Recommendation Engine for Videos using Cosine Similarity and MMoE

Gulnara Timokhina
Software Engineering Department
San Jose State University
San Jose, USA
gulnara.timokhina@sjsu.edu

Mirsaeid Abolghasemi
Computer Information System
De Anza College
Cupertino, USA
abolghasemimirsaeid@fhda.edu

Poornapragna Vadiraj
Software Engineering Department
San Jose State University
San Jose, USA
poornapragna.vadiraj@sjsu.edu

Varun Bhaseen
Software Engineering Department
San Jose State University
San Jose, USA
varun.bhaseen@sjsu.edu

1 ABSTRACT

In this paper we are working on a video recommendation Engine project which is using Multi gate Mixture of Experts (MMoE) to create a ranking model. Any recommendation engine has two components namely: candidate generation and ranking. For candidate generation we will be using a cosine similarity whereas for ranking the candidates we will be using MMoE. MMoE addresses and optimizes efficiently the many challenges of traditional Recommendation system like implicit bias, scalability and so on. The primary objective is that given a video which a user is currently watching, recommend the next video that the user might watch and enjoy. The entire project is executed on traditional Jupyter notebook (colab), Interactive context TFX pipeline and finally deployed on a web server for a web application.

Keywords: Multitask learning, Mixture of Experts, Recommendation System, Deep Learning, Multi gate mixture of experts

2 INTRODUCTION

2.1 PURPOSE

The primary purpose of this project is to create a recommendation engine using cosine similarity for generating candidate list and MMoE (Multi gate Mixture of Experts) for generating ranking. This is followed up with a web application which can use the recommendation engine model to recommend YouTube videos for users.

2.2 OBJECTIVES

Following are the three objectives targeted to achieve from this project:

- Create a web application for recommending Videos
- Colab to train the MMoE model to give scores for ranking
- Using TFX as ML pipeline

2.3 BACKGROUND

In recent years, deep neural network models have been successfully applied in many real-world large-scale applications, such as recommendation systems. Such recommendation systems often need to optimize multiple objectives at the same time. For example, when recommending movies for users to watch, we may want the users to not only purchase and watch the movies, but also to like the movies afterwards so that they will come back for more movies. That is, we can create models to predict both users' purchases and their ratings simultaneously. Researchers have reported multi-task learning models can improve model predictions on all tasks by utilizing regularization and transfer learning. This is different from a single task recommendation engine (as seen in Figure 1).

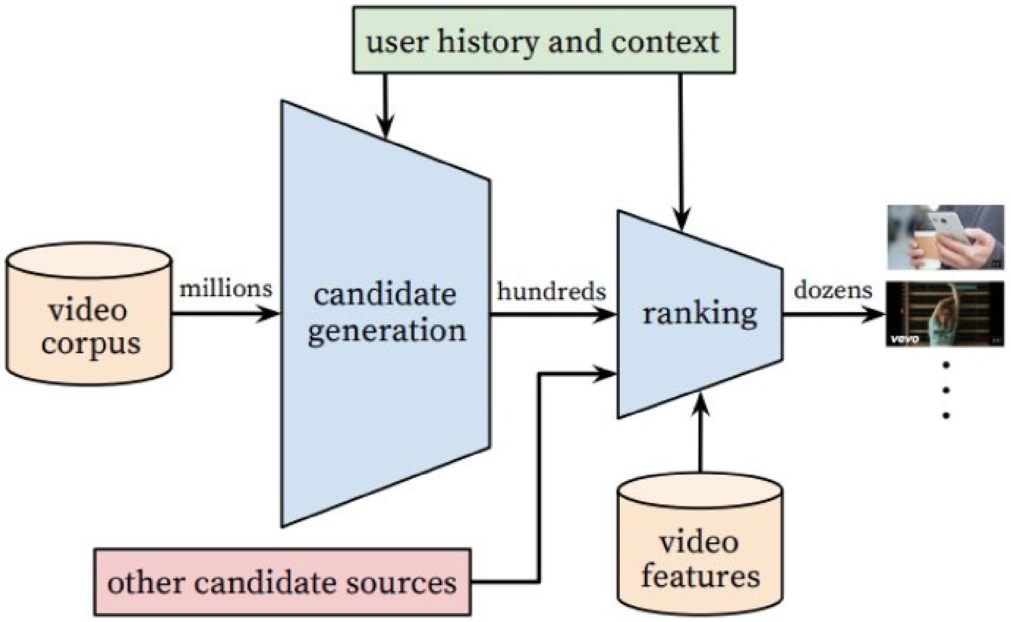


Figure 1: Single Task recommendation Engine

In this paper, we use a large-scale ranking system for video recommendation. Typical recommendation systems follow a two-stage design with a candidate generation and a ranking. We will see in detail the architecture for both generating candidate list and ranking.

2.4 Problem Statement

Designing and developing a real-world large-scale video recommendation system is full of challenges, including:

- There are often different and sometimes conflicting objectives which we want to optimize for. For example, we may want to recommend videos that users rate highly and share with their friends, in addition to watching.
- There is often implicit bias in the system. For example, a user might have clicked and watched a video simply because it was being ranked high, not because it was the one that the user liked the most. Therefore, models trained using data generated from the current system will be biased, causing a feedback loop effect.

2.5 Solution and Approach

To address these challenges, we are using an efficient multitask neural network architecture for the ranking system, as shown in Figure 2. It extends the Wide & Deep model architecture by adopting Multi-gate Mixture-of-Experts (MMoE) for multitask learning. In addition, it introduces a shallow tower to model and remove selection bias. We apply the architecture to a web application for video recommendation as a case study: given what user currently is watching, recommend the next video to watch.

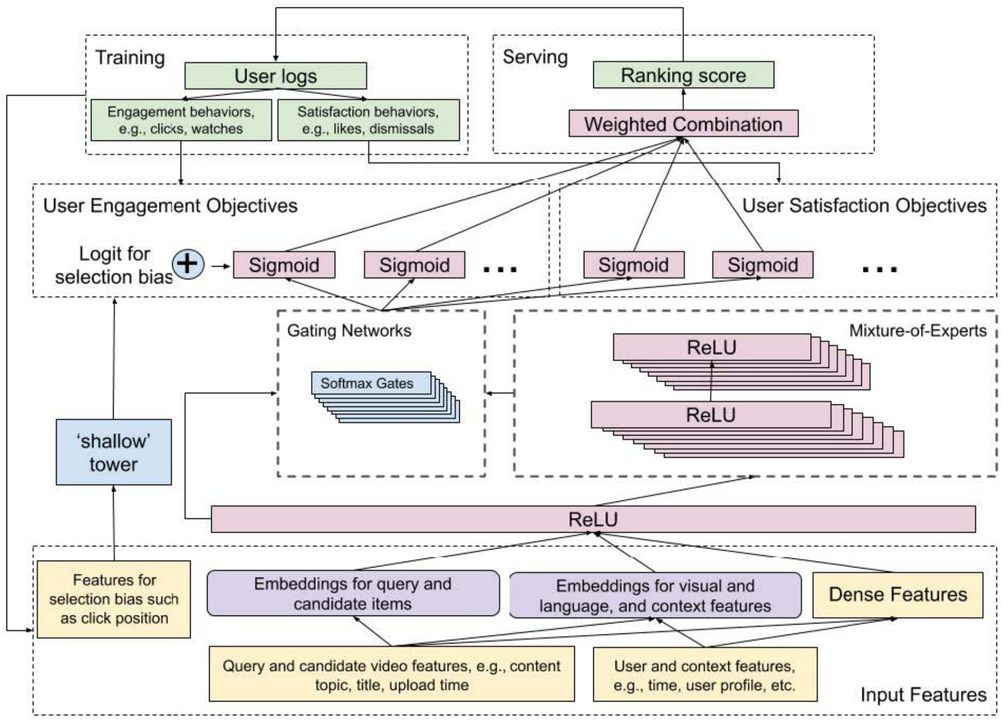

Training
User logs
Engagement behaviors, e.g., clicks, watches
Satisfaction behaviors, e.g., likes, dismissals
Serving
Ranking score
Weighted Combination
User Engagement Objectives
User Satisfaction Objectives
Logit for selection bias
Sigmoid
Sigmoid
Sigmoid
Sigmoid
Gating Networks
Softmax Gates
Mixture-of-Experts
ReLU
ReLU
'shallow' tower
ReLU
Features for selection bias such as click position
Embeddings for query and candidate items
Embeddings for visual and language, and context features
Dense Features
Query and candidate video features, e.g., content topic, title, upload time
User and context features, e.g., time, user profile, etc.
Input Features

Figure 2: Model architecture of the ranking system. It consumes user logs as training data, builds Multi-gate Mixture-of-Experts layers to predict two categories of user behaviors, i.e., engagement and satisfaction. It corrects selection bias with a side tower.

we use MMoE to automatically learn parameters to share across potentially conflicting objectives. The Mixture-of-Experts architecture modularizes input layer into experts, each of which focuses on different aspects of input. This improves the representation learned from complicated feature space generated from multiple modalities. Then by utilizing multiple gating networks, each of the objectives can choose experts to share or not share with others.

3 Related Work

3.1 Multi-task Learning in DNNs

Multi-task models can learn commonalities and differences across different tasks. Doing so can result in both improved efficiency and model quality for each task [5, 6, 7]. One of the widely used multi-task learning models is proposed by Caruana [6, 8], which has a shared-bottom model structure, where the bottom hidden layers are shared across tasks. This structure substantially reduces the risk of overfitting but can suffer from optimization conflicts caused by task differences, because all tasks need to use the same set of parameters on shared-bottom layers.

3.2 Ensemble of Subnets & Mixture of Experts

In this paper, we apply some recent findings in deep learning such as parameter modulation and ensemble method to model task relationships for multi-task learning. In DNNs, ensemble models and ensemble of subnetworks have been proven to be able to improve model performance [8, 9].

Eigen et al [10] and Shazeer et al [11] turn the mixture-of-experts model into basic building blocks (MoE layer) and stack them in a DNN. The MoE layer selects subnets (experts) based on the input of the layer at both training time and serving time. Therefore, this model is not only more powerful in modeling but also lowers computation cost by introducing sparsity into the gating networks.

Similarly, PathNet [12], which is designed for artificial general intelligence to handle different tasks, is a huge neural network with multiple layers and multiple submodules within each layer. While training for one task, multiple pathways are randomly selected and trained by different workers in parallel. The parameters of the best pathway are fixed, and new pathways are selected for training new tasks.

3.3 Multi-task Learning Applications

Thanks to the development of distributed machine learning systems [13], many large-scale real-world applications have adopted DNN-based multi-task learning algorithms and observed substantial quality improvements. On multi-lingual machine translation tasks, with shared model parameters, translation tasks having limited training data can be improved by jointly learning with tasks having large amount of training data [15]. For building recommendation systems, multi-task learning is found helpful for providing context-aware recommendations [16, 17]. In [14], a text recommendation task is improved by sharing feature representations and lower-level hidden layers. In [11], a shared-bottom model is used to learn a ranking algorithm for video recommendation.

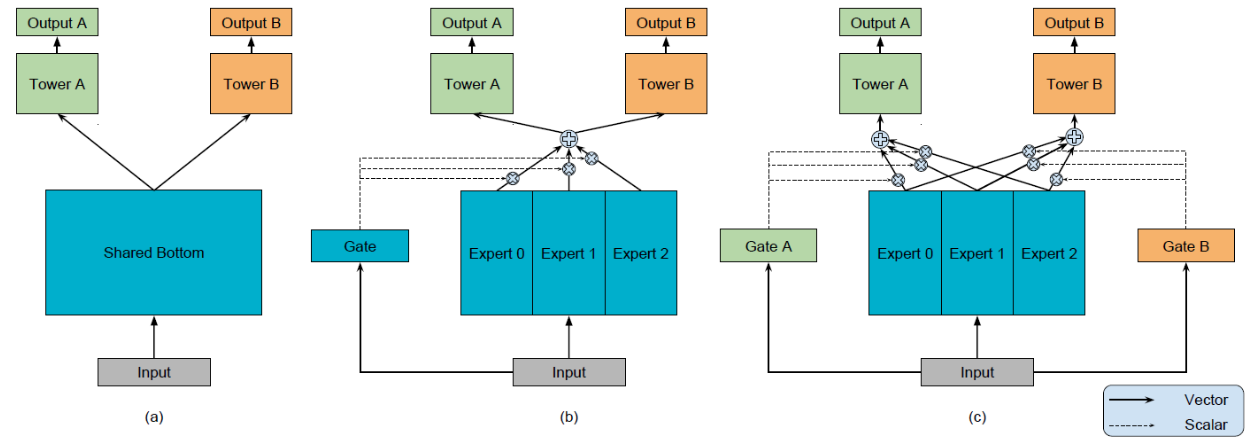


Figure 3: (a) Shared-Bottom model. (b) One-gate MoE model. (c) Multi-gate MoE model.

4 Data

4.1 Data Retrieval

The dataset that has been taken is from **Kaggle** and can be accessed from this link. The dataset is called trending YouTube video statistics. This dataset was collected using the YouTube API.

This dataset includes several months (and counting) of data on daily trending YouTube videos. Data is included for the US, GB, DE, CA, RU, MX, KR, JP, IN and FR regions (USA, Great Britain, Germany, Canada, Russia, Mexico, South Korea, Japan, India, and France, respectively), with up to 200 listed trending videos per day. Each region's data is in a separate file. Data includes the video title, channel title, publish time, tags, views, likes, and dislikes, description, and comment count.

4.2 Data Preprocessing

The data also includes a category_id field, which varies between regions. To retrieve the categories for a specific video, find it in the associated JSON. One such file is included for each of the five regions in the dataset.

The dataset that we are using for the project is for only united states region. We are focusing on following features:

- video titles
- channel title
- publish time
- tags
- views
- likes
- dislikes
- description
- comment count

4.3 Data Enrichment

Apart from this we are also generating some user specific data which can help us with simulation. We were unable to find any user specific data since all user details, user id, user clicks, spend duration etc. is confidential information and internal to google and hence, not publicly available.

Following are the features that we are generating to simulate user experience:

- user click
- user rating
- time spend
- position
- position bias
- device information
- video embedding
- user embedding

The video embedding and user embedding is generated using BERT tokenizer and BERT model, whereas other features are generated using random choice and random integers

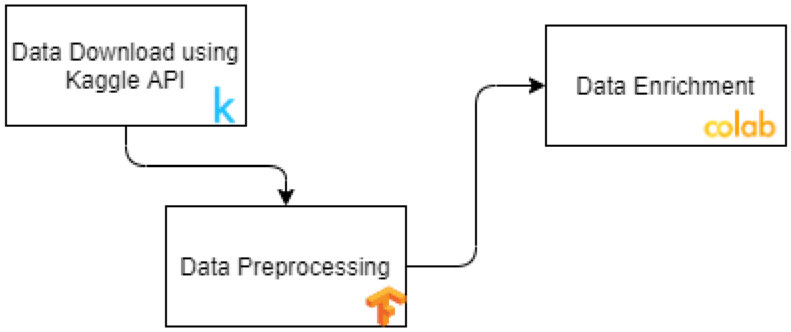


Figure 4: Data Architecture

5 Methods and Implementation

5.1 Model Architecture

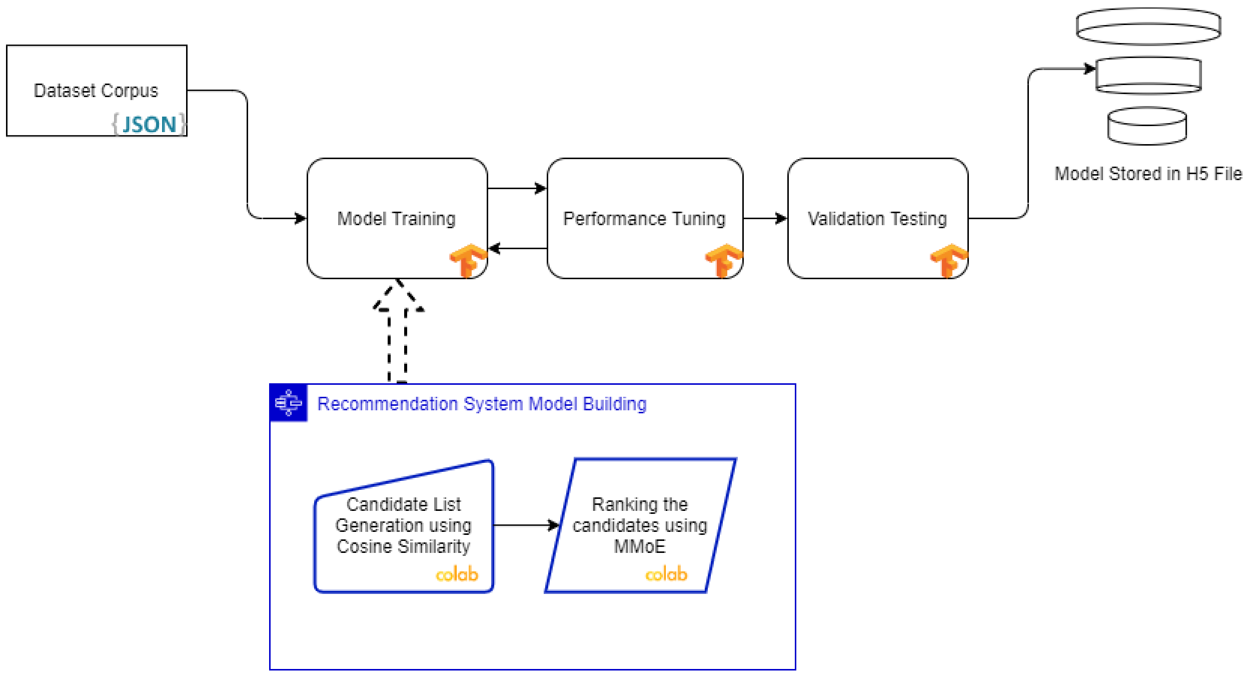

Dataset Corpus
{JSON}
Model Training
Performance Tuning
Validation Testing
Model Stored in H5 File
Recommendation System Model Building
Candidate List Generation using Cosine Similarity
colab
Ranking the candidates using MMoE
colab

Figure 5: Model Architecture. The Recommendation System has two parts: Candidate and Ranking. Ranking is determining Satisfaction and Engagement Score which gives final output

In our model we are using the dataset corpus that we have created from previous stage. The dataset is provided to our model for training. As in case of any recommendation engine we have two components:

- Candidate List Generation
- Generate Ranking

5.1.1 Candidate List Generation

Our video recommendation system uses cosine similarity for multiple candidate generation, each of which captures one aspect of similarity between query from user and candidate video. In many large-scale systems and organizations this algorithm can be replaced with any other state of the art technique used for candidate generation. The model we selected is content based model (rather than collaborative or hybrid based) and hence dependent on features like user clicks and user profile which we have generated in data preprocessing for ease of simulation

5.1.1.1 Alternative Approaches

Candidate Generation as stated earlier can be created using any techniques or even combination of techniques. For example, one algorithm generates candidates by matching topics of query video. Another algorithm retrieves candidate videos based on how often the video has been watched together with the query video. We can also construct a sequence model similar for generating personalized candidate given user history. We can also use techniques to generate context-aware high recall relevant candidates.

5.1.1.2 Current Approach

We chose Cosine similarity for candidate generation simply for the sake of simplicity as the project was specific to academic environment and was not part of any state-of-the-art production model. Also, time to run and execute the model to generate candidate is very quick which is helpful in web application used for demo. At the end, all candidates are pooled into a set and subsequently scored by the ranking system.

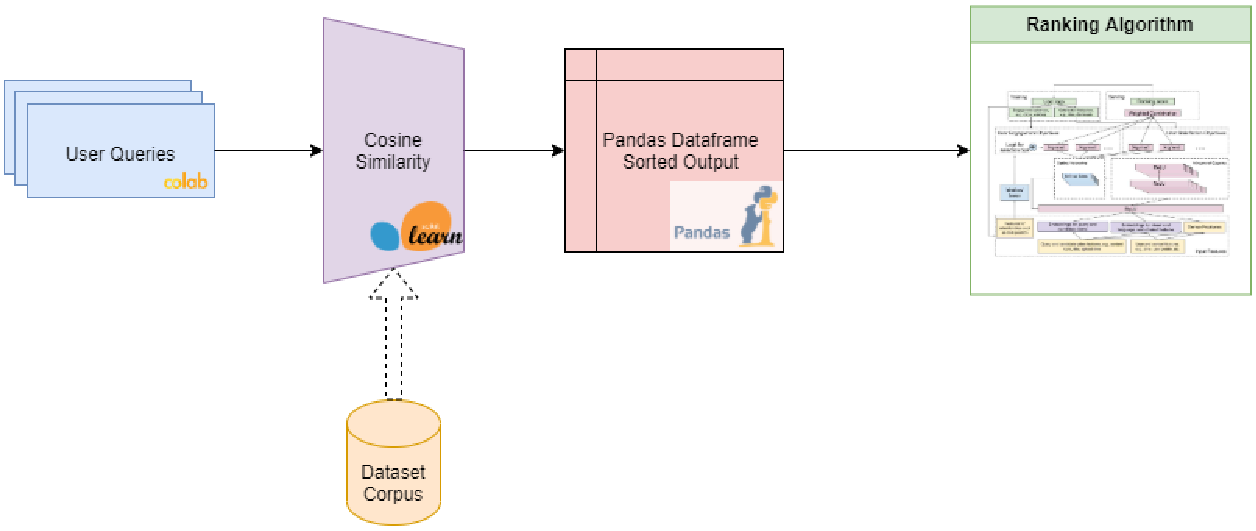
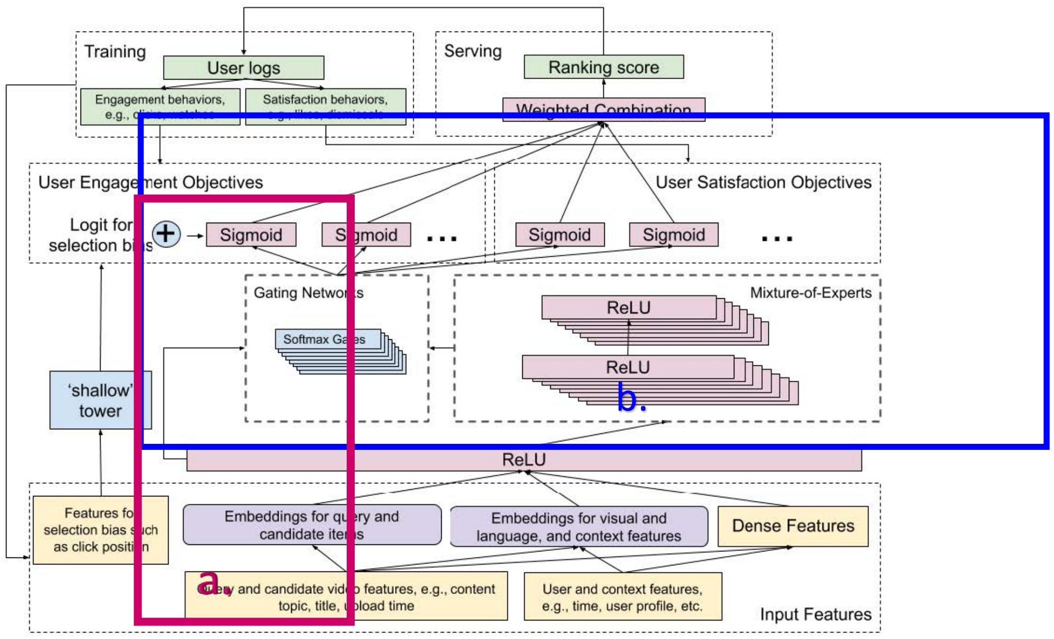


Figure 6: Generate Candidate List Using Cosine Similarity

5.1.2 Ranking Generation

Since the primary purpose of the project is to create a recommendation engine for a web application for sorting videos using MMoE as a ranking algorithm, hence, we are implementing MMoE (Multi gate Mixture of Experts) as the ranking algorithm. MMoE address and eliminate the challenges of scalability and implicit bias as we will see below.



Training
User logs
Engagement behaviors, e.g., clicks, watches
Satisfaction behaviors, e.g., likes, dismissals
Serving
Ranking score
Weighted Combination
User Engagement Objectives
Logit for selection bias
Sigmoid
Sigmoid
User Satisfaction Objectives
Sigmoid
Sigmoid
Gating Networks
Softmax Gates
Mixture-of-Experts
ReLU
ReLU
b.
'shallow' tower
ReLU
Features for selection bias such as click position
Embeddings for query and candidate items
Embeddings for visual and language, and context features
Dense Features
a.
Query and candidate video features, e.g., content topic, title, upload time
User and context features, e.g., time, user profile, etc.
Input Features

Figure 7: Multi gate mixture of experts Model architecture. (a) Shallow Tower: It is used to handle Implicit biad. (b) Mixture of experts managed by a gating using two objective functions (Engagement and Satisfaction)

5.1.2.1 MMoE Overview

MMoE structure is a combination of Multi-Layer Perceptrons followed by ReLU activations. There are experts in the MMoE layer which each of them is to learn a different feature of the input data.

The output of the Mixture of Experts (MoE) layer goes to a Gating Network. The output of the Gating Networks and the output of the shared hidden layer are inputs for the objective functions such as engagement and satisfaction. A sigmoid activation function represents each objective function.

Since users can have different types of behaviors towards recommended items, our ranking system can support multiple objectives. Each objective is to predict one type of user behavior related to user utility. The objectives are separate into two categories: engagement objectives and satisfaction objectives.

- Engagement objectives capture user behaviors such as clicks and watches. The prediction of these behaviors can be formulated into two types of tasks: binary classification task for behaviors such as clicks, and regression task for behaviors related to time spent.
- Similarly, for satisfaction objectives, the prediction of behaviors can be related to user satisfactions into either binary classification task or regression task. For example, behavior such as clicking like for a video is formulated as a binary classification task, and behavior such as rating is formulated as regression task.

For binary classification tasks, we are computing cross entropy loss. And for regression tasks, we compute squared loss.

5.1.2.2 Shallow Tower Overview

We are using implicit feedback data for training because explicit feedback data which are ideal for training are not available or are expensive, might not be ideal as there is a usual bias in this data which can increase with feedback. To manage the bias, a shallow tower is introduced into the model architecture. It is addressing the challenge as below:

- The shallow tower is trained using features that contribute to the bias like position of the recommendation and tries to predict whether there is a bias component involved in the current instance.
- It is removing the selection bias, takes input related to the selection bias, e.g., ranking order decided by the current system, and outputs a scalar serving as a bias term to the final prediction of the main model.
- The Shallow tower factorizes the label in training data in two parts the unbiased user utility learned from the main model, and the estimated propensity score learned from the shallow tower.

5.2 TFX Pipeline and Web Application Architecture

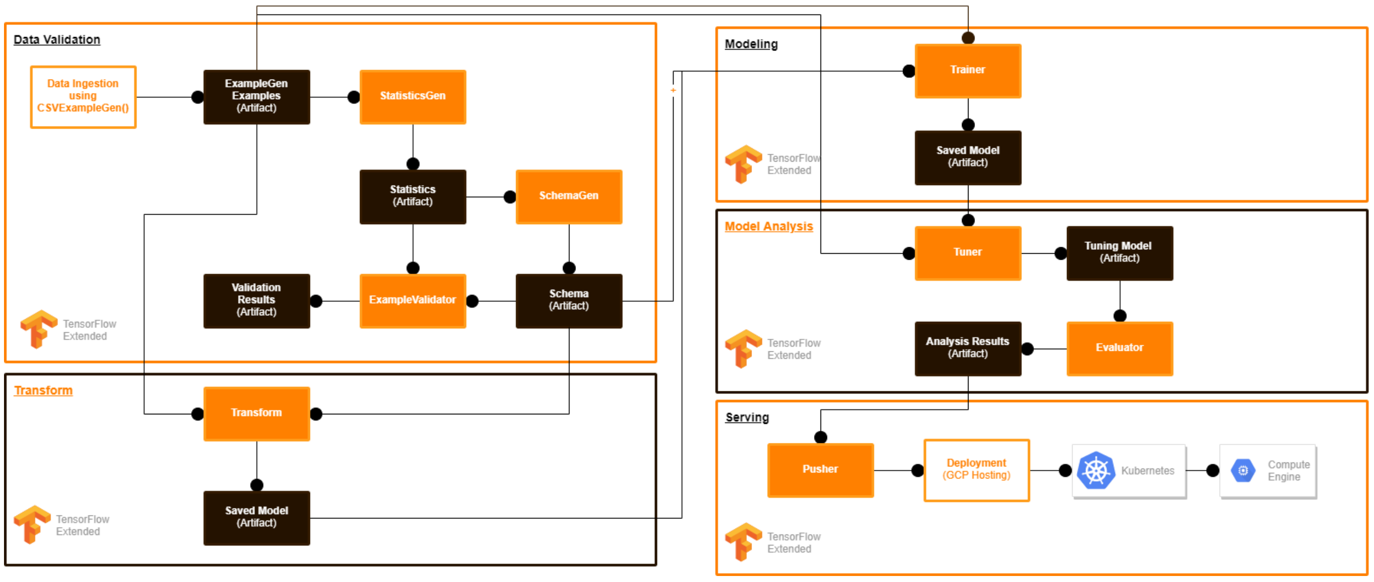
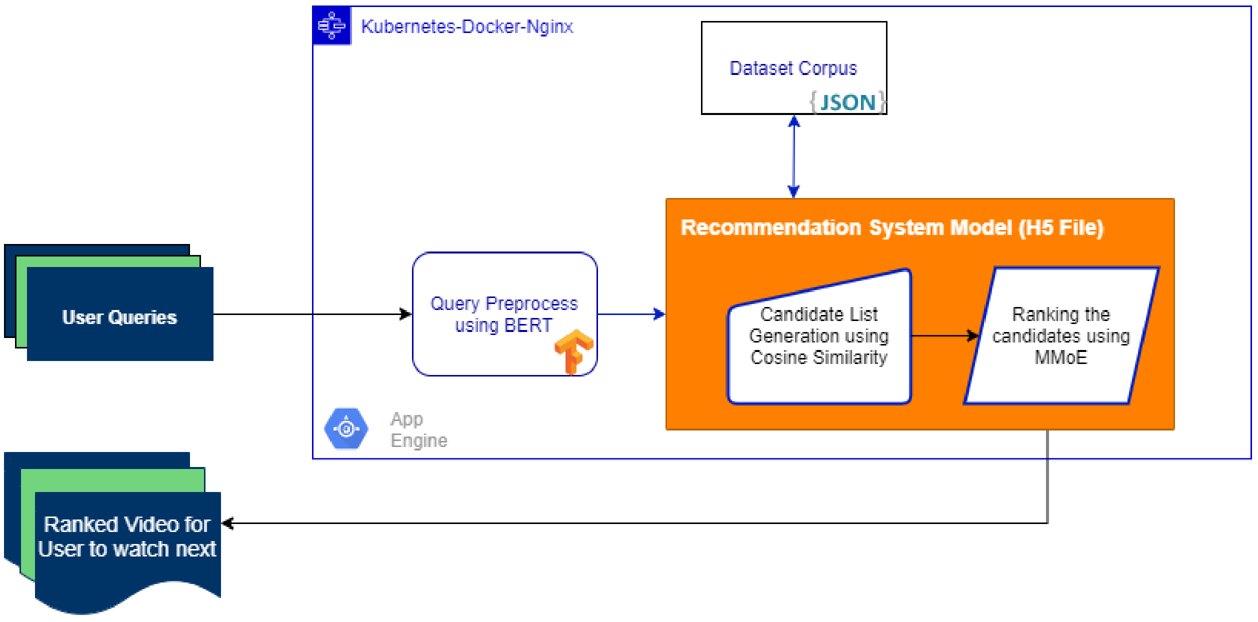


Figure 8: TFX Deployment Architecture to GCP using interactive Context

The colab model can be managed using TFX pipeline. Following are the list of standard components that we are using as Interactive Context in TFX pipeline:

- ExampleGen
- StatisticsGen
- SchemaGen
- ExampleValidator
- Transform
- Trainer
- Tuner
- Evaluator
- Pusher

We are using a custom component for deploying the model in Kubernetes and docker for the web application. Below is the use case for web application for user.



Kubernetes-Docker-Nginx
Dataset Corpus
{JSON}
Recommendation System Model (H5 File)
User Queries
Query Preprocess using BERT
Candidate List Generation using Cosine Similarity
Ranking the candidates using MMoE
App Engine
Ranked Video for User to watch next

Figure 9: Web Application Architecture

The TFX pipeline deploys the model directly on a Kubernetes-Nginx web application engine. Following are the sequence of events happening at web application:

- The Web application takes user queries as input
- The application preprocesses the user queries using a BERT model
- The recommendation engine scans through the dataset corpus and generates the candidate list
- The recommendation engine then from the ranking algorithm selects the top 50 ranked videos based on various features
- The user gets the list of next recommended videos to watch

6 EXPERIMENTATIONS

6.1 THE CHALLENGES OF RECOMMENDATION SYSTEM

The challenges of Recommendation system as stated earlier:

- There are often different and sometimes conflicting objectives which we want to optimize for. For example, we may want to recommend videos that users rate highly and share with their friends, in addition to watching.
- There is often implicit bias in the system. For example, a user might have clicked and watched a video simply because it was being ranked high, not because it was the one that the user liked the most. Therefore, models trained using data generated from the current system will be biased, causing a feedback loop effect.

6.2 ADDRESSING MULTI OBJECTIVE FUNCTIONS

There are two common objective functions that may create a conflict and should be optimized:

- Engagement Objectives: These objectives can be measured using dataset on clicks, time spent of the user while watching the recommended video, etc.
- Satisfaction Objective: These objectives can be measured by dataset in likes, shares, comments, rating, etc.

Both these objectives contain:

- binary classification tasks (click or not, like or not, etc.)
- regression tasks (time spent, rating given, etc.)

6.3 REMOVAL OF IMPLICIT BIAS

For training the model, the used dataset contains some implicit bias:

The reason is because a user historically might have clicked and watched a video simply because it was being ranked high, not because it was the one that the user liked the most.

So, if the model is trained using such data, it will produce biased non-optimal recommendations which the user might not like.

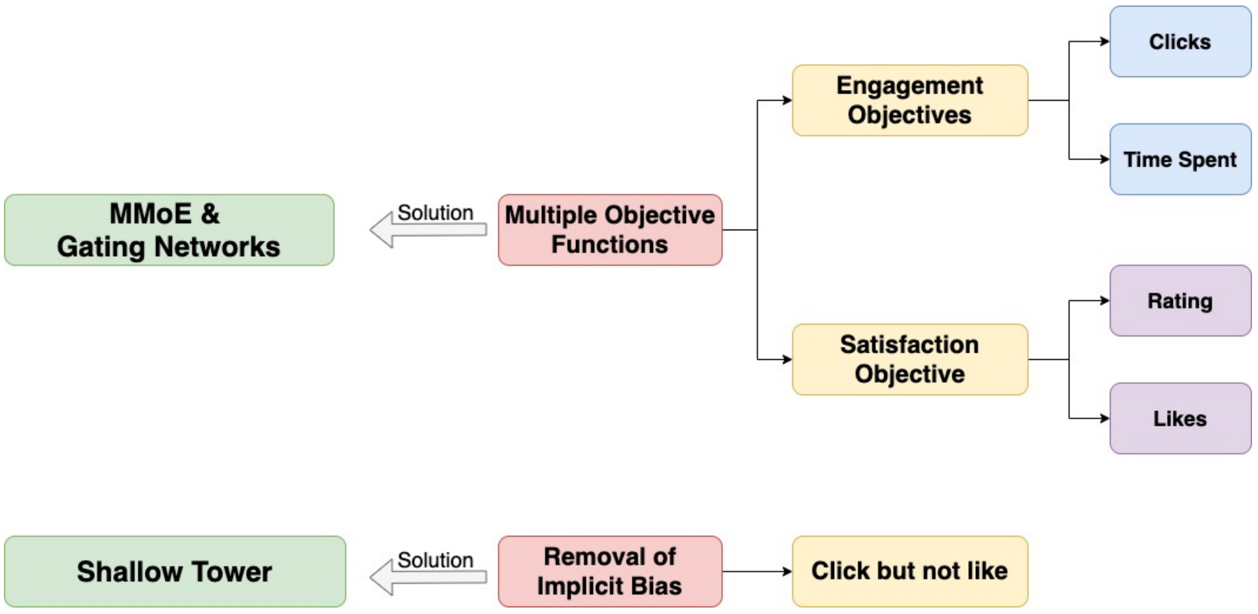


Figure 10: Experimentations to Address the Challenges of Recommendation Systems

7 Conclusion

In this project, we are trying to replicate the results of a Recommendation Engine using MMoE as ranking generation. We try to address the challenges in designing and developing industrial recommendation systems, especially ranking systems. These challenges include the presence of multiple competing ranking objectives, as well as implicit selection biases in user feedback.

To tackle these challenges, we have used a large-scale multi-objective ranking system and applied it to the problem of recommending what video to watch next. To efficiently optimize multiple ranking objectives, we extended Multi-gate Mixture-of-Experts model architecture to utilize soft-parameter sharing.

Below are the results from tensor board for the training of recommendation engine and we can see with each epoch the loss in context of user rating is reducing. What it means is that Users are giving more positive rating* for the model.

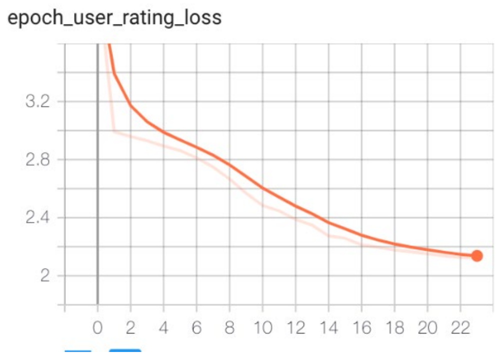


Figure 11: Training results and reduction in loss in user rating per epoch

*We are simulating the user experience as real-world data was unavailable.

8 References

Below is the list of references used for this project:

1. Zhao, Z., Hong, L., Wei, L., Chen, J., Nath, A., Andrews, S., . . . Chi, E. (2019). Recommending what video to watch next. Proceedings of the 13th ACM Conference on Recommender Systems. doi:10.1145/3298689.3346997

2. Ma, J., Zhao, Z., Yi, X., Chen, J., Hong, L., &amp; Chi, E. H. (2018). Modeling Task Relationships in Multi-task Learning with Multi-Gate-Mixture-of-Experts. Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery &amp; Data Mining. doi:10.1145/3219819.3220007
3. Bhatia, S. (2020, February 22). A Multitask Ranking System: How YouTube recommends the Next Videos. Retrieved December 11, 2020, from https://medium.com/@bhatia.suneet/a-multitask-ranking-system-how-youtube-recommends-the-next-videos-a23a63476073
4. Mitchell, J. (2017). Trending YouTube Video Statistics. Retrieved 2019, from https://www.kaggle.com/datasnaek/youtube-new.
5. Jonathan Baxter et al. 2000. A model of inductive bias learning. J. Artif. Intell. Res.(JAIR) 12, 149-198 (2000), 3.
6. Rich Caruana. 1998. Multitask learning. In Learning to learn. Springer, 95–133.
7. Sebastian Ruder. 2017. An overview of multi-task learning in deep neural networks. arXiv preprint arXiv:1706.05098 (2017).
8. R Caruna. 1993. Multitask learning: A knowledge-based source of inductive bias. In Machine Learning: Proceedings of the Tenth International Conference. 41–48.
9. Geoffrey Hinton, Oriol Vinyals, and JeDean. 2015. Distilling the knowledge in a neural network. arXiv preprint arXiv:1503.02531 (2015).
10. David Eigen, Marc'Aurelio Ranzato, and Ilya Sutskever. 2013. Learning factored representations in a deep mixture of experts. arXiv:1312.4314 (2013).
11. Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Georey Hinton, and Jeff Dean. 2017. Outrageously large neural networks: The sparsely gated mixture-of-experts layer. arXiv preprint arXiv:1701.06538 (2017).
12. Chrisantha Fernando, Dylan Banarse, Charles Blundell, . . . Daan Wierstra. 2017. Pathnet: Evolution channels gradient descent in super neural networks. arXiv preprint arXiv:1701.08734 (2017).
13. Jeffrey Dean, Greg Corrado, Rajat Monga, . . . , Quoc V Le, et al. 2012. Large scale distributed deep networks. In Advances in neural information processing systems. 1223–1231.
14. Trapit Bansal, David Belanger, and Andrew McCallum. 2016. Ask the gru: Multitask learning for deep text recommendations. In Proceedings of the 10th ACM Conference on Recommender Systems. ACM, 107–114.
15. Melvin Johnson, Mike Schuster, Quoc V Le, . . . , Greg Corrado, et al. 2016. Google's multilingual neural machine translation system: enabling zeroshot translation. arXiv preprint arXiv:1611.04558 (2016)
16. Xia Ning and George Karypis. 2010. Multi-task learning for recommender system. In Proceedings of 2nd Asian Conference on Machine Learning. 269–284.
17. Zhe Zhao, Zhiyuan Cheng, Lichan Hong, and EdHChi. 2015. Improving user topic interest profiles by behavior factorization. In Proceedings of the 24th International Conference on World Wide Web. International World Wide Web Conferences Steering Committee, 1406–1416.