Henry Braid, Diana Estrada, Jack Elliott, Jack Stapholz
COM SCI 145 - Data Mining
Professor Sun
4 June 2025

Team JACKS: Data Mining Project Final Report

Introduction and Team Information

Recommendation systems are useful tools for websites to sell and showcase their products to the proper users. We experimented with different model types, including the baseline recommenders, content-based filtering, sequence-based models, and graph-based models. Each approach handled the data differently, whether through static features, sequential user behavior, or graph structures. By comparing performance across models, we were able to see how different representations and objectives impacted performance and average revenue output.

Methodology Overview

Content-Based Recommenders:

The gradient-boosted model implemented used XGBoost's XGBClassifier. The model was hyperparameter optimized and used these best values: 100 estimators, 0.3 learning rate, max depth of 6, and minimum child weight of 1. 5 rounds of early stopping were used to prevent overfitting, however it was never invoked due to the high performance of the model. The model was built to predict the highest probability for the "relevance" feature. These results then constructed the top-k recommendations.

The k-Nearest Neighbors model makes use of the scikit-learn package for modeling, normalization, encoding, and preprocessing. Our implementation allows for choosing distance metrics and setting a range of k-values to be validated through a multi-fold cross-validation process.

The Decision Tree model was implemented using scikit-learn's DecisionTreeClassifier to predict item relevance for each user-item pair. The model generates recommendations by predicting relevance probabilities for all possible user-item combinations, filtering out previously seen items, and ranking the top-k items by predicted relevance score. While performing at par with baseline recommenders, the Decision Tree's interpretable nature provides valuable insights into which user and item features most strongly influence recommendation decisions.

Another model we used is Logistic Regression. We framed recommendation as a binary relevance prediction using a Logistic Regression model classifier with an ElasticNet Penalty. We started by merging the interaction logs and attribute tables to engineer features such as price–income ratios, quantile-binned price ranges, segment–category and one-hot category flags, user activity and item popularity. All features in the set flow through a column transformer which one-hot encodes the categorical data and standardizes the numerical data with a standard scaler. To reduce dimensionality, we used ElasticNet with an emphasis on L1-based pruning to discard low-importance features on the decision boundary. We then employ a 3-fold GridSearchCV to ultimately refine everything in the model.

Sequence-Based Recommenders:

The RNN model was built using PyTorch's framework. The model created was titled as a “Revenue RNN”since it aimed to predict the highest revenue based on user and item history. Sequence data was constructed to properly capture the users' history and be passed through sequentially to the RNN. The model was optimized to use: 128 hidden dimension sizes, 3 layers, dropout of 0.0, and a learning rate of 0.001.

Our Long Short-Term Memory recommender was implemented using the tensorflow.keras package for the main model architecture and the scikit-learn package for supplementary components, such as categorical variable encoding and data preprocessing pipelines. Additionally, the model allows for the use of different lstm units, dropout rates, learning rates, batch sizes, and feature selection dimensions. In our model, the chosen parameters were 128, 0.4, 0.0001, 32, and 15 respectively. Additional aspects added to the model to increase predictive ability were the choice of using bi-directional LSTM, dropout, and early stopping.

Another implementation we used for sequence-based recommenders was an Auto-Regressive model. Our model specifically combines a second-order auto-regressive model combined with an L2 normalization protocol. In this model, interaction histories are tokenized into length-2 n-grams (n = 2) and are smoothed with Laplace additive smoothing to best estimate next-item probabilities. Additionally, we engineered multiple interaction features, including price, income ratios, category preferences, and user engagement, by merging historical logs with user and item attributes. Features in this model are preprocessed through a column transformer pipeline where numerical variables are standardized and categorical features are one-hot encoded. After this, we applied an L2-based pruning method to refine the feature set.

Graph-Based Recommenders:

The constructed GCN is performed by reading a graph of user and item nodes whose links were weighted by their respective relevance values. The GCN has two convolution layers with a ReLU activation inserted in between. This somewhat simple design was believed to work well by capturing the proper relationships between the data. The model architecture used a common dimension of 64, a hidden dimension of 64, and an output dimension of 32.

Experimental Setup and Implementation

Data Preprocessing is an important step in ensuring data quality for training and prediction. Each model utilized a similar data preprocessing pipeline to ensure compatibility with different types of data representations, such as sequences or graphs. Creating new features is helpful based on the different types of models built. For the gradient boost, RNN, and GCN models, augmented features included “User Average Price” and “Price vs. User Mean.” These features help the models capture user-specific spending behavior, improving its ability to personalize predictions and detect deviations from typical patterns. The data is then normalized using StandardScalar and categorical features are encoded through one-hot encoding. Missing values in numerical features are replaced with the mean of their respective row and missing values in categorical features are replaced with their most frequent values.

The kNN model made use of a preprocessing pipeline that scales numerical features and encodes categorical ones while accounting for missing values. The LSTM model goes even further to ensure data quality by selecting the most informative features and sampling the data to prevent overfitting.

Training is an important step in the data modeling process. The models are trained after the data is preprocessed on user and item interaction logs. Each user and item has different attributes that the models train on in order to maximize revenue, predict relevance, or get the highest “score.” The content-based recommenders train purely on the logs and attributes to make their predictions. The sequence-based recommenders train on the same logs converted into sequences with timestamps imprinted. The sequence-based recommenders are trained on the instances of the sequences. The graph-based model, GCN, is trained on a user and item graph whose edge weights are determined by “relevance.” The node embeddings are used to determine a predicted revenue.

Each model needs to have the most optimal hyperparameters in order to achieve maximal performance. The model’s select hyperparameters through 5-fold cross-validation. On the first training call, the models fit on a GridSearchCV with 5 folds. The best hyperparameters are then carried over for the rest of the training iterations. This ensures that the models are cross-validated only once, and training can continue efficiently.

The models use different evaluation frameworks and metrics in order to minimize loss. For example, the gradient boost recommender is evaluated through binary cross-entropy on the predicted relevance probabilities. This loss penalizes confident but incorrect predictions more heavily, encouraging well-calibrated probability outputs. This is especially important in a gradient-boost recommendation system, where accurately ranking items by predicted relevance is more useful than just classifying them. For the RNN Recommender, the model is evaluated through Mean-Squared Error on the predicted revenue score and optimized with Adam Stochastic Gradient Descent. The score is calculated by manipulating the predicted relevance and incorporating price so that recommendations prioritize both relevance and economic impact. Using Mean-Squared Error allows the model to directly minimize the difference between predicted and actual revenue scores, aligning the objective with monetary outcomes. The GCN recommender computes predicted relevance scores for unseen user-item pairs using the dot product of their embeddings, adjusts the scores by item price to estimate expected revenue, and ranks items accordingly. By aligning predicted relevance with item price, it directly optimizes for expected revenue, leading to stronger performance in the recommendation tasks.

Results and Analysis

Figure 1:

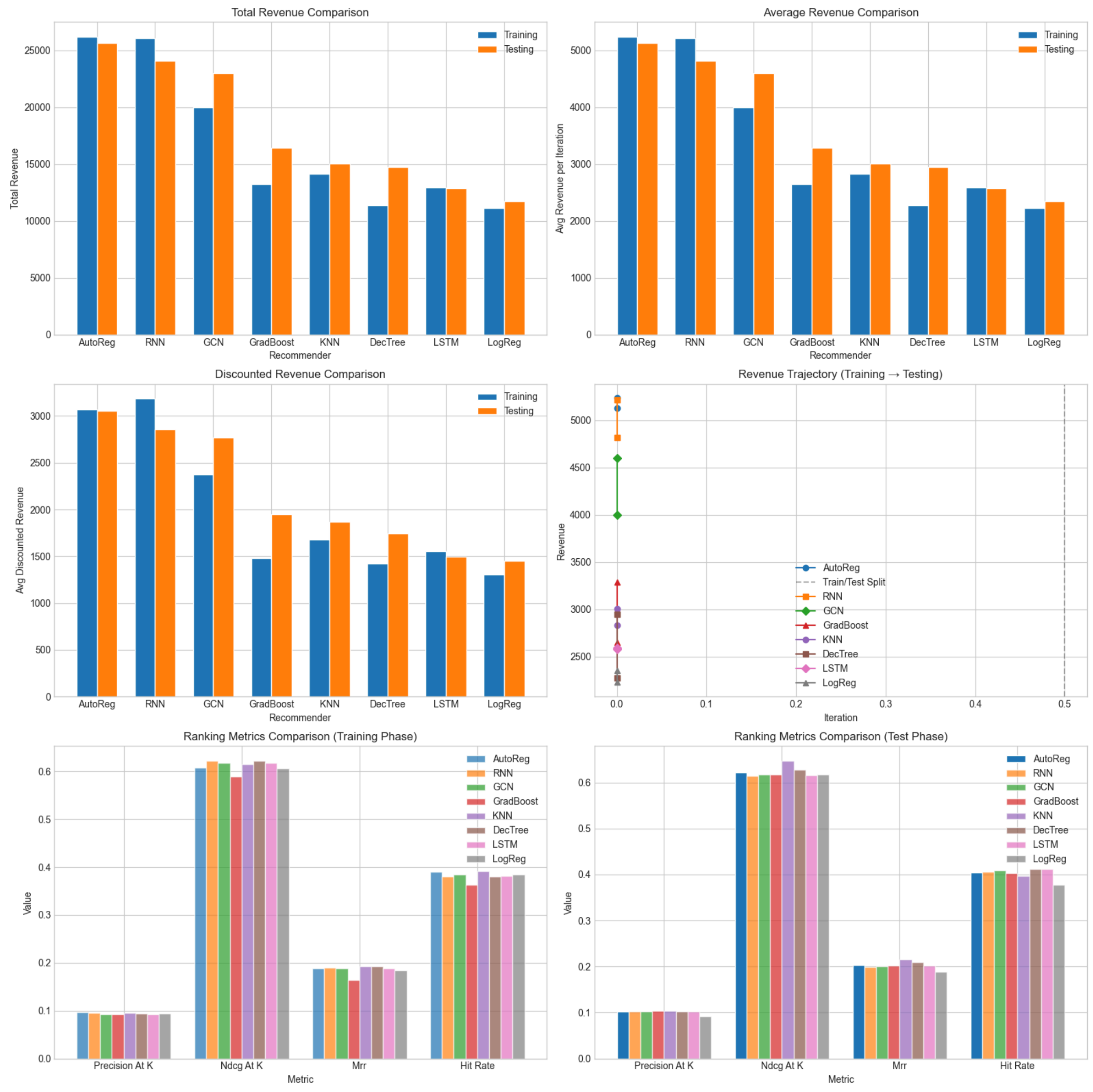
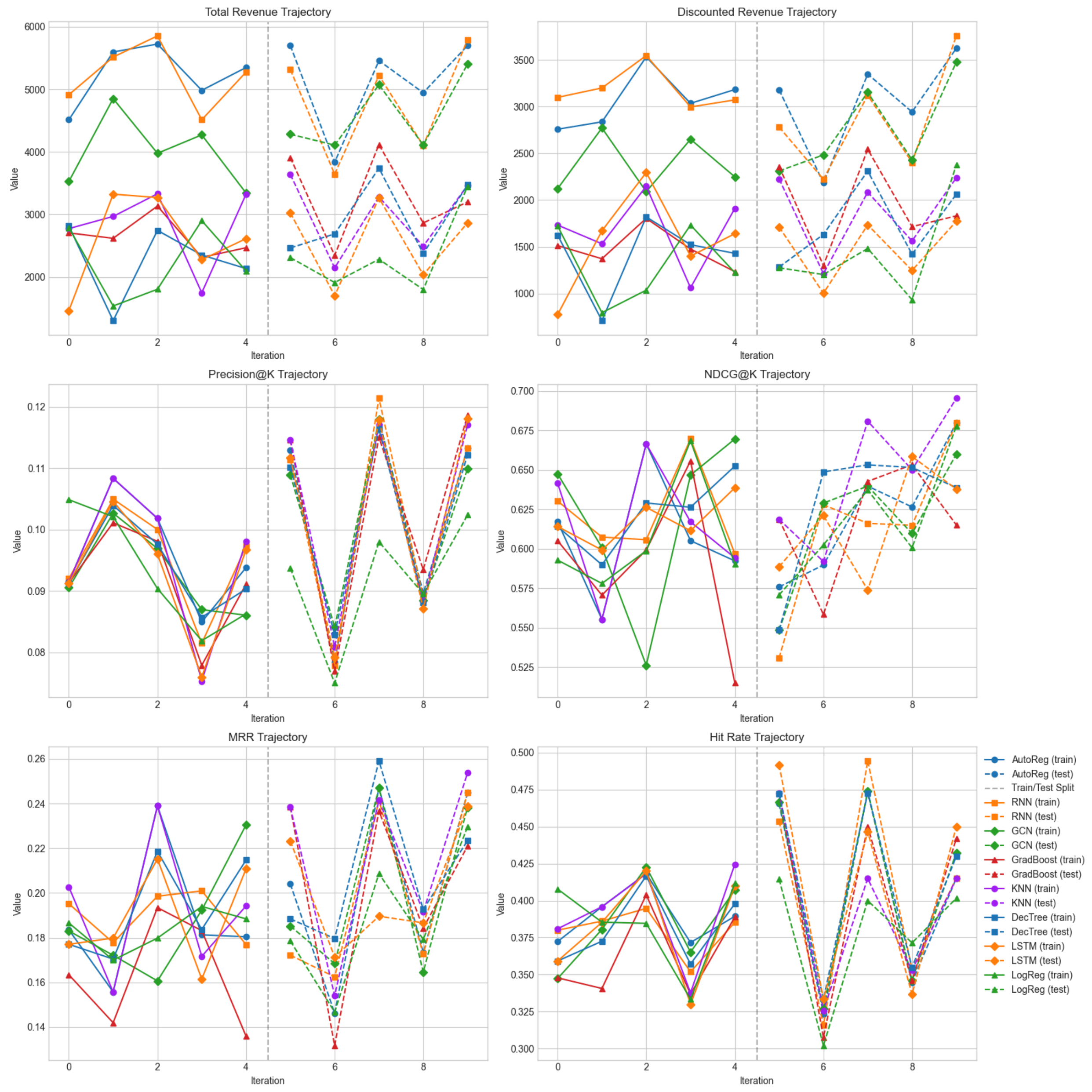


Figure 2:



The three content-based recommenders (logistic regression, decision tree, and gradient boosting) did not perform well compared to the sequence-based and graph-based recommenders. The best content-based recommender used gradient boosting to significantly improve testing performance. The logistic regression and decision tree models show promising results from testing to training, however when compared to the GCN, RNN, or AutoRegression recommendations they become insignificant.

The sequence-based recommenders dominate the charts with the high performance of both the RNN and the AutoRegression recommenders. The average discounted revenue gain increases from a best content-based value of $1900 to around $3000. This approximate $1100 increase insinuates that the sequence recommendation algorithms provide better recommendations than the purely content-based algorithm. The sequential models separate data into ordered-data that carry more relational data on user interactions and behavior. This modification of the data leads to larger revenue gains in both recommendation systems. Temporal patterns in user behavior are captured more efficiently which separates the sequence recommenders from other recommendation systems. User intent can be modeled more accurately which will lead to higher prediction performance.

The final recommendation system developed significantly outperforms the baseline models and runs the best compared to the rest of our implementations. Results can be seen in *Figure 3* of the Appendix.

Ablation Study:

Hyperparameter optimization is crucial to identifying the best models to use within the recommendation systems. In order to determine the best parameters to use, our models incorporated scikit-learn's grid search framework.

For example, the gradient boosting recommender was hyperparameter-optimized on the number of estimators, the maximum depth, the learning rate, and the minimum child weight. The key parameters here that determined most of the discrepancies were the max depth and the number of estimators. The optimized version includes 5 early stopping rounds, unlike the non-optimized recommender. In *Figure 4* (check Appendix), the optimized gradient boost recommender outperformed the non-optimized. The optimal model generalizes better to unseen test data which can be attributed to the early stopping rounds added that help mitigate overfitting. Early stopping is beneficial as it prevents the model from performing unnecessary iterations or separations that would increase complexity with little gain.

Error Analysis:

Some of our most successful recommenders were able to consistently yield strong performance with relatively low training complexity. For example, the AutoRegression could model sequential user behavior without overfitting, which made it both fast and reliable, especially compared to more complex models like RNNs or LSTMs, which were slower to train and more sensitive to hyperparameter settings. On the other hand, certain models underperformed due to structural or representational mismatches.

Models such as kNN, Logistic Regression, and Decision Trees that lack the ability to retain the sequence structure of the data set and therefore struggled to provide better revenue than randomly recommending items to users. These models treated interactions as independent, whereas in the real world these relationships are valuable sources of information for prediction.

The results from our models suggest that a more streamlined recommender with carefully thought out feature engineering outperformed more complex and convoluted ones. Despite having multiple layers and directions, our LSTM model failed to match the performance of RNN and AutoRegression.

Final Strategy and Optimization Path

When deciding on which model to base our final recommendation system on, it became clear that the content-based recommenders would not be sufficient. These models, while on-par with the baseline models, did not perform exceedingly well from training to testing and they did not provide sufficient revenue leaps relatively as well. Of the three content-based models, the best (Gradient Boost) reached about $1,900 in discounted revenue. This is a nice leap but we believed that this could be extended even further. In comparison, the baseline models reached nearly $1600-1700 in average discounted revenue.

The sequence-based recommenders resulted in the best prediction results thus far. The RNN and AutoRegression models performed well compared to the content-based recommenders. The RNN, while performing well, struggled with training time due to the size of the layers and the volume of data. The LSTM struggled to reach high performance leading to us valuing the AutoRegressive model the most. The training time was quick and efficient and the model yielded large revenue gains.

When we moved on to the graph-based recommendations, we saw some promise as well. We originally found that the Graph Convolutional Network (GCN) recommender provided the best results consistently on discounted revenue. When submitting to the leaderboard, however, the GCN Recommender did not perform as well as anticipated and received a lower score. A possible pitfall of the GCN approach is how the edge weights are defined. The potentially overly simplistic approach of just providing the “relevance” values could lead to the model potentially overfitting and skipping over other key relationships. Thus, we decided to follow through and optimize the AutoRegression recommender.

The AutoRegression recommender ended up providing the most benefit because of its ability to model user behavior as a sequence, rather than as isolated interactions. By leveraging a user’s past actions to predict future preferences, it captures evolving trends and context that static models like GCN might miss. This sequential structure makes the recommendations feel more personalized and adaptive, which likely contributed to its strong performance in our final evaluations.

For our auto-regressive model, we performed a joint grid search over both sequence-model and classifier hyperparameters, optimizing for held-out revenue. Key tuning dimensions are the following. An N-gram Order (n) tested orders $\{1,2,3\}$ to balance context depth vs. data sparsity. We employed a Laplace Smoothing (α) grid over $\{0.1,0.5,1.0\}$ to control the strength of additive smoothing in the next-item probability estimates. For our feature-selection threshold, we varied the L2-penalty threshold over the median and 75th percentile of feature importances to find the optimal sparsity level. Our CV strategy employed time-aware 5-fold splits, ensuring each fold preserves the temporal order of user interactions, thereby avoiding peeking into future events. Our scoring metric was a custom revenue scorer that weights predicted relevance by actual revenue impact, guiding selection, and penalizing overly complex models via an L2 coefficient penalty. Each candidate configuration retrains the whole pipeline—including feature construction, scaling, one-hot encoding, AR probability computation, and L2-based pruning—before scoring on the validation fold.

Conclusion & Course Connection

This project provided a hands-on opportunity to apply a wide range of data mining techniques from the course to a real-world recommendation task. Our objective was to develop models that could predict user-item relevance in a way that prioritized high-revenue items and potentially maximize revenue. By starting with the baseline models and working through content-based filtering, sequentially modelling, and graph-based construction we were able to better capture user preferences, item attributes, and temporal behavior.

Many of the main concepts within the course directly affected our thought process and execution. For example, we used feature engineering and preprocessing pipelines to create meaningful user-item interaction features. We additionally applied supervised learning with logistic regression, decision trees, and gradient boosting. We practiced model selection as well by using cross-validation and grid search for model selection. Our final recommendation algorithm based on AutoRegression strongly embodied time-aware modeling and showcased how ordered data can enhance performance and revenue.

Main takeaways from this project include the importance of aligning model objectives with the evaluation metric. For example, the evaluation must properly handle the trade-offs between interpretability and predictive power across model types. We also learned how the types of data representation such as pure data logs, extracted sequences, or graphs significantly impact model performance and generalization. Improvements can still be made, and some possible areas to explore include improving the graph construction process for the GCN recommender, or exploring deeper sequential models like Transformers. This project deepened our understanding of recommendation systems and highlighted the practical challenges and design choices involved in building models that relate to real-world impact.

Separation of Work:

Henry worked on various feature engineering methods in each checkpoint, by including a unique preprocessing pipeline that would be passed through to each of his models. He built the Gradient Boost model, the RNN Recommender, and the Graph Convolutional Network Recommender. Jack S. was responsible for the k-Nearest Neighbors model as well as the Long Short-Term Memory model. Each model has its own data preprocessing pipeline as well. Diana built the Decision Tree Recommender, which performed at par with the base recommenders.Jack E. refined the Logistic and Autoregressive models with an emphasis on ElasticNet and L2 regression respectively. These techniques long with their respective normalizations helped us achieve our best results for this project.

Appendix:

Figure 1:

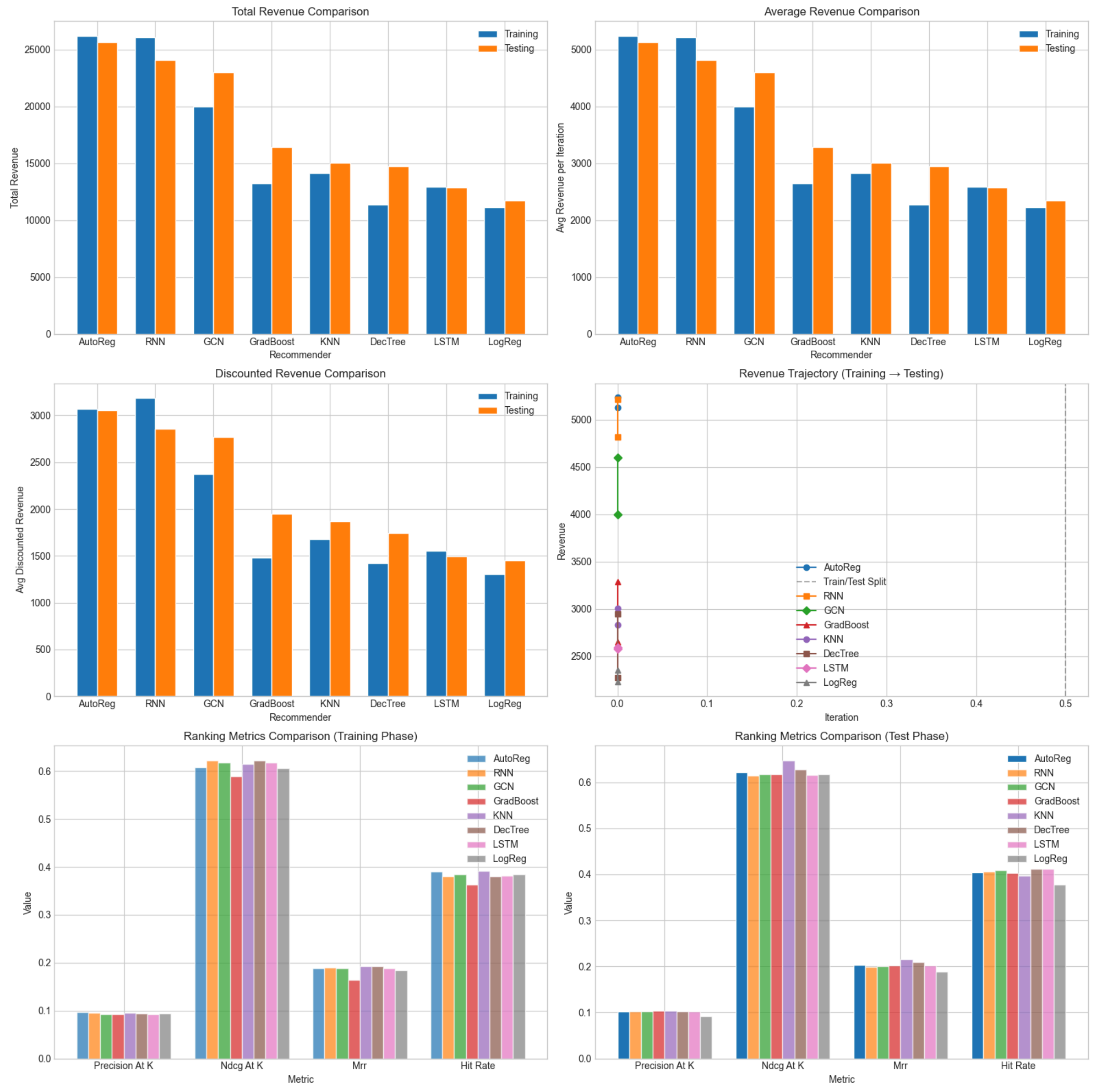
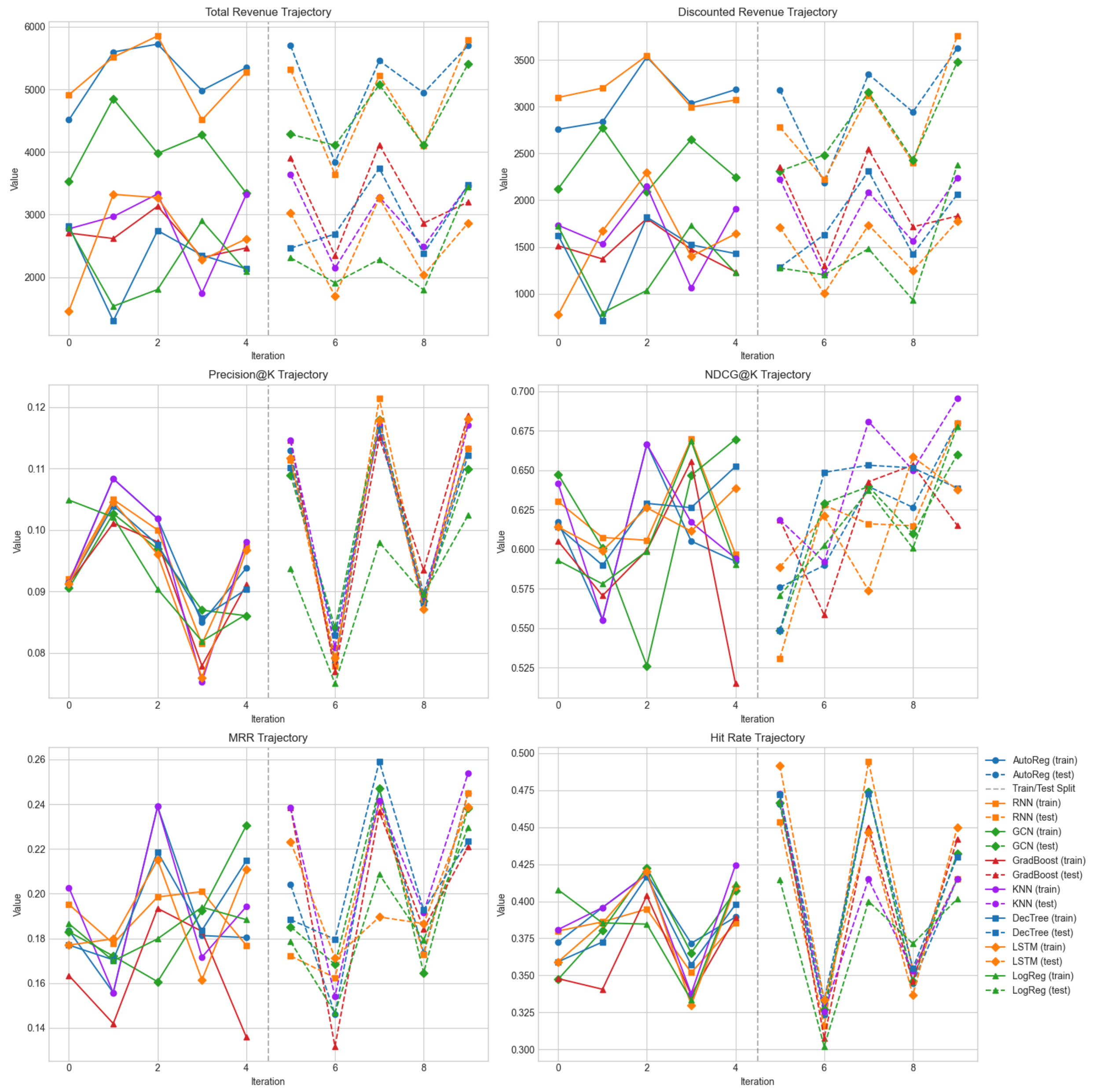


Figure 2:



Total Revenue Trajectory
Discounted Revenue Trajectory
Precision@K Trajectory
NDCG@K Trajectory
MRR Trajectory
Hit Rate Trajectory
Value
Iteration
AutoReg (train)
AutoReg (test)
Train/Test Split
RNN (train)
RNN (test)
GCN (train)
GCN (test)
GradBoost (train)
GradBoost (test)
KNN (train)
KNN (test)
DecTree (train)
DecTree (test)
LSTM (train)
LSTM (test)
LogReg (train)
LogReg (test)

Figure 3:

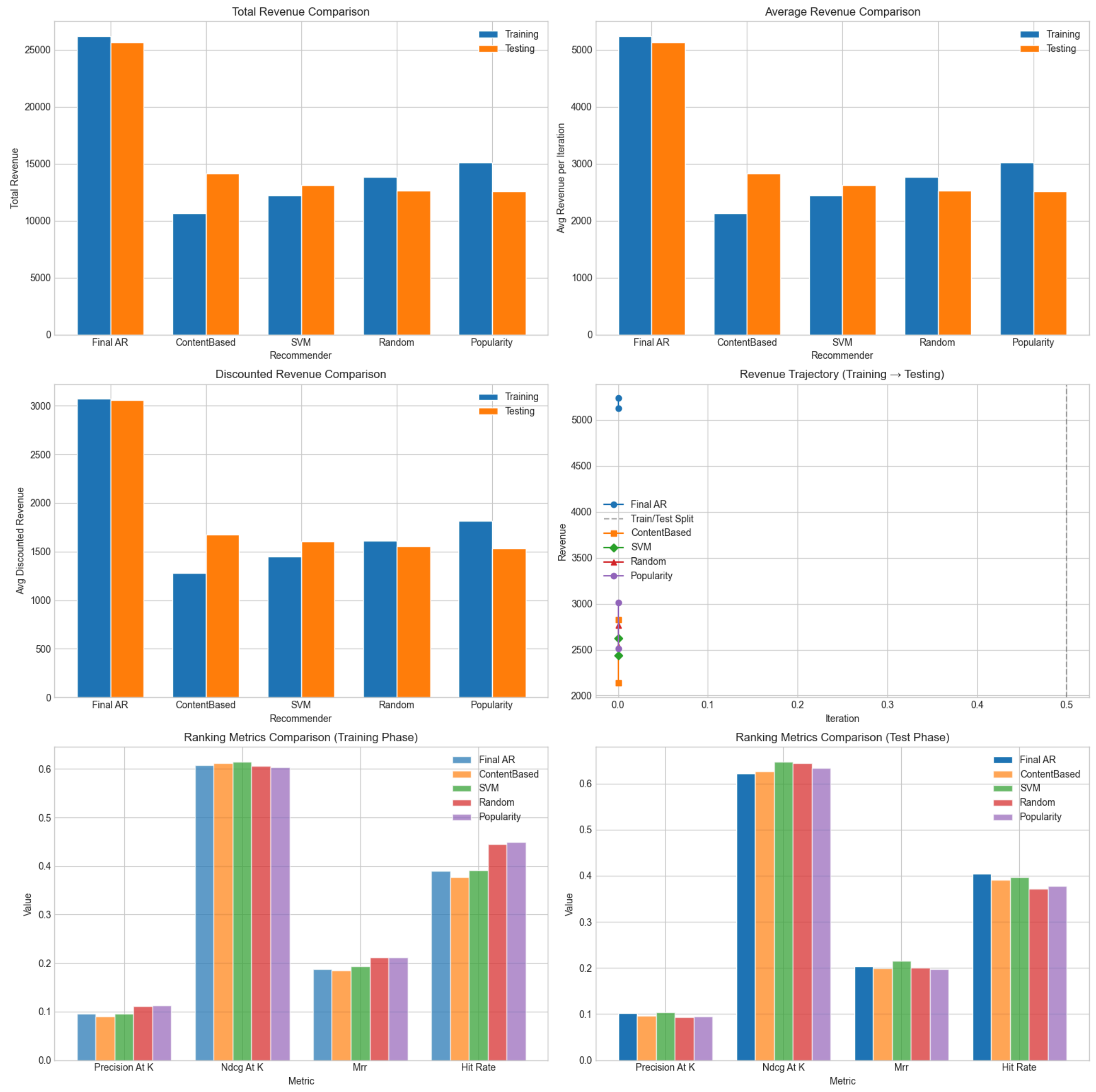
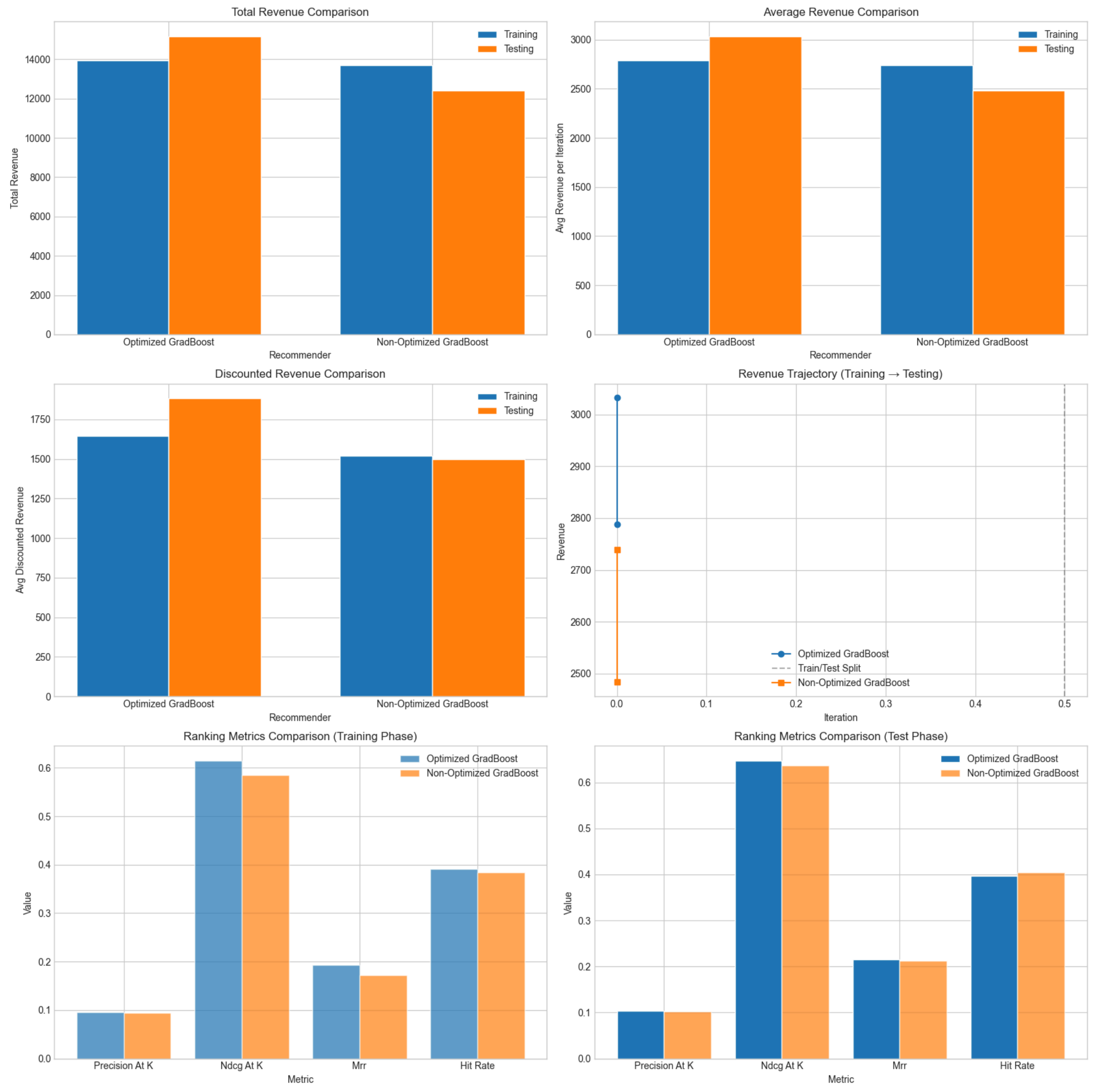

Total Revenue Comparison
Average Revenue Comparison
Discounted Revenue Comparison
Revenue Trajectory (Training → Testing)
Ranking Metrics Comparison (Training Phase)
Ranking Metrics Comparison (Test Phase)
Training
Testing
Total Revenue
Avg Revenue per Iteration
Avg Discounted Revenue
Revenue
Iteration
Recommender
Final AR
ContentBased
SVM
Random
Popularity
Train/Test Split
Value
Metric
Precision At K
Ndcg At K
Mrr
Hit Rate

Figure 4:



Total Revenue Comparison
Training
Testing
Total Revenue
Optimized GradBoost
Non-Optimized GradBoost
Recommender
Average Revenue Comparison
Avg Revenue per Iteration
Discounted Revenue Comparison
Avg Discounted Revenue
Revenue Trajectory (Training → Testing)
Revenue
Iteration
Train/Test Split
Ranking Metrics Comparison (Training Phase)
Ranking Metrics Comparison (Test Phase)
Value
Metric
Precision At K
Ndcg At K
Mrr
Hit Rate