UNIVERSITAT DE BARCELONA

FUNDAMENTAL PRINCIPLES OF DATA SCIENCE MASTER'S THESIS

Using Deep Learning Techniques in Click-Through Rate Prediction, Focusing on a Deep and Cross Network and an Analysis on the Use of Different Hashing Buckets

Author:
CELINE ODDING

Supervisor:
Dr. Jordi VITRIA

A thesis submitted in partial fulfillment of the requirements for the degree of MSc in Fundamental Principles of Data Science

in the

Facultat de Matemàtiques i Informàtica

June 30, 2023

UNIVERSITAT DE BARCELONA

Contents

UNIVERSITAT DE BARCELONA

Abstract

Facultat de Matemàtiques i Informàtica

MSc

Using Deep Learning Techniques in Click-Through Rate Prediction, Focusing on a Deep and Cross Network and an Analysis on the Use of Different Hashing Buckets

by CELINE ODDING

Using deep learning in prediction of click-through rate (CTR) is becoming main stream for advertisers engaging in real time bidding (RTB). However, there are implications for adopting a deep learning algorithm to predict CTR and to evaluate a user impression while engaging in real time bidding. In this paper, we explore two state of the art deep learning methods, DeepFM and DCN, using Logistic Regression and Factorization Machine models as a benchmark.

We explore their predictive power and the trade off of each model time with respect to training times, inference times, effects of dimension of input data when using hashing buckets, and volatility of prediction of models from training to training. We experiment comprehensively throughout our research with the goal of striking a balance between discovering the best predictor of CTR to enhance a company's RTB strategy whilst understanding the cost of chasing a (usually) more complex implementation in order to obtain an increase in predictive power.

The deep learning models outperform Logistic Regression in RIG, with the DeepFM model achieving the best RIG however the opposite is true for model complexity, training, and inference times. Increasing the hashing bucket size leads to better performances across Logistic Regression. Finally, we look at the volatility of a models prediction vector under retraining with different training data conditions while keeping in mind the goal of developing a real bidding algorithm that takes as input the output of our CTR prediction model.

Acknowledgements

I would like to express my gratitude to several individuals who have played a significant role in the completion of this thesis:

First and foremost, I want to extend my sincere appreciation to my research partner JD O'Hea. It has been a pleasure working together on this project, and your contributions and insights have been incredibly valuable.
I am very grateful to my supervisor, Jordi Vitria, for his guidance and support throughout the master's program. Your expertise and dedication as a professor have been instrumental in shaping my understanding of the subject matter.
I would like to express my profound gratitude to Victor Delgado and Elchanan Solomon from Smadex for their incredibly valuable contributions. Their extensive experience and feedback have greatly enriched this project, and I have learned a great deal from their expertise in the field of Data Science.
I want to end with expresssing my heartfelt thanks to my family, who have been a pillar of support throughout this journey.
Lastly, I am incredibly grateful to my boyfriend Emmet for his unwavering encouragement and presence. Your belief in me has been a continuous source of motivation, and I truly appreciate all the support and everything you have done for me.

To all those mentioned above, I am sincerely thankful for your contributions, guidance, and support. This thesis would not have been possible without each and every one of you.

Chapter 1

Introduction

As advertisements now reach a much larger audience online, ones place on the web have become indispensable. Online advertisements are retrieved through real time bidding (RTB), where advertisers bid for an ad space with the highest bidder getting the opportunity to place this advertisement on the desired place, making this method similar to the stock market.

A helpful technique for companies in this process of bidding on advertisements is the use of click-through rate (CTR) prediction, in which the likelihood of a user clicking on an advertisement is calculated. Several approaches have been evaluated for this technique, varying from Logistic Regression as used by Richardson, Dominowska, and Ragno, 2007 to multiple deep learning models (Zhang et al., 2021).

Smadex is a company that creates such algorithms to automate the process of buying and selling advertisements. Since the company is using Logistic Regression now for CTR prediction, we will introduce deep learning models in this research to broaden the scope of possible algorithms. Three new techniques are being deployed, Factorisation Machine (FM) (Blondel et al., 2016), Deep Factorisation Machine (DeepFM) (Guo et al., 2017) and Deep & Cross Network (DCN) (Wang et al., 2017), with our main focus on the latter two.

We run different experiments on both Logistic Regression and the deep learning models. For each deep learning model, we try different hyperparameters to find the optimal result that may outperform Logistic Regression. The metrics used in this thesis to evaluate the differences between models are Relative Information Gain (RIG) and Area Under the ROC Curve (AUC), which will be explained in detail further in this thesis.

Additional to our initial experiments, we deployed two other smaller experiments in this research. We assess different hashing bucket sizes when preprocessing the data, leading to a smaller collision rate within the hashed data. We analyse the hypothesis if a smaller collision rate would lead to better performances for the models.

We also go on to investigate prediction volatility of the models. We define the prediction vector volatility of a model to be the extent to which the vector predictions of a model on a dataset changes when there is a change to the conditions under which the model is trained on. That being the model and it's parameters remain the same while the nature of the training data changes in some way. This change can be a different sampling, data drift, or reordering of the dataset to give a few examples.

Contributions

This thesis is written by JD O'Hea and Celine Odding as a team project. We will briefly describe each of our contributions to this work, beginning with the work of Celine followed by the work of JD. Finally, we provide a link to our GitHub repository where you can find the experiments conducted for our thesis.

Celine Odding:
In this thesis, my main focus was on implementing and deploying the Deep & Cross Network (DCN). To accomplish this, it was important to first understand the complexity of this model, and ultimately running several experiments. The additional experiment was the impact of the use of different hashing bucket sizes, studying the outcomes on Logistic Regression and a small study on different hyperparameters of the DCN model. This study on the DCN model was of particular importance, given that it served as the central focus of my research.

JD O'Hea:
My primary interest is on the Deep Factorisation Machine (DeepFM) model. I implemented and experimented with a DeepFM model in line with our agreed experimentation process. Furthermore with our resulting models I undertook a prediction volatility analysis which I proceed to define and explore in my respective sections. I measure volatility using two metrics, Kullback-Leibler KL Divergence and Mean Absolute Difference (MAD) between two prediction vectors, and interpret the results for each of the models in our study. The volatility study helps to provide a fuller picture when evaluating the pros and cons of each model and deciding which model to develop into a production application.

Github:
Github Repository Thesis

Chapter 2

Background

In this chapter, we discuss three important elements of the project. We introduce our case study, Smadex. Then we describe our dataset. Finally we clarify core concepts of the research: real time bidding and click through rate.

2.1 Smadex

This research has been put forward by Smadex, an Entravision company. Smadex is a Demand Side Platform (DSP). A DSP is a company that creates software and algorithmic solutions to automate the process of buying and selling ad impressions in real time. A typical customer of Smadex would be an entity that has a product or service they wish to advertise for on a given medium. Smadex purchases internet user impressions on ad exchanges. Ad exchanges are similar to stock exchanges in that they allow advertisers and publishers to buy and sell ad inventory directly through real time bidding (RTB), and without the need to have an intermediary involved in the transaction.

2.1.1 Technology at Smadex

Smadex uses the distributed computing framework Apache Spark. Apache Spark is an open-source unified analytics engine for large-scale data processing. We keep in mind the limitations and the cost of switching technology when weighing the pros and cons of our results.

2.1.2 Dataset background

The dataset Smadex provides is a fully labelled user impression dataset generated in-house from their own current RTB algorithms. It contains a row per user impression. The specific dataset are tasked with is a gaming dataset where Smadex's clients are gaming company (details on columns are available in Table **??**, Appendix A). Smadex advertises on behalf of the gaming company through in app advertisements. In this dataset a conversion is when a user impression results in an installation within 2 weeks of having had an impression on the advertisement that Smadex exposed the user to.

Smadex challenges us to beat their baseline Logistic Regression model on the given dataset using a Deep Learning model. Again, we take into account the machine learnign capabilities of the Apache Spark MLlib when we are weighing the pros and cons of our results.

2.2 Real time bidding

Real time bidding (RTB) is a popular and optimistic business model for online computational advertising. RTB operates similarly to the stock market, as it uses computer algorithms to automatically buy and sell advertisements in real-time (Yuan, Wang, and Zhao, 2013). Advertisers bid for ad space, and the highest bidder gets to show their ads to the desired audience, maximizing their reach and visibility among the target audience.

When an ad is bought by the highest bidder, it is shown to specific people based on their data information. Specifically, ads are shown to individuals who are most likely to be interested in advertisers' offerings based on their demographic information, browsing behavior, and other relevant data. This personalized approach increases the effectiveness of display advertising, ensuring that the right message reaches the right people at the right time.

RTB has revolutionized online advertising by changing the way ads are bought and sold. Instead of the traditional methods of purchasing media space or specific ad slots, RTB focuses on buying ad space that targets specific audiences. This shift in approach is expected to become the standard business model for online advertising in the future (Yuan et al., 2014).

2.3 Click-through rate prediction

Click-through rate (CTR) is an important metric in the world of online advertising, serving as a measure of the effectiveness and engagement of ad campaigns. CTR represents the likelihood of a user clicking on a particular advertisement after being exposed to it (Yang and Zhai, 2022). Predicting the CTR of an advertisement is valuable because it helps advertising platforms generate more revenue and increases user satisfaction. Therefore, it is an important topic in the field of online advertising for both marketing purposes and research (Xiong et al., 2019).

Several factors influence the CTR of online advertisements, including gender, age, type of advertisement, and the timely and effective prediction of the CTR of online advertising and the advertisement text itself (Wang, 2020).

Following Wang, 2020, advertising prediction models are generally divided into two categories: shallow learning models and deep learning models. Shallow learning models refer to traditional machine learning algorithms that have been widely used in the field of advertising prediction. These models typically involve linear regression, Logistic Regression, decision trees, random forests, and gradient boosting algorithms (Ruppert, 2004).

On the other hand, deep learning models have become very popular in recent years due to their ability to discover complicated patterns and comprehend complex information present in large datasets. These sophisticated models, such as deep neural networks, convolutional neural networks, recurrent neural networks, and transformers, have consistently delivered impressive performance across various domains. This demonstrates their effectiveness in accurately predicting the performance of advertisements (LeCun, Bengio, and Hinton, 2015).

Chapter 3

Method

This chapter contains a detailed analysis of the data, focusing on data exploration and preprocessing techniques. Following the data analysis, we introduce and discuss the various models employed in our experiments.

Given that there is yet to be a comparative study of DeepFM and Deep & Cross (DCN) on a real time bidding dataset, the experimentation is designed to explore the strength and weaknesses of both techniques and to compare the results from each.

3.1 Data

The final dataset obtained from Smadex was for the month of December 2022. Each row represents a user impression i.e. an internet/app user saw the ad, and has a label in {1,0}. It was preprocessed by downsampling the non-converted impressions at a rate of 1 in 200, this was done due to memory limitations. The total number of remaining rows is 675,625. There is a bias in the data given that this is only data that Smadex themselves have already deemed worth bidding on. They explore and exploit a number of techniques in order to effectively bid an win impressions on the ad exchanges. These algorithms are outside the scope of this study.

3.1.1 Data Exploration

Target Label Imbalance

Even after down sampling of the non-converted (negative) class we still have a large label imbalance of approximately 1 to 28 as shown in Figure 3.1, with the majority class being the non-converted.

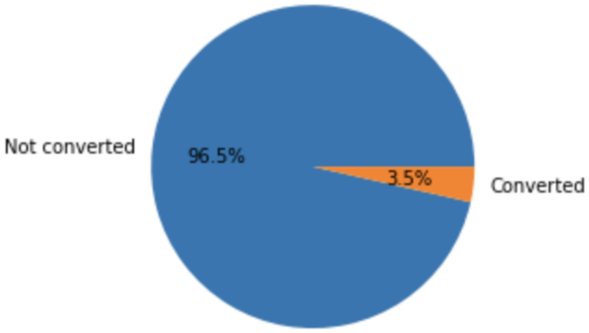

FIGURE 3.1: Pie chart showing data label imbalance.

Data Types

There are originally 24 feature columns in total. They contain information at the time of impression such as timestamps, locations, user device information, mobile/internet carrier, ad exchange information, advertisement information, and details about the website that the impression was made on. See Table **??**, Appendix A, for a breakdown and description of each of the columns. The data set contains, numeric, categorical, and ordinal data.

3.1.2 Data Preprocessing

After exploring the data, the preprocessing stage of the data is done in multiple steps. Initially, we address any missing values present in the dataset. Subsequently, we create additional columns for timezones and release dates. To encode categorical variables, both one-hot-encoding and the hashing trick are applied. Finally, the dataset is split into training, validation, and test sets.

Dealing with missing values

The first step in preprocessing the data is dealing with any missing values in the dataset. Some columns have a much higher proportion of missing values than others as you can see in Figure 3.2. We utilise missing values across the dataset by making it a unique value for the categorical columns. We decided to remove two features with exceptional high missing values, since they added no relevant information to our research.

There is a single numerical column that has missing values. "release_mrsp" is the release price of the phone model. If there is a missing value here we fill it with the average price for the phone make. If that is missing the average price for the device operating system is used. If that is also missing it is filled with the dataset average release_mrsp. The column is then normalised using a Min-max scaler.

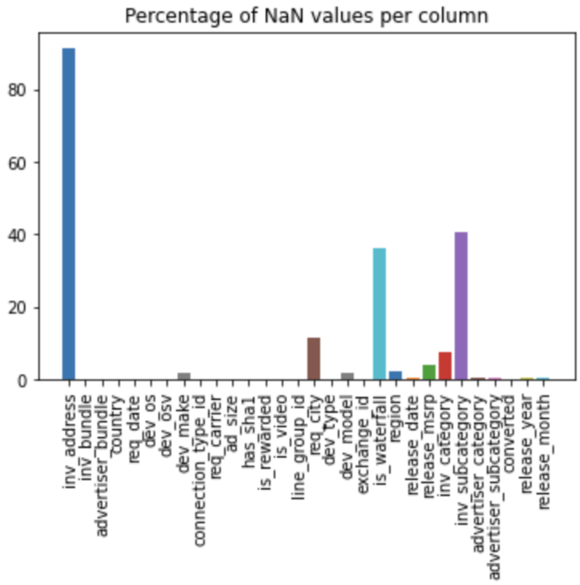

FIGURE 3.2: Percentage of missing values per column

Creating New Columns

The dataset records impressions (an advertisement shown to a user) within a specific time range, storing each action along with its corresponding date and time. However, these values are stored for multiple different countries, where each country has its own timezone. Therefore, both information of the date and the country region were needed to get time zones for each sample. Subsequently, the date is converted to its corresponding local time in a specific time zone.

After retrieving the local time of each impression, we made a new feature that returns the time of the day categorized into five possible outcomes: "late night", "morning", "afternoon", "evening" and "early night". Additionally, another feature was created to indicate the day of the week.

There is a feature "release_date" which is this release date of the device model. This feature was transformed into an ordinal value from oldest to newest, i.e. the oldest date in the training data set gets 1 and the newest gets the largest value available. This is also subsequently min-max scaled.

One-hot-encoding

To handle categorical data, specifically non-numerical values, we have used both one-hot-encoding (OHE) and hashing. OHE is a technique that transforms all categorical values into binary vectors. For each categorical value, a new vector is represented to signify the presence of the categorical value, e.g. [0,1,0].

By hand selecting columns which we identified through domain knowledge as likely to have high value but also having a low number of unique ensures when we OHE the column all the information will be preserved and the increase in dimensionality as a result of OHE will be reasonable.

Hashing Bucket

Hashing refers to the technique in which categorical data is transformed into numerical data, using a hashing function, where each value is mapped into a specific bucket in a fixed-size hash table. This causes hashing to be more memory-efficient than OHE since it is significantly reducing the memory required compared to creating a binary vector for each unique category in OHE. Moreover, hashing allows for faster computations compared to OHE when dealing with diverse categorical variables. Hashing is a widely adopted technique when developing CTR prediction algorithms such as in Pan et al., 2018, and in Smadex also. For these reasons, we use hashing to deal with features with many unique values and use OHE for elements with only a small number of categories.

We use python's implementation of MurmurHash3 hash function *mmh3.hash()*, which is a non-cryptographic hash function. It is widely used for generating hash values of input data, such as strings or byte arrays. It takes the input data and produces a 32-bit hash value as output.

The MurmurHash3 algorithm involves a series of bitwise and arithmetic operations, including shifts, rotations, and XOR operations, applied to the input data. It aims to generate hash values with good distribution and performance characteristics. The output of the function is an integer number. In order to get this output within the range of our number of buckets, we take the modulus of the output;

The number of buckets is determined based on the available memory, and specific details can be found in Section 4.1.1. Subsequently, as we obtained additional memory storage, we increased the bucket size and conducted further experiments.
As there are more unique values than buckets, there are multiple values from multiple columns getting assigned to the same bucket. When this happens we call this a collision. To calculate how often this happens we define the collision rate in Equation 3.1. The value *#UniqueHashValues* is the number of buckets and the *#UniqueInputValues* is a count of number of unique values across all of the columns that we are hashing. You can see an overview of which columns were hashes in Table **??**, Appendix A.

$$\text{Collision Rate (\%)} = \left(1 - \frac{\#UniqueHashValues}{\#UniqueInputValues}\right) \times 100 \tag{3.1}$$

Train, Validation, & Test Split

There is a full month of data. We sorted the data by the "req_date" column, which is the request date of the user to the website. All data before the 15th of December 2022 was taken as training data, from the 15th to before the 22nd as validation data, and the remainder up to and including the 30th of December as test data.

3.2 Models

We will explain all models used in our experiments, starting with Logistic Regression, followed by Factorisation Machine, Deep & Cross Network and Deep Factorisation Machine. Since our main focus is on the DCN and DeepFM model, we will explain the differences between the two, followed by the framework with which the models are developed and deployed.

3.2.1 Logistic Regression

Logistic Regression (LR) is a statistical model used for binary problems, where the dependent variable acquires two possible outcomes. LR is widely used across fields like statistics, machine learning, and social sciences due to its simplicity and interpretability (Agresti, 2015). Because of this simplicity, LR is a good fit in estimating CTR predictions since it can handle large-scale datasets with numerous features. Moreover, interpretable results allows advertisers to understand the factors influencing ad bidding decisions.

In LR, the coefficients show us how much each independent variable affects the likelihood of a positive outcome. These coefficients are estimated using methods like maximum likelihood estimation. Once we have the estimated coefficients, we can use them to understand the importance of each independent variable in predicting whether the outcome will be positive or negative (Lemeshow, Sturdivant, and Hosmer Jr, 2013). The formula for computing Logistic Regression can be seen below in equation 3.2.

$$f(z) = \frac{1}{1 + e^{-z}} \tag{3.2}$$

3.2.2 Factorisation Machine

Factorisation Machines (FMs) are a type of supervised learning method that can effectively use second-order combinations of features, even when dealing with data that has a high number of dimensions (Blondel et al., 2016). Second-order combinations of features stand for looking at how pairs of features interact with each other. In FMs, these combinations involve considering the relationships between different pairs of features. FMs can use these interactions to make better predictions and handle data with many different features. By understanding how pairs of features work together, FMs can find important patterns and make more accurate predictions.

Another advantage of FMs is that they excel at handling sparse data, which refers to datasets that are characterized by a substantial number of missing or zero values, making them particularly suitable for applications such as recommender systems, where the data often contain high sparsity (Rendle, 2010).

3.2.3 Deep & Cross Network

Since the use of FMs, there have been efforts to improve them by making them more complex by extending them to higher orders (Yang and Gittens, 2015). However, this often resulted in a large number of parameters, which can render them computationally expensive. In contrast, Deep Neural Networks (DNN) are able to capture complex relationships between features because they use lower dimensional representations of the features (embedding vectors) and non-linear activation functions. As a result, this allows DNNs to learn high-degree feature interactions in a more efficient manner than FMs.

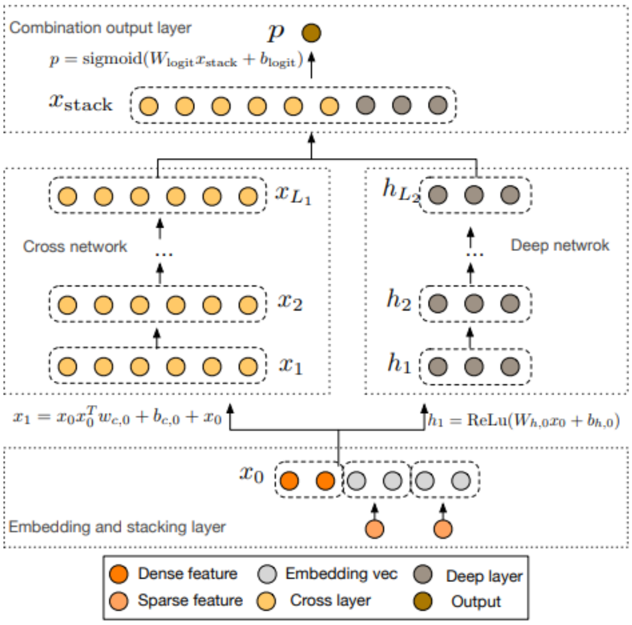
Combination output layer
p
$p = \text{sigmoid}(W_{\text{logit}} x_{\text{stack}} + b_{\text{logit}})$
x_{stack}
x_{L_1}
Cross network
...
x_2
x_1
h_{L_2}
Deep netwrok
h_2
h_1
$x_1 = x_0 x_0^T w_{c,0} + b_{c,0} + x_0$
$h_1 = \text{ReLu}(W_{h,0} x_0 + b_{h,0})$
x_0
Embedding and stacking layer
Dense feature
Embedding vec
Deep layer
Sparse feature
Cross layer
Output

FIGURE 3.3: Complete DCN model (Wang et al., 2017)

Figure 3.3 shows the representation of a full DCN model. The model starts with an embedding layer, which serves as a dimensionality reducer. Since many input features consist of categorical variables, these features are often one-hot-encoded. This means that for every category a new feature is consisting a binary value, which shows whether a category is present or not. With many features and the numerous categories per feature, this can lead to high-dimensional feature spaces. The introduction of an embedding layer can reduce the high-dimensionality by transforming binary features to dense vectors (Wang et al., 2017). Furthermore, the stacking layer stacks the embedding vector along with the normalized dense features into one vector which will then be used as an input.

The Deep & Cross Network (DCN) consists of both a DNN and a cross network. The latter is a new network structure that has been introduced to apply feature crossing automatically (Wang et al., 2017). The special structure of the cross network causes the degree of cross features to grow with layer depth, which can be seen in the figure 3.4 below.

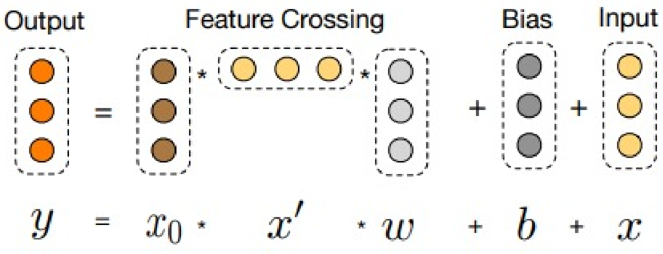

FIGURE 3.4: Representation of a cross layer (Wang et al., 2017)

The cross network has few parameters, which limits its ability to handle complex interactions. To overcome this, a deep network is included alongside it to capture those complex interactions. This deep network is a fully connected neural network.

Next, a combination layer takes the outputs from the cross network and deep network, combines them into a single vector by concatenating them, and then inputs this combined vector into a logit layer with a sigmoid activation function. The sigmoid function is used to generate a probability value between 0 and 1, which can be interpreted as the likelihood of a particular outcome.

3.2.4 Deep Factorisation Machine

Like DCN, DeepFM is a hybrid model that leverages the strengths of both feed forward neural networks and factorisation machines. It consists of two main components: a factorisation machine (FM) as described earlier and a neural network. It's implementation for CTR predictions was first proposed by Guo et al., 2017.

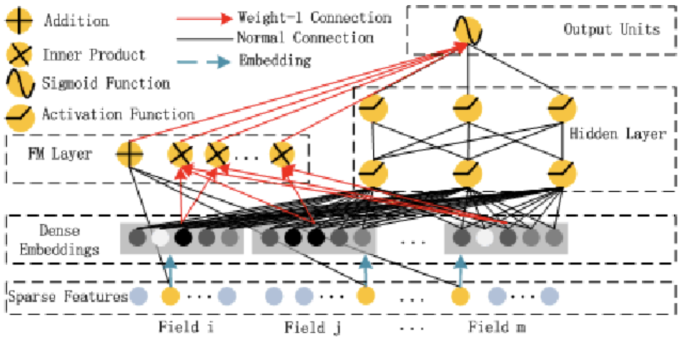

FIGURE 3.5: Complete DeepFM Model (Guo et al., 2017)

As we can see in Figure 3.5 the DeepFM model is similar to the DCN architecture as seen in Figure 3.3. There is an embedding layer to reduce dimensionality through a dense vector. In a parallel process there is a neural network and a Factorisation Machine. The outputs of these two processes are then combined in an output layer consisting of a sigmoid function.

3.2.5 DCN vs DeepFM

DeepFM and DCN have many common components differingin only one. They have the embedding layers, neural network, and output layer in common. The only component they differ on is of course their respective Factorisation Machine and Cross Network Components. Leveraging these commonalities we can draw many parallels in their implementation and complete a rigorous comparison of their behaviours and performances. A caveat worth noting is that the factorisation machine component of DeepFM is singled layered, where as cross net component of DCN has 3 layers. This configuration is kept constant throughout the study. Their respective widths is determined by the output dimension of the embedding layer.

3.2.6 FuxiCTR Repository

To develop FM, DeepFM, and DCN we leverage the FuxiCTR repository (Zhu et al., 2021). FuxiCTR provides an open-source library for CTR prediction, with key features in configurability, tunability, and reproducibility. The goal of the FuxiCTR project is to benefit both researchers and practitioners with the goal of open benchmarking for CTR prediction tasks. The repository is primarily leveraging the Pytorch framework, however there are many implementations of models in TensorFlow also.

3.3 Analysing Volatility of Predictions of Models

In click through rate prediction and real time bidding, it is one thing to be able to have an accurate prediction of how likely an impression is to convert, and it is another thing to estimate an expected value of that conversion, and the another thing again to strike a balance between putting in a bid for the impression high enough to win the bid whilst at the same time not over-bidding for the impression.

Solving bid price optimisation problem is outside of the scope of this study. The outputs of the models developed as part of this work are inputs for such a RTB algorithm. However such a bidding algorithm would desire certain properties from our models to maximise it's ability to successfully bid on impressions while maximising value realisation.

To analyse a model retraining under changing data conditions I take two approaches. In the first approach I simply shuffle the training data and re-train the model on the same data and analyse the prediction vectors for volatility. In the second approach I downsample the training data and compare the prediction variables to the same prediction vector as in the first training of the model which was trained on the full training set. Downsampling vs using a full training set better represents reality in that in reality there is increasing amounts of data and we want to know how susceptible a model is to prediction volatility while we have changing data volumes and we retrain a model.

3.3.1 Changing Distribution of Predictions

If there is a large shift in the distribution of predictions our models produce at each retraining, the bidding algorithms would need to adapt it's strategy in order to account for this shifting. This adds further complexity to the bidding strategy thus this is an undesirable property. To analyse a shift in prediction from training to training I use the Kullback-Leibler (KL) Divergence to measure how one generated distribution of predictions differs from another distribution generated from the same model after retraining.

KL Divergence quantifies how much one distribution diverges from another. It is non-negative and becomes zero when the two distributions are identical.

$$\mathrm{KL}(P||Q) = \sum P(x) \log \left(\frac{P(x)}{Q(x)} \right) \tag{3.3}$$

In Equation 3.3 we see the the definition KL divergence where P and Q represent the two probability distributions being compared, and x represents the individual probabilities of the prediction value of a sample being in a range. x is obtained by binning the prediction values, counting them, and normalising the counts in order to obtain a probability distribution for each model.

KL divergence is not symmetrical and does not satisfy the triangle inequality, however it is a good reference for us to consider how divergent our models become after a retraining from a reference distribution. Here I took the reference distribution to be the model trained on the original training data set and then to create a data change for the model I shuffled the training data and made predictions on the same sample data. With the resulting prediction vectors.

3.3.2 Changing Values of Sample Predictions

Theoretically the distribution of a models predictions could remain constant from training to training but the values each individual sample is assigned could change dramatically. I calculated the sum of the Mean Absolute Difference (MAD) for two prediction vectors from the same sample of data with different models to give us an intuition to the extent of the difference in predictions.

$$\text{Mean Absolute Difference (MAD)} = \sum_{i=1}^{n} |p_i - q_i| \tag{3.4}$$

Chapter 4

Experimental Setup

This chapter gives an in-depth description of the experiments that have been completed in this thesis. We discuss training setup and experimentation with different variations of hyperparameters and then we compare for all three models. We also introduce log loss and our Amazon web service instances on which we ran our experiments. Finally, we discuss our evaluation metrics used to compare our model performances.

4.1 Training

4.1.1 Hyperparameters

We selected the embedding dimensions and net dropout as the parameters to compare across all three models in our experiments. Additionally, for the DCN and DeepFM models, we also considered the hidden units parameter. The reason for excluding the hidden units parameter for the FM model is that it does not have any hidden units and therefore cannot be modified. Then we will also discuss the parameters related to hashing, which were applied during the data preprocessing stage.

Logistic Regression

As a baseline we implemented scikit-learn's Logistic Regression (Pedregosa et al., 2011). The setup includes 'l2' penalty, a regularisation method also referred to as ridge regression. The penalty term is the squared sum of the model's coefficients, multiplied by a regularization parameter. This penalty encourages the model to have smaller and more balanced weights, reducing the likelihood of overfitting.
Limited-memory Broyden-Fletcher-Goldfarb-Shanno (lbfgs) was set as the solver. This solver is specifically for solving optimization problems associated with differentiable functions. It is a quasi-Newton method that approximates the Hessian matrix of the objective function.

Embedding Dimensions

The embedding dimensions were first optimised for, searching for the best dimensions from the set of {2,4,5,7,10,20,40,60,80,100}. Since our dataset was much smaller than the dataset used in Guo et al., 2017 and Yang and Gittens, 2015, we decided to use a variety of smaller embedding layers next to the larger embedding layers.

Hidden Units

Next we optimised for hidden units, we searched through the configuration of {[32,32], [64,64], [128,128]}. This configuration results in a two layer feed forward neural network with the respective units per layer. The number of hidden units per layer was kept constant, as concluded in Guo et al., 2017, increasing the complexity of the model did not always bring with it benefits and they found the "constant" network, each layer having the same dimension, was empirically better than any other options explored.

Net Dropout

As with any model that grows in complexity and begins to overfit, using dropout is a useful technique for regularising a model and ensuring generalisability. With our more complex models we experimented with dropout rates in the set {0.0, 0.01, 0.1} to see if we could begin to see an improvement in the larger models.

Hashing Parameters

Memory and time limitations forced us to set the number of buckets to 1000. This value resulted in a collision rate of 95.4%. Since this rate is quite high, we decided to use extra resources and increase the bucket size to 5000 and 10000. For Logistic Regression, we conducted experiments using both bucket sizes, while for the DCN model, we utilized a bucket size of 5000 in the experiments.

A hashing bucket size of 5000 led to a collision rate of 77.3%, whereas a hashing bucket size of 10000 led to a collision rate of 59.5%, showing a great decrease in collision rate. The experiments with the bigger hashing bucket sizes are done to research whether a lower collision rate would lead to better performances.

4.1.2 Log Loss Function

Log Loss (a.k.a. Binary Cross Entropy) was used as our loss function as we have a binary classification tasks. It measures the dissimilarity between the predicted probabilities and the true labels for each instance. The Log Loss is computed using the following equation:

$$LogLoss = -\frac{1}{N}\sum_{i=1}^{N}(y_i \cdot \log(p_i) + (1-y_i)\cdot \log(1-p_i)) \tag{4.1}$$

Here, N represents the total number of instances, y_i is the true label (either 0 or 1) for the ith instance, and p_i is the predicted probability for the ith instance. The Log Loss function quantifies the dissimilarity between the predicted probabilities (p_i) and the true labels (y_i). For positive instances ($y_i = 1$), the first term $(y_i \cdot \log(p_i))$ penalizes low predicted probabilities (p_i) as they deviate from the true label of 1. For negative instances ($y_i = 0$), the second term $((1-y_i)\cdot\log(1-p_i))$ penalizes high predicted probabilities (p_i) as they deviate from the true label of 0. The Log Loss function is averaged over all instances in the dataset, providing a measure of the overall dissimilarity between the predicted probabilities and the true labels.

Down-sampling the data due to memory restrictions means the data set is not representative of the true balance. To ensure we optimised the models for the real world data balance the loss of the individual samples were weighted relative to the down sample rate. Thus the final loss function used is the Weighted Log Loss.

$$WeightedLogLoss = -\frac{1}{N}\sum_{i=1}^{N}(w_i \cdot [y_i \cdot \log(p_i) + (1-y_i)\cdot \log(1-p_i)]) \tag{4.2}$$

The variable w_i is introduced here. In this case, if y_i is 1 the w_i takes the value of 1 and if y_i is 0 then w_i takes value 200. This corrects for the down sampling of the

0 labelled data. Finally this loss function is utilised in all of our relevant employed models.

4.1.3 Amazon Web Services Instance

We ran our experiments on a G5 Amazon Web Services (AWS) instance (Types, 2023). This instance contains a 24GB memory Graphics Processing Unit (GPU), 4 virtual Central Processing Units (vCPUs) and 16GB Random Access Memory (RAM). Due to the nature of our smaller models and low dimensional data set we were not able to realise an increase in speed of training time with the GPUs. This is likely because the bottle neck in our problem was not in the vast matrix multiplication you experience with larger models, but in the transfer of large amounts of data between the RAM and the GPU's memory. In the end of the initial experiments it was faster to train and run our models on the CPU only. The instances also come preinstalled with Python version 3.10.10 and Pytorch version 2.0.0 and these packages were used throughout our experimentation.

Allocating swap memory was necessary when loading the large datasets into memory. The methodology for loading the data set exceeded the available 24GB of the instances. Swap memory is a portion of the hard drive designated for temporarily storing data that cannot fit entirely in RAM. Using swap memory involves transferring data between RAM and disk, which can significantly slow down data access and processing times. Although swap memory was required for the data load, we monitored the training and inference processes and the swap memory was not utilised at any other point in our experiments.

4.2 Evaluation

4.2.1 Metrics

The performance metrics monitored in this research was the Relative Information Gain (RIG) and Area Under the ROC Curve (AUC).

Relative Information Gain

Normalised Log Loss (NLL), also known as Normalised Cross Entropy, is an evaluation metric which enables us to measure and compare model performance. It was introduced by He et al., 2014 with the following formula 4.3.

$$NormalisedLogLoss = \frac{-\frac{1}{N}\sum_{i=1}^{N} w_i \left(y_i \cdot \ln(p_i) + (1-y_i)\cdot \ln(1-p_i)\right)}{-\frac{1}{N}\sum_{i=1}^{N} w_i \left(y_i \cdot \ln(\bar{p}) + (1-y_i)\cdot \ln(1-\bar{p})\right)} \tag{4.3}$$

- N is the size of the test set (total number of ad impressions).
- y_i is the observed label for sample i
- $\bar{p}$ is the observed click-through rate (proportion of clicks to ad impressions) in the test set. This also needs to be adjusted for the down sampling and is done so as in Equation 4.4. Note this equation can be derived given that we down sampled from 200 to 1 for the negative class. $\mathbf{y}$ is the true label vector of our test dataset and the average of this is the number of positive labels divided by the total number of labels in $\mathbf{y}$
- p_i, where $p_i \in \mathbb{R}$ and $0 \leq p_i \leq 1$, is our model's predicted probability score (that the user will click).

- p, where $p \in \{0,1\}$, is the observed probability score (often referred to as the label). 1 indicates that the user did click (the probability of click is 1 since we're certain they clicked): $P(click|user, ad) = 1$. 0 indicates that the user didn't click (the probability of click is 0 since we're certain they didn't click): $P(click|user, ad) = 0$.

$$\bar{p} = \frac{1}{1 - 200 + \frac{200}{avg(\mathbf{y})}} \tag{4.4}$$

$$RelativeInformationGain = 1 - NormalisedLogLoss \tag{4.5}$$

The numerator in the Equation 4.3 calculates the cross-entropy between the predicted probability scores (p_i) and the observed labels (y_i), while the denominator term calculates the cross-entropy between the observed labels y_i and the observed click-through rate ($\bar{p}$). The cross-entropy is averaged over all the instances in the test set, giving the Normalized Log Loss value.

Intuitively, the Normalised Log Loss can be interpreted as the relative improvement of the trained models predictions p_i compared to a model that predicts a constant value of the observed probability of converting $\bar{p}$. Subsequently $1 - NormalisedLogLoss$ gives us RIG, the greater the RIG score the better the model.

Area Under the ROC Curve

The Area Under the Receiver Operating Characteristic Curve (AUC) is a widely used performance metric in binary classification tasks, providing an evaluation of the classifier's ability to discriminate between positive and negative instances. It quantifies the overall quality of the model's predictions across different classification thresholds.

The Receiver Operating Characteristic (ROC) curve is a graphical representation that illustrates the trade-off between the true positive rate (TPR) and the false positive rate (FPR) at various classification thresholds. The TPR, also known as sensitivity or recall, measures the proportion of positive instances correctly classified as positive. On the other hand, the FPR represents the proportion of negative instances incorrectly classified as positive.

The AUC is calculated by integrating the ROC curve, which represents the probability that a randomly chosen positive instance is ranked higher than a randomly chosen negative instance by the classifier. A higher AUC value indicates a better classifier performance, where an AUC of 1 represents a perfect classifier, and an AUC of 0.5 suggests random guessing.
The AUC can be computed using the following equation:

$$\text{AUC} = \int_{-\infty}^{\infty} \text{TPR}(FPR)\delta FPR \tag{4.6}$$

TPR(FPR) represents the interpolated true positive rate at each false positive rate value, and the integration is performed over the range of possible FPR values. The integral essentially computes the area under the ROC curve.

The AUC metric is advantageous as it remains unaffected by the classification threshold selection, making it suitable for comparing different classifiers and assessing their overall performance. Additionally, AUC is particularly useful in imbalanced datasets where the number of positive and negative instances differs significantly, as it does in our case.

Chapter 5

Results

In this chapter, we will discuss the analysis of the experimental outcomes. The results will be organized based on various hyperparameters across the three deep learning models. Following that, we will assess the performance of Logistic Regression in comparison to the deep learning models, using their best parameters which resulted into their best outcomes.

5.1 Embedding Dimensions

We ran the three deep learning models on different embedding sizes, ranging from 4 to 100. For all models and different embedding sizes, prediction time, training time, AUC and RIG were calculated and shown in this section.

5.1.1 Prediction Time

Figure 5.1 shows prediction time for different numbers of dimensions, where time is represent in seconds. For all models, prediction time increased when the number of dimensions increased as well. We see the biggest change in the DCN model, which had a lower prediction time than the other models with lower embedding dimensions. However, as the number of embedding dimensions increased, prediction time for the DCN model increased too, leading into the highest prediction time of all models.

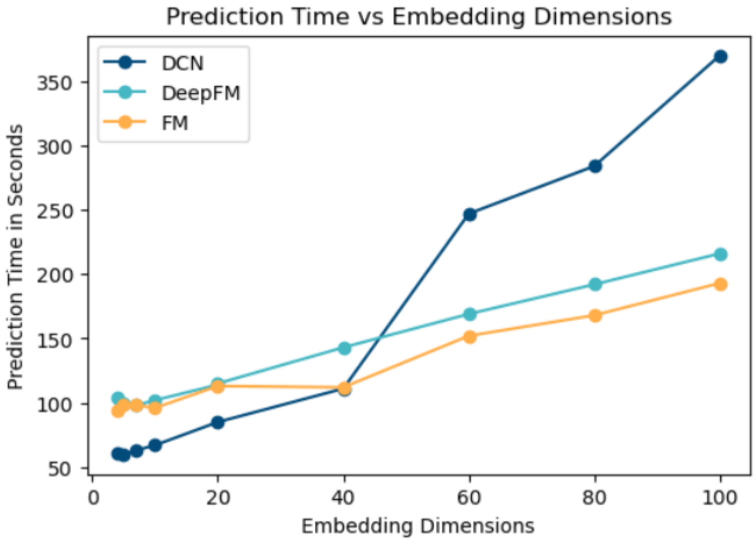

FIGURE 5.1: Graph of the prediction time vs embedding dimensions

5.1.2 Training time

The next figure, Figure 5.2, shows training time in minutes against embedding dimensions. A big outlier in this graph is the FM model, which showed a high training time for 80 dimensions. Nevertheless, the DCN and DeepFM models exhibited more stable performance, with the DCN model slightly outperforming the DeepFM model for 60 dimensions.

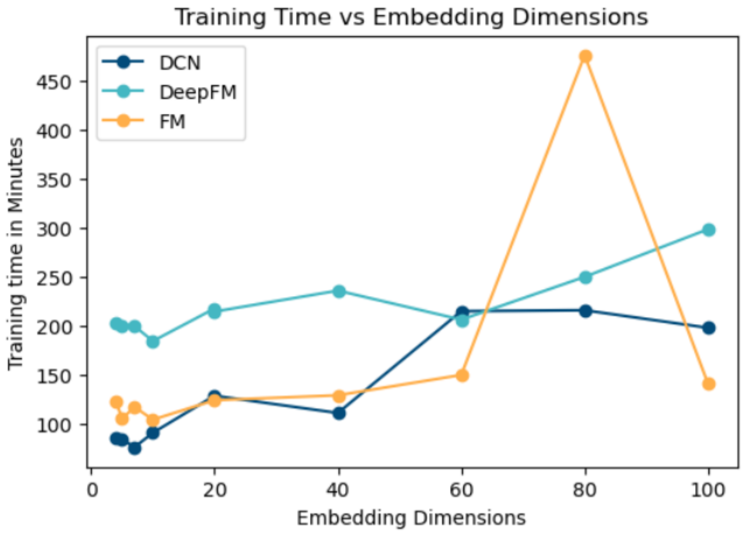

FIGURE 5.2: Graph of the training time vs embedding dimensions

5.1.3 Relative Information Gain

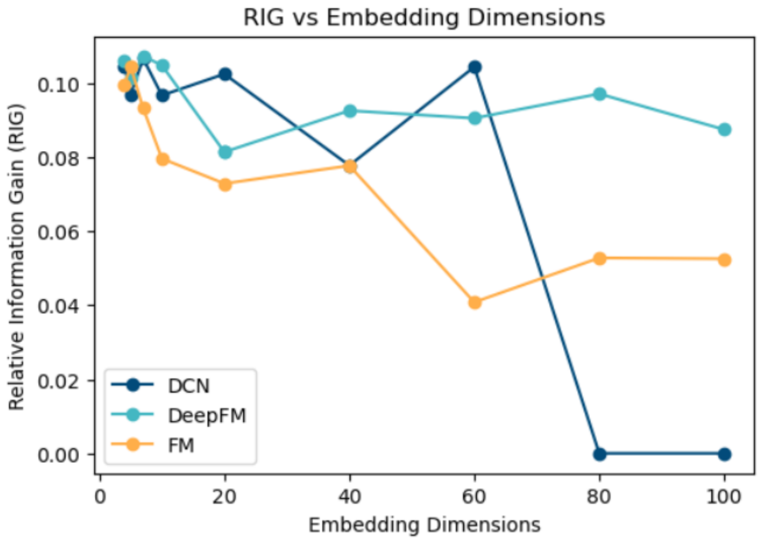

FIGURE 5.3: Graph of the relative information gain vs embedding dimensions

The graph presented in Figure 5.3 shows the relationship between different embedding dimensions and their corresponding Relative Information Gain (RIG). It is evident that as the number of embedding dimensions increased, all models exhibited a decrease in performance. However, the DCN model demonstrated two notable outliers. Specifically, when utilizing higher embedding dimensions of 60 and 80, the RIG for the DCN model dropped to zero.

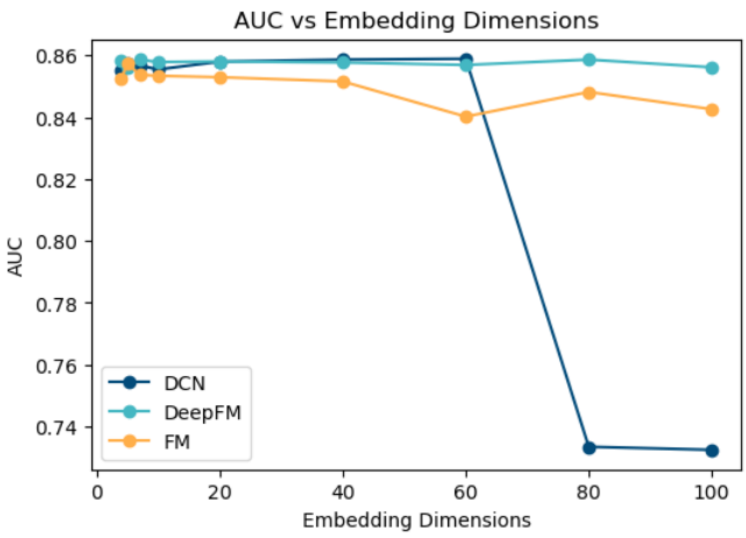

FIGURE 5.4: Graph of AUC vs embedding dimensions

5.1.4 Area Under the ROC Curve

Within this paragraph, the final graph, presented in Figure 5.4, illustrates the AUC score across different embedding dimensions. The performance of all three models remained relatively stable, yielding similar outcomes. However, as observed in previous graphs, an outlier was apparent in the DCN model. Notably, for the highest dimensions of 60 and 80, the AUC score for the DCN model experienced a significant decline, differing from the consistent performance observed in other cases.

5.2 Hidden Units

As there are no hidden units in the FM model, we only compared the DCN model and the DeepFM model for different hidden units. Again, prediction time, training time, NLL and AUC are plotted for the two models.

5.2.1 Prediction Time

Like mentioned, we only evaluated the DCN and DeepFM model for this case. Figure 5.5 shows us that prediction time (in seconds) overall took longer for the DeepFM model than for the DCN model. To show the difference within the different hidden units for both models, we have created a Table 5.1 which can be seen below.

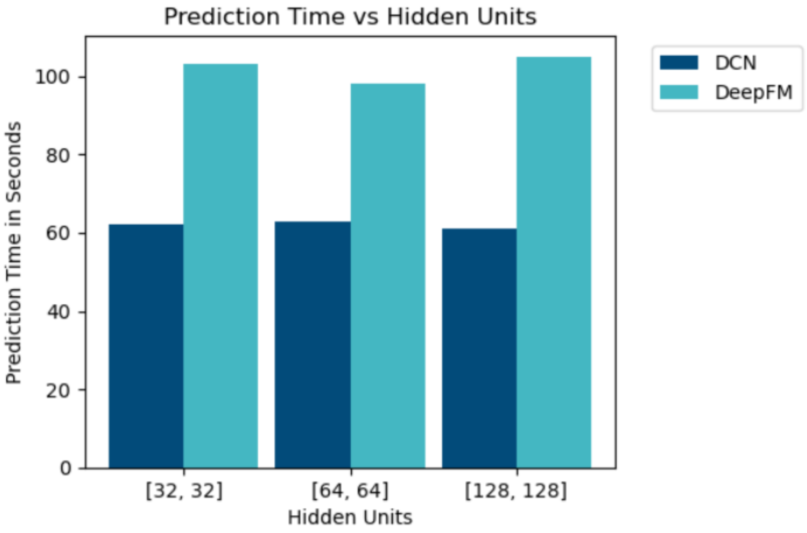

FIGURE 5.5: Bar chart of prediction time vs hidden units

The table shows us that the DCN model achieved its fastest prediction time with hidden units of [64,64], while the DeepFM model performed fastest with hidden units of [32,32]. Nevertheless, both models did not show great differences between different hidden units within each model.

TABLE 5.1: Prediction Time for DCN and DeepFM Models with Different Hidden Units

Hidden Units	DCN (time in s)	DeepFM (time in s)
32,32	63	98
64,64	61	105
128,128	62	103

5.2.2 Training Time

The graph in Figure 5.6 displays the training time in minutes for each model across different hidden units. As observed, the DeepFM model required more time to train compared to the DCN model, with the longest training time observed for hidden units of [32,32]. Examining Table 5.2 in detail, we can observe that the training time for the DCN model did not significantly vary across different hidden units. However, the DeepFM model showed a notable difference of approximately 12 minutes between the hidden units [32,32] and [128,128].

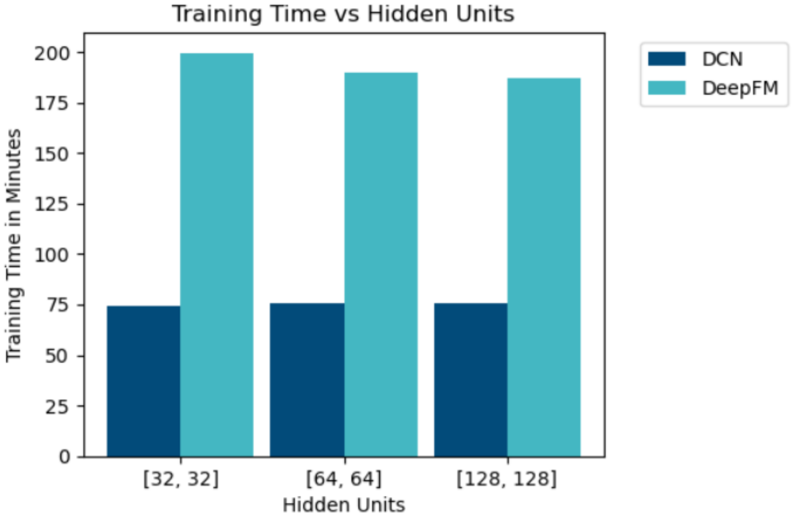

FIGURE 5.6: Bar chart of training time vs hidden units

TABLE 5.2: Training Time for DCN and DeepFM Models with Different Hidden Units

Hidden Units	DCN (time in min)	DeepFM (time in min)
32,32	75.4	199.4
64,64	75.6	190.0
128,128	74.3	187.3

5.2.3 Relative Information Gain

The RIG is compared for each model and the different hidden units, as shown in Figure 5.7. The results indicated that there was not a significant difference between the two models, as well as among the different hidden units, suggesting similar performances.

Upon examining Table 5.3, it can be observed that the DCN model resulted into the same RIG for all hidden units. On the other hand, the RIG performance of the DeepFM model decreased as the hidden units increased in size. Note that the results are not rounded, as even the slightest variation can make a difference. This is important to ensure that the smallest changes in results are accounted for.

5.2.4 Area Under the ROC Curve

Similar to the results seen in Figure 5.7, the results in Figure 5.8 shows similar results for both models in terms of performance of the AUC. Again, results did not differ

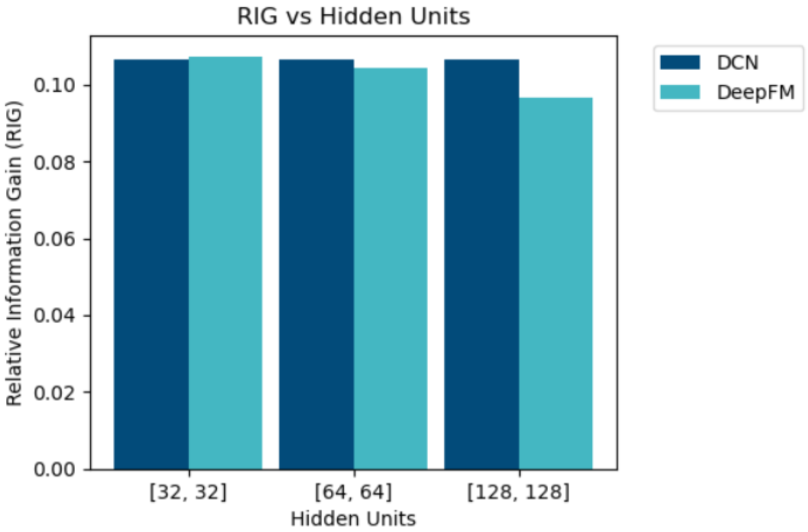

FIGURE 5.7: Bar chart of RIG vs hidden units

TABLE 5.3: RIG for DCN and DeepFM Models with Different Hidden Units

Hidden Units	DCN (RIG)	DeepFM (RIG)
32,32	0.106626	0.107288
64,64	0.106626	0.104496
128,128	0.106626	0.096661

significantly between both the two models and for each different value for the hidden units.

As observed in Table 5.4, it becomes evident that the AUC yielded identical results for each of the different hidden units within the DCN model. This similarity was consistent with the observations made for the RIG results of the DCN model, as presented in Table 5.3. In contrast, the DeepFM model demonstrated slight variations across the different hidden units, with the best performance achieved when using hidden units of [128,128].

TABLE 5.4: AUC for DCN and DeepFM Models with Different Hidden Units

Hidden Units	DCN (AUC)	DeepFM (AUC)
32,32	0.856674	0.858533
64,64	0.856674	0.858465
128,128	0.856674	0.858952

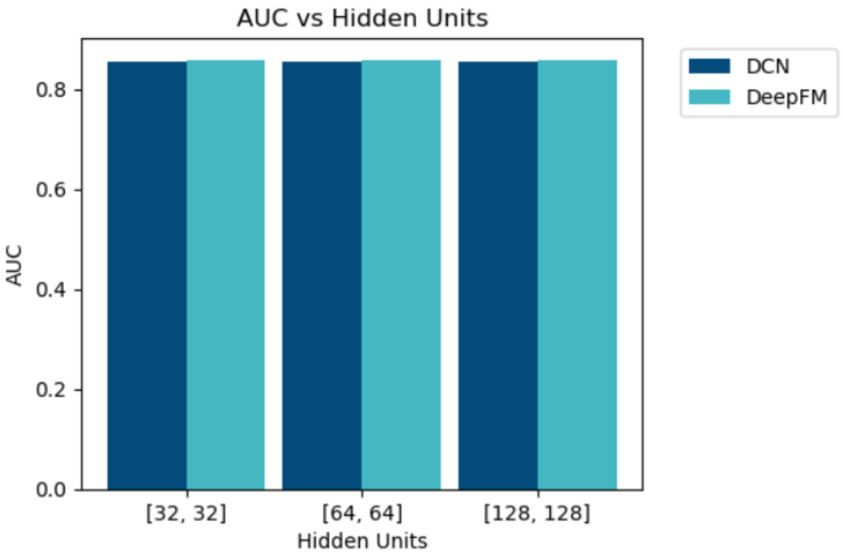

FIGURE 5.8: Bar chart of AUC vs hidden units

5.3 Net Dropout

Three different dropout rates, namely 0, 0.1 and 0.01, were applied to the DCN model and DeepFM model. We selected the embedding layer with the best result for both models, which was 7. However, we chose a different value for the hidden units than the one that achieved the best performance. This decision was made because the best performance for both models was observed with hidden units of [32,32], and we wanted to test the dropout rates against a higher value ([128,128]) for hidden units. This is done for the reason that larger hidden units could reduce overfitting of the models with a dropout rate. Again we show the four different metrics for each model and each dropout rate.

5.3.1 Prediction Time

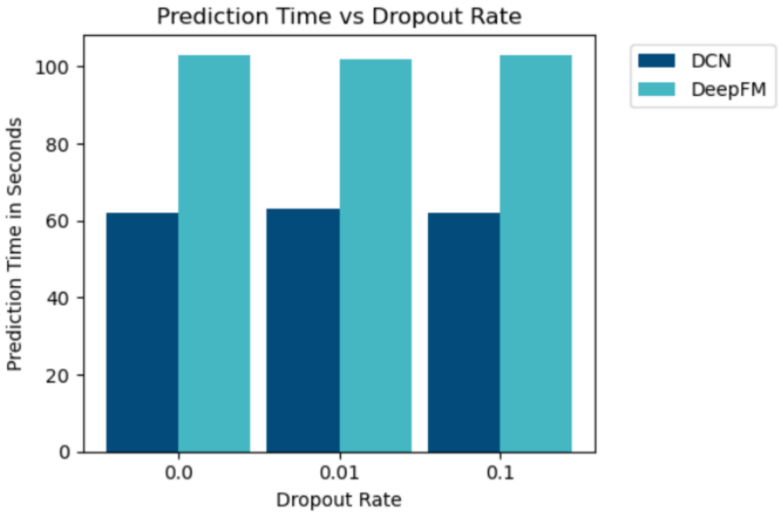

FIGURE 5.9: Bar chart of prediction time vs net dropout

Starting with observing the prediction time for different dropout rates, Figure 5.9 displays that the DeepFM model required more time for predictions compared to the DCN model. Having a closer look at the results in Table 5.5, the DCN model showed the shortest prediction times in seconds for both dropout rates of 0 and 0.01.

On the other hand, the DeepFM model performed optimally with a dropout rate of 0.1.

TABLE 5.5: Prediction Time for DCN and DeepFM Models with Different Dropout Rates

Dropout Rates	DCN (time in s)	DeepFM (time in s)
0	62	103
0.1	63	102
0.01	62	103

5.3.2 Training Time

Just like the prediction time we have seen in 5.9, the DeepFM model took longer in time to train than the DCN model did. We can see small differences for the different dropout rates in the DCN model, whereas the DeepFM showed bigger different results.

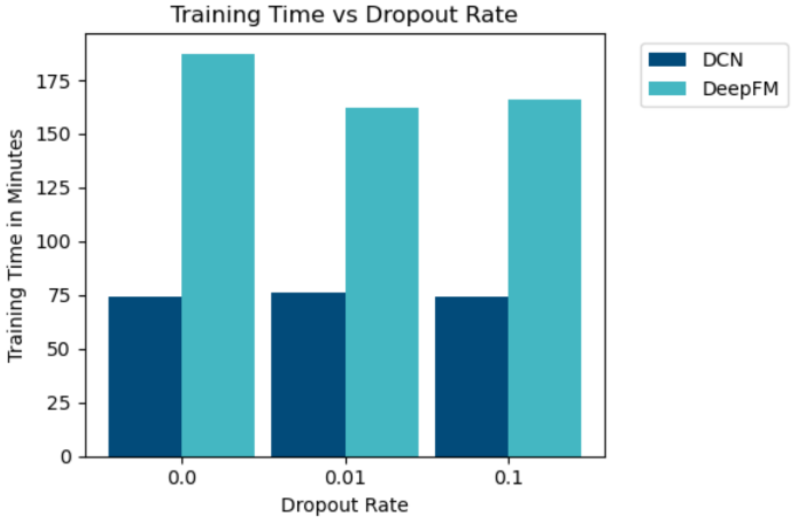

FIGURE 5.10: Bar chart of training time vs net dropout

The differences in results for both models can be observed in Table 5.6. We can observe the DCN model required the least amount of minutes for training with a dropout rate of 0, so without using dropout. The DeepFM, on the other hand, exhibited the shortest training duration with a dropout rate of 0.01, indicating a significant difference of approximately 21 minutes compared to training without dropout.

TABLE 5.6: Training Time for DCN and DeepFM Models with Different Dropout Rates

Dropout Rates	DCN (time in minutes)	DeepFM (time in minutes)
0	74.3	187.3
0.1	76.4	162.1
0.01	74.0	166.0

5.3.3 Relative Information Gain

Figure 5.12 displays the RIG for both models and their best performances, compared for different dropout rates. Overall, the DCN model showed a higher RIG compared to the DeepFM model.

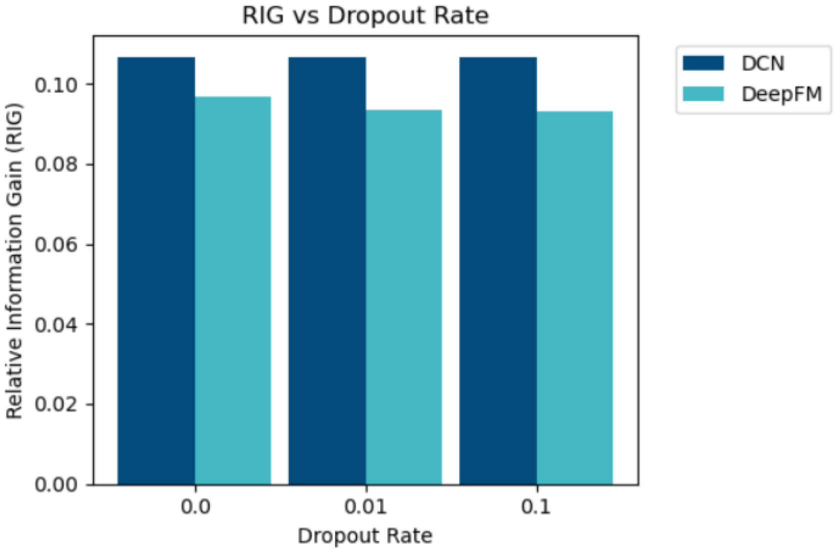

FIGURE 5.11: Bar chart of RIG vs net dropout

Again, we do not round results in Table 5.7 given that even the slightest variation can yield a notable impact. Observable is the fact that the DCN model did not change in RIG for different dropout rates. On the other hand, the DeepFM model exhibited only minor variations in RIG when subjected to different dropout rates.

TABLE 5.7: RIG for DCN and DeepFM Models with Different Dropout Rates

Dropout Rates	DCN (RIG)	DeepFM (RIG)
0	0.106626	0.096661
0.1	0.106626	0.093338
0.01	0.106626	0.093016

5.3.4 Area Under the ROC Curve

Regarding the optimal hyperparameters for both models, the AUC results demonstrated minimal variation across different dropout rates. This observation, shown in Figure 5.12, indicated that the performance of both models remained relatively consistent regardless of the dropout rate employed.

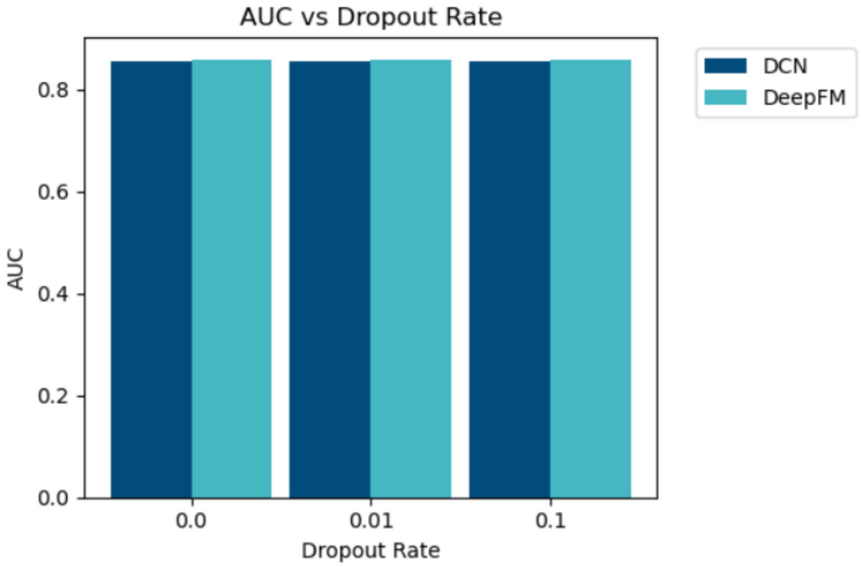

FIGURE 5.12: Bar chart of AUC vs net dropout

Table 5.8 gives us a closer look for different results in AUC for the three dropout rates. Remarkably, the DCN model showed the same results for all three dropout rates, whereas the DeepFM model again did not show much difference in AUC perfomance for the different dropout rates.

TABLE 5.8: AUC for DCN and DeepFM Models with Different Dropout Rates

Dropout Rates	DCN (RIG)	DeepFM (AUC)
0	0.856674	0.858952
0.1	0.856674	0.857745
0.01	0.856674	0.857997

5.4 Logistic Regression

Logistic Regression is evaluated using the hyperparameters discussed in Chapter 4.1.1. Table 5.9 presents the results for all four metrics obtained from Logistic Regression.

TABLE 5.9: Logistic Regression Results

Metric	Logistic Regression
Prediction Time (s)	0.51
Training Time (minutes)	1.2
RIG	0.103333
AUC	0.852689

5.5 Best Performances

Finally, we are comparing all three models with their best outcomes against Logistic Regression, these include FM, DeepFM and DCN. For each model we have chosen the hyperparameters that led to the best RIG. All four metrics are evaluated again.

5.5.1 Prediction Time

The graph in Figure 5.13 displays that Logistic Regression had the shortest prediction time, while the DCN model had the longest, with a difference of approximately 100 seconds. The DeepFM and FM models have had the same prediction time for their best configurations.

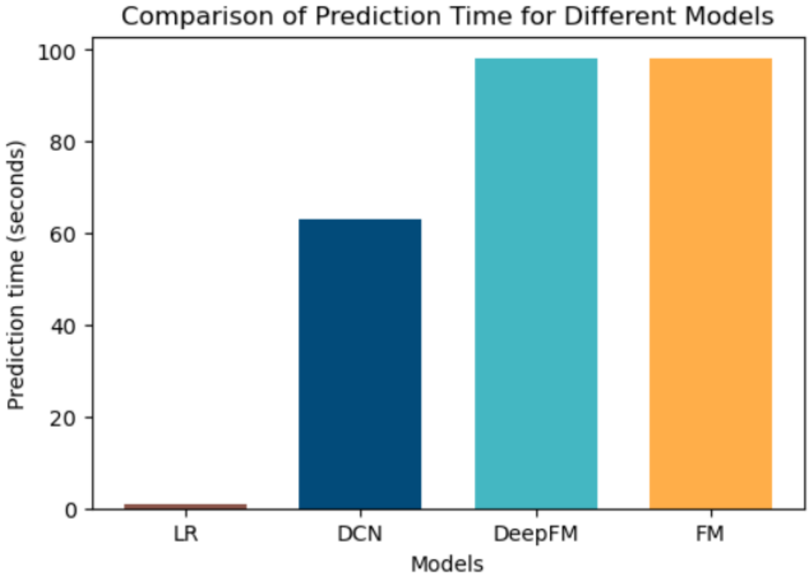

FIGURE 5.13: Bar chart of prediction time for all models

5.5.2 Training Time

When comparing the training times of the different models, Figure 5.14 highlights significant variations among them. Logistic Regression required the shortest training time, while the DeepFM model took the longest, with a difference of over 180 minutes, equivalent to more than three hours. It is worth noting the substantial difference between Logistic Regression and the three deep learning models. While the deep learning models took at least an hour to train, Logistic Regression only required less than a minute.

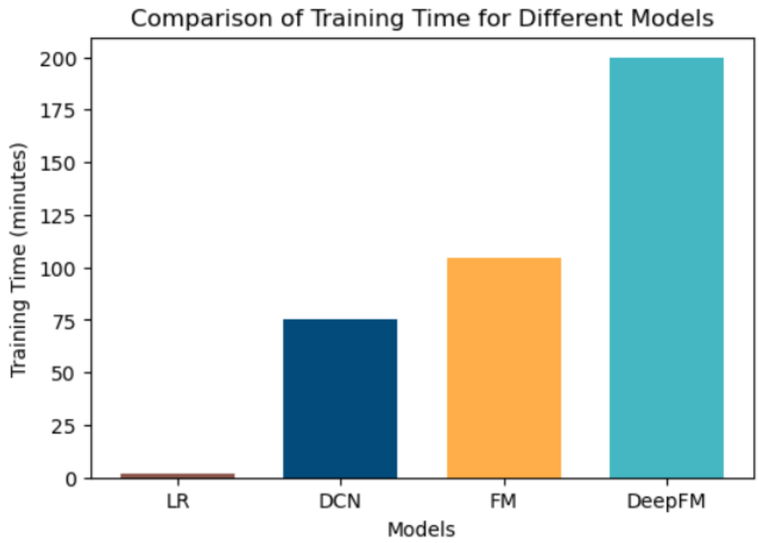

FIGURE 5.14: Bar chart of training time for all models

5.5.3 Relative Information Gain

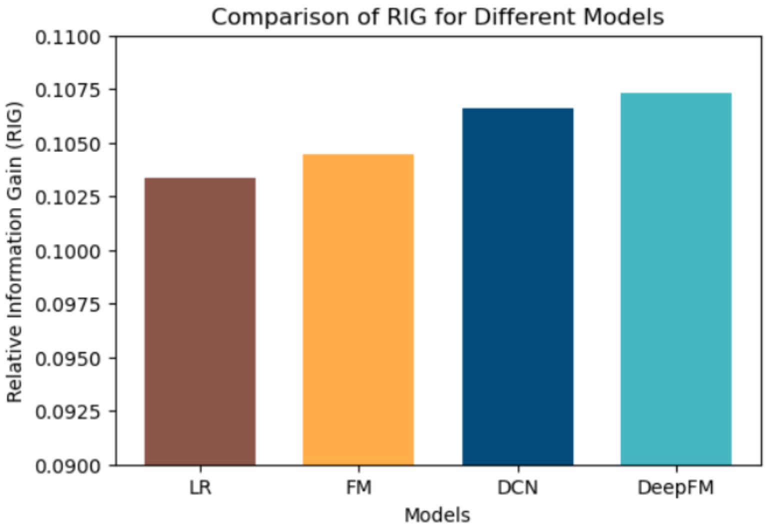

FIGURE 5.15: Bar chart of RIG for all models

Considering that the best performance was determined based on RIG, it is intriguing to examine the performance of all models in this regard. Figure 5.15 illustrates that Logistic Regression had the poorest performance compared to all other models. Although the differences in outcomes are not substantial, the DeepFM model outperformed the others.

5.5.4 Area Under the ROC Curve

The final metric we are observing for all four models is the AUC, which can be seen in Figure 5.16. Remarkably, these results show the same order of performance of the models as seen with RIG in Figure 5.15. Logistic Regression again had the poorest performance, whereas the DeepFM model (slightly) outperformed all the other models.

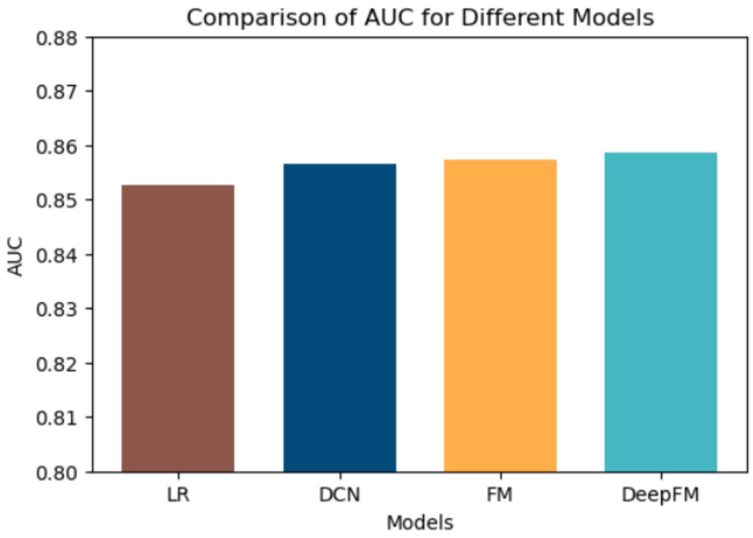

FIGURE 5.16: Bar chart of AUC for all models

5.6 Results on the Test Set

In Table 5.10 we see the results from running the best hyperparameters for each model type on the test set. The model was trained on the full set of the train and validation set. We see the DeepFM model achieved the highest RIG again closely followed by DCN, then surprisingly FM achieved a better score than LR.

TABLE 5.10: Test Set Results

Model	RIG
Logistic Regression	0.103136
FM	0.105089
DCN	0.111865
DeepFM	0.112846

5.7 Hashing buckets

After evaluation of the best models, the method of hashing is considered. Previous experiments were done with a hashing bucket of 1000. In this section hashing buckets of 5000 and 10000 were determined where Logistic Regression will be evaluated first, followed by the DCN model.

5.7.1 Logistic Regression

Prediction Time

As shown in Figure 5.17, the prediction time increased as the hashing bucket size increased. Moreover, the increase in prediction time was twice as much for each different number of buckets. However, it is worth noticing that this time is in seconds, leading even the largest bucket size to a prediction time of only 7 seconds.

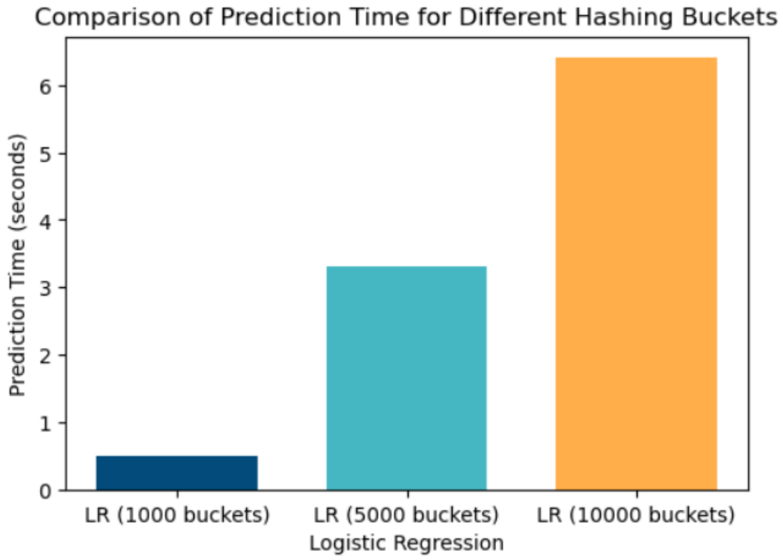

FIGURE 5.17: Bar chart of prediction times for different hashing buckets in LR

Training Time

Training time is measured in minutes, and also shows an increase as the bucket size increased, as seen in Figure 5.18. With 1000 buckets, the training time was less than a minute, whereas the largest bucket size of 10000 resulted in a training time of more than 7 minutes.

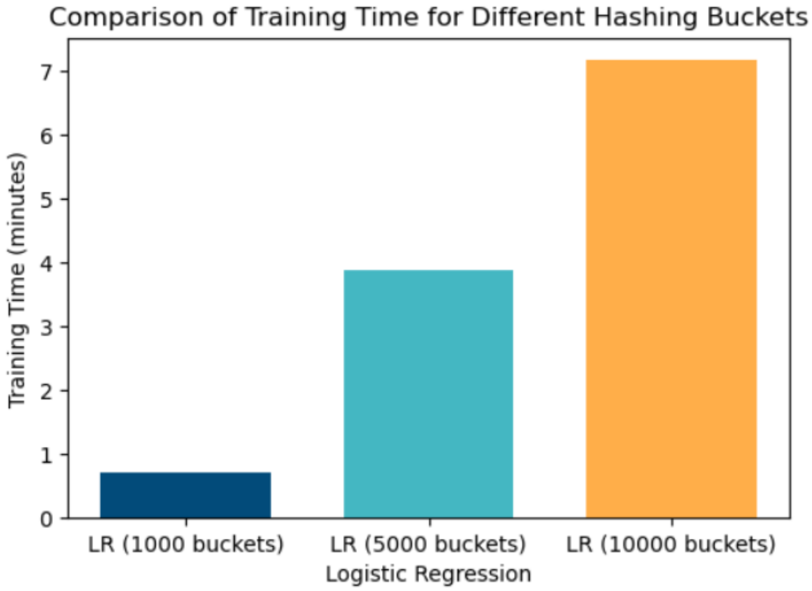

FIGURE 5.18: Bar chart of training time for different hashing buckets in LR

Relative Information Gain

The graph in Figure 5.19 shows the most important results, namely the RIG for each of the different hashing buckets. We see an increase in RIG as the bucket size increased, surpassing the performance in RIG achieved by the best deep learning models, making Logistic Regression the best model in terms of performance. This improvement is noticeable already with a bucket size of 5000, and further improved with a hashing bucket size of 10000.

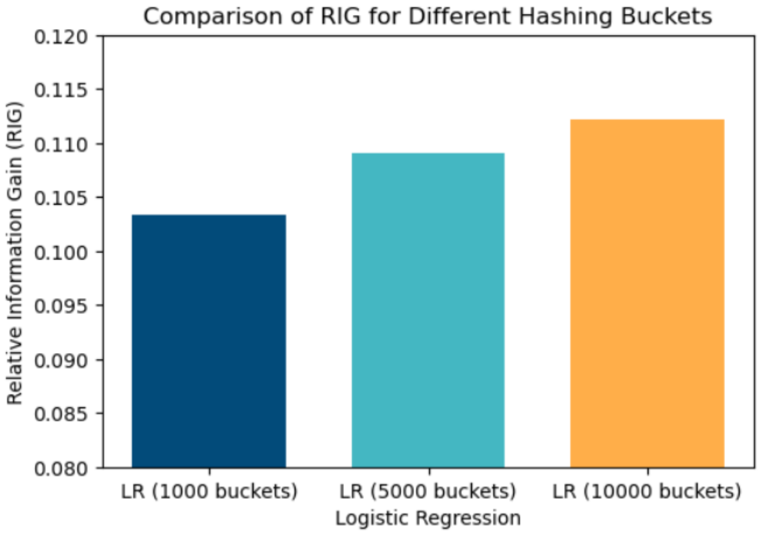

FIGURE 5.19: Bar chart of RIG for different hashing buckets in LR

Area under the ROC Curve

Even though there was not much of a change in results for AUC, Figure 5.20 shows again improvement in performance as the bucket size increased. Just like the RIG performances seen in Figure 5.19, Logistic Regression beat the best deep learning in AUC. Again, this happened already with a bucket size of 5000 and further improved with a bucket size of 10000.

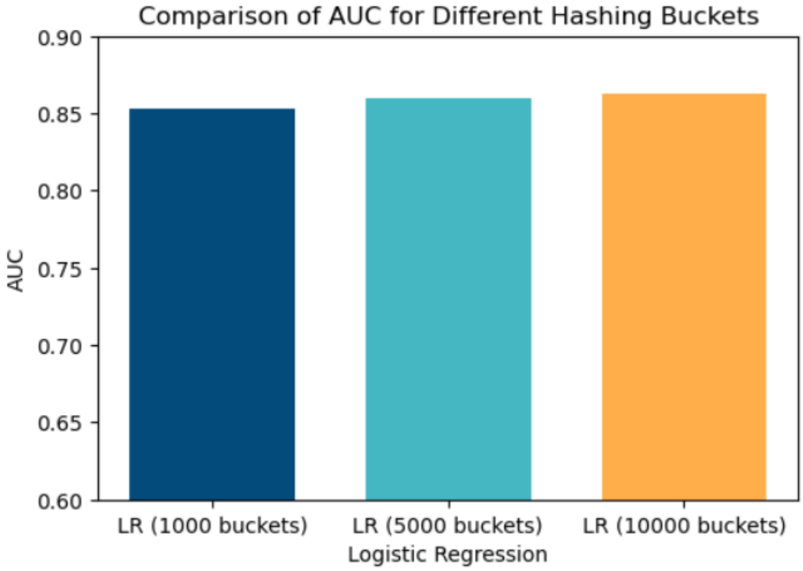

FIGURE 5.20: Bar chart of AUC for different hashing buckets in LR

5.7.2 Deep & Cross Network

For the DCN model, we decided to use the best hyperparameters and run the same experiment with a hashing bucket size of 5000 instead of 1000. Just to show a small study on tweaking hyperparameters, we only increased the embedding size where all results can be seen in Table 5.11.

As the embedding size increases, all the hyperparameters increase too. There is an exception on this behaviour, where both RIG and AUC decrease between the embedding sizes of 30 and 50. Moreover, all experiments showed worse results than obtained with a hashing bucket size of 1000.

TABLE 5.11: DCN (5000 buckets)

Metric	Embedding 7	Embedding 15	Embedding 30	Embedding 50
Prediction Time (s)	69	84	101	131
Training Time (minutes)	101	110	121	124
RIG	0.088781	0.089149	0.089212	0.088038
AUC	0.843055	0.843689	0.842541	0.840875

5.8 Volatility of Prediction Vectors

5.8.1 Shuffling Training Set

The effects on volatility of simply shuffling the training set is explored here. As expected in Figure 5.12 and Figure 5.13 we see a KL Divergence and Mean absolute difference of 0.00. This is because training a LR model is equivalent to solving a convex optimisation problem thus the order of data has no impact on the global optimum the model converges to given the same set of data.
Interestingly, DCN appeared to have a large KL Divergence and Mean Absolute Difference compared to it's counterparts.

TABLE 5.12: KL Divergence of Prediction Vector Distributions for Different Models with a Shuffled Training Set

Model	KL Divergence
LR	0.000000
FM	0.000413
DeepFM	0.001368
DCN	0.021580

TABLE 5.13: Mean Absolute Difference of Prediction Variables with a Shuffled Training Set

Model	Mean Absolute Difference
LR	0.000000
FM	0.000060
DeepFM	0.000075
DCN	0.001216

5.8.2 Downsampling Training Set

By downsampling the training set by 50% and comparing the prediction vector of a model trained on the downsampled set with the original training set, we simulated a changing volume of data and we could get an understanding of the implications of that change in volume in training on the volatility in the predictions for the validation set.

TABLE 5.14: KL Divergence of Prediction Vector Distributions for Different Models with a Fractional Training Set

Model	KL Divergence
FM	0.000098
DeepFM	0.000331
LR	0.000495
DCN	0.014095

Interestingly in Figure 5.14 the DeepFM appeared to have a lesser distribution divergence than LR. The DCN also appeared to have a relatively more significant divergent distribution. The same appeared to also be true for the Mean Absolute Difference between the prediction vectors in Figure 5.15.
We see the Logistic Regression did not do as relatively well on the downsampled data with only DCN being more volatile than it. Interestingly, FM was the least volatile model on both metrics.

TABLE 5.15: Mean Absolute Difference of Prediction Vection with a Shuffled Training Set

Model	**Mean Absolute Difference**
FM	0.000009
DeepFM	0.000062
LR	0.000075
DCN	0.001182

Chapter 6

Discussion

6.1 Inference Time & Relative Information Gain

The nature of RTB algorithms requires a trade off between a model with the best predictive power and a model that can make predictions quickly. A company that can make achieve high predictive power (in the form of RIG, for example) with a low inference time will have a huge advantage on the ad exchanges. Conversely, if you use an algorithm with high predictive power and a long inference time you will miss the auction, and an algorithm with a low predictive power and short inference time will result in winning many bids with a reduced return. This is why we focused our analysis on the trade off of prediction time and RIG.

The model with the most superior prediction time was the DCN model, the DeepFM model was a close second. Surprisingly, Logistic Regression had the lowest prediction time. Note that both of these models were experimented on the same machine, both on the same CPU.

When it comes to predictive power and RIG, the DeepFM model made the larges improvement from the LR model and the DCN model a close second. What is interesting is that even with the simplistic deep learning models like we have implemented here, we have been able to improve on the standard LR model in both inference time and RIG.

When trying to strike a balance between inference time and predictive power, there is a strong case for the DCN model. The DCN model had the lowest training time out of the DeepFM and FM models and the quickest inference time, whilst still making a significant improvement on LR.

6.2 Logistic Regression vs Deep Learning Models

We have managed to find deep learning models that are able to outperform LR in predictive power when comparing on both RIG or AUC. However, this is not the only consideration one has to make when trying to decide what will be your production model.

LR is superior over the deep learning models when it comes to training speed, and Apache Spark is limiting in the choice that is available having only a multi-layer perceptron and factorisation machine components available to implement, not a cross network component. Thus in Apache Spark MLlib, the DeepFM and FM models were the only achievable architectures. This is great news from the perspective of switching costs as cost of changing technologies is significantly reduced. The

DeepFM model was also satisfactory with respect to inference time. Although it did not achieve the best inference time, it still manages to outperform the current incumbent LR.

We quickly noticed throughout our study that as we made the DCN and DeepFM models more and more complex the models quickly began overfitting the data and were unable to generalise. This is particularly relevant in Figure 5.3 where we see the DCN RIG plummet with increasing complexity. This was the first indication we saw that we may not have enough data, even with the low dimensional hashing size, to be able to generalise. Despite this we managed to find a set of parameters for both the DCN and DeepFM models that outperformed the LR model.

6.3 Embedding Dimensions and Hidden Units

The first parameter we evaluated for the four deep learning models was the embedding dimensions. We conducted experiments with embedding sizes ranging from 5 to 100, with a step size of 20. However, we observed a decrease in performance for most models. This was particularly noticeable for the DCN model, which exhibited a decline in performance with the highest embedding dimensions, as depicted in Figure 5.3 and Figure 5.4.

Instead of opting for higher embedding dimensions, we decided to use multiple embedding dimensions within the range of 4 to 10. Since our dataset is not relatively large, we believe that the best results were achieved with an embedding dimension of 7.

Both the DCN and the DeepFM models were compared using different numbers of hidden units. When examining the RIG and AUC metrics for the DCN model (see Table 5.4 and Table 5.3), no changes were observed across different hidden unit configurations. The reasons behind this observation will be discussed in section 6.4.

On the other hand, the DeepFM model achieved the best RIG when using [32,32] hidden units. Although the AUC value remained unchanged, the best results for AUC were obtained with [128,128] hidden units.

While [128,128] hidden units yielded the best AUC results, we still believe that the [32,32] hidden unit configuration worked best for the DeepFM model. This belief is based on the larger difference in results for the RIG metric and the minimal difference for AUC.

6.4 DCN Overfitting

As can be seen in some of the DCN model results, it's metrics degrade and became constant, such as in Table 5.3 and in Table 5.4. After some analysis and testing the outcome with multiple hyperparameters, it seems that the model converged to the same point and predicted the same value for each sample. This mainly happened on runs with the larger embedding dimensions. We suspect that the cross network component of the DCN model began to take precedent over the feed forward network component and overfitted to the same point as the embedding dimensions were kept constant, resulting in the same predictions and a constant evaluation metric. It was surprising to see such a rapid degradation of predictive power as the complexity of the DCN model increased. Based on this finding, it is important to pay special attention to any signs of overfitting when implementing a DCN model.

6.5 Use of Net Dropout

In Section 5.3, we decided to use a higher number of hidden units to address overfitting in the models when dropout was applied. For the RIG metric, the DCN model consistently produced the same results, as shown in Table 5.7, likely due to reasons discussed earlier. However, the DeepFM model exhibited performance differences and generally performed less effectively when dropout was used.

Regarding the AUC metric, the results presented in Table 5.8 confirmed our findings. The DCN model maintained consistent results, while the DeepFM model achieved its best performance without utilizing dropout.

6.6 Unsuccessful Experiments

Surprisingly, the best performing embedding dimension parameter for DCN and DeepFM was of size only 7, see Figure 5.3. This stayed true even when we experimented with increasing embedding dimensions and size of the hidden unit layers with a regulariser in the form of a drop out rate. In the end the smaller simpler model with an embedding dimension of 7 won out. We suspect that with a more larger more complex dataset, which can be achieved by increasing the number of hashing buckets, that we would start to see the benefits of such experimentation.

6.7 Hashing Buckets

Like explained in Section 4.1.1, we used 3 different hashing bucket sizes, namely 1000, 5000 and 10000. All initial experiments were done with a bucket size of 1000, and further exploration was done with the latter two. Since increasing the bucket size led to a lower collision rate, we wanted to know if that would also result in better performances for the models.

Starting with Logistic Regression, Figure 5.19 showed us that increasing the bucket size indeed led to better performances. Not only did it lead to better performance within Logistic Regression, but it also outperformed the best deep learning models. This shows evidence for our hypothesis and we thus believe that increasing the hashing bucket size plays a big role in increasing performances.

Although the DCN model did not show better results with a larger hashing bucket, as seen in Table 5.11, we believe it would with further experiments when tweaking the hyperparameters and having a larger dataset. We believe this since the model showed improvement using larger embedding dimensions, and could be even further improved with more experiments. However, due to the limited scope of this project, it was not possible to identify the optimal hyperparameters for the DCN model with a hashing bucket size of 5000. Additionally, a larger dataset would be advantageous for the deep learning models as there are numerous hyperparameters that can be fine-tuned. The complexity of such models can be better captured with an increased volume of data. Although we had more data available by using a larger hashing bucket, additional data from the original dataset would likely further enhance the model's performance.

While it is true that increasing the bucket size yields improved performances, it is important to note that this approach comes with a significant memory cost. The increased memory requirements should be taken into consideration when considering the adoption of this technique. However, if the necessary resources are available to accommodate the larger memory requirements, the results obtained from increasing the bucket size can be highly promising and beneficial.

6.8 Test Set Results

There were two surprising results from running our models on the test set. The first is that the RIG improved significantly for all models on the test set. Second, Logistic Regression was the worst performing model on the test set with respect to RIG.

It appears that in order for these deep learning methods to be effective, they required huge amounts of data. It is clear that having a combination of the training set and validation set to learn from the models were able to learn and generalise better to the test set because of having a larger volume of data. There is further evidence that there was not enough data to employ deep learning techniques when we look at the best hyperparameters of the models. Relatively tiny deep learning models with embedding dimensions of 7 and [32,32] hidden units in the neural network. This is a very small deep learning model but is best suited to this dataset to avoid overfitting.

The DCN model (Yang and Gittens, 2015) and the DeepFM model (Guo et al., 2017) both rely on a dataset obtained from Criteo Display Ads [1], which consists of over 40 million samples of impression data. In contrast, our combined dataset consisted of 790,000 samples. This is encouraging as Smadex has the data available to meet this volume of data, and despite not having that volume of data we were able to discover a deep learning model that was able to generalise to the test set better than the current logistic regression solution.

6.9 Volatility of Prediction Vectors

From Section 5.8.1, as expected because of the convex nature of LR, there is no change to the prediction vector. For both the metrics the models had the same ranking of volatility from best volatility LR to worst the worst model DCN. This ranking corresponds to the same ranking you would do for complexity with DCN having the most number of parameters (three cross layers and the hidden units in parallel), then DeepFM (one FM layer and the hidden units in parallel), and then FM (a single FM layer). Therefore an increase in complexity could be one explanation for the increase in volatility for the non-convex models.

In Section 5.8.2 we see a more realistic simulation of data volume changing and the effects this has on volatility. DCN remained the worst for volatility on both KL Divergence and MAD metrics. FM had the most consistent predictor vector across both metrics with also having a lower score and LR close behind DeepFM. FM likely performed well in the downsampling analysis of volatility because it has a low complexity, therefore a lower search space and given a reasonable amount of data it will converge to a similar optimum on each run. The opposite is true for DeepFM and DCN.

LR is likely overfitting the sampled data and then overfitted again on the full dataset. However, it is likely the optimum on the new data that is different enough for it to get a worse volatility score than DeepFM and FM. This however may be rectified if it generalised better with hyperparameter tuning and more data.

When taking volatility and RIG into consideration when choosing a model to push to production, DeepFM is the most attractive option as it perform relatively well on

[1] https://www.kaggle.com/competitions/criteo-display-ad-challenge/data

both volatility and RIG, where as the other models fell short on a combination of one or the other or both.

Chapter 7

Conclusion

We developed models from end to end, including an extensive exploratory data analysis from production generated data of user impressions labelled with conversions, to having fully developed tuned and tested deep learning models with an extensive breakdown of the pros and cons of each.

An initial Logistic Regression model was developed inline with Smadex's own internal algorithm to act as a baseline for the Deep Learning models. The most cutting edge deep learning models, Deep Factorisation Machines and Deep Cross Net, in the Real Time Bidding space were developed and compared with each other and Logistic Regression.

Analysis was taken with the intent to find the best performing model for production, in the case of Real Time Bidding this means a trade off between inference time and predictive power. We found that both the Deep Cross Network and the Deep Factorisation Machine improved on Logistic Regression on both of these metrics, falling short on only training time. As the Deep Factorisation Machine achieved the best predictive power and quicker inference time than Logistic Regression, we believe this to be the best resulting model despite Deep Cross Network achieving a quicker inference time. Deep Factorisation Machine also has the advantage of having it's components available on Apache Spark MLlib, readily available for implementation.

7.1 Hashing Buckets

Our experiments with bigger hashing bucket sizes showed improvement in particular cases, specifically, in Logistic Regression. We have shown that performance got better as the hashing bucket increased. We think this was demonstrated in Logistic Regression due to its convexity, not having all the hyperparameters which the deep learning models have to deal with.

In contrast, the deep learning models did not show improvement as the buckets increased, but rather showed a decrease in performance. However, when tuning hyperparameters performance increased, giving the indication this could lead to bigger improvement in the long run.

Given that deep learning models have previously outperformed Logistic Regression when using 5000 hashing buckets, it is reasonable to believe that similar improvements can be achieved with even larger bucket sizes. To accomplish this, a combination of hyperparameter tuning and the use of more data should be employed. By optimizing the model's hyperparameters and incorporating a larger dataset, the deep learning models have the potential to show better performance.

Moreover, considering the observed performance improvement in Logistic Regression with increased hashing bucket sizes, we highly recommend the utilization of larger hashing buckets in future applications. By leveraging the benefits of larger bucket sizes, there is a greater likelihood of achieving better performances across different models. Using this approach can make predictions more accurate and effective, leading to stronger and more reliable solutions.

7.2 Volatility of Prediction Vectors

Volatility is an additional consideration when deciding the best model to implement for use in production on top of RIG, train time, and prediction time which we have discussed thoroughly throughout this research. We have defined model prediction volatility and outlined a methodology for testing the extent for which a model is susceptible to volatility by using a combination of KL Divergence and MAD on it's prediction values. This is set out with the goal of being able to develop a more stable model for a RTB algorithm to maximising return on ad spend.

Two perspectives of volatility were explored: effect of order of training data, and effect of changing volume. Changing volume is a more suitable simulation because it is a realistic scenario in RTB as data volume increases then models are retrained so it becomes essential for the RTB algorithm placing bid values to maintain stable prediction values from the CTR prediction component.

The results showed that DeepFM performed consistently well on each type of volatility analysis where as DCN was always the most volatile. We saw that FM showed the most promise on volatility but it remains to be seen the extent to which volatility effects the performance capabilities of RTB algorithms.

These results are an initial indication for the development of an end to end RTB algorithm. There are many more experiments and requirements that need to be tested before deployment, as discussed in Section 8.4.

Chapter 8

Future work

8.1 Data Volume

Due to engineering constraints we have limited our study to a single machine and training our models in memory. Because of the nature of the data, with only 1000 hashing buckets and 300 more OHE columns, we did not require a distributed data system, nor did our models benefit from the sue of a GPU while training as the cost of data transfer outweighed the benefits of improved iteration speed.

Throughout the study we were subject to overfitting. We saw this with the resulting models all being "small" deep learning models, and we saw a quick degradation in the generalisability on the validation set as we increased the complexity of our models, such as in Figure 5.4 for the DCN model.

8.2 Logistic Regression

LR received minimal hyperparameter tuning. It requires further hyperparameter tuning on the regularisation method and perhaps testing of other solvers. There may be an improvement on LR's metrics through testing of different combinations of regularisation and solver and we leave this for further work.

8.3 Hashing buckets

As results shown us that increasing the hashing bucket size led to great developments, more on this research can still be analysed when having the proper resources to do so. First of all, the hashing bucket size can be further increased, leading to a lower collision rate since we have seen the positive outcome of increasing the buckets sizes. Ideally, this would be tested on Logistic Regression, benefiting from its convex nature, and later deployed on the deep learning models as well.

Furthermore, once experiments with larger hashing buckets are conducted, a fresh examination of hyperparameters can be conducted on the deep learning models. This study would involve incorporating additional data and increasing the bucket size, potentially revealing further improvements in performance.

8.4 Volatility of predictions

Although we have developed a methodology and an intuition for the volatility of a models predictions due to time and resource constraints our results are not empirical. Next steps include running this experiment until there is a desired level of confidence in the results. For maximum robustness the experiments should be done in line with the future work outlined in Section 8.1 on increasing data volume. Another approach which was not explored here was to explore two completely independent training samples on the same validation set as opposed to what was outlined in Section 3.3 where the full dataset and a sample of the full dataset was compared against each other.

Finally this work should be extended and developed in tandem with a RTB algorithm and the ultimate performance metric that should be used should be to maximise the return on spend an algorithm achieves and a CTR prediction model combined with a RTB algorithm that achieves that maximum should be chosen. However, it is clear that having a model that can give a reliable CTR prediction is a major step in achieving that outcome

Appendix A

Column Name	Transformation	Description
inv_id	deleted	Inventory id. This is an internal ID that identifies inventory at its most granular, and is computed by hashing the publisher app/site, location, etc. Two impressions with the same inv_id correspond to the same piece of "real estate".
inv_bundle	hash	A platform-specific application identifier intended to be unique to the app and independent of the exchange. On Android, this should be a bundle or package name (e.g., com.foo.mygame). On iOS, it is a numeric ID.
advertiser_bundle	OHE	Advertiser bundle. This is the bundle specified in the line.
country	hash	Country where the ad was shown.
req_date	new column	Timestamp of the request.
dev_os	OHE	Device operating system (e.g., 'ios', 'android')
dev_osv	deleted	Device operating system version (e.g., '12.4.1', '8.1.0')
dev_make	OHE	Device make (e.g., 'Apple', 'Samsung')
connection_type_id	OHE	Identifier of connection type of the device (wifi, cellular data 2G, 3G, 4G,..)
req_carrier	deleted	Carrier or ISP (e.g., 'VERIZON'), using exchange curated string names which should be published to bidders a priori.
ad_size	OHE	Size of the creative used. (e.g., 'xlarge', 't-interstitial')
has_sha1	new column	Boolean: True if dpid_hash is set, false otherwise.
is_rewarded	boolean	Whether the user receives a reward for viewing the ad.
is_video	boolean	Boolean to indicate if the bid is for a video: 1 = true, 0 = false
line_group_id	OHE	Identifier of the campaign used in the bid.
req_city	hash	City using United Nations Code for Trade & Transport Locations.

Column Name	Transformation	Description
dev_type	deleted	The general type of device. (e.g., ’smartphone’, ’tablet’)
dev_model	hash	Device model (e.g., ’iphone xs’, ’SM-J260T1’)
exchange_id	hash	Identifier of the exchange in our DB
is_waterfall	deleted	Boolean to indicate if the request belongs to a waterfall: 1 = true, 0 = false, null = undetermined
region	OHE	Region code using ISO-3166-2; 2-letter state code if USA. (e.g., ’TX’, ’BA’)
release_date	new column	The release date of the device
release_msrp	new column	The release price of the device
inv_category	hash	The category of the inventory (publisher app)
inv_subcategory	deleted	The subcategory of the inventory, when relevant.
advertiser_category	hash	The category of the app we are advertising
advertiser_subcategory	hash	The subcategory of the app we are advertising, when relevant
converted	boolean	1 if the impression resulted in an install, 0 otherwise. This is the target variable

Bibliography

Agresti, Alan (2015). *Foundations of Linear and Generalized Linear Models*. Wiley.

Blondel, Mathieu et al. (2016). "Higher-Order Factorization Machines". In: *Advances in Neural Information Processing Systems*. Ed. by D. Lee et al. Vol. 29. Curran Associates, Inc. URL: https://proceedings.neurips.cc/paper_files/paper/2016/file/158fc2ddd52ec2cf54d3c161f2dd6517-Paper.pdf.

Guo, Huifeng et al. (2017). "DeepFM: a factorization-machine based neural network for CTR prediction". In.

He, Xinran et al. (2014). "Practical lessons from predicting clicks on ads at facebook". In: *Proceedings of the eighth international workshop on data mining for online advertising*, pp. 1–9.

LeCun, Yann, Yoshua Bengio, and Geoffrey Hinton (2015). "Deep learning". In: *nature* 521.7553, pp. 436–444.

Lemeshow, Stanley, Rodney X Sturdivant, and David W Hosmer Jr (2013). *Applied logistic regression*. John Wiley & Sons.

Pan, Junwei et al. (2018). "Field-weighted factorization machines for click-through rate prediction in display advertising". In: *Proceedings of the 2018 World Wide Web Conference*, pp. 1349–1357.

Pedregosa, F. et al. (2011). "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12, pp. 2825–2830.

Rendle, Steffen (2010). "Factorization machines". In: *2010 IEEE International conference on data mining*. IEEE, pp. 995–1000.

Richardson, Matthew, Ewa Dominowska, and Robert Ragno (2007). "Predicting clicks: estimating the click-through rate for new ads". In: *Proceedings of the 16th international conference on World Wide Web*, pp. 521–530.

Ruppert, David (2004). *The elements of statistical learning: data mining, inference, and prediction*.

Types, AWS Instance (2023). URL: https://aws.amazon.com/ec2/instance-types/.

Wang, Ruoxi et al. (2017). "Deep & Cross Network for Ad Click Predictions". In: *CoRR* abs/1708.05123. arXiv: 1708.05123. URL: http://arxiv.org/abs/1708.05123.

Wang, Xinfei (2020). "A Survey of Online Advertising Click-Through Rate Prediction Models". In: *2020 IEEE International Conference on Information Technology,Big Data and Artificial Intelligence (ICIBA)*. Vol. 1, pp. 516–521. DOI: 10.1109/ICIBA50161.2020.9277337.

Xiong, Xi et al. (2019). "A clickthrough rate prediction algorithm based on users' behaviors". In: *IEEE Access* 7, pp. 174782–174792.

Yang, Jiyan and Alex Gittens (2015). "Tensor machines for learning target-specific polynomial features". In: *CoRR* abs/1504.01697. arXiv: 1504.01697. URL: http://arxiv.org/abs/1504.01697.

Yang, Yanwu and Panyu Zhai (2022). "Click-through rate prediction in online advertising: A literature review". In: *Information Processing & Management* 59.2, p. 102853.

Yuan, Shuai, Jun Wang, and Xiaoxue Zhao (2013). "Real-time bidding for online advertising: measurement and analysis". In: *Proceedings of the seventh international workshop on data mining for online advertising*, pp. 1–8.

Yuan, Yong et al. (2014). "A survey on real time bidding advertising". In: *Proceedings of 2014 IEEE International Conference on Service Operations and Logistics, and Informatics*. IEEE, pp. 418–423.

Zhang, Weinan et al. (2021). “Deep learning for click-through rate estimation”. In: *arXiv preprint arXiv:2104.10584*.

Zhu, Jieming et al. (2021). “Open Benchmarking for Click-Through Rate Prediction”. In: *CIKM '21: The 30th ACM International Conference on Information and Knowledge Management, Virtual Event, Queensland, Australia, November 1 - 5, 2021*. Ed. by Gianluca Demartini et al. ACM, pp. 2759–2769. DOI: 10.1145/3459637.3482486. URL: https://doi.org/10.1145/3459637.3482486.