y$ will help deciding the relevance of $e \in E$. We will call x and y contextual entries. — we use powersets $\wp(..)$ because our formalism allows to associate multiple elements of $x_1, x_2 \in X$ with $e \in E$ without stating any connection to elements of Y, in this example $A = (\{e\}, \{x_1, x_2\}, \emptyset)$.
- query-specific relevance features can be derived for some of the following:
  - features of target entries $e \in E$ (i.e., those that are to be ranked)
  - features of contextual entries of other types, say $x \in X$ or $y \in Y$
  - combinations of target entries and contextual entries $(e, x) \in E \times X$
  - or multi-way combinations such as $(e, x, y) \in E \times X \times Y$

Training: Given a specification of such a candidate pools, training ground truths, associations, and relevance features, ENT-rank-lips trains a linear model optimize rank quality across all training queries, in terms of MAP (using the relevance ground truth).

Prediction: Given a model and candidate pools, associations, and relevance features, ENT-rank-lips will predict a ranking for each query.

[1] With $\wp(X)$ being the powerset of X.

Algorithm 1 Integrating features over associations for computing $\vec{\mathrm{f}}(e, A_{qe}|q)$.

```
[ ( targetEntry
    , M.fromListWith (+)  -- sum scores of features with the same name
    $ scale normalizationFactor featList  -- multiply each feature value with 1/|a|
    )
| (featureEntry, featList) <- M.toList featMap -- featureEntry is document in run.
                      -- featList :: [(featName, fvalue)]
, let !targetAssocs = matchingAssocs queryId featureEntry
                      --  list of targetEntries $e$ associated with this feature
, let !normalizationFactor = 1.0 / length targetAssocs
                      -- 1/|a|, number of entities asociated with this feature
, targetEntry <- targetAssocs
]

matchingAssocs queyId featureEntry =
[ projectTarget assoc -- emit all target entries,
                    -- if a=({e1,e2}, {p1}, ....), it will emit {e1, e2}
 | let assocs = queryId `M.lookup` associations -- $A_q$
, assoc <- assocs  -- a
, partialMatch featureEntry assoc -- True, iff current featureEntry matches an association
                  -- (if the current feature is relevant for an entry in the candidate set)
]
```

2 Underlying ENT-rank-lips Model

ENT-rank-lips uses a linear model represented by weight vector $\boldsymbol{\theta}$, to predict the query-specific rank score of a target entry e for query q, by marginalizing over all associations A that involve the target entry e.

$$\mathrm{score}_q(e) = \int_{A_{qe}} \left(\sum_{a \in A_{qe}} \boldsymbol{\theta} \frac{1}{|a|} \vec{\mathrm{f}}(a|q) \right) \tag{1}$$

We use the shorthand A_{qe} to denote all of the query's associations $(E_i, \dots) \in A_{qe}$ that involve the target entry, i.e., $e \in E_i$. For an association $a = (E_i, \dots)$ we denote the number of target entries in E_i as $|a|$.

The feature vector $\vec{\mathrm{f}}_q(a)$ of an association a will consist of all features given of the association set. Note that one association $a = (E_i, \dots) \in A_{qe}$ can related multiple target entries E_i, in which case $|a| > 1$. If some feature f_i is undefined for the association a, a user-specified neutral value will be used, e.g. 0.

The effect of the model is that in order to produce a rank score for entry e, we use the linear parameters $\boldsymbol{\theta}$ to predict a pertinence weight for each association a, then marginalize the pertinence weights for all associations that the target entry e is involved in. Whenever more than one target entry is involved in an association, the effect of the scalar factor $\frac{1}{|a|}$ is to evenly split the pertinence weights among the target entries.

Of course, each target entry can also participate in an association solely with itself, e.g. $A_{qe} = \{(\{e\}, \emptyset, \dots \emptyset\}$, in which case the feature vector $\vec{\mathrm{f}}_q(a) = \vec{\mathrm{f}}(e)$ is a typical feature vector for a target entry used in traditional learning-to-rank settings, and the rank score reduces to

$$\mathrm{score}_q(e) = \boldsymbol{\theta} \vec{\mathrm{f}}(e, A_{qe}|q) \tag{2}$$

Since we are focusing on ranking problems, where the rank score of an entry is a relative measure of relevance in comparison to other target candidates, we can ignore all proportional constants and log factors in the score function. This allows us to perform linear projections, such as z-score normalization on the features without affecting the rank quality. This is convenient, because z-score normalization allows for a better optimization problem, especially when using coordinate ascent to optimize for non-differentiable evaluation metrics such as mean-average precision.

3 Example: ENT Rank

Figure 1 displays an example configuration of ENT-rank-lips for entity ranking with ENT Rank (Dietz, 2019).

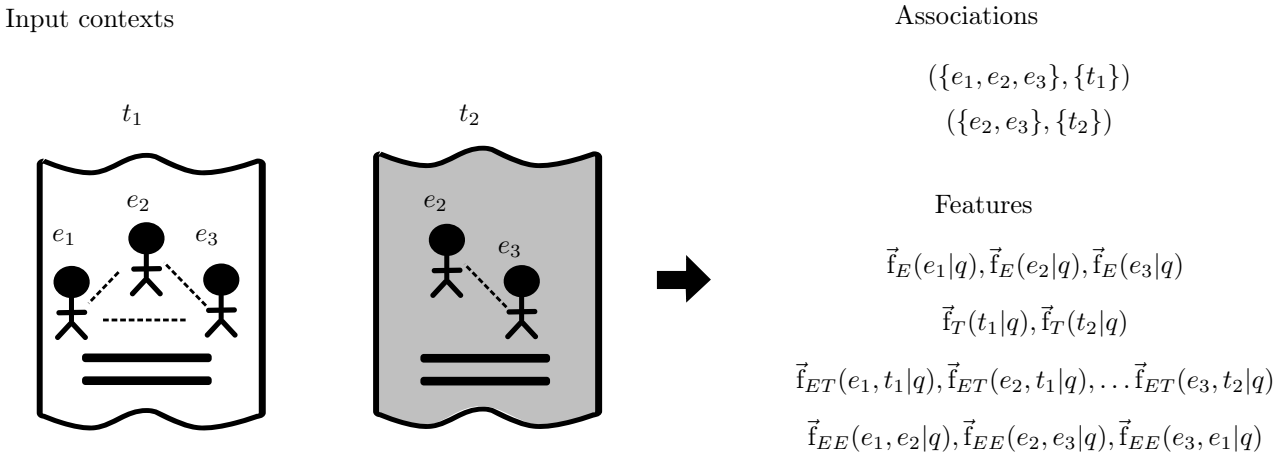

Figure 1: ENT-rank-lips example for entity ranking with neighbors and context. Left three entities e_1, e_2, e_3 mentioned in two contexts t_1, t_2. Right: features and associations.

In response to a given query, the task is to rank the entities e_1, e_2, e_3 by exploiting information about the relevance of their context and neighbors.

For example if the relevance of a context t_2 can be modeled through a relevance feature $\vec{\mathrm{f}}_T(t_1|q)$. For neighbors e_1 and e_2, neighbor features $\vec{\mathrm{f}}_{EE}(e_1, e_2|q)$ can express that if e_2 if relevant, then it is likely that e_1 is also relevant. ENT rank also incorporates all traditional entity features featyre Furthermore, ENT Rank can incorporate combination features such as $\vec{\mathrm{f}}_{ET}(e_1, t_1|q)$that describe the joint relevance of entity e_1 and its context t_1. One example of such an f_{ET} feature is the number of query terms in text t_1 that are near the mention of entity e_1. While it is similar to a text relevance feature f_T, it allows to more fine-grained features to be incorporated into the learning problem.

All that remains to define are query-specific associations, of which ENT-rank derives association per context t, based on all entities e_j that are mentioned therein: $A_{qe} = (\{e_j|\forall e_j \in t\}, \{t\})$

Given such features, ENT-rank-lips will define the relevance score (Equation 1) as

$$\begin{aligned}
\text{score}_q(e) = &\boldsymbol{\theta}_E \vec{\mathrm{f}}_E(e|q) \\
&+ \sum_{t_i : e \in t_i} \boldsymbol{\theta}_T \frac{1}{|\{e \in t_i\}|} \vec{\mathrm{f}}_T(t_i|q) \\
&+ \sum_{(\{e, e_j, \ldots\}, \{t_i, \ldots\}) \in A_{qe}} \frac{1}{|\{e_j \in t_i\}|} \boldsymbol{\theta}_{ET} \vec{\mathrm{f}}_{ET}(e_j, t_i|q) \\
&+ \sum_{(\{e, e_j, \ldots\}, \ldots) \in A_{qe}} \frac{1}{|\text{neighbors of } e|} \boldsymbol{\theta}_{EE} \vec{\mathrm{f}}_{EE}(e, e_j|q)
\end{aligned}$$

where the sum includes all associations A_{qe} that involve the target entity e and neighbor e_i and/or context t_i respectively.

Because of the associativity of inner products and sums, this is can be re-arranged as in Equation 2 to a single inner product of stacked weight parameter vectors $\boldsymbol{\theta}$ and various marginalized feature vectors $\vec{\mathrm{f}}(e, A_{qe})$

$$\begin{aligned}
\text{score}_q(e) &= \begin{pmatrix} \boldsymbol{\theta}_E \\ \boldsymbol{\theta}_T \\ \boldsymbol{\theta}_{ET} \\ \boldsymbol{\theta}_{EE} \end{pmatrix} \begin{pmatrix} \vec{\mathrm{f}}_E(e|q) \\ \sum_{t_i : e \in t_i} \frac{1}{|\{e \in t_i\}|} \vec{\mathrm{f}}_T(t_i|q) \\ \sum_{(\{e, e_j, \ldots\}, \{t_i, \ldots\}) \in A_{qe}} \frac{1}{|\{e_j \in t_i\}|} \vec{\mathrm{f}}_{ET}(e_j, t_i|q) \\ \sum_{(\{e, e_j, \ldots\}, \ldots) \in A_{qe}} \frac{1}{|\text{neighbors of } e|} \vec{\mathrm{f}}_{EE}(e, e_j|q) \end{pmatrix} \\
&= \boldsymbol{\theta} \vec{\mathrm{f}}(e, A_{qe}|q)
\end{aligned}$$

This allows for efficient training of weight parameter vector $\boldsymbol{\theta}$ to optimize ranking performance. Rank-lips uses mini-batched coordinate ascent to optimize for mean-average precision.

4 Example: Support-Passage Ranking

Despite its name, ENT-rank-lips can also improve other tasks, such as support passage retrieval: In response to a given query q and entity e, the task is to rank text passages $t_1, t_2, t_3, \ldots$ by how well they explain the relevance of the entity for the query.

The task is addressed by combining features modelling the relevance through

1. the relevance of the passage for the query $\vec{\mathrm{f}}_T(t_i|q)$
2. the (query-independent) salience of the entity in the passage $\vec{\mathrm{f}}_{ET'}(e_i,t_i)$
3. the query-specific relevance of an entity in the passage $\vec{\mathrm{f}}_{ET}(e_i,t_i|q)$

Similar to ENT-rank, associations are derived per context t, based on all entities e_j that are mentioned therein: $A_{qt} = (\{t\}, \{e_j|\forall e_j \in t\})$. The main difference is that here we are ranking passages, not entities.

5 Example: Extending ENT Rank with Relations

The extensibility of the ENT-rank-lips paradigm is one of its main advantages. For example ENT Rank (Section 3), can be extended to incorporate information about relations, $r = (e_1,p,e_2,t)$ between two entities e_1 and e_2 of relation predicate p that are extracted from the contextual passage t. All previously defined features and associations are extended to incorporate information from relations as follows.

Additional associations are derived from relation triples: $A = (\{e1, e2\}, \{t\}, \{p\})$. All previously defined associations are included also, since they are not associated with relation predicates, the position of $\{p\}$ is set with an empty set $\emptyset$.

Features of relation triples can include (1) similarity between query and predicate type, (2) BM25 score of the text span covering the relation mentions, as well as (3) query independent features such as the predicate type. The relative importance of these features is learned by ENT-rank-lips. An additional set of neighbor features $\vec{\mathrm{f}}_{EE'}(e, e_j|q)$ can be introduced that only apply to neighbors e, e_j that are in a relation, as opposed to any co-mentioned entity pair.

6 Example: Entity Aspect Linking

The for the task of entity aspect linking, an entity link e in a given text passage c is to be refined to link to one of the entity e's k_e aspects $x_{e1}, x_{e2}, \ldots x_{ek_e}$. The task can be interpreted as a retrieval task, where the context c takes the role of a query.

6.1 Simple Aspect Linking

A simplified version of rank-lips was used to reproduce the entity aspect linking method by Nanni et al. It entailed a list of features $\vec{\mathrm{f}}_X(x_{ei}|c)$ that model the similarity of the context c with the title and content of each aspect x_{ei}, using similarities based on words, entities, and embeddings. In this simplified setting, the associations are merly a list of aspects $A_c = (\{x_{ei}\})$.

Entity salience and entity relatedness can be used to emphasize different entities in the entity-based similarity measure that leads to the feature vector $\vec{\mathrm{f}}_X(x_{ei}|c)$.

6.2 Learn Entity Importance

However, we can also exploit salience information to learn to predict entities that are contained in relevant aspects. For this, we temporarily switch to a differnt prediction problem of ranking entities e by how likely they are contained in the correct aspect. We produce a ground truth from entities that are mentioned in the correct aspect (all other entities are considered non-relevant).

For any entity mentioned in one of the candidate aspects, we produce features for $\vec{f}_M(e|c)$ target mentions[2] based on their salience in the context c. We include entity relatedness through pair-wise entity features of the target mention e and any entity e_j mentioned in the context $\vec{f}_{ME}(e, e_j|c)$.

It is sufficient to use associations of lists of target entities $A_c = (\{e\})$ since during prediction and training, all features that include the target mention e will be aggregated, $\sum_{a \in A_{qe}} \vec{f}(a|q)$, since A_{qe} will include all their argument tuples.

Additionally, the local context of target mentions e in context and aspect can be modeled as a feature $\vec{f}_{ME}(e, a_i|c)$.

Once this entity selection task has been trained, the models is used to make predictions on which entities e are likely mentioned on target aspects. The prediction can be used to produce better features for entity aspect linking (see Section 6.1).[3]

6.3 Joint Aspect Linking

A similar two-step approach is used to exploit similarities between linked aspects of entities mentioned in the same context. The ultimate prediction problem is the same as in Section 6.1, link a mentioned target entity $\tilde{e}$ to its correct aspect x_e. However, we first focus on other entities e_i that are co-mentioned in the same context c, and link them to their respective best aspects x_{e_i}.

This is accomplished in the first step by training a model as in Section 6.1 or 6.2, then using it to predicting aspects x_{e_i} for all contextual entities e_i. A technical issue is to keep separate aspects of different entities separate, since c is not sufficient. We recommend to predict pairs of entities and aspects, $\vec{f}_{EX}(e_i, x_{e_i}|c)$, then during prediction to keep the highest ranked aspect per entity.

In the second step, we augment the set of "direct" features $\vec{f}_{EX}(e_i, x_{ei}|c)$. Knowledge from predicted contextual aspects gives rise to query-independent aspect-to-aspect features $\vec{f}_{XX}(x_{ei}, x_e)$ such as the similarity of aspect names and content.

It is possible to use associations to inform ENT-rank-lips about which contextual entity e_i should be considered when predicting the aspect x_e for the target entity e, by using associations in the form $A = (\{x_e, \dots\}, \{e_i|\forall i\})$.[4]

7 Usage and Feature Format

ENT-rank-lips uses a trec_eval-inspired jsonl.gz format for features (run), associations (run) and ground truth for relevance (qrel). Figure 2 displays examples of the respective formats. The "document" entry can define any fields of your choice, but it has to be flat JSON object with no further nesting, where values are either strings or lists of strings (bools and numbers will be automatically converted to strings).

You need to create a dedicated directory in which the jsonl.gz files for your features will be placed. Keep in mind that the filename will become the name of the feature. If you train a model with features in directory TrainFeatures, then you need a separate directory TestFeatures for test features—these have to use exactly the same filenames as during training!

If you can't remember the filenames you trained a model with, just open the model file with your json viewer.

You train a model with a command line call like this:

```
ent-rank-lips train -q qrel.jsonl.gz \
 -a "assocs.jsonl"  -P aspect \
 -d trainFeatures/ --jsonl.gz --z-score \
 -O out/ -o first_attempt  -e "first-pass" \
 --default-any-feature-value 0.0 --feature-variant FeatScore  \
 --convergence-threshold 0.001 --mini-batch-size 1000  \
 --convergence-drop-initial-iterations 2 \
 --train-cv --threads 20 \
 +RTS -A64M -RTS \
 > out/run.log 2>&1  &
```

[2]Target mentiones are entities mentioned on the correct aspect — not the target entity to which the aspects belong.

[3]While ENT-rank-lips currently does not support joint learning of this two-step approach, it is possible to implement it in the optimization procedure.

[4]While the current ENT-rank-lips implementation does not weight associations, it is possible to endow associations with relative confidence weights, such as the confidence of the co-aspect prediction.

Association (score and rank information is ignored, method is optional)

```
{"query":"4e","document":{"target_entity":"enwiki:Gautama%20Buddha"
    ,"target_aspect":"enwiki:Gautama%20Buddha/Biography"},"rank":1, "score":1
    ,"method":"assocs"}
{"query":"4e","document":{"target_entity":"enwiki:Gautama%20Buddha"
   ,"target_aspect":"enwiki:Gautama%20Buddha/Depiction%20in%20arts%20and%20media"}
   ,"rank":1,"score":1,"method":"assocs"}
```

Feature file (each feature in its own file, the filename will become the feature name).

```
{"query":"ff","document":{"target_aspect":"enwiki:Cycloid/Arc%20length"}
    ,"rank":1",score":4.982495080405202e-2,"method":"run"}
{"query":"ff","document":{"target_aspect":"enwiki:Cycloid/Cycloidal%20pendulum"}
   ,"rank":2,"score":1.5072488571476724e-2,"method":"run"}
{"query":"ff","document":{"target_aspect":"enwiki:Cycloid/History"}
   ,"rank":3,"score":0,"method":"run"}
```

Qrel for ground truth of relevance:

```
{"query":"7f", "document":{"target_aspect":"enwiki:Dual%20Shock%20Wave/2014"}
    ,"relevance":1}
{"query":"7f", "document":{"target_aspect":"enwiki:Dual%20Shock%20Wave/2013"}
   ,"relevance":0}
```

Figure 2: Example of ENT-rank-lips format for associations, features, and qrels.

- The –jsonl.gz flag tells rank-lips that the train feature files are in gz-compressed jsonl format. Without gz-compression use –jsonl. The flags "--default-any-feature-value" specifies that any feature that is missing should take this value (there are more flexible options to specify default features also).
- When "–z-score" is enabled, automatically z-score normalization is performed during training and the model file will be aware of it. So you don't have to worry about z-score normalization (Please don't do it yourself, let ent-rank-lips do it for you).
- The parameter "-P aspect" indicates that we seek to predict a ranking of the type in the "aspect" field of associations. The association file defines the candidate set, so only the target entries listed in the association files will be ranked. You can also use one of the feature files as association (because the formats are identical). Just keep in mind that it needs to specify all target entries you want to be ranked.
- Here "–train-cv" is enabled, meaning that it will both train one large model in addition to a five-fold cross validation. Resulting model files and predicted rankings will be placed in the output directory (specified with '-O'). Make sure it exists before you run.
- The resulting run-file format is not in out JSONL format, but a traditional trec_eval style run file.[5]
- Various parameters are controlling the optimization such as convergence threshold and the mini-batch size (choose as high as possible on the hardware)
- "–threads" indicates how many threads you will use in parallel. If you run on jelly, I recommend to include "+RTS -A64M -RTS" which tells the haskell runtime system to use a larger initial heap size.
- "... > out/run.log 2>&1" tells bash to write both stdout and stderr to the same log file.

For a detailed description of the parameters, call "ent-rank-lips train -h"

```
Usage: ent-rank-lips train (-d|--feature-runs-directory DIR)
                           [-f|--feature FEATURE] [--feature-variant FVAR]
```

[5]This is likely to change in the future.

```
(--jsonl | --jsonl.gz | --trec-eval)
(-a|--assocs JSONL) (-O|--output-directory OUTDIR)
[-o|--output-prefix FILENAME] (-q|--qrels QRELS)
(-e|--experiment FRIENDLY_NAME)
[[--mini-batch-steps STEPS] [--mini-batch-size SIZE]
  [--mini-batch-eval EVAL]] [--train-cv] [--z-score]
[--save-heldout-queries-in-model]
[--convergence-threshold FACTOR]
[--convergence-max-iter ITER]
[--convergence-drop-initial-iterations ITER]
[--convergence-eval-cutoff K] [-r|--restarts N]
[--folds K]
[--default-any-feature-value VALUE |
  --default-feature-variant-value FVariant=VALUE |
  --default-feature-value FNAME-FVariant=VALUE]
[-j|--threads J] (-P|--predict FIELD)
```

If you have a trained model that would like to predict new rankings, use "ent-rank-lips predict", which uses similar command line arguments.

Ent-rank-lips comes with several helper tools (each offer extensive help with '-h'):

- ent-rank-lips conv-qrels: converts a trec_eval compatible qrel file into relevance data of our jsonl.gz format.
- ent-rank-lips conv-runs: converts a trec_eval compatible run file into a features in our jsonl.gz format (this is handy whenever you want to use retrieval features).
- ent-rank-lips export-runs: converts a run in our jsonl.gz format into a trec_eval compatible run file.
- ent-rank-lips qrels-assocs: converts a trec_eval qrel file into an association file (this needs to be an appropiate qrel file – its not going to work if the qrel file only contains relevant entries....) — use conv-runs to create an association file from a run file (its the same format)
- ent-rank-lips rank-aggregation: unsupervised prediction method (does not use training!) that interprets all features as ranking functions sums of reciprocal ranks: $\text{score}_q(e) = \sum_{\text{f}_i} \frac{1}{\text{rank}(e|f_i)}$ – you might be laughing, but its really strong!
- there are more options which you see with "ent-rank-lips -h", then inspect with "ent-rank-lips $cmd -h".

You can use "ent-rank-lips filter-top-k" with "ent-rank-lips filter-assocs" to restrict associations to a set of first pass predictions (e.g. for joint aspect linking).