ARCADA

Click-Through Rate Prediction in Practice

A study of a click-through rate prediction system

Kurt-Eerik Ståhlberg

Master's Thesis

Master of Engineering – Big Data Analytics

2019

MASTER'S THESIS	
Arcada	
Degree Programme:	Big Data Analytics
Identification number:	7251
Author:	Kurt-Eerik Ståhlberg
Title:	Click-Through Rate Prediction in Practice A study of a click-through rate prediction system
Supervisor (Arcada):	Leonardo Espinosa Leal
Commissioned by:	ReadPeak Oy

Abstract:
Digital advertising is a huge business with tough competition. One of the ways to be more effective in the business is to serve better chosen ads to each user. One way to improve the ad selection is to predict the click-through-rate of each prospective ad and then select the one with the highest predicted CTR. In this thesis three possible choices – naïve, linear and factorization machine – for a predictive model are studied, tuned, evaluated and their results are compared with factorization machine showing the best results.

Keywords:	CTR Prediction, Factorization machines, Digital advertising, ReadPeak
Number of pages:	54
Language:	English
Date of acceptance:	31.5.2019

OPINNÄYTE	
Arcada	
Koulutusohjelma:	Big Data Analytics
Tunnistenumero:	7251
Tekijä:	Kurt-Eerik Ståhlberg
Työn nimi:	Click-Through Rate Prediction in Practice A study of a click-through rate prediction system
Työn ohjaaja (Arcada):	Leonardo Espinosa Leal
Toimeksiantaja:	ReadPeak Oy

Tiivistelmä:
Digitaalisessa mainostamisessa liikkuu massiivisia määriä rahaa ja se on erittäin kilpailtu ala. Eräs tapa tehostaa toimintaansa kyseisellä alalla on toimittaa paremmin valittuja mainoksia käyttäjille. Mainosten valintaa voi muun muassa parantaa ennustamalla mainoksen klikkaustodennäköisyys kaikille tarjolla oleville mainoksille ja valitsemalla niistä paras. Tässä opinnäytetyössä tutkitaan, säädetään ja vertaillaan kolmea vaihtoehtoa – naiivi, lineaarinen ja factorization machine – ennustemalliksi. Kokeiden perusteella factorization machine tuottaa parhaat tulokset.

Avainsanat:	
Sivumäärä:	54
Kieli:	Englanti
Hyväksymispäivämäärä:	31.5.2019

CONTENTS

Figures

Tables

FOREWORD

I would like to thank the following people:

First and foremost, my employer ReadPeak and specifically Tomas Forsbäck for making it possible to study for a master's degree while working full time and supporting this all the way through.

My colleague Tuomo Tilli for all the good ideas he's had about my work and that he's shared on our weekly walks around Lauttasaari. Walking meetings are the best!

My supervisor Leonardo Espinosa, my professor Magnus Westerlund, and Yrkeshögskolan Arcada.

And last but definitely not least, my wife Sophie for her never-ending support and love that get me through the tough times.

Vantaa
May 2019
Kurre Ståhlberg

1 INTRODUCTION

Digital advertising is a huge business. In the first half of 2018, in the United States, the revenues of online advertising were $49.5 billion (Internet Advertising Bureau, 2018). Every day, billions of people load trillions of webpages with ads on them and become targets of marketing teams with budgets big and small all around the world. The overall spend in advertising being very large, any savings in efficiency gets multiplied into a significant amount of money quite easily.

1.1 Background and Need

When digital advertising started in 1994, it was simply a static banner shaped image on a webpage (Singel, 2010). The website owners, webmasters, sold each banner to an advertiser for a set time and manually changed the HTML code in the page to reflect any changes to advertisers. This was obviously very laborious and quite soon, tools and specialized systems appeared to help with this problem. These new systems provided an easy way for the webmaster to change the ad without having to change the actual code, and soon they started to offer syndicated ad sales as well. At this point, ad engagement tracking started appearing. The ad technology companies started tracking who clicked the ad as well as who saw ads, what kinds of ads, where and whether they seemed to interact with the ad itself. This data was and still is stored to generate user profiles.

In the early 2000s, Real-Time Bidding (RTB) was born and with it, the possibility to optimize the money spent on ads by using algorithms. RTB enables advertisers to compete against each other for each and every ad impression that is available. As the price of the impression varies based on demand and perceived value, a sophisticated tool to select the impressions to bid on and the price to bid is very valuable. To optimize the price to be offered for any individual impression, the value of that particular impression needs to be evaluated. For most advertisers, the value is determined by the likelihood of the user seeing the ad actually buying something from – or otherwise engaging with – the advertiser, and then the value of said purchase or engagement. In a very simplified example, consider the following scenario: A user is browsing a webpage with an ad placement on it. The placement requests an ad from an RTB system which requests an

ad from a particular Demand Side Platform (DSP). This DSP determines that this particular user will engage with an ad campaign it has running, at 0.1% likelihood based on the user's profile match to the campaign data. The DSP also knows that an engagement for this campaign is worth 5€. This gives the maximum value of the impression at $\frac{5€}{1000} = 0.005€$. Depending on the type of auction running at the exchange, the DSP will either offer this value or less for the impression.

Traditionally, the value of the impressions for any given campaign has been evaluated using human deduction and campaign performance using different metrics. The detailed audience segmentation many major companies like Facebook and Google offer, enable the advertisers to target their advertising to very small groups of people they have evaluated as their target market. For example, an ad campaign could be targeted to "engaged women between 30 and 35 years old who want a dog and live in a suburban home" if that was considered the user segment that would engage and make the purchase of the product being advertised, whatever it may be.

These methods of targeting require high level of knowledge of the target audience – to choose the segments correctly - as well as large enough budget to gather a meaningful amount of data to make improved targeting decisions. This means that smaller advertisers using smaller advertising budgets, and with less knowledge on how to best target their ad campaigns, don't get the benefits bigger advertisers get.

1.2 The Practical Problem

Many things make advertising more efficient. One important aspect is targeting. When an advertisement is only shown to people who will engage it, everyone wins. The advertiser gets the results they want for fewer ad impressions and can therefore pay more per impression which increases the revenue the media gets per impression. The technology providers sitting in the middle get a percentage of each transaction and they too make higher revenue per impression. Even the users benefit as they are more likely to only see ads they actually care about. For this reason, the competitive advantage - perceived or real – often revolves around targeting.

Targeting gains can be achieved using one of two strategies, or a combination of the two. The first strategy revolves around analyzing the context of the impression and its relation to the characteristics of the prospective ad. In the simplest form, this might mean targeting an ad for a car brand to an impression shown on an automotive website but a more complex contextual targeting system might analyze the content of the ads in the system and learn what types of impressions work well with the ad content, possibly finding unexpected connections between them.

The second strategy is based on storing and analyzing the behavior of the user across the internet. To analyze user behavior, a system uses web beacons to track what pages the user visits and what actions they perform in any websites and generates a profile for the user. This profile is then connected to one or more segments that the ads are targeted towards. An example of a segment could be something simple like “men who like cars” or something a bit more sinister like “single, pregnant women living with their parents”. Approximately 81% of EU websites share their users’ behavior with one or more external tracking systems (Turcios Rodriguez, 2018).

A combination strategy analyses both the context and the user and uses the information gained from both to get to the final conclusion. Using both methods is especially useful when the impression originates from a website where not much information is known and impression context is not very useful, and when the user is either completely or relatively unknown because of tracking restrictions or other reasons they cannot be connected back to historical data.

1.3 Aim of the Project

This thesis is written to study improvements in targeting in the special case of native advertising, when the actual webpage content the ad leads to can be considered and analyzed before the campaign starts. The assumption is that this early analysis can be used specifically to improve the early results of a campaign before even any intermediate results can be measured and analyzed. Being able to improve the early results of a campaign is especially significant for small campaigns that gather only a little bit of actual data in the statistical sense before completing. As the commissioning company specifi-

cally targets smaller companies, as its customers having a system in place that is very efficient for smaller campaigns is important.

The purpose is to build a system that is able to take the campaign materials and make reasonable deductions from them that can then be used to make reasonable assumptions about the campaign's performance on different medias and for different types of users before the campaign even starts. To put it in another way, the purpose is to be able to generate "smart defaults" for the campaign properties within the bidding system. These smart defaults will then be improved upon during the campaign's run to produce best results for the advertiser.

The system to generate the starting parameters for the campaign takes all of the data of all the campaigns running in the system and improves its internal model further. The original plan is to have an offline update cycle running at specific intervals instead of updating the model as soon as new data comes in. The longer-term plan, that is outside of the scope of this thesis, is to have streaming updates to the model to make sure the latest information is used for new campaign creation at any given time.

1.4 Definitions

Buyer: Someone who has ads to show and money to pay for it. This can be individual advertisers, agencies or intermediaries who deal with the advertisers and agencies but show a unified front to the seller and/or exchange. The intermediaries are often called DSPs.

Clear Price: The final price for the impression. In first price auction, this is the price the winner of the auction bid, in a second price auction it is the second place bid plus one cent.

Click Through Rate: The rate at which users click an ad. This is simply the division of clicks/impressions in percent. In the early days of online advertising CTR was up to 44% but nowadays it is usually around 0.06% or so.

Conversion: Any target action the advertiser is optimizing for and measures. This can be a purchase at a webstore, a sign-up to a mailing list or any other kind of action the user takes that is considered valuable by the advertiser.

Cost-Per-Acquisition: The price the advertiser pays for each customer acquired, or conversion. This is usually abbreviated as CPA.

Cost-Per-Click: The price the advertiser pays for every user clicking on their ad. This is usually abbreviated as CPC and can be a purely mathematical value if the billing method is CPM in which case the CPC is calculated using the following formula: $CPC = \frac{CPM * CTR}{1000}$

Cost-Per-Conversion: See Cost-Per-Acquisition.

Cost-Per-Impression: The most common basic pricing method of online advertising. The advertiser pays for each impression their ad is shown to a user. The price is usually given as thousand impressions, cost per mille or CPM. With CPM pricing the advertiser is fully responsible for optimizing their campaign performance as the media and any intermediaries get paid the same regardless of the results.

CPA: Cost-Per-Conversion

CPC: Cost-Per-Click

CPM: Cost-Per-Impression

CTR: Click-Through-Rate

Demand Side Platform: The tool buyers use to construct their campaigns and that automates the bidding process for RTB auctions.

DSP: Demand Side Platform

Exchange: An ad server that can do an auction between different buyers/DSPs and select the ad to send to seller. Some notable exchanges are AppNexus and Google DoubleClick.

First Price Auction: The regular blind auction where each bidder does not know what the other bidders are bidding but chooses their bid completely independently. The high-

est bid wins and the bidder pays that amount. This type of auction requires the bidder to optimize the bid as there is no limit to how much the bidder has to pay, in case they win the auction. On the other hand, as the bidders don't know how much others bid it is a complicated problem to bid the minimum amount that still wins the impression.

Floor Price: The minimum price a media is willing to sell an impression in a placement they control. There are many reasons for setting a minimum price. For example, the perception of premium content.

Impression: The displaying of the ad in the user's browser. The ad impression is what is being sold and bought in RTB systems.

Native Advertisement: A form of advertising that aims to build brand awareness and interest by providing relevant information to the user. Also known as infomercial in other contexts.

Pay-Per-Click: A billing method where the advertiser pays for each time a user clicks the ad. In the pay-per-click (PPC) model, the click price is predetermined and is indicated as CPC (Cost Per Click).

Pay-Per-Conversion: A billing method where the advertiser pays for each conversion.

Placement: The actual ad frame or similar. Also known as the tag. This is usually a piece of JavaScript code that adds an iframe element to the page in question. The iframe contents are the actual ad that are loaded from the ad exchange.

Real Time Bidding: A digital ad delivery system where the ad impression on a webpage is sold in an auction just as it is being displayed (the "Real Time" in RTB), and the buyers in the auction decide the price they will bid as it happens based on any information they have about the user and placement of the particular impressions.

The RTB process in brief:

- A user browses a page with ads
- Ad placement code loads and requests an ad from an exchange
- Exchange requests bids from DSPs

- Each buyer decides what they want to bid for the impression based on different kinds of criteria including, but not limited, to user profile, ad profile, placement profile etc
- Exchange picks the highest bid(s) that meet some internal criteria (such as image size, text length, text content etc.) and sends it forward to the placement
- Ad is shown in the placement and a beacon pixel is fired to notify the buyer, usually with the final impression price.

RTB: Real Time Bidding

Second Price Auction: A more modern type of auction. It is used for example by Google (Google, 2019). The auction process is the same as in first price auction and the highest bid wins. The difference is the price the winning bidder must pay, which in this case is the first losing bid plus a minimum increase, usually one cent. In practice what this means is if there is one impression per sale, bidder A offers 15€ CPM and bidder B offers 13€ CPM, then A wins and pays 13.01€ CPM for the impression. For the bidders this means, they don't have to try to optimize their bid as they will always pay whatever is the "fair" price. They can simply bid whatever is the highest price they are willing to pay for the impression and sit back and expect savings. The actual effect to the average price paid per impression in popular placements is the price increases due to everyone bidding their maximum.

Seller: Whoever has a website with visitors and can show an ad in said page. Also referred to as the Media, the Publisher or the Site.

Semantic Analysis: A method of text analysis where the content of the text is defined using predetermined keywords and weights for each. The results of semantic analysis can be used to find related content and match ad campaigns to other similar ad campaigns as well as related content in a publication.

SSP: Seller Side Platform, the tool medias and other sellers use to offer impressions on their websites, apps or any other digital locations showing ads.

User: The person viewing a web page.

1.5 Limitations

This work is purely limited to predicting the probability of an impression resulting in a click. This information can be used in combination with price offer and other pricing information to improve system performance in an RTB environment but all of that is out of the scope here.

1.6 Ethical Considerations

A well-known possible ethical issue with all online advertisement systems is user privacy. As most targeting mechanisms are either based on or at least utilize user profiling data the concerns are very understandable. For this work no user identifying data was used at any point and the resulting predictive model is anonymous by nature and therefore follows GDPR rules.

2 FRAMEWORK

2.1 The Commissioning Party

The commissioning part is a Finnish advertisement technology company ReadPeak[1] founded in 2015 that specializes in online native advertising. The company has successfully been running its proprietary ad delivery system since 2016 but has identified that more modern approaches might improve the system's efficiency. This is becoming more and more an issue as the company has expanded first within Finland and lately to other European countries while ad delivery numbers have grown hundreds of percent per year. The company expects to use this work in deciding future direction.

2.2 Literature Review

Click probability estimation is considered a very important tool to improve profits of digital advertisement systems. Because of this, they have been studied extensively and

[1] https://www.readpeak.com

many different approaches have been proposed over the years. One thing common to all these approaches is that they take a selection of information around the ad impression and the click events themselves to produce the model that predicts click probabilities for future ad impressions. This information can be divided into four distinct categories as in (Chapelle, Manavoglu, & Rosales, 2015):

- Context features such as the query in sponsored search or the contents of the page where the ad will be shown
- Content features like ad image colors and contents (Haibin Cheng et al., 2012) or the textual content of the ads (Ciaramita, Murdock, & Plachouras, 2008; Hillard, Schroedl, Manavoglu, Raghavan, & Leggetter, 2010)
- User features like age, gender and language (Haibin Cheng & Cantú-paz, 2010)
- Feedback features like CTR of an ad-page pair (Chakrabarti, Agarwal, & Josifovski, 2008)

2.2.1 Shallow Models

A very common model for predicting click through rates is logistic regression (Chapelle et al., 2015; Mcmahan et al., 2013; Richardson, Ragno, & Dominowska, 2007). It is considered a simple and easily scalable model for large scale systems (Chapelle et al., 2015). A logistic regression model takes a, usually sparse, set of features belonging in the categories listed above and predicts a CTR for an ad on a page. Logistic regression based models benefit from feature engineering (Richardson et al., 2007) so using one generally requires extensive domain expertise as well as significant work up-front to get the best results.

To tackle the feature engineering problem more novel approaches have been proposed, among them factorization machines (FM) (Rendle, 2010) along with its further developed versions field-aware factorization machines (FFM) (Juan, Lin, Zhuang, Chin, & Lin, 2016) and field-weighted factorization machines (FwFM) (Pan et al., 2018). Factorization machines are similar to Support Vector Machines (SVM) (Cortes & Vapnik, 1995) but instead of a support vector they take advantage of factorization models and learn a latent vector for each feature to model interactions with other features. This property of factorization machines cuts the model complexity down from a naïve FM

implementation's $O(n^2)$ to $O(nk)$, where k is user specified dimensionality of the latent vectors and gives them the real-world capability to generalize from high-dimensional sparse input data without any specific feature engineering. The issue with factorization machines is only having one latent vector for each feature and thus any information of feature categorization, or fields, is lost.

Field-aware factorization machines (FFM) (Juan et al., 2016) are proposed to improve performance by training a latent vector for each feature-field pair in the input data, and then using the vector corresponding to the field of the paired feature for the inner product. FFMs have quite a bit better prediction accuracy as compared to FMs and they went on to win multiple CTR prediction competitions at the time of invention (Juan et al., 2016). On the other hand, the complexity of FFMs is quite high, $O(n^2k)$, rendering them difficult or impossible to use in a real-time high-dimensional system as feature count in a CTR prediction system can exceed millions.

Field-weighted factorization machines (FwFM) are proposed as a model to make field-awareness usable in production (Pan et al., 2018). In FwFM only FM-like generic vectors for features are learned but in addition a field-to-field weight matrix is learned to capture the differences in effect each feature pair has instead of the full latent vectors.

2.2.2 Deep Models

In addition to linear models deep neural networks have also been proposed. The benefit of deep learning when predicting CTR is the deep neural networks' capability to learn complex relationships between sets of features that is not possible with a linear model (Zhang, Du, & Wang, 2016). However, it is not feasible to directly use a DNN with high dimensional sparse data as the fully connected first layer would have number of features times number of neurons connections. To tackle the problem Factorization Machine supported Neural Network (FNN) was proposed (Zhang et al., 2016). FNN uses a factorization machine to reduce the dimensionality for the DNN. Using this combination, the overall results are improved from a regular FM.

Another approach for dimension reduction for DNN is Sampling-based Neural Network (SNN) also proposed by (Zhang et al., 2016). In SNN the sparse input vector is directly connected to the bottom layer of the DNN, but only positive samples are calculated for each sample in training which greatly reduces the calculations needed even though the number of connections in the bottom layer is still significant.

In addition a long short-term memory network has been proposed for CTR prediction (Deng et al., 2018). This model concentrates on capturing and predicting the temporal change in CTR of any given ad.

2.2.3 Hybrid Models

Google proposed a hybrid model (Heng-tze Cheng et al., 2016) which has a wide element (linear regression, LR) and a deep element (DNN) that work side-by-side to capture both low-order and high-order feature interactions. The proposed model works more accurately but still requires manual feature engineering work to reduce the dimensionality of the features for the wide element of the system.

To counter this the LR in the original wide & deep model can be replaced with a factorization machine to create DeepFM (Guo et al., 2017) which improves results, removes the need for separate feature engineering and also nicely provides the embedding layer for the deep part of the model. The shortcoming with DeepFM, as with a regular FM, is that it ignores the implicit information of the field interactions in the input data.

Another similar approach, xDeepFM (Lian et al., 2018) aims to address this shortcoming by introducing Compressed Interaction Network (CIN) to supplement FM and DNN in DeepFM. CIN resembles RNN in structure and aims to explicitly capture low order feature interactions. Adding the CIN component to the DeepFM model increases the complexity of the already fairly complex model which might render it infeasible for production use in certain environments

Deep Interest Network (DIN) (Zhou, Zhu, et al., 2018) attempts to capture the user's interest as well as the contextual features. Deep Interest Evolution Network (DIEN)

(Zhou, Mou, et al., 2018) expands on this premise by attempting to capture the evolution of the user's interest as well.

Attention Stacked Autoencoder (ASAE) (Wang, Liu, Xing, & Zhao, 2018) is an evolution of DeepFM that uses an Attentional Factorization Machine (AFM) in place of a regular FM that adds an attentional component that learns the interaction weights for the latent feature vectors of the FM using a perceptron. In addition to replacing FM with AFM the regular DNN in the deep layer is replaced with a stacked autoencoder which the authors consider a more effective choice for the deep part.

In Field-aware Probabilistic Embedding Neural Network (FPENN) (Liu et al., 2018) a probabilistic embedding layer is fed to a deep component and a quadratic component. The deep component will, as in any other proposed solution, capture implicit high-order feature interactions, whereas the quadratic component is essentially an FFM. The probabilistic properties of the embedding layer make the predictions more robust as they effectively work as a regularization for the deep and quadratic components.

2.3 Project Framework

The core of CTR prediction problem is a simple regression. We have a feature vector, usually sparse, that we use to predict the outcome of an interaction, in this case the probability of a click. An ad's CTR is basically the average probability of a click of each impression of the ad. Having a binary click/no-click classification is not enough as the real-world system we have the problem we are solving is selecting an ad from a set of candidate ads, and the reasonable way to make that selection is to predict how likely each one of them is to yield a click and pick the one with the highest probability.

To solve this issue different kinds of regression models have been proposed and used and our interest lies in choosing the one that is best suited to our use-case and input data.

2.4 Data Set

The data used for the system in question is gathered from two sources, historical bidding and engagement data, as well as semantic profile of the ad campaign material. This data is combined to the final dataset in the system to provide an individual data point containing any campaign context available, any impression context available and any user context available along with label.

The first part of the dataset, the campaign context, is the information about the campaign and ad that the impression relates to. This data contains, among other things, the following

- Semantic analysis of the campaign materials
- Campaign id
- Ad's past performance
- Ad id

The second part of the dataset, the impression context, is the combined information about the impression itself. This data contains, among other things, the following.

- Timestamp
- Ad placement id
- Site domain
- Media name / ID
- Page URL
- Bid amount

The third part of the data is the user context, any information that is available about the user who is browsing the web page in question. This data contains, among other things, the following.

- Country where the request originated from
- City where the request originated from
- The type of device used

In addition, each sample in the dataset has a label indicating whether the impression resulted in a click or not.

2.4.1 Undersampling

The full collected dataset is very large and highly imbalanced, approximately one positive sample per 2000 negative samples. For these reasons, the dataset is undersampled by taking all of the positive samples of a time period and 1 in 100 negative samples.

2.4.2 Dataset Statistics

The full dataset consists of the data of 251569051 impressions delivered by the target ad system to a set of websites over a course of eight consecutive days, from April 13 2019 to April 20 2019. The rate of positive samples in the full dataset is 0.04693%. After undersampling there are 3773380 samples with a positive rate of 4.69478%.

Each impression, or row in the data, consists of a number of fields that spread out to 13887 features. To illustrate the variance of the data there are 761 different site ids, 176 countries and 10162 cities.

2.4.3 Dataset Evaluation, Preprocessing and Feature Extraction

The dataset consists of ad impression data from an eight-day span in April 2019. Each row of data has all of the known information about the ad impression as well as whether or not the impression resulted in a click. There are dozens of features and some of those features are numerical, like bid amount, and some of them are categorical, like the type of device the user is using. Some features, or columns, in the source data are either unused or deemed irrelevant for a prediction task. A day's worth of usable raw data amounts to multiple gigabytes and it is very imbalanced. A CTR of 0.5% (which is higher than average in digital advertising) would yield 200:1 ratio of imbalance.

The evaluation of the dataset consists of finding the usable columns and deciding whether or not they should be dealt with as numerical or categorical data. For most cat-

egorical features the decision is easy (like the abovementioned device type), but for some it is not so clear.

For the preprocessing portion a python program is written to drop unused features, scale numerical features and one-hot encode all categorical features. As class imbalance tends to skew results it will be mitigated by undersampling the negative samples. Finally, as millions of rows of thousands of columns is a very large dataset the training data will be converted into a sparse matrix.

For the training process itself the eighth day data will be separated out as the test set and the models will be trained using seven full days worth of data.

2.5 Value for The Commissioning Party

This thesis consists of a thorough study into a predictive model that can be used to improve the ad delivery system's performance. The results of this work can and will be used to help decide on methods and systems to improve said performance by the commissioning party.

3 RESEARCH METHODS

3.1 Research Design

The research will be conducted in three steps, each step feeding information to the following step. The first step is pre-processing the input data. The data coming in is simple JSON formatted data directly from the ad delivery pipeline and it needs to be transformed to a format that is directly usable by the algorithms we are evaluating. The second step is offline evaluation of the three models chosen for this research using the dataset created in the first step. The third and final step is implementing an online system that is able to do the ad selection based on the predictions of each of the models and measuring their real-world performance.

3.2 Primary Tools

The research consists for a large part of writing actual program code and evaluating results. The code is written in Python in Jupyter notebooks using some well-known helper libraries, Pandas and TensorFlow/Keras. It is executed and the data is stored in AWS using systems like EC2, SageMaker and S3.

3.2.1 Pandas

Pandas is an open-source library providing high-performance, easy-to-use data structures and data analysis tools[2]. For this research it will be used for the data preparation tasks, namely scaling numerical data, one-hot encoding categorical data and dropping unused data.

3.2.2 TensorFlow

TensorFlow[3] is an open-source software library for dataflow programming across a range of tasks. It is a symbolic math library and is also used for machine learning appli-

[2] https://pandas.pydata.org/
[3] https://www.tensorflow.org/

cations such as neural networks. In this research it is used for the model training and subsequent predictions, both for offline and online evaluations. Using TensorFlow allows having very comparable environments for all of the models for offline tests as well as easy transition to production.

The main TensorFlow library also contains Keras, a higher-level neural network API. The actual code in this work is written using Keras.

3.3 Model Evaluation Methods

3.3.1 Log Loss

Log loss, also known as cross-entropy loss, is basically cross entropy between two probability distributions. In our case it can be considered the cross entropy between the impressions being predicted to result in clicks and the actual result.

Log loss gives us understanding how well the models can predict the correct CTR for an individual ad impression. A lower log loss number means more accurate predictions.

3.3.2 ROC and ROC Curve

ROC stands for Receiver Operator Characteristic for historical reasons, but in machine learning context means the relationship of true positive rate and false positive rate when the discrimination threshold is varied. A ROC curve of a predictive model looks like an arc from bottom left corner of the graph to top right corner of the graph, and a direct line between the corners can be considered as purely random regressor or classifier. The closer to the top left corner the curve gets and the smoother it is the better the predictive capabilities of the model are.

3.3.3 ROC AUC

AUC means Area Under Curve and when used in conjunction with ROC it means the area under the ROC curve. The most interesting and important property of ROC AUC is

that it equals the probability of a random positive sample being predicted a higher probability than a random negative sample (Fawcett, 2006). As one of the goals of the prediction system is to be able to order the ads by likelihood of a click this is a very good measurement for this task.

3.3.4 Prediction Histogram

To visualize the results of each model's predictions in a tangible way the predicted values are plotted on a histogram. In the histogram positive and negative samples are plotted separately. This illustrates how well the model is able to separate positive samples from negative samples and whether they cluster in any specific level of predicted CTR.

3.4 Models Evaluated

The offline model evaluation part of the research consists of evaluating the three regression models that were selected from all the available options based on the literature review. The models will be evaluated and compared using the following criteria:

1. Prediction accuracy using log loss and ROC AUC.
2. Training time. A shorter training time means cheaper and more timely updates to the prediction model.
3. Prediction time. A shorter prediction time means the system requires less time for making individual predictions (an important quality in a real-time bidding environment) as well as indicates that less resources is required to making each prediction which subsequently means that running the prediction service is cheaper.

3.4.1 Naïve Model

As a baseline a naïve model is used. This model is not really predictive as it simply uses the historical data and assumes what the past performance was will also be the future performance. In practice the naïve model will take any ad's historical CTR and consider that as the future CTR. This has a couple of repercussions. First, any new ads will basically have zero "predicted" CTR and any samples with positive label will produce a

large loss. Second, as the true overall CTR is fairly limited the "prediction" will never really be that close for positive samples again resulting in fairly large loss.

3.4.2 Linear Model

The first and simplest predictive model is a linear regressor

$$\hat{y}(x) = w_0 + \sum_{i=1}^{p} w_i x_i$$

The linear model can be improved by adding a regularization term to the objective function to reduce overfitting (James, Witten, Hastie, & Tibshirani, 2013). As can easily be deducted from the formula the computational complexity of a linear model is linear $O(n)$, and its number of parameters is also linear only depending on the number of features in the input vector. Since the linear model considers each feature separately from all other features any interactions, simple or complex, between features are ignored. The working assumption is that this model is the fastest to train and predict with but produces the worst results.

For this thesis the l2 regularization enabled ridge regressor from scikit-learn[4] was used.

3.4.3 Factorization Machine

A factorization machine (FM) learns latent vectors for each feature that describe the feature's interactions with all other features in the input data. To put it in other way each of the latent vectors are a dense representation of the sparse relationships of one feature and all other features. This can be formulated as

$$\hat{y}(x) = w_0 + \sum_{i=1}^{p} w_i x_i + \sum_{j_1=1}^{p} \sum_{j_2=j_1+1}^{p} \langle v_{j_1}, v_{j_2} \rangle x_{j_1} x_{j_2}$$

where v_{j_1} and v_{j_2} are the latent vectors of features j_1 and j_2 respectively. This form of FM is computationally much more complex, $O(n^2 k)$, but can be further developed into the form

[4] https://scikit-learn.org

$$\hat{y}(x) = w_0 + \sum_{i=1}^{p} w_i x_i + \frac{1}{2}\sum_{f=1}^{k}\left(\left(\sum_{j_1=1}^{p} v_{j_1,f} x_{j_1}\right)^2 - \sum_{j_2=1}^{p} v_{j_2,f}^2 x_{j_2}^2\right)$$

where k is the number of dimensions of the latent vectors. The latter form of FM is only $O(nk)$. The number of parameters in a FM can be deducted to be $p + pk$.
The working assumption is that FM performs significantly better than a LR with little to no feature engineering while still performing reasonably well computationally.

As with linear regressor, the results of an FM can also be improved by adding a regularization term to the objective function (James et al., 2013; Rendle, 2010)

The Factorization Machine used in this thesis is listed in appendix A

4 RESULTS

The experiments listed below were run locally on a Macbook Pro laptop. The laptop has 2,9 GHz Intel Core i5 processor, 8GB of memory, and is running macOS Mojave (version 10.14.4). No GPU acceleration was used.

4.1 Data Preparation

The data is pushed to a data lake through AWS Kinesis Firehose and it is indexed using AWS Glue. From data lake it is mined using AWS Athena. The Athena run selects 8 full days worth of data, joins impressions to clicks to get the proper label for each sample and down samples the negative samples 1 to 100. This process produces a CSV file that is 4-5 GB in size

The CSV file is then copied to the test laptop and read into a Pandas dataframe from which unused columns are dropped, values are sorted by timestamp, some columns are renamed to better reflect their purpose and any duplicates are dropped. After the initial cleanup the labels are separated out to a different file to make it easier to use them separately afterwards, and then all numeric features are scaled appropriately.

The final stage of data preparation first turns the dataframe into a sparse one, one hot encodes all categorical features and finally converts the whole dataset into a SciPy CSR matrix which is stored to disk. The resulting file is roughly 60MB.

The reading of the original CSV file takes about 10 minutes with dozens of gigabytes of memory used while the npz file can be read in a matter of seconds and only takes a few gigabytes of memory.

4.2 Linear Model Hyper Parameter Study

A linear model itself does not have any hyper parameters, but when the regularization term is added its alpha acts as a hyper parameter. As Table 1 shows the regularization

parameter value does not make much of a difference. Only with value of 10 there seems to be any difference.

Table 1 - Linear Model Regularization Study

Regularization	10	1	0.1	0.01	0.001	0.0001
ROC AUC	0.809836	0.809840	0.809841	0.809841	0.809840	0.809841

For the rest of the thesis the linear model is trained with regularization alpha of 0.01.

4.3 Factorization Machine Hyper Parameter Study

Factorization machine has two hyper parameters that can be tuned to improve the results: regularization and latent vector size. The effect of different values for these parameters were studied. In addition, different loss functions and optimizers were compared. And finally, the development of prediction accuracy with increasing number of training epochs was tested and plotted.

4.3.1 Regularization

Regularization tries to counter overfitting by reducing the variance of the model and thus reducing the effect of any outlier samples that don't really represent the characteristics of the data. Applying too much regularization will, on the other hand, make actual characteristics to disappear. It is therefore important to choose the level of regularization correctly for the specific data at hand.

Four levels of regularization were tested with three different latent vector sizes. Table 2 shows that the results improved consistently with all different sizes up to regularization value 0.001 after which they started degrading indicating overfitting. A slight exception here is size 4 which still improves until the next regularization step. The difference is likely caused by the model with low latent vector size not being very prone to overfitting regardless of regularization.

Table 2 – FM Regularization vs ROC AUC with different latent vector sizes

Regularization	Size 4	Size 16	Size 24	Size 32
0.1	0.80096	0.80941	0.81792	0.80960
0.01	0.82025	0.81507	0.81200	0.81505
0.001	0.81405	0.82099	0.81599	0.81858
0.0001	0.81895	0.81604	0.81617	0.81593
0.00001	0.82035	0.81607	0.81683	0.81756

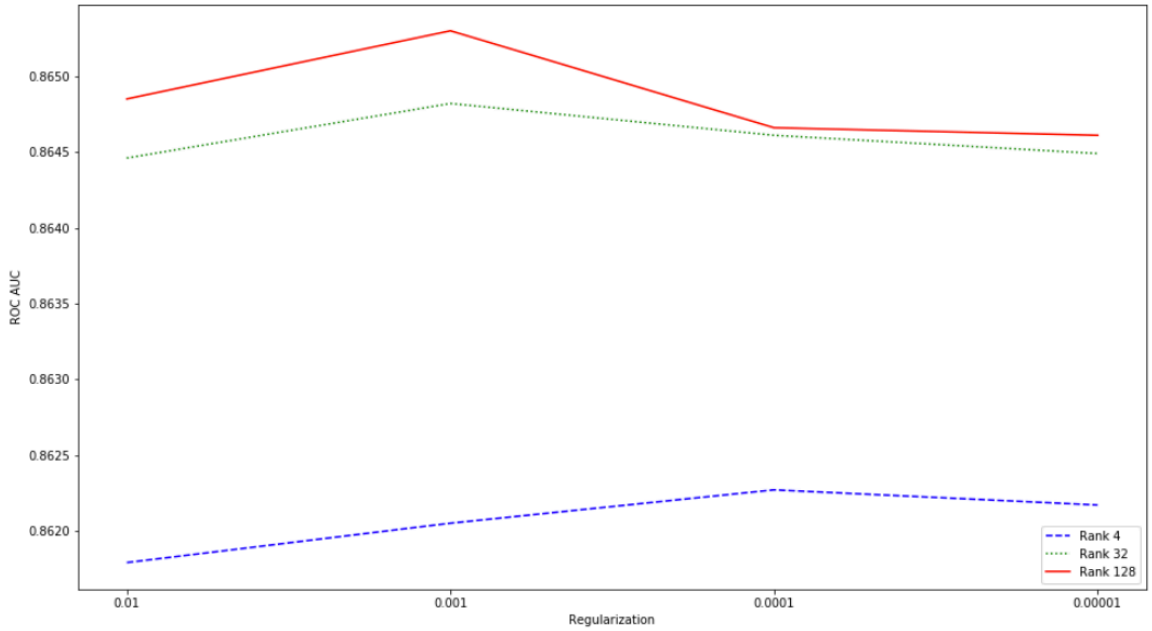

Figure 1 - FM ROC AUC with different regularization values

4.3.2 Latent Vector Size

The latent vector layer is the layer of neurons in FM that learns the interactions between features in the input. There is one latent vector for each feature. Higher number of neurons in that layer will mean more granularity for the embedding but on the other hand it increases both probability to overfit as well as training and prediction time as there are simply more calculations to make per sample.

Table 3 – Latent Vector Size vs ROC AUC and Log Loss

Latent Vector Size	ROC AUC	Log Loss
4	0.86205	0.11507
8	0.86386	0.11472
16	0.86435	0.11460
32	0.86482	0.11440
64	0.86494	0.11437
96	0.86486	0.11433
128	0.86530	0.11428

From Table 3 we can see that latent vector size definitely has an effect, but it is not very dramatic. The improvement from size 4 to size 8, and to a lesser degree all the way to 32, is somewhat significant but after that it seems that the gains are more or less negligible. Even though log loss (Figure 3) does keep improving area under ROC does not (Figure 2)

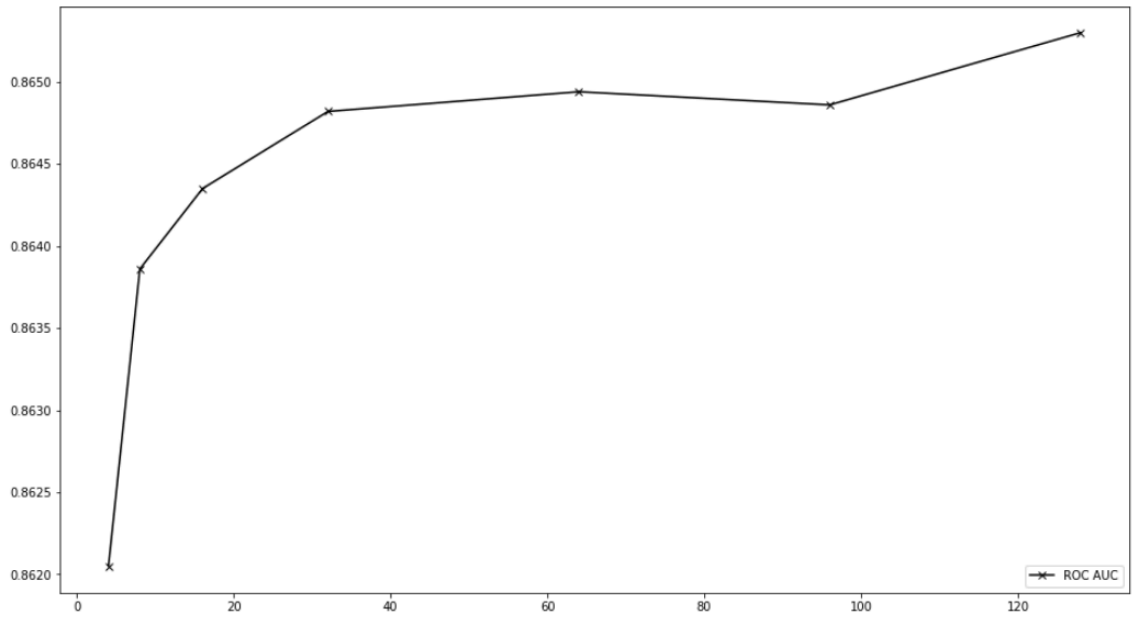

Figure 2 – Latent Vector Size vs ROC AUC

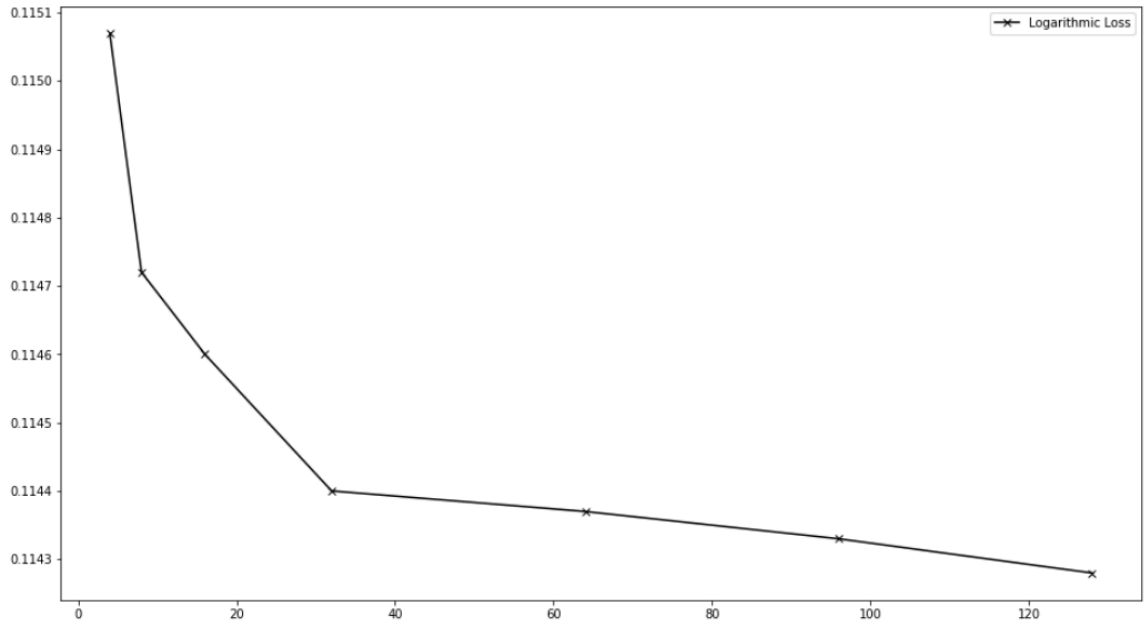

Figure 3 – Latent Vector Size vs Log Loss

Latent vector size does, however, have a fairly large effect on training time which climbs linearly when size gets bigger as is clearly demonstrated in Figure 4.

Table 4 – FM Training time with different latent vector sizes

Latent Vector Size	Total Training Time	Epochs	Training Time Per Epoch
4	26:49	9	2:59 / 179s
8	35:57	11	3:16 / 196s
16	39:59	11	3:38 / 218s
32	43:26	10	4:21 / 261s
64	52:40	10	5:16 / 316s
96	58:03	9	6:27 / 387s
128	76:43	10	7:40 / 460s

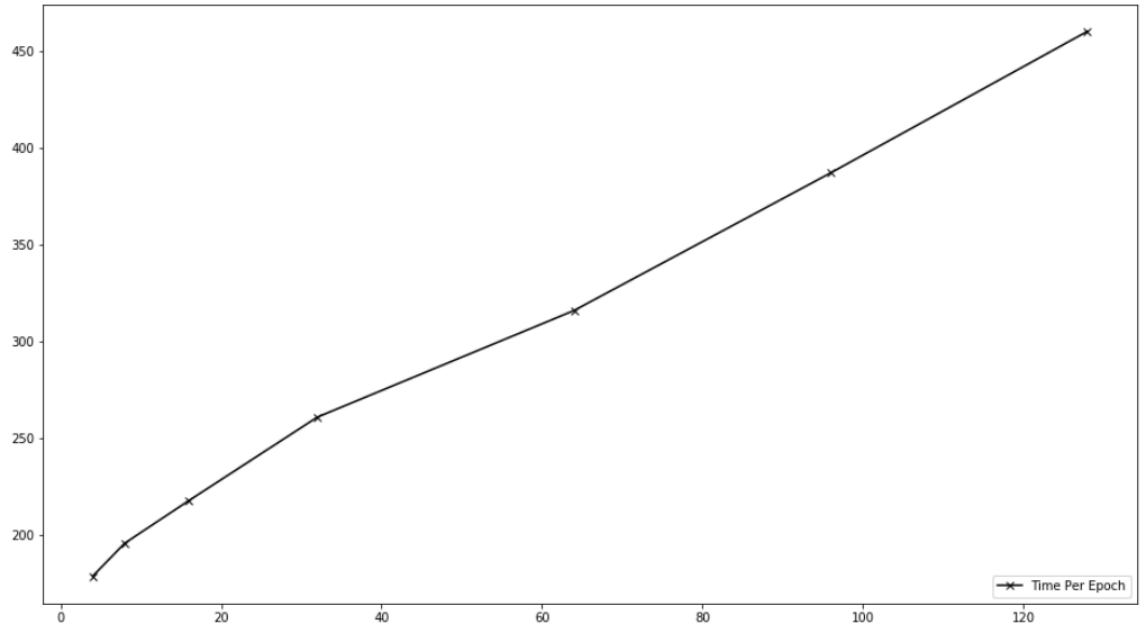

Figure 4 – FM Training time with different latent vector sizes

The conclusion we can draw from the development of prediction quality and training time increase is that it is not reasonable to make latent vector bigger than 32 as beyond that we're seeing diminishing returns.

4.3.3 Loss Function

A loss function is a function for calculating a penalty, or loss, for the training process for the error between the predicted and true label. During training the loss is used to adjust the gradient for the optimizer, or in other words the optimizer tries to minimize the loss function.

Four different loss functions were tested for this work. Table 5 shows that all the loss functions produce results that are relatively close to each other although mean square error seems to perform slightly better than the others.

Table 5 - Loss Functions

	Log Loss	ROC AUC
Mean Square Error	0.15599	0.82604
Log loss	0.15716	0.82362
Mean Squared Logarithmic Error	0.15710	0.82015
Poisson	0.15605	0.82561

4.3.4 Epochs

When training the model, the training data can be run through the process multiple times. Each run of the data is called an epoch. Multiple epochs allow for more iterations of the main learning algorithms without extra data.

Testing for this work indicates that with the data in question the process plateaus around 5 epochs. There is a slight bump back up after which no changes in log loss are seen. It can be concluded that running more than 5-6 epochs will not improve results in a meaningful way.

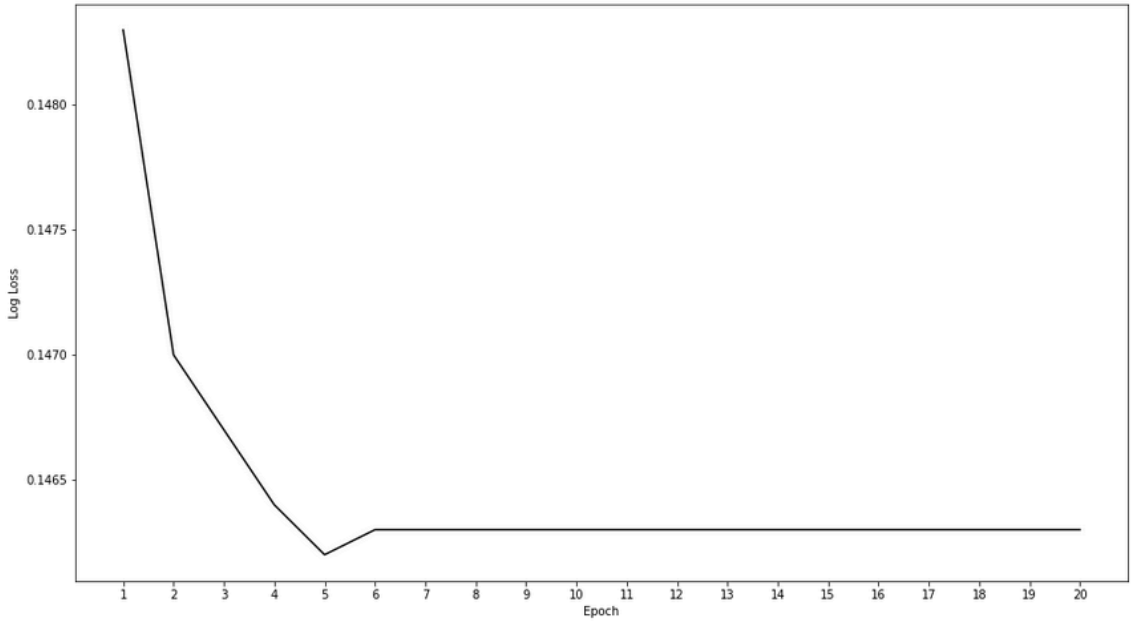

Figure 5 - Epochs vs Log Loss

4.3.5 Optimizer

Optimizer function is the mathematical function that adjusts the weights and other variables in a model to minimize the loss function. It works iteratively and runs between batches.

The optimizer in TensorFlow is basically the variation of stochastic gradient descent used for optimizing the loss function. Four different optimizer functions were tested, Adagrad (Duchi, Hazan, & Singer, 2011), Nadam (Dozat, 2016) as well as Adam and Adamax (Kingma & Ba, 2014)

Table 6 – Optimizer comparison

	Adagrad	Adamax	Nadam	Adam
Log Loss	0.15663	0.15499	0.15594	0.15719
ROC AUC	0.82231	0.82708	0.82599	0.82296
Training Time Per Epoch	8:02	8:07	8:03	7:46
Epochs	15	9	9	8

It can be clearly seen from Table 6 that Adagrad is quite a bit slower to converge than the other optimizers, but the other three produce fairly similar results in fairly similar time.

4.4 Model Comparison

The hypothesis was that a factorization machine is superior to any other relatively simple model. To verify it three different models were considered using the same data. First, a naïve model that simply predicts the future using past performance – namely CTR. Secondly, a linear regressor using L2 regularization – also known as ridge regression. And thirdly, a factorization machine.

4.4.1 Naïve Model

The naïve model is only moderately better predictor of future performance than random chance as can be seen in Figure 6. This can be attributed, at least in part, to it taking some time to gather enough data for each ad for the prediction to be anything more than purely random as any contextual information is ignored. The area under the RO curve at 0.7085 is a lot lower than for the other models tested.

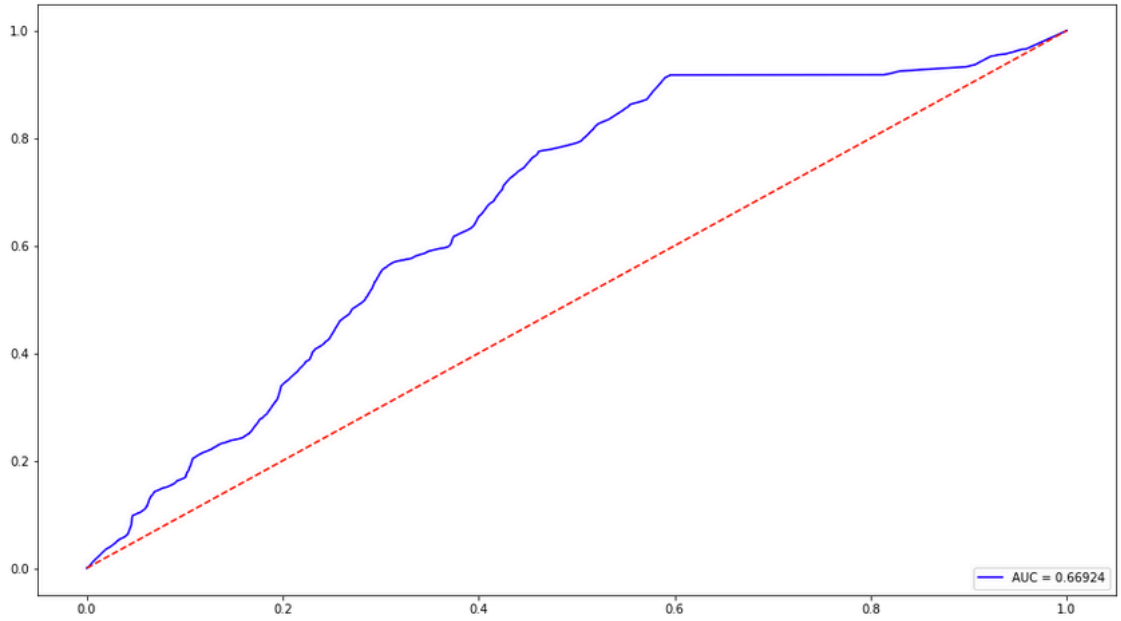

Figure 6 – Naïve Model ROC Curve

Another sign of limited predictive capability is the relatively large number of positive samples that have a predicted (or in this case actual) CTR of 0 as can be seen in Figure 7. The first clicks of any new ad will have CTR 0 but any actually predictive system should never predict their CTR to be that.

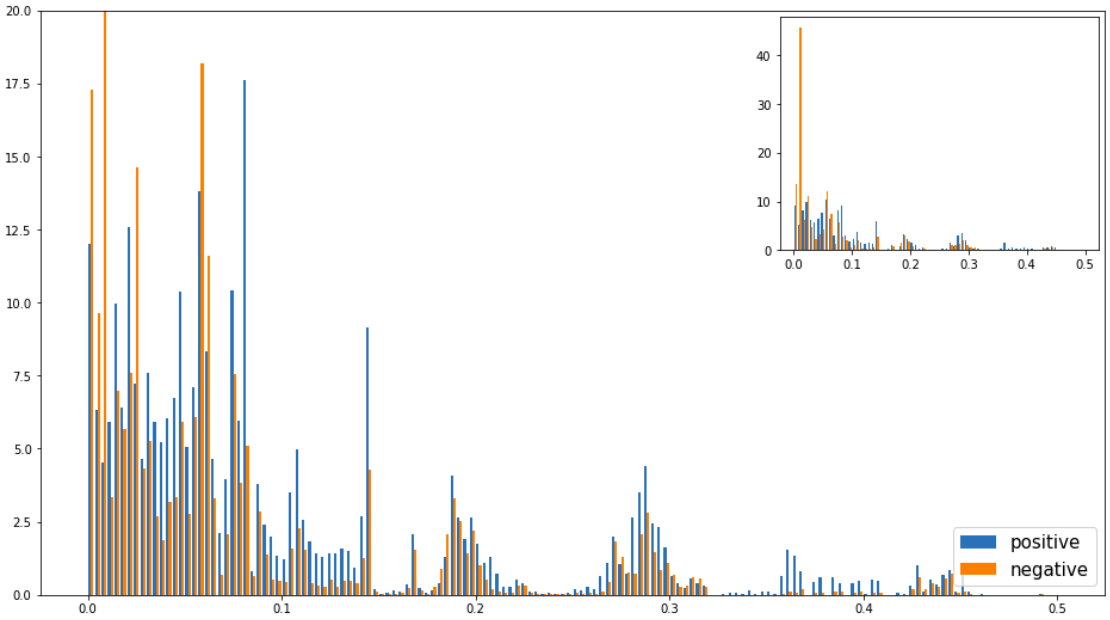

Figure 7 – Naïve Model histogram

4.4.2 Linear Model

Ridge regression improves from the naïve model a lot. The ROC is a nicely shaped curve arcing towards the top left corner as can be seen in Figure 8. The area under the curve is also significantly better at 81%. An interesting observation can be noted at the bottom left corner of the graph: the curve has a very clearly visible nudge. This seems to indicate that even though a lot of the negative samples are clustered to very low predicted CTR there is a significant number of positive samples in the low predicted CTR area as well.

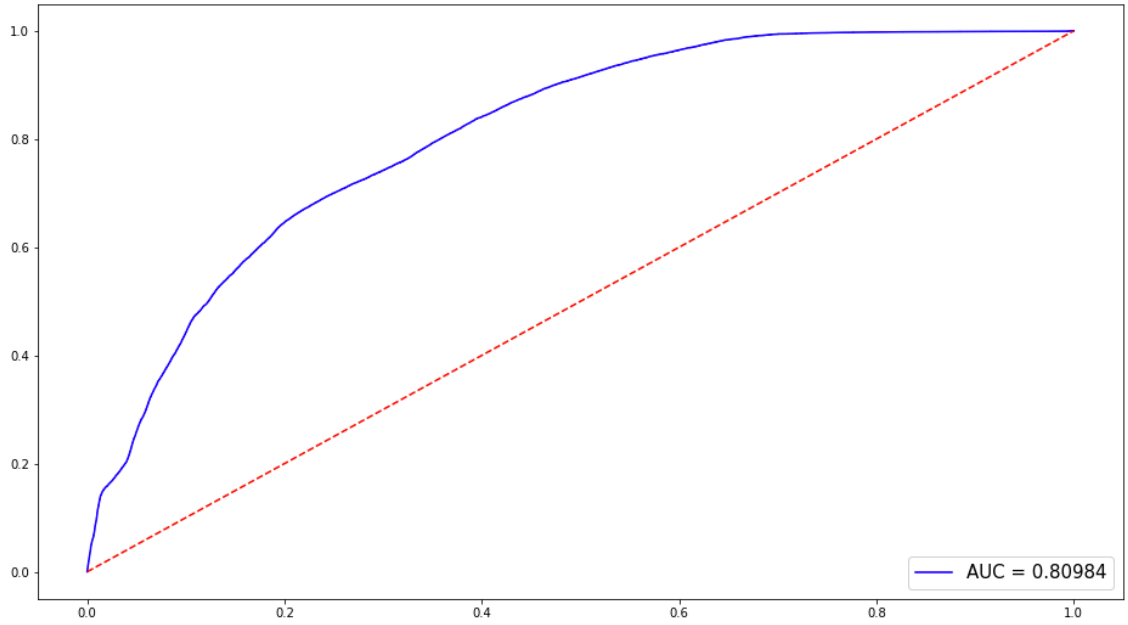

Figure 8 – Linear Model ROC Curve

Careful examination of the histogram (Figure 9) reveals that there are indeed some negative samples at almost the highest CTRs predicted. On the other hand, only a cursory glance reveals that most of the negative samples have a very low predicted CTR and therefore are likely to have lower predicted CTR than any positive samples. It would also seem to be feasible to implement a filter in the production system that would simply not show any ads with a very low predicted CTR and that would presumably reduce the number of negative outcomes by a significant amount.

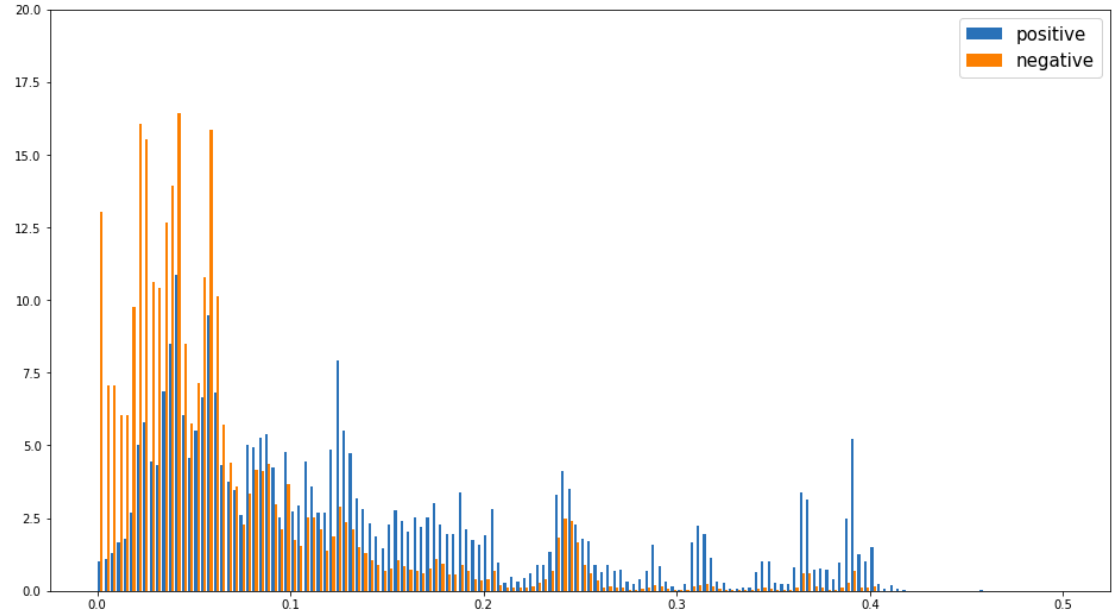

Figure 9 – Linear Model histogram

4.4.3 Factorization Machine

Factorization machine produces a very smooth ROC curve as can be seen in Figure 10. It is notable that the curve does not have any dips as ridge regression but instead it reaches full 1.0 at a relatively early stage. This would indicate that any negative samples were not predicted to have a higher CTR than a certain threshold.

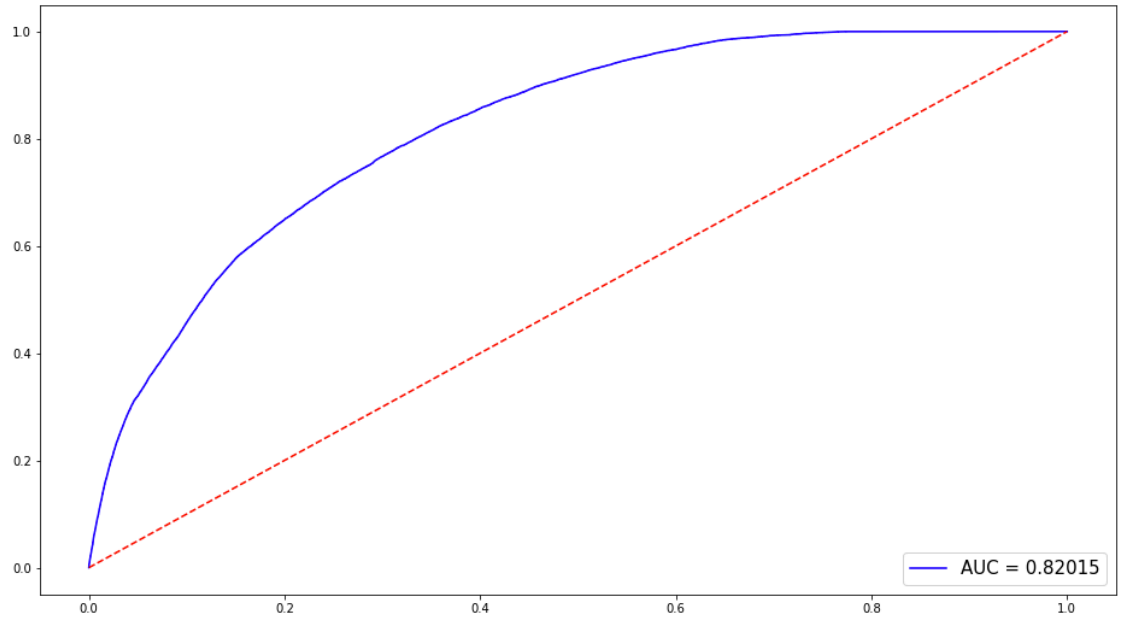

Figure 10 – Factorization Machine ROC

Another thing to note is how a very large portion of negative samples get very low predicted CTR (Figure 11). It opens up a possibility of using this predictor as a simple filter to filter out a lot of certainly negative samples in a case of individual impression prediction.

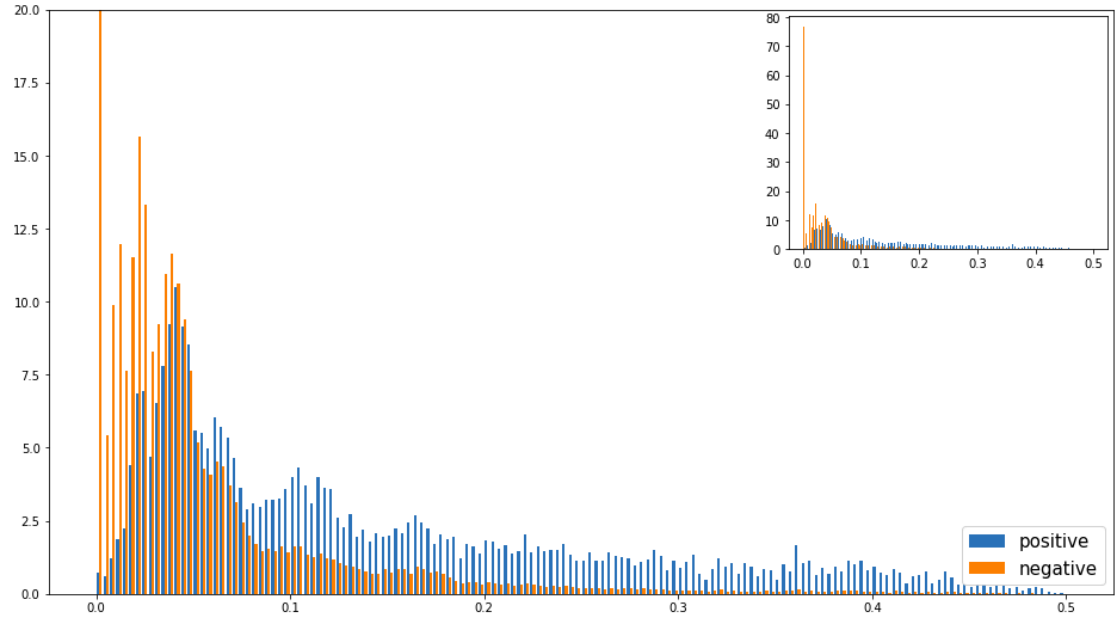

Figure 11 – Factorization Machine histogram

For this evaluation embedding vector size of 32 and regularization value of 0.001 were used.

4.4.4 Overall Comparison

To see any differences between the models were compared directly. From Table 7 it is easy to see that factorization machine training and prediction are much heavier operations than with the linear model. As the naïve model really only uses historical data there is no training and prediction needed.

The difference in quality of prediction is very clear. The naïve model is very bad when compared to the other models, with log loss being double and ROC AUC more than 15% lower than either of the other models. The simple conclusion is that either linear model and factorization machine would be a significant improvement over the naïve model.

The comparison between linear model and factorization machine is also fairly straight forward when purely considering prediction quality. FM has almost 1.5% higher ROC AUC and over 16% lower log loss. On the other hand, linear model's training and prediction times are significantly lower so the decision whether to use one or the other then comes down to the cost of computing.

For completeness a factorization machine was also trained with only one day's worth of data after which it was tested using the same test set as the other models. As can be seen in Table 7 faster training time achieved with less training data comes with a severe penalty in prediction accuracy.

Table 7 – Model evaluation comparison over log loss, ROC AUC, training time and prediction time

	Log loss	ROC AUC	Training Time	Prediction Time[5]
Naïve	0.26653	0.66991	n/a	n/a
Ridge Regression	0.16578	0.80984	38m45s	0.133ms
Factorization Machine	0.15617	0.81985	62m24s	2250ms
FM w/ One Day Data	0.18728	0.77487	8m14s	2190ms

The combined ROC curve (Figure 12) shows that even though the difference is not big factorization machine is slightly ahead of linear model all the way through while the naïve model is basically all over the place.

[5] Using 20 samples, averaged over 1000 runs with different data each run

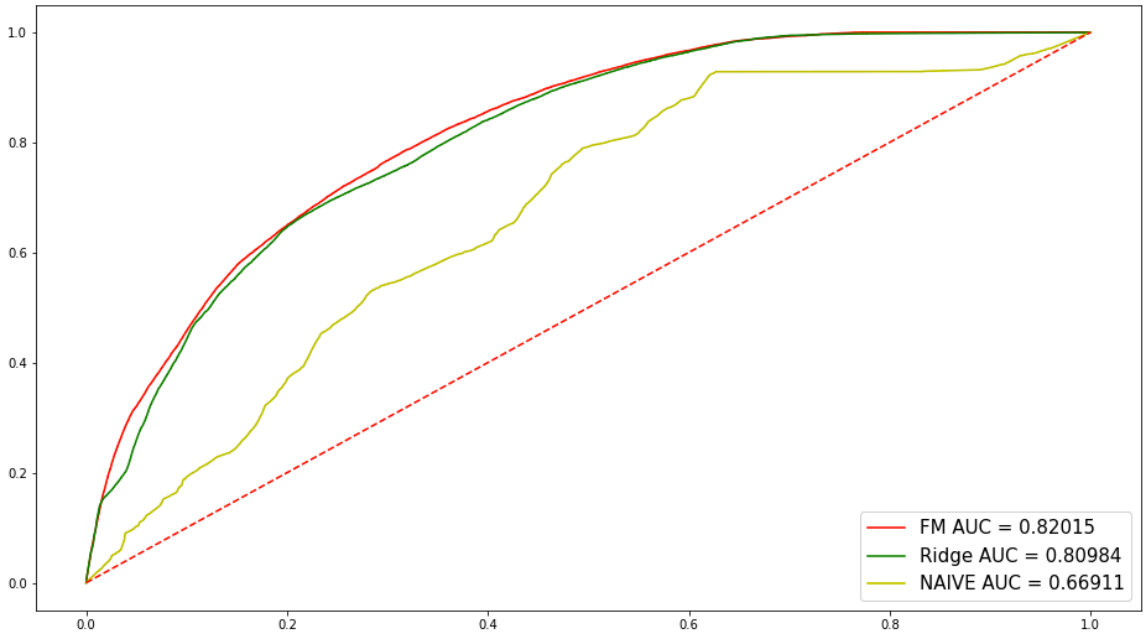

Figure 12 –Comparison of the ROC curves of all evaluated models

4.5 Prediction Quality Degradation

In the real-life use-case the CTR predictor only predicts data that is further and further away in time from the data it was trained with. This means that the model must either be retrained regularly, or new data must constantly be added to make sure the model is able to predict accurately. To see how the model's prediction quality degrades over time, and to test how often the model needs to be retrained, the test set was divided into 188 equal size parts that were then predicted and measured separately and the results were plotted in a graph.

The hypothesis was that the prediction quality will degrade over time as the data used to train the model gets older in relation to the inputs used for the prediction. The measurements do not support this as can be seen in Figure 13. Instead of degrading quality it seems that there is a cyclical nature to the output quality where the quality drops sharply when a new day starts and then actually improves towards the end of the day. This effect is very clearly visible around test batch 10 and batch 65 in Figure 13.

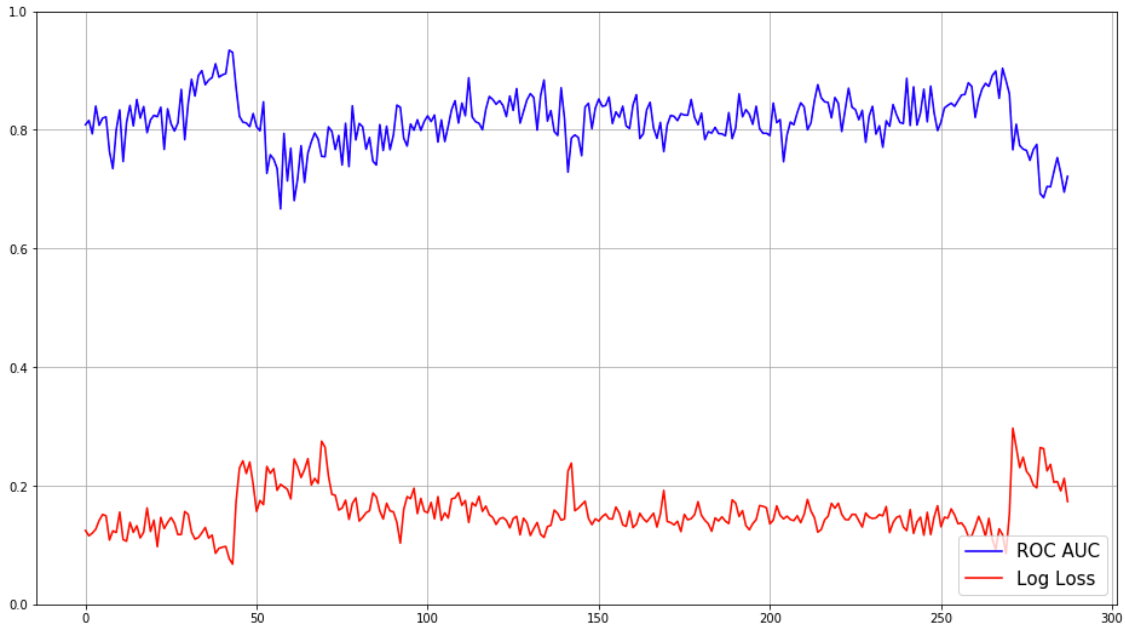

Figure 13 – Log Loss and ROC AUC development over time

It is unclear why this is but the working theory is that there are multiple small, or in other words campaigns with small daily budget, campaigns that get restarted when midnight rolls over and since there is less data about them in the training set the quality of predictions for them is lower. As an important factor in this theory is that these campaigns are small, they then also run out of budget quickly and that allows the prediction quality to improve again.

To find out why model degradation is slower than expected and why there is a large discontinuity near midnight more research is needed.

4.6 Field Drop-out

Most original features of the input data are categorical. They are converted to numeric features by one-hot encoding. To distinguish a feature in the data set used for training and a feature in the original data in this section we will call the original data features fields.

To find out how big an effect each of the fields have on the final result each field and pair of fields was zeroed out one by one from the test set and ROC AUC was calculated from the result. Figure 14 is the resulting heat map and shows each of the fields in both

x and y axis. The training data and test set includes one hot encoded month index, but as it is the same for every sample there is only one feature and it is always 1. For this reason, dropping it will cause the results to be very poor and those results were left out of this graph otherwise it would have shadowed any other results we get from dropping features.

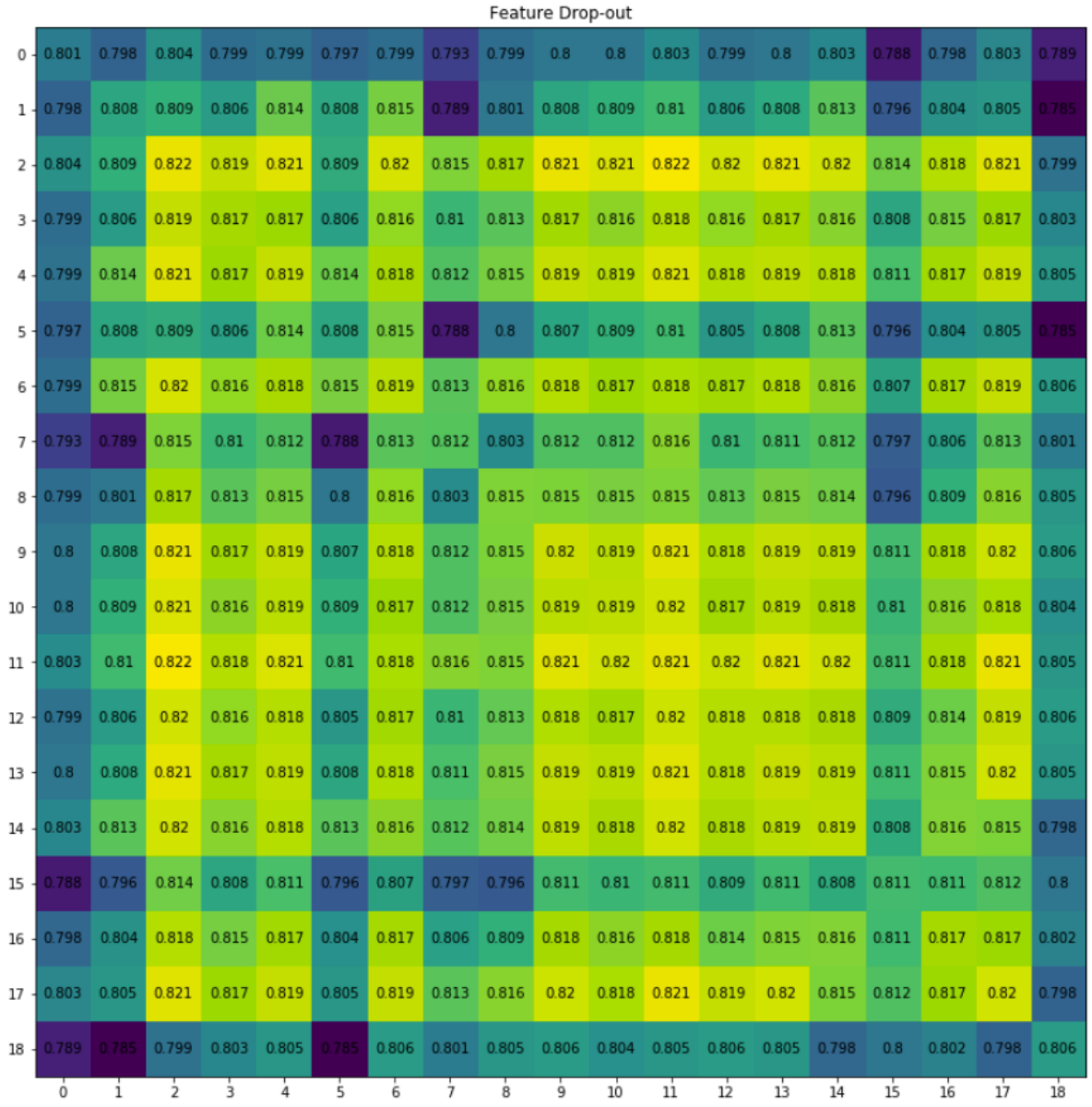

Figure 14 - Feature Drop-out

The baseline ROC AUC for the full set of fields and features is 0.81985 and an interesting finding is that dropping certain features, like feature 2, ROC AUC actually gets higher than baseline. In addition, some fields can be labeled as very important and some as not important at all from the map. The rows 0, 1, 5, 7, 15 and 18 generally have much

lower ROC AUC than the baseline indicating that the respective fields make a big difference in correctly predicting the outcome. On the other hand, 2, 4, 6, 9, 10, 11, 12, 14, 16 and 17 seem to have little to no, or even negative, effect on the end result.

4.7 Class Balancing

The positive and negative classes in the input data are very far from a balanced distribution. The dataset was down sampled, keeping all positive class samples and a random selection of 1/100 negative class samples. The main reason for that was to make the dataset easier to operate on, with the assumption that with such wide imbalance the sampling would have little to no effect on prediction quality. Since undersampling was not done evenly across classes the classes were more balanced in the down sampled dataset with one positive sample for approximately 20 negative samples.

It is suggested in literature that with imbalanced classes the results might not be optimal (Japkowicz & Stephen, 2002; Oommen, Baise, & Vogel, 2011). To determine what effect class imbalance might have with the dataset we are working with the dataset was further undersampled by taking all the positive class samples and random 1 in 10 negative class samples creating a dataset that had all of the positive class samples from the original dataset but only 1 in 1000 of the negative class samples. The undersampled dataset has 339900 samples with a positive rate 45.93996%.

The more heavily undersampled dataset was tested using the same grid search method using different regularization alphas and latent vector sizes as the larger dataset.

Table 8 - Balanced Classes Dataset ROC AUC

Regularization	Size 4	Size 16	Size 24	Size 32
0.1	0.81787	0.81970	0.81798	0.81794
0.01	0.81902	0.81908	0.81955	0.81863
0.001	0.81943	0.82061	0.82186	0.81994
0.0001	0.82177	0.82187	0.82134	0.82090
0.00001	0.82116	0.82135	0.82152	0.82110

As can be seen from Table 8, the best result for this smaller dataset is 0.82187 that was achieved with latent vector size 16 and regularization alpha 0.0001. This is slightly better ROC AUC result than with the larger dataset 0.82099 (Table 2). It is unclear whether this improvement is due to the classes being more balanced.

A more notable difference is that as the dataset is significantly smaller training takes significantly less time. Training for the best result of the smaller dataset took 12 minutes 53 seconds with 13 epochs which means just about a minute per epoch. Training the best result of the larger dataset too 35 minutes and 56 seconds with only 8 epochs which means about 9 minutes per epoch. The ratio of the epoch times thus seems to follow the amount of data linearly.

4.8 Reflection on Results and Outcome

The most surprising result is that the naïve model is not very good at predicting anything. It is understandable that it does not generalize at all to new campaigns and ads but on the other hand most of the impressions are with ads and campaigns that already have history and it was expected that history repeats itself to some extent. These results seem to indicate that both linear regressor and factorization machine should yield much better results in a live system than the naïve model.

It is possible that the weak results of the naïve model result in part from the different type of model and its results don't lend themselves very well to this type of comparison. If that is the case the online test results should indicate this very clearly.
Another somewhat surprising result was that the linear model performed quite well and lost to factorization machine by fairly narrow margin. The difference to naïve was very large indicating that a properly tuned linear model already is quite a bit better than no predictive capabilities at all. Considering that the linear model is a lot faster to train as well as get predictions from it could be considered as the production model in any environment where computing resources are either expensive or sparse.

The factorization machine results are not surprising at all and are quite well in line with the expectations laid out by the literature around the subject. FM shows a very con-

sistent improvement over the surprisingly good linear model and it seems to be a very robust tool in predicting CTR. The most effective features and the optimal amount of data must still be studied though.

The field drop-out study revealed some surprising behaviors of the fields. Some fields seem to be more harmful than helpful while on the other hand no field seems to be absolutely crucial to getting results even though leaving some fields out shows prediction accuracy going down significantly.

Balancing the classes does not seem to have a major effect on the outcome but on the other hand as the dataset is very imbalanced undersampling all the way to balanced classes the result seems to be the same or slightly better than with a bigger dataset. This opens the possibility of faster training and simpler data handling.

4.9 Learnings

Some important enough learnings that they should be written down were gathered during this research. These learnings are probably more or less obvious to the more experienced reader but for someone with less experience they might prove valuable.

First, it is important to really consider how to use sparse data structures when working with sparse data. Incorrect fill value, for example, does not give any obvious indication but any sparsity in the data is practically rendered non-existent. It is also important to consider when to convert a dataframe to sparse format as some operations are slowed down by orders of magnitude when run on a sparse dataframe but on the other hand doing the conversion too late will probably cause one to run into swapping issues, that slow any operations down, or even running completely out of memory crashing the process. Using sparse, or at least domain appropriate, data formats for storing data to disk is also clearly beneficial not just for data reading speed but also for disk and memory requirements.

Secondly the AWS tools (and most likely other modern cloud services like Google Cloud and Azure) are very useful when the need for big data handling is only occasion-

al. For example being able to index and query the hundreds of gigabytes of raw data available for this research was invaluable and would have taken a lot more time if it would have had to have been dealt with using self-managed tools. The same, with caveats, applies to hosted notebook systems like AWS Sagemaker.

5 CONCLUSION

5.1 Summary

In this thesis we studied the practical results of three different predictive models for predicting the CTR of a real-life digital advertisement system. The results clearly show an advantage of using a generalizing predictive model over a naïve model that only considers historical data. The generalization capability of a model that uses a wider range of information for making the predictions exceeds any capability of the naïve model not only with campaigns with no history but also with campaigns that have extensive recorded history. The difference is larger than expected.

Factorization machine is a very capable tool for CTR prediction. It outperformed the other models tested in this research very consistently with the only downside being extended training time when compared to for example linear model.

A predictive model is fairly good at generalizing over at least a day's worth of new data. Our testing showed almost no degradation in results over the course of the day following the data used in training.

One day's data is not enough to train a factorization machine. Our testing clearly showed that the prediction quality is inferior to a factorization machine trained with a week's worth of data.

5.2 Recommendations

Based on this work the recommendation to commissioning party is to take a factorization machine with latent vector size 32 into use and train it with at least a week's worth of data as the results suggest that less is not as effective. Further recommendation is to train a new model once a day to make sure it does not go stale.

6 FUTURE WORK

6.1 Further Feature Study

The feature drop-off study done for this thesis shows there being significant differences in the importance of the different features. This indicates that a further, more detailed study on the features would be beneficial. Reducing the number of features reduces the computations needed to train the model and to get predictions.

In addition, some features seemed to be outright harmful. The underlying causes should be studied as well as testing training without these features should be tested to conclude whether this is just a fluke or whether there is an actual issue with the features in question.

6.2 Undersampling Study

The results of heavy undersampling were surprisingly promising but inconclusive. To better understand what the effects of different levels of undersampling are, more research is needed. On the other hand, there are multiple more advanced methods of undersampling (Galar, Fernandez, Barrenechea, Bustince, & Herrera, 2012; Japkowicz & Stephen, 2002; Oommen et al., 2011; Seiffert, Khoshgoftaar, Van Hulse, & Napolitano, 2010) and these methods might provide an even better result.

6.3 Online Testing

The actual production environment behavior changes with the ad selection and it is quite possible that the actual performance of the models is different in production an online testing will also be performed. For online tests the models would be deployed to production for a limited time and their performance evaluated against the current system as a baseline.

6.3.1 Production Deployment

To deploy each of the models to production they are trained and deployed to a stand-alone prediction instance manually to simplify the testing process. The production ad delivery pipeline is modified to be able to use the prediction service offered by the stand-alone instance at will. The pipeline is then, for a predetermined period of time, instructed to use the prediction service.

6.3.2 Performance Evaluation

To determine the quality of predictions and overall performance of each of the models the following things will be measured:

1. Log loss of made predictions. This is mostly a validation measure to make sure the model performs similarly online and offline. It is expected that the value will differ from the offline log loss as the online predictions change the behavior of the system as a whole, but the difference should be relatively small.
2. Overall CTR. This value will indicate whether the model is doing better than the baseline and by how much. As there are other factors that affect overall CTR a fairly large margin of error must be considered.
3. Per media CTR. These values will give more detailed information about the generalization ability of the models. The assumption is that the CTR should change to the same direction for each media.
4. Per campaign CTR. These values will also provide more information about the generalization capabilities of each of the models.

REFERENCES

Chakrabarti, D., Agarwal, D., & Josifovski, V. (2008). Contextual advertising by combining relevance with click feedback. In *Proceedings of the 17th international conference on World Wide Web* (pp. 417–426). ACM.

Chapelle, O., Manavoglu, E., & Rosales, R. (2015). Simple and scalable response prediction for display advertising. *ACM Transactions on Intelligent Systems and Technology (TIST)*, *5*(4), 61. https://doi.org/10.1145/0000000.0000000

Cheng, Haibin, & Cantú-paz, E. (2010). Personalized Click Prediction in Sponsored Search, 351–359.

Cheng, Haibin, Zwol, R. Van, Azimi, J., Manavoglu, E., Zhang, R., Zhou, Y., & Navalpakkam, V. (2012). Multimedia Features for Click Prediction of New Ads in Display Advertising.

Cheng, Heng-tze, Koc, L., Harmsen, J., Shaked, T., Chandra, T., Aradhye, H., … Shah, H. (2016). Wide & deep learning for recommender systems. In *Proceedings of the 1st Workshop on Deep Learning for Recommender Systems* (pp. 7–10). ACM.

Ciaramita, M., Murdock, V., & Plachouras, V. (2008). Online Learning from Click Data for Sponsored Search, 227–236.

Cortes, C., & Vapnik, V. (1995). Support-vector networks. *Machine Learning*, *20*(3), 273–297.

Deng, W., Ling, X., Qi, Y., Tan, T., Manavoglu, E., & Zhang, Q. (2018). Ad Click Prediction in Sequence with Long Short-Term Memory Networks: an Externality-aware Model. In *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval* (pp. 1065–1068). ACM. https://doi.org/10.1145/3209978.3210071

Dozat, T. (2016). Incorporating nesterov momentum into adam.

Duchi, J., Hazan, E., & Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, *12*(Jul), 2121–2159.

Fawcett, T. (2006). An introduction to ROC analysis. *Pattern Recognition Letters*, *27*(8), 861–874.

Galar, M., Fernandez, A., Barrenechea, E., Bustince, H., & Herrera, F. (2012). A review on ensembles for the class imbalance problem: bagging-, boosting-, and hybrid-

based approaches. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, *42*(4), 463–484.

Google. (2019). About the Display Network ad auction. Retrieved April 27, 2019, from https://support.google.com/google-ads/answer/2996564

Guo, H., Tang, R., Ye, Y., Li, Z., He, X., & Dong, Z. (2017). Deepfm: a factorization-machine based neural network for ctr prediction. *ArXiv Preprint ArXiv:1703.04247*, *14*(8), 1–14.

Hillard, D., Schroedl, S., Manavoglu, E., Raghavan, H., & Leggetter, C. (2010). Improving Ad Relevance in Sponsored Search, 361–369.

Internet Advertising Bureau. (2018). *IAB internet advertising revenue report 2018 - first six months results*.

James, G., Witten, D., Hastie, T., & Tibshirani, R. (2013). *An introduction to statistical learning* (Vol. 112). Springer.

Japkowicz, N., & Stephen, S. (2002). The class imbalance problem: A systematic study. *Intelligent Data Analysis*, *6*(5), 429–449.

Juan, Y., Lin, C.-J., Zhuang, Y., Chin, W.-S., & Lin, C.-J. (2016). Field-aware factorization machines for CTR prediction. In *Proceedings of the 10th ACM Conference on Recommender Systems* (pp. 43–50). ACM.

Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. *ArXiv Preprint ArXiv:1412.6980*.

Lian, J., Chen, Z., Zhou, X., Zhang, F., Chen, Z., Xie, X., & Sun, G. (2018). xDeepFM: Combining Explicit and Implicit Feature Interactions for Recommender Systems. *ArXiv Preprint ArXiv:1803.05170*.

Liu, W., Tang, R., Li, J., Yu, J., Guo, H., He, X., & Zhang, S. (2018). Field-aware probabilistic embedding neural network for CTR prediction. In *Proceedings of the 12th ACM Conference on Recommender Systems* (pp. 412–416). ACM. https://doi.org/10.1145/3240323.3240396

Mcmahan, H. B., Holt, G., Sculley, D., Young, M., Ebner, D., Grady, J., … Kubica, J. (2013). Ad click prediction: a view from the trenches. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining* (pp. 1222–1230). ACM.

Oommen, T., Baise, L. G., & Vogel, R. M. (2011). Sampling Bias and Class Imbalance in Maximum-likelihood Logistic Regression, 99–120.

https://doi.org/10.1007/s11004-010-9311-8

Pan, J., Xu, J., Ruiz, A. L., Zhao, W., Pan, S., Sun, Y., & Lu, Q. (2018). Field-weighted Factorization Machines for Click-Through Rate Prediction in Display Advertising. In *Proceedings of the 2018 World Wide Web Conference on World Wide Web* (pp. 1349–1357). International World Wide Web Conferences Steering Committee.

Rendle, S. (2010). Factorization machines. In *Data Mining (ICDM), 2010 IEEE 10th International Conference on* (pp. 995–1000). IEEE.

Richardson, M., Ragno, R., & Dominowska, E. (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). ACM.

Seiffert, C., Khoshgoftaar, T. M., Van Hulse, J., & Napolitano, A. (2010). RUSBoost: A hybrid approach to alleviating class imbalance. *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, *40*(1), 185–197.

Singel, R. (2010). Oct. 27, 1994: Web Gives Birth to Banner Ads. Retrieved May 15, 2019, from https://www.wired.com/2010/10/1027hotwired-banner-ads/

Turcios Rodriguez, E. (2018). Tracking Cookies in the European Union, an Empirical Analysis of the Current Situation.

Wang, Q., Liu, F. F., Xing, S., & Zhao, X. (2018). A New Approach for Advertising CTR Prediction Based on Deep Neural Network via Attention Mechanism. *Computational and Mathematical Methods in Medicine*, *2018*. https://doi.org/10.1155/2018/8056541

Zhang, W., Du, T., & Wang, J. (2016). Deep learning over multi-field categorical data - a case study on user response prediction. In *European conference on information retrieval* (pp. 45–57). Springer.

Zhou, G., Mou, N., Fan, Y., Pi, Q., Bian, W., Zhou, C., … Gai, K. (2018). Deep Interest Evolution Network for Click-Through Rate Prediction. https://doi.org/10.1145/3219819.3219823

Zhou, G., Zhu, X., Song, C., Fan, Y., Zhu, H., Ma, X., … Gai, K. (2018). Deep interest network for click-through rate prediction. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining* (pp. 1059–1068). ACM.

APPENDIX A – FACTORIZATION MACHINE WITH KERAS

```
import tensorflow as tf
K = tf.keras.backend

def auc(y_true, y_pred):
    auc = tf.metrics.auc(y_true, y_pred)[1]
    tf.keras.backend.get_session().run(tf.local_variables_initializer())
    return auc

class FMLayer(tf.keras.layers.Layer):
    def __init__(self, input_dim, output_dim=10, **kwargs):
        self.input_dim = input_dim
        self.output_dim = output_dim
        super(FMLayer, self).__init__(**kwargs)

    def build(self, input_shape):
        self.kernel = self.add_weight(name='kernel',
                            shape=(self.input_dim, self.output_dim),
                            initializer='glorot_uniform',
                            trainable=True)
        super(FMLayer, self).build(input_shape)

    def call(self, x):
        a = K.pow(K.dot(x,self.kernel), 2)
        b = K.dot(K.pow(x, 2), K.pow(self.kernel, 2))
        return K.mean(a-b, 1, keepdims=True)*0.5

    def compute_output_shape(self, input_shape):
        return (input_shape[0], self.output_dim)

def FM(feature_dim, l2, l1, rank, loss_function='binary_crossentropy',
        optimizer='adam', activation='sigmoid'):
    inputs = tf.keras.Input((feature_dim,))
    liner = tf.keras.layers.Dense(units=1,
      bias_regularizer=tf.keras.regularizers.l2(l2),
      kernel_regularizer=tf.keras.regularizers.l1(l1),
      )(inputs)
    cross = FMLayer(feature_dim, rank)(inputs)
    add = tf.keras.layers.Add()([liner, cross])
    predictions = tf.keras.layers.Activation(activation)(add)
    model = tf.keras.Model(inputs=inputs, outputs=predictions)
    model.compile(loss=loss_function,
                  optimizer=optimizer,
                  metrics=['mse', 'binary_crossentropy', auc])

    return model
```