Metadata Embeddings for User and Item Cold-start Recommendations

Maciej Kula
Lyst
maciej.kula@lyst.com

ABSTRACT

I present a hybrid matrix factorisation model representing users and items as linear combinations of their content features' latent factors. The model outperforms both collaborative and content-based models in cold-start or sparse interaction data scenarios (using both user and item metadata), and performs at least as well as a pure collaborative matrix factorisation model where interaction data is abundant. Additionally, feature embeddings produced by the model encode semantic information in a way reminiscent of word embedding approaches, making them useful for a range of related tasks such as tag recommendations.

Categories and Subject Descriptors

H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval—*Information Filtering*

Keywords

Recommender Systems, Cold-start, Matrix Factorization

1. INTRODUCTION

Building recommender systems that perform well in cold-start scenarios (where little data is available on new users and items) remains a challenge. The standard matrix factorisation (MF) model performs poorly in that setting: it is difficult to effectively estimate user and item latent factors when collaborative interaction data is sparse.

Content-based (CB) methods address this by representing items through their metadata [10]. As these are known in advance, recommendations can be computed even for new items for which no collaborative data has been gathered. Unfortunately, no transfer learning occurs in CB models: models for each user are estimated in isolation and do not benefit from data on other users. Consequently, CB models perform worse than MF models where collaborative information is available and require a large amount of data on each user, rendering them unsuitable for user cold-start [1].

CBRecSys 2015, September 20, 2015, Vienna, Austria.

At Lyst, solving these problems is crucial. We are a fashion company aiming to provide our users with a convenient and engaging way to browse—and shop—for fashion online. To that end we maintain a very large product catalogue: at the time of writing, we aggregate over 8 million fashion items from across the web, adding tens of thousands of new products every day.

Three factors conspire to make recommendations challenging for us. Firstly, our system contains a very large number of items. This makes our data very sparse. Secondly, we deal in fashion: often, the most relevant items are those from newly released collections, allowing us only a short window to gather data and provide effective recommendations. Finally, a large proportion of our users are first-time visitors: we would like to present them with compelling recommendations even with little data. This combination of user and item cold-start makes both pure collaborative and content-based methods unsuitable for us.

To solve this problem, I use a hybrid content-collaborative model, called LightFM due to its resemblance to factorisation machines (see Section 3). In LightFM, like in a collaborative filtering model, users and items are represented as latent vectors (embeddings). However, just as in a CB model, these are entirely defined by functions (in this case, linear combinations) of embeddings of the content features that describe each product or user. For example, if the movie 'Wizard of Oz' is described by the following features: *'musical fantasy'*, *'Judy Garland'*, and *'Wizard of Oz'*, then its latent representation will be given by the sum of these features' latent representations.

In doing so, LightFM unites the advantages of content-based and collaborative recommenders. In this paper, I formalise the model and present empirical results on two datasets, showing that:

1. In both cold-start and low density scenarios, LightFM performs at least as well as pure content-based models, substantially outperforming them when either *(1)* collaborative information is available in the training set or *(2)* user features are included in the model.

2. When collaborative data is abundant (warm-start, dense user-item matrix), LightFM performs at least as well as the MF model.

3. Embeddings produced by LightFM encode important semantic information about features, and can be used for related recommendation tasks such as tag recommendations.

This has several benefits for real-world recommender systems. Because LightFM works well on both dense and sparse data, it obviates the need for building and maintaining multiple specialised machine learning models for each setting. Additionally, as it can use both user and item metadata, it has the quality of being applicable in both item and user cold-start scenarios.

To allow others to reproduce the results in this paper, I have released a Python implementation of LightFM[1], and made the source code for this paper and all the experiments available on Github[2].

2. LIGHTFM

2.1 Motivation

The structure of the LightFM model is motivated by two considerations.

1. The model must be able to learn user and item representations from interaction data: if items described as 'ball gown and 'pencil skirt' are consistently all liked by users, the model must learn that ball gowns are similar to pencil skirts.

2. The model must be able to compute recommendations for new items and users.

I fulfil the first requirement by using the latent representation approach. If ball gowns and pencil skirts are both liked by the same users, their embeddings will be close together; if ball gowns and biker jackets are never liked by the same users, their embeddings will be far apart.

Such representations allow transfer learning to occur. If the representations for ball gowns and pencil skirts are similar, we can confidently recommend ball gowns to a new user who has so far only interacted with pencil skirts.

This is over and above what pure CB models using dimensionality reduction techniques (such as latent semantic indexing, LSI) can achieve, as these only encode information given by feature co-occurrence rather than user actions. For example, suppose that all users who look at items described as aviators also look at items described as wayfarers, but the two features never describe the same item. In this case, the LSI vector for wayfarers will not be similar to the one for aviators even though collaborative information suggests it should be.

I fulfil the second requirement by representing items and users as linear combinations of their content features. Because content features are known the moment a user or item enters the system, this allows recommendations to be made straight away. The resulting structure is also easy to understand. The representation for denim jacket is simply a sum of the representation of denim and the representation of jacket; the representation for a female user from the US is a sum of the representations of US and female users.

2.2 The Model

To describe the model formally, let U be the set of users, I be the set of items, F^U be the set of user features, and F^I the set of item features. Each user interacts with a number of items, either in a favourable way (a positive interaction), or in an unfavourable way (a negative interaction). The set of all user-item interaction pairs $(u, i) \in U \times I$ is the union of both positive S^+ and negative interactions S^-.

Users and items are fully described by their features. Each user u is described by a set of features $f_u \subset F^U$. The same holds for each item i whose features are given by $f_i \subset F^I$. The features are known in advance and represent user and item metadata.

The model is parameterised in terms of d-dimensional user and item feature embeddings $\boldsymbol{e}_f^U$ and $\boldsymbol{e}_f^I$ for each feature f. Each feature is also described by a scalar bias term (b_f^U for user and b_f^I for item features).

The latent representation of user u is given by the sum of its features' latent vectors:

$$\boldsymbol{q}_u = \sum_{j \in f_u} \boldsymbol{e}_j^U$$

The same holds for item i:

$$\boldsymbol{p}_i = \sum_{j \in f_i} \boldsymbol{e}_j^I$$

The bias term for user u is given by the sum of the features' biases:

$$b_u = \sum_{j \in f_u} b_j^U$$

The same holds for item i:

$$b_i = \sum_{j \in f_i} b_j^I$$

The model's prediction for user u and item i is then given by the dot product of user and item representations, adjusted by user and item feature biases:

$$\widehat{r}_{ui} = f\left(\boldsymbol{q}_u \cdot \boldsymbol{p}_i + b_u + b_i\right) \tag{1}$$

There is a number of functions suitable for $f(\cdot)$. An identity function would work well for predicting ratings; in this paper, I am interested in predicting binary data, and so after Rendle *et al.* [16] I choose the sigmoid function

$$f(x) = \frac{1}{1 + \exp(-x)}.$$

The optimisation objective for the model consists in maximising the likelihood of the data conditional on the parameters. The likelihood is given by

$$L\left(\boldsymbol{e}^U, \boldsymbol{e}^I, \boldsymbol{b}^U, \boldsymbol{b}^I\right) = \prod_{(u,i) \in S^+} \widehat{r}_{ui} \times \prod_{(u,i) \in S^-} (1 - \widehat{r}_{ui}) \tag{2}$$

I train the model using asynchronous stochastic gradient descent [14]. I use four training threads for experiments performed in this paper. The per-parameter learning rate schedule is given by Adagrad [6].

2.3 Relationship to Other Models

The relationship between LightFM and the collaborative MF model is governed by the structure of the user and item feature sets. If the feature sets consist solely of indicator variables for each user and item, LightFM reduces to the standard MF model. If the feature sets also contain metadata features shared by more than one item or user, LightFM extends the MF model by letting the feature latent factors explain part of the structure of user interactions.

This is important on three counts.

[1] `https://github.com/lyst/lightfm/`

[2] `https://github.com/lyst/lightfm-paper/`

1. In most applications there will be fewer metadata features than there are users or items, either because an ontology with a fixed type/category structure is used, or because a fixed-size dictionary of most common terms is maintained when using raw textual features. This means that fewer parameters need to be estimated from limited training data, reducing the risk of overfitting and improving generalisation performance.

2. Latent vectors for indicator variables cannot be estimated for new, cold-start users or items. Representing these as combinations of metadata features that *can* be estimated from the training set makes it possible to make cold-start predictions.

3. If only indicator features are present, LightFM should perform on par with the standard MF model.

When only metadata features and no indicator variables are present, the model in general does not reduce to a pure content-based system. LightFM estimates feature embeddings by factorising the collaborative interaction matrix; this is unlike content-based systems which (when dimensionality reduction is used) factorise pure content co-occurrence matrices.

One special case where LightFM does reduce to a pure CB model is where each user is described by an indicator variable and has interacted only with one item. In that setting, the user vector is equivalent to a document vector in the LSI formulation, and only features which occur together in product descriptions will have similar embeddings.

The fact that LightFM contains both the pure CB model at the sparse data end of the spectrum and the MF model at the dense end suggests that it should adapt well to datasets of varying sparsity. In fact, empirical results show that it performs at least as well as the appropriate specialised model in each scenario.

3. RELATED WORK

There are a number of related hybrid models attempting to solve the cold-start problem by jointly modelling content and collaborative data.

Soboroff *et al.* [21] represent users as linear combinations of the feature vectors of items they have interacted with. They then perform LSI on the resulting item-feature matrix to obtain latent user profiles. Representations of new items are obtained by projecting them onto the latent feature space. The advantage of the model, relative to pure CB approaches, consists in using collaborative information encoded in the user-feature matrix. However, it models user preferences as being defined over individual features themselves instead of over items (sets of features). This is unlike LightFM, where a feature's effect in predicting an interaction is always taken in the context of all other features characterising a given user-item pair.

Saveski *et al.* [18] perform joint factorisation of the user-item and item-feature matrices by using the same item latent feature matrix in both decompositions; the parameters are optimised by minimising a weighted sum of both matrices' reproduction loss functions. A weight hyperparameter governs the relative importance of accuracy in decomposing the collaborative and content matrices. A similar approach is used by McAuley *et al.* [11] for jointly modelling ratings and product reviews. Here, LightFM has the advantage of simplicity as its single optimisation objective is to factorise the user-item matrix.

Shmueli *et al.* [20] represent items as linear combinations of their features' latent factors to recommend news articles; like LightFM, they use a single-objective approach and minimise the user-item matrix reproduction loss. They show their approach to be successful in a modified cold-start setting, where both metadata and data on other users who have commented on a given article is available. However, their approach does not extend to modelling user features and does not provide evidence on model performance in warm-start scenario.

LightFM fits into the hybrid model tradition by jointly factorising the user-item, item-feature, and user-feature matrices. From a theory standpoint, it can be construed as a special case of Factorisation Machines [15].

FMs provide an efficient method of estimating variable interaction terms in linear models under sparsity. Each variable is represented by a k-dimensional latent factor; the interaction between variable i and j is then given by the dot product of their latent factors. This has the advantage of reducing the number of parameters to be estimated.

LightFM further restricts the interaction structure by only estimating the interactions between user and item features. This aids the interpretability of resulting feature embeddings.

4. DATASETS

I evaluate LightFM's performance on two datasets. The datasets span the range of dense interaction data, where MF models can be expected to perform well (MovieLens), and sparse data, where CB models tend to perform better (CrossValidated). Both datasets are freely available.

4.1 MovieLens

The first experiment uses the well-known MovieLens 10M dataset[3], combined with the Tag Genome tag set [22].

The dataset consists of approximately 10 million movie ratings, submitted by $71,567$ users on $10,681$ movies. All movies are described by their genres and a list of tags from the Tag Genome. Each movie-tag pair is accompanied by a relevance score (between 0 and 1), denoting how accurately a given tag describes the movie.

To binarise the problem, I treat all ratings below 4.0 (out of a 1 to 5 scale) as negative; all ratings equal to or above 4.0 are positive. I also filter out all ratings that fall below the 0.8 relevance threshold to retain only highly relevant tags.

The final dataset contains $69,878$ users, $10,681$ items, $9,996,948$ interactions, and 1030 unique tags.

4.2 CrossValidated

The second dataset consists of questions and answers posted on CrossValidated[4], a part of the larger network of StackExchange collaborative Q&A sites that focuses on statistics and machine learning. The dataset[5] consists of 5953 users, $44,200$ questions, and $188,865$ answers and comments. Each question is accompanied by one or more of 1032 unique tags (such as 'regression' or 'hypothesis-testing'). Additionally,

[3] `http://grouplens.org/datasets/movielens/`
[4] `http://stats.stackexchange.com`
[5] `https://archive.org/details/stackexchange`

user metadata is available in the form of 'About Me' sections on users' profiles.

The recommendation goal is to match users with questions they can answer. A user answering a question is taken as an implicit positive signal; all questions that a user has not answered are treated as implicit negative signals. For the training and test sets, I construct 3 negative training pairs for each positive user-question pair by randomly sampling from all questions that a given user has not answered.

To keep the model simple, I focus on a user's willingness to answer a question rather than their ability, and forego modelling user expertise [17].

5. EXPERIMENTAL SETUP

For each dataset, I perform two experiments. The first simulates a warm-start setting: 20% of all interaction pairs are randomly assigned to the test set, but all items and users are represented in the training set. The second is an item cold-start scenario: all interactions pertaining to 20% of items are removed from the training set and added to the test set. This approximates a setting where the recommender is required to make recommendations from a pool of items for which no collaborative information has been gathered, and only content metadata (tags) are available.

I measure model accuracy using the mean receiver operating characteristics area under the curve (ROC AUC) metric. For an individual user, AUC corresponds to the probability that a randomly chosen positive item will be ranked higher than a randomly chosen negative item. A high AUC score is equivalent to low rank-inversion probability, where the recommender mistakenly ranks an unattractive item higher than an attractive item. I compute this metric for all users in the test set and average it for the final score.

I compute the AUC metric by repeatedly randomly splitting the dataset into a 80% training set and a 20% test set. The final score is given averaging across 10 repetitions.

I test the following models:

1. **MF**: a conventional matrix factorisation model with user and item biases and a sigmoid link function [8].

2. **LSI-LR**: a content-based model. To estimate it, I first derive latent topics from the item-feature matrix through latent semantic indexing and represent items as linear combinations of latent topics. I then fit a separate logistic regression (LR) model for each user in the topic mixture space. Unlike the LightFM model, which uses collaborative data to produce its latent representation, LSI-LR is purely based on factorising the content matrix. It should therefore be helpful in highlighting the benefit of using collaborative information for constructing feature embeddings.

3. **LSI-UP**: a hybrid model that represents user profiles (UP) as linear combinations of items' content vectors, then applies LSI to the resulting matrix to obtain latent user and item representations ([21], see Section 3). I estimate this model by first constructing a user-feature matrix: each row represents a user and is given by the sum of content feature vectors representing the items that user positively interacted with. I then apply truncated SVD to the normalised matrix to obtain user and feature latent vectors; item latent vectors are obtained through projecting them onto the latent feature space. The recommendations score for a user-item pair is then the inner product of their latent representations.

4. **LightFM (tags)**: the LightFM model using only tag features.

5. **LightFM (tags + ids)**: the LightFM model using both tag and item indicator features.

6. **LightFM (tags + about)**: the LightFM model using both item and user features. User features are available only for the CrossValidated dataset. I construct them by converting the 'About Me' sections of users' profiles to a bag-of-words representation. I first strip them of all HTML tags and non-alphabetical characters, then convert the resulting string to lowercase and tokenise on spaces.

In both LightFM (tags) and LightFM (tags + ids) users are described only by indicator features.

I train the LightFM models using stochastic gradient descent with an initial learning rate of 0.05. The latent dimensionality of the models is set to 64 for all models and experiments. This setting is intended to reflect the balance between model accuracy and the computational cost of larger vectors in production systems (additional results on model sensitivity to this parameter are presented in Section 6.2). I regularise the model through an early-stopping criterion: the training is stopped when the model's performance on the test set stops improving.

Table 1: Results

	CrossValidated		MovieLens	
	Warm	Cold	Warm	Cold
LSI-LR	0.662	0.660	0.686	0.690
LSI-UP	0.636	0.637	0.687	0.681
MF	0.541	0.508	0.762	0.500
LightFM (tags)	0.675	0.675	0.744	0.707
LightFM (tags + ids)	0.682	0.674	**0.763**	**0.716**
LightFM (tags + about)	**0.695**	**0.696**		

6. EXPERIMENTAL RESULTS

6.1 Recommendation accuracy

Experimental results are summarised in Table 1. LightFM performs very well, outperforming or matching the specialised model for each scenario.

In the **warm-start, low-sparsity** case (warm-start MovieLens), LightFM outperforms MF slightly when using both tag and item indicator features. This suggest that using metadata features may be valuable even when abundant interaction data is present.

Notably, LightFM (tags) almost matches MF performance despite using only metadata features. The LSI-LR and LSI-UP models using the same information fare much worse. This demonstrates that *(1)* it is crucial to use collaborative information when estimating content feature embeddings, and *(2)* LightFM can capture that information much more accurately than other hybrid models such as LSI-UP.

In the **warm-start, high-sparsity** case (warm-start CrossValidated), MF performs very poorly. Because user interaction data is sparse (the CrossValidated user-item matrix is 99.95% sparse vs only 99% for the MovieLens dataset), MF is unable to learn good latent representations. Content-based models such as LSI-LR perform much better.

LightFM variants provide the best performance. LightFM (tags + about) is by far the best model, showing the added advantage of LightFM's ability to integrate user metadata embeddings into the recommendation model. This is likely due to improved prediction performance for users with little data in the training set.

Results for the **cold-start** cases are broadly similar. On the CrossValidated dataset, all variants of LightFM outperform other models; LightFM (tags + about) again provides the best performance. Interestingly, LightFM (tags + indicators) outperforms LightFM (tags) slightly on the MovieLens dataset, even though no embeddings can be estimated for movies in the test set. This suggests that using both metadata and per-movie features allows the model to estimate better embeddings for both, much like the use of user and item bias terms allows better latent factors to be computed. Unsurprisingly, MF performs no better than random in the cold-start case.

In all scenarios the LSI-UP model performs no better than the LSI-LR model, despite its attempt to incorporate collaborative data. On the CrossValidated dataset it performs strictly worse. This might be because its latent representations are estimated on less data than in LSI-LR: as there are fewer users than items in the dataset, there are fewer rows in the user-feature matrix than in the item-feature matrix.

The results confirm that LightFM encompasses both the MF and the LSI-LR model as special cases, performing better than the LSI-LR model in the sparse-data scenario and better than the MF model in the dense-data case. This means not only that a single model can be maintained in either settings, but also that the model will continue to perform well even when the sparsity structure of that data changes.

Good performance of LightFM (tags) in both datasets is predicated on the availability of high-quality metadata. Nevertheless, it is often possible to obtain good quality metadata from item descriptions (genres, actor lists and so on), expert or community tagging (*Pandora* [23], *StackOverflow*), or computer vision systems where image or audio data is available (we use image-based convolutional neural networks for product tagging). In fact, the feature embeddings produced by LightFM can themselves be used to assist the tagging process by suggesting related tags.

6.2 Parameter Sensitivity

Figure 1 plots the accuracy of LightFM, LSI-LR, and LSI-UP against values of the latent dimensionality hyperparameter d in the cold-start scenario (averaged over 30 runs of each algorithm). As d increases, each model is capable of modelling more complex structures and achieves better performance.

Interestingly, LightFM performs very well even with a small number of dimensions. In both datasets LightFM consistently outperforms other models, achieving high performance with as few as 16 dimensions. On CrossValidated data, it achieves the same performance as the LSI-LR model for much smaller d: it matches the accuracy of the 512-dimensional LSI-LR model even when using fewer than 32 dimensions.

This is an important win for large-scale recommender systems, where the choice of d is governed by a trade-off between vector size and recommendation accuracy. Since smaller vectors occupy less memory and use fewer computations during query time, better representational power at small d allows the system to achieve the same model performance at a smaller computational cost.

Table 2: Tag similarity

Query tag	Similar tags
'regression'	'least squares', 'multiple regression', 'regression coefficients', 'multicollinearity'
'MCMC'	'BUGS', 'Metropolis-Hastings', 'Beta-Binomial', 'Gibbs', 'Bayesian'
'survival'	'epidemiology', 'Cox model', 'Kaplan-Meier', 'hazard'
'art house'	'pretentious', 'boring', 'graphic novel', 'pointless', 'weird'
'dystopia'	'post-apocalyptic', 'futuristic', 'artificial intelligence'
'bond'	'007', 'secret service', 'nuclear bomb', 'spying', 'assassin'

6.3 Tag embeddings

Feature embeddings generated by the LightFM model capture important information about the semantic relationships between different features. Table 2 gives some examples by listing groups of tags similar (in the cosine similarity sense) to a given query tag.

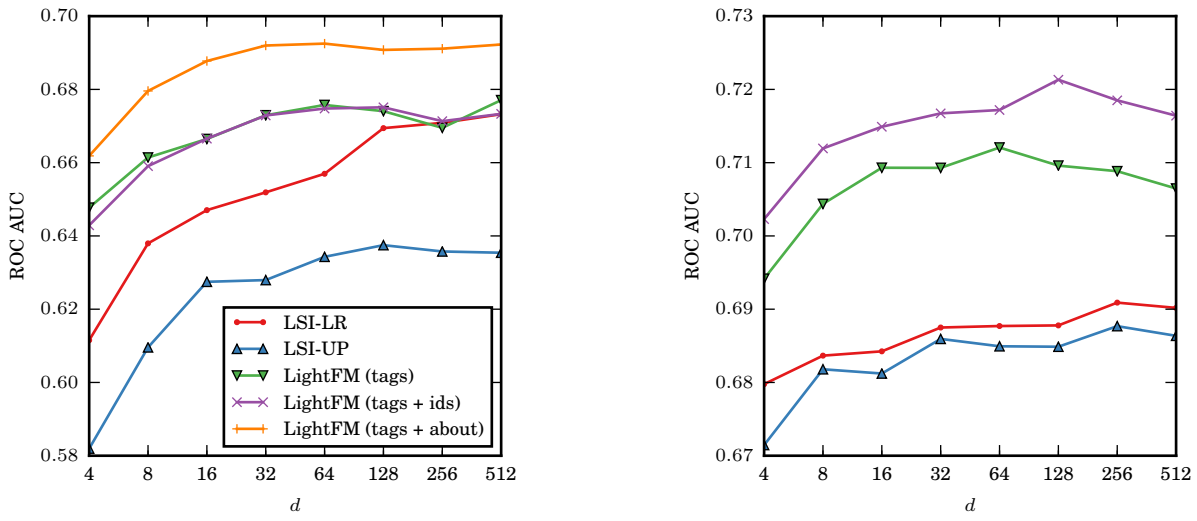
In this respect, LightFM is similar to recent word embedding approaches like word2vec and GloVe [12, 13]. This is perhaps unsurprising, given that word embedding techniques are closely related to forms of matrix factorisation [9]. Nevertheless, LightFM and word embeddings differ in one important respect: whilst word2vec and GloVe embeddings are driven by textual corpus co-incidence statistics, LightFM is based on user interaction data.

LightFM embeddings are useful for a number of recommendation tasks.

1. **Tag recommendation.** Various applications use collaborative tagging as a way of generating richer metadata for use in search and recommender system [2, 7]. A tag recommender can enhance this process by either automatically applying matching tags, or generating suggested tags lists for approval by users. LightFM-produced tag embeddings will work well for this task without the need to build a separate specialised model for tag recommendations.

2. **Genre or category recommendation.** Many domains are characterised by an ontology of genres or categories which play an important role in the presentation of recommendations. For example, the Netflix interface is organised in genre rows; for Lyst, fashion designers, categories and subcategories are fundamental. The degree of similarity between the embeddings of genres or categories provides a ready basis for genre or category recommendations that respect the semantic structure of the ontology.

Figure 1: Latent dimension sensitivity



(a) CrossValidated

(b) MovieLens

3. **Recommendation justification.** Rich information encoded in feature embeddings can help provide explanations for recommendations made by the system. For example, we might recommend a ball gown to a user who likes pencil skirts, and justify it by the two features' similarity as revealed by the distance between their latent factors.

7. USAGE IN PRODUCTION SYSTEMS

The LightFM approach is motivated by our experience at Lyst. We have deployed LightFM in production, and successfully use it for a number of recommendation tasks. In this section, I describe some of the engineering and algorithm choices that make this possible.

7.1 Model training and fold-in

Thousands of new items and users appear on Lyst every day. To cope with this, we train our LightFM model in an online manner, continually updating the representations of existing features and creating fresh representations for features that we have never observed before.

We store model state, including feature embeddings and accumulated squared gradient information in a database. When new data on user interaction arrives, we restore the model state and resume training, folding in any newly observed features. Since our implementation uses per-parameter diminishing learning rates (ADAGRAD), any updates of established features will be incremental as the model adapts to new data. For new features, a high learning rate is used to allow useful embeddings to be learned as quickly as possible.

No re-training is necessary for folding in new products: their representation can be immediately computed as the sum of the representations of their features.

7.2 Feature engineering

Each of our products is described by a set of textual features as well as structured metadata such as its type (dress, shoes and so on) or designer. These are accompanied by additional features coming from two sources.

Firstly, we employ a team of experienced fashion moderators, helping us to derive more fine-grained features such as clothing categories and subcategories (peplum dress, halterneck and so on).

Secondly, we use machine learning systems for automatic feature detection. The most important of these is a set of deep convolutional neural networks deriving feature tags from product image data.

7.3 Approximate nearest neighbour searches

The biggest application of LightFM-derived item representations are related product recommendations: given a product, we would like to recommend other highly relevant products. To do this efficiently across 8 million products, we use a combination of approximate (for on-demand recommendations) and exact (for near-line computation) nearest neighbour search.

For approximate nearest neighbour (ANN) queries, we use Random Projection (RP) trees [4, 5]. RP trees are a variant of random-projection [3] based locality sensitive hashing (LSH).

In LSH, k-bit hash codes for each point $\boldsymbol{x}$ are generated by drawing random hyperplanes $\boldsymbol{v}$, and then setting the k-th bit of the hash code to 1 if $\boldsymbol{x} \cdot \boldsymbol{v} \geq 0$ and 0 otherwise. The approximate nearest neighbours of $\boldsymbol{x}$ are then other points that share the same hash code (or whose hash codes are within some small Hamming distance of each other).

While extremely fast, LSH has the undesirable property of sometimes producing very highly unbalanced distribution of points across all hash codes: if points are densely concentrated, many codes of the tree will apply to no products

while some will describe a very large number of points. This is unacceptable when building a production system, as it will lead to many queries being very slow.

RP trees provide much better guarantees about the size of leaf nodes: at each internal node, points are split based on the median distance to the chosen random hyperplane. This guarantees that at every split approximately half the points will be allocated to each leaf, making the distribution of points (and query performance) much more predictable.

8. CONCLUSIONS AND FUTURE WORK

In this paper, I have presented an effective hybrid recommender model dubbed LightFM. I have shown the following:

1. LightFM performs at least as well as a specialised model across a wide range of collaborative data sparsity scenarios. It outperforms existing content-based and hybrid models in cold-start scenarios where collaborative data is abundant or where user metadata is available.

2. It produces high-quality content feature embeddings that capture important semantic information about the problem domain, and can be used for related tasks such as tag recommendations.

Both properties make LightFM an attractive model, applicable both in cold- and warm-start settings. Nevertheless, I see two promising directions in extending the current approach.

Firstly, the model can be easily extended to use more sophisticated training methodologies. For example, an optimisation scheme using Weighted Approximate-Rank Pairwise loss [24] or directly optimising mean reciprocal rank could be used [19].

Secondly, there is no easy way of incorporating visual or audio features in the present formulation of LightFM. At Lyst, we use a two-step process to address this: we first use convolutional neural networks (CNNs) on image data to generate binary tags for all products, and then use the tags for generating recommendations. We conjecture that substantial improvements could be realised if the CNNs were trained with recommendation loss directly.

9. REFERENCES

[1] G. Adomavicius and A. Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *Knowledge and Data Engineering, IEEE Transactions on*, 17(6):734–749, 2005.

[2] M. Bastian, M. Hayes, W. Vaughan, S. Shah, P. Skomoroch, H. Kim, S. Uryasev, and C. Lloyd. Linkedin skills: large-scale topic extraction and inference. In *Proceedings of the 8th ACM Conference on Recommender systems*, pages 1–8. ACM, 2014.

[3] S. Dasgupta. Experiments with random projection. In *Proceedings of the Sixteenth conference on Uncertainty in artificial intelligence*, pages 143–151. Morgan Kaufmann Publishers Inc., 2000.

[4] S. Dasgupta and Y. Freund. Random projection trees and low dimensional manifolds. In *Proceedings of the fortieth annual ACM symposium on Theory of computing*, pages 537–546. ACM, 2008.

[5] S. Dasgupta and K. Sinha. Randomized partition trees for exact nearest neighbor search. *arXiv preprint arXiv:1302.1948*, 2013.

[6] J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. *The Journal of Machine Learning Research*, 12:2121–2159, 2011.

[7] R. Jäschke, L. Marinho, A. Hotho, L. Schmidt-Thieme, and G. Stumme. Tag recommendations in folksonomies. In *Knowledge Discovery in Databases: PKDD 2007*, pages 506–514. Springer, 2007.

[8] Y. Koren, R. Bell, and C. Volinsky. Matrix factorization techniques for recommender systems. *Computer*, (8):30–37, 2009.

[9] O. Levy and Y. Goldberg. Neural word embedding as implicit matrix factorization. In *Advances in Neural Information Processing Systems*, pages 2177–2185, 2014.

[10] P. Lops, M. De Gemmis, and G. Semeraro. Content-based recommender systems: State of the art and trends. In *Recommender systems handbook*, pages 73–105. Springer, 2011.

[11] J. McAuley and J. Leskovec. Hidden factors and hidden topics: understanding rating dimensions with review text. In *Proceedings of the 7th ACM conference on Recommender systems*, pages 165–172. ACM, 2013.

[12] T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[13] J. Pennington, R. Socher, and C. D. Manning. Glove: Global vectors for word representation. *Proceedings of the Empiricial Methods in Natural Language Processing (EMNLP 2014)*, 12, 2014.

[14] B. Recht, C. Re, S. Wright, and F. Niu. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In *Advances in Neural Information Processing Systems*, pages 693–701, 2011.

[15] S. Rendle. Factorization machines. In *Data Mining (ICDM), 2010 IEEE 10th International Conference on*, pages 995–1000. IEEE, 2010.

[16] S. Rendle, C. Freudenthaler, Z. Gantner, and L. Schmidt-Thieme. BPR: Bayesian personalized ranking from implicit feedback. In *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence*, pages 452–461. AUAI Press, 2009.

[17] J. San Pedro and A. Karatzoglou. Question recommendation for collaborative question answering systems with RankSLDA. In *Proceedings of the 8th ACM Conference on Recommender systems*, pages 193–200. ACM, 2014.

[18] M. Saveski and A. Mantrach. Item cold-start recommendations: learning local collective embeddings. In *Proceedings of the 8th ACM Conference on Recommender systems*, pages 89–96. ACM, 2014.

[19] Y. Shi, A. Karatzoglou, L. Baltrunas, M. Larson, N. Oliver, and A. Hanjalic. CLiMF: learning to maximize reciprocal rank with collaborative less-is-more filtering. In *Proceedings of the 6th ACM onference on Recommender systems*, pages 139–146. ACM, 2012.

[20] E. Shmueli, A. Kagian, Y. Koren, and R. Lempel. Care to Comment?: Recommendations for commenting on news stories. In *Proceedings of the 21st international conference on World Wide Web*, pages 429–438. ACM, 2012.

[21] I. Soboroff and C. Nicholas. Combining content and collaboration in text filtering. In *Proceedings of the IJCAI*, volume 99, pages 86–91, 1999.

[22] J. Vig, S. Sen, and J. Riedl. The tag genome: Encoding community knowledge to support novel interaction. *ACM Transactions on Interactive Intelligent Systems (TiiS)*, 2, 2012.

[23] T. Westergren. The music genome project. *Online: http://pandora. com/mgp*, 2007.

[24] J. Weston, S. Bengio, and N. Usunier. WSABIE: Scaling up to large vocabulary image annotation. In *IJCAI*, volume 11, pages 2764–2770, 2011.