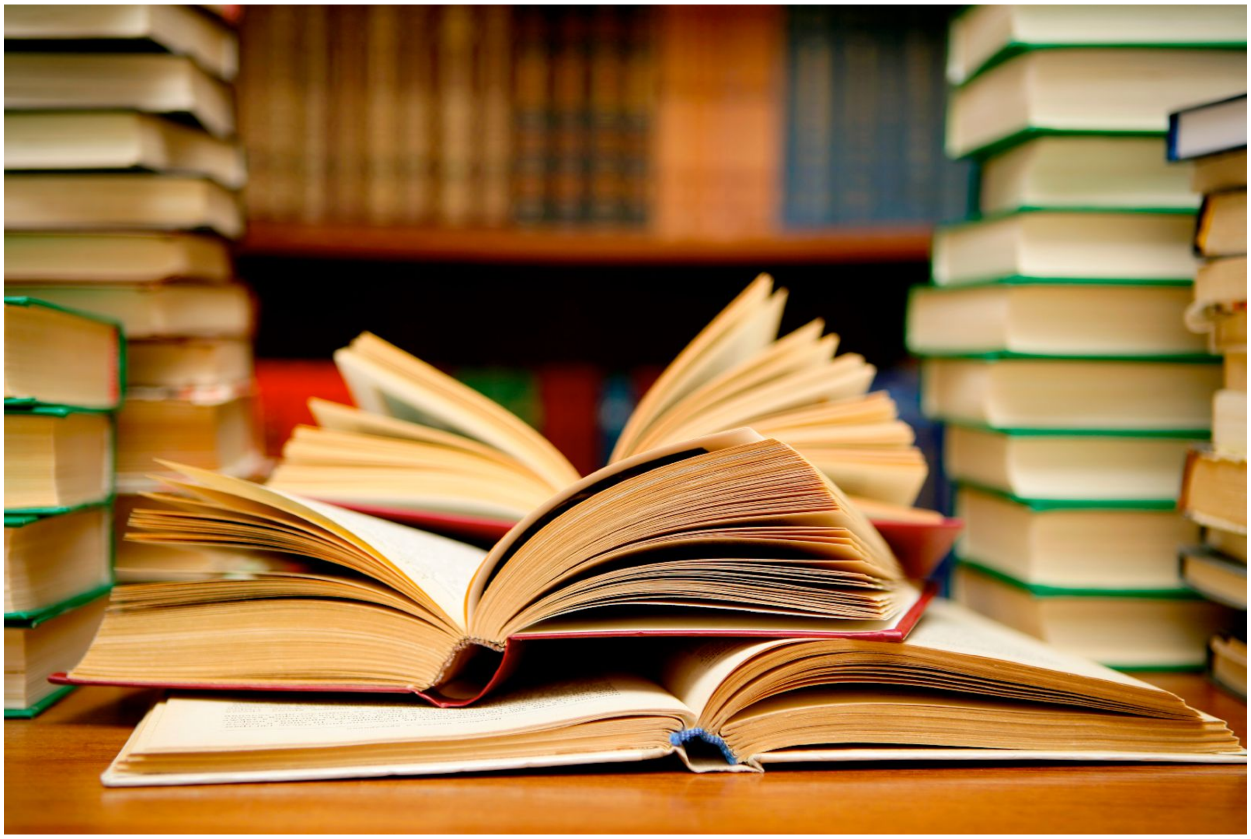
Summary	In this project we will run a comparison of emerging deep learning approaches to recommendation with classic and current state of the art approaches.
URL	https://github.com/manoj0237/DeeplearningandAI-finalproject
Category	Deep learning
Environment	Python 3, Keras, Surprise, Scikit-Learn
Status	In progress
Feedback Link	
Author	Manoj Karru, Nishant Gandhi, Emily Strong

Personalized Book Recommendations



04.14.2019

Emily Strong
Manoj Karru
Nishant Gandhi

Overview

Many companies face the challenge of providing personalized recommendations to users based only on the users' interactions with content on the site. Two classic approaches to this problem are logistic regression and collaborative filtering. Recent advances in deep learning have introduced counterparts to these approaches: collaborative deep learning and wide and deep learning. There is the question however of how these emerging techniques compare to the current state-of-the-art contextual multi-armed bandit approach. In this project we will compare the performance of these five algorithms in the problem of personalized book recommendations.

Goals

Evaluate the performance of emerging deep learning approaches to recommendation against classic and state-of-the-art techniques.

Data

Data set: https://sites.google.com/eng.ucsd.edu/ucsdbookgraph/home

The GoodReads data set is scraped from public data on GoodReads.com. The full data set contains over 200 million records for approximately 876k users and 2.4 million books. Since it is so large and Goodreads segments its recommendations based on category, we will focus on a single category in our experiments of *Mystery, Thriller & Crime*.

The public data on GoodReads does not contain any user data beyond their interaction history, so we will need to engineer features for the models that require them.

Algorithms

Collaborative Filtering

For collaborative filtering we will be be using the collaborative filtering algorithms available in the Surprise package. Two methods for collaborative filtering are KNN and matrix factorization. KNNs can only make recommendations for items that already have interaction data, and also require user data for the neighbors selection. Since we have to engineer all of our features from interaction data, this will not be an appropriate method for new users for us. Matrix factorization on the other hand does not require any user or content features, but also cannot model users or items with no history. We will test out both approaches with our data set as part of the baseline comparison.

Historically, collaborative filtering systems dealt with new users by requiring users rate a few items before making recommendations.

Logistic Regression

For logistic regression we will combine our engineered user features and content features to train a logistic regression using scikit-learn that will predict how the user would rate the content.

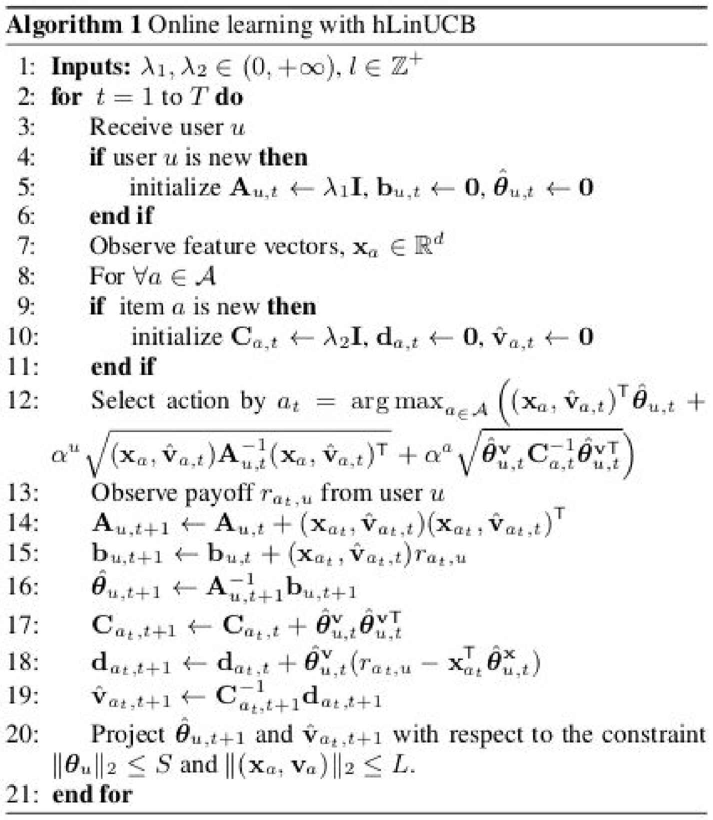
The logistic regression uses a combination of user and content features to make predictions for the given user for the given content. Both of these can be generated from non-interaction data depending on the data available, which reduces the impact of cold starts. However for our data set all of our user features will need to be generated from the interactions. We will thus need to use a vector of zeros to represent the user features and the output will be the default output of the logistic regression.

Contextual Multi-Armed Bandits

Contextual multi-armed bandit algorithms are the current standard family of algorithms used in recommendation problems. These are reinforcement learning algorithms that leverage probabilistic and regression techniques to address the exploration-exploitation tradeoff. Since none of the other methods we are using involve reinforcement learning, we only perform an offline experiment.

The specific algorithm we will use is hLinUCB. This algorithm trains a ridge regression for each item available for recommendation with hidden feature learning to account for limited user data, calculates a predicted value for the given user and uses the covariance to calculate the upper confidence bound on the prediction. The item with the highest predicted upper confidence bound is recommended.

Since a ridge regression is used, new items will still have models initialized with an identity matrix, and new users would receive the default recommendations.



Algorithm 1 Online learning with hLinUCB

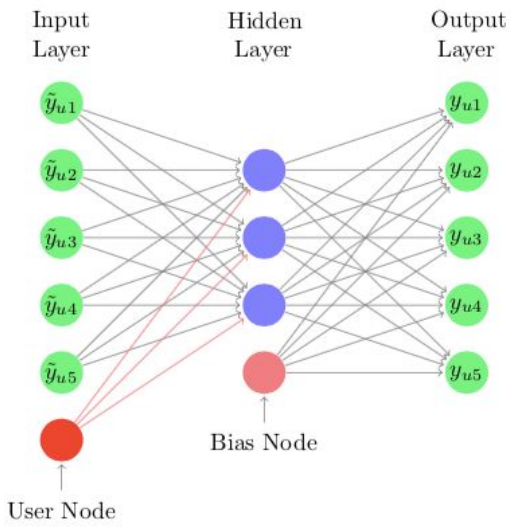
```
1:  Inputs: $\lambda_1, \lambda_2 \in (0, +\infty)$, $l \in \mathbb{Z}^+$
2:  for $t = 1$ to $T$ do
3:      Receive user $u$
4:      if user $u$ is new then
5:          initialize $\mathbf{A}_{u,t} \leftarrow \lambda_1 \mathbf{I}$, $\mathbf{b}_{u,t} \leftarrow \mathbf{0}$, $\hat{\boldsymbol{\theta}}_{u,t} \leftarrow \mathbf{0}$
6:      end if
7:      Observe feature vectors, $\mathbf{x}_a \in \mathbb{R}^d$
8:      For $\forall a \in \mathcal{A}$
9:      if item $a$ is new then
10:         initialize $\mathbf{C}_{a,t} \leftarrow \lambda_2 \mathbf{I}$, $\mathbf{d}_{a,t} \leftarrow \mathbf{0}$, $\hat{\mathbf{v}}_{a,t} \leftarrow \mathbf{0}$
11:     end if
12:     Select action by $a_t = \arg\max_{a \in \mathcal{A}} \Big( (\mathbf{x}_a, \hat{\mathbf{v}}_{a,t})^\mathsf{T} \hat{\boldsymbol{\theta}}_{u,t} + \alpha^u \sqrt{(\mathbf{x}_a, \hat{\mathbf{v}}_{a,t}) \mathbf{A}_{u,t}^{-1} (\mathbf{x}_a, \hat{\mathbf{v}}_{a,t})^\mathsf{T}} + \alpha^a \sqrt{\hat{\boldsymbol{\theta}}_{u,t}^{\mathbf{v}} \mathbf{C}_{a,t}^{-1} \hat{\boldsymbol{\theta}}_{u,t}^{\mathbf{v}\mathsf{T}}} \Big)$
13:     Observe payoff $r_{a_t,u}$ from user $u$
14:     $\mathbf{A}_{u,t+1} \leftarrow \mathbf{A}_{u,t} + (\mathbf{x}_{a_t}, \hat{\mathbf{v}}_{a_t,t})(\mathbf{x}_{a_t}, \hat{\mathbf{v}}_{a_t,t})^\mathsf{T}$
15:     $\mathbf{b}_{u,t+1} \leftarrow \mathbf{b}_{u,t} + (\mathbf{x}_{a_t}, \hat{\mathbf{v}}_{a_t,t}) r_{a_t,u}$
16:     $\hat{\boldsymbol{\theta}}_{u,t+1} \leftarrow \mathbf{A}_{u,t+1}^{-1} \mathbf{b}_{u,t+1}$
17:     $\mathbf{C}_{a_t,t+1} \leftarrow \mathbf{C}_{a_t,t} + \hat{\boldsymbol{\theta}}_{u,t}^{\mathbf{v}} \hat{\boldsymbol{\theta}}_{u,t}^{\mathbf{v}\mathsf{T}}$
18:     $\mathbf{d}_{a_t,t+1} \leftarrow \mathbf{d}_{a_t,t} + \hat{\boldsymbol{\theta}}_{u,t}^{\mathbf{v}} (r_{a_t,u} - \mathbf{x}_{a_t}^\mathsf{T} \hat{\boldsymbol{\theta}}_{u,t}^{\mathbf{x}})$
19:     $\hat{\mathbf{v}}_{a_t,t+1} \leftarrow \mathbf{C}_{a_t,t+1}^{-1} \mathbf{d}_{a_t,t+1}$
20:     Project $\hat{\boldsymbol{\theta}}_{u,t+1}$ and $\hat{\mathbf{v}}_{a_t,t+1}$ with respect to the constraint $\|\boldsymbol{\theta}_u\|_2 \leq S$ and $\|(\mathbf{x}_a, \mathbf{v}_a)\|_2 \leq L$.
21: end for
```

An implementation of hLinUCB is available in a library called BanditLib that we can use as a starting point for our work.

Collaborative Deep Learning

For collaborative deep learning we will use Collaborative Denoising Auto-Encoders. This is a technique that treats the missing ratings in the interaction data as noise and uses stacked auto-encoders to “denoise” them and predict their values. CDAE further attempts to improve the predictions by injecting latent features of the user.

The latent features of new users will be represented by a vector of zeros.



The authors have released a dataset-specific version of their code which we will be able to use as a model for how to design the architecture.

Wide and Deep Learning

Our final approach will be the wide and deep learning approach developed by engineers at Google. This combines a generalized linear model with a feed-forward network to create an ensemble method. For the generalized linear model, the cross product of binary features is calculated to capture interactions.

Features for new users will be represented by a vector of zeros.

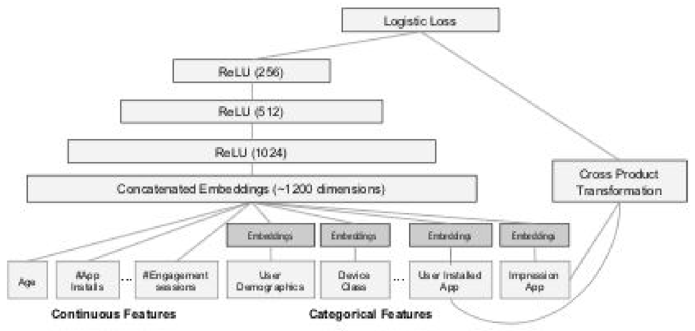


Figure 4: Wide & Deep model structure for apps recommendation.

The authors have released a dataset-specific version of their code which we will be able to use as a model for how to design the architecture.

Model Evaluation

We will evaluate each model based on replay with historic data, a method of evaluation in which only those recommendations that have historic ratings are scored. In this method, the model makes recommendation for an historic test set, and the top-k recommendations are evaluated.

The specific metrics we will use are mean average precision, and average precision and recall for all users. Another common metric for this type of problem is cumulative gains, however since there is no monetary reward associated with our recommendations we will not be using this. We will look at the top 10 recommendations, similar to what might be displayed for a website user.

Process Outline

1. Data Wrangling
2. Exploratory Data Analysis
3. Feature Engineering
    a. Text features of the books will be embedded using GloVe
    b. User-book interaction history will be used to generate additional features
4. Development of models and comparison of approaches

Milestones

Timeframe	Delivery
April 11-15	Exploratory Data Analysis & Feature Engineering
April 16-18	Model Building and Preliminary Training
April 19-23	Model Tuning and Evaluation
April 24-25	Final documentation

Deployment Details

1. Language: Python
2. Frameworks: Keras, Scikit-Learn, Surprise
3. NLP Embedding Tool: GloVe

References

Mengting Wan, Julian McAuley, "Item Recommendation on Monotonic Behavior Chains", in Proc. of 2018 ACM Conference on Recommender Systems (RecSys'18), Vancouver, Canada, Oct. 2018.

Koren, Yehuda, Robert Bell, and Chris Volinsky. "Matrix factorization techniques for recommender systems." *Computer* 8 (2009): 30-37.

Wang, Huazheng, Qingyun Wu, and Hongning Wang. "Learning hidden features for contextual bandits." *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*. ACM, 2016.

https://github.com/huazhengwang/BanditLib

Wu, Yao, et al. "Collaborative denoising auto-encoders for top-n recommender systems." *Proceedings of the Ninth ACM International Conference on Web Search and Data Mining*. ACM, 2016.

https://github.com/gtshs2/Collaborative-Denoising-Auto-Encoder

Cheng, Heng-Tze, et al. "Wide & deep learning for recommender systems." *Proceedings of the 1st workshop on deep learning for recommender systems*. ACM, 2016.

https://github.com/tensorflow/models/tree/master/official/wide_deep