Retrieval Task

Task: Satisfy the Joint Personalize Search and Recommendation (JPSR) Problem proposed. Inputs are a set of users, and their interactions (searches and recommendation interactions). Output is a ranked list of recommendation considering the input.

User	Recommendations
RockLover2000	Top 5 hits: 'The Strokes', 'Snow Patrol', 'Copeland', 'The Kooks', 'Jimmy Eat World'

The difference between a search and recommendation comes from the presence of a query. A search includes query, while a recommendation does not include one. A model that tackles this problem is to construct a personalized list of items sorted by relevance based on a user, their interactions, and a possibly empty query. The idea of joining search and recommendation models makes sense because they have very similar inputs and goals. Joining them allows us to train these models simultaneously, and with more relevant data, hopefully increasing performance.

Data Set: The data set used is Lastfm[1]. Here, listening to a certain artist is considered a recommendation interaction, and tagging an artist is considered a search interaction. This data set is the smallest, with less than 200,000 training interactions. Contextually, it is limited due to it's simple queries because of simple tags like "classical", or "alternative".

Two other datasets were used in the paper, but for convinience of the experiment we use the lastfm dataset. Each data set has a unique way of defining users, items, queries, and different interactions.

The Point-of-Interest search engine (POI) data set is collected from the South-Korean search engine Naver[2]. Six months of total data is collected, from 01/09/2020 to 07/09/2020. A search interaction happens when a user clicked an item(page) after issuing a query of keywords. A recommendation interaction happens when a user clicks on an item without specifying a query.

MovieLens 25m[3]. This dataset contains user ratings, and taggings of movies. Any user rating is considered a recommendation interaction, with a cutoff of 4+ stars out of 5, being considered a positive recommendation. All added tags are used as search interactions. This is the largest data set in terms of interactions, with over 2 million training interactions in total.

Evaluation Metrics Evaluation is split into two sections, Search and Recommendation. Two metrics are used for evaluation; H@20 and NDCG@20. HitRate at 20 (H@20), measures if a relevant item appears in the top 20 recommendations and search results. Normalized Discounted Cumulative Gain at 20 (NDCG@20), measures ranking quality for the model's returned search and recommendation results by normalizing usefulness (gain) based on the positions of relevant results (a highly relevant document returned lower on the list of results will result in a lower score than if it were higher). Resulting scores are then compared against other retrieval algorithms including; BM25, DREM, JSR, LGCN, and MF.

Model	POI				MovieLens				Lastfm			
	Rec.		Search		Rec.		Search		Rec.		Search	
	H@20	N@20	H@20	N@20	H@20	N@20	H@20	N@20	H@20	N@20	H@20	N@20
MF [10]	8.72	3.30	7.20	2.87	4.88	1.81	2.75	1.02	23.42	10.36	17.03	7.91
LGCN [9]	9.24	3.52	8.21	3.34	5.74	2.17	3.30	1.23	26.32	12.09	20.34	9.94
FM [13]	9.46	3.59	48.23	26.43	5.08	1.89	12.91	5.57	25.07	11.10	27.35	12.80
DeepFM [8]	9.35	3.54	48.71	26.70	5.33	1.99	13.39	5.88	24.56	10.95	26.82	12.56
JSR [28]	9.05	3.42	59.67	36.76	4.95	1.87	12.35	4.82	24.33	10.68	5.01	1.92
BM25 [17]	–	–	60.23	33.89	–	–	11.06	3.99	–	–	3.20	1.16
DREM [2]	9.28	3.53	24.37	11.37	4.09	1.47	6.84	2.63	23.45	10.32	23.58	10.95
HyperSaR	**10.04**†	**3.84**†	**63.17**†	**39.26**†	**5.93**†	**2.27**†	**21.87**†	**11.04**†	**27.11**†	**12.38**†	**30.05**†	**14.76**†
p-value	6e-7	1e-6	2e-14	1e-12	6e-6	5e-7	1e-14	4e-15	4e-5	1e-4	4e-7	4e-8
Effect size	6.30	5.68	56.89	33.12	4.69	6.48	60.83	67.29	3.66	3.04	6.71	8.86

Figure 1: Example of evaluation table[**1**]

[1]https://grouplens.org/datasets/hetrec-2011/
[2]https://www.naver.com/
[3]https://grouplens.org/datasets/movielens/25m/

Retrieval Models

Model Used: The model used is a Hypergraph convolutional network approach for Search and Recommendation (HyperSaR)[**1**]. HyperSaR uses undirected hypergraphs to represent connections between users,items, and queries.This hypergraph is then used in the layered convolutional network to train the model based on two calculated loss values.

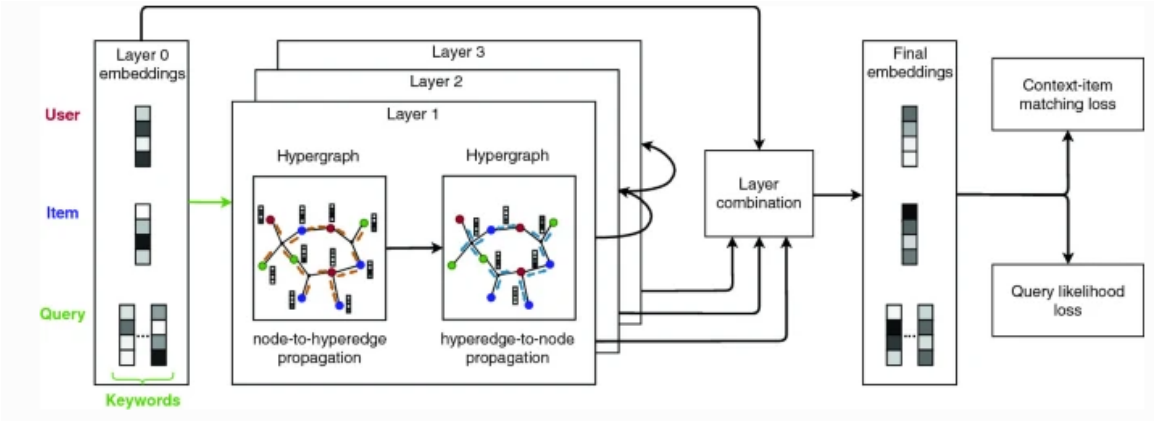

Figure 2: HyperSaR Model Pipline [**1**]
Hypergraph convolutional network. Used for representing connections between users items and queries. Layer 1 of the model represents direct relationship interactions (listening to a song). Further layers represent increasingly higher level, abstract relations between nodes in the hypergraph.

Hypergraph: The hypergraph used has two distinct types of hyperedges. The first hyperedge consists of a user, an item, and at least one query. This hyperedge represents a search instance. The second hyperedge consists of only a user and an item, without a query. This represents a recommendation instance. Once the hypergraph is constructed, it can be turned into a binary incidence matrix, H, that has dimensions $|V| \times |E|$, where V is the set of all nodes/vertices, and E is the set of all hyperedges. For the element $H_{n,m}$ in the matrix, each value is either 1 or 0. A value of 1 indicates that the n-th node is connected to the m-th hyperedge, while a 0 represents no connection.

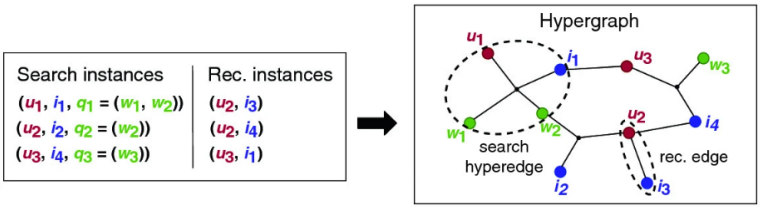

Figure 3: Example of hypergraph built from search and recommendation instances. Nodes u*, i* and w* denote users, items and query keywords. [**1**] Nodes involving user, items and query represent a search interaction and are 3 or more dimensions (hyperedge). Node connections with a user and item are recommendation instances (edge). Edges within the same neighborhood are more closely related, and positions change as the model is trained.

Convolution and embeddings: Once the incidence matrix is contructed, we can perform matrix operations to generate embeddings and propogate them into however many layers are required. After all layers have been constructed. Each layer embedding is decomposed back into three separate parts, (E_u, E_I, E_W). These three parts represent the embedding for a user, item, and query respectively. Once decomposed, all layers are aggregated by summing their embedding results. These embeddings are then used to calculate two different loss functions

Loss Functions: Two separate loss functions are used, Context-Item matching loss and query-likelihood loss. Context-Item matching sums the dot product of all embeddings together, and then uses pairwise Bayesian Personalized Ranking (BPR) [**b2**] $L_{\text{CIM}} = -\frac{1}{|\mathcal{X}|} \sum_{x \in \mathcal{X}} \sum_{i_{\text{n}} \in \mathcal{I} \setminus \{i_x\}} \log \sigma(\hat{y}_{u_x, i_x, q_x} - \hat{y}_{u_x, i_{\text{n}}, q_x}) + \lambda \left\| E_{\mathcal{V}}^{(0)} \right\|_F^2$.

This method weighs the difference between items the user recommends, and items that it doesn't. The query likelihood loss uses embeddings to find the probability of a query given the user and the probability of a query given an item $L_{\text{QL}} = -\frac{1}{|\mathcal{X}_S|}\sum_{x\in\mathcal{X}_S}\frac{1}{|q_x|}\sum_{w\in q_x}\log p(w\,|\,u_x) - \frac{1}{|\mathcal{X}_S|}\sum_{x\in\mathcal{X}_S}\frac{1}{|q_x|}\sum_{w\in q_x}\log p(w\,|\,i_x)$. The probabilities of all interactions are then added to give us the query-likelihood loss value. These two loss values are then linearly combined using a balancing hyperparameter η as $L = LCIM + \eta LQL$.

Experimental modifications: FastText embeddings (Baseline) and Pre-trained BERT Embeddings, Specifically; BERT Base and BERT Large. [**3**] These are used in layer 0 of the pipeline in figure 2.

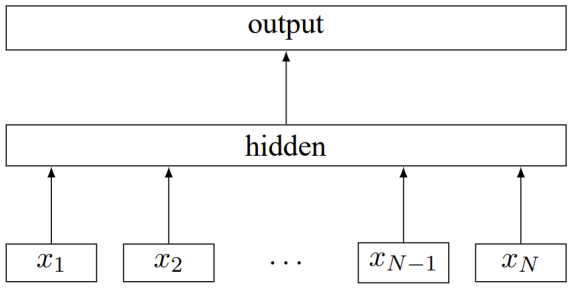

Figure 4: FastText architecture with N n-gram features [**4**]

FastText embeddings consists of a number of embeddings for each ngram, seperating words into character strings. This allows it to accurately capture misspelt words or words with similar meaning. FastText takes into account a relatively small amount of context around it. Because of this FastText may not be as effective as other models at tasks that require a deeper understanding of semantics and the relationships between words.

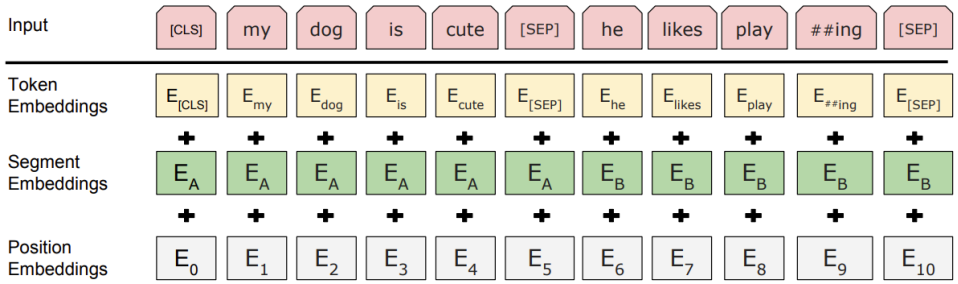

Figure 5: Representation of BERT Token, Segment, and Position embeddings from input[**3**]

BERT embeddings are more complex and much more contextual. As shown in Figure 5, each embedding is calculated in reference to a large amount of words surrounding it. Because of this BERT often is exepcted to outperform FastText, however it is much more computationally expensive and takes much longer to train. BERT large differs from BERT base in terms of scale, having many more transformers and parameters and can generally undertake more complex retrieval tasks.

Experiment

Research Question: What are the effects of changing baseline pre-trained word embeddings in the HyperSaR model in the proposed Joint Personalized Search and Recommendation task?

Independent Variable	Control Variables	Dependent Variables
Pre-trained word embeddings*	Dataset (LastFM)	Recall@1,10,20
	Weight Decay: 0	NDCG@20
	Number of layers: 2	generated output/artists
	Epochs: 100	
	Learning Rate: 0.001	
	Batch Size: 1024	
	Embedding Dimensions: 64	
	Number of Negative samples: 0	
	Number of Keywords: 2000	

*Pre-trained word embeddings include: FastText (Baseline), Bert-Base, and Bert-Large pretrained embeddings, from the hugging face transformer library[4] [5]

Hypothesis: Each of the different pre-trained BERT embeddings will affect the models's ability to meet a user's needs and preferences based on their queries and recommendations (verified with recall and NDCG scores); The more complex the pre-trained embedding, the better user needs will be met. FastText is expected to do worse because it is the simplest model. BERT-Large will likely do this the best since it is the most complex of the pre-trained embeddings tested.

Due to the fact that BERT embeddings store more context, like segments and positions, it should give the HyperSaR model a more ideal set of initial embeddings. The HyperSaR model should be able to keep the extra context and retrieve better results.

Methodology:

Implementation: The HyperSaR model is used for all iterations of experimentation. The model is consistently run with the hyperparameters as seen in the Control Variables column in the experiment chart above.

Pre-processing: Using the LastFM dataset, features extracted are users, items, query keywords, and user-item interactions. These features are gathered through tokenization before being converted to embeddings. To generate embeddings, tokens were aiejrojita into the pretrained word-embedding models, which were then stored to be used as inputs to layer0 of the hypergraph neural network.

Model Training: Hypergraphs contstructed based on similar embeddings and populated into neighborhoods of edges and hyperedges. The weights are then fed into deeper layers for higher-level connections to be found, and then combined and used to calculate the two loss functions. After they loss is calculated, the hypergraphs are updated and the process repeats for 100 epochs.

Scoring: In order to compare the performance for each independent variable, we use Recall(@1, 10, 20) and NDCG@20. These are metrics are calculated by comparing generated results against a ground truth item for each item interaction [**1**].

[4]https://huggingface.co/bert-large-uncased
[5]https://huggingface.co/bert-base-uncased

Code Changes: Although the HyperSaR code base has some existing architecture that allows for changing embeddings, this only allows for word vector embeddings, like FastText.

- Huggingface Transformer library is imported to load pretrained Bert-Base-Uncased, and Bert-Large-Uncased
- To use BERT Embeddings, The data processing code section had to be changed. The embedding function is given a vectorizer, that contains the TF-IDF features of all tokens in the dataset. Each token in this vectorizer is fed in as an input into the desired BERT model, and the outputted embeddings are then stored.
- Vectorizer dimension is changed to support Bert Base (768) and BERT Large (1024)
- Creation of a bash make file for ease of experiment executions.
- Option parsing file changed to allow for specification of embedding to be used.
- Evaluation file modified to print out an example search and retrieval case.

Results and Discussion

After running the HyperSaR model using the three different embeddings, all metrics were recorded and reported in the tables below. Table 3 records a combination of both search and recommendation instances.

Embedding	R@1	R@10	R@20	NDCG@20
FastText	0.05425	0.21901	**0.29928**	0.14620
BERT Base	0.05455	0.21835	0.29843	0.14596
BERT Large	**0.05548**	**0.21988**	0.29685	**0.14665**

Table 1: search metrics. Best Model in bold

Embedding	R@1	R@10	R@20	NDCG@20
FastText	0.03880	0.19036	**0.27065**	**0.12423**
BERT Base	0.03819	**0.19217**	0.27032	0.12374
BERT Large	**0.03946**	0.18822	0.26757	0.12293

Table 2: Recommendation metrics. Best Model in bold

Embedding	R@1	R@10	R@20	NDCG@20
FastText	0.04912	**0.20949**	**0.28976**	**0.1389**
BERT Base	0.04913	0.20919	0.28934	0.13864
BERT Large	**0.05016**	0.20936	0.28712	0.13877

Table 3: both search and recommendation metrics. Best Model in bold

Observations: The recorded metrics for all embeddings are extremely similar, with the largest difference under 1%. This implies that embeddings didn't have a large impact on the effectiveness of the HyperSaR model. These results contradict our original hypothesis that the BERT Embeddings would increase the effectiveness. There are a few possible reasons why we received these results.

1. The BERT Embeddings do not add any value compared to FastText embeddings.

   This is the least likely reason because BERT embeddings are known to add more context than FastText embeddings. This includes more context based on passages as a whole, instead of each word.

2. The HyperSaR model does not utilize or require complex embeddings for better performance.

   Because the HyperSaR model uses both search and recommendation instances, and heavily utilizes connections between items inside of it's hypergraph, the initialized embeddings may not influence the model as much compared to others. Recommendation instances don't even use a query that the embeddings are used on, and the hypergraph generation doesn't use embeddings. We suspect that, although an embedding is necessary, this model does not fully utilize more complex embeddings like BERT.

3. The Lastfm dataset does not have many large phrases to properly utilize the larger context of BERT embeddings.

   This is likely the the largest reason why results were so similar. A search item is defined as a user tagging a song. These tags are usually one word terms, like "Pop", or "Rock". The larger context that BERT embeddings supply are lost on these simple tags.

Generated Output: To get a better idea of how the different models generate different output, a specific user and set of keywords were given to the model, and generated outputs were recorded. This output came from user 213, and the keywords given were 'burnap', 'carrie underwood', 'pure aural sex', 'fucking awensome', 'darkwave', and 'silent intensity'.

Embedding	Generated output
FastText	'Timbaland', 'Blue', 'Girls Aloud', 'S Club 7', 'Sean Paul', 'Joe McElderry', 'Nelly Furtado', 'Will Smith', 'Pharrell', 'Jesse McCartney'
BERT Base	'Christina Aguilera', 'Kelly Clarkson', 'Jonas Brothers', 'Lindsay Lohan', 'Metro Station', 'Girls Aloud', 'Ashley Tisdale', 'Leighton Meester', 'High School Musical', 'Justin Bieber'
BERT Large	'Ashley Tisdale', 'Girls Aloud', 'Joe McElderry', 'High School Musical', 'Rihanna', 'Paris Hilton', 'Rachel Stevens', 'Kelly Clarkson', 'Aly & AJ', 'The Saturdays'

Table 4: ordered list of artists by relevance for user 213 given the same keywords

This table reveals that different embeddings do change the behavior of the model. This example is a fairly extreme one, with many specific keywords. Because of this, the models have more chances to distinguish themselves from each other. There are some common artists between models, like "High School Musical", and "Girls Aloud", but each model ranks them differently, and many ranked artists are unique. This tells us that embeddings do impact generated output.

Future Work: Future work entails establishing a proper dataset with context that would likely establish a clear answer for our hypothesis. In this case it failed to support or reject the claim clearly. Out of the three datasets in the paper, the Point of Interest Search engine Naver would be the most ideal, but unfortunately, MovieLens and Lastfm were the only two that were readily available. Both of which use tagging as a search interaction. Given this, we would need to find another compatible dataset, or process our own. This new dataset would include more complex searches and items that are more like the data that BERT embeddings are trained on. Some possible candidates are social media website like twitter, reddit, etc, or web fiction websites like Royal Road. This model could even be extracted to video hosting sites like Youtube, where search and recommendation interactions are more clearly defined.

References

[1] Thibaut Thonet, Jean-Michel Renders, Mario Choi, and Jinho Kim. 2022. Joint Personalized Search and Recommendation with Hypergraph Convolutional Networks. In Advances in Information Retrieval: 44th European Conference on IR Research, ECIR 2022, Stavanger, Norway, April 10–14, 2022, Proceedings, Part I. Springer-Verlag, Berlin, Heidelberg, 443–456. https://doi.org/10.1007/978-3-030-99736-6_30

[2] Rendle, S., Freudenthaler, C., Gantner, Z., Schmidt-Thieme, L.: BPR: Bayesian personalized ranking from implicit feedback. In: UAI, pp. 452–461 (2009)

[3] J. Devlin et al, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," ArXiv.Org, 2019. Available: https://ezproxy.rit.edu/login?url=https://www.proquest.com/working-papers/bert-pre-training-deep-bidirectional-transformers/docview/2118630252/se-2.

[4] Armand J., Edouard G., Piotr B., Tomas M.: "Bag of Tricks for Efficient Text Classification" ArXiv.org, 2016. Available: https://arxiv.org/abs/1607.01759.