Audience Recommender Systems for Best Buy Marketing
Fall Practicum 2022

Christopher Dae-Hyun Kim
ckim612@gatech.edu

Abstract

Recommendation Systems (RecSys) are commonly used for both predicting ratings of unknown items and learning to rank (LTR) (Aggarwal, 2016). Customer interaction data and audience scores were used to create the ideal ranking of audiences for each customer. Several LTR models using gradient boosted trees and neural networks were deployed with the neural network models vastly outperforming the gradient boost tree models. Top performing neural network model utilizing the ApproxNDCG loss function is reported with a normalized discounted cumulative gain at 3 (NDCG) score of 0.9234.

1 Introduction

Best Buy is a multinational American corporation specializing in retail of consumer electronics and appliances. Best Buy has generated approximately $52B in revenue in the fiscal year of 2022, operates approximately 1000 stores across the United States and Canada, and has approximately 100,000 employees.

2 Problem Statement

Best Buy customers are segmented into audiences that are in the market to purchase various produce types with many customers belonging to many audiences. Marketing communication to customers via email and notifications are organized by audience types and campaign objectives. Since many customers belong to many audiences, Best Buy does not want to inundate their customers by sending multiple marketing communications; Best Buy instead wishes to prioritize the top three relevant audience types for each customer to drive personalized marketing communication and customer conversions.

3 Methods

Best Buy's business objective can be formulated as a LTR problem using RecSys. RecSys can have two goals: rating prediction and ranking. RecSys are typically associated with rating determination and recommendation of unseen items as seen in the classical Netflix open competition. This project's objective is to rank relevant audiences for customers and not predict/recommend new audiences.

LTR requires scored candidates. As Best Buy has detailed in their practicum proposal, candidate generation and scoring information has already been provided by audience scores alongside a second data set that contains pertinent customer information[1].

LTR also assumes that candidates are also ranked in relevancy prior to model training. That is, a true label of the ranking is already exists and is either explicitly or implicitly determined by humans. Ratings are an example of explicit data, as seen in the Netflix case, whereas clicks and browsing are examples of implicit data.

Finally, this project has trained several LTR models using several loss functions using NDCG as the evaluation metric. The loss functions and evaluation metric will be discussed extensively in Section 4.

4 Literature Review

As an overview, LTR is concerned with learning a function to determine relevancy of candidates and distinguished by formulation of loss functions. These consist of supervised and supervised tasks. There are four forms of LTR loss functions: Pointwise (Caruana et al., 1995), pairwise (Burges et al., 2005), listwise (Cao et al., 2007), and combinations of the aforementioned. Majority of the literature review and project usage is dedicated to pairwise and listwise loss functions.

4.1 Pairwise - RankNet

As the first pairwise LTR model ever published by Microsoft, RankNet (Burges et al., 2005) is the standard benchmark and foundation for LTR loss functions. It was originally used for search optimization for user queries and documents. For the intents of the project, we can extend the original framework of user queries and documents to customer and audience scores/interaction data [2].

The learning task is the classification of pairwise objects into the two distinct categories of correct rank and incorrect rank. As mentioned in Section 3, we assume an ideal ranked order of audience the customer belongs to based on their interaction history exists. We then generate all pairwise instances of audiences for each user. For each user, we would feed each audience from all pairwise instances into a neural network with two hidden layers and a linear activation function (Burges et al., 2005). Retrieved ranked order is consistent[3].

[1]Due to the NDA agreement, further information of the data provided by Best Buy will not be divulged other than the above information provided by Best Buy in their practicum proposal.

[2]Throughout the paper, we will refer the original paper's framework of user queries and documents to customer and customer audience respectively for the sake of the reader and project case. Equation

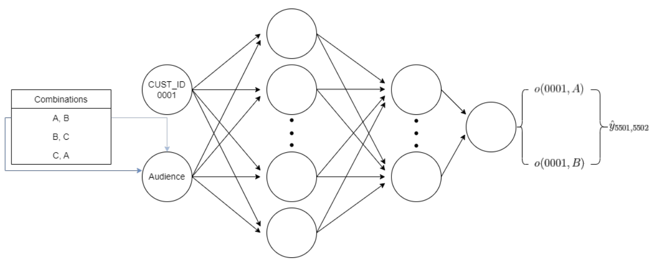

Figure 1: RankNet neural network diagram with first pairwise instance for the first customer being fed into the neural architecture.

For each output from the pairs, we determine the probability of the audience appearing before the other where the probabilistic output is defined as

$$\hat{y}_{ij} = P(i > j) = \frac{e^{o^i - o^j}}{1 + e^{o^i - o^j}} = \frac{1}{1 + e^{-(o^i - o^j)}} \tag{1}$$

$$o_i = f(c, a_i),\ o_j = f(c, a_j)$$

where o is the output from a function $f \to NN$ with customer c and audience a, and the cross entropy cost function is defined as

$$L(y_{ij}, \hat{y_{ij}} = -\sum_{i \neq j} y_{ij} \log \hat{y}_{ij} + (1 - y_{ij}) \log(\hat{y_{ij}}) \tag{2}$$

and optimized using the stochastic gradient descent algorithm (Burges et al., 2005)

$$\omega^{t+1} = \omega^t - \alpha \nabla_L = \omega^t - \alpha [\frac{\partial L}{\partial o_i}\frac{\partial o_i}{\partial w^t} + \frac{\partial L}{\partial o_j}\frac{\partial o_j}{\partial w^t}] \tag{3}$$

It is important to note the complexity of this framework. We can see the time complexity of the model is $O(m \cdot n^2)$ as we are iterating through all pairwise instances of n audiences per m customers (Cao et al., 2007). Another important note concerning model complexity is model training; we need the gradient of the cost with respect to the model output in order to calculate the gradient of the cost with respect to model parameters.

4.2 Listwise - ListNet

Seen as an improvement to its predecessor, ListNet is the first ever listwise LTR model published by Microsoft (Cao et al., 2007). As implied in the name, ListNet pertains to a set of customers where each customer has a list of audiences with a list of rankings associated with each audience. At a high level, ListNet finds the permutation probability distributions of the list of scores and uses their loss function to calculate the distance between the distributions.

Formally, given a set of customers $C = \{c^1, ..., c^m\}$, a list of their respective audiences $a^i = a^i_1, ..., a^i_n$, and the respective scores associated with each audience $y = y^i_1, ..., y^i_n$, the probability of permutation π given y is defined as

$$P(\pi) = \prod_{j=1}^{n} \frac{\phi(o_j)}{\sum_{k=j}^{n} \phi(o_k)} \tag{4}$$

where $\phi(o_j) = f(c^i, x^i_j)$ is an increasing and strictly positive scoring function where x^i_j is the feature vector for audience a^i_j. Since this extremely computationally expensive as there are $n!$ permutations, Cao et al.(2007) generalized the formula and proved that the top one probability of an audience is equal to the sum of permutation probabilities in which the respective audience is on the top of the ranked list of scores

$$P(\pi) = \prod_{j=1}^{n} \frac{\phi(o_j)}{\sum_{k=j}^{n} \phi(o_k)} \equiv P_y(j) = \frac{\phi(o_j)}{\sum_{k=1}^{n} \phi(o_k)} \tag{5}$$

Cao et al.(2007) also improved the time complexity to $O(n \cdot m)$ by considering all audiences of a customer as a list instead of iterating through pairwise instances of the audiences of a customer.

4.3 LambdaRank/MART/Loss

There are two main distinctions between LambdaRank/MART and its predecessors: LambdaMART uses Multiple Additive Regression Trees or gradient boosted trees as the underlying model structure and LambdaRank is a cost function directly optimizing for information retrieval (IR) measures (Burges, 2010), specifically normalized discounted cumulative gain (NDCG)[4][5].

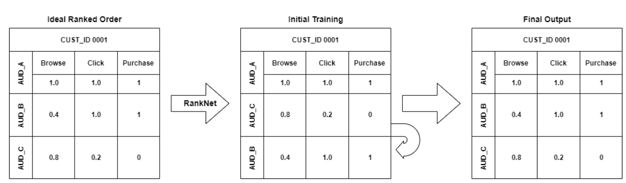

Figure 2: Example of LambdaRank training using illustrative and typical interaction data variables with an inversion seen in the middle table.

As seen in the RankNet and ListNet framework, cost functions are specifically minimizing the cross entropy loss and therefore wholly concerned with the classification learning task. Furthermore, RankNet updates the model on a pairwise basis. LambdaRank addresses these by introducing mini batch stochastic gradient descent and a new variable λ that introduces the gain function NDCG into the loss function so that the loss function is now wholly concerned with the ranked order instead of just pairs (Burges, 2010):

$$\frac{\partial L}{\partial w^t} = \lambda_{ij}(\frac{\partial o_i}{\partial w^t} - \frac{\partial o_j}{\partial w^t})$$

variables have also been adjusted to fit the project context.

[3]Using a simple example, suppose there exists document A, B, and C. All pairwise instances or combinations would be: {A,B}, {B,C}, and {C,A}. If A had a larger probability of being ranked higher than B and B had a larger probability of being ranked higher than C, ranked order would be A,B,C.

[4]We can use other IR measures, such as mean reciprocal rank and mean average precision.

[5]See Appendix for NDCG definition

$$\lambda_{ij} := \sigma(\frac{1}{2}(1 - O_{ij}) - \frac{1}{1 + e^{\sigma(o_i - o_j)}})$$

$$\lambda_{ij} = \frac{\partial L(o_i - o_j)}{\partial o_i} = \frac{-\sigma}{1 + e^{\sigma(o_i - o_j)}} \cdot |\Delta_{NDCG}|$$

$$\frac{\partial L}{\partial w^t} = \sum_i \lambda_i \frac{\partial o_i}{\partial w^t} \quad (6)$$

Δ_{NDCG} is the change in swapping the positions of candidate i and j and λ_i is difference of the sums of λ_{ij} when regarding candidate i and j. Furthermore, we can see that the new update function does not include the partial derivative with respect to the loss function as alluded to in Section 4.1.

Burges (2010) had prefaced that LambdaRank did not have an analytical solution as the model did not define the gradient as a gradient of the loss function itself, but that the model was empirically justified, Wang et al. (2018) at Google provided an analytical solution to LambdaRank and other LTR models by framing it as an Expectation-Maximization (EM) problem and treating the ranked list as a hidden variable where the E-step would estimate the distribution of the model outputs for the hidden variable and the M-step would update the model parameters using the loss functions.

5 Results

Data processing, feature engineering, and model training was coded using Python 3.8. LightGBM (Ke et al., 2017) and Tensorflow-Ranking (Abadi et al., 2016) were used to build the pairwise tree and listwise neural network structures respectively. Tensorflow-Ranking leverages the LambdaLoss framework for its implementation of LTR.

Dask (Dask Development Team, 2016) was used for exploratory data analysis. Customer interaction data was converted to NumPy (Harris et al., 2020) data formats and standardized as required in both tree and neural network models. The data was then converted to SVM-light (Joachims, 1998) format to be used in model training.

As mentioned in Section 3, true labels of the ranking can be explicitly or implicitly determined by humans. We have aggregated audience scores and standardized consumer interaction data to metrics to create the "true" audience ranking.

The training split consisted of 50% of the data while the testing split covered the rest. Two models for each tree and neural network models were trained per NDCG@k size. FLAML (Wang and Wu, 2019) was used for hyper parameter optimization and cross validation[6] for the LightGBM models. Azure Machine Learning Studio was used to train the Tensorflow-Ranking neural network models. NDCG at 1, 3, and 5 were used to evaluate the models since the problem statement required at best the top three audiences per customer. Neural network models were fully connected with 16 nodes in the hidden layer, and trained using a batch size of 16 over 1000 steps, learning rate of 0.01, and dropout rate of 0.80. Training data was shuffled per epoch. The two loss functions used were the ApproxNDCG loss and the standard pairwise cross entropy loss. Adagrad was the optimization algorithm.

[6]Cross validation scheme was K-Fold where $k = 5$.

k	**Learning rate** η	**L1** α	**L2** λ
1	0.18926	0.00868	0.32856
1	0.10000	0.00090	1.00000
3	0.15662	0.00098	0.00643
3	0.03735	0.00098	1.00000
5	0.14981	0.00098	0.10100
5	0.10000	0.00098	1.00000

Table 1: Hyper parameters for regularized and non regularized LambdaMART models at varying NDCG@k.

Model	**NDCG@1**	**NDCG@3**	**NDCG@5**
$\lambda_{MART_{reg}}$	0.8762	0.8044	0.7716
$\lambda_{MART_{nonreg}}$	0.8671	0.7985	0.7638
$\lambda_{Loss_{ApproxNDCG}}$	1.0000	0.9375	0.9289
$\lambda_{Loss_{CrossEntropy}}$	0.9893	0.9301	0.9280

Table 2: NDCG@k for various models using training split.

where k refers to consideration of position of ranked order, rel_i is the score relevance of audience, and $IDCG$ is the ideal discounted cumulative gain of the ideal or true ranked order.

Model	**NDCG@1**	**NDCG@3**	**NDCG@5**
$\lambda_{MART_{reg}}$	0.8524	0.7820	0.7489
$\lambda_{MART_{nonreg}}$	0.8412	0.7555	0.7207
$\lambda_{Loss_{ApproxNDCG}}$	0.9917	0.9234	0.9143
$\lambda_{Loss_{CrossEntropy}}$	0.9725	0.9111	0.9076

Table 3: NDCG@k for various models using test split.

6 Discussion

As we can see from Table 1 and Table 2, the neural network models using the LambdaLoss frameworks performed better than the tree models. It is also interesting to note that the LambdaLoss framework outperformed the MART models using the same cross entropy loss function. As we can see from the training results, there may be some concerns about model overfitting when taking a cursory look at Table 2 and 3. However, this concern can be mitigated by the methodology undertaken throughout this project; the training split was 50% of the data, dropout rate was set at 0.80, and the results from the test split indicated that the models learned from the appropriate signals.

A further informal investigation was also undertaken upon reaching the results from Section 5. One neural network model was trained using only one customer and tested on another unspecified number of customers with the same model specifications as seen in Section 5. The one neural network model demonstrated similar results as the neural network models reported.

The results of the reported models and the informal investigation implies that the signals or customer feature variables are most important in deploying the models and these signals can be appropriately learned. Feature importance cannot be shared due to NDA constraints but is available to Best Buy personnel upon request. Results of the project can be further tested by deploying other analytical methods on the results from the marketing campaign, such as A/B tests.

Future research can be dedicated to a more granular scope; that is, if Best Buy is interested in recommending product types that the customers have previously been in the market for or recommending unknown items based on the customers' previous purchases. Such endeavors would require different neural network models, such as sequential aware and context aware models.

7 Appendix

$$NDCG_k = \frac{DCG_k}{IDCG_k}$$

$$DCG_k = \sum_k \frac{2^{rel_i} - 1}{\log_2(i+1)}$$

References

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Gregory S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian J. Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Józefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Gordon Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul A. Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda B. Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2016. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *CoRR*, abs/1603.04467.

Charu C. Aggarwal. 2016. *Recommender systems: The Textbook*. Springer.

Chris Burges, Tal Shaked, Erin Renshaw, Ari Lazier, Matt Deeds, Nicole Hamilton, and Greg Hullender. 2005. Learning to rank using gradient descent. In *Proceedings of the 22nd International Conference on Machine Learning*, ICML '05, page 89–96, New York, NY, USA. Association for Computing Machinery.

Christopher J. C. Burges. 2010. From RankNet to LambdaRank to LambdaMART: An overview. Technical report, Microsoft Research.

Zhe Cao, Tao Qin, Tie-Yan Liu, Ming-Feng Tsai, and Hang Li. 2007. Learning to rank: From pairwise approach to listwise approach. volume 227, pages 129–136, 01.

Rich Caruana, Shumeet Baluja, and Tom Mitchell. 1995. Using the future to "sort out" the present: Rankprop and multitask learning for medical risk evaluation. In D. Touretzky, M.C. Mozer, and M. Hasselmo, editors, *Advances in Neural Information Processing Systems*, volume 8. MIT Press.

Dask Development Team, 2016. *Dask: Library for dynamic task scheduling*.

Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. 2020. Array programming with NumPy. *Nature*, 585(7825):357–362, September.

T. Joachims. 1998. Making large-scale svm learning practical. LS8-Report 24, Universität Dortmund, LS VIII-Report.

Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. 2017. Lightgbm: A highly efficient gradient boosting decision tree. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

Chi Wang and Qingyun Wu. 2019. FLO: fast and lightweight hyperparameter optimization for automl. *CoRR*, abs/1911.04706.

Xuanhui Wang, Cheng Li, Nadav Golbandi, Mike Bendersky, and Marc Najork. 2018. The lambdaloss framework for ranking metric optimization. In *Proceedings of The 27th ACM International Conference on Information and Knowledge Management (CIKM '18)*, pages 1313–1322.