The AI Movie Buff: Using Collaborative Topic Regression to incorporate scripts into Movie Recommendation

All code for this project can be found under the CTR repository on my github: https://github.com/siddsach

INTRODUCTION

Recommendation is a highly multifaceted task. For ideal performance, a recommender system must be able to effectively use multiple kinds of data to deliver high quality recommendations, avoid being biased toward items that have had many ratings, be able to understand new content, and be fast enough that the system is able to deliver recommendations in a reasonable timeframe. In this paper, I explore the application of the novel recommender algorithm Collaborative Topic Regression to the task of recommending movies using both ratings data and the scripts of the movies. This algorithm combines collaborative filtering and content analysis to make significant progress over other recommender algorithms in the objectives described above. With the advent of big data, big compute power, and online media, businesses that effectively apply the power of targeting can form the foundation of startups in virtually every B2C industry, and this algorithm certainly accomplishes that task.

BACKGROUND KNOWLEDGE

Collaborative Filtering via Matrix Factorization

Collaborative filtering is an approach to recommendation where one uses the relationships in the ratings data to make recommendations to users, based on both their ratings history and the ratings history of the items they interact with. Though there are multiple approaches to collaborative filtering, because of its ability to encode important underlying relationships in the data, Probabilistic matrix factorization is both empirically one of the most accurate methods to predict ratings and one of the most easily generalizable.

$$R^{m*n} = [r_{ij}]$$

where R is a ratings matrix with **m users** and **n items** and r_{ij} **is user i's rating of item j.** Because ratings by users of items consist of a many to many relationship, R is sparse, where most of the values in R are empty. I can make predictions for empty values of R by approximating the non-empty values in R **via decomposition**:

$$R^{m*n} \approx (U^{k*m})^T V^{k*n}$$

where k is a hyper parameter signifying the number of latent variables. In this method, I arrive at **latent variable representations** for both each user and each item.

$$U^{k*m} = [u_i]_{1,m}$$

$$V^{k*n} = [v_j]_{1,n}$$

where u_i **is a k-dimensional latent factor vector representing user i** and v_j **is k-dimensional latent factor vector representing item j**. I get the following likelihood for U and V given the ratings data

$$\text{Lik(U,V)} = \frac{-\lambda_u}{2}\sum_i u_i^T u_i - \frac{\lambda_v}{2}\sum_j (v_j)^T (v_j) + \sum_{i,j}(r_{ij} - u_i^T v_j)^2$$

Where λ_u **and** λ_v **are regularization parameters**. I arrive at U and V via a gradient descent algorithm using the likelihood function, and once I have them I can **make a prediction as follows**:

$$r_{ij} \approx u_i^T v_j$$

Though this method is generally effective at producing quality recommendations, it suffers from the cold-start problem of recommendation. If a particular has not been rated, PMF has no way to predict ratings. On a place like Amazon where users and items do not change too frequently, this doesn't necessarily matter so much, whereas in a context where content has high churn, like in media or news, this makes recommendations effectively useless as PMF is often only able to capture information about the most popular and thus commonly rated items in an environment that requires effectively empowering new content.

Content Analysis

Another approach to recommendation tries to understand how the actual substance of the items being recommended relates to users' preferences. Most approaches to content analysis are difficult, however, because understanding the relevant attributes and performing the right feature engineering is a difficult and time-consuming process that requires deep contextual knowledge, as content is often very high-dimensional data that is challenging to use effectively. Topic modeling provides a surprisingly effective solution.

Latent Dirichlet Allocation is an **unsupervised learning** approach that takes in a corpus of documents and **models each document as a collection of latent topics learned using word frequencies**. The generative process of LDA with some prior number of topics K is as follows.

For each topic k, for each word w in the possible vocabulary :
choose $\beta_{k,w} = P(\textit{getting word } w \mid \textit{topic } k)$

For each document j in the corpus,
choose θ_j *from Dirichlet*(α) *where* $k-$*dimensional* θ_j
is topic distribution for document j

for each w in length(*document*),
Draw topic k from Multinomial(θ_j)
Draw word w from Multinomial(β_k)

Now, with these assumptions of how the data might have been generated, I can **learn beta and theta** using the following algorithm.

1) Assign all the words in all the documents random topics.

2) **Update Step:** Repeat until topic assignments don't change

For each document j in the corpus,
for each word w in document j,
Assume all current topic assignments are correct
Assign word w the new topic = $argmax_k\ (\theta_{jk} * \beta_{kw})$

3) **Output**
I now have for each document a k-dimensional vector with norm one that **represents that document as a distribution over topics** as well as for each topic a v-dimensional vector with norm 1, that **represents a topic as a distribution of words** with vocabulary size v.

Now that I can represent every document as a distribution over these topics, I can make recommendations by comparing the weighted mean vector of the topic distributions of the movies they've watched to make predictions about which new movies they'd like based on the topic distributions of those movies. As a result, **LDA is able to make recommendations on new movies with no rating history**. Its drawback, however, is that its recommendations tend to be weak, as **topic distributions by themselves tend not to be as effective at PMF at capturing deeper relationships** in the ratings data because they depend solely on word frequencies.

HYBRID MODEL

Collaborative Topic Regression

As it may have already implicitly been illustrated, the two mentioned models of recommendations are similar in the sense that they both come up with **k-dimensional vector representations of items being rated**, and it is in this manner that I can **combine the two methods to naturally incorporate the advantages of both**. I motivate a similar model to Probabilistic Matrix Factorization, except that I will have:

$$v_j = \epsilon_j + \theta_j$$

where ϵ_j is a latent factor offset to the topic distribution θ_j. As a result, v_j **encodes rating information in the same latent space as the topic distribution**, adding information where needed to maximize the likelihood function of the data which goes as follows:

$$L = -\ \frac{-\lambda_u}{2} \sum_i u_i^T u_i - \frac{\lambda_v}{2} \sum_j (v_j - \theta_j)^T (v_j - \theta_j)$$

$$+ \sum_j \sum_k log\,(\sum_k \theta_{jk}\beta_{k,w_{jn}}) - \sum_{i,j} \frac{c_{ij}}{2}(r_{ij} - u_i^T v_j)^2$$

The first two terms represent a **penalty for overfitting the latent variable vectors u_i and $\epsilon_j = v_j - \theta_j$, with regularization parameters λ_u and λ_v**. The third term represents maximizing the likelihood of the topic and word distributions via the LDA algorithm, and the fourth term represents errors in prediction errors, weighted by the confidence term c_ij. Because the ratings matrix R is sparse, and the likelihood function incorporates all of the elements of R, I **must impute null values and impose a weighting system** where real ratings are weighted with greater importance than imputed ratings.

$$C = [c_{ij}] = \{ \begin{array}{l} a \text{ if } r_{ij} \text{ exists} \\ \text{b if I use an imputed value} \\ \text{where } a > b > 0 \end{array}$$

where a and b are hyperparameters. I then take the gradient of this likelihood function, and, given number of free variables must optimize via coordinate ascent by solving the system of equations I get from setting:

$$\frac{dL}{du_i} = 0$$

$$\frac{dl}{dv_j} = 0$$

I could also do the same for theta and beta, but it turns out as Wang and Blei note that the parameters from pure LDA give comparable results. **I end up with the following update steps**:

$$u_i = (V C_i V^T + \lambda_u I_k)^{-1} V C_i R_i$$
$$v_j = (U C_j U^T + \lambda_v I_k)^{-1} (U C_j R_j + \lambda_v \theta_j)$$

where I_k is a k-dimensional identity matrix, and i and j are row and column indexing respectively. In summary, **I initialize U and V with Gaussian priors as in PMF**, and I update U and V iteratively until the test-error rate converges. I then make predictions for ratings in the same way as in PMF. I end up with hyper parameters:

$$\lambda_u,\ \lambda_v,\ (a,\ b),\ k\ (\textit{topic number}),\ v\ (\textit{vocabulary size}),$$
$$\textit{and } n\ (\textit{for imputing null values})$$

Cross-validation

To evaluate this model using a form of cross-validation appropriate in this context, I split the ratings data into four folds: Training, Testing-in-Matrix, Testing-out-of-Matrix, and

CTR-LDA In-matrix. The idea is that because this model allows us to provide recommendations regarding content with no ratings data, I want to differentiate in-matrix and out-of-matrix predictions. In-matrix predictions are predicted ratings for items that have already been rated, and out-of-matrix predictions are predicted ratings for items that have no ratings history, but have a topic distribution I can use to make recommendations by setting:

$$v_j = \theta_j$$

Data

To apply this model, I used the canonical MovieLens-100k ratings dataset, which contains 100,000 ratings from 1000 users on 1700 movies with at least 20 ratings, linking the ids for these movies with scripts scraped from www.springfield.co.uk, where 9,755 movies were common to both datasets. In addition, because of the low memory on my computer, I used a subset of the data that included only movies with greater than 50 ratings to decrease the size of U and V so that the model used less memory and trained fast enough for me to search over a larger number of hyper parameter combinations, leaving 38657 ratings.

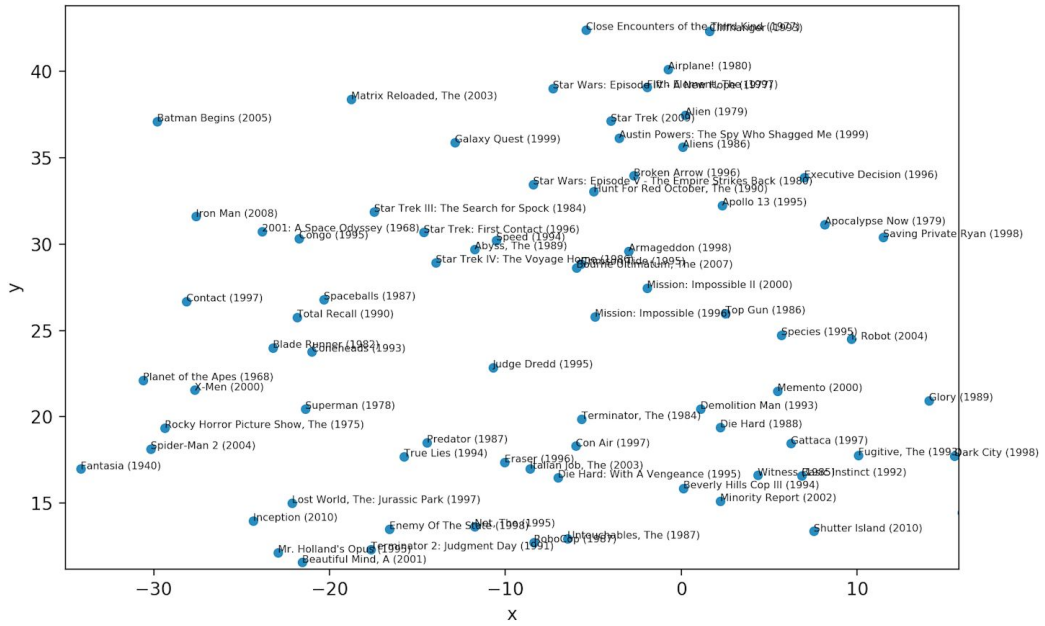
EXPERIMENTAL RESULTS

Qualitative Results

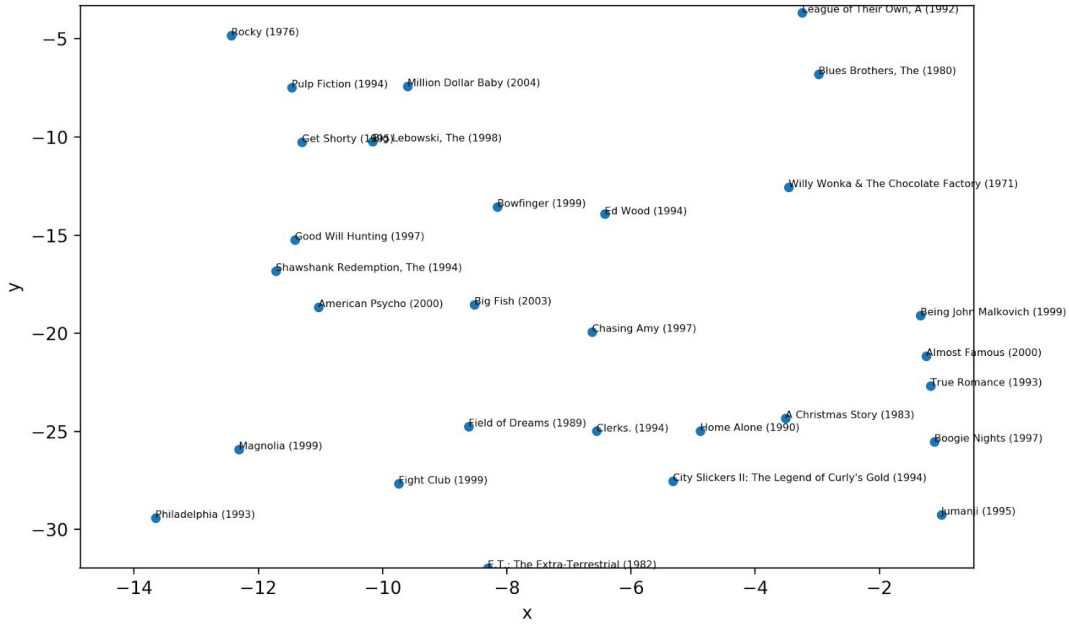
Graphing movie vectors

Other than just ratings predictions, the Collaborative Topic Regression model outputs latent factor representations of each user and movie in a latent space. Because users are anonymous, visualizing them won't yield much meaning, but I can visually evaluate the interpretative quality of each of the vectors v_j for each movie j, so I projected movies with greater than 50 ratings into 2 variables using the t-SNE dimension reduction algorithm.

Some of the clusters that vectors forms are ones like genre, which one might expect, like this group of Sci-fi movies (as well as movie trilogies like Mission Impossible being close together):



More interestingly, though, are the clusters one perhaps wouldn't imagine at first, but make sense in retrospect, like this cluster of classic 80s and 90s movies.



Interpreting Topics

The output of LDA gives word distributions for each topic, and inspecting these topics can

give us a sense of what the algorithm is actually learning. Here are a few of the topics clearly signifying some easily interpretable likely meaning as represented by their top 20 words.

> *Education → teachers library hawk gentlemen science robinson teaching thank campus study education teach sir university student teacher students college class school*
>
> *Film → movies scene stuff feel picture started life camera work big real gonna came things wanted story lot didn kind film*
>
> *Christmas → turkey toy bells jingle toys presents mo marley lll vic walt la ho claus year poppy tree lm snow 0000*
>
> *Music → hear doo stage piano gonna dance night songs guitar record hey playing singing rock play yeah band sing la do*
>
> *Law → believe attorney question prison justice murder jury truth state witness years evidence lawyer trial guilty honor didn law judge case*
>
> *Military → fight order stand ship soldiers gentlemen enemy aye orders officer soldier thank lieutenant sergeant army colonel general war men captain*

Interestingly, because I used scripts as text data, a few topics seemed to capture patterns in vernacular rather than subject material, which of course might still be quite useful for recommendation:

> *Casual/Young → cause ow shh whoo hi aah thank sorry mmm ooh whoa okay mm ah huh gonna hmm um hey yeah*
>
> *Medieval Europe/Fantasy → blood live sir speak true brother heart majesty dead leave fight world die death master men life kill god lord*
>
> *People who swear a lot → christ listen sorry head bit dead mate ass didn kill shut motherfucker bitch fucked money gonna fuckin yeah shit fucking*

Some of the topic meanings weren't as clearly interpretable, but do clearly capture something, for example:

> *ringing groans screams chattering hey cheering shouting grunts grunting gasps speaking yeah screaming door uh woman laughs continues chuckles sighs*

Quantitative Results

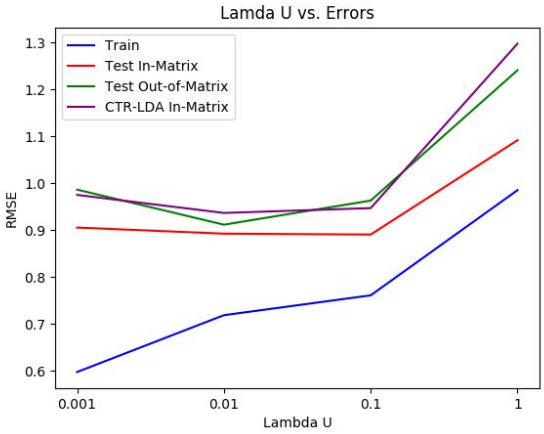
Hyper-parameter tuning

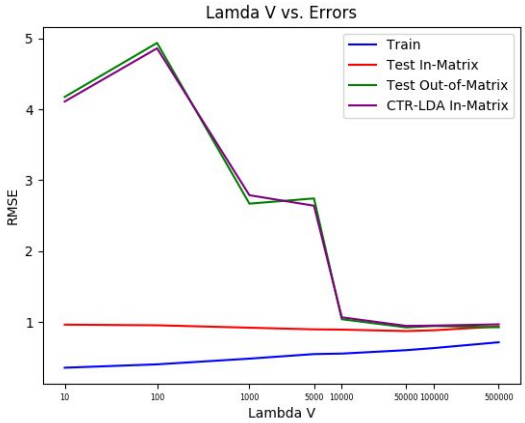
To find the optimal hyperparameters, I used the smaller dataset of movies with more than 50 ratings. Because of the high number of possible permutations of the 6 important hyperparameters, I couldn't use a blind grid search. As a result, I split the hyperparameter optimization process into three categories within which inter-parameter interaction really matters: regularization (λ_u, λ_v), dealing with null values ((a,b), n), and text interpretation (k, v). Within each of these categories, I used grid search over a large number of combinations to find the optimal set within that category, while using reliable baseline inputs for the hyper parameters in other categories. I've graphed the hyperparameter results below with all other parameters at optimal values.

→ Regularization

λ_u **and** λ_v are parameters controlling the penalty on overfitting the user vectors u_i and item

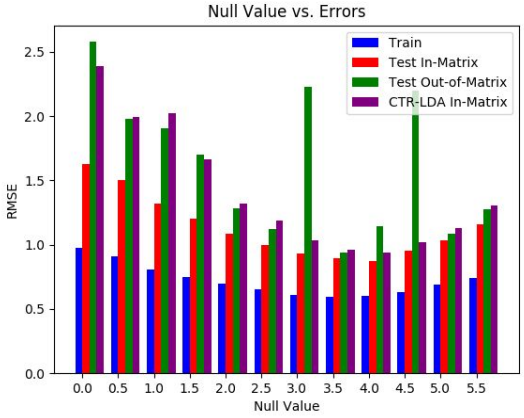
latent offset vectors $\epsilon_j = v_j - \theta_j$, and so when we vary them, we observe a tradeoff between train and all of the testing errors. In addition, because λ_v controls the overfitting of the latent offset that is learned based on ratings data, we observe that a high λ_v to limit the size of the latent offset is necessary to effective out-of-matrix predictions using θ_j.

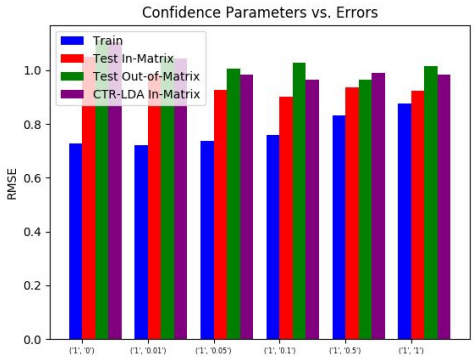




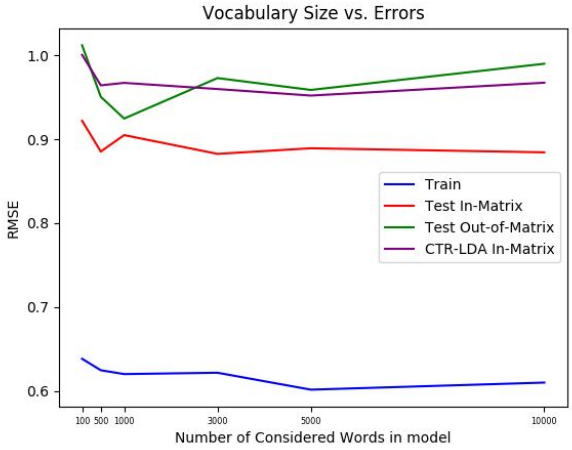
→ Dealing with Null Values

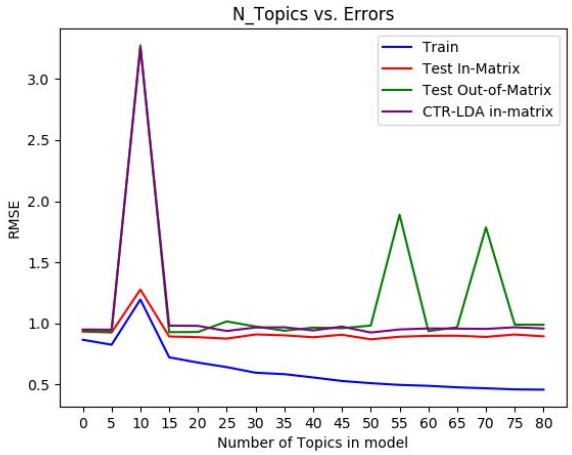
While imputing null-values, I found that using 3.5, which is around the mean of ratings in the dataset was consistently the most effective. In addition, I found that using the confidence parameters (a=1, b = 0.1) also ended up being the most effective.





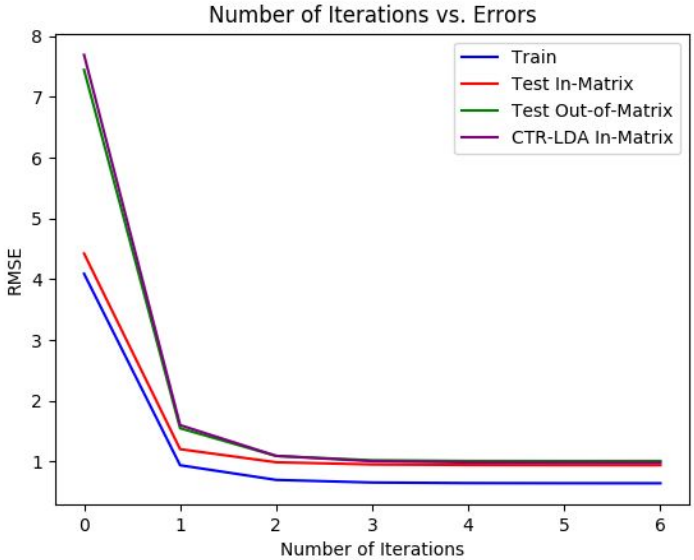
→ Text interpretation
The vocabulary size and number of topics led to erratic changes in the various error rates, probably because of the changing information that more words or topics provide. I chose a vocab size of 5000 because it gave reasonable performance for both in and out of matrix error. In addition, though increasing the topic size at first improved the model test errors, increases after 60 topics made no improvements in-matrix or out-of-matrix





Final Results
The CTR model takes a relatively small number of iterations to converge, and the error rate converges as follows:

After optimizing hyperparameters, the test performance on the full 100k dataset was as follows:



	Vanilla PMF	CTR In-Matrix	CTR out-of-matrix	CTR-LDA In-Matrix
Test RMSE	**0.942**	**0.882**	**0.977**	**0.953**

Remarkably, Collaborative Topic Regression not only made a X % improvement over simple PMF on in-matrix predictions, CTR performed quite well out-of-matrix, solely making predictions on the basis of topic distributions/scripts, and was able to carry that performance to out-of-matrix predictions, meaning that the model is able to deliver high-quality recommendations on new data.

Conclusion

Though CTR did improve on traditional PMF recommendation, its primary benefits are its strong interpretability and its ability to recommend new content well. Because the latent factors are encoded in the same latent space as the topic vectors given by LDA, and because I can interpret the topic vectors by looking at the distributions, CTR allows me to understand both users and items as distributions over these interpretable topics. This means ultimately that CTR not only gives strong quality recommendations, it makes significant progress in allowing one to interpret why those recommendations are being made. In addition, traditional PMF tends to perform well on the simple measure of RMSE, but if a recommender system is only able to recommend old, well-rated content well, it's not effective. Because CTR gives quality recommendations for new content, it makes an important improvement over PMF. In a business context, the interpretability that this kind of algorithm provides is crucial, as it turns targeting from a potentially dangerous black box to a source of ideas for inquiry and research. Moreover, the ability to understand new content allows a business to perform truly optimal targeting, less biased by cases for which there is more data.

As machine learning starts to become more and more automated, it becomes more and more important not to grow entirely dependent on well-documented libraries. In addition to giving me a strong foundation in recommender systems and text analysis, this project has also given me the confidence to build a model from just a paper and stay on the cutting edge. More than anything else, that is the significant business application of the output of this project.